

Rational Application Developer for WebSphere Software V8 Programming Guide

Develop applications using
Java EE 6 and beyond

Test, debug, and profile with
local and remote servers

Deploy applications to
WebSphere servers



Martin Keen
Rafael Coutinho
Sylvi Lippman
Salvatore Sollami
Sundaragopal Venkatraman
Steve Baber
Henry Cui
Craig Fleming
Venkata Krishna Kumari Gaddam
Brian Hainey
Lara Ziosi



International Technical Support Organization

**Rational Application Developer for WebSphere
Software V8 Programming Guide**

April 2011

Note: Before using this information and the product it supports, read the information in “Notices” on page xxxi.

First Edition (April 2011)

This edition applies to IBM Rational Application Developer for WebSphere Software Version 8.0.1 and to IBM WebSphere Application Server Version 7.0 and WebSphere Application Server V8.0 Beta.

© Copyright International Business Machines Corporation 2011. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	xxxix
Trademarks	xxxix
Preface	xxxiii
The team who wrote this book	xxxiii
Residency team in Raleigh	xxxiii
Residency team working remotely	xxxv
Rational Application Developer development team authors	xxxvii
Additional contributors	xxxviii
Now you can become a published author, too!	xi
Comments welcome	xli
Stay connected to IBM Redbooks	xli
Part 1. Introduction to Rational Application Developer	1
Chapter 1. Introduction	3
1.1 Concepts	4
1.1.1 IBM Rational Software Delivery Platform	4
1.1.2 Eclipse and IBM Rational Software Delivery Platform	8
1.1.3 Challenges in application development	9
1.2 Rational Application Developer supported platforms and databases	10
1.2.1 Supported operating system platforms	10
1.2.2 Supported runtime environments	10
1.3 New features and specifications	12
1.3.1 New features in Rational Application Developer	12
1.3.2 Specification versions	13
1.4 Migration	14
1.5 Sample code	15
1.6 Summary	16
Chapter 2. Programming technologies	17
2.1 Desktop applications	18
2.1.1 Simple desktop applications	18
2.1.2 Database access	22
2.1.3 Graphical user interfaces	23
2.1.4 Extensible Markup Language (XML)	26
2.2 Web applications	27
2.2.1 Hypertext Transfer Protocol (HTTP)	28
2.2.2 Hypertext Markup Language (HTML)	30

2.2.3	Dynamic web applications	32
2.2.4	JavaServer Faces and persistence using JPA	39
2.2.5	Web 2.0 development	41
2.2.6	Portal applications	44
2.3	Enterprise JavaBeans and Java Persistence API	46
2.3.1	EJB 3.1 specification: What is new	47
2.3.2	Types of EJBs	48
2.3.3	Java Persistence API	50
2.3.4	Other EJB and JPA features	51
2.4	Web services	52
2.4.1	Interoperability considerations	53
2.4.2	Web services in Java EE 6	54
2.5	Messaging systems	59
2.5.1	Java Message Service	60
2.5.2	Message-driven beans (MDBs)	60
2.5.3	Requirements for the development environment	61
2.6	OSGi applications	62
2.6.1	OSGi features	62
2.6.2	Benefits of OSGi	63
2.7	Other applications	63
2.7.1	Java EE application clients	64
2.7.2	Enterprise information system applications	67
2.7.3	Service Component Architecture applications	67
2.7.4	Session Initiation Protocol applications	68
2.7.5	Communications Enabled Applications (CEA)	68
	Chapter 3. Workbench setup and preferences	71
3.1	Workbench basics	72
3.1.1	Workbench basics	75
3.2	Preferences	80
3.2.1	Automatic builds	82
3.2.2	Manual builds	83
3.2.3	File associations	83
3.2.4	Content types	85
3.2.5	Local history	85
3.2.6	Perspectives preferences	87
3.2.7	Web browser preferences	88
3.2.8	Internet preferences	89
	Chapter 4. Perspectives, views, and editors	91
4.1	Integrated development environment	92
4.1.1	Perspectives	92
4.1.2	Views	92

4.1.3	Editors	93
4.1.4	Perspective layout	94
4.1.5	Switching perspectives	96
4.1.6	Specifying the default perspective	97
4.1.7	Organizing and customizing perspectives	98
4.2	Help system for Rational Application Developer	100
4.2.1	Context-sensitive help	103
4.3	Available perspectives	104
4.3.1	CVS Repository Exploring perspective	106
4.3.2	Data perspective	107
4.3.3	Database Debug perspective	110
4.3.4	Database Development perspective	111
4.3.5	Debug perspective	112
4.3.6	Java perspective	114
4.3.7	Java Browsing perspective	115
4.3.8	Java EE perspective	117
4.3.9	Java Type Hierarchy perspective	118
4.3.10	JavaScript perspective	120
4.3.11	JPA perspective	121
4.3.12	Modeling perspective	123
4.3.13	Plug-in Development perspective	124
4.3.14	Profiling and Logging perspective	126
4.3.15	Report Design perspective	127
4.3.16	Resource perspective	128
4.3.17	Team Synchronizing perspective	129
4.3.18	Test perspective	131
4.3.19	Web perspective	132
4.3.20	XML perspective	136
4.3.21	Progress view	138
4.4	Summary	139
Chapter 5. Projects		141
5.1	Java Enterprise Edition 6	142
5.1.1	Enterprise application modules	144
5.1.2	Web modules	145
5.1.3	EJB modules	145
5.1.4	Application client modules	146
5.1.5	Resource adapter modules	146
5.1.6	Java utility libraries	146
5.2	Project basics	146
5.2.1	Creating a new project	147
5.2.2	Project properties	155
5.2.3	Deleting projects	156

5.2.4	Transferring projects between workspaces	156
5.2.5	Closing projects	157
5.3	Java EE 6 project types	157
5.3.1	Enterprise application projects	158
5.3.2	Application client project	159
5.3.3	Dynamic web project	159
5.3.4	EJB project	160
5.3.5	Connector project	161
5.3.6	Utility project	161
5.4	Project wizards	162
5.5	Sample projects	167
5.5.1	Help system samples	167
5.5.2	Example projects wizard	170
5.6	Summary	171
 Chapter 6. Unified Modeling Language		173
6.1	Overview	174
6.2	Constructing and visualizing applications with UML	174
6.2.1	UML visualization capabilities	176
6.2.2	Unified Modeling Language	177
6.3	Working with UML class diagrams	180
6.3.1	Creating class diagrams	181
6.3.2	Creating, editing, and viewing Java elements by using UML class diagrams	184
6.3.3	Creating, editing, and viewing EJB components within UML class diagrams	188
6.3.4	Creating, editing, and viewing WSDL elements within UML class diagrams	194
6.3.5	Class diagram preferences	205
6.4	Exploring relationships in applications	206
6.4.1	Browse diagrams	206
6.4.2	Topic diagrams	208
6.5	Describing interactions with UML sequence diagrams	212
6.5.1	Creating sequence diagrams	214
6.5.2	Creating lifelines	215
6.5.3	Creating messages	216
6.5.4	Creating combined fragments	219
6.5.5	Creating references to external diagrams	221
6.5.6	Exploring Java methods with static method sequence diagrams	222
6.5.7	Sequence diagram preferences	224
6.6	More information about UML	226
 Part 2. Java and XML development		227

Chapter 7. Developing Java applications	229
7.1 Java perspectives, views, and editor overview	230
7.2 Java perspective	232
7.2.1 Package Explorer view	232
7.2.2 Hierarchy view.	233
7.2.3 Outline view.	233
7.2.4 Problems view.	234
7.2.5 Declaration view	236
7.2.6 Console view.	237
7.2.7 Call Hierarchy view	238
7.3 Java Browsing perspective	239
7.4 Java Type Hierarchy perspective	240
7.5 Developing the ITSO Bank application	240
7.5.1 ITSO Bank application overview	240
7.5.2 Packaging structure	240
7.5.3 Interfaces and classes overview	241
7.5.4 Interfaces and classes structure	242
7.5.5 Interface and class fields and getter and setter methods	243
7.5.6 Interface methods	244
7.5.7 Class constructors and methods.	246
7.5.8 Class diagram	249
7.6 ITSO Bank application step-by-step development guide	250
7.6.1 Creating a Java project	250
7.6.2 Creating a UML class diagram	254
7.6.3 Creating Java packages	256
7.6.4 Creating Java interfaces	257
7.6.5 Creating Java classes	259
7.6.6 Creating Java attributes (fields) and getter and setter methods	262
7.6.7 Adding method declarations to an interface	268
7.6.8 Adding constructors and Java methods to a class	270
7.6.9 Creating relationships between Java types.	271
7.6.10 Implementing the classes and methods	275
7.6.11 Running the ITSO Bank application	277
7.6.12 Creating a run configuration	277
7.6.13 Understanding the sample code	280
7.6.14 Additional features used for Java applications	284
7.6.15 Using scripting inside the JRE	284
7.6.16 Analyzing the source code	286
7.6.17 Debugging a Java application	290
7.7 Using the Java scrapbook.	290
7.7.1 Pluggable Java Runtime Environment	292
7.7.2 Exporting Java applications to a JAR file	293
7.7.3 Running Java applications that are external to Rational Application	

Developer	296
7.7.4 Importing Java resources from a JAR file into a project	297
7.7.5 Javadoc tooling	298
7.8 Generating the Javadoc	299
7.8.1 Generating the Javadoc from an existing project	299
7.8.2 Generating the Javadoc from an Ant script.	301
7.8.3 Generating the Javadoc with diagrams from existing tags	302
7.8.4 Generating the Javadoc with diagrams automatically.	303
7.9 Java editor and rapid application development	304
7.9.1 Navigating through the code	305
7.9.2 Source folding	307
7.9.3 Type hierarchy	309
7.9.4 Smart insert.	309
7.9.5 Marking occurrences.	309
7.9.6 Smart compilation	310
7.9.7 Java and file search	310
7.9.8 Working sets	314
7.9.9 Quick fix	315
7.9.10 Quick assist.	316
7.9.11 Content assist	317
7.9.12 Import generation	318
7.9.13 Adding constructors	319
7.9.14 Using the delegate method generator.	322
7.9.15 Refactoring	326
7.10 More information	329
Chapter 8. Developing XML applications	331
8.1 XML overview and associated technologies	332
8.1.1 XML processors	332
8.1.2 DTDs and XML schemas	333
8.1.3 XSL	334
8.1.4 XML namespaces	335
8.1.5 XPath	335
8.1.6 XML catalog	336
8.2 Rational Application Developer XML tools	336
8.2.1 Creating an XML schema	338
8.2.2 Generating HTML documentation from an XML schema file	350
8.2.3 Generating an XML file from an XML schema file.	351
8.2.4 Editing an XML file	352
8.2.5 Working with XSL transformation files	354
8.2.6 Transforming an XML file into an HTML file	358
8.2.7 XML mapping	360
8.3 Service Data Objects and XML	370

8.3.1	Generating SDOs from an XML schema	371
8.3.2	Marshal SDO objects to XML	372
8.3.3	Unmarshal XML to an SDO data graph	375
8.4	JAXB and XML	378
8.4.1	Generating JAXB classes from an XML schema	379
8.4.2	Marshal JAXB objects to XML	381
8.4.3	Unmarshal the XML file to JAXB objects	383
8.4.4	JAXB customization	385
8.5	Feature Pack for XML	387
8.6	More information	388

Part 3. Persistence and enterprise information system integration development 391

Chapter 9.	Developing database applications	393
9.1	Introduction	394
9.2	Connecting to the ITSOBANK database	394
9.2.1	Connecting to databases	395
9.2.2	Creating a connection to the ITSOBANK database	395
9.2.3	Browsing a database with the Data Source Explorer	400
9.3	Creating SQL statements	401
9.3.1	Creating a Data Development project	402
9.3.2	Populating the transactions table	403
9.3.3	Creating a select statement	404
9.3.4	Running the SQL query	412
9.4	Developing Java stored procedures	413
9.4.1	Creating a Java stored procedure	413
9.4.2	Deploying a Java stored procedure	417
9.4.3	Running the stored procedure	418
9.5	Developing SQLJ applications	419
9.5.1	Creating SQLJ files	420
9.5.2	Examining the generated SQLJ file	424
9.5.3	Testing the SQLJ program	425
9.6	Data modeling	427
9.6.1	Creating a Data Design project	428
9.6.2	Creating a physical data model	429
9.6.3	Modeling with diagrams	433
9.6.4	Generating DDL from a physical data model and deploying	436
9.6.5	Analyzing the data model	439
9.7	More information	440
Chapter 10.	Persistence using the Java Persistence API	443
10.1	Introducing the Java Persistence API	444
10.1.1	JPA entity object	445
10.1.2	Object-rational mapping	446

10.1.3	Entity inheritance	453
10.1.4	Persistence units	454
10.1.5	Entity Manager	455
10.1.6	JPA Manager Bean	458
10.1.7	Java Persistence Query Language	459
10.1.8	Criteria API	464
10.1.9	Persistence provider	466
10.1.10	JPA 2.0 enhancements	467
10.2	Creating a JPA project	469
10.2.1	Setting up the ITSOBANK database	469
10.2.2	Create a new JPA project	470
10.2.3	Adding JPA support to an existing project	474
10.2.4	Converting a Java project to a JPA project	475
10.3	Creating JPA entities	476
10.3.1	Creating a new JPA entity with the wizard	477
10.3.2	Creating a JPA entity when adding persistence to a POJO	480
10.3.3	Generating database tables from JPA entities	483
10.3.4	Generating JPA entities from database tables	483
10.3.5	Adding business logic	492
10.3.6	Adding named queries	494
10.4	Creating a JPA Manager Bean	495
10.5	Visualizing JPA entities	498
10.6	Testing JPA entities	501
10.6.1	Creating the Java project for entity testing	502
10.6.2	Creating a Java class for entity testing	502
10.6.3	Setting up the build path for OpenJPA	503
10.6.4	Setting up the persistence.xml file	508
10.6.5	Creating the test	510
10.6.6	Running the JPA entity test	514
10.6.7	Displaying the SQL statements	518
10.6.8	Adding inheritance	519
10.7	Preparing the entities for deployment in the server	526
10.8	More information	529
Chapter 11. Developing applications to connect to enterprise information systems		
		531
11.1	Introduction to Java EE Connector Architecture	532
11.1.1	System contracts	532
11.1.2	Resource adapter	534
11.1.3	Common Client Interface	535
11.1.4	WebSphere adapters	535
11.2	Application development for EIS	536
11.2.1	Importers	537

11.2.2	J2C wizards	537
11.3	Sample application overview	538
11.4	CICS outbound scenario	538
11.4.1	Prerequisites	539
11.4.2	Creating the Java data binding class	539
11.4.3	Creating the J2C bean	540
11.4.4	Deploying the J2C bean as an EJB 3.0 session bean	544
11.4.5	Generating a JSF client	546
11.4.6	Running the JSF client	550
11.5	CICS channel outbound scenario	550
11.5.1	Creating the Java data binding for the channel and containers	551
11.5.2	Creating the J2C bean that accesses the channel	555
11.5.3	Developing a web service to invoke the COBOL program	557
11.5.4	Testing the web service with CICS access	562
11.6	SAP outbound scenario	564
11.6.1	Required software and configuration	564
11.6.2	Creating a connector project and J2C beans	565
11.6.3	Generating the sample web application	570
11.6.4	Testing the web application	572
11.7	Monitoring inbound events for resource adapters	572
11.7.1	Monitoring inbound events using WebSphere Business Monitor	572
11.7.2	Monitoring inbound events using WebSphere Business Events	573
11.8	More information	573

Part 4. Enterprise and service-oriented architecture (SOA) application development 575

Chapter 12.	Developing Enterprise JavaBeans (EJB) applications	577
12.1	Introduction to Enterprise JavaBeans	578
12.1.1	EJB 3.1 specification	578
12.1.2	EJB component types	579
12.1.3	EJB services and annotations	590
12.1.4	EJB 3.1 application packaging	600
12.1.5	EJB 3.1 Lite	600
12.1.6	EJB 3.1 features in Rational Application Developer	601
12.2	Developing an EJB module	601
12.2.1	Sample application overview	602
12.2.2	Creating an EJB project	605
12.2.3	Making the JPA entities available to the EJB project	608
12.2.4	Setting up the ITSOBANK database	609
12.2.5	Implementing the session facade	612
12.3	Testing the session EJB and the JPA entities	623
12.3.1	Testing with the Universal Test Client	624
12.3.2	Creating a web application to test the session bean	626

12.3.3	Testing the sample web application	633
12.3.4	Visualizing the test application	634
12.4	Invoking EJBs from web applications	635
12.4.1	Implementing the RAD8EJBWeb application	636
12.4.2	Running the web application	639
12.4.3	Cleaning up	645
12.4.4	Adding a remote interface	646
12.5	More information	647
Chapter 13. Developing Java Platform, Enterprise Edition (Java EE)		
application clients. 649		
13.1	Introduction to Java EE application clients	650
13.2	Overview of the sample application.	652
13.3	Preparing the sample application	654
13.3.1	Importing the enterprise application sample	654
13.3.2	Setting up the sample database	656
13.4	Developing the Java EE application client	659
13.4.1	Creating the Java EE application client projects	660
13.4.2	Configuring the Java EE application client projects	663
13.4.3	Importing the graphical user interface and control classes	666
13.4.4	Creating the BankDesktopController class	668
13.4.5	Completing the BankDesktopController class	669
13.4.6	Creating an EJB reference and binding	671
13.4.7	Registering the BankDesktopController class as the main class	673
13.5	Testing the Java EE application client.	675
13.6	Packaging the Java EE application client	679
13.6.1	Packaging the application	679
13.6.2	Running the deployed application client	680
Chapter 14. Developing web services applications 681		
14.1	Introduction to web services	683
14.1.1	SOA.	683
14.1.2	Web services as an SOA implementation.	684
14.2	New function in Java EE 6 for web services	686
14.2.1	JSR 224: Java API for XML-Based Web Services (JAX-WS) 2.2.	686
14.2.2	JSR 222: Java Architecture for XML Binding (JAXB) 2.2	688
14.2.3	JSR 109: Implementing Enterprise Web Services	688
14.2.4	Related web services standards	689
14.3	JAX-WS programming model	690
14.3.1	Better platform independence for Java applications	690
14.3.2	Annotations	691
14.3.3	Invoking web services asynchronously	691
14.3.4	Dynamic and static clients.	693

14.3.5	Message Transmission Optimization Mechanism support	693
14.3.6	Multiple payload structures	693
14.3.7	SOAP 1.2 support	694
14.4	Web services development approaches	694
14.5	Web services tools in Rational Application Developer	694
14.5.1	Creating a web service from existing resources	695
14.5.2	Creating a skeleton web service	695
14.5.3	Client development	695
14.5.4	Testing tools for web services	696
14.6	Preparing for the JAX-WS samples	696
14.6.1	Importing the sample	697
14.6.2	Testing the application	697
14.7	Creating bottom-up web services from a JavaBean	698
14.7.1	Creating a web service using annotations	698
14.7.2	Creating web services using the Web Service wizard	708
14.7.3	Resources generated by the Web Service wizard	717
14.8	Creating a synchronous web service JSP client	718
14.8.1	Generating and testing the web service client	718
14.9	Creating a web service JavaServer Faces client	726
14.10	Creating a web service thin client	734
14.11	Creating asynchronous web service clients	737
14.11.1	Polling client	738
14.11.2	Callback client	741
14.11.3	Asynchronous message exchange client	743
14.12	Creating web services from an EJB	746
14.13	Creating a top-down web service from a WSDL	749
14.13.1	Designing the WSDL by using the WSDL editor	749
14.13.2	Generating the skeleton JavaBean web service	756
14.13.3	Testing the generated web service	757
14.14	Creating web services with Ant tasks	758
14.14.1	Creation procedure	758
14.14.2	Running the web service Ant task	759
14.15	Sending binary data using MTOM	760
14.15.1	Creating a web service project and importing the WSDL	761
14.15.2	Generating the web service and client	762
14.15.3	Implementing the JavaBean skeleton	765
14.15.4	Testing and monitoring the MTOM-enabled web service	768
14.15.5	How MTOM was enabled on the client	773
14.16	JAX-RS programming model	774
14.16.1	Implementation of JAX-RS in WebSphere Application Server	776
14.16.2	JAX-RS project setup	777
14.16.3	Exposing a JPA application as a RESTful service	781
14.17	Web services security	796

14.17.1	Authentication	796
14.17.2	Message integrity	796
14.17.3	Message confidentiality	797
14.17.4	Policy set	797
14.17.5	Applying WS-Security to a web service and client	798
14.17.6	WS-I Reliable Secure Profile	807
14.18	WS-Policy	808
14.18.1	Configuring a service provider to share its policy configuration	809
14.18.2	Configuring the client policy using a service provider policy	811
14.19	WS-MetadataExchange (WS-MEX)	814
14.20	Security Assertion Markup Language (SAML) support	817
14.20.1	SAML assertions defined in the SAML Token Profile standard	818
14.20.2	SAML APIs	819
14.20.3	SAML Bearer sample: Prerequisites	819
14.20.4	SAML Bearer sample: Bindings	821
14.20.5	SAML Bearer sample: Programmatic token generation	829
14.20.6	SAML Bearer sample: Testing	831
14.21	More information	834

Chapter 15. Developing Open Services Gateway initiative (OSGi) applications

		837
15.1	OSGi overview	838
15.1.1	OSGi architecture	839
15.2	Introduction to OSGi bundles	841
15.2.1	OSGi classloading	841
15.2.2	Bundle manifest file	842
15.2.3	Life cycle of a bundle	844
15.2.4	Blueprint Container Specification	846
15.2.5	Types of bundle archives	849
15.2.6	Relationships among bundles, application archives, and composite archives	850
15.3	Installation of the Feature Pack for OSGi	850
15.4	Tools for OSGi application development	852
15.5	Creating OSGi bundle projects	854
15.5.1	Creating OSGi bundle projects	854
15.5.2	Creating an OSGi application project	856
15.5.3	Creating a composite bundle project	859
15.5.4	Working with the Composite Bundle Manifest	860
15.6	Developing OSGi applications	861
15.6.1	API bundle	862
15.6.2	Persistence bundle	864
15.6.3	Business logic bundle	870
15.6.4	Web interface bundle	874

15.6.5 Application OSGi	877
15.6.6 Deploying the OSGi application	879
15.7 Further information	883

Chapter 16. Developing Service Component Architecture (SCA) applications 885

16.1 Introduction to SCA	886
16.1.1 Concepts	886
16.1.2 Runtime support	895
16.2 SCA project creation or augmentation	896
16.3 Developing a Java component from a WSDL interface	898
16.3.1 Creating a composite	900
16.3.2 Creating a component	901
16.3.3 Implementing the Java component	903
16.4 Creating a contribution to include the deployable composites	904
16.5 Deploying the contribution to WebSphere Application Server	906
16.6 Testing the services provided by the SCA application	907
16.7 Wiring a component to a service on another component	911
16.7.1 Creating a reference to an external Atom feed provider	912
16.7.2 Exposing a service with an Atom binding	920
16.7.3 Adding a contribution and testing the initial implementation	924
16.7.4 Adding a second component to the composite	926
16.7.5 Wiring the reference on one component to the service on the other component	929
16.7.6 Using a property defined in a component and a composite	932
16.7.7 Testing the implementation by exporting the contribution	935
16.8 Reusing an existing Java EE application to create a component	937
16.8.1 Explore the existing EAR	938
16.8.2 Creating a new SCA Enhanced EAR file to hold the web project	939
16.8.3 Creating a new SCA project with a contribution	946
16.8.4 Testing the completed application	951
16.9 Adding intents and policies	953
16.10 More information	953

Chapter 17. Developing Modern Batch jobs on computing grids 957

17.1 Introduction to Modern Batch	958
17.2 New Modern Batch job tools in Rational Application Developer	958
17.3 Working with the Compute-Intensive sample	959
17.3.1 Installing the sample	959
17.3.2 Understanding the sample	961
17.3.3 Deploying the sample	965
17.3.4 Running the sample	966
17.4 Overview of the Transactional batch capabilities	971

17.4.1	Sequence diagram for the Transactional batch pattern	972
17.4.2	Available patterns	975
17.5	Additional information	978
Part 5.	Web application development	979
Chapter 18. Developing web applications using JavaServer Pages (JSP)		
and servlets 981		
18.1	Introduction to Java EE web applications	983
18.1.1	Java EE applications	984
18.1.2	Model view controller pattern	989
18.2	Web development tooling	990
18.2.1	Web perspective and views	991
18.2.2	Page Designer	993
18.2.3	Page templates	995
18.2.4	CSS Designer	996
18.2.5	Security Editor	996
18.2.6	File creation wizards	998
18.3	Rational Application Developer new features	999
18.4	RedBank application design	1001
18.4.1	Model	1001
18.4.2	View layer	1002
18.4.3	Controller layer	1002
18.5	Implementing the RedBank application	1005
18.5.1	Creating the web project	1005
18.5.2	Importing the Java RedBank model	1012
18.5.3	Defining the empty web pages	1012
18.5.4	Creating frameset pages	1014
18.5.5	Customizing frameset web page areas	1017
18.5.6	Customizing a style sheet	1019
18.5.7	Verifying the site navigation and page templates	1020
18.5.8	Developing the static web resources	1022
18.5.9	Developing the dynamic web resources	1026
18.5.10	Working with JSP	1036
18.6	Web application testing	1050
18.6.1	Prerequisites to run the sample web application	1050
18.6.2	Running the sample web application	1050
18.6.3	Verifying the RedBank web application	1051
18.7	More information	1055
Chapter 19. Developing web applications using JavaServer Faces . . . 1057		
19.1	Introduction to JSF	1058
19.1.1	JSF 1.x features and benefits	1058
19.1.2	JSF 2.0 features and benefits	1059

19.1.3	JSF 2.0 application architecture	1060
19.1.4	JSF features in Rational Application Developer	1063
19.2	Developing a web application using JSF and JPA	1064
19.2.1	Setting up the ITSOBANK database	1064
19.2.2	Creating the JSF Project	1065
19.2.3	Creating Facelet templates	1069
19.2.4	Creating Facelets	1075
19.2.5	Creating JPA Manager Beans	1076
19.2.6	Creating JPA page data	1082
19.2.7	Editing the login page	1082
19.2.8	Editing the customer details page	1088
19.2.9	Using Ajax	1092
19.2.10	Running the JSF application	1094
19.2.11	Final code	1096
19.3	More information	1096
Chapter 20. Developing web applications using Web 2.0		1097
20.1	Introduction to Web 2.0 architecture and development practices	1098
20.1.1	Web 2.0 architecture	1098
20.1.2	Technologies used in Web 2.0 applications	1100
20.2	Overview of Web 2.0 tooling features	1104
20.2.1	JavaScript editing	1104
20.2.2	Dojo development	1104
20.2.3	Testing and debugging	1105
20.2.4	JAX-RS services development	1105
20.2.5	Using other server-side technologies	1106
20.3	Developing the Web 2.0 sample application	1106
20.3.1	Setting up the project	1106
20.3.2	Creating the web page	1111
20.3.3	Building a custom Dojo widget	1116
20.3.4	Adding to a page and testing a custom Dojo widget	1121
20.3.5	Adding a Dojo DataGrid to your web page	1125
Chapter 21. Developing portal applications		1131
21.1	Introduction to portal technology	1132
21.1.1	Portal concepts and definitions	1132
21.1.2	IBM WebSphere Portal	1135
21.1.3	Portal and portlet development features in Rational Application Developer	1136
21.1.4	Setting up Rational Application Developer with the Portal test environment	1138
21.2	Developing applications for WebSphere Portal	1138
21.2.1	Portal samples and tutorials	1139

21.2.2	Development strategy	1140
21.2.3	Portal tools for developing portals.	1143
21.3	New WebSphere portal and portlet development tools in Rational Application Developer.	1150
21.3.1	Support for WebSphere Portal Server V7	1150
21.3.2	Site Designing Portlet	1150
21.3.3	New portlet project features	1150
21.3.4	RPC tooling for portlet projects	1151
21.4	Developing portal solutions using portal tools	1151
21.4.1	Developing event handling portlets.	1151
21.4.2	Creating Ajax and Web 2.0 portlets	1158
21.4.3	Deploying and running the application	1162
21.4.4	Creating a portal site with the Site Designing Portlet feature . . .	1163
21.4.5	Developing Dojo-based inter-portlet communication	1167
21.4.6	Consuming RPC adapter services	1172
21.4.7	Creating iWidget projects	1173
21.4.8	JPA tooling support for portlet projects	1175
21.5	More information	1181
 Chapter 22. Developing Lotus iWidgets		1183
22.1	Introduction to iWidgets.	1184
22.1.1	Content	1184
22.1.2	Events and event descriptions	1185
22.1.3	Itemsets and items	1185
22.1.4	Resources	1186
22.2	Developing iWidgets in Rational Application Developer	1186
22.2.1	Accessing the tutorials and samples.	1186
22.2.2	Configuring Rational Application Developer for iWidget development tools	1187
22.3	Working with the sample iWidget application	1187
22.3.1	Preparing the sample iWidget application.	1187
22.3.2	Developing the sample iWidget application	1189
22.3.3	Testing the sample iWidget application.	1193
22.3.4	Deploying into WebSphere Portal V7	1196
22.4	Additional resources	1198
22.4.1	Further information	1200
 Part 6. Deploying, testing, profiling, and debugging applications		1201
 Chapter 23. Cloud environment and server configuration		1203
23.1	Introduction to server configurations	1204
23.1.1	Application servers that are supported by Rational Application Developer	1204
23.1.2	Local and remote test environments.	1207

23.2	Cloud extensions: Developing and testing applications on the IBM Smart Business, Development, and Test Cloud	1207
23.2.1	Installing IBM Rational Desktop Connection Toolkit for Cloud Environments	1208
23.2.2	Working with the IBM Development and Test Cloud	1211
23.2.3	Working with the Cloud Client for Eclipse	1229
23.2.4	Requesting instances from the web client.	1233
23.2.5	Resources for additional information.	1236
23.3	Understanding WebSphere Application Server v8.0 profiles	1237
23.3.1	Types of profiles	1238
23.3.2	Using the profiles	1238
23.4	WebSphere Application Server v8.0 Beta installation.	1239
23.5	Using WebSphere Application Server profiles	1240
23.5.1	Creating a new profile using the WebSphere Profile wizard	1240
23.5.2	Deleting a WebSphere profile	1246
23.5.3	Defining the new server in Rational Application Developer	1247
23.5.4	Customizing a server	1250
23.5.5	Sharing a WebSphere profile between developers.	1253
23.5.6	Defining a server for each workspace.	1255
23.6	Migrating the server resources from Rational Application Developer V7.0 or V7.5 to V8.0	1255
23.7	Adding and removing applications to and from a server	1256
23.7.1	Adding an application to the server.	1256
23.7.2	Removing an application from a server.	1257
23.8	Configuring application and server resources	1258
23.8.1	Creating a data source in the Enhanced EAR editor	1262
23.8.2	Setting the substitution variable	1267
23.8.3	Configuring server resources	1268
23.9	Configuring security.	1268
23.9.1	Configuring security in the server	1268
23.9.2	Configuring security in the workbench	1270
23.10	AJAX Test Server	1271
23.10.1	Configuring the AJAX Test Server	1271
23.10.2	Configuring the AJAX Proxy	1273
23.11	Developing automation scripts	1275
23.12	Tips: Enhancing server interaction performance.	1275
23.12.1	Speeding up server start time	1276
23.12.2	Speeding up application publishing time	1276
23.13	More information	1277
	Chapter 24. Building applications with Apache Ant	1279
24.1	Introduction to Ant	1280
24.1.1	Ant build files.	1280

24.1.2	Ant tasks	1281
24.2	Ant features in Rational Application Developer	1282
24.2.1	Preparing for the sample	1282
24.2.2	Creating a build file	1283
24.2.3	Project definition	1284
24.2.4	Global properties	1285
24.2.5	Building targets	1285
24.2.6	Content assist	1287
24.2.7	Code snippets	1288
24.2.8	Formatting an Ant script	1290
24.2.9	Defining the format of an Ant script	1290
24.2.10	Problems view	1292
24.3	New Ant features in Rational Application Developer	1293
24.3.1	SCA Ant task	1293
24.3.2	OSGi Ant tasks	1294
24.3.3	Other new Ant tasks	1295
24.4	Building a Java EE application	1296
24.4.1	Java EE application deployment packaging	1297
24.4.2	Preparing for the sample	1297
24.4.3	Creating the build script	1298
24.4.4	Running the Ant Java EE application build	1300
24.5	Running Ant outside of Rational Application Developer	1301
24.5.1	Preparing for the headless build	1301
24.5.2	Running the headless Ant build script	1302
24.6	Using the Rational Application Developer Build Utility	1303
24.6.1	Overview of the build utility	1303
24.6.2	Example of using the build utility	1304
24.7	More information about Ant	1307
Chapter 25. Deploying enterprise applications		1309
25.1	Introduction to application deployment	1310
25.1.1	Common deployment considerations	1310
25.1.2	Java EE application components and deployment modules	1311
25.1.3	Deployment descriptors	1311
25.1.4	WebSphere deployment architecture	1315
25.1.5	Java and WebSphere class loader	1320
25.2	Preparing for the EJB application deployment	1325
25.2.1	Reviewing the deployment scenarios	1325
25.2.2	Installing the prerequisite software	1325
25.2.3	Importing the sample application archive files	1327
25.2.4	Sample database	1327
25.3	Packaging the application for deployment	1328
25.3.1	Removing the enhanced EAR data source	1328

25.3.2	Generating the deployment code	1329
25.3.3	Exporting the EAR files	1329
25.4	Manual deployment of enterprise applications	1330
25.4.1	Configuring the data source in the application server	1331
25.4.2	Installing the enterprise applications	1338
25.4.3	Starting the enterprise applications	1342
25.4.4	Verifying the application after manual installation	1343
25.4.5	Uninstalling the application	1345
25.5	Automated deployment using Jython-based wsadmin scripting	1345
25.5.1	Overview of wsadmin	1346
25.5.2	Overview of Jython	1347
25.5.3	Developing a Jython script to deploy the ITSO Bank	1348
25.5.4	Executing the Jython script	1357
25.5.5	Verifying the application after automatic installation	1360
25.5.6	Generating WebSphere admin commands for Jython scripts	1360
25.5.7	Debugging Jython scripts	1363
25.6	More information	1364
Chapter 26. Testing using JUnit		1365
26.1	Introduction to application testing	1366
26.1.1	Test concepts	1366
26.1.2	Test phases	1366
26.1.3	Test environments	1368
26.1.4	Calibration	1369
26.1.5	Test case execution and recording results	1369
26.1.6	Benefits of unit and component testing	1369
26.1.7	Benefits of testing frameworks	1370
26.2	JUnit testing without TPTP	1371
26.2.1	JUnit fundamentals	1371
26.2.2	Test and Performance Tools Platform (TPTP)	1371
26.2.3	New in JUnit 4	1372
26.3	Preparing the JUnit sample	1377
26.3.1	Creating a JUnit test case	1378
26.3.2	Creating a JUnit test suite	1384
26.3.3	Running the JUnit test case or JUnit test suite	1385
26.3.4	Launching individual test methods	1387
26.3.5	Using the JUnit view	1389
26.4	JUnit testing of JPA entities	1390
26.4.1	Preparing the JPA unit testing sample	1390
26.4.2	Setting up the ITSOBANK database	1390
26.4.3	Configuring the RAD8JUnit project	1390
26.4.4	Creating a JUnit test case for a JPA entity	1391
26.4.5	Setting up the persistence.xml file	1393

26.4.6	Running the JPA unit test	1394
26.5	JUnit testing using TPTP	1396
26.5.1	Running the TPTP JUnit test.	1401
26.5.2	Analyzing the test results	1402
26.6	Web application testing	1405
26.6.1	Preparing for the sample.	1406
26.6.2	Recording a test	1407
26.6.3	Editing the test	1409
26.6.4	Generating an executable test	1411
26.6.5	Running the test	1411
26.6.6	Analyzing the test results	1412
26.6.7	Generating test reports	1416
26.7	Cleaning the workspace	1418
Chapter 27. Profiling applications		1419
27.1	Introduction to profiling	1420
27.1.1	Profiling features	1421
27.1.2	Profiling architecture	1424
27.1.3	Profiling and Logging perspective	1426
27.2	Preparing for the profiling sample	1427
27.2.1	Installing the prerequisite software	1427
27.2.2	Enabling the Profiling and Logging capability	1428
27.3	Profiling a Java application	1429
27.3.1	Importing the sample project archive file.	1429
27.3.2	Creating a profiling configuration	1430
27.3.3	Running the EntityTester application	1434
27.3.4	Analyzing profiling data	1435
27.3.5	Execution statistics	1435
27.3.6	Execution flow	1440
27.3.7	UML sequence diagrams	1441
27.3.8	Memory analysis	1444
27.3.9	Thread analysis.	1445
27.3.10	Reports	1447
27.3.11	Cleanup.	1447
27.4	Profiling a web application running on the server	1447
27.4.1	Importing the sample project archive file.	1447
27.4.2	Setting up environment variables to profile a server.	1447
27.4.3	Publishing and running the sample application.	1448
27.4.4	Starting the server in profiling mode	1448
27.4.5	Running the sample application to collect profiling data	1450
27.4.6	Statistics views	1450
27.4.7	Execution statistics	1451
27.4.8	Execution flow	1454

27.4.9	UML sequence diagrams	1456
27.4.10	Refreshing the views and resetting data	1459
27.4.11	Ending the profiling session	1459
27.4.12	Profile on server: Memory and thread analysis.	1459
27.5	More information	1460
Chapter 28. Debugging local and remote applications		1461
28.1	Introducing Rational Application Developer new features.	1463
28.2	Reviewing Rational Application Developer debugging tools	1463
28.2.1	Supported languages and environments	1463
28.2.2	Java debugging features.	1464
28.2.3	XSLT debugging	1469
28.2.4	Stored procedure debugging for DB2 V9	1471
28.2.5	Service Component Architecture debugger	1472
28.2.6	Java tracepoints	1472
28.2.7	Collaborative debugging using Rational Team Concert client . .	1473
28.3	Debugging a web application on a local server.	1474
28.3.1	Importing the sample application	1474
28.3.2	Running the sample application in debug mode	1475
28.3.3	Setting breakpoints in a Java class.	1476
28.3.4	Using the Debug perspective	1479
28.3.5	Watching variables	1480
28.3.6	Evaluating and watching expressions.	1481
28.3.7	Using the Display view	1482
28.3.8	Working with breakpoints	1483
28.3.9	Setting breakpoints in JSP	1484
28.3.10	Debugging JSP	1485
28.4	Debugging a web application on a remote server.	1487
28.4.1	Removing the WebSphere configuration from the workspace . .	1488
28.4.2	Configuring debug mode to start on a remote WebSphere Application Server V8 Beta	1488
28.4.3	Attaching to the remote WebSphere Application Server in Rational Application Developer.	1490
28.4.4	Debugging a remote application	1492
28.5	Using the Jython debugger	1492
28.5.1	Considerations for the Jython debugger	1493
28.5.2	Debugging a sample Jython script	1493
28.6	Using the JavaScript debugger	1495
28.6.1	Setting the default browser to Firefox	1496
28.6.2	JavaScript debugging	1497
28.7	Using Dojo Debug Extension for Firebug	1502
28.7.1	Launching the Dojo Debugger	1502
28.7.2	Exploring the All widgets view.	1503

28.7.3 Exploring the All connections view	1509
28.7.4 Exploring the All Subscriptions view	1513
28.7.5 Exploring the Info side panel.	1515
28.8 Using the debug extension for the Rational Team Concert client (Team Debug)	1516
28.8.1 Supported environments	1517
28.8.2 Prerequisites	1518
28.8.3 Sharing a Java application debug session by transferring it to another user.	1519
28.9 Obtaining more information	1530

Part 7. Management and team development 1531

Chapter 29. Concurrent Versions System (CVS) integration	1533
29.1 Introduction to CVS	1534
29.1.1 CVS features.	1534
29.1.2 CVS support within Rational Application Developer	1535
29.2 Configuring the CVS client for Rational Application Developer.	1537
29.2.1 CVS Server Installation	1537
29.2.2 Configuring the CVS team capabilities	1537
29.2.3 Accessing the CVS repository.	1538
29.3 Configuring CVS in Rational Application Developer	1540
29.3.1 Label decorations	1540
29.3.2 File content	1541
29.3.3 Ignored resources	1543
29.3.4 CVS-specific settings	1545
29.3.5 CVS keyword substitution	1546
29.4 Development scenario.	1550
29.4.1 Creating and sharing the project (step 1, cvsuser1)	1551
29.4.2 Adding a shared project to the workspace (step 2a, cvsuser2)	1554
29.4.3 Modifying the servlet (step 2b, cvsuser1)	1559
29.4.4 Synchronizing with the repository (step 3a, cvsuser1)	1560
29.4.5 Synchronizing with the repository (step 3b, cvsuser2)	1562
29.4.6 Parallel development (step 4, cvsuser1 and cvsuser2).	1562
29.4.7 Creating a version (step 5, cvsuser1)	1568
29.5 CVS resource history	1569
29.6 Comparisons in CVS.	1571
29.6.1 Comparing a workspace file with the repository	1571
29.6.2 Comparing two revisions in the repository	1572
29.7 Annotations in CVS	1574
29.8 Branches in CVS.	1575
29.8.1 Branching	1576
29.8.2 Merging	1581

29.9	Working with patches	1585
29.10	Disconnecting a project	1585
29.11	Team Synchronizing perspective	1587
29.11.1	Custom configuration of resource synchronization	1588
29.11.2	Schedule synchronization	1591
29.12	More information	1592
Chapter 30. IBM Rational Application Developer integration with Rational Team Concert		1595
30.1	System architecture	1596
30.2	Installing Rational Team Concert Client into the Rational Application Developer workbench	1597
30.2.1	Installing Rational Team Concert Client 3.0 into the same workbench as Rational Application Developer	1597
30.2.2	Installing Rational Team Concert Client 2.0.0.2 into the Rational Application Developer workbench	1598
30.3	Collaborative Code Coverage	1599
30.3.1	Configuring a build definition	1599
30.3.2	Creating an Ant build script to generate coverage statistics	1601
30.3.3	Viewing coverage statistics in Rational Application Developer	1607
30.4	Collaborative Debug	1611
30.4.1	Installing the Collaborative debug extensions for Rational Team Concert Client	1611
30.4.2	Installing Rational Debug Extension for IBM Rational Team Concert Server	1612
30.4.3	Using Collaborative Debug	1614
Chapter 31. IBM Rational ClearCase		1619
31.1	Rational Application Developer team support	1621
31.1.1	Team preferences	1621
31.1.2	Team context menu	1622
31.1.3	Derived files and folders	1622
31.2	Integrating Rational Application Developer with ClearCase	1623
31.2.1	ClearCase terminology	1623
31.3	ClearCase SCM Adapter	1625
31.3.1	Installing ClearCase SCM Adapter	1625
31.3.2	Connecting to ClearCase with the SCM Adapter	1628
31.3.3	ClearCase SCM Adapter preferences	1632
31.3.4	Clearcase SCM Adapter and dynamic views	1636
31.4	ClearCase Remote Client	1637
31.4.1	Connecting to ClearCase with the ClearCase Remote Client	1638
31.4.2	ClearCase Remote Client preferences	1639
31.4.3	ClearCase Remote Client menus	1640

31.4.4	ClearCase Explorer perspective	1642
31.4.5	ClearCase Remote Client decorators	1648
31.5	ClearCase views and Rational Application Developer workspaces . .	1649
31.6	Populating Rational Application Developer workspaces: Using Team Project Set files.	1650
31.7	Working in Base ClearCase with SCM Adapter and dynamic views . .	1651
31.7.1	Prerequisites	1652
31.7.2	Project setup	1653
31.7.3	Making an unreserved checkout to work on the same file	1656
31.7.4	Merging changes.	1660
31.8	Working in ClearCase UCM with ClearCase Remote Client.	1666
31.8.1	Prerequisites	1669
31.8.2	Connecting to the ClearCase Change Management Server and joining a UCM project.	1669
31.8.3	Initiating work in the development view or stream	1674
31.8.4	Delivering activities to the integration stream	1679
31.8.5	Reviewing the results and creating a new baseline	1683
31.8.6	A new user joins the project	1686
31.8.7	Another user modifies the same project	1692
31.9	More information	1696
Chapter 32. Code Coverage		1697
32.1	Overview	1698
32.1.1	Instrumentation	1698
32.1.2	Basic blocks versus executable units	1698
32.2	Generating coverage statistics in Rational Application Developer . . .	1700
32.2.1	Viewing results in the Package Explorer	1701
32.2.2	Viewing results in the Java Editor	1703
32.3	Generating reports	1704
32.3.1	Workbench reports	1705
32.3.2	HTML reports	1706
32.4	Generating statistics outside of the workbench.	1707
32.4.1	Static instrumentation	1707
32.4.2	Dynamic instrumentation.	1709
32.4.3	Report generation	1712
32.5	Coverage report comparison.	1714
32.5.1	Generating a coverage comparison report in Rational Application Developer	1715
32.5.2	Generating coverage comparison report with Ant.	1716
32.6	Importing the coverage data statistics file	1718
32.7	Generating statistics for web applications.	1720
32.7.1	Support for WebSphere Application Server	1722
32.7.2	Generic application server support	1723

32.8 Rational Team Concert integration	1725
Chapter 33. Developing Session Initiation Protocol applications	1727
33.1 Introduction to SIP	1728
33.1.1 SIP 1.1 specification	1728
33.1.2 Converged SIP applications	1731
33.1.3 SIP 1.1 annotations	1731
33.1.4 SIP application packaging	1733
33.2 Developing a SIP application	1734
33.2.1 SIP tooling overview	1735
33.2.2 Sample application overview	1746
33.2.3 Setting up the project	1750
33.2.4 Implementing the classes	1758
33.2.5 SIP deployment descriptor	1769
33.2.6 Preparing for deployment	1770
33.2.7 Deploying SIP from Rational Application Developer	1773
33.3 Testing the SIP 1.1 application	1775
33.3.1 Test environment	1775
33.3.2 Running the application	1776
33.4 SIP-specific annotations in SIP 1.1 applications	1780
33.5 More information	1780
Part 8. Appendixes	1781
Appendix A. Installing the products	1783
Download locations	1784
Installation Launchpad	1784
IBM Installation Manager	1785
Installing Rational Application Developer	1788
Installing the license for Rational Application Developer	1803
Updating Rational Application Developer	1804
Uninstalling Rational Application Developer	1805
Rational Desktop Connection Toolkit for Cloud Environments	1805
Installing WebSphere Portal V7	1806
Installing WebSphere Portal V7	1806
Adding WebSphere Portal V7 to Rational Application Developer	1813
Optimizing the WebSphere Portal Server for development	1818
Verifying development mode	1819
Defining remote servers for testing portals	1820
Defining page creation settings	1822
Enabling the debugging service	1823
Stopping the server	1823
Installing IBM Rational Team Concert	1824
Installing Rational Team Concert Standard Edition server	1824

Installing Rational Team Concert Build Engine and Build Toolkit	1834
Installing the client and the debug extensions	1835
Installing Rational Application Developer Build Utility	1840
Installing IBM Rational ClearCase	1842
Creating a Storage Location	1845
Creating a VOB for use in Base ClearCase	1847
Creating a dynamic view	1849
Installing IBM Rational ClearCase Remote Client Extension	1855
Verifying the installation	1858
Configuring ClearCase for UCM development	1860
Appendix B. Performance tips for Rational Application Developer . . .	1871
Better hardware	1872
Shared EARs (binary modules)	1872
Annotations	1873
Publishing	1873
Shorter build time by tuning validation	1874
Only install what you need	1875
No circular dependencies	1875
Using a remote test server	1875
Tuning your anti-virus program	1876
Defragmenting disks	1876
Appendix C. Additional material	1877
Locating the web material	1878
Accessing the web material	1878
System requirements for downloading the web material	1878
Using the sample code	1878
Unpacking the sample code	1878
Description of the sample code	1879
Importing sample code from a project archive file	1880
Setting up the ITSOBANK database	1880
Derby	1881
DB2	1881
Configuring the data source in WebSphere Application Server	1882
Starting the WebSphere Application Server	1882
Configuring the environment variables	1882
Configuring J2C authentication data	1883
Configuring the JDBC provider	1884
Creating the data source	1884
Abbreviations and acronyms	1887
Related publications	1891

IBM Redbooks publications	1891
Other publications	1892
Online resources	1892
How to get IBM Redbooks publications	1897
Help from IBM	1897

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®	i5/OS®	Rational Team Concert™
Build Forge®	IBM®	Rational®
CICS®	IMS™	Redbooks®
ClearCase®	Informix®	Redbooks (logo)  ®
ClearQuest®	iSeries®	RequisitePro®
DB2 Universal Database™	Jazz™	Sametime®
DB2®	Lotus®	System z®
developerWorks®	Maximo®	WebSphere®
FileNet®	Passport Advantage®	z/OS®
Global Business Services®	Rational Rose®	zSeries®

The following terms are trademarks of other companies:

Adobe, the Adobe logo, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Snapshot, and the NetApp logo are trademarks or registered trademarks of NetApp, Inc. in the U.S. and other countries.

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

IBM® Rational® Application Developer for WebSphere® Software V8 is the full-function Eclipse 3.6 technology-based development platform for developing Java™ Platform, Standard Edition Version 6 (Java SE 6) and Java Platform, Enterprise Edition Version 6 (Java EE 6) applications. Beyond this function, Rational Application Developer provides development tools for technologies, such as OSGi, Service Component Architecture (SCA), Web 2.0, and XML. It has a focus on applications to be deployed to IBM WebSphere Application Server and IBM WebSphere Portal.

Rational Application Developer provides integrated development tools for all development roles, including web developers, Java developers, business analysts, architects, and enterprise programmers.

This IBM Redbooks® publication is a programming guide that highlights the features and tooling included with Rational Application Developer V8.0.1. Many of the chapters provide working examples that demonstrate how to use the tooling to develop applications and achieve the benefits of visual and rapid application development. This publication is an update of *Rational Application Developer V7.5 Programming Guide*, SG24-7672.

The team who wrote this book

This book was produced by a team of specialists from around the world. The residency team was led by:

Martin Keen is a Consulting IT Specialist at the ITSO, Raleigh Center. He writes extensively about WebSphere products and service-oriented architecture (SOA). He also teaches IBM classes worldwide about WebSphere, SOA, and enterprise service bus (ESB). Before joining the ITSO, Martin worked in the EMEA WebSphere Lab Services team in Hursley, U.K. Martin holds a Bachelors degree in Computer Studies from Southampton Institute of Higher Education.

Residency team in Raleigh

The following authors joined the Redbooks residency working at the International Technical Support Organization, Raleigh Center.

Rafael Coutinho is an IBM Advisory Software Engineer working for Software Group in the Brazil Software Development Lab. His professional expertise covers many technology areas ranging from embedded to platform-based solutions. He is currently working on Maximo® Spatial, which is the geographic information system (GIS) add-on of IBM Maximo Enterprise Asset Management (EAM). He is a certified Java enterprise architect and Accredited IT Specialist, specialized in high-performance distributed applications on corporate and financial projects.

Rafael is a computer engineer graduate from the State University of Campinas (Unicamp), Brazil, and has a degree in Information Technologies from the Centrale Lyon (ECL), France.

Sylvi Lippmann is a Software IT Specialist in the GBS Financial Solutions team in Germany. She has over seven years of experience as a Software Engineer, Technical Team Leader, Architect, and Customer Support representative. She is experienced in the draft, design, and realization of object-oriented software systems, in particular, the development of Java EE-based web applications, with a priority in the surrounding field of the WebSphere product family. She holds a degree in Business Informatic Engineering.

Salvatore Sollami is a Software IT Specialist in the Rational brand team in Italy. He has been working at IBM with particular interest in the change and configuration area and web application security. He also has experience in the Agile Development Process and Software Engineering. Before joining IBM, Salvatore worked as a researcher for Process Optimization Algorithmic, Mobile Agent Communication, and IT Economics impact. He developed the return on investment (ROI) SOA investment calculation tool. He holds the “Laurea” (M.S.) degree in Computer Engineering from the University of Palermo. In cooperation with IBM, he received an M.B.A. from the MIP - School of Management - polytechnic of Milan.

Sundaragopal Venkatraman is a Technical Consultant at the IBM India Software Lab. He has over 11 years of experience as an Architect and Lead working on web technologies, client server, distributed applications, and System z®. He works on the WebSphere stack on process integration, messaging, and the SOA space. In addition to handling training on WebSphere, he also gives back to the technical community by lecturing at WebSphere technical conferences and other technical forums.



Figure 1 Raleigh team (left-to-right): Rafael, Sylvi, Salvatore, Sundar, and Martin

Residency team working remotely

The following authors joined the Redbooks residency remotely working from their home locations:

Steve Baber has been working in the Computer Industry since the late 1980s. He has over 15 years of experience within IBM, first as a consultant to IBM and then as an employee. Steve has supported several industries during his time at IBM, including health care, telephony, and banking and currently supports the IBM Global Finance account as a Team Lead for the Global Contract Management project.

Henry Cui works as an independent consultant through his own company, Kaka Software Solution. He provides consulting services to large financial institutions in Canada. Before this work, Henry worked with the IBM Rational services and support team for eight years, where he helped many clients resolve design, development, and migration issues with Java EE development. His areas of expertise include developing Java EE applications with Rational Application Developer tools and administering WebSphere Application Server servers, security, SOA, and web services. Henry is a frequent contributor of developerWorks® articles. He also co-authored five IBM Redbooks publications. Henry holds a degree in Computer Science from York University.

Craig Fleming is a Solution Architect who works for IBM Global Business Services® in Auckland, New Zealand. He has worked for the last 15 years leading and delivering software projects for large enterprises as a solution developer and architect. His area of expertise is in designing and developing middleware solutions, mainly with WebSphere technologies. He has worked in several industries, including Airlines, Insurance, Retail, and Local Government.

Craig holds a Bachelor of Science (Honors) in Computer Science from Otago University in New Zealand.

Venkata Krishna Kumari Gaddam is a Software Test Specialist at ISL, IBM India. She holds an Engineering Degree in Electronics and Communications from the JNTU-affiliated Engineering college. She has a total experience of five years in the IT industry. She started her career with IBM in 2005 and has been working in the WebSphere Application Server Functional Test team. She has experience working on various releases of WebSphere Application Server and was actively involved in testing WebSphere Application Server components. She is a Certified Software Tester (CSTE) and Certified WebSphere Application Server V7.0 Administrator. She also performed many mentoring sessions for CSTE. Her other achievements include submitting an abstract for a Poster session on Product Collaboration for Innovation, which got selected in the Academy of Technology in 2008. She is also one of the co-authors of two forthcoming articles on the IP database. Additionally, she was one of the top three winners across India for writing an essay on IBM values in 2008.

Brian Hainey is a Senior Lecturer at Glasgow Caledonian University in Scotland, United Kingdom (U.K.). He currently teaches as part of the undergraduate and postgraduate programs offered in the School of Engineering and Computing. In addition, he teaches training courses in enterprise software development and Java. He holds a Master of Science degree in Electronic Engineering from Heriot-Watt University, Edinburgh. He has more than 20 years experience in the field of software development and has worked at companies, such as National Westminster Bank, Hewlett-Packard, QA Training, and IBM. He holds industry certifications in Java and enterprise software development. His areas of expertise include Java enterprise systems, web services, XML, Unified Modeling Language (UML) modeling, Rational Unified Process, Rational Rose®, Rational Application Development, Rational Software Architecture, and WebSphere Application Server.

Lara Ziosi is an Advisory Software Engineer in IBM Netherlands. She holds a Doctorate in computational methods of Physics from the University of Bologna, Italy. She has 11 years of experience in Rational client support. Her areas of expertise include Java Enterprise support in Rational Application Developer, object-oriented analysis and design with UML, the extensibility of the modeling features of Rational Software Architect, and the integrations with configuration management tools, such as Rational ClearCase® and Rational Team Concert™. Lara is a frequent contributor of Rational Application Developer and Rational Software Architect Technotes (web docs).

Rational Application Developer development team authors

The following authors contributed chapters and sections to this IBM Redbooks publication:

Nitin Dahyabhai is an Advisory Software Engineer for IBM Rational in Research Triangle Park. He has over a decade of experience in Eclipse-based Web Tools and holds a Bachelor of Science degree in Computer Engineering from North Carolina State University.

Chris Jaun is a Staff Software Engineer in Raleigh, NC, U.S. He has six years of software development experience, including two focused on JavaScript and Dojo tooling. Chris holds a Bachelors degree in Computer Science from the Rochester Institute of Technology.

Gary Karasiuk is a Senior Performance Analyst in the IBM Canada Lab. He has 30 years of experience in developing tools for programmers. He holds a degree in Computer Science from the University of Manitoba. His areas of expertise include J2EE development, object-oriented programming, and performance analysis.

Paul Klicnik is a Software Developer at the IBM Toronto lab in Markham, Ontario. He has four years of experience in the areas of testing and performance tools, with specific focus on Code Coverage. Paul holds a degree in Computer Science from the University of Waterloo.

Ernest Mah is a Senior Architect and Development Manager in IBM Rational. He is currently based out of the IBM Toronto Lab and has been with IBM for 14 years. He has a Bachelor of Science Honors degree in Computing Science from the University of Alberta. His experience is in a wide range of areas from C/C++, Java, XML, Java EE, J2C, and Java problem determination tools.

Julio Cesar Chavez Ortiz is a Developer in the Rational Application Developer team at the IBM Mexico Software Lab. He has five years of experience in Java technologies (Java SE/EE), frameworks, such as Spring, Struts, and Wicket, and in the use of application servers (WebSphere, WebLogic, and Apache Tomcat). He holds a degree in Computer Science Engineering from Bonaterra University in Aguascalientes, México. His areas of expertise include web applications based on Java development and Rational Application Developer Session Initiation Protocol (SIP) tools development.

Ravinder Panwar is an Information Developer at IBM India Software Lab. He has created technical documentation on web technologies, client server, and distributed applications, and has 11 years of experience in technical documentation. He holds a post graduation diploma in Computer Applications from the National Institute of Information Technology in Hyderabad, India.

Christine Rice is a Software Engineer in Littleton, Massachusetts. She has seven years of experience with JavaServer Faces tooling in Rational Application Developer. She holds a degree in Computer Science from Smith College. Christine has previously written articles for IBM developerWorks.

Kim Tsao is a Staff Software Developer at the IBM Toronto Lab. She has 12 years of experience in software development and has spent many years developing for WebSphere Message Broker Toolkit before joining the Rational team. Her areas of focus in Rational Application Developer include Java EE, SIP, and Java Visual Editor. She also has co-authored articles on developerWorks. Kim holds a degree in Computer Science from the University of Toronto.

Sheldon Wosnick is an Advisory Software Developer in the IBM Canada Lab in Toronto, Ontario. He has 14 years of experience in software development and holds a degree from York University. His areas of expertise include Rational Application Developer and WebSphere Application Server with a specific focus on Cloud Computing, SCA, and server integration and tools.

Jim Zhang is the Senior Architect of the web development tools for Rational Application Developer at IBM RTP software development lab. He has 11 years of software development experience with Java. Jim holds a Masters degree in Computer Sciences from Northern Illinois University.

Additional contributors

Thanks to the following people for their contributions to this project:

- ▶ Anita Rass Wan, Product Manager - Rational Application Developer
- ▶ Billy Rowe, Manager - Rational Web Development Tools
- ▶ Jay Cagle, Manager - Rational Web Page Tools
- ▶ Prasad Kashyap, Rational Web Development Tools
- ▶ Ian Tewksbury, Rational Web Development Tools
- ▶ Kevan Holdaway, Rational Web Development Tools
- ▶ Nick Sandonato, Rational Web Development Tools
- ▶ Orlando Ezequiel Rincón Ferrera, Rational Web Development Tools
- ▶ Justin Berstler, Rational Web Development Tools
- ▶ Musa Yassin, Rational Web Development Tools
- ▶ Heidi Stadel, Rational Application Developer User Assistance
- ▶ Chris Brealey, Rational Application Developer Senior Technical Staff Member
- ▶ Tim DeBoer, Rational Application Developer Senior Technical Staff Member

- ▶ Daniel Lee, Rational Requirements Composer Web UI
- ▶ Jonathan West, Eclipse Test and Performance Tools Platform
- ▶ Elson Yuen, WebSphere Server Tools
- ▶ Arun Shivaswamy, Rational Application Developer Analysis, Design, and Construction
- ▶ Zina Mostafia, Rational Tools for OSGi Applications
- ▶ Sean Zhou, Rational SCA Tools
- ▶ William Smith, Rational Market and Product Manager
- ▶ Hollis Chui, Rational Architectural Management Release Team
- ▶ Kathy Chan, Test and Profiling Tools
- ▶ Mike Reid, Test and Performance Tools
- ▶ Joel Cayne, Test and Profiling Tools
- ▶ Troy Bishop, Rational Application Developer Level 3 Support
- ▶ Rodrigo Dombrowski, Advisory Software Engineer
- ▶ Ivy Ho, Senior Development Advisor, Rational
- ▶ Neeraj Agrawal, Rational Java EE Tools
- ▶ Umberto Ghio, ClearQuest® Support
- ▶ Francois Panaget, Rational RCS EMEA
- ▶ Fred Bickford, Software Advisory Team
- ▶ Kate Price, Information Development for Rational Application Developer
- ▶ Frances Overby, WebSphere Application Server Information Development
- ▶ Marie Wagner, WebSphere Information Development
- ▶ Dusko Mistic, Aurora Modeling UML
- ▶ Anthony Hunter, Eclipse Open Source Components Development
- ▶ Yury Kats, JSF tooling for Rational Application Developer
- ▶ Yen Lu, WebSphere Web Services Tools Development
- ▶ Valentin Baciu, Rational SCA Tools
- ▶ Kim Tsao, SIP Tools/Java Visual Editor
- ▶ Andrew Ivory, CEA Widget Development
- ▶ Rebecca Nin, Data Beans
- ▶ Marichu Scanlon, ODS Team Lead
- ▶ Sal Ledezma, Data Studio Common Components Development

- ▶ Charles Hart, Team System and Integration Test
- ▶ Morris Kwan, Debugger Development
- ▶ Heidi Stadel Kan, Rational Application Developer User Assistance
- ▶ Pavan Ananth, Rational Analysis, Design and Construction
- ▶ John Pitman, Rational Application Developer Release Architect
- ▶ Markus Keller, Eclipse JDT UI lead, Text and Platform UI committee
- ▶ Rajiv Senthilnathan, Rational Application Developer Tools
- ▶ Steven Hung, Rational Application Developer Tools
- ▶ Raymond Lai, Rational Application Developer XML Web Services Tools
- ▶ Mattia Parigiani, Rational Application Developer - Application Build
- ▶ Patricio Reyna Almandos, Technical Sales
- ▶ Fernando Gomez, Technical Sales
- ▶ Ashutosh Dhiman, Rational Application Developer Portal Toolkit
- ▶ Awanish K Singh, Rational Application Developer Portal Toolkit
- ▶ Gaurav Bhattacharjee, Rational Application Developer Portal Toolkit
- ▶ Jaspreet Singh, Rational Application Developer Portal Toolkit
- ▶ Manish Aneja, Rational Application Developer Portal Toolkit
- ▶ Puneet Babbar, Rational Application Developer Portal Toolkit
- ▶ Brian Pulito, SIP Development
- ▶ Asaf Zinger, SIP Container Lead
- ▶ Chris Jeffs, Rational Application Developer Information Development

Thanks to the authors of the previous edition of this book:

- ▶ Authors of the previous edition, *Rational Application Developer V7.5 Programming Guide*, SG24-7672, were Ueli Wahli, Miguel Gomes, Brian Hainey, Ahmed Moharram, Juan Pablo Napoli, Marco Rohr, Henry Cui, Patrick Gan, Celso Gonzalez, Pinar Ugurlu, and Lara Ziosi

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance

and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- ▶ Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- ▶ Look for us on LinkedIn:

<http://www.linkedin.com/groups?home=&gid=2130806>

- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- ▶ Stay current on recent Redbooks publications with RSS Feeds:
<http://www.redbooks.ibm.com/rss.html>



Part 1

Introduction to Rational Application Developer

In this part, we introduce IBM Rational Application Developer for WebSphere Software. The introduction includes packaging, product features, the Eclipse base, installation, licensing, migration, and an overview of the tools. We then discuss setting up the workbench, the perspectives, views, and editors, and the various types of projects, and introduce Unified Modeling Language (UML).

This part contains the following chapters:

- ▶ Chapter 1, “Introduction” on page 3
- ▶ Chapter 2, “Programming technologies” on page 17
- ▶ Chapter 3, “Workbench setup and preferences” on page 71
- ▶ Chapter 4, “Perspectives, views, and editors” on page 91
- ▶ Chapter 5, “Projects” on page 141
- ▶ Chapter 6, “Unified Modeling Language” on page 173



Introduction

IBM Rational Application Developer for WebSphere Software V8.0.1 is an integrated development environment (IDE). It is a platform for building Java Platform, Standard Edition (Java SE) and Java Platform, Enterprise Edition (Java EE) applications. Beyond this function, Rational Application Developer provides development tools for technologies, such as OSGi, Service Component Architecture (SCA), Web 2.0, and XML. Rational Application Developer has a focus on applications to be deployed to IBM WebSphere Application Server and IBM WebSphere Portal Server.

In this chapter, we provide an introduction to the concepts, packaging, and features of the IBM Rational Application Developer product.

The chapter is organized into the following sections:

- ▶ Concepts
- ▶ Rational Application Developer supported platforms and databases
- ▶ New features and specifications
- ▶ Migration
- ▶ Sample code
- ▶ Summary

1.1 Concepts

This section provides an introduction to the IBM Rational Software Delivery Platform, Eclipse, and Rational Application Developer.

The Rational product suite helps businesses and organizations manage the entire software development and delivery process. Software modelers, architects, developers, and testers can use the same team-unifying Rational Software Delivery Platform tooling to be more efficient in exchanging assets, following common processes, managing change and requirements, maintaining status, and improving quality.

1.1.1 IBM Rational Software Delivery Platform

The Rational Software Delivery Platform offers an array of products, services, and best practices. It is an open, modular, and proven solution that spans the entire software and systems delivery life cycle. Its products are composed of five life-cycle categories. Figure 1-1 on page 5 shows each of the life-cycle categories with a selection of the embedded Rational tooling.

For more information about the Rational Software Delivery Platform Strategy, see this website:

<http://www.ibm.com/software/rational/strategy>

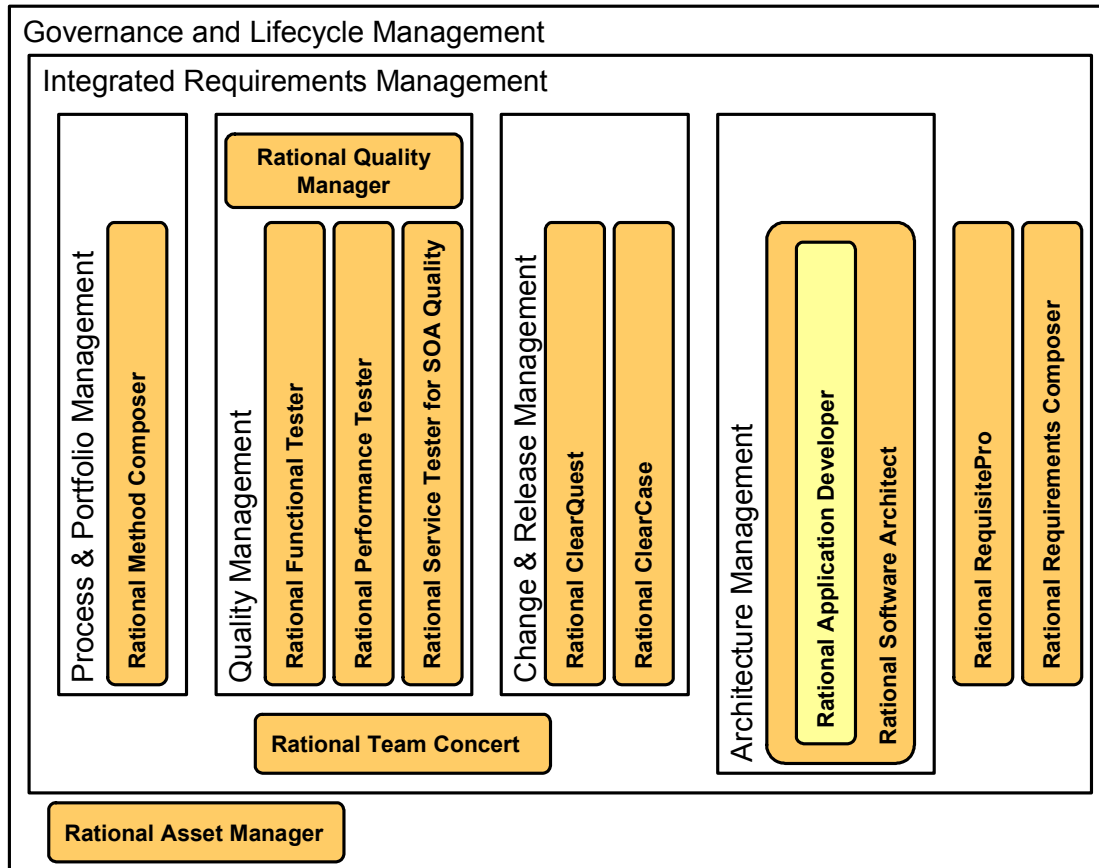


Figure 1-1 Rational Software Delivery Platform life-cycle categories and products

Here is a brief description of the products included in the IBM Rational Software Delivery Platform:

► Rational Software Architect

Rational Software Architect is a design and construction tool that uses model-driven development with UML 2.0 to create well-architected applications, including those applications that are based on service-oriented architecture (SOA).

Rational Software Architect unifies modeling, Java structural review, web services, Java SE, Java EE, database, XML, web development, and process guidance for architects and senior developers creating applications in Java. Rational Software Architect provides business process design through support for BPEL and BPMN, and integrates with WebSphere Integration Developer and WebSphere Business Monitor.

- ▶ Rational Application Developer for WebSphere Software

Rational Application Developer is a full suite of development, analysis and test, and deployment tools for rapidly implementing Java SE and EE, Portal, Web and Web 2.0, web services, OSGi, and SOA applications.

Rational Application Developer is available in two editions: Rational Application Developer for WebSphere Software, and Rational Application Developer Standard Edition for WebSphere Software. Standard Edition contains all the features of the full Rational Application Developer except for WebSphere adapter support, team development, code quality, testing and deployment, and code visualization. For more information about the two editions of Rational Application Developer, see this website:

http://www.ibm.com/software/awdtools/developer/application/features/index.html?S_CMP=wspace

- ▶ Rational Asset Manager

Rational Asset Manager helps create, modify, govern, find, and reuse any type of development assets, including SOA and system development assets.

- ▶ Rational Team Concert

Rational Team Concert is a Jazz™ collaborative software delivery environment that empowers project teams to simplify, automate, and govern software delivery. Automated data collection and reporting reduces administrative overhead and provides the real-time insight required to effectively govern software projects. It extends the capabilities of the team with integrated work items, release planning, build, software configuration management (SCM), and the collaborative infrastructure of the Jazz Team Server.

Jazz: Jazz is a technology platform for collaborative software delivery from IBM Rational. It is an extensible framework that dynamically integrates and synchronizes people, processes, and assets associated with software development projects. For more information about the Jazz technology platform, see the following web page:

<http://www.ibm.com/software/rational/jazz/>

- ▶ Rational Functional Tester

Rational Functional Tester is a tool for automated functional, regression, GUI, and data-driven testing. It provides the capability to record robust scripts that can be played back to validate new builds of an application.

- ▶ Rational Performance Tester
Rational Performance Tester supports the creation, execution, and analysis of performance tests to measure application scalability and performance.
- ▶ Rational Service Tester for SOA Quality
Rational Service Tester for SOA Quality tooling provides the tester with script-free testing capabilities for functional, regression, and performance testing of web services.
- ▶ Rational Quality Manager
Rational Quality Manager is a Jazz-based and Web-based centralized test management environment for business, system, and IT decision makers, and quality professionals who seek a collaborative and customizable solution for test planning, workflow control, tracking, and metrics reporting capable of quantifying how project decisions and deliverables impact and align with business objectives.
- ▶ Rational Method Composer
Rational Method Composer is a flexible process management platform. It includes a rich process library to help companies implement effective processes for successful software and IT projects.
- ▶ Rational RequisitePro®
RequisitePro is a requirements management tool for project teams who want to manage their requirements, write good use cases, improve traceability, strengthen collaboration, reduce project risks, and increase quality.
- ▶ Rational Requirements Composer
Requirements Composer is a Jazz-based tool that helps teams define and use requirements effectively across the project life cycle.
- ▶ Rational ClearQuest
ClearQuest is a software change management tool. It provides defect, task, and change request tracking, process automation, reporting, and life-cycle traceability for better visibility and control of the software development life cycle.
- ▶ Rational ClearCase
ClearCase is a complete software configuration management tool. It provides sophisticated version control, workspace management, parallel development support, and build auditing to improve productivity.

1.1.2 Eclipse and IBM Rational Software Delivery Platform

This section provides an overview of the Eclipse Project and illustrates how Eclipse relates to the IBM Rational Software Delivery Platform and Rational Application Developer.

Eclipse Project

The Eclipse Project is an open source software development project devoted to creating a development platform and integrated tooling. Figure 1-2 shows the high-level Eclipse Project architecture and shows the relationship of the following subprojects:

- ▶ Eclipse Platform
- ▶ Eclipse Java development tools
- ▶ Eclipse Plug-in Development Environment

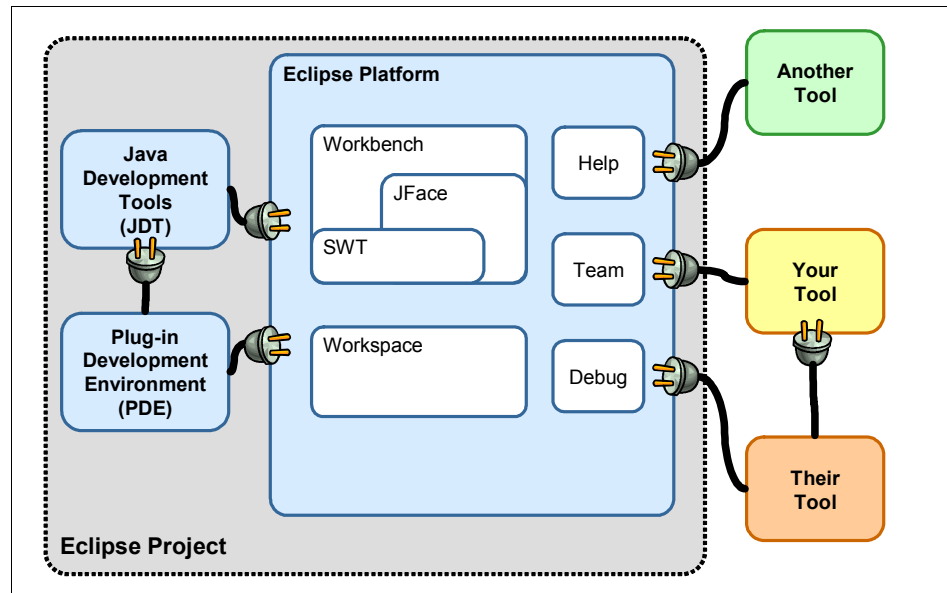


Figure 1-2 Eclipse Project overview

With a common public license that provides royalty-free source code and worldwide redistribution rights, the Eclipse Platform provides tool developers with great flexibility and control over their software technology.

Industry leaders, such as IBM, Borland, Merant, QNX Software Systems, Red Hat, SUSE, TogetherSoft, and WebGain, formed the initial eclipse.org board of directors of the Eclipse open source project.

Eclipse V3.6: IBM Rational Application Developer is based on Eclipse V3.6.

For more information about Eclipse, see the following website:

<http://www.eclipse.org>

Eclipse Platform

The Eclipse Platform provides a framework and services that serve as a foundation for tools developers to integrate and extend the functionality of the platform. The platform includes a workbench, concept of projects, user interface libraries (JFace and SWT), built-in help engine, and support for team development and debug. Various software development tasks can use this platform, including modeling and architecture, IDE (Java, C/C++, and COBOL), testing, and so on.

1.1.3 Challenges in application development

To better grasp the business value that Rational Application Developer provides, it is important to understand the challenges that businesses face in application development.

Table 1-1 highlights the key application development challenges and desired development tooling solutions.

Table 1-1 Application development challenges

Challenges	Solution tooling
Application development is complex, time consuming, and error prone.	Raise productivity by automating time-consuming and error-prone tasks.
Highly skilled developers are required and in short supply.	Assist less knowledgeable developers where possible by providing wizards, online context sensitive help, an integrated environment, and visual tooling.
Learning curves are long.	Shorten learning curves by providing rapid application development tooling (visual layout and design, reusable components, and code generators) and ensure that development tools have a consistent way of working.

1.2 Rational Application Developer supported platforms and databases

In this section, we describe the platforms and databases that are supported by Rational Application Developer.

1.2.1 Supported operating system platforms

Rational Application Developer supports the following operating systems:

- ▶ Microsoft® Windows® 7 Professional/Enterprise/Ultimate Editions (x32/x64)
- ▶ Microsoft Windows Vista Professional/Enterprise/Ultimate Editions (x32/x64)
- ▶ Microsoft Windows XP Professional (x32/x64)
- ▶ Microsoft Windows Server 2008 Standard/Enterprise Edition (x32/x64)
- ▶ Red Hat Enterprise Linux® Server 5 (x32/x64)
- ▶ Red Hat Enterprise Linux Desktop 5 (x32)
- ▶ SUSE Linux Enterprise Server Version 10/11 (x32/x64)
- ▶ SUSE Linux Enterprise Desktop Version 10/11 (x32/x64)
- ▶ Ubuntu Linux 10.4 LTS
- ▶ Citrix Presentation Server Version 4/4.5/5

1.2.2 Supported runtime environments

The following IBM application servers are supported by Rational Application Developer:

- ▶ IBM WebSphere Application Server Version 6.0, including the Feature Pack for Web 2.0

Note: WebSphere Application Server V6 does not ship with Rational Application Developer. To use WebSphere Application Server V6 with Rational Application Developer, install a stand-alone WebSphere Application Server V6.0 server separately and connect as a local or remote server.

- ▶ IBM WebSphere Application Server Version 6.1, including the Feature Pack for Web Services, Feature Pack for EJB 3.0, and Feature Pack for Web 2.0
- ▶ IBM WebSphere Application Server V7.0, including the Feature Pack for Service Component Architecture, Feature Pack for XML, Feature Pack for OSGi Applications and Java Persistence API 2.0, and the Feature Pack for Web 2.0
- ▶ IBM WebSphere Application Server Version 8.0 Beta

- ▶ IBM WebSphere Portal Server V6.1 and IBM WebSphere Portal Server 6.1 on WebSphere Application Server V7
- ▶ IBM WebSphere Portal Server Version 7.0

Additional server adapters are supported by the Web Tools Platform 3.0 based on Eclipse technology, which is included in Rational Application Developer. The server adapters in the following list are included, by default, in the Web Tools Platform installed with Rational Application Developer:

- ▶ Apache Tomcat versions 3.2, 4.0, 4.1, 5.0, 5.5, 6.0, and 7.0
- ▶ JBoss versions 3.2, 3.3, 4.0, 4.2, and 5.0
- ▶ ObjectWeb Java Open Application Server (JOnAS) Version 4
- ▶ Oracle Containers for J2EE (OC4J) Standalone Server versions 10.1.3 and 10.1.3.n

Supported databases

The following databases are compatible with Rational Application Developer:

- ▶ IBM Cloudscape Version 5.1
- ▶ IBM DB2® UDB versions 7.2, 8.1, 8.2, 9.1, 9.5, and 9.7
- ▶ IBM DB2 UDB iSeries® versions 5R2, 5R3, and 5R4
- ▶ IBM DB2 for z/OS® V7 and versions 8 and 9 (compatibility mode and new function mode)

New function mode: In addition to the compatibility mode, the new function mode includes the generated data model that has all the new catalog features of DB2 for z/OS V8 and V9. Use the new function mode if you plan to work with the generated data models available in IBM Rational Software Delivery Platform products.

- ▶ Apache Derby versions 10.0, 10.1, 10.2, and 10.3
- ▶ Generic Java Database Connectivity (JDBC) Version 1.0
- ▶ IBM Informix® Dynamic Server versions 9.2, 9.3, 9.4, 10.0, 11.1, and 11.5
- ▶ MySQL versions 4.0 and 4.1
- ▶ Oracle versions 8, 9, 10, and 11g
- ▶ Microsoft SQL Server Enterprise versions 2000 and 2005
- ▶ Sybase Adaptive Server Enterprise versions 12.x and 15

1.3 New features and specifications

In this section, we provide a summary of the new features and specifications supported by Rational Application Developer.

1.3.1 New features in Rational Application Developer

Rational Application Developer has many new features that we highlight in detail in the remaining chapters of this book.

There have been major advances and additions in product features and technology since Rational Application Developer V7.5.0 in the following areas:

- ▶ Assembly and deployment tools
- ▶ Code Coverage analysis
- ▶ IBM Rational Desktop Connection Toolkit for Cloud Environments
- ▶ Debug tools
- ▶ Java EE 6
- ▶ Java EE Connector (J2C) Tools
- ▶ JSF Tools
- ▶ OSGi application development tools
- ▶ Page designer
- ▶ Portal tools
- ▶ Profiling tools
- ▶ Rational License Key Server update
- ▶ Service Component Architecture (SCA) Tools
- ▶ Token licensing
- ▶ UML modeling
- ▶ Web 2.0
- ▶ Web development tools
- ▶ Web Services
- ▶ XML tools

For detailed information about new features and enhancements introduced in Rational Application Developer, refer to the following resources:

- ▶ Rational Application Developer for WebSphere Software, Version 8.0: New features and enhancements

<http://www.ibm.com/support/docview.wss?uid=swg27018924>

- ▶ Rational Application Developer Version 8.0: What's New

<http://www.ibm.com/developerworks/wikis/display/rad/Rational+Application+Developer+Version+8.0+-+What%27s+New>

To request new features to be added to future versions on Rational Application Developer, visit the IBM RFE Community and open your requirements for IBM to consider:

<http://www.ibm.com/developerworks/support/rational/rfe/>

1.3.2 Specification versions

In this section, we highlight the specification versions in Rational Application Developer.

Table 1-2 compares the technology versions supported by Rational Application Developer V7.5 and V8. Most of the listed technologies are part of the Java EE specification.

Table 1-2 Technology versions comparison

Specification	Rational Application Developer V7.5	Rational Application Developer V8
IBM Java Runtime Environment (JRE)	1.6	1.6
JavaServer Pages (JSP)	2.1	2.2
Java Servlet	2.5	3.0
Enterprise JavaBeans (EJB)	3.0	3.1
Java Message Service (JMS)	1.1	1.1
Java Transaction API (JTA)	1.1	1.1
JavaMail	1.4.1	1.4.1
Java Activation Framework (JAF)	1.1.1 Included in Java SE 6	1.1.1 Included in Java SE 6
Java API for XML Processing (JAXP)	1.4 Included in Java SE 6	1.4 Included in Java SE 6
Java EE Connector	1.5	1.6
Java API for XML-based RPC (JAX-RPC)	1.1	1.1
SOAP with Attachments API for Java (SAAJ)	1.3 Included in Java SE 6	1.3 Included in Java SE 6
Java API for XML Web Services (JAX-WS)	2.0	2.2
Java Architecture for XML Binding (JAXB)	2.0	2.2

Specification	Rational Application Developer V7.5	Rational Application Developer V8
Java Authentication and Authorization Service (JAAS)	Included in Java SE 6	Included in Java SE 6
Java Database Connectivity API (JDBC)	4.0 Included in Java SE 6	4.0 Included in Java SE 6
Java API for XML Registries (JAXR)	1.0	1.0
Java EE Management	1.1	1.1
Java Management Extensions (JMX)	1.2 Included in Java SE 6	1.2 Included in Java SE 6
Java EE Deployment	1.2	1.2
Java Authorization Service Provider Contract for Containers (JACC)	1.1	1.4
JavaServer Pages Debugging	1.0	1.0
JavaServer Pages Standard Tag Library (JSTL)	1.2	1.2
Web Services Metadata	2.0	2.1
JavaServer Faces	1.2	2.0
Common Annotations	1.0	1.1
Streaming API for XML (StAX)	1.0 Included in Java SE 6	1.0 Included in Java SE 6
Java Persistence API (JPA)	1.0	2.0
Service Data Objects (SDO)	2.0	2.0
Struts	1.3	1.3
OSGi Service Platform Specifications	N/A	4.2
Service Component Architecture (SCA)	1.0	1.0

1.4 Migration

Rational Application Developer can migrate workspaces and projects from Rational Application Developer versions 7.5.x and 7.0.x. Projects from these versions of Rational Application Developer are migrated with the Workspace Migration wizard. When projects from Rational Application Developer versions

7.5.x or 7.0.x are detected in the current version, the Workspace Migration wizard starts automatically and selects the projects to migrate. There are three general methods to bring projects from Version 7.5.x or Version 7.0.x into the current version:

- ▶ Export and import projects as archive files:
You can export projects from Rational Application Developer versions 7.5.x or 7.0.x with the Project Interchange feature (**File** → **Export** → **Other** → **Project Interchange**) and import them into your workspace (**File** → **Import** → **Existing projects into workspace**).
- ▶ Share projects from a source code management system
You can import Rational Application Developer V7.5.x or V7.0.x projects that exist in a source code management (SCM) system.
- ▶ Open a Rational Application Developer V7.5.x or V7.0.x workspace in the current version
When a workspace from Rational Application Developer V7.5.x or V7.0.x is opened in the current version, the projects within that workspace can be migrated to the current version. The workspace itself is also migrated; therefore, the workspace can no longer be loaded by previous versions of Rational Application Developer.

The following application server run times that were available in Rational Application Developer versions 7.5.x or 7.0.x are not supported in this version:

- ▶ WebSphere Application Server V5.1.x
- ▶ WebSphere Application Server V5.1.x Express
- ▶ WebSphere Portal Server V6.0.x
- ▶ WebSphere Portal Server V5.1.x

If you have a project that targets any of these server run times, you can change the target run time during the migration process.

1.5 Sample code

The chapters in this book are written so that you can follow along and create the code from the beginning. In places where a significant amount of typing is involved, we provide snippets of code for you to cut and paste.

Alternatively, you can import the completed sample code from an existing project file. For details about the sample code, including details for downloading and unpacking the code, a description of the code, and instructions for importing the

project files and creating databases, see Appendix C, “Additional material” on page 1877.

1.6 Summary

In this chapter, we have introduced the concepts behind Rational Application Developer, provided an overview of the features of the various members of the Rational product suite, and described where Rational Application Developer fits with the other products. We also provided a summary of the version numbers of the various features.



Programming technologies

This chapter highlights the tools that are provided by Rational Application Developer to facilitate a series of application development scenarios. Throughout this chapter, we use a simple banking application to illustrate these scenarios.

This chapter is organized into the following sections:

- ▶ Desktop applications
- ▶ Web applications
- ▶ Enterprise JavaBeans and Java Persistence API
- ▶ Web services
- ▶ Messaging systems
- ▶ OSGi applications
- ▶ Other applications

2.1 Desktop applications

Desktop applications are applications that run on a single machine and the user interacts directly with the application by using a user interface (UI) on the same machine. When this idea is extended to include database access, part of the work might be performed by another process, possibly on another machine. Although this situation begins to move us into the client/server environment, often the application uses only the database as a service. The user interface, business logic, and control of flow are all implemented within the desktop application. This concept contrasts with full client/server applications in which these elements are clearly separated and might be provided by separate technologies running on separate machines.

This type of application is the simplest type that we consider. Many of the technologies and tools involved in developing desktop applications, such as the Java editor and the Extensible Markup Language (XML) tooling, are used widely in Rational Application Developer.

The first scenario deals with a situation in which a bank requires a desktop application to allow workers in a bank call center to view and update customer account information. We call this scenario the *Call Center Desktop*.

2.1.1 Simple desktop applications

A starting point for the Call Center Desktop might be a simple stand-alone application designed to run on desktop computers without a separate database process hosted on a server machine.

Java Platform, Standard Edition (Java SE) provides all the elements necessary to develop such applications. It includes, among other elements, a complete object-oriented programming language specification, a wide range of useful classes to speed development, and a runtime environment on which programs can be executed.

For the complete Java SE specification, see the following web address:

<http://www.oracle.com/technetwork/java/javase/overview/index.html>

Java language

Java is a general-purpose, object-oriented language. The basic language syntax is similar to C and C++, although there are significant differences. Java is a higher-level language than C or C++, in that the developer is presented with a more abstract view of the underlying computer hardware and is not expected to take direct control of issues, such as memory management. The compilation

process for Java does not produce directly executable binaries, but rather an intermediate byte code, which can be executed directly by a virtual machine or can be further processed by a just-in-time compiler at run time to produce platform-specific binary output.

New in Java Platform, Standard Edition, Version 6.0

Version 6 of the Java Platform, Standard Edition (Java SE 6) includes many useful new features:

- ▶ **Web Services:** New support is provided for writing XML web service client applications. Parsing and XML to Java object APIs, previously only available in the Java Web Services Pack and Java Platform, Enterprise Edition (Java EE) platform implementations, have been added to Java SE. The following list highlights important support for Web Services in Java SE 6:
 - *Java Specification Request (JSR) 173 Streaming API for XML (StAX):* Java-based API for pull-parsing XML
<http://jcp.org/en/jsr/detail?id=173>
 - *JSR 181 Web Services Metadata:* An annotated Java format to enable the easy definition of Java Web Services in a Java EE container
<http://jcp.org/en/jsr/detail?id=181>
 - *JSR 222 Java Architecture for XML Binding (JAXB) 2.0:* Next generation of the API that makes it easier to access XML documents from Java applications
<http://jcp.org/en/jsr/detail?id=222>
 - *JSR 224 Java API for XML-based Web Services (JAX-WS) 2.0:* Next generation web services API replacing JAX-RPC 1.0
<http://jcp.org/en/jsr/detail?id=224>
- ▶ **Scripting:** JavaScript technology source code can now be mixed with normal Java source code, which might be useful for prototyping purposes:
 - *JSR 223 Scripting for the Java Platform:* Scripting language programs can access information developed in Java and can use scripting language pages in Java server-side applications.
<http://jcp.org/en/jsr/detail?id=223>
- ▶ **More Desktop APIs:** A `SwingWorker` class has been added that helps graphical user interface (GUI) developers with implementing tasks in a worker thread in GUI applications. `JTable` now includes sorting, filtering, and highlighting possibilities. In addition, a new facility for quickly presenting splash screens to users is now available.

- ▶ Database: JDK co-bundles the Java DB, a pure Java Database Connectivity (JDBC) database, based on Apache Derby. JDBC API support has been updated to 4.0: It now supports XML as an SQL data type and integrates better with Binary Large Objects (BLOBs) and Character Large Objects (CLOBs) types:
 - *JSR 221 JDBC 4.0*: Java application access to SQL stores
<http://jcp.org/en/jsr/detail?id=221>
- ▶ Monitoring and Management: More diagnostic information has been added and the memory-heap analysis tool *jhat* for forensic explorations of core dumps is included.
<http://download.oracle.com/javase/6/docs/technotes/tools/share/jhat.html>
- ▶ Compiler Access: Java development tool and framework creators get a programmatic access to **javac** for in-process compilation of dynamically generated Java code:
 - *JSR 199 Java Compiler API*: Service provider that allows a Java program to select and invoke a Java Language Compiler programmatically
<http://jcp.org/en/jsr/detail?id=199>
- ▶ Pluggable Annotations: Pluggable Annotations help you define your own annotations and give you core support for plug-ins and executing the annotation processors:
 - *JSR 269 Pluggable Annotation Processing API*: Creating and processing custom annotations
<http://jcp.org/en/jsr/detail?id=269>
- ▶ Desktop Deployment: Desktop Deployment offers a better platform look-and-feel in the following areas:
 - Swing
 - Liquid Crystal Display text rendering
 - Higher GUI performance
 - Better integration of native platforms
 - New access to the system tray and start menu of the platform
 - Unification of Java Plug-in technology and Java WebStart engines
- ▶ Security: The XML Digital Signature API has been added to allow the creation and manipulation of digital signatures.

Also available is simplified access to native security services, such as native public key infrastructure (PKI), cryptographic services on Microsoft Windows for secure authentication and communication, Java Generic Security Services (Java GSS), Kerberos services for authentication, and access to Lightweight Directory Access Protocol (LDAP) servers:

- *JSR 105 XML Digital Signature APIs (XML-DSIG)*: Implementation of the W3C specification

<http://jcp.org/en/jsr/detail?id=105>

- ▶ Libraries (quality, compatibility, and stability): Libraries support array relocation, the new collection type Deque (*double-ended queue*, a linear collection that supports element insertion and removal at both ends), sorted sets and maps with bidirectional navigation, new core IEEE754 (floating point) functions, a new password prompting feature, and an update of the Java Class File specification:

- *JSR 202 Java Class File Specification Update*: Increases class file size limits and adds split verification support

<http://jcp.org/en/jsr/detail?id=202>

Java virtual machine

The Java virtual machine (JVM) is a runtime environment designed for executing compiled Java byte code, contained in `Java.class` files, which results from the compilation of Java source code. Several types of JVMs exist, ranging from simple interpreters to those JVMs including just-in-time compilers that dynamically translate byte code instructions to platform-specific instructions, as required.

Requirements for the development environment

The developer of the Call Center Desktop must have access to a development tool that provides a range of features to enhance developer productivity:

- ▶ A specialized code editor, providing syntax highlighting
- ▶ Assistance with completing code and correcting syntactical errors
- ▶ Facilities for visualizing the relationships between the classes in the application
- ▶ Assistance with documenting code
- ▶ Automatic code review functionality to ensure that code is being developed according to recognized best practices
- ▶ A simple way of testing, analyzing, and debugging applications

Rational Application Developer provides developers with an integrated development environment with these features.

2.1.2 Database access

Most likely, the Call Center Desktop accesses data residing in a relational database, such as IBM DB2 Universal Database™.

Java SE 6.0 includes several integration technologies:

- ▶ JDBC is the Java standard technology for accessing data stores.
- ▶ Java Remote Method Invocation (RMI) is the standard way of enabling remote access to objects within Java.
- ▶ Java Naming and Directory Interface (JNDI) is the standard Java interface for naming and directory services.
- ▶ Java IDL is the Java implementation of the Interface Definition Language (IDL) for Common Object Request Broker Architecture (CORBA), allowing Java programs to access objects hosted on CORBA servers.

We focus on the Java Database Connectivity (JDBC) technology in this section.

JDBC

Java SE 6.0 includes support for JDBC 4.0. You can download the specification from the following web address:

<http://jcp.org/aboutJava/communityprocess/final/jsr221/index.html>

Although JDBC supports a wide range of data store types, most often, JDBC accesses relational databases by using SQL. Classes and interfaces are provided to simplify database programming:

- ▶ `java.sql.DriverManager` and `javax.sql.DataSource` can be used to obtain a connection to a database system.
- ▶ `java.sql.Connection` represents the connection that an application has to a database system.
- ▶ `java.sql.Statement`, `PreparedStatement`, and `CallableStatement` represent executable statements that can be used to update or query the database.
- ▶ `java.sql.ResultSet` represents the values returned from a statement that has queried the database.
- ▶ Various types, such as `java.sql.Date` and `java.sql.Blob`, are Java representations of SQL data types that do not have a direct equivalent primitive type in Java.

Requirements for the development environment

The development environment must provide access to all the facilities of JDBC 4.0. However, because JDBC 4.0 is part of Java SE 6.0, we cover this requirement in 2.1.1, “Simple desktop applications” on page 18. In addition, the development environment has the following requirements:

- ▶ A way of viewing information about the structure of an external database
- ▶ A mechanism for viewing sample contents of tables
- ▶ Facilities for importing structural information from a database server so that it can be used as part of the development process
- ▶ Wizards and editors allowing databases, tables, columns, relationships, and constraints to be created or modified
- ▶ A feature to allow databases created or modified in this way to be exported to an external database server
- ▶ A wizard to help create and test SQL statements

With these features, developers can develop test databases and work with production databases as part of the overall development process. Database administrators can also use them to manage database systems, although they might prefer to use dedicated tools provided by the vendor of their database systems.

IBM Rational Application Developer includes these features.

2.1.3 Graphical user interfaces

A further enhancement of the Call Center Desktop is to make the application easier to use by providing a GUI.

Abstract Window Toolkit

The Abstract Window Toolkit (AWT) is the original GUI toolkit for Java. It has been enhanced since it was originally introduced, but the basic structure remains the same. The AWT includes the following items:

- ▶ A wide range of user interface components, represented by Java classes such as [java.awt.] Frame, Button, Label, Menu, and TextArea
- ▶ An event-handling model to deal with events, such as button clicks, menu choices, and mouse operations
- ▶ Classes to deal with graphics and image processing
- ▶ Layout manager classes to help with the positioning of components in a GUI
- ▶ Support for drag-and-drop functionality in GUI applications

The AWT is implemented natively for the JVM of each platform. AWT graphical interfaces typically perform relatively quickly and have the same look-and-feel as the operating system. However, the range of GUI components that can be used is limited to the lowest common denominator of operating system components, and the look-and-feel cannot be changed.

For more information about the AWT, see the following web address:

<http://download.oracle.com/javase/6/docs/technotes/guides/awt/>

Swing

Swing is a newer GUI component framework for Java. It provides Java implementations of the components in the AWT and adds a number of more sophisticated GUI components, such as tree views and list boxes. For the basic components, Swing implementations have the same name as the AWT component with a J prefix and a separate package structure, for example, `java.awt.Button` becomes `javax.swing.JButton` in Swing.

Swing GUIs do not normally perform as quickly as AWT GUIs, but they have a richer set of controls and have a pluggable look-and-feel.

For more information about Swing, see the following web address:

<http://download.oracle.com/javase/6/docs/technotes/guides/swing/>

Standard Widget Toolkit

The Standard Widget Toolkit (SWT) is the GUI toolkit that is provided as part of the Eclipse Project and used to build the Eclipse GUI. The SWT is written entirely in Java and uses the Java Native Interface (JNI) to pass the calls through to the operating system where possible, which is done to avoid the *lowest common denominator* problem. The SWT uses native calls where they are available and builds the component in Java where they are not.

Widget: In the context of windowing systems, a *widget* is a reusable interface component, such as a menu, scroll bar, button, text box, or label.

In many respects, the SWT provides the best of both worlds (AWT and Swing):

- ▶ It has a rich, portable component model, such as Swing.
- ▶ It has the same look-and-feel as the native operating system, such as the AWT.
- ▶ GUIs that have been built by using the SWT perform well, such as the AWT, because most of the components pass through to operating system components.

A disadvantage of the SWT is that, unlike the AWT and Swing, it is not a standard part of Java SE V6.0. Consequently, any application that uses the SWT has to be installed along with the SWT class libraries. However, the SWT, like the rest of the components that make up Eclipse, is open source and freely distributable under the terms of the Common Public License.

For more information about the SWT, see the following web address:

<http://www.eclipse.org/swt/>

Another technology that is based on SWT and Eclipse is the Eclipse Rich Client Platform (RCP). The architecture of Eclipse allows its components to be used to create client applications.

For more information about Eclipse RCP, see the following web address:

http://wiki.eclipse.org/index.php/Rich_Client_Platform

Java implementations providing a GUI

Two types of Java components might provide a GUI:

- ▶ Stand-alone Java applications: Started in their own process (JVM). This category might include Java EE application clients, which we describe later.
- ▶ Java applets: Normally, the Java applets are run in a JVM provided by a web browser or a web browser plug-in.

An applet normally runs in a JVM with a strict security model, by default. The applet is not allowed to access the file system of the machine on which it is running. The applet can make network connections only back to the machine from which it was originally loaded. Consequently, applets are not normally suitable for applications that require access to databases, because this situation might require the database to reside on the same machine as the web server. If the security restrictions are relaxed, as might be possible if the applet was being used only on a company intranet, this problem is not encountered.

An applet is downloaded on demand from the website that hosts it. This design is advantageous in that the latest version automatically downloads each time that it is requested, so distributing new versions is trivial. This design is disadvantageous in that often the applet is downloaded several times even if it has not changed, which is a pointless use of bandwidth, and the developer has little control over the environment on which the applet runs.

Requirements for the development environment

The development environment must provide a specialized editor that allows a developer to design GUIs by using various component frameworks (such as AWT, Swing, or SWT). The developer must be able to focus mainly on the visual

aspects of the layout of the GUI, rather than the coding that lies behind it. Where necessary, the developer must be able to edit the generated code to add event-handling code and business logic calls. The editor must be dynamic, reflecting changes in the visual layout immediately in the generated code and changes in the code immediately in the visual display. The development environment must also provide facilities for testing visual components that make up a GUI and the entire GUI. Rational Application Developer has a visual editor with this functionality included.

2.1.4 Extensible Markup Language (XML)

Communication between computer systems is often difficult, because separate systems use separate data formats for storing data. XML has become a common way to resolve this problem.

It might be desirable for the Call Center Desktop application to exchange data with other applications. For example, we might want to export tabular data so that it can be read into a spreadsheet application to produce a chart. Or, we might want to read information about a group of transactions that can then be carried out as part of an overnight batch operation.

A convenient technology for exchanging information between applications is XML, which is a standard, simple, and flexible way of exchanging data. The structure of the data is described in the XML document itself. Mechanisms are available for ensuring that the structure conforms to an agreed format. These mechanisms are known as *Document Type Definitions* (DTDs) and *XML Schema Definitions* (XSDs).

XML is increasingly also being used to store configuration information for applications. For example, many aspects of Java EE use XML for configuration files called *deployment descriptors*, and WebSphere Application Server uses XML files for storing its configuration settings. OSGi Blueprint files and Service Component Architecture (SCA) composite files are written in XML.

For more information about XML, see the World Wide Web Consortium (W3C) at the following address:

<http://www.w3.org/XML/>

Using XML in Java code

Java SE 6.0 includes the Java API for XML Processing (JAXP). JAXP contains several elements:

- ▶ A parser interface based on the Document Object Model (DOM) from the W3C, which builds a complete internal representation of the XML document

- ▶ The Simple API for XML (SAX), which allows the document to be parsed dynamically by using an event-driven approach
- ▶ Extensible Stylesheet Language Transformation (XSLT) 1.0, which uses Extensible Stylesheet Language (XSL) to define how to transform XML documents from one form into another
- ▶ XPath V1.0 for processing XML
- ▶ The Streaming API for XML (StAX), which supports an iterative, event-based approach to reading and writing XML documents

Because JAXP is a standard part of Java SE V6.0, all these features are available in any Java code running in a JVM.

The IBM WebSphere Application Server Feature Pack for XML also delivers support for the following versions of the XML processing languages:

- ▶ XPath 2.0
- ▶ XSLT 2.0
- ▶ XQuery 1.0

Requirements for the development environment

In addition to allowing developers to write code to create and parse XML documents, the development environment must provide features, such as the following features, to help developers to create and edit XML documents and related resources:

- ▶ An XML editor that checks the XML document for well-formedness (conformance with the structural requirements of XML) and for consistency with a DTD or XML schema
- ▶ Wizards for these tasks:
 - Creating XML documents from DTDs and XML schemas
 - Creating DTDs and XML schemas from XML documents
 - Converting between DTDs and XML schemas
 - Generating JavaBeans to represent data stored in XML documents
 - Creating XSL
- ▶ An environment to test and debug XSLT

Rational Application Developer includes all these features.

2.2 Web applications

A *static website* is a website in which the content viewed by users accessing the site by using a web browser is determined only by the contents of the file system

on the web server machine. Because the user experience is determined only by the content of these files and not by any action of the user or any business logic running on the server machine, the site is described as *static*. Nowadays, almost all websites have a degree of dynamic behavior with Javascript (Ajax) without having to involve application servers.

In most cases, the communication protocol used for interacting with static websites is the Hypertext Transfer Protocol (HTTP).

In the context of our sample scenario, the bank might want to publish a static website to inform customers of bank services, such as branch locations and opening hours, and to inform potential customers of services provided by the bank, such as account interest rates. This client/server information can safely be provided statically, because it is the same for all visitors to the site and it changes rarely.

2.2.1 Hypertext Transfer Protocol (HTTP)

HTTP follows a request/response model. A client sends an HTTP request to the server providing information about the request method being used, the requested Uniform Resource Identifier (URI), the protocol version being used, various other header information, and often other details, such as details from a form completed on the web browser. The server responds by returning an HTTP response consisting of a status line, including a success or error code, and other header information followed by a Hypertext Markup Language (HTML) code for the static page requested by the client.

For full details of HTTP, see the following web address:

<http://www.w3.org/Protocols/>

For information about HTML, see the following web address:

<http://www.w3.org/html/>

Methods

HTTP 1.1 defines several request methods: GET, HEAD, POST, PUT, DELETE, OPTIONS, and TRACE. Of these request methods, only GET and POST are commonly used in web applications:

- ▶ GET requests are normally used in situations where the user has entered an address into the address or location field of a web browser, used a bookmark or favorite stored by the browser, or followed a hyperlink within an HTML document.
- ▶ POST requests are normally used when the user has completed an HTML form displayed by the browser and has submitted the form for processing.

This request type is most often used with dynamic web applications, which include business logic for processing the values entered into the form.

Status codes

The status code returned by the server as the first line of the HTTP response indicates the outcome of the request. In an error, this information can be used by the client to inform the user of the problem. In certain situations, such as redirection to another URI, the browser acts on the response without any interaction from the user. The status codes have the following classes:

1xx: Informational	The request has been received, and processing is continuing.
2xx: Success	The request has been correctly received and processed; an HTML page accompanies a 2xx status code as the body of the response.
3xx: Redirection	The request did not contain all the information required, or the browser needs to take the user to another URI.
4xx: Client error	The request was incorrectly formed or was not fulfilled.
5xx: Server error	Although the request was valid, the server failed to fulfill it.

The most common status code is 200 (OK), although 404 (Not Found) is commonly encountered. A complete list of status codes can be found at the W3C site mentioned previously.

Cookies

Cookies are a general mechanism that server-side connections can use to both store and retrieve information about the client side of the connection. Cookies can contain any piece of textual information, within an overall size limit per cookie of 4 KB. Cookies have the following attributes:

Name	The name of the cookie.
Value	The data that the server wants passed back to it when a browser requests another page.
Domain	The address of the server that sent the cookie and that receives a copy of this cookie when the browser requests a file from that server. The domain can be set to equal the subdomain that contains the server so that multiple servers in the same subdomain receive the cookie from the browser.
Path	Used to specify the subset of URLs in a domain for which the cookie is valid.

Expires	Specifies a date string that defines the valid lifetime of that cookie.
Secure	Specifies that the cookie is only sent if HTTP communication is taking place over a secure channel (known as <i>HTTPS</i>).

A cookie life cycle proceeds in this manner:

1. The user gets connected to a server that wants to record a cookie.
2. The server sends the name and the value of the cookie in the HTTP response.
3. The browser receives the cookie and stores it.
4. Every time that the user sends a request for a URL at the designated domain, the browser sends any cookies for that domain that have not expired with the HTTP request.
5. When the expiration date has been passed, the cookie expires.

Non-persistent cookies are created without an expiration date. They only last for the duration of the user browser session. Persistent cookies are set one time and remain on the user hard drive until the expiration date of the cookie. Cookies are widely used in dynamic web applications, which we address later in this chapter, for associating a user with server-side state information.

For more information about cookies, see the following web address:

<http://www.cookiecentral.com/faq>

2.2.2 Hypertext Markup Language (HTML)

HTML is a language for publishing hypertext on the Web. HTML uses tags to structure text into headings, paragraphs, lists, hypertext links, and so forth. It is distinct from XML by way of having unpaired tags. Table 2-1 lists several common HTML tags.

Table 2-1 Common HTML tags

Tag	Description
<html>	Tells the browser that the following text is marked up in HTML. The closing tag </html> is required and is the last tag in your document.
<head>	Defines information for the browser that might or might not be displayed to the user. Tags that belong in the <head> section are <title>, <meta>, <script>, and <style>. The closing tag </head> is required.

Tag	Description
<title>	Shows the title of your web page and is usually displayed by the browser at the top of the browser pane. The closing tag </title> is required.
<body>	Defines the primary portion of the web page. Attributes of the <body> tag enable setting the background color, the text color, the link color, and the active and visited link colors. The closing tag </body> is required.

Cascading style sheets

Although web developers can use HTML tags to specify styling attributes, the best practice is to use a *cascading style sheet* (CSS). A CSS file defines a hierarchical set of style rules that the creator of an HTML (or XML) file uses to control how that page is rendered in a browser or viewer, or how it is printed.

CSS enables the separation of the presentation content of documents from the information content of documents. A CSS file can be referenced by an entire website to provide continuity to titles, fonts, and colors.

Consider an example CSS rule for setting the H2 elements to the color red. Rules are made up of two parts: *selector* and *declaration*. The selector (H2) is the link between the HTML document and the style sheet, and all HTML element types are possible selectors.

The declaration has two parts: property (color) and value (red):

```
H2 { color: red }
```

For more information about CSS, see the following web address:

<http://www.w3.org/Style/CSS/>

Requirements for the development environment

The development environment has the following requirements:

- ▶ An editor for HTML pages, providing WYSIWYG (*what you see is what you get*), HTML code, and preview (browser) views to assist HTML page designers
- ▶ A CSS editor
- ▶ A view showing the overall structure of a site as it is being designed
- ▶ A built-in web server and browser to allow websites to be tested

IBM Rational Application Developer provides all of these features.

2.2.3 Dynamic web applications

By *web applications*, we mean applications that are accessed using HTTP, typically using a web browser as the client-side user interface to the application. The flow of control logic and business logic and the generation of the web pages for the web browser are all handled by software running on a server machine. Many technologies exist for developing this type of application, but we focus on the Java technologies that are relevant in this area.

Because the technologies are based on Java, most of the features that we describe in 2.1, “Desktop applications” on page 18 are also relevant here. (The GUI features are less significant.) In this section, we focus on the additional features that are required for developing web applications.

In the context of our example banking application, thus far we have provided workers in the bank’s call center with a desktop application to allow them to view and update account information and provided members of the web browsing public with information about the bank and its services. Next we move into the Internet banking web application, which is called *RedBank* in this document. We want to extend the system to allow bank customers to access their account information online, such as balances and statements, and to perform certain transactions, such as transferring money between accounts and paying bills.

The simplest way to provide Web-accessible applications using Java is to use Java servlets and JavaServer Pages (JSP). These technologies form part of the Java Enterprise Edition (Java EE), although they can also be implemented in systems that do not conform to the Java EE specification, such as Apache Jakarta Tomcat. For more information, see the following web address:

<http://jakarta.apache.org/tomcat/>

You can find information about these technologies (including specifications) at the following web addresses:

- ▶ Servlets:

<http://www.oracle.com/technetwork/java/index-jsp-135475.html>

- ▶ JSP:

<http://java.sun.com/products/jsp/>

In this book, we discuss Java EE 6, because this version is supported by Rational Application Developer V8 and IBM WebSphere Application Server V8. Java EE 6 supports the Servlet 3.0 and JSP 2.2 specifications. For full details about Java EE 6, see the following website:

<http://www.oracle.com/technetwork/java/javaee/overview/index.html>

Servlets

A *servlet* is a Java class that is managed by server software known as a *Web container* (sometimes referred to as a *servlets container* or *servlets engine*). The purpose of a servlet is to read information from an HTTP request, perform processing, and generate dynamic content to be returned to the client in an HTTP response.

The Servlet Application Programming Interface includes a class, `javax.servlet.http.HttpServlet`, which can be subclassed by a developer. The developer must override methods, such as the following methods, to handle various types of HTTP requests (in these cases, POST and GET requests; other methods are also supported):

```
public void doPost (HttpServletRequest request, HttpServletResponse
response)
public void doGet (HttpServletRequest request, HttpServletResponse
response)
```

When an HTTP request is received by the Web container, it consults a configuration file, known as a *deployment descriptor*, to establish which servlet class corresponds to the URL provided. If the class is already loaded in the Web container and an instance has been created and initialized, the Web container invokes a standard method on the servlet class:

```
public void service (HttpServletRequest request, HttpServletResponse
response)
```

The `service` method, which is inherited from `HttpServlet`, examines the HTTP request type and delegates processing to the `doPost` or `doGet` method, as appropriate. One of the responsibilities of the Web container is to package the HTTP request received from the client as an `HttpServletRequest` object and to create an `HttpServletResponse` object to represent the HTTP response that will ultimately be returned to the client.

Within the `doPost` or `doGet` method, the servlet developer can use the wide range of features available within Java, such as database access, messaging systems, connectors to other systems, or Enterprise JavaBeans (EJBs).

If the servlet has not already been loaded, instantiated, and initialized, the Web container is responsible for carrying out these tasks. The executing method performs the initialization step:

```
public void init ()
```

A corresponding method is called when the servlet is being unloaded from the Web container:

```
public void destroy ()
```

Within the code for the `doPost` and `doGet` methods, the following pattern is the usual processing pattern:

1. Read information from the request. This step often includes reading cookie information and getting parameters that correspond to fields in an HTML form.
2. Check that the user is in the appropriate state to perform the requested action.
3. Delegate the processing of the request to the appropriate type of business object.
4. Update the user's state information.
5. Dynamically generate the content to be returned to the client.

The last step might be carried out directly in the servlet code by writing HTML to a `PrintWriter` object obtained from the `HttpServletResponse` object:

```
PrintWriter out = response.getWriter();
out.println("<html><head><title>Page title</title></head>");
out.println("<body>The page content:");
// .....
```

Do not use this approach, because the embedding of HTML within the Java code means that HTML page design tools, such as those provided by Rational Application Developer, cannot be used. Also, development roles cannot be separated easily. Java developers are forced to maintain HTML code. The leading practice is to use a dedicated display technology, such as JSP, which we cover next.

The Servlet 3.0 specification introduces the following changes by embracing the Java EE 6 model:

- ▶ Annotations
Now available are annotations, such as `@WebServlet`, `@ServletFilter`, and `@WebServletContextListener`. Annotations reduce `web.xml` configuration to the point that it can be eliminated altogether.
- ▶ Web fragments
Introduces the idea of web fragments.
- ▶ Servlet context
Adds the ability to programmatically add Servlets, Filters, and Listeners through the `ServletContext`.
- ▶ Deployment descriptors
Follows EJB 3.0 in making deployment descriptors completely optional.

JavaServer Pages (JSP)

JSP provide a server-side scripting technology that enables Java code to be embedded within web pages, so JSP have the appearance of HTML or XML pages with embedded Java code. When the page is executed, the Java code can generate dynamic content to appear in the resulting web page. JSP are compiled at run time into servlets that execute to generate the resulting HTML or XML. Subsequent calls to the same JSP execute the compiled servlet.

JSP scripting elements are used to control the page compilation process, create and access objects, define methods, and manage the flow of control. For more information about these elements, see this website:

http://www.sentex.net/~pkomisar/J4/2_ProgrammingTech.html

The JSP scripting elements can be extended, using a technology known as *tag extensions* (or *custom tags*), to allow the developer to make up new tags and associate them with code that perform a wide range of tasks in Java. Tag extensions are grouped in *tag libraries*, which we discuss shortly.

Several of the standard JSP tags are only provided in an XML-compliant version, such as `<jsp:useBean ... />`. Others are available in both traditional form (for example, `<%= ... %>` for JSP expressions) or XML-compliant form (for example, `<jsp:expression ... />`). These XML-compliant versions have been introduced to allow JSP to be validated using XML validators.

JSP generate HTML output by default. The Multipurpose Internet Mail Extensions (MIME) type is `text/html`. It might be desirable to produce XML (`text/xml`) instead in certain situations. For example, a developer might want to produce XML output, which can then be converted to HTML for web browsers, Wireless Markup Language (WML) for wireless devices, or VoiceXML for systems with a voice interface. Servlets can also produce XML output in this way. The content type returned is set by using a method on the `HttpServletResponse` object.

The JSP 2.1 specification now defines annotations for dependency injection on JSP tag handlers and context listeners. Moreover, the Unified Expression Language (UEL) has the following key additions:

- ▶ A pluggable API resolves variable references into Java objects and resolves the properties applied to these Java objects.
- ▶ Support is added for deferred expressions, which can be evaluated by a tag handler when needed.
- ▶ Support for lvalue expression. A UEL expression used as an lvalue represents a reference to a data structure.

Tag libraries

Tag libraries are a standard way of packaging tag extensions for applications using JSP.

Tag extensions address the problem that arises when a developer wants to use non-trivial processing logic within a JSP. Java code can be embedded directly in the JSP using the standard tags described before. The mixture of HTML and Java makes it difficult to separate development responsibilities (the HTML/JSP designer has to maintain the Java code) and makes it hard to use appropriate tools for the tasks (a page design tool will not provide the same level of support for Java development as a Java development tool). This is essentially the reverse of the problem described when discussing servlets.

To address this problem, developers have documented the *View Helper* design pattern, as described in *Core J2EE Patterns: Best Practices and Design Strategies* by Crupi, et al. You can find the pattern catalog in this book at the following web address:

<http://java.sun.com/blueprints/corej2eepatterns>

Tag extensions are the standard way of implementing View Helpers for JSP. By using tag extensions, a Java developer can create a class that implements specific view-related logic. This class can be associated with a particular JSP tag using a Tag Library Descriptor (TLD). The TLD can be included in a web application, and the tag extensions defined within it can then be used in a JSP. The JSP designer can use these tags in exactly the same way as other standard JSP tags. The JSP specification includes classes that can be used as a basis for tag extensions and a simplified mechanism for defining tag extensions that does not require detailed knowledge of Java.

Many convenient tags are provided in the JSP Standard Tag Library (JSTL), which includes several tag libraries:

- ▶ Core tags: Flow control (such as loops and conditional statements) and various general-purpose actions
- ▶ XML tags: Allow XML processing within a JSP
- ▶ Formatting tags: Internationalized data formatting
- ▶ SQL tags: Database access for querying and updating
- ▶ Function tags: Various string handling functions

Tag libraries are also available from other sources, such as those from the Jakarta Taglibs project at the following address:

<http://jakarta.apache.org/taglibs/>

Additionally, new tag libraries can be developed to suit specific web application needs.

Expression Language

Expression Language (EL) was originally developed as part of the JSTL, but it is now a standard part of JSP (from JSP 2.0). EL provides a standard way of writing expressions within a JSP using implicit variables, objects available in the various scopes within a JSP and standard operators. In JSP 2.1, EL was updated to Unified Expression Language (UEL).

Filters

Filters are objects that can transform a request or modify a response. They can process the request before it reaches a servlet or process the response leaving a servlet before it is finally returned to the client. A filter can examine a request before a servlet is called and can modify the request and response headers and data by providing a customized version of the request or response object that wraps the real request or response. The deployment descriptor for a web application, or the `@ServletFilter` annotation, is used to configure specific filters for a particular servlet or JSP. Filters can also be linked together in chains.

Life-cycle listeners

Life-cycle events enable listener objects to be notified when servlet contexts and sessions are initialized and destroyed, as well as when attributes are added or removed from a context or session.

Any listener interested in observing the `ServletContext` life cycle can implement the `ServletContextListener` interface, which has two methods, `contextInitialized` (called when an application is first ready to serve requests) and `contextDestroyed` (called when an application is about to shut down).

A listener interested in observing the `ServletContext` attribute life cycle can implement the `ServletContextAttributesListener` interface, which has three methods, `attributeAdded` (called when an attribute is added to the `ServletContext`), `attributeRemoved` (called when an attribute is removed from the `ServletContext`), and `attributeReplaced` (called when an attribute is replaced by another attribute in the `ServletContext`).

Similar listener interfaces exist for `HttpSession` and `ServletRequest` objects:

- ▶ `javax.servlet.http.HttpSessionListener`: `HttpSession` life-cycle events
- ▶ `javax.servlet.http.HttpSessionAttributeListener`: Attributes events on an `HttpSession`
- ▶ `javax.servlet.http.HttpSessionActivationListener`: Activation or passivation of an `HttpSession`

- ▶ `javax.servlet.HttpSessionBindingListener`: Object binding on an `HttpSession`
- ▶ `javax.servlet.ServletRequestListener`: Processing of a `ServletRequest` has begun
- ▶ `javax.servlet.ServletRequestAttributeListener`: Attribute events on a `ServletRequest`

Requirements for the development environment

The development environment has the following requirements:

- ▶ Wizards for creating servlets, JSP, listeners, filters, and tag extensions
- ▶ An editor for JSP that enables the developer to use all the features of JSP in an intuitive way, focusing mainly on page design
- ▶ An editor for web deployment descriptors allowing these components to be configured
- ▶ Validators to ensure that all the technologies are being used correctly
- ▶ A test environment that allows dynamic web applications to be tested and debugged
- ▶ Support in a Java editor for Servlet 3.0 annotations

Rational Application Developer includes all these features. Figure 2-1 shows the interaction between the web components and a relational database, as well as the desktop application that is described in 2.1, “Desktop applications” on page 18.

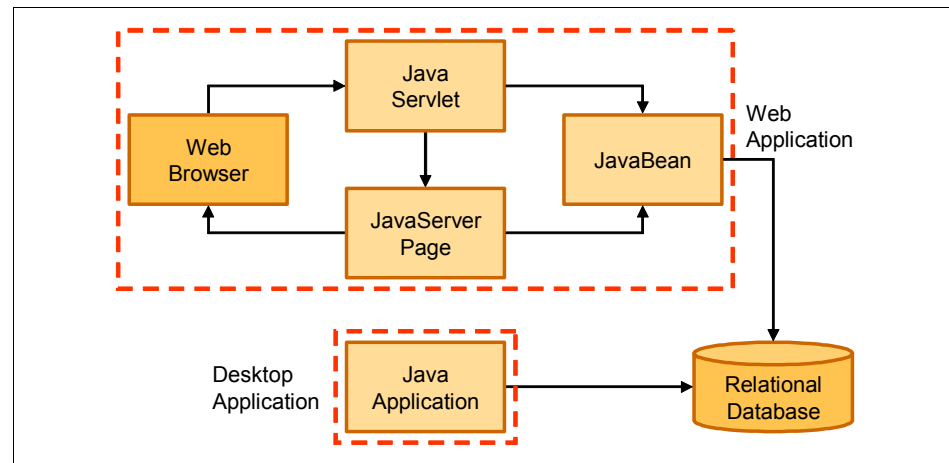


Figure 2-1 Web application interaction

Struts

Struts was introduced as a way of providing developers with a model view controller (MVC) framework for applications using the Java web technologies, servlets and JSP. Complete information about Struts is available at this website:

<http://struts.apache.org/>

Also, consult *Rational Application Developer V7.5 Programming Guide*, SG24-7672, for information about Struts.

2.2.4 JavaServer Faces and persistence using JPA

When we build a GUI for stand-alone Java applications, we can include event-handling code, so that when UI events take place, they can be used immediately to perform business logic processing or update the UI. Users are familiar with this type of behavior in desktop applications, but the nature of web applications has made this difficult to achieve using a browser-based interface. The user interface provided through HTML is limited, and the request-response style of HTTP does not naturally lead to flexible, event-driven user interfaces.

Many applications require access to data, and there is often a requirement to represent this data in an object-oriented way within applications. Many tools and frameworks exist for mapping between data and objects, but often these methods are proprietary or excessively complex systems.

In the RedBank web application, we want to make the user interface richer, while using the MVC architecture. In addition, developers want a simple, lightweight, and object-oriented database access system, which will remove the need for direct JDBC coding.

JavaServer Faces

JavaServer Faces (JSF) is a framework for developing Java web applications. The JSF framework aims to unify techniques for solving a number of common problems in web application design and development, including:

- ▶ User interface development: JSF allows direct binding of UI components to model data. It abstracts request processing into an event-driven model. Developers can use extensive libraries of prebuilt UI components that provide both basic and advanced web functionality.
- ▶ Navigation: JSF introduces a layer of separation between business logic and the resulting UI pages; stand-alone flexible rules drive the flow of pages.
- ▶ Session and object management: JSF manages designated model data objects by handling their initialization, persistence over the request cycle, and cleanup.

- ▶ Validation and error feedback: With JSF, reusable validators can be directly bound to UI components. The framework also provides a queue mechanism to simplify error and message feedback to the application user. These messages can be associated with specific UI components.
- ▶ Globalization: JSF provides tools for globalizing web applications, supporting number, currency, time, and date formatting, and externalizing UI strings.
- ▶ Extensibility: JSF is extended easily in a variety of ways to suit the requirements of your particular application. You can develop custom components, renderers, validators, and other JSF objects and register them with the JSF run time.

For more information about JSF framework, see the following web address:

<http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html>

JSF and Java Persistence API

Rational Application Developer provides comprehensive tooling to develop JSF applications that use Java Persistence API (JPA) for persistence in relational databases.

See Chapter 10, “Persistence using the Java Persistence API” on page 443, for information about JPA, and Chapter 19, “Developing web applications using JavaServer Faces” on page 1057, for an example of a JSF application with JPA.

Figure 2-2 shows how JSF and JPA can be used to create a flexible, powerful MVC-based web application with simple database access.

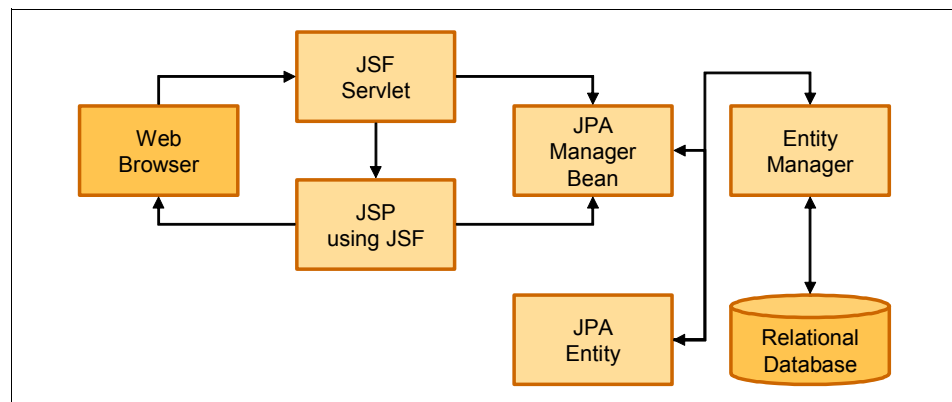


Figure 2-2 JSF and JPA

2.2.5 Web 2.0 development

Rational Application Developer comes with features to aid the development of responsive, rich Internet applications. Here, we briefly touch upon a few key technologies for developing Web 2.0 applications. We describe this information in more detail in Chapter 20, “Developing web applications using Web 2.0” on page 1097.

For more information about Web 2.0 development, see the IBM Redbooks publication *Building Dynamic Ajax Applications Using WebSphere Feature Pack for Web 2.0*, SG24-7635.

Ajax

Ajax is an acronym for Asynchronous JavaScript and XML. *Ajax* is a Web 2.0 development technique used for creating interactive web applications. The intent is to make web pages feel more responsive by exchanging small amounts of data with the server in the background without causing the page to reload in the browser. This is achieved by applying two techniques: sending asynchronous requests to the server for data or services, and using JavaScript to manipulate the DOM in order to change portions of the web page that need updates.

Figure 2-3 on page 42 illustrates the overall Ajax interaction between the client browser and the server.

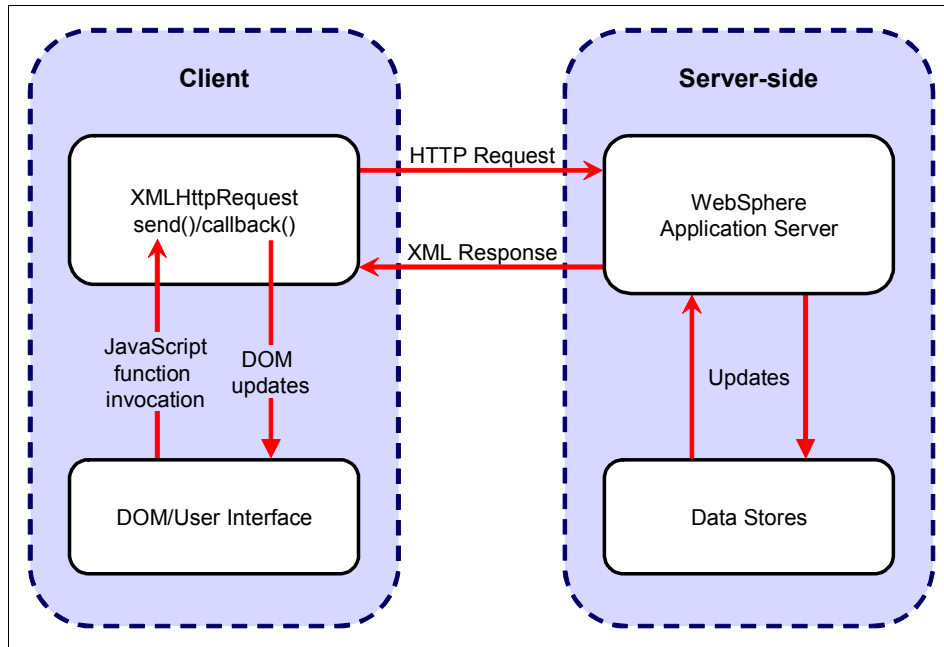


Figure 2-3 Ajax overview

The following example shows a typical user interaction that is implemented with Ajax:

1. JavaScript functional invocation

In response to a user action, such as clicking a button, certain JavaScript functions are invoked. The JavaScript code can do many things, which, in general, fall into two categories: modifying the DOM to change the presentation of the web page and invoking services via XMLHttpRequest.

2. Submitting XMLHttpRequest

Many user interactions require the involvement of the server. An XMLHttpRequest is a request sent from the client to the server. The request is asynchronous; that is, the client does not block waiting for the server response. For instance, the user might be logging in, which can be handled with Ajax. The JavaScript code collects the user ID and password from the login form using the DOM API and then sends an XMLHttpRequest to the particular URL for authentication service.

3. Server-side processing

From the server's point of view, XMLHttpRequests are identical to HTTP requests that are submitted by the browser directly. In this step, the server-side code processes the requests normally and produces the

appropriate HTTP response. However, special considerations must be given to the format (MIME type) of the responses so that the services are consumable by Ajax clients. JavaScript Object Notation (JSON) (application/json) is the most widely used data format for such purposes. Other choices include XML (text/xml), Atom (application/atom+xml), and text (text/plain).

4. XMLHttpRequest.callback

When the server response is received by the client, the Ajax callback function is invoked. The callback technique is necessary because of the asynchronous nature of XMLHttpRequest. As part of processing the response, the JavaScript function can take the response values and update the page (by changing or updating the DOM).

For more information about Ajax, see the following web address:

<http://www.ibm.com/developerworks/ajax>

Representational State Transfer (REST)

REST services are widely regarded as the technology of choice for building the services layer in Web 2.0 applications. *Representational State Transfer (REST)* is a collection of software architecture principles to build services according to a resource-centered design paradigm.

REST is based on HTTP. This reliance on existing standards makes REST easier to learn and simpler to use than most other Web-based messaging standards, because little additional overhead is required to enable effective information exchange.

The REST principle uses uniform resource identifiers (URIs) to locate and access a given representation of a resource. The resource representation, known as *representational state*, can be created, retrieved, modified, or deleted.

A REST-based conversation operates within stateless conversations, thereby making it a prime facilitator for subscription-based technologies, such as RSS, RDF, OWL, and Atom, in which content is delivered to pre-subscribed clients.

For more information about REST, see the following web address:

<http://www.ibm.com/developerworks/webservices/library/ws-restful/index.html>

JavaScript Object Notation

JavaScript Object Notation (JSON) is a lightweight data-interchange format. It is easy to read and write and is supported natively by JavaScript. It is based on a

subset of the JavaScript Programming Language and is built on two structures: a collection of name/value pairs and an ordered list of values.

Because of native support by JavaScript, and compact syntax, JSON is the primary choice for data format in REST services.

For further information about JSON, see the following web address:

<http://www.json.org>

Dojo

The Dojo Toolkit is a powerful open-source JavaScript library that can be used to create rich user interfaces running within a browser. The library requires no browser-side runtime plug-in and runs natively on all major browsers. This is a boon for JavaScript developers, because it helps abstract away the eccentricity of separate browser implementations.

We describe Dojo Toolkit in more detail in Chapter 20, “Developing web applications using Web 2.0” on page 1097.

For more information about Dojo Toolkit, see the following web address:

<http://www.dojotoolkit.org/>

2.2.6 Portal applications

Portal applications run on a portal server and consist of portal pages that are composed of portlets. *Portlets* can share and exchange resources and information to provide a seamless web interface.

Portal applications have several important features:

- ▶ They can collect content from a variety of sources and present the content to the user in a single unified format.
- ▶ The presentation can be personalized so that each user sees a view based on that user’s own characteristics or role.
- ▶ The presentation can be customized by the user to fulfill the user’s specific needs.
- ▶ They can provide collaboration tools, which allow teams to work in a virtual office.
- ▶ They can provide content to a range of devices, formatting and selecting the content appropriately according to the capabilities of the device.

In the context of our sample scenario, we can use a portal application to enhance the user experience. The RedBank web application can be integrated with the

static web content providing information about branches and bank services. If the customer has credit cards, mortgages, personal loans, savings accounts, shares, insurance, or other products provided by the bank or business partners, they can also be seamlessly integrated into the same user interface, providing the customer with a convenient single point of entry to all these services.

The content of these applications can be provided from a variety of sources, with the portal server application collecting the content and presenting it to the user. The user can customize the interface to display only the required components, and the content can be varied to allow the customer to connect using a web browser, personal digital assistant (PDA), or mobile phone.

Within the bank, the portal can also be used to provide convenient intranet facilities for employees. Sales staff can use a portal to receive information about the latest products and special offers, information from human resources, leads from colleagues, and so on.

IBM WebSphere Portal

WebSphere Portal runs on top of WebSphere Application Server, using the Java EE standard services and server management capabilities as the basis for portal services. WebSphere Portal provides its own deployment, configuration, administration, and communication features.

Java Portlet Specifications

Based on its history, the following portlet specifications are in use:

- ▶ **IBM Portlet API**

The IBM Portlet API is being deprecated for WebSphere Portal V6.0, but it is still supported. Because no new functionality will be added, use the Standard Portlet API. See the following web address:

<http://publib.boulder.ibm.com/infocenter/wpdoc/v6r0>

- ▶ *JSR 168 Portlet Specification*

The *JSR 168 Portlet Specification* defines a set of APIs for portal computing addressing the areas of aggregation, personalization, presentation, and security. See the following web address:

<http://www.jcp.org/en/jsr/detail?id=168>

- ▶ *JSR 286 Portlet Specification 2.0*

Since its release in 2003, JSR 168 has gone through many real-life tests in portal development and deployment. Gaps identified by the community take time to evolve and become available to the public as a standard. Meanwhile, many portal vendors have been filling those gaps with their own custom

solutions, which unfortunately cause portlets not to be portable. That is the major reason for a new standard. See the following web address:

<http://www.jcp.org/en/jsr/detail?id=286>

Requirements for the development environment

The development environment must provide wizards for creating portal applications and the associated components and configuration files, as well as editors for all these files. A test environment is required to allow portal applications to be executed and debugged.

Rational Application Developer includes the required tooling and is compatible with WebSphere Portal V6.1 and V7.0 unit test environments.

Figure 2-4 shows how portal applications fit in with the other technologies that are mentioned in this chapter.

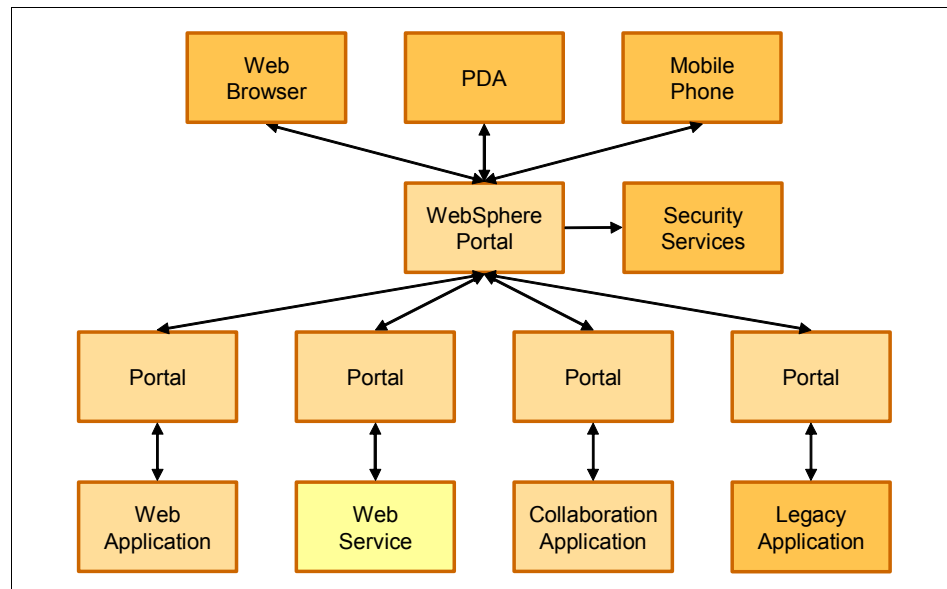


Figure 2-4 Portal applications

2.3 Enterprise JavaBeans and Java Persistence API

Now that the RedBank web application is up and running, more issues arise. Several of these issues relate to the services provided to customers and bank workers, and other issues relate to the design, configuration, and functionality of the systems that perform the back-end processing for the application.

First, we want to provide the same business logic in a new application that will be used by administration staff working in the bank's offices. We want the ability to reuse the code that has already been generated for the RedBank web application without introducing the overhead of having to maintain several copies of the same code. Integration of these business objects into a new application must be made as simple as possible.

Next we want to reduce development time by using an object-relational mapping system that will provide an in-memory, object-oriented view of data from the relational database view automatically, and provide convenient mapping tools to set up the relationships between objects and data. This system must be capable of dealing with distributed transactions, because the data might be located on several databases around the bank's network.

Because we are planning to make business logic available to multiple applications simultaneously, we want a system that will manage issues, such as multithreading, resource allocation, and security so that developers can focus on writing business logic code without having to worry about these infrastructure matters.

Finally, the bank has existing systems, not written in Java, which we want to update to use the new functionality provided by these business objects. We want to use a technology that can allow this type of interoperability between separate platforms and languages.

We can get all this functionality by using EJB and the JPA to provide our back-end business logic and access to data. Later, we show how EJB can help you to integrate messaging systems and web services clients with the application logic.

2.3.1 EJB 3.1 specification: What is new

EJB 3.x is a major enhancement to the EJB specification, introducing a new plain old Java object (POJO)-based programming model that greatly simplifies development of Java EE applications. Java EE 5 provided EJB 3.0 and Java EE 6 EJB 3.1. EJB 3.x offers the following major features:

- ▶ EJB components are now POJOs that expose regular business interfaces (plain old Java interfaces (POJI)), and there is no requirement for home interfaces.
- ▶ The deployment descriptor information is replaced by annotations.
- ▶ A completely new persistence model, Java Persistence API (JPA), is provided, which complements EJB 2.x entity beans.

- ▶ An Interceptor facility invokes user methods at the invocation of business services or at life-cycle events.
- ▶ EJB 3.x adopts an annotation-based dependency injection pattern to obtain Java EE resources (JDBC data sources, Java Message Service (JMS) factories and queues, and EJB references).
- ▶ Default values are provided whenever possible (“configuration by exception” approach).
- ▶ The use of checked exceptions is reduced.
- ▶ All life-cycle methods are optional now.

In moving from EJB 3.0 to EJB 3.1, the improvements include a No-Interface View for EJB components, Singleton EJBs, Async session bean invocation, simplified packaging of EJB components, and a Java EE profile called *EJB Lite*. For more information about EJB 3.x and JPA, see these resources:

- ▶ Chapter 10, “Persistence using the Java Persistence API” on page 443
- ▶ Chapter 12, “Developing Enterprise JavaBeans (EJB) applications” on page 577
- ▶ *WebSphere Application Server Version 6.1 Feature Pack for EJB 3.0*, SG24-7611.
- ▶ Enterprise JavaBeans Technology
<http://www.oracle.com/technetwork/java/index-jsp-140203.html>

2.3.2 Types of EJBs

In this section, we describe two types of EJB 3.x: session beans (stateless and stateful) and message-driven beans (MDBs).

Entity beans: Entity beans as specified in EJB specification 2.x have been replaced by Java Persistence API entities.

Session beans

Session beans are task-oriented objects, which are invoked by client code. They are non-persistent and do not survive an EJB container shutdown or crash.

Session beans often act as the external face of the business logic provided by an EJB. The session facade pattern, described in many pattern catalogs, including *Core J2EE Patterns: Best Practices and Design Strategies* by Crupi, et al., describes this idea. The client application that needs to access the business logic provided by an EJB sees only session beans. The low-level details of the persistence mechanism are hidden behind these session beans (the session

bean layer is known as the *session facade*). As a result, the session beans that make up this layer are often closely associated with a particular application and might not be reusable between applications.

It is also possible to design reusable session beans, which might represent a common service that can be used by many applications.

Stateless session beans

Stateless session beans are the preferred type of session bean, because they generally scale better than stateful session beans. Stateless beans are pooled by the EJB container to handle multiple requests from multiple clients. To permit this pooling, stateless beans cannot contain any state information that is specific to a particular client. Because of this restriction, all instances of a stateless bean are equivalent, allowing the EJB container to assign an instance to any client. Stateless session beans are marked with the `@Stateless` annotation and the business interface is annotated with the `@Local` (default) or `@Remote` annotation.

Stateful session beans

Stateful session beans are useful when an EJB client needs to call several methods and store state information in the session bean between calls. Each stateful bean instance must be associated with exactly one client, so the container is unable to pool stateful bean instances. Stateful session beans are annotated with the `@Stateful` annotation.

Singleton session beans

Singleton session beans follow the Singleton design pattern allowing for only one instance per JVM. They are similar to stateless session beans, because they hold no conversation state between clients and can be used by any client. They differ from both stateless and stateful session beans, because the Singleton instance is shared between multiple clients and therefore must support concurrent access.

Message-driven beans (MDBs)

MDBs receive and process messages. They can be accessed only by sending a message to the messaging server to which the bean is configured to listen. MDBs are stateless and can be used to allow asynchronous communication between client EJB logic by using a type of messaging system. MDBs are normally configured to listen to JMS resources, although since EJB 2.1, other messaging systems are also supported.

MDBs are normally used as adapters to allow logic provided by session beans to be invoked by using a messaging system. As such, they can be thought of as an asynchronous extension of the session facade concept described before, known

as the *message facade pattern*. Message-driven beans can only be invoked in this way and therefore have no specific client interface. Message-driven EJBs are annotated with the `@MessageDriven` annotation.

2.3.3 Java Persistence API

The JPA provides an object-relational mapping facility for managing relational data in Java applications. Entity beans as specified in the EJB 2.x specification have been replaced by JPA entity classes. These classes are annotated with the `@Entity` annotation. Entities can either use persistent fields (mapping annotation is applied to an entity's instance variables) or persistent properties (mapping annotation is applied to getter methods for JavaBeans-style properties). All fields of an entity not annotated with the `@Transient` annotation or not marked with the `transient` Java keyword will be persisted to the data store. The object-relational mapping annotation must be applied to the instance variables. The primary key field is annotated with the `@Id` annotation.

Entity relationships have the following types of multiplicities:

- ▶ One-to-one (`@OneToOne`): Each entity instance is related to a single instance of another entity.
- ▶ One-to-many (`@OneToMany`): An entity instance can be related to multiple instances of the other entities.
- ▶ Many-to-one (`@ManyToOne`): Multiple instances of entity can be related to a single instance of another entity.
- ▶ Many-to-many (`@ManyToMany`): The entity instances can be related to multiple instances of each other.

Entities are managed by the *Entity Manager*. The Entity Manager is an instance of `javax.persistence.EntityManager` and is associated with a persistence context. A *persistence context* defines the scope under which particular entity instances are created, persisted, and removed. The `EntityManager` API provides functionality to allow a developer to create and remove persistent entity instances, find an entity by its primary key, and allow queries to be run on entities.

The following Entity Managers are available:

- ▶ Container-managed Entity Manager

The persistence context is automatically propagated by the container to all application components that use the `EntityManager` instance within a single Java Transaction API (JTA) transaction. To obtain an `EntityManager` instance, inject the Entity Manager into the application component:

```
@PersistenceContext
```

```
EntityManager em;
```

► **Application-managed Entity Manager**

This Entity Manager is used when applications need to access a persistence context that is not propagated with the JTA transaction across EntityManager instances in a particular persistence unit. In this case, each EntityManager creates a new, isolated persistence context.

To obtain an EntityManager instance, inject an EntityManagerFactory into the application component by means of the @PersistenceUnit annotation:

```
@PersistenceUnit  
EntityManagerFactory emf;
```

Then obtain an EntityManager from the EntityManagerFactory instance:

```
EntityManager em = emf.createEntityManager();
```

2.3.4 Other EJB and JPA features

In this section, we describe other EJB features not discussed previously.

Java Persistence Query Language

The Java Persistence API specifies a query language that allows a developer to define queries over entities and their persistent states. The Java Persistence Query Language (JPQL) provides a way to specify the semantics of queries in a portable way, independent of the particular database used in the enterprise environment.

JPQL is an extension of the EJB query language (EJB QL) and combines the syntax and simple query semantics of SQL with the expressiveness of an object-oriented expression language.

For more information about JPQL, see the Java EE 6 Tutorial at the following address:

<http://download.oracle.com/javaee/6/tutorial/doc/bnbtg.html>

EJB timer service

The EJB timer service was introduced with EJB 2.1. A bean provider can choose to implement the `javax.ejb.TimedObject` interface, which requires the implementation of a single method, `ejbTimeout`. The bean creates a `Timer` object by using the `TimerService` object obtained from the bean's `EJBContext`. After the `Timer` object has been created and configured, the bean will receive messages from the container according to the specified schedule; the container calls the `ejbTimeout` method at the appropriate interval.

In EJB 3.x, instead of implementing the `javax.ejb.TimerObject` interface, the method that gets called by the timer service can be annotated with the `@Timeout` annotation only.

Requirements for the development environment

The development environment must provide wizards for creating the various types of EJBs, tools for mapping JPA entities to relational database systems, and test facilities.

Rational Application Developer provides all these features.

Figure 2-5 shows how EJB components work with other technologies that we have already discussed.

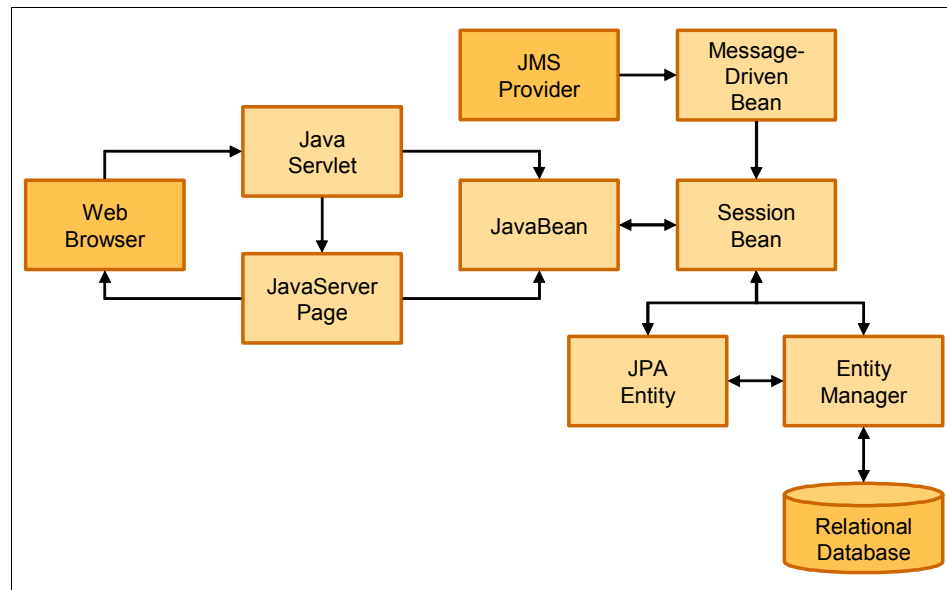


Figure 2-5 EJB as part of an enterprise application

2.4 Web services

The bank's computer system is now quite sophisticated and includes the following items:

- ▶ A database to store the bank's data
- ▶ A Java application that allows bank employees to access the database

- ▶ A static website that provides information about the bank's branches, products, and services
- ▶ A web application that provides Internet banking facilities for customers, with various technology options available
- ▶ An EJB back end that provides the following access:
 - Centralized access to the bank's business logic through session beans
 - Transactional, object-oriented access to data in the bank's database through JPA entities
- ▶ A Java EE application client that can use the business logic in session beans

2.4.1 Interoperability considerations

So far, everything is quite self-contained. Although clients can connect from the web to use the Internet banking facilities, the business logic is all contained within the bank's systems, and even the Java application and Java EE application client are expected to be within the bank's private network.

The next step in developing our service is to enable mortgage agents, who search many mortgage providers to find the best deal for their customers, to access business logic provided by the bank to get the latest mortgage rates and repayment information. While we want to enable this capability, we do not want to compromise security. We must consider the fact that the mortgage brokers might not be using systems based on Java at all.

The League of Agents for Mortgage Enquiries has published a description of services that its members might use to get this type of information. We want to conform to this description to allow the maximum number of agents to use our bank's systems.

We might also want the ability to share information with other banks. For example, we might want to exchange information about funds transfers between banks. Standard mechanisms to perform these tasks have been provided by the relevant government body.

These issues are all related to interoperability, which is the domain addressed by web services. By using web services, we can enable all these separate types of communication between systems. We can use our existing business logic where applicable and develop new web services easily where necessary.

2.4.2 Web services in Java EE 6

Web services provide a standard means of communication among separate software applications. Because of the simple foundation technologies used in enabling web services, it is easy to call a web service, regardless of the platform, operating system, language, or technology used to implement it.

A *service provider* creates a web service and publishes its interface and access information to a *service registry* (or *service broker*). A *service requestor* locates entries in the service registry and then binds to the service provider to invoke its web service.

Web services use the following standards:

SOAP A protocol for exchanging XML-based messages over computer networks, normally using HTTP or HTTPS

Web Services Description Language (WSDL)
Describes web service interfaces and access information

Universal Description, Discovery, and Integration (UDDI)
A standard interface for service registries, which allows an application to find organizations and services

The specifications for these technologies are available at the following web addresses:

- ▶ w3/SOAP
<http://www.w3.org/TR/soap/>
- ▶ w3/wsdl
<http://www.w3.org/TR/wsdl>
- ▶ uddi.xml.org
<http://uddi.xml.org>

Figure 2-6 shows how these technologies fit together.

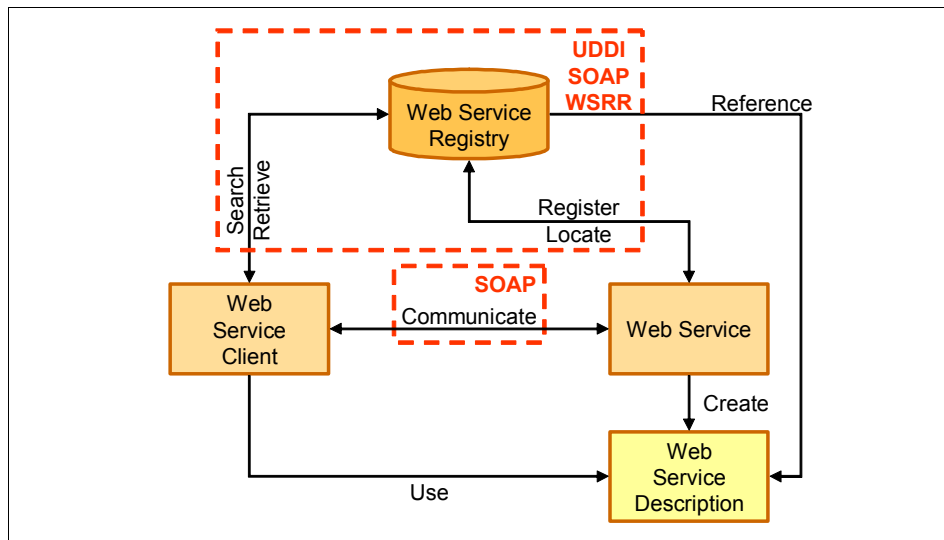


Figure 2-6 Web services foundation technologies

WebSphere Service Registry and Repository: WebSphere Service Registry and Repository is the recommended implementation of a Web Services Registry in place of Universal Description, Discovery, and Integration (UDDI).

Since the release of Java EE 1.4, web services are included in the specification. Therefore, all Java EE application servers that support Java EE 1.4 or later have the same standard level of support for web services, and certain Java EE application servers that support Java EE 1.4 or later also provide enhancements, as well.

Java EE 6 provides full support for both clients of web services and web services providers. The following Java technologies work together to provide support for web services:

► Java API for XML Web Services (JAX-WS) 2.2

JAX-WS is the primary API for web services and is a follow-on to the Java API for XML-based Remote Procedure Call (JAX-RPC). JAX-WS offers extensive web services functionality with the help of Java annotations, with support for multiple bindings/protocols, and RESTful web services. JAX-WS and JAX-RPC are fully interoperable when using SOAP 1.1 over the HTTP

protocol as constrained by the WS-I basic profile specification. For more information, see the following web address:

<http://www.jcp.org/en/jsr/detail?id=224>

- ▶ Java Architecture for XML Binding (JAXB) 2.0
JAXB provides a convenient way to bind an XML schema to a representation in Java code. This support makes it easy to incorporate XML data and processing functions in Java applications without having to know much about XML itself. For more information, see the following web address:
<http://www.jcp.org/en/jsr/detail?id=222>
- ▶ SOAP with Attachments API for Java (SAAJ) 1.3
SAAJ describes the standard way to send XML documents as SOAP documents over the Internet from the Java platform. It supports SOAP 1.2. For more information, see the following web address:
<http://www.jcp.org/en/jsr/detail?id=67>
- ▶ Streaming API for XML (StAX) 1.0
StAX is a streaming Java-based, event-driven, pull-parsing API for reading and writing XML documents. StAX enables you to create bidirectional XML parsers that are fast, relatively easy to program, and have a light memory footprint. For more information, see the following web address:
<http://www.jcp.org/en/jsr/detail?id=173>
- ▶ Web Services Metadata for the Java Platform
The Web Services Metadata specification defines Java annotations that make it easier to develop web services. This specification and JAX-WS together provide a comprehensive set of annotations for Java web service and web service client implementations. For more information, see the following web address:
<http://www.jcp.org/en/jsr/detail?id=186>
- ▶ Java API for XML Registries (JAXR) 1.0
JAXR provides client access to XML registry and repository servers. For more information, see the following web address:
<http://www.jcp.org/en/jsr/detail?id=93>
- ▶ Java API for XML Web Services Addressing (JAX-WSA) 1.0
JAX-WSA is an API and framework for supporting transport-neutral addressing of web services. For more information, see the following web address:
<http://www.jcp.org/en/jsr/detail?id=261>

- ▶ SOAP Message Transmission Optimization Mechanism (MTOM)

MTOM enables SOAP bindings to optimize the transmission or wire format of a SOAP message by selectively encoding portions of the message, while still presenting an XML infoset to the SOAP application.

For more information, see the following web address:

<http://www.w3.org/TR/soap12-mtom/>
- ▶ Web Services Reliable Messaging (WS-RM)

WS-RM is a protocol that allows messages to be delivered reliably between distributed applications in the presence of software component, system, or network failures. For more information, see the following web address:

<http://www.ibm.com/developerworks/library/specification/ws-rm/>
- ▶ Web Services for Java EE

Web Services for Java EE defines the programming and deployment model for web services in Java EE. It includes details of the client and server programming models, handlers (a similar concept to servlet filters), deployment descriptors, container requirements, and security. For more information, see the following web addresses:

 - JSR 109
<http://www.jcp.org/en/jsr/detail?id=109>
 - JSR 921
<http://www.jcp.org/en/jsr/detail?id=921>

Because interoperability is a key goal in web services, an open, industry organization, which is known as the *Web Services Interoperability Organization* (WS-I), has been created to allow interested parties to work together to maximize the interoperability between web services implementations.

WS-I: For more information about WS-I, see the following website:

<http://ws-i.org/>

WS-I has produced the following set of interoperability profiles:

- ▶ WS-I Basic Profile 1.1
<http://ws-i.org/Profiles/BasicProfile-1.1.html>
- ▶ WS-I Simple SOAP Binding Profile 1.0
<http://ws-i.org/Profiles/SimpleSoapBindingProfile-1.0.html>
- ▶ WS-I Basic Security Profile 1.0

<http://ws-i.org/Profiles/BasicSecurityProfile-1.0.html>

- ▶ WS-I Attachments Profile 1.0

<http://ws-i.org/Profiles/AttachmentsProfile-1.0.html>

Requirements for the development environment

The development environment must provide facilities for creating web services from existing Java resources, including JAX-WS and JAX-RPC service endpoint implementations for stateless session EJB components and web components. As part of the creation process, the tools must also produce the required deployment descriptors and WSDL files. Editors must be provided for WSDL files and deployment descriptors. The tooling must support and encourage the development of WS-I interoperable web services.

The tooling must also allow skeleton web services to be created from WSDL files and must provide assistance in developing web services clients, based on information obtained from WSDL files.

A range of test facilities must be provided, so that a developer can test web services and clients.

Rational Application Developer provides all this functionality.

Figure 2-7 on page 59 shows how the web services technologies fit into the overall programming model.

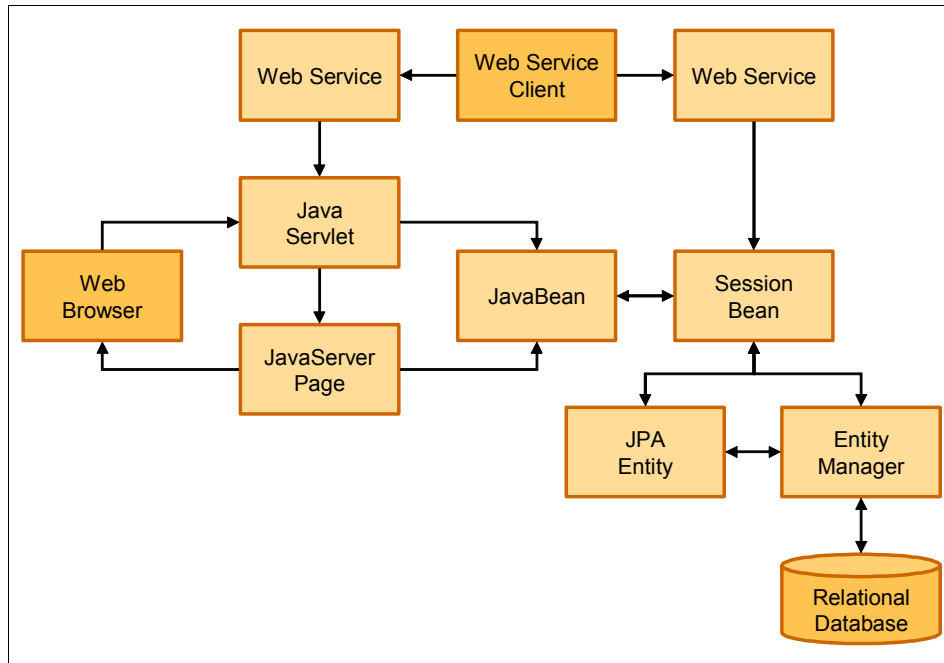


Figure 2-7 Web services

2.5 Messaging systems

The bank has several automatic teller machines (ATMs) with a user interface and communication support. The ATMs are designed to communicate with the bank's central computer systems using a secure, reliable, highly scalable messaging system. We want to integrate the ATMs with our system so that transactions carried out at an ATM can be processed using the business logic we have already implemented. Ideally, we also want the option of using EJB components to handle the messaging for us.

Many messaging systems exist that provide these features. IBM's solution in this area is IBM WebSphere MQ, which is available on many platforms and provides application programming interfaces in several languages. From the point of view of our sample scenario, WebSphere MQ provides Java interfaces that we can use in our applications. In particular, we consider the interface that conforms to the JMS specification. The idea of JMS is similar to that of JDBC. A standard interface provides a layer of abstraction for developers who want to use messaging systems without being tied to a specific implementation.

2.5.1 Java Message Service

JMS defines the following messaging, among other things:

- ▶ A messaging model
The structure of a JMS message and an API for accessing the information contained within a message. The JMS interface is `javax.jms.Message`, implemented by several concrete classes, such as `javax.jms.TextMessage`.
- ▶ Point-to-point (PTP) messaging
A queue-based messaging architecture, similar to a mailbox system. The JMS interface is `javax.jms.Queue`.
- ▶ Publish/Subscribe (Pub/Sub) messaging
A topic-based messaging architecture, similar to a mailing list. Clients subscribe to a topic and then receive any messages that are sent to the topic. The JMS interface is `javax.jms.Topic`.

For more information about JMS, see the following web address:

<http://www.oracle.com/technetwork/java/index-jsp-142945.html>

2.5.2 Message-driven beans (MDBs)

MDBs were introduced into EJB 2.0, extended in EJB 2.1, and simplified in EJB 3.x. MDBs consume incoming messages sent from a destination or endpoint system to which the MDB is configured to listen. From the point of view of the message-producing client, it is impossible to tell how the message is being processed, for example, whether by a stand-alone Java application, an MDB, or a message-consuming application that is implemented in another language. This is one of the advantages of using messaging systems. The message-producing client is well decoupled from the message consumer (similar to web services in this respect).

From a development point of view, MDBs are the simplest type of EJB, because they do not have clients in the same sense as session and entity beans. The only way to invoke an MDB is to send a message to the endpoint or destination to which the MDB is listening. In EJB 2.0, MDBs only dealt with JMS messages, but in EJB 2.1, this capability is extended to other messaging systems. The development of an MDB differs depending on the messaging system being targeted, but most MDBs are still designed to consume messages through JMS, which requires the bean class to implement the `javax.jms.MessageListener` interface, as well as `javax.ejb.MessageDrivenBean`.

A common pattern in this area is the message facade pattern, as described in *EJB Design Patterns: Advanced Patterns, Processes and Idioms* by Marinescu. You can download this book from the following web page:

<http://www.theserverside.com/news/1369776/Free-Book-EJB-Design-Patterns>

According to this pattern, the MDB acts as an adapter, receiving and parsing the message, and then invoking the business logic to process the message using the session bean layer.

2.5.3 Requirements for the development environment

The development environment must provide a wizard to create MDBs and facilities for configuring the MDBs in a suitable test environment. The test environment must also include a JMS-compliant server.

Testing MDBs is challenging, because they can only be invoked by sending a message to the messaging resource to which the bean is configured to listen. However, WebSphere Application Server V8.0 Beta, which is provided as a test environment within Rational Application Developer, includes an embedded JMS that can be used for testing purposes. A JMS client must be developed to create the test messages.

Figure 2-8 shows how messaging systems and MDBs fit into the application architecture.

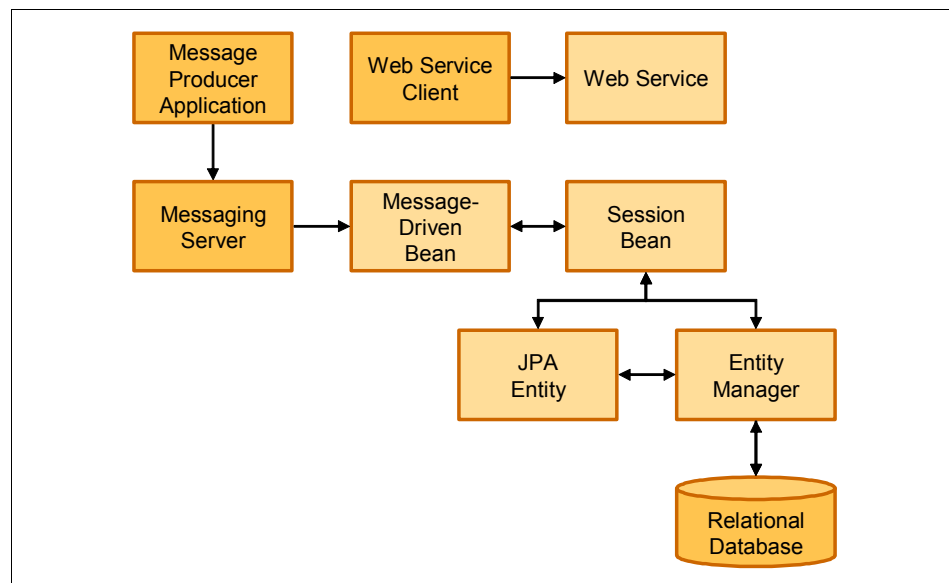


Figure 2-8 Messaging systems

2.6 OSGi applications

OSGi is a module system that is compatible with Java-based systems and implements a dynamic component model. Enterprise systems can use OSGi to improve the maintainability of runtime infrastructures. Applications, in the form of bundles, can be remotely installed, started, stopped, updated, and uninstalled without requiring a reboot.

2.6.1 OSGi features

OSGi tools include the following major features.

Support for OSGi Blueprint components

The OSGi Version 4.2 Blueprint component model defines a standard dependency injection mechanism for Java components. The implementation is derived from the Spring Framework and extended for OSGi to declaratively register component interfaces as services in the OSGi service registry.

Model for assembling bundles

The OSGi tools include a model for assembling an application into a deployable unit. The unit can consist of multiple bundles and includes the metadata that describes the version and external location of the constituent bundles of the application.

Runtime components

The OSGi tools support the development of OSGi applications that run in an OSGi framework, exploiting enterprise Java technologies common in web applications and integration scenarios including web application bundles, remote services integration, and JPA.

Extensions

The OSGi tools include extensions that go beyond the OSGi Enterprise Expert Group specifications to provide a more complete integration of OSGi modularity with Java enterprise technologies. In particular, it delivers support that includes but is not restricted to the following features:

- ▶ Isolated enterprise applications composed of multiple, versioned bundles with dynamic life cycles
- ▶ Declarative transactions and security for Blueprint components
- ▶ Container-managed JPA for Blueprint components
- ▶ Message-driven Blueprint components

- ▶ Configuration of resource references in module Blueprint Services
- ▶ Annotation-based Blueprint configuration
- ▶ Federation of lookup mechanisms between local JNDI and the OSGi service registry
- ▶ Fully declarative application metadata to enable reflection of an SCA component type definition

2.6.2 Benefits of OSGi

OSGi modularity provides standard mechanisms to address the issues faced by Java EE applications. The OSGi framework provides the following benefits:

- ▶ Applications are portable, easier to re-engineer, and adaptable to changing requirements.
- ▶ The framework provides the declarative assembly and simplified unit test of the Spring Framework, but in a standardized form that is provided as part of the application server run time rather than being a third-party library deployed as part of the application.
- ▶ The framework integrates with the Java EE programming model, giving you the option of deploying a web application as a set of versioned OSGi bundles with a dynamic life cycle.
- ▶ It supports the administration of application bundle dependencies and versions, simplifying and standardizing third-party library integration.
- ▶ The framework provides isolation for enterprise applications that are composed of multiple, versioned bundles with dynamic life cycles.
- ▶ It has a built-in bundle repository that can host common and versioned bundles shared between multiple applications, so that each application does not deploy its own copy of each common library.
- ▶ OSGi applications can access external bundle repositories.
- ▶ The framework reinforces service-oriented design at the module level.
- ▶ OSGi applications can be composed of coarse-grained SCA assemblies.

2.7 Other applications

This section addresses the following applications:

- ▶ Java EE application clients
- ▶ Enterprise information system applications
- ▶ Service Component Architecture applications

- ▶ Session Initiation Protocol applications
- ▶ Communications Enabled Applications (CEA)

2.7.1 Java EE application clients

Java EE application clients are one of the four types of components defined in the Java EE specification. The others are EJB, web components (servlets and JSP), and Java applets. They are stand-alone Java applications that use resources provided by a Java EE application server, such as EJB, data sources, and JMS resources.

In the context of our banking sample application, we want to provide an application for bank workers who are responsible for creating accounts and reporting on the accounts held at the bank. Because a lot of the business logic for accessing the bank's database has been developed using EJB, we want to avoid duplicating this logic in our new application. Using a Java EE application client for this purpose allows us to develop a convenient interface, possibly a GUI, while still allowing access to this EJB-based business logic. Even if we do not want to use EJB components for business logic, with a Java EE application client, we can access the data sources or JMS resources provided by the application server and integrate with the security architecture of the server.

Required Java EE Client Container APIs

Java EE: Information about Java EE is available from the following web address:

<http://www.oracle.com/technetwork/java/javaee/overview/index.html>

The Java EE 6 specification requires the following APIs to be provided to Java EE application clients. Java Platform, Standard Edition 6.0 requires these APIs:

- ▶ Java Interface Definition Language (IDL)
- ▶ Java Database Connectivity (JDBC) 4.0
- ▶ Java Remote Method Invocation over Internet Inter-Orb Protocol (RMI-IIOP)
- ▶ Java Naming and Directory Interface (JNDI)
- ▶ Java API for XML Processing (JAXP) 1.4
- ▶ Java Authentication and Authorization Service (JAAS)
- ▶ Java Management Extension (JMX)

The following additional packages are available:

- ▶ Enterprise JavaBeans 3.1 Client API
- ▶ Java Message Service 1.1
- ▶ JavaMail 1.4

- ▶ Java Activation Framework (JAF) 1.1
- ▶ Web Services 1.2
- ▶ Java API for XML-Based RPC (JAX-RPC) 1.1
- ▶ Java API for XML Web Services (JAX-WS) 2.2
- ▶ Java Architecture for XML Binding (JAXB) 2.2
- ▶ SOAP with Attachments API for Java (SAAJ) 1.3
- ▶ Java API for XML Registries (JAXR) 1.0
- ▶ Java EE Management 1.1
- ▶ Java EE Deployment 1.2
- ▶ Web Services Metadata 2.0
- ▶ Common Annotations 1.0
- ▶ Streaming API for XML (StAX) 1.0
- ▶ Java Persistence API 2.0

Security

The Java EE specification requires that the same authentication mechanisms be made available for Java EE application clients as for other types of Java EE components. The authentication features are provided by the Java EE application client container, as they are in other containers within Java EE. With a Java EE platform, the Java EE application client container can communicate with an application server to use its authentication services. WebSphere Application Server allows this function.

Naming

The Java EE specification requires that Java EE application clients have exactly the same naming features available as are provided for web components and EJB components. Java EE application clients must be able to use the Java Naming and Directory Interface (JNDI) to look up objects using object references and real JNDI names. The reference concept allows a deployer to configure references that can be used as JNDI names in lookup code. The references are bound to real JNDI names at deployment time, so that if the real JNDI name is subsequently changed, the code does not have to be modified or recompiled. Only the binding needs to be updated.

References can be defined for the following items:

- ▶ EJB
 - For Java EE application clients, only remote references, because the client cannot use local interfaces
- ▶ Resource manager connection factories
- ▶ Resource environment values
- ▶ Message destinations
- ▶ User transactions
- ▶ ORBs

The following simplified example shows code to look up an EJB component:

```
accountHome = (AccountHome)initialContext
                .lookup("java:comp/env/ejb/account");
```

`java:comp/env/` is a standard prefix used to identify references, and `ejb/account` is bound at deployment time to the real JNDI name used for the Account bean.

Deployment

The Java EE specification only specifies the packaging format for Java EE application clients, not how to deploy them. This information is left to the platform provider. The packaging format is specified, based on the standard Java JAR format, and it allows the developer to specify which class contains the *main* method to be executed at run time.

Java EE application clients for the WebSphere Application Server platform run inside the *Application Client for WebSphere Application Server*. This product is available for download from IBM developerWorks and is available on the WebSphere Application Server installation CD.

See the WebSphere Application Server Information Center at the following address for more information about installing and using the application client for WebSphere Application Server:

<http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp>

The application client for WebSphere Application Server provides a **launchClient** command, which sets up the correct environment for Java EE application clients and runs the main class.

Requirements for the development environment

In addition to the standard Java tooling, the development environment must provide the following tools:

- ▶ A wizard for creating Java EE application clients
- ▶ Editors for the deployment descriptor for a Java EE application client module
- ▶ A mechanism for testing the Java EE application client

Rational Application Developer provides these features.

Figure 2-9 on page 67 shows how Java EE application clients fit into the picture. Because these applications can access other Java EE resources, we can now use the business logic in our session EJB from a stand-alone client application. Java EE application clients run in their own JVM, normally on a separate machine from the EJB, so they can only communicate using remote interfaces.

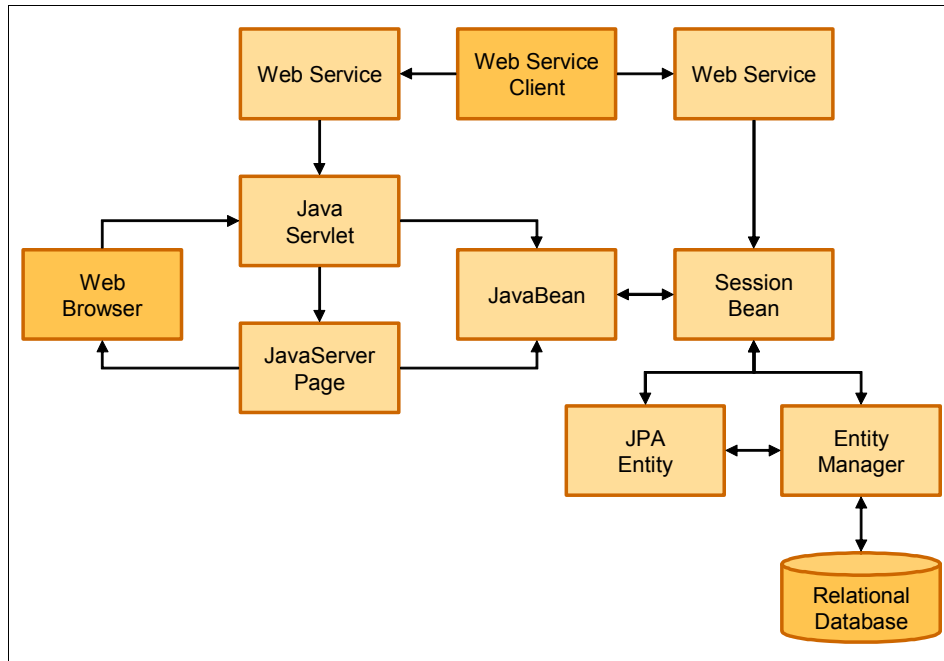


Figure 2-9 Java EE application client

2.7.2 Enterprise information system applications

Java EE Connector Architecture (JCA) plays a key role in the integration of applications and data using open standards. In developing applications to connect to enterprise information systems in Chapter 11, “Developing applications to connect to enterprise information systems” on page 531, we introduce JCA and demonstrate by example how to access operations and data on enterprise information systems (EIS), such as CICS®, IMS™, SAP, Siebel, PeopleSoft, JD Edwards, and Oracle, within the Java EE platform.

2.7.3 Service Component Architecture applications

Service Component Architecture (SCA) is a programming model for the service-oriented architecture (SOA) style. Rational Application Developer, WebSphere Application Server 7.0 Feature Pack for Service Component Architecture, and WebSphere Application Server 8.0 support the Open SCA assembly model specified by the Open Service Oriented Architecture (OSOA) Collaboration (<http://osoa.org>).

For more information about SCA, refer to Chapter 16, “Developing Service Component Architecture (SCA) applications” on page 885.

2.7.4 Session Initiation Protocol applications

Session Initiation Protocol (SIP) is a peer-to-peer protocol used to establish, modify, and terminate multimedia IP sessions between two endpoints, including telephony and instant messaging.

A SIP application is a Java program that uses at least one SIP servlet where a SIP servlet is a Java-based application component that is managed by a SIP Servlet Container (for example, WebSphere Application Server).

SIP Servlet Specifications were developed under the Java Community Process. See the JSR 289 SIP Servlet 1.1 API at this website:

<http://jcp.org/aboutJava/communityprocess/final/jsr289/index.html>

The following examples are common usage examples of SIP in telecommunications-based applications:

- ▶ Voice-over-IP (VoIP)
- ▶ Instant messaging
- ▶ Click to call
- ▶ Call notification, forwarding, and blocking

2.7.5 Communications Enabled Applications (CEA)

IBM WebSphere Application Server Feature Pack for CEA V1.0 is a set of libraries, widgets, and runtime components that provides the ability to add dynamic web communications to any application or business process. This functionality includes the ability to establish a call between two users, to collaboratively browse the same web application, to integrate communications features in applications with PBX systems, and the additional features required to support these functions.

The Dojo widgets packaged with the CEA feature pack are pre-packaged components of JavaScript and HTML code that add interactive features that work across platforms and browsers. CEA widgets are extensible, allowing developers to customize them to handle more advanced tasks. These widgets provide capabilities, such as making and disconnecting calls and receiving incoming call notifications. The CEA feature pack comes with four core widgets and three mobile widgets. These widgets have been built using the Dojo Toolkit, and they are provided in the CEA custom Dojo Toolkit that ships with the feature pack.

CEA support is available in WebSphere Application Server V7 through the IBM WebSphere Application Server Feature Pack for CEA V1.0 and is embedded directly into WebSphere Application Server V8 Beta.

CEA core widgets

The following list shows the CEA core widgets:

- ▶ **Call Notification**
Allows users to enter their phone number and receive notifications of incoming calls.
- ▶ **Click To Call**
Allows users to enter their phone number and request an immediate callback from your company.
- ▶ **Cobrowse**
Allows users to share the same browsing session, with one user controlling the session.
- ▶ **Two Way Form**
Allows you to create an HTML form in which two people, operating as a reader and a writer, can collaboratively edit and validate fields. Both parties can see the same form. The fields in the form change in response to input provided by either person.

CEA mobile widgets

The following list shows the CEA mobile widgets:

- ▶ **Mobile Call Notification**
Allows users to enter their mobile phone number and receive notifications of incoming calls
- ▶ **Mobile Click To Call**
Allows users to enter their mobile phone number and request an immediate callback from your company
- ▶ **Mobile Cobrowse**
Allows a mobile phone number to be used for collaborating and cobrowsing

For more information about using the CEA feature pack, refer to *Getting Started with the WebSphere Application Server Feature Pack for Communications Enabled Applications V1.0*, REDP-4613.



Workbench setup and preferences

In this chapter, we describe the most commonly used Rational Application Developer preferences.

The chapter is organized into the following sections:

- ▶ Workbench basics
- ▶ Preferences

3.1 Workbench basics

After starting Rational Application Developer, you see a window with the Welcome page (Figure 3-1). You can always return to the Welcome page by selecting **Help** → **Welcome** from the workbench menu bar. The Welcome page guides a new user of Rational Application Developer to information about various aspects of the tool.

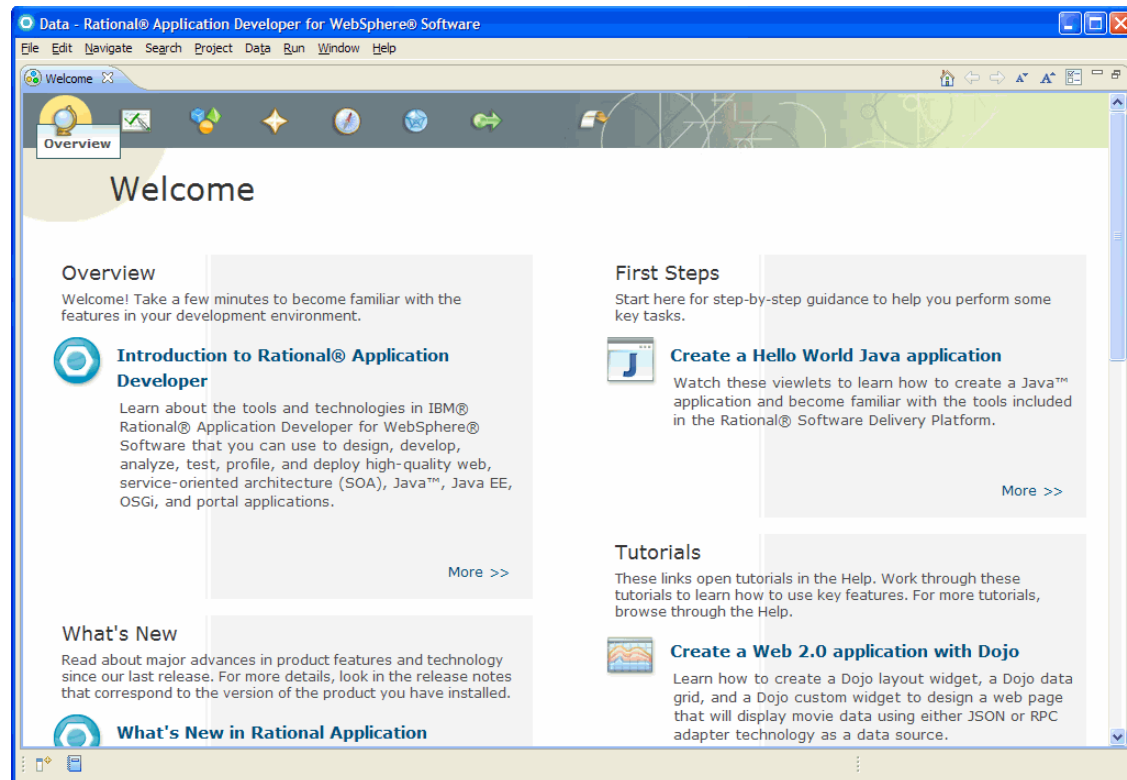











Figure 3-1 Rational Application Developer Welcome page

The Welcome page presents seven icons, each including a name that is visible when hovering over the icon (hover help). Table 3-1 on page 73 provides a summary of each icon.

Table 3-1 Welcome page assistance capabilities

Icon image	Name	Description
	Overview	An overview of the key functions in Rational Application Developer.
	Tutorials	Tutorial screens to learn how to use the key features of Rational Application Developer. Provides a link to Tutorials Gallery.
	Samples	Sample code for the user to begin working with “live” examples with minimal assistance. Provides a link to the Samples Gallery.
	What's New	A description of the major new features and highlights of this release.
	First Steps	Step-by-step guidance to help first-time users to perform key tasks.
	Web Resources	URL links to web pages where you can find relevant and timely tips, articles, updates, and references to industry standards.
	Migrate	Guidance about migrating projects that you created using Rational Application Developer V7.5.x or V7.0.x to Rational Application Developer V8.0.
	Workbench	Minimizes the Welcome page into the workbench window's trim, continuing to offer smaller versions of these seven buttons while allowing the user to freely explore the workbench.

You can customize the Welcome page from the Preferences page. You can click the **Customize Page** icon () in the upper-right corner of the Welcome page to open the Customize window (Figure 3-2 on page 74). You can use the Customize window to select one of the predefined themes that affect the overall look of the Welcome window. You can also select which pages to show and the visibility, layout, and priority of the items within each page.

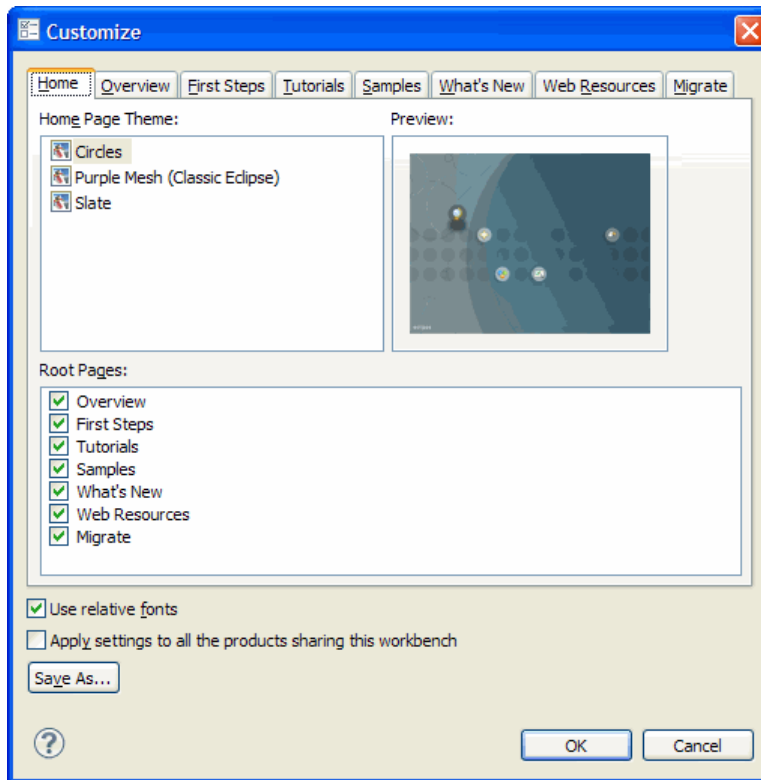


Figure 3-2 Welcome page preferences

Experienced users of Rational Application Developer, or anyone who knows the concepts that the product provides, can dismiss the Welcome page by clicking the X on the page's folder tab or clicking the workbench icon in the page itself. Users are then taken to the default perspective in the workbench.

The term *workbench* refers to the desktop development environment. Each workbench window of Rational Application Developer contains one or more perspectives. Perspectives control the initial layout of views and editors and what is displayed in certain menus and toolbars. Each perspective in Rational Application Developer contains multiple views, such as the Enterprise Explorer view and the Outline view. For more information about perspectives and views, see Chapter 4, "Perspectives, views, and editors" on page 91.

By clicking the shortcut icon in the window's perspective bar (Figure 3-3 on page 75), you can open the available perspectives and place them in the shortcut bar next to it. After the icons are on the shortcut bar, you can navigate between perspectives that are already open. The name of the active perspective is shown

in the title of the window, and its icon is in the shortcut bar on the right side as a push button.

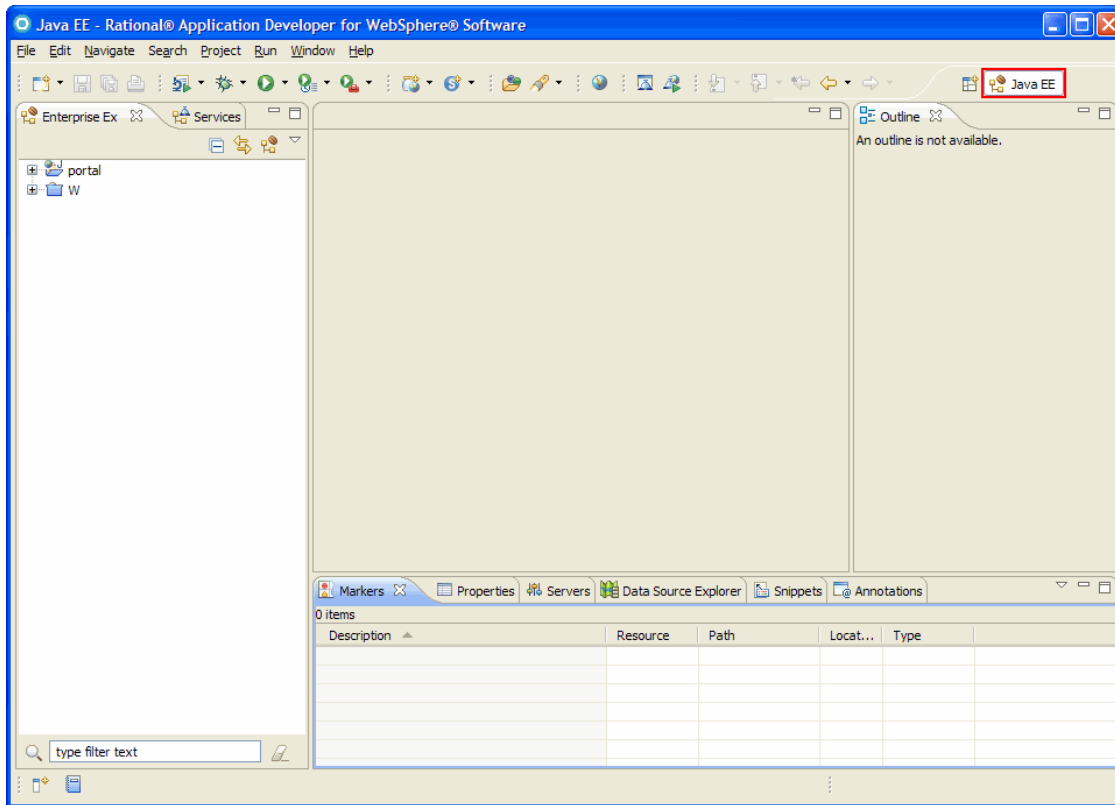


Figure 3-3 The Java EE perspective in Rational Application Developer

3.1.1 Workbench basics

When you start Rational Application Developer, you are prompted to provide a location for the workspace. The Rational Application Developer workspace is a private work area created for an individual developer. It holds the following information:

- ▶ Projects that the developer has created, including source code, images, configuration files, and generated files, such as .class files

- ▶ Metadata including information about the workspace's projects, configuration information specific to that machine and that workspace, preferences affecting the entire workspace, and temporary files

Performance: For optimal performance with Rational Application Developer, choose a location on a fast, local disk.

Resources that are modified and saved are reflected on the local file system. Users can have many workspaces on their local file system to contain separate projects that they are working on, including multiple versions of those projects. Each of these workspaces can be configured differently, because each workspace has its own metadata area.

Important: Do not copy or attempt to use the metadata from one workspace with another workspace. Instead, create a new workspace and then configure it appropriately. To facilitate this task, many workspace preferences can be imported and exported from the respective wizards or stored directly within the projects themselves, which can then be shared.

With Rational Application Developer, you can open more than one window at the same time. New windows open on the same workspace, allowing you to work quickly in two differing perspectives. Changes to the workspace that are made in one window are reflected to the other windows. You are not permitted to work in more than one window at a time; you cannot switch between windows when you are in the process of using a wizard in one window.

To open an additional workbench window, select **Window** → **New Window**. A new workbench window will then open with the same perspective. By default, new perspectives are opened in the current window. You can, however, choose to open new perspectives in their own windows. To configure this default behavior, select **Window** → **Preferences**. In the Preferences window, expand **General** → **Perspectives** (see 3.2.6, “Perspectives preferences” on page 87).

You can set a default workspace on the start-up of Rational Application Developer by specifying the workspace location on the local machine and selecting the **Use this as the default and do not ask again** check box, as shown in Figure 3-4 on page 77. This action ensures that on the next start-up of Rational Application Developer, the same workspace location will be used automatically and Rational Application Developer will not prompt for the workspace location in the future.

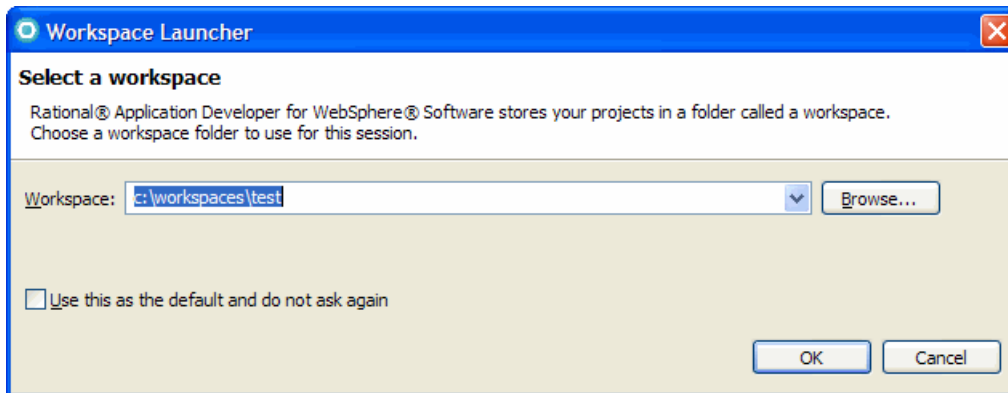


Figure 3-4 Setting the default workspace at start-up

The other way to enforce the use of a particular workspace is to use the **-data <workspace>** command-line argument on Rational Application Developer. Here, *<workspace>* is a path name on the local machine where the workspace is located and must be a full path name to remove any ambiguity about the location.

By using the **-data** argument, you can start a second instance of Rational Application Developer using a separate workspace. For example, if your second instance uses the MyWorkspace folder, you can start Rational Application Developer by entering the following command (assuming that the product has been installed in the default installation directory):

```
c:\Program Files\IBM\SDP\eclipse.exe -data c:\MyWorkspace
```

Tip: On a machine where multiple workspaces are used by the developer, create a dedicated shortcut in setting up the starting workspace location. Use the following target:

```
"<RAD Install Dir>\eclipse.exe" -product  
com.ibm.rational.rad.product.ide -data <workspace>"
```

You can add several arguments when starting Rational Application Developer.

Table 3-2 on page 78 lists useful arguments. For more advanced arguments, search for *Running Eclipse* in the Help under Help Contents.

Table 3-2 Start-up parameters

Command	Description
-configuration <i>configurationFileURL</i>	The location for the platform configuration file, expressed as a URL. The configuration file determines the location of the platform, the set of available plug-ins, and the primary feature. Relative URLs are not allowed. The configuration file is written to this location when Rational Application Developer is installed or updated.
-consolelog	Mirrors the Eclipse platform's error log to the console used to run Eclipse. Is convenient when combined with -debug .
-data <i><workspace directory></i>	Starts Rational Application Developer with a specific workspace located in <i><workspace directory></i> .
-debug [<i>optionsFile</i>]	Puts the platform in debug mode and loads the debug options from the file at the given location, if specified. This file indicates which debug points are available for a plug-in and whether they are enabled. If a file location is not given, the platform looks in the directory that eclipse was started from for a file called <i>.options</i> . Both URLs and file system paths are allowed as file locations.
-refresh	Option for performing a global refresh of the workspace on start-up to reconcile any changes made on the file system since the platform was last run.
-showlocation <i>[workspaceName]</i>	Option for displaying the location of the workspace in the window title bar. The Workspace preference page also provides the ability to specify a name for the workspace to be shown in the window title bar.
-vm vmPath	This option allows you to set the location of the Java Runtime Environment (JRE) to run Rational Application Developer. Relative paths are interpreted relative to the directory that Eclipse was started from. The JRE provided with Rational Application Developer is preferred.
-vmargs -Xmx512M	Allows for the passing of additional arguments to the JRE's VM executable. For instance, when doing large-scale development, you might want to make more heap space available. This example allows the Java heap to grow to 512 MB, although 512 MB might not be enough for even larger workspaces.

Use the `-vmargs` argument to set limits to the memory that is used by Rational Application Developer. For example, on a system with only 1 GB RAM, you might achieve better performance by limiting the amount of memory Rational Application Developer is allowed to use:

```
-vmargs -Xmx512M
```

You can also modify VMArgs initialization parameters in the `eclipse.ini` file (under the installation directory):

```
VMArgs=-Xms256M -Xmx512M
```

These arguments significantly limit the memory utilization. Setting the `-Xmx` argument lower than 512M begins to degrade performance. Each option meant for Eclipse in the `eclipse.ini` file must be on its own line.

Setting the workspace with a prompt window

The default behavior on installation is that Rational Application Developer prompts for the workspace on start-up. If you selected the check box on the start-up window to not ask again (Figure 3-4 on page 77), you can enable this option again in the following manner:

1. Select **Window** → **Preferences**.
2. In the Preferences window, select **General** → **Startup and Shutdown** → **Workspaces**.
3. Select **Prompt for workspace on startup** and click **OK** (Figure 3-5 on page 80).

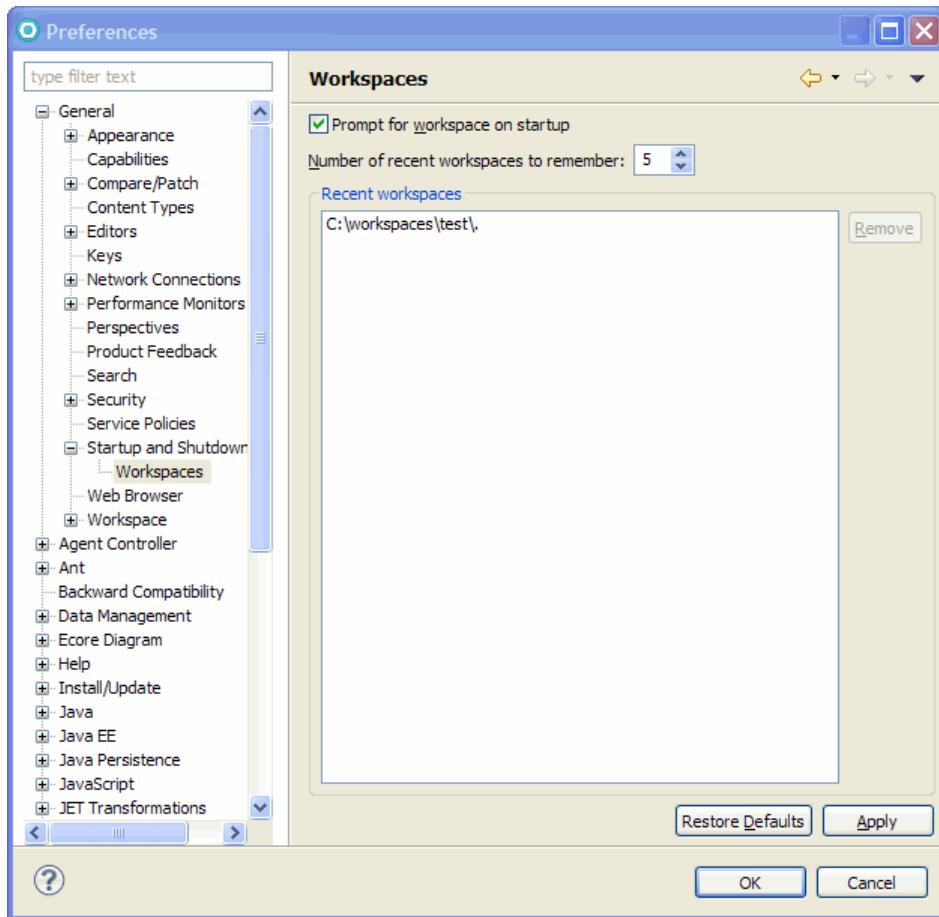


Figure 3-5 Setting the prompt message window for workspace selection on start-up

On the next start-up of Rational Application Developer, the workspace selection dialog is displayed and prompts the user to specify which workspace to use.

3.2 Preferences

You can modify the Rational Application Developer workbench preferences by selecting **Window** → **Preferences**. In the left pane of the Preferences window (Figure 3-6 on page 81), you can search through the preferences pages by typing keywords into the filter text field, or navigate through the categories of preference pages yourself.

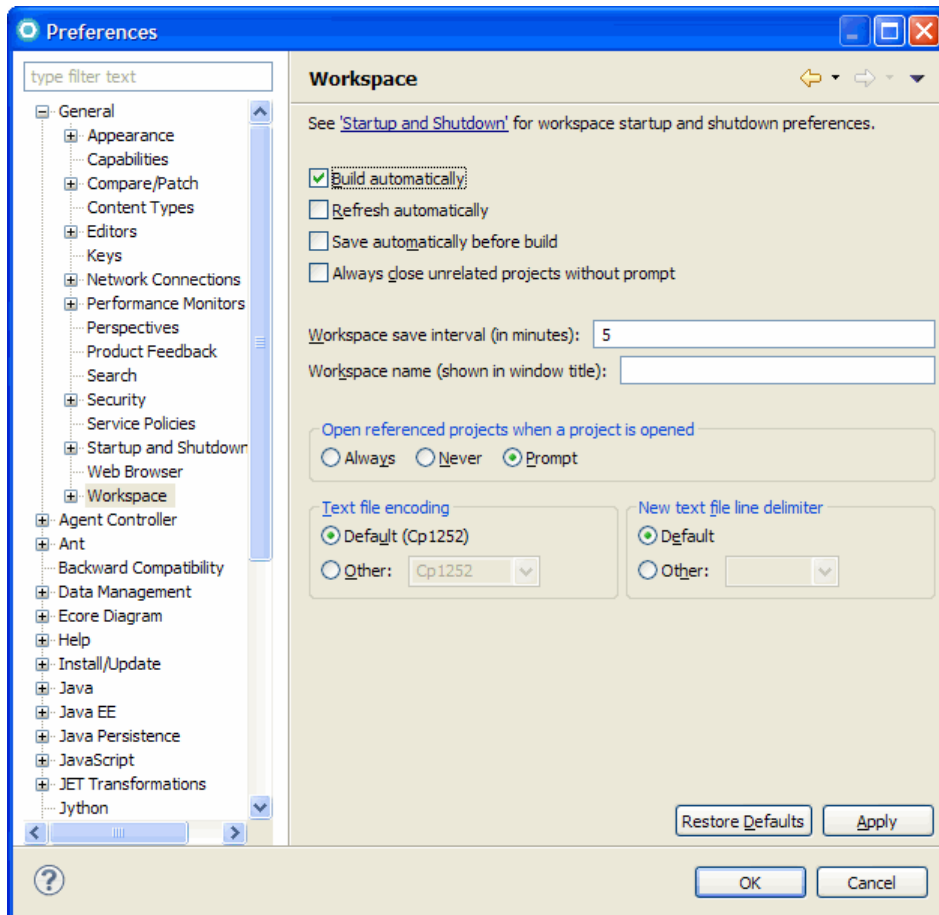


Figure 3-6 Workspace Preferences

In this section, we describe the most important workspace preferences. Rational Application Developer contains a complete description of all the options available in the Preferences window in the help information for Rational Application Developer.

Tip: Most pages within the Preferences window in Rational Application Developer have a **Restore Defaults** button (see Figure 3-6 on page 78). When you click the button, Rational Application Developer restores the settings of the current page to their initial values.

3.2.1 Automatic builds

Builds, or a compilation of Java code in Rational Application Developer, are done automatically whenever a resource has been modified and saved. If you require more control regarding builds, you can disable the automatic build feature. Builds must then be explicitly started. This capability might be desirable in cases where you know that building is of no value until you finish a large set of changes.

To turn off the automatic build feature, select **Windows** → **Preferences** → **General** → **Workspace** and clear **Build automatically** (Figure 3-7). In this same window, you can specify whether you want to save unsaved resources before performing a manual build. Select **Save automatically before build** to enable this feature.

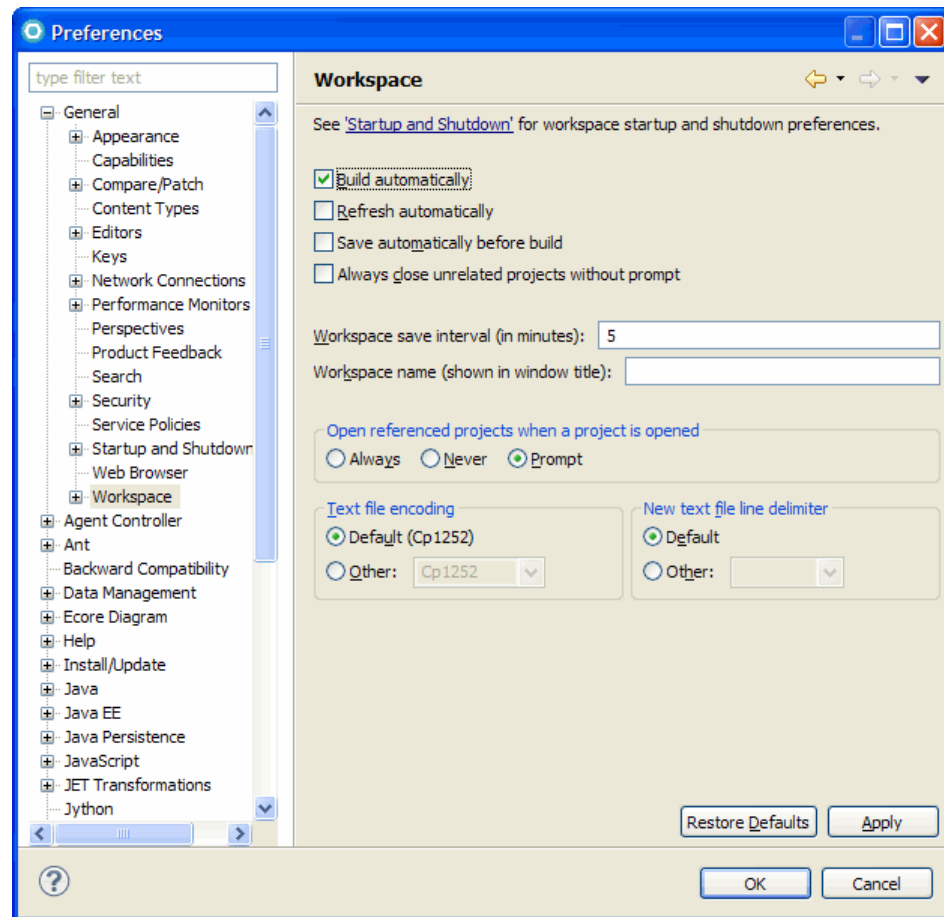


Figure 3-7 Workspace Preferences: Automatic builds

3.2.2 Manual builds

Although the automatic build feature might be adequate for many developers, there are scenarios in which a developer might want to perform a build manually. First, certain developers do not want to build automatically, because it can slow down development. In this case, the developer needs a method of building at a time of the developer's choosing. Second, there are cases when a complete rebuild of a project or all projects is needed to resolve build errors and dependency issues. To address these types of issues, Rational Application Developer provides the ability to perform a manual build, known as a *clean build*.

To perform a manual build, follow these steps:

1. Select the desired project in the Enterprise Explorer.
2. Select **Project** → **Build Automatically** to clear the check mark associated with that selection. The manual build option is only available when the automatic build is disabled.
3. Select **Project** → **Build Project**. Alternatively, select **Project** → **Build All** to build all projects in the workspace. Both of these commands search through the projects and only build the resources that have changed since the last build.

To build all resources, even those resources that have not changed since the last build, follow these steps:

1. Select the desired project in the Enterprise Explorer.
2. Select **Project** → **Clean**.
3. In the Clean window, select one of the following options and click **OK**:
 - Clean all projects. This option performs a build of all projects.
 - Clean selected projects.

The project selected in the previous step is chosen by default, or you can select it from the projects list.

3.2.3 File associations

On the File Associations preferences page, you can add or remove file types recognized by the workbench. You can also associate editors or external programs with file types in the file types list. Follow these steps to add a file association:

1. Open the Preferences window by selecting **Window** → **Preferences**.
2. In the Preferences window (Figure 3-8 on page 84), expand **General** → **Editors** and select **File Associations**. In the upper-right pane, you can add

and remove the file types. In the lower-right pane, you can add or remove the associated editors. We add the Microsoft Internet Explorer as an additional program to open database definition language (.ddl) files.

3. Select ***.ddl** from the file types list and click **Add** next to the Associated editors pane.
4. In the Editor Selection window, select **External Programs** and click **Browse**.
5. Locate `ieexplore.exe` in the folder where Internet Explorer is installed (for example, `C:\Program Files\Internet Explorer`) and click **Open**.
6. In the Editor Selection window, click **OK** and the program is added to the editors list.

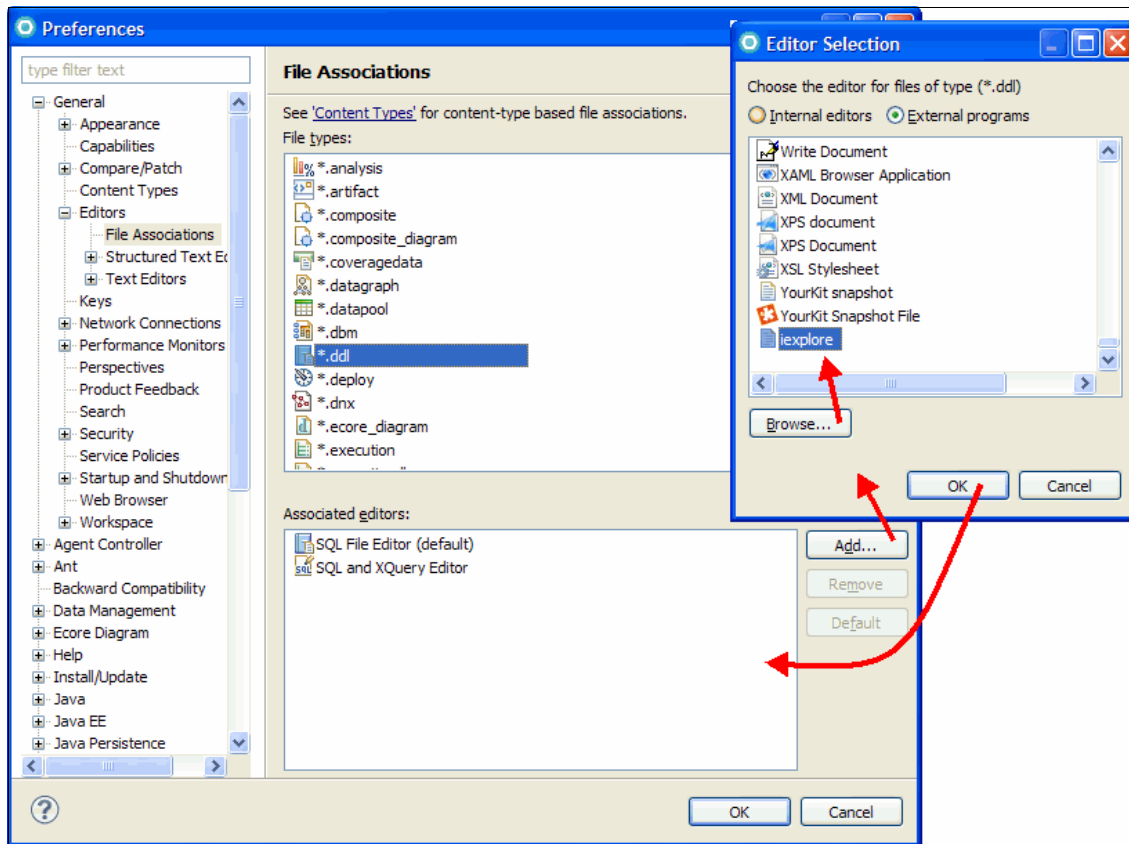


Figure 3-8 File Associations preferences

Optional: You can set this program as the default program for this file type by clicking Default.

Now you can open a .ddl file by using the context menu on the file, selecting **Open With**, and selecting the appropriate program.

3.2.4 Content types

The Content Types preferences page allows you to modify the default encoding used with certain content types, as well as specify new filename patterns containing a known content type (Figure 3-9).

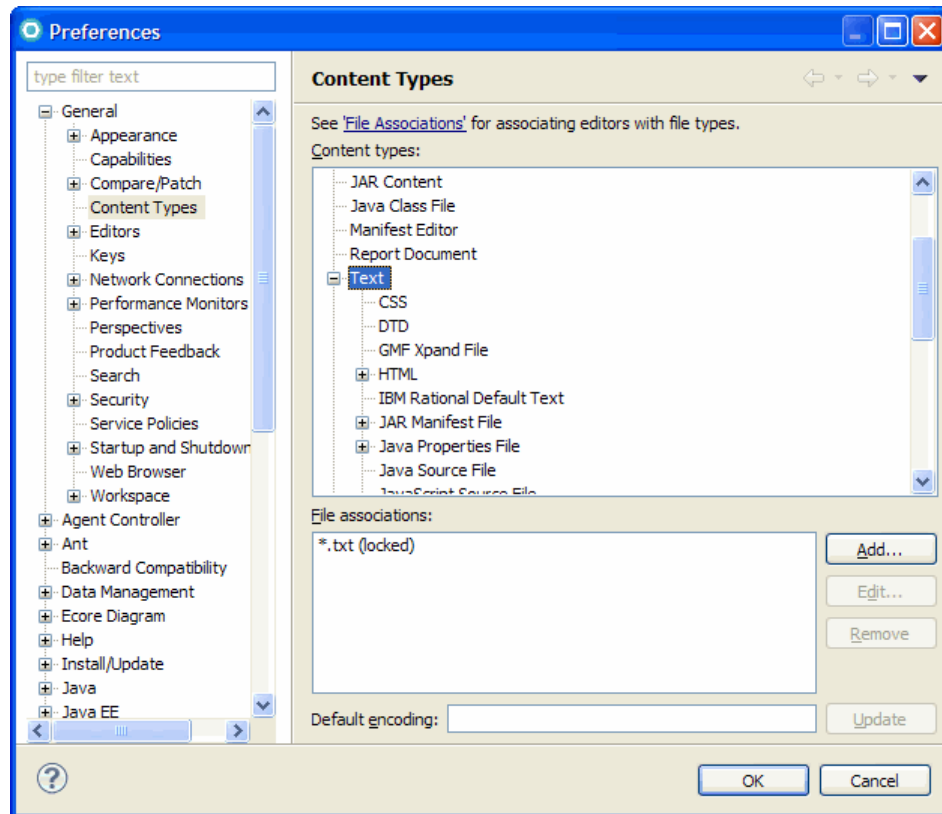


Figure 3-9 Workspace Preferences: Content Types

3.2.5 Local history

A local history of a file is maintained whenever you create or modify a file in the workspace. By default, a copy is saved each time that you edit and save the file. This local history allows you to replace the current file with a previous edition or even restore a deleted file. You can also compare the content of all the local

editions. Each edition in the local history is uniquely represented by the data and the time that the file was saved.

Files versus projects and folders: Only a file can have a local history. Projects and folders do not have local histories beyond whether a file existed or not. The local history is not meant as a complete substitute for a true source control system.

To configure local history settings, select **Window** → **Preferences**. Expand **General** → **Workspace** and select **Local History** to open the Preferences page (Figure 3-10).

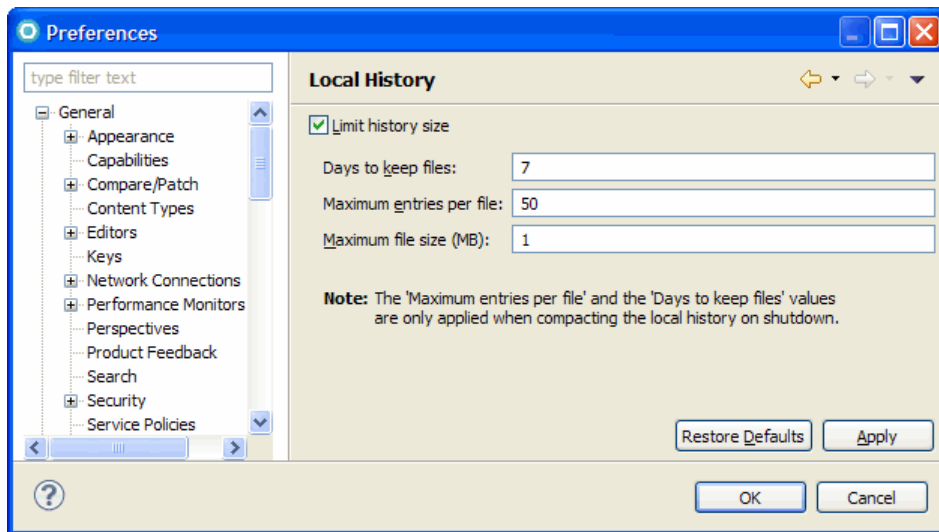


Figure 3-10 Local History preferences

Table 3-3 explains the options for the local history preferences.

Table 3-3 Local history settings

Option	Description
Limit history size	Enabled by default, controls whether the following options take effect.
Days to keep files	Indicates the number of days to maintain changes in the local history. History states older than this value are lost.
Maximum entries per file	Indicates the number of history states per resource that you want to maintain in the local history. History states older than this value are lost.

Option	Description
Maximum file size (MB)	Indicates the maximum size of individual states in the history store. If a resource is over this size, no local history is kept for that resource.

Comparing, replacing, and restoring local history

To compare a file with the local history, follow these steps:

1. Select the file, right-click, and select **Compare With** → **Local History**. In the upper pane of the Compare with Local History window, all available editions of the file in the local history are displayed.
2. Select an edition in the upper pane to view the differences between the selected edition and the edition in the workspace.
3. When you are finished with the comparison, click **OK**.

To replace a file with an edition from the local history, follow these steps:

1. Select the file, right-click, and select **Replace With** → **Local History**.
2. Select the desired file time stamp and then click **Replace**.

To restore a deleted file from the local history, follow these steps:

1. Select the folder or project from which the file was deleted.
2. Right-click and select **Restore from Local History**.
3. Select the files that you want to restore and click **Restore**.

3.2.6 Perspectives preferences

The Perspectives preferences page enables you to manage the various perspectives that are defined in the workbench. To open the page, select **Window** → **Preferences** and expand **General** → **Perspectives**.

You can change the following options in the Perspectives preferences window:

- ▶ Open a new perspective in the same or in a new window.
- ▶ Open a new view within the perspective or as a fast view (docked to the side of the current perspective).
- ▶ Always switch, never switch, or prompt when a particular project is created to switch to the appropriate perspective.

There is also a list with all available perspectives from which you can select the default perspective. If you have added one or more customized perspectives, you can delete them from here. See Figure 3-11 on page 88.

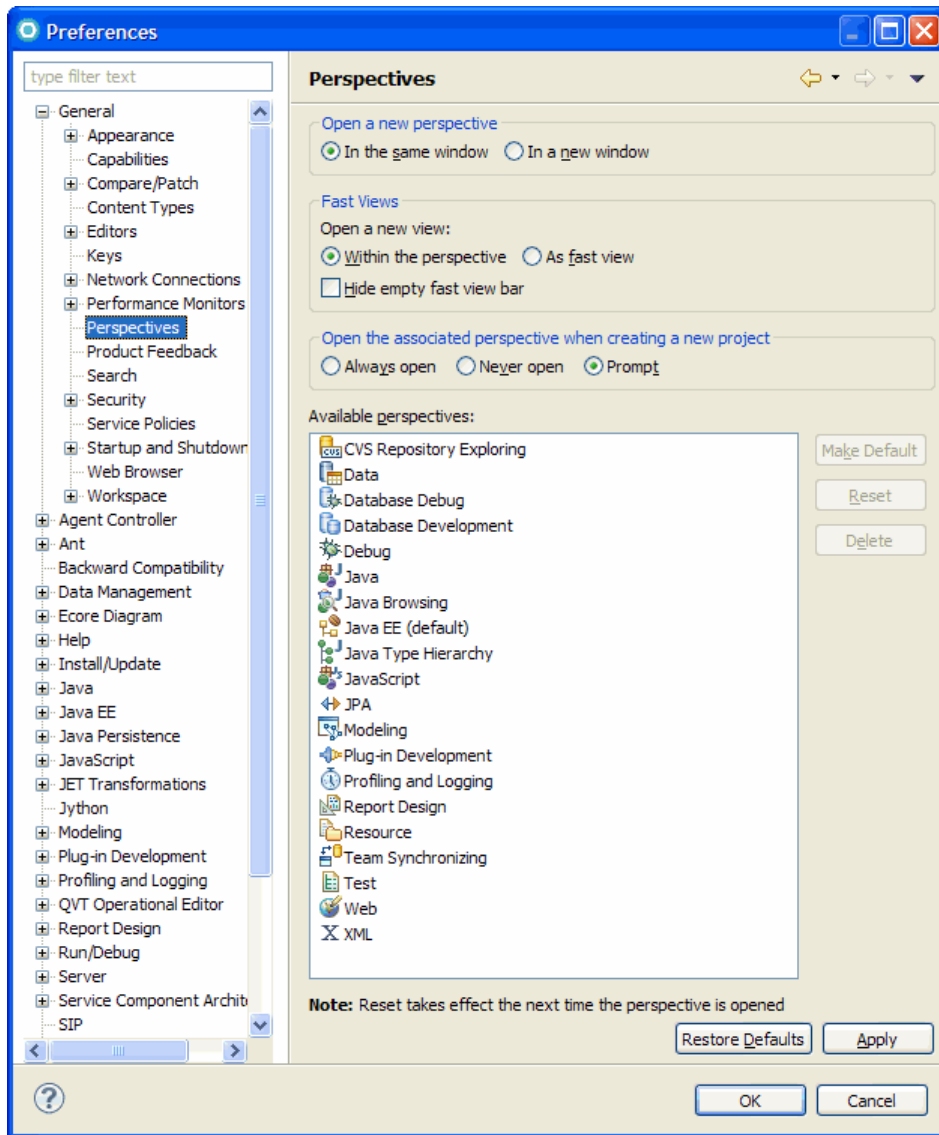


Figure 3-11 Perspectives preferences

3.2.7 Web browser preferences

With the web browser settings, the user can select which web browser is the default browser used by Rational Application Developer for showing web information.

To change the web browser settings, follow these steps:

1. Select **Window** → **Preferences**. Expand **General** → **Web Browser** (Figure 3-12). The default option is to use an internal web browser.
2. To change, select **Use external web browser** and select a browser from the available list. Otherwise, click **New** to add a new web browser.

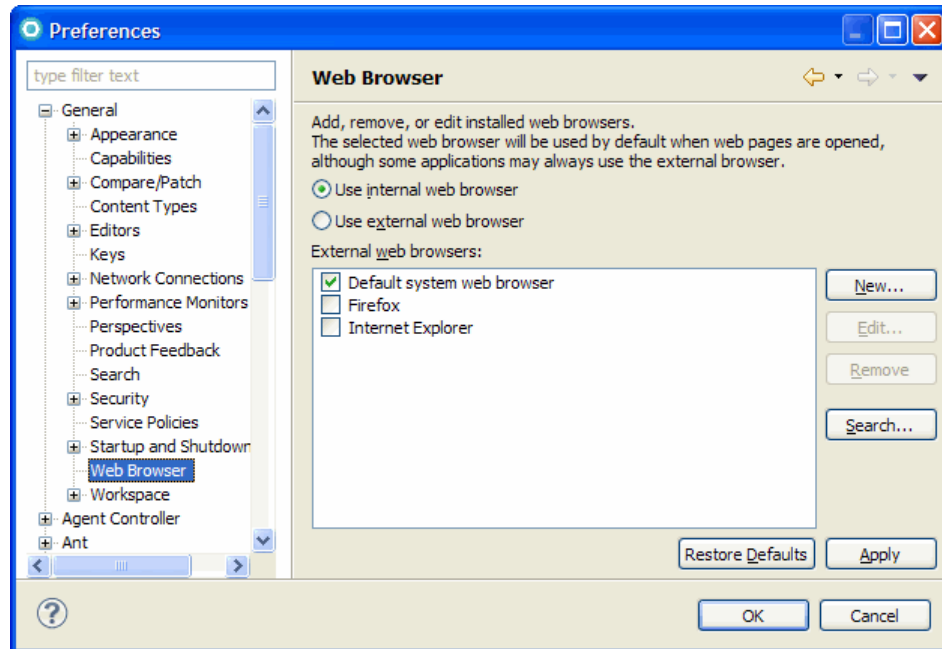


Figure 3-12 Web browser preferences

3.2.8 Internet preferences

You can configure the following types of settings in Internet preferences in Rational Application Developer:

- ▶ Cache
- ▶ FTP
- ▶ Proxy settings

Only proxy settings are covered in this section. For the other two settings, see the help information for Rational Application Developer.

Proxy settings

To set the preferences for the proxy server within the workbench to allow Internet access from Rational Application Developer, follow these steps:

1. Select **Window** → **Preferences**.
2. In the Preferences window (Figure 3-13), in the left pane, expand **General** → **Network Connections**.
3. In the Network Connections pane on the right, follow these steps:
 - a. Based on your environment proxy settings, select **HTTP** or **HTTPS** and click **Edit**.
 - b. Enter the proxy host and port. Additional optional settings are available for the use of SOCKS and to enable proxy authentication.
 - c. Click **Apply** and then click **OK**.

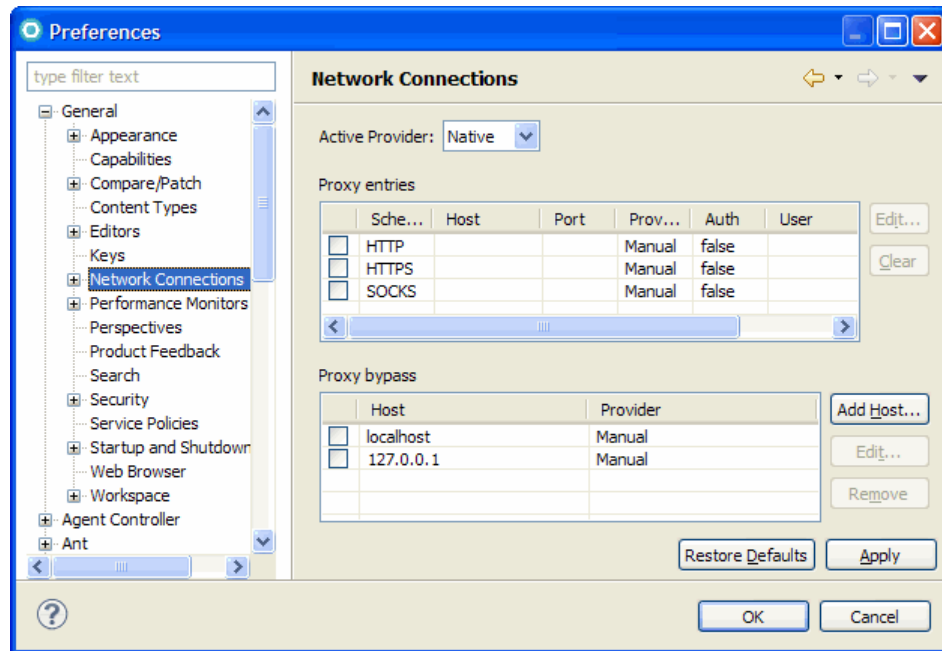


Figure 3-13 Network Connections preferences page



Perspectives, views, and editors

This chapter starts with an introduction to the common structures and features applicable to all perspectives in Rational Application Developer and then describes how these mechanisms integrate with the help facility. Then we provide a brief overview of the major features for each perspective available in Rational Application Developer. Most of the perspectives described here are explored in detail in the chapters in this book.

The chapter is organized into the following sections:

- ▶ Integrated development environment
- ▶ Help system for Rational Application Developer
- ▶ Available perspectives
- ▶ Summary

4.1 Integrated development environment

An *integrated development environment* (IDE) is a set of software development tools, such as source editors, compilers, and debuggers, that are accessible from a single user interface.

In Rational Application Developer, the IDE is called the *workbench*. When using the workbench, the first step of any user is to choose in which perspective to work. The Rational Application Developer workbench provides many customizable perspectives organized around various development duties. This design provides a common way for all members of a project team to create, manage, and navigate the same set of resources easily.

4.1.1 Perspectives

In Application Developer terminology, *views* are the windows that provide various ways to look at the resources on which you are working, and *editors* allow you to create and modify the resources. Each perspective consists of a set of views and editors that show various aspects of the workspace resources for a particular developer role or task. For example, a Java developer might work in the Java perspective, which contains views for Java coding that aid in working with the Java editor, and a web designer might work in the Web perspective, which contains views for web page design that are useful with Rational Application Developer's Page Designer. Open editors are available from all perspectives in the same workbench window.

Several default perspectives are provided in Rational Application Developer, and team members also can customize them according to their current role and personal preference. More than one perspective can be opened at a time, and users can switch perspectives while working within Rational Application Developer. If you find that a particular perspective does not contain the views or editors that you require, you can add them to the perspective and position them to suit your requirements. We explain this capability further in 4.1.7, "Organizing and customizing perspectives" on page 98.

4.1.2 Views

Views provide various presentations of resources or ways of navigating through the information in your workspace. For example, the Enterprise Explorer view provides a hierarchical view of the resources in the workbench, arranged in a way to facilitate Java EE development. From here, you can open files for editing or select resources for operations, such as exporting a file as an EAR file. The Outline view shows an outline of a structured file that is currently open in the

editor area and lists structural elements. Rational Application Developer provides synchronization between views and editors, so that changing the focus or a value in an editor or view can automatically update another editor or view. In addition, certain views display information obtained from other software products, such as database systems or software configuration management (SCM) systems.

A view can be displayed by itself or stacked with other views in a tabbed notebook arrangement. To quickly move between views in a given perspective, you can hold down the Ctrl key and press F7 to see all the open views and move quickly to the desired view. While continuing to hold the Ctrl key, press F7 until the desired view is selected and then release the key to move to that view. Pressing Shift-F7 allows you to move through the list in reverse order.

Tip: *Quick Access*, which is invoked by using Ctrl+3, allows you to open and switch among editors, views, and perspectives, and to trigger commands by filtering on their names.



4.1.3 Editors

When you open a file, Rational Application Developer automatically opens the editor that is associated with that file type. For example, for .html, .htm, and .jsp files, the Page Designer opens, and for .java and .jpage files, the Java editor opens.

Editors that have been associated with specific file types open in the editor area of the workbench. By default, editors are stacked in a notebook arrangement inside the editor area. If a resource has no associated editor, Rational Application Developer opens the file in the default editor, which is a text editor. It is also possible to open a resource in another editor by selecting the **Open With** option from the context menu.

To quickly move between editors open on the workspace, you can hold down the Ctrl key and press F6 to view all the open editors and move quickly to the desired editor. Press F6 until the required editor is selected and then release the Ctrl key or press Shift+F6 to move through the files in reverse order.



The following icons are in the toolbar of a perspective to facilitate navigation and basic operations in editors:

- ▶ Next and Previous accessed files ( and )


These icons move the focus around recent cursor positions.

- ▶ Last Edit Location ()

This icon shifts the cursor to where the last edit occurred.

- ▶ Next and Previous Annotation ( and )

Depending on the options selected in the associated drop-down menu, these icons move the cursor to the next or previous annotation in the associated list. For example, if errors are chosen, these buttons move the cursor to the next or previous source code error in the resource being edited.

- ▶ Toggle Breadcrumb ()

In supported editors, this option activates a breadcrumb navigation at the top of the editor, as shown in Figure 4-1. This is available only in certain editors.

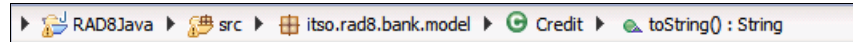



Figure 4-1 Breadcrumb navigation

- ▶ Toggle Mark Occurrences ()

If this option is activated, all other instances of a highlighted text will be marked by gray shading.

- ▶ Toggle Block Selection Mode ()

If this option is selected, it is possible to select, cut, copy, and paste rectangular blocks of text from the active editor.

- ▶ Show Whitespace Characters ()

If this option is selected, the whitespace, new-line, and other control characters will be visible in the active editor.

4.1.4 Perspective layout

Many of Rational Application Developer's perspectives use a similar layout. Figure 4-2 on page 95 shows the general layout that is used for most default perspectives.

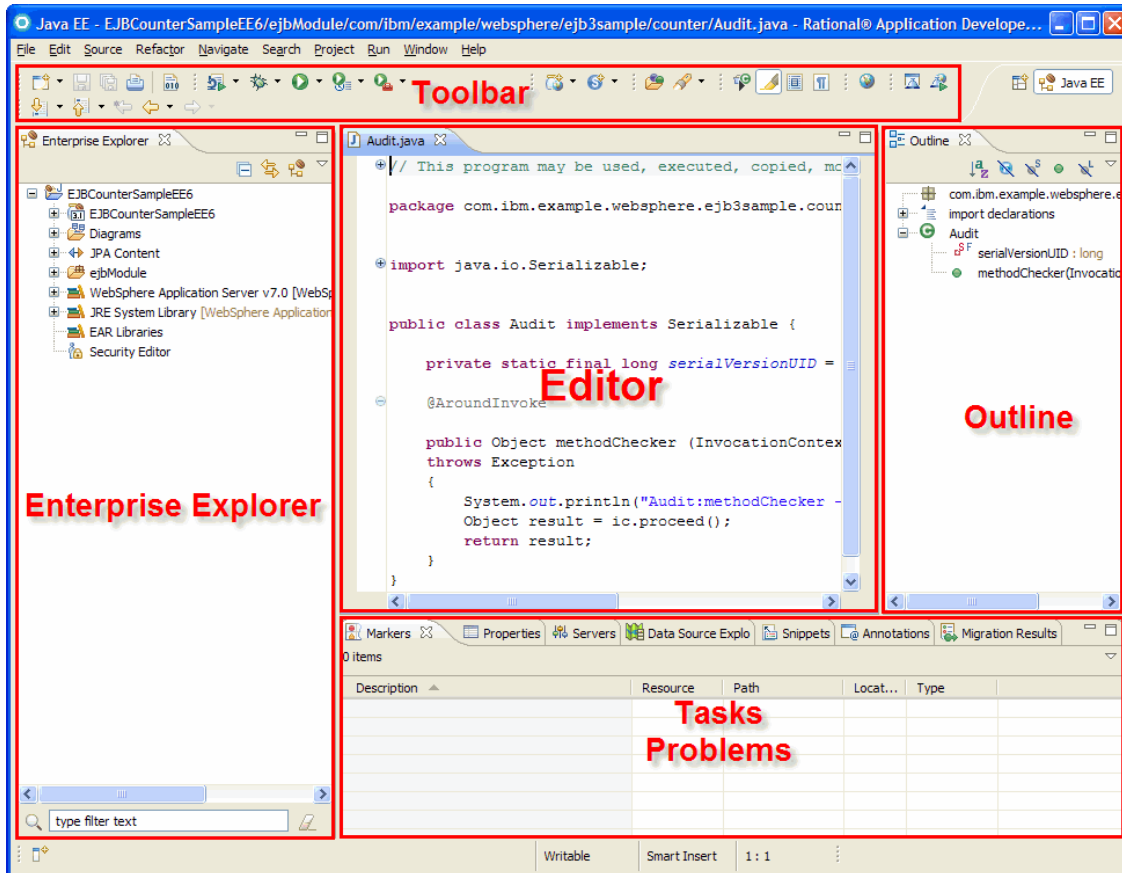



Figure 4-2 Perspective layout

On the left side are views for navigating through the workspace. In the middle of the workbench is a larger pane, where the main editors are shown. The right pane usually contains Outline or Palette views, views for working with the file open in the main editor. In certain perspectives, the editor pane is larger and the outline view is located at the lower-left corner of the perspective. In the lower-right corner is a tabbed series of views, including the Tasks view, the Problems view, and the Properties view. This area is where smaller miscellaneous views, which are not associated with resource navigation, editing, or outline information, are shown.

4.1.5 Switching perspectives

There are two ways to open another perspective:

- ▶ Click the **Open a perspective** icon () in the Perspective bar that is located in upper-right corner of the workbench window and select the appropriate perspective from the list.
- ▶ Select **Window** → **Open Perspective** and select a perspective from the drop-down list shown.

In both cases, an Other option is available. When you select **Other**, the Open Perspective window (Figure 4-3) opens and shows a list of all perspectives. Here, you can select the required perspective and click **OK**.

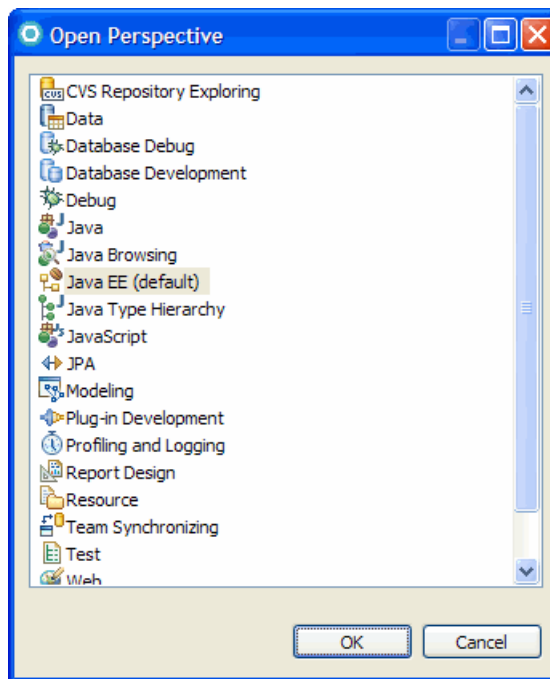



Figure 4-3 Open Perspective window

In all perspectives, you can see a group of buttons displayed in the upper-right corner of the workbench (an area known as the *shortcut bar*). Each button corresponds to an open perspective, and if clicked, the  icon shows a list of all open perspectives (see Figure 4-4 on page 97). Clicking one of these buttons switches Rational Application Developer to the associated perspective.

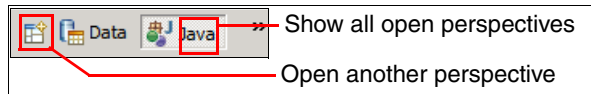


Figure 4-4 Perspective shortcut bar

Tips:

- ▶ The name of the perspective is shown in the window title area along with the name of the file that is open in the editor, which is currently active.
- ▶ To close a perspective, right-click the perspective's button on the shortcut bar (top right) and select **Close**.
- ▶ To display only the icons for the perspectives, right-click somewhere in the shortcut bar and clear the **Show Text** option.
- ▶ Each perspective requires memory. Therefore, it is a good practice to close perspectives, which are not used, to improve performance.

4.1.6 Specifying the default perspective

The Java EE perspective is Rational Application Developer's default perspective, but you can change this default perspective by using the Preferences window:

1. From the workbench, select **Window** → **Preferences**.
2. In the Preferences window, in the left pane, expand **General** and select **Perspectives**. The Java EE perspective has the word default after it.
3. In the right pane, select the perspective that you want to define as the default and click **Make Default**. The selected perspective will become the default and have the word default after it.

4.1.7 Organizing and customizing perspectives

With Rational Application Developer, you can open, customize, reset, save, and close perspectives. You can find these options in the Window menu.

To customize the commands and shortcuts available within a perspective, select **Window** → **Customize Perspective**. The Customize Perspective window opens (see Figure 4-5).

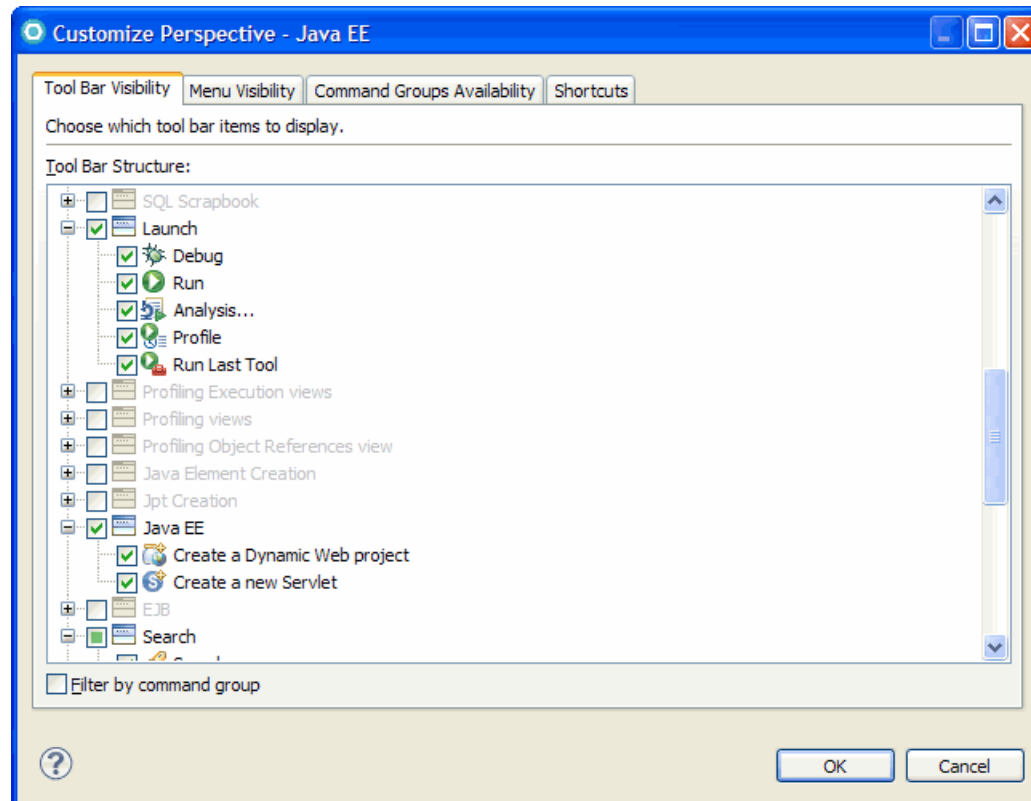


Figure 4-5 Customize Perspective window

The tabs on this window allow users to customize the following commands and options:

- ▶ Tool Bar Visibility tab







Provides the ability to customize which icons will appear on the Rational Application Developer tool bar.

- ▶ **Menu Visibility tab**
Provides the ability to customize which options will appear in the Rational Application Developer menus.
- ▶ **Command Groups Availability tab**
Rational Application Developer comes with a number of command groups to perform specific tasks. The Command Groups Availability tab allows you to customize whether a given command group is available in a perspective, and if so, in which menu option or tool bar it will appear.
- ▶ **Shortcuts tab**
The Shortcuts tab allows you to customize which options will appear when a menu item, such as **File** → **New** → **Project**, is selected. The items that you do not select are still accessible by clicking the Other menu option, which is always present for these options.

In addition to customizing the commands and options available as shortcuts and menu items, you can reposition any of the views and editors and add or remove other editors as desired and save the changes as a customized perspective. The following features are available to create a customized perspective:

- ▶ **Add and remove views**
It is possible to customize a perspective by adding a new view. To add a view to a perspective, select **Window** → **Show View** and choose the view that you want to add. To remove a view, close it from its title bar.
- ▶ **Move**
You can move a view to another pane by using the drag-and-drop method. To do this, select its title bar and drag the view to another place on the workspace. While you drag the view, the mouse cursor changes to a drop cursor, indicating where the view will be displayed when it is dropped. In each case, the area that is filled with the dragged view is highlighted with a rectangular outline.

The drop cursor looks like one of the following icons:

-  The view docks beneath the view under the cursor.
-  The view docks to the left of the view under the cursor.
-  The view docks to the right of the view under the cursor.
-  The view docks over the view under the cursor.
-  The view is displayed as a tab in the same pane as the view under the cursor.
-  The view docks in the status bar (at the bottom of the Rational Application Developer window) and becomes a fast view (described

next). This icon is displayed when a view is dragged to the lower-left corner of a workspace.






The view becomes a separate child window of the main Rational Application Developer window. This icon is displayed when you drag a view to an area outside the workspace. To return the view back into the workspace, right-click its title bar and clear the **Detached** menu item.

► Fast view

A *fast view* is displayed as a button in the status bar of Rational Application Developer in the lower-left corner of the workspace. Clicking the button toggles whether the view is displayed on top of the other views in the perspective.

► Maximize and minimize a view

To maximize a view to fill the whole working area of the workbench, you can double-click the title bar of the view, press Ctrl+M, or click the Maximize icon () in the view's toolbar. To restore the view, double-click the title bar, select the restore button () or press Ctrl+M again. The Minimize button () in the toolbar of a view minimizes the tab group so that only the tabs are visible. Click the **Restore** button or one of the view tabs to restore the tab group.

► Save

After you configure the perspective to your preferences, you can save it as your own perspective by selecting **Window** → **Save Perspective As** and type a new name. The new perspective is now displayed as an option in the Open Perspective window. Unsaved changes to a perspective will be lost if the perspective is closed.

► Restore

To restore the currently open perspective to its original layout, select **Window** → **Reset Perspective**.

4.2 Help system for Rational Application Developer

With the Help system in Rational Application Developer, you can browse, search, bookmark, and print help documentation. The documentation is organized into sets of information that are analogous to books. The Help system also supplies a text search capability for finding the information that you need by search phrase or keyword, and context-sensitive help for finding information to describe the particular function with which you are working.

You can view the Help contents in a separate window by selecting **Help** → **Help Contents** from the menu bar (Figure 4-6).

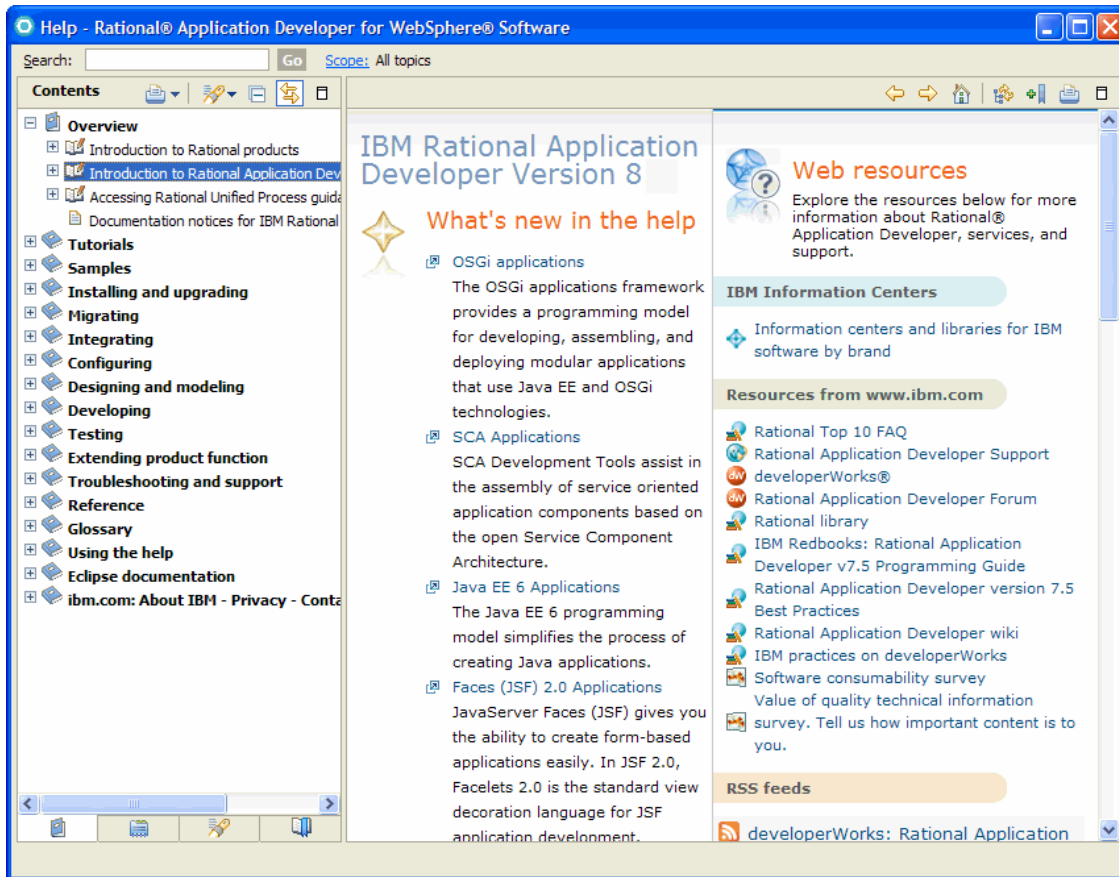







Figure 4-6 Help window

In the Help window, you see the available books in the left pane and the content in the right pane. When you select a book (📖) in the left pane, the appropriate table of contents opens and you can select a topic (📄) within the book. When a page (📄) is selected, the page content is displayed in the right pane.

You can navigate through the help documents by clicking the **Go Back** icon (⏪) and **Go Forward** icon (⏩) in the toolbar of the right pane. The **Home** icon (🏠) returns the Help window back to the home page.

The following buttons are also available in the toolbar:

- ▶ Show in Table of Contents ()
This button synchronizes the navigation frame with the current topic, which is helpful when the user follows several links to related topics in several files, and wants to see where the current topic fits into the navigation path.
- ▶ Bookmark Document ()
This button adds a bookmark to the Bookmarks view, which is one of the tabs on the left pane.
- ▶ Print Page ()
This button provides the option to print the page currently displayed in the right window.
- ▶ Maximize ()
This button maximizes the rightmost pane to fill the whole Help window. When this pane is maximized, the icon changes to the **Restore** icon (), which allows the user to return the page back to normal.

Also, the left pane of the Help window can be tabbed between the Contents, Index, Search Results, and Bookmarks views, which provide separate methods of accessing information in the help contents.

You can use the Search tab to do a search of all the help contents by default. If you want to do a refined search, from the Rational Application Developer menus, select **Help** → **Search**, then expand the **Search Scope** link, and select a scope for your search (Figure 4-7 on page 103). This window shows any previously defined search scopes and gives the user the opportunity to create a new scope or even add new sources of Help information.

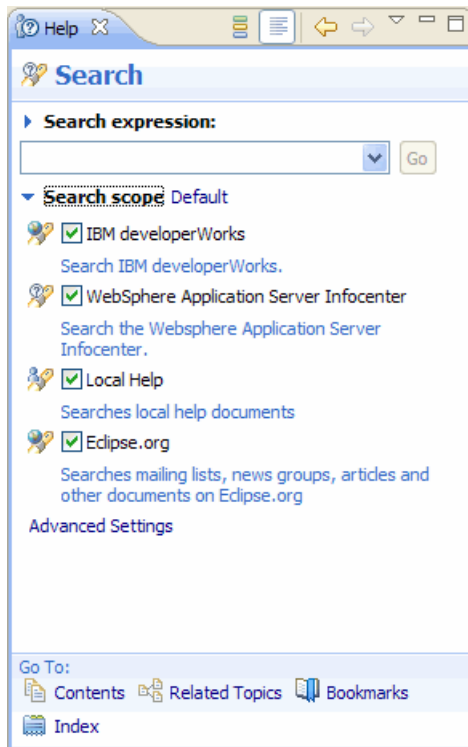


Figure 4-7 Search dialog

You can click **Go** to perform the search across the selected scope and display the results in the Search Results view, and from there, you can open the pages within the Help facility.

4.2.1 Context-sensitive help

While performing any task within Rational Application Developer, you can press F1 at any time, and the Help view shows the context help with a list of relevant topics for the current view, editor, and perspective. For example, Figure 4-8 on page 104 shows the context help when editing normal Java code.

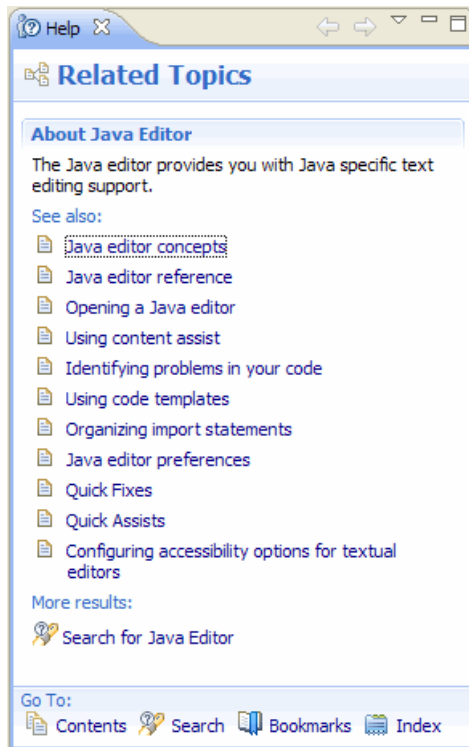


Figure 4-8 Context-sensitive help for Java editing

You can see the Help contents as a view within the workspace, by selecting **Window** → **Show View** → **Help** → **Help** or **Help** → **Dynamic Help**.

4.3 Available perspectives

In this section, we briefly describe the perspectives that are available in Rational Application Developer. We present the perspectives in alphabetical order, which is how they are displayed in the Open Perspective window. For this section, every installation option available has been installed. If this is not done, certain perspectives might not be available.

Rational Application Developer includes the following perspectives:

- ▶ CVS Repository Exploring perspective
- ▶ Data perspective
- ▶ Database Debug perspective
- ▶ Database Development perspective

- ▶ Debug perspective
- ▶ Java perspective
- ▶ Java Browsing perspective
- ▶ Java EE perspective
- ▶ Java Type Hierarchy perspective
- ▶ JavaScript perspective
- ▶ JPA perspective
- ▶ Modeling perspective
- ▶ Plug-in Development perspective
- ▶ Profiling and Logging perspective
- ▶ Report Design perspective
- ▶ Resource perspective
- ▶ Team Synchronizing perspective
- ▶ Test perspective
- ▶ Web perspective
- ▶ XML perspective

4.3.1 CVS Repository Exploring perspective

With the CVS Repository Exploring perspective (Figure 4-9), you can connect to the Concurrent Versions System (CVS) repositories and inspect the revision history of resources in those repositories.

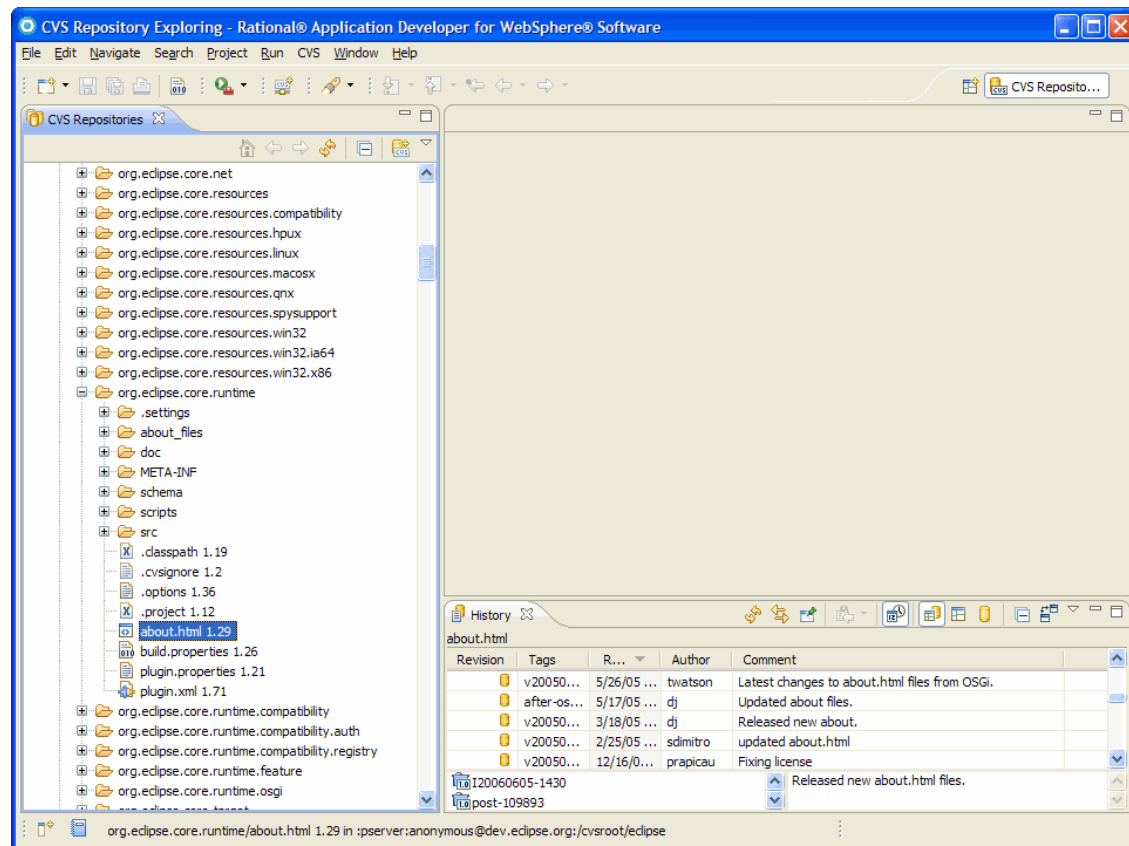


Figure 4-9 CVS Repository Exploring perspective

The CVS Repository has the following views:

- ▶ CVS Repositories view

This view shows the known CVS repository locations. Expanding a location reveals the main trunk (HEAD), project versions, and branches in that repository. You can further expand the project versions and branches to reveal the folders and files that are contained in them.

The context menu for this view also allows you to specify new repository locations. The CVS Repositories view can be used to check out resources

from the repository into the workspace, configure the branches and versions of a repository, view a resource's history, and compare resource versions.

▶ Editor

You can view files that exist in the repositories by double-clicking them in a branch or version. The version of the file specified opens in the editor pane. The contents of the editor are read-only.

▶ History view

This view shows a detailed history of each file, providing a list of all the revisions of it in the repository. From this view, you can also compare two revisions or open an editor on a specific revision.

▶ CVS Annotation view

To see this view, select a resource in the CVS Repositories view, right-click, and select **Show Annotation**. The CVS Annotate view comes to the front and will display a summary of all the changes made to the resource since it came under the control of the CVS server. The CVS Annotate view links with the main editor, showing which CVS revisions apply to which source code lines.

For more information about using the CVS Repository Exploring perspective, and other aspects of CVS functionality in Rational Application Developer, see Chapter 29, “Concurrent Versions System (CVS) integration” on page 1533.

4.3.2 Data perspective

With the Data perspective (Figure 4-10 on page 108), you can access a set of relational database tools, where you can create and manipulate the database definitions for your projects.

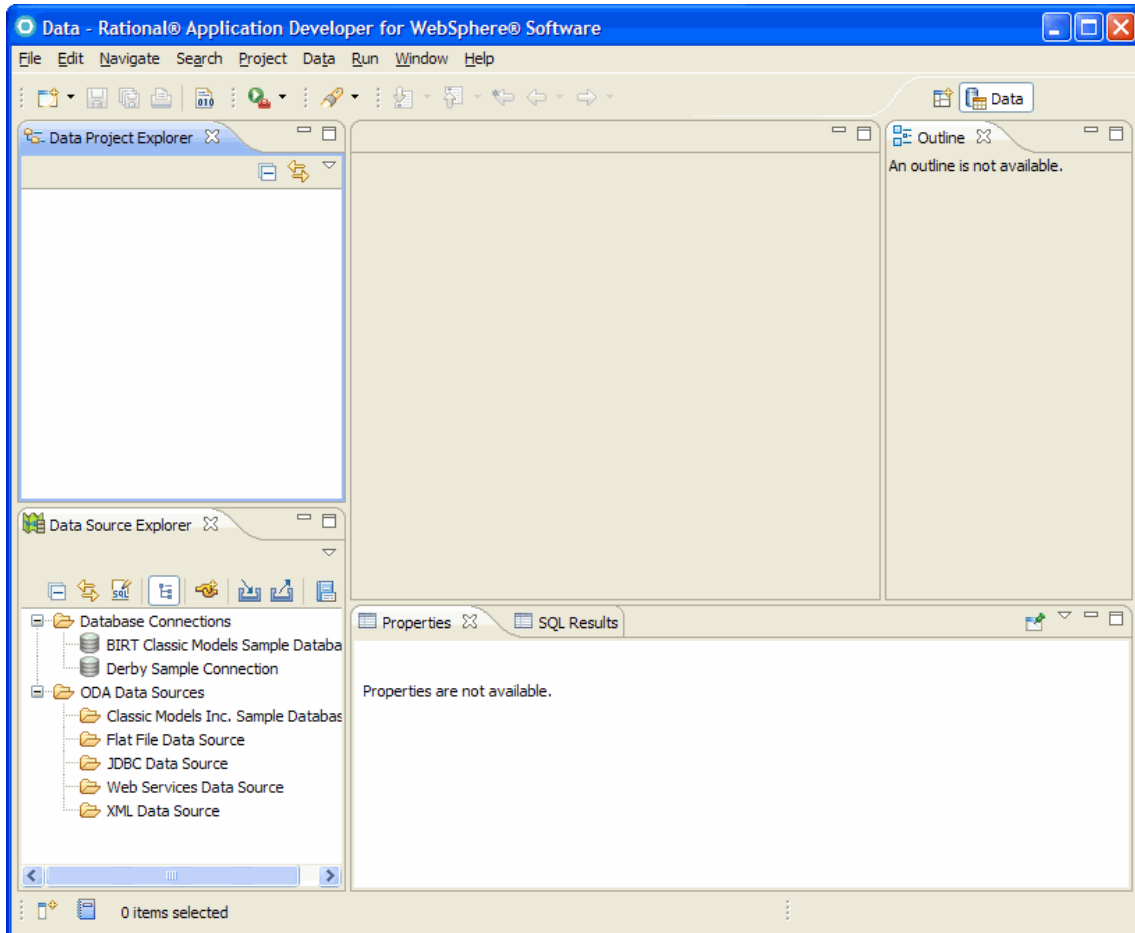



Figure 4-10 Data perspective

The Data perspective has the following views:

- ▶ Data Project Explorer

The main navigator view in the Data perspective shows only the data projects in the workspace. With this view, you can work directly with data definitions and define relational data objects. This view can hold local copies of imported data definitions, designs created by running Data Definition Language (DDL) scripts, or new designs that you have created directly in the workbench.

- ▶ Data Source Explorer view

This view provides a list of configured connection profiles. If the **Show Category** button () is selected, you can see the list grouped into categories, for example, Databases and ODA (Open Data Access) Data

Sources. Use the Data Source Explorer to connect to, navigate to, and interact with the resources associated with the selected connection profile. It also provides import  and export  capabilities to share connection profile definitions with other workbenches.

- ▶ SQL Results view

The SQL Results view shows information about actions that are related to running SQL statements, stored procedures, and user-defined functions (UDFs), or creating database objects. For example, when you run a stored procedure on the database server, the SQL Results view shows messages, parameters, and the results of any SQL statements that are run by the stored procedure. The SQL Results view also shows results when you sample the contents of a selected table. The SQL Results view consists of a history pane and a details pane. The history pane shows the history for past queries. The details pane shows the status and results of the last run. Use the view's pull-down menu to filter history results and set preferences.

- ▶ SQL Builder/Editor

This view shows specialized wizards for creating and editing SQL statements.

- ▶ Data Diagram Editor

This view shows an Entity Relationship diagram of the selected database.

- ▶ Tasks view

The Tasks view shows system-generated tasks associated with a resource, typically produced by builders. You can manually add tasks and optionally associate them with a resource in the workspace.

- ▶ Navigator view

The optional Navigator view provides a hierarchical view of all the resources in the workbench. By using this view, you can open files for editing or select resources for operations, such as exporting. The Navigator view is essentially a file system view, showing the contents of the workspace and the directory structures used by any projects that have been created outside the workspace.

- ▶ Console view

The Console view shows the output of a process and allows you to provide keyboard input to a process. The console shows three kinds of text, each in a separate color: standard output, standard error, and standard input.

For more details about using the Data perspective, see Chapter 9, “Developing database applications” on page 393.

4.3.3 Database Debug perspective

By using the Database Debug perspective (Figure 4-11), you can debug your database stored procedures, where you can watch the values of the variables and monitor the breakpoints.

This perspective includes the Debug, Variables, Breakpoints, Outline, and SQL Results views. We explain the views associated with debugging in 4.3.5, “Debug perspective” on page 112.

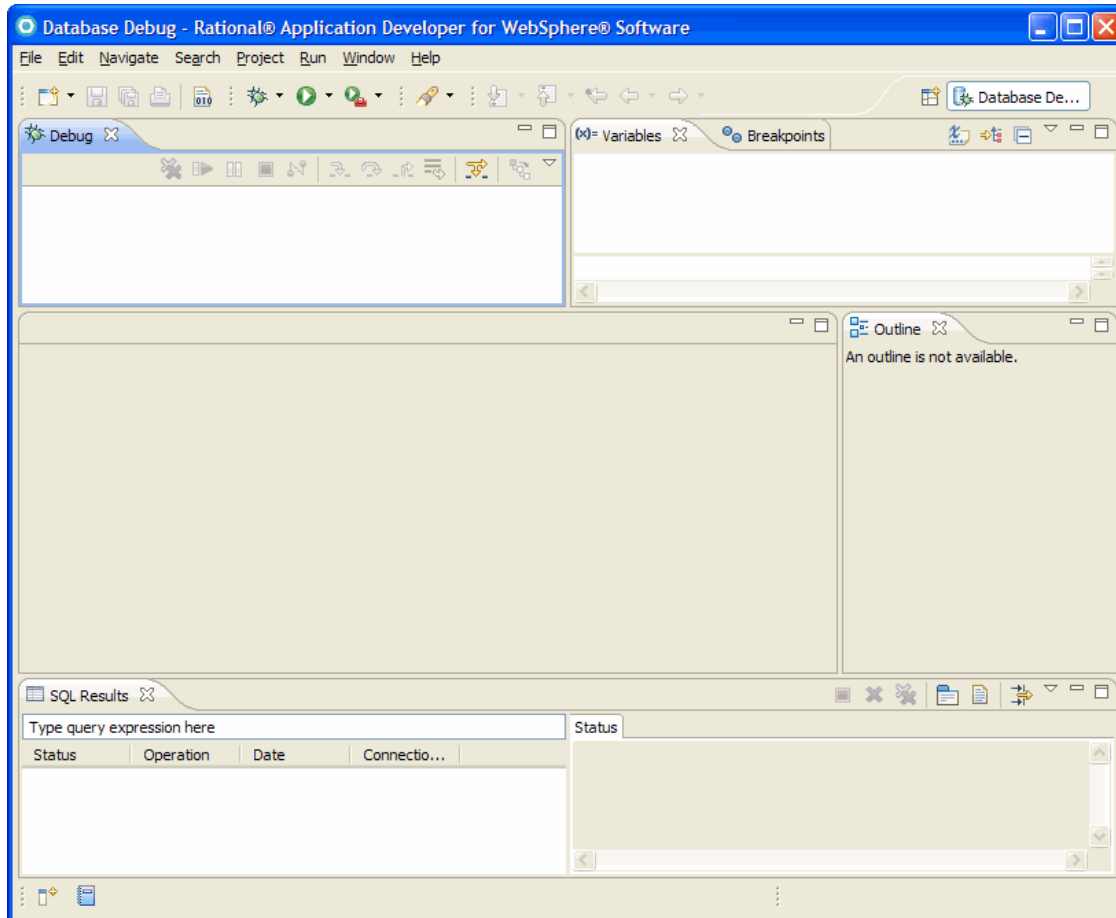


Figure 4-11 Database Debug perspective

4.3.4 Database Development perspective

The Database Development perspective (Figure 4-12) is a simpler version of the Data perspective with only one view added, which is the Execution Plan view. With this view, you can see your current SQL execution plans, which helps you optimize the execution of your queries. You can also see a history of execution plans and read SQL execution plans from files. For details about this perspective, see Chapter 9, “Developing database applications” on page 393.

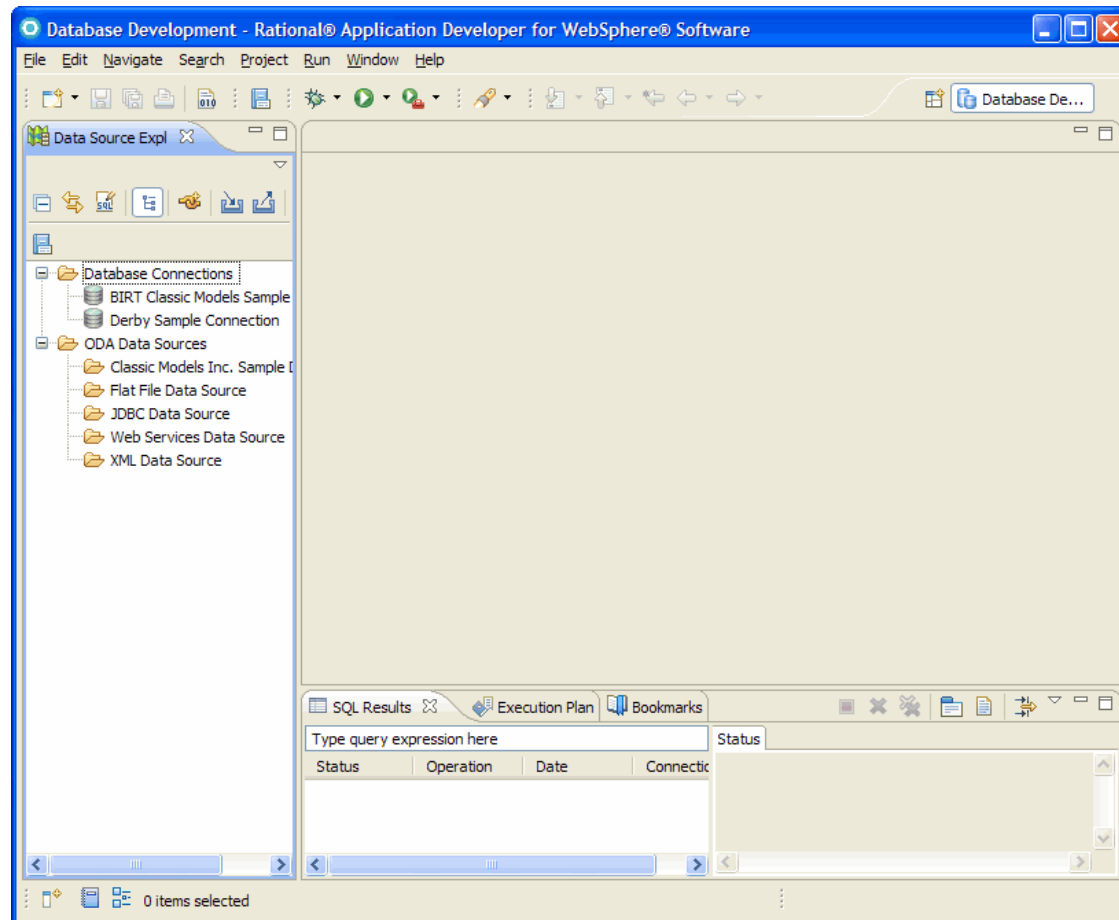


Figure 4-12 Database Development perspective

4.3.5 Debug perspective

By default, the Debug perspective (Figure 4-13) contains the following panes, each of which contains specific views:

- ▶ Upper left: Shows Debug and Servers views
- ▶ Upper right: Shows Breakpoints and Variables views
- ▶ Middle left: Shows the editor for the resource being debugged
- ▶ Middle right: Shows the Outline view of the resource being debugged
- ▶ Bottom: Shows the Console and the Tasks views

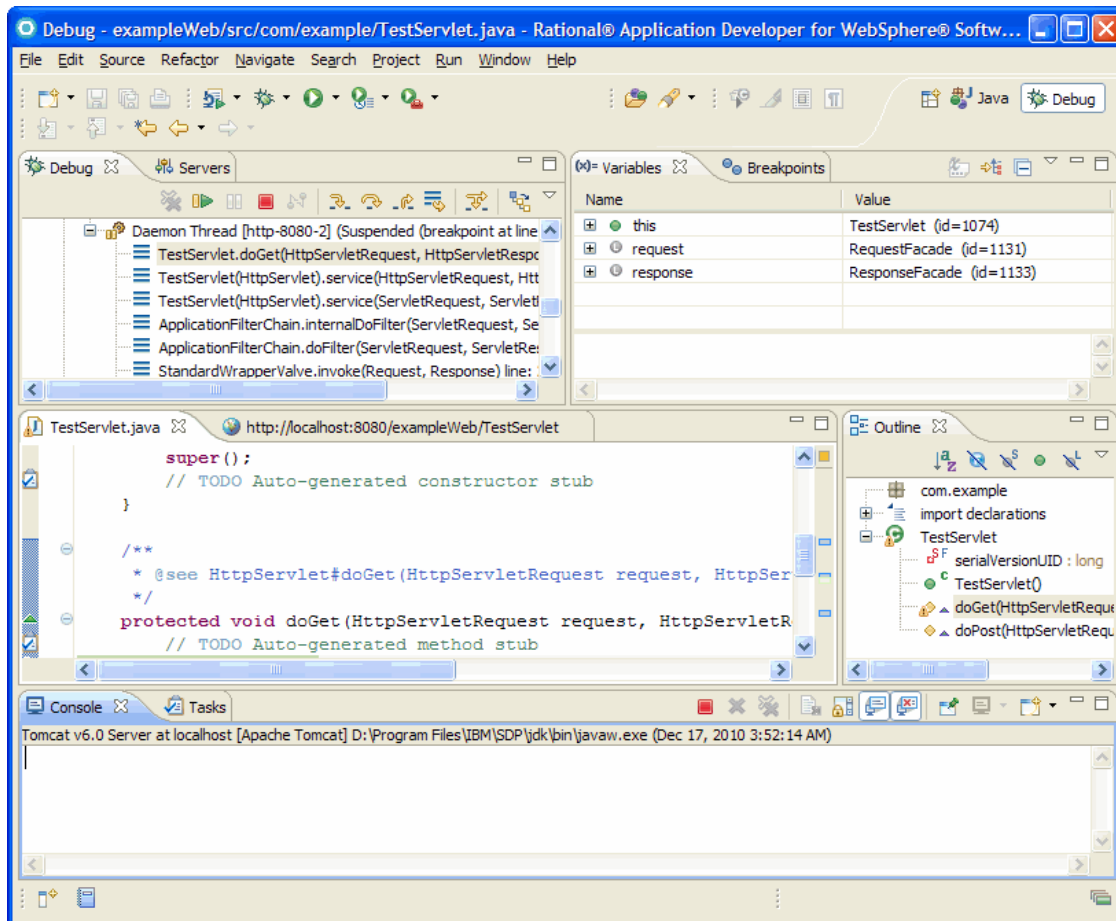


Figure 4-13 Debug perspective

The Debug perspective includes the following views:

- ▶ Debug view

The Debug view shows the stack frame for the suspended threads for each program that you are debugging. Each thread in your program appears as a node in the tree. If the thread is suspended, its stack frames are shown as child elements.

If the resource containing a selected thread is not open or active, the file opens in the editor and becomes active, focusing on the point in the source where the thread is currently positioned.

The Debug view contains a number of command buttons that enable users to perform actions, such as start, terminate, and step-by-step debug actions.

- ▶ Variables view

The Variables view shows information about the variables in the currently selected stack frame.

- ▶ Breakpoints view

The Breakpoints view lists all the breakpoints that you have set in the workspace's projects. You can double-click a breakpoint to display its location in the editor. In this view, you can also enable or disable breakpoints, remove them, change their properties, or add new breakpoints. This view also lists Java exception breakpoints, which suspend execution at the point where the exception is thrown.

- ▶ Servers view

The Servers view lists all the defined servers and their statuses. The context menu for a server allows the server to be started or stopped, and to republish the current applications.

- ▶ Outline view

The Outline view shows the elements (for example, imports, class, fields, and methods) that exist in the source file in the active editor. Clicking an item in the outline will position you in the editor view at the line where that structure element is defined.

The Console and Tasks views are also applicable to the Debug perspective. We discussed these views in previous sections of this chapter.

For more information about the Debug perspective, see Chapter 28, "Debugging local and remote applications" on page 1461.

4.3.6 Java perspective

The Java perspective (Figure 4-14) supports developers with the tasks of creating, editing, and compiling Java code.

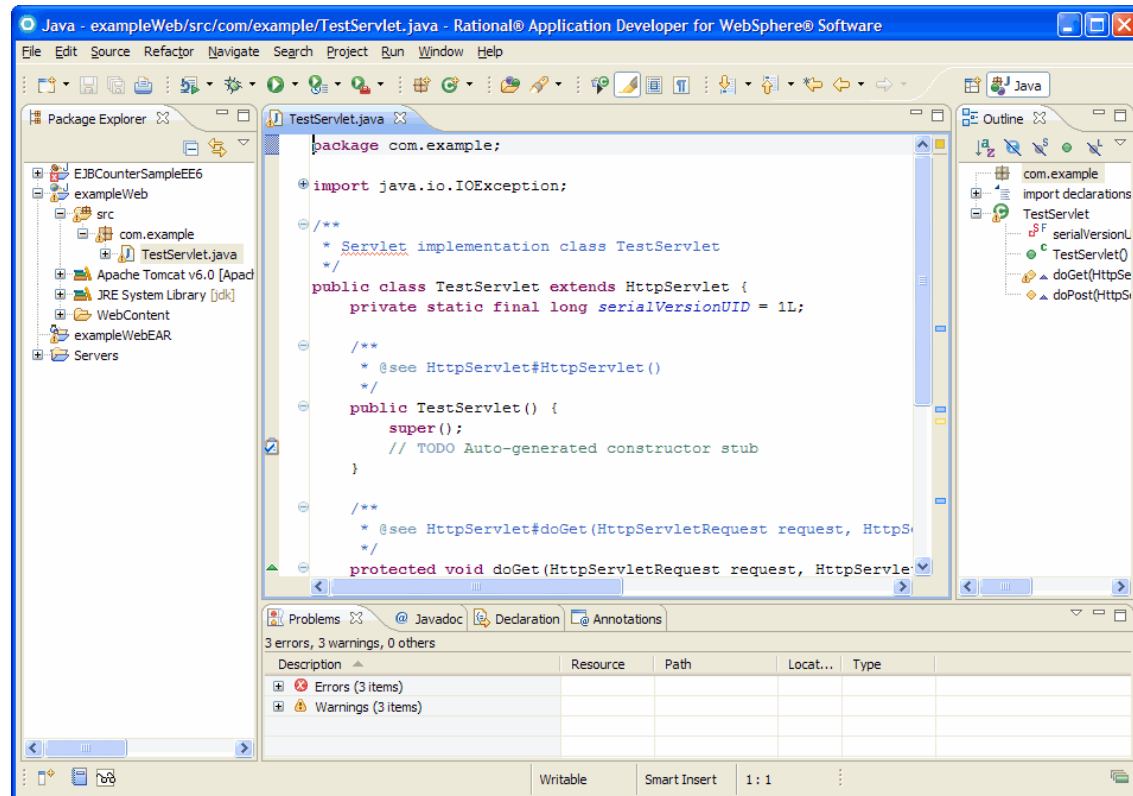


Figure 4-14 Java perspective




The Java perspective consists of a main editor area and shows, by default, the following views:

- ▶ Package Explorer view

This view shows the Java element hierarchy of all the Java projects in your workbench. This is a Java-specific view of the resources shown in the Navigator view (which is not shown, by default, in the Java perspective). For each project, its source folders and referenced libraries are shown in the tree view and from here it is possible to open and browse the contents of both internal and external JAR files.

► Hierarchy view

This view can be opened for a selected type to show its superclasses and subclasses. It offers three separate ways to look at a class hierarchy, by selecting the icons buttons at the top of the view:

- The **Type Hierarchy** icon () shows the type hierarchy of the selected type, including its position in the hierarchy along with all its superclasses and subclasses.
- The **Supertype Hierarchy** icon () shows the supertype hierarchy of the selected type and any interfaces that the type implements.
- The **Subtype Hierarchy** icon () shows the subtype hierarchy of the selected type or, for interfaces, shows classes that implement the type.

For more information about the Hierarchy view, see 4.3.9, “Java Type Hierarchy perspective” on page 118.

► Javadoc view

This view shows the Javadoc comments associated with the element selected in the editor or outline view.

► Declaration view

This view shows the source code declaration of the element selected in the editor or outline view.

► Annotations view

This view summarizes all the annotations on the Java class file being editing and provides menu options to add or delete annotations quickly.

The Outline and Tasks views are also applicable to the Java perspective. We discussed these views in previous sections of this chapter.

For more information about how to work with the Java, Java Browsing, and Java Type Hierarchy perspectives, see Chapter 7, “Developing Java applications” on page 229.

4.3.7 Java Browsing perspective

The Java Browsing perspective is also for Java development (Figure 4-15 on page 116), but it provides various views from the Java perspective.

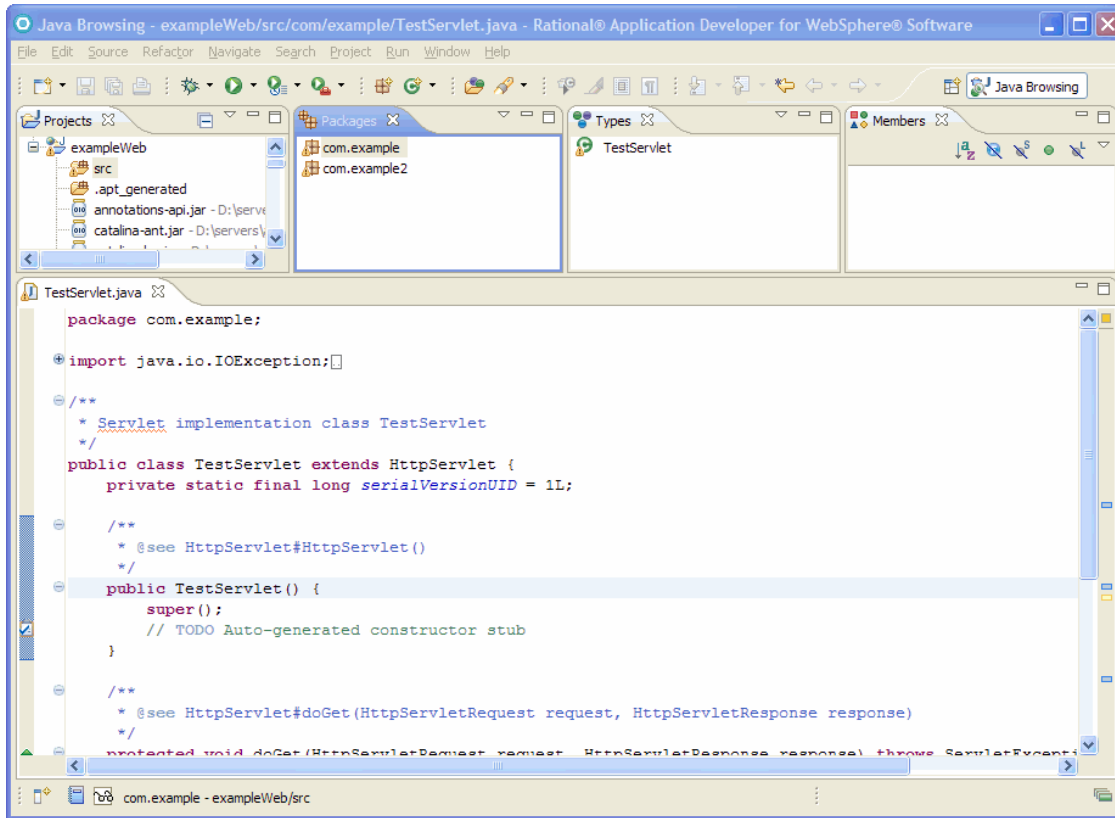


Figure 4-15 Java Browsing perspective

This perspective includes a larger area for the editor and several views to select the Java programming element that you want to edit:

- ▶ **Projects view**
This view lists all Java projects in the workspace.
- ▶ **Packages view**
This view shows the Java packages within the selected project.
- ▶ **Types view**
This view shows the types defined within the selected package.
- ▶ **Members view**
This view shows the members of the selected type.

These views are synchronized so that changing the selection in one view will update the available options in other views.

4.3.8 Java EE perspective

The Java EE perspective (Figure 4-16) includes views that you can use when developing resources for enterprise applications, Enterprise JavaBeans (EJB) modules, application client modules, and connector projects or modules.

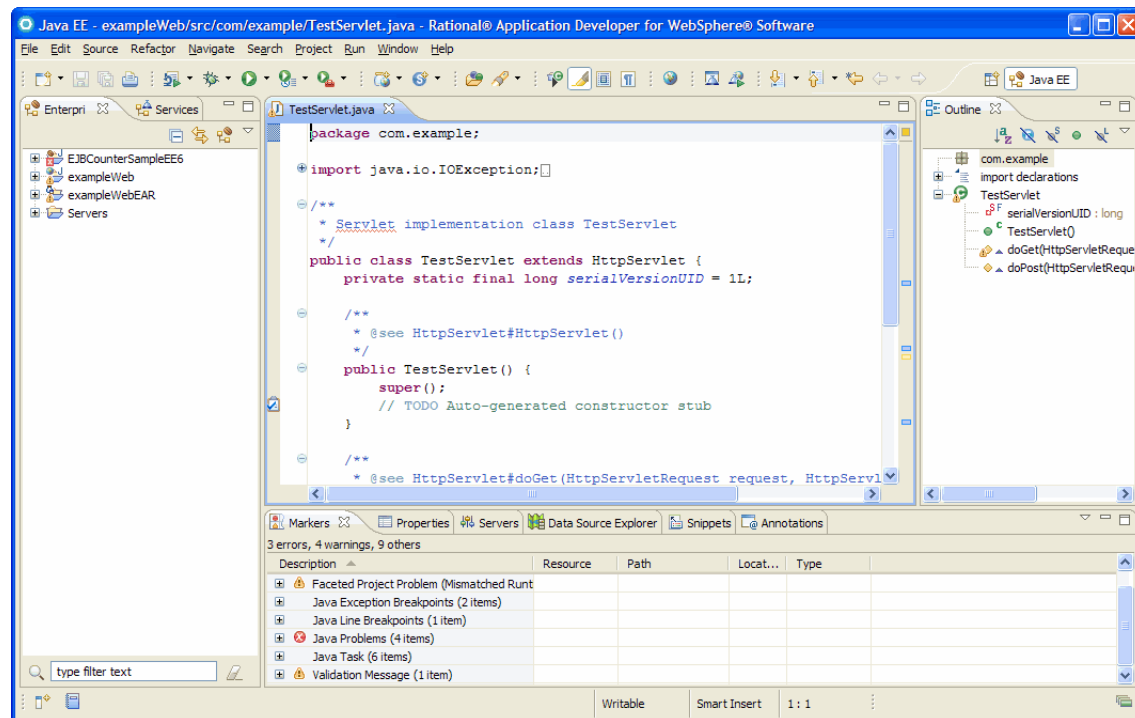


Figure 4-16 Java EE perspective

The Java EE perspective contains the following views that are typically used when developing Java EE applications:

- ▶ Enterprise Explorer view

This view provides an integrated view of your projects and their artifacts related to Java EE development. You can show or hide your projects based on working sets. This view shows navigable models of Java EE deployment descriptors, Java artifacts (source folders, packages, and classes), navigable models of the available web services, and specialized views of web modules to simplify the development of dynamic web applications. In addition, EJB database mapping and the configuration of projects for a Java EE application server are made readily available.

- ▶ Snippets view

The Snippets view lets you catalog and organize reusable programming objects, such as web services, EJB, and JavaServer Pages (JSP) code snippets. The view can be extended based on additional objects that you define and include. The available snippets are arranged in *drawers*. The drawers can be customized by right-clicking a drawer and selecting **Customize**.

- ▶ Properties view

This view provides a tabular view of the properties and associated values of objects in files that you have open in an editor. The format of this view depends on the active editor or view and its selection.

- ▶ Markers view

Similar to the Tasks view, it shows a combination of errors, warnings, and tasks together with other markers, such as breakpoints.

- ▶ Service view

This view lists all the web services in the workspace and categorizes them according to their underlying implementation (Java API for XML-based Remote Procedure Call (JAX-RPC), Java API for XML Web Services (JAX-WS), and RPC Adapter).

The Outline, Servers, Problems, Annotations, and Data Source Explorer views are also relevant to the Java EE perspective. We discussed these views in previous sections of this chapter.

For more details about using the Java EE perspective, see Chapter 12, “Developing Enterprise JavaBeans (EJB) applications” on page 577.

4.3.9 Java Type Hierarchy perspective

The Java Type Hierarchy perspective is for Java developers to explore which classes inherit from each other. You can open this perspective on types, compilation units, packages, projects, or source folders. This perspective consists of the Hierarchy view and an editor.

The Hierarchy view shows only an information message until you select a type.

To display the type hierarchy, select a type (for example, in the Outline view or in the editor) and select **Open Type Hierarchy**. Alternatively, you can drag and drop an element (for example, a project, package, or type) onto this view.

To open a type in the Hierarchy view, open the context menu for a Java class in any view or editor (for example, the main source code editor) and select **Open Type Hierarchy**. Figure 4-17 shows the Hierarchy view of the `Credit` class from Chapter 7, “Developing Java applications” on page 229.

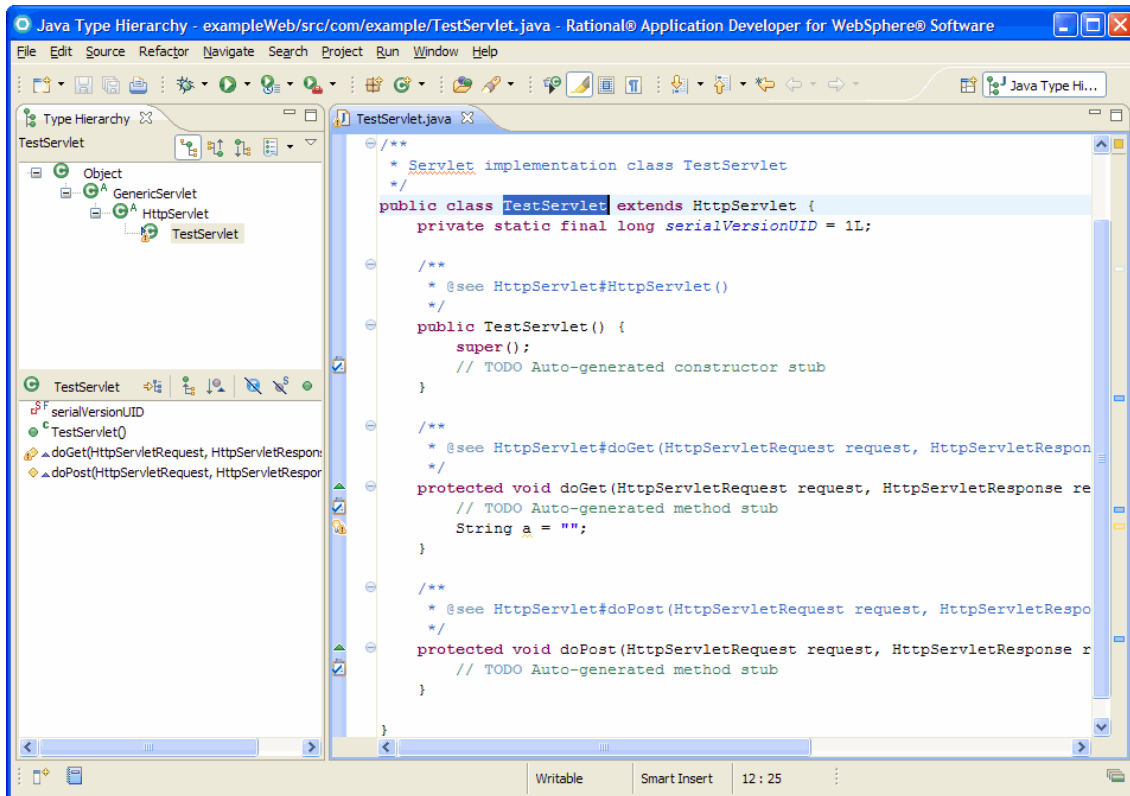


Figure 4-17 Java Type Hierarchy perspective with Hierarchy view

Although the Hierarchy view is also present in the Java perspective and the Java Type perspective only contains two views, it is useful because it provides a way for developers to explore and understand complex object hierarchies without the clutter of other information.

4.3.10 JavaScript perspective

The JavaScript perspective (Figure 4-18) is mainly used in coding, exploring, and documenting JavaScript.

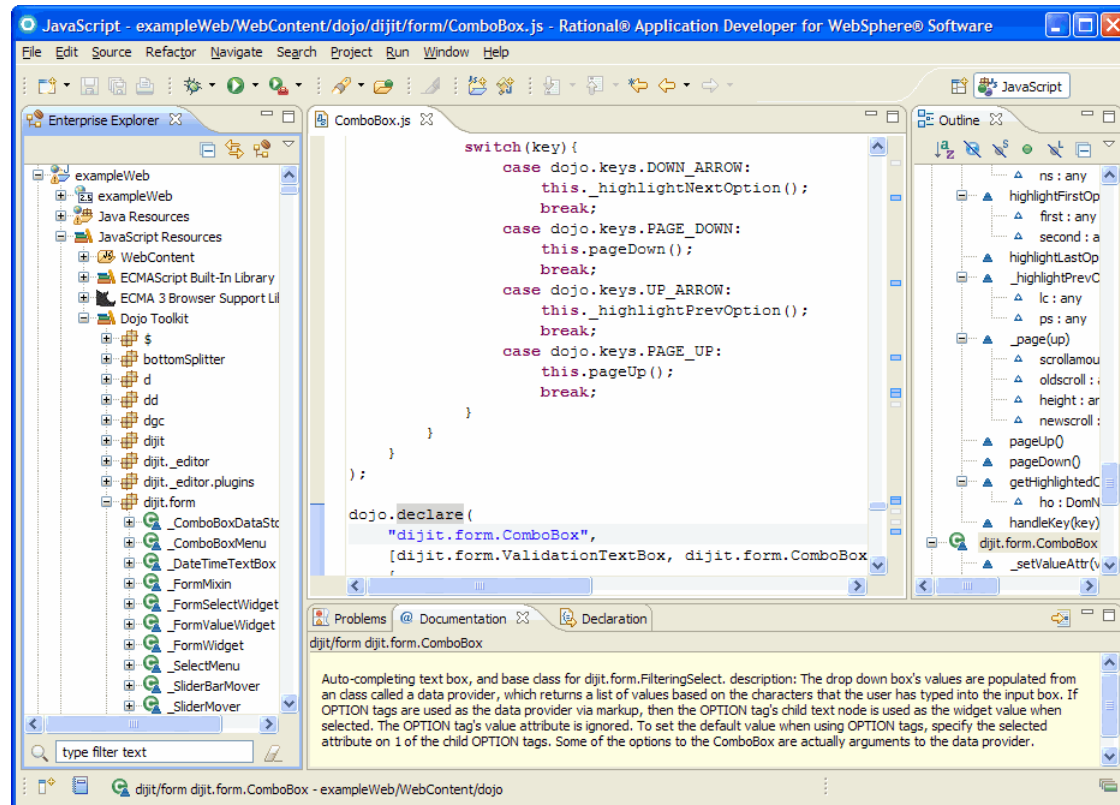


Figure 4-18 JavaScript perspective

This perspective has the following key views and editors:

- ▶ JavaScript Editor

Rational Application Developer includes this editor for working with JavaScript. It synchronizes with the Enterprise Explorer and Outline views and contains a number of context menu options to help navigate between JavaScript Type definitions.

- ▶ Documentation view

This view shows the JavaScript documentation for the selected JavaScript element in the Editor view or in the Outline view.

The Enterprise Explorer, Outline, and Declaration views are also displayed in this perspective. We discussed these views previously in this chapter.

For more details about working with JavaScript, see Chapter 20, “Developing web applications using Web 2.0” on page 1097.

4.3.11 JPA perspective

With the Java Persistence API (JPA) perspective (Figure 4-19), you can manage relational data in Java applications by using the Java Persistence API. You can take advantage of new capabilities, such as defining and editing object-relational mappings for EJB 3.0 JPA entities and adding JPA support to a plain Java project.

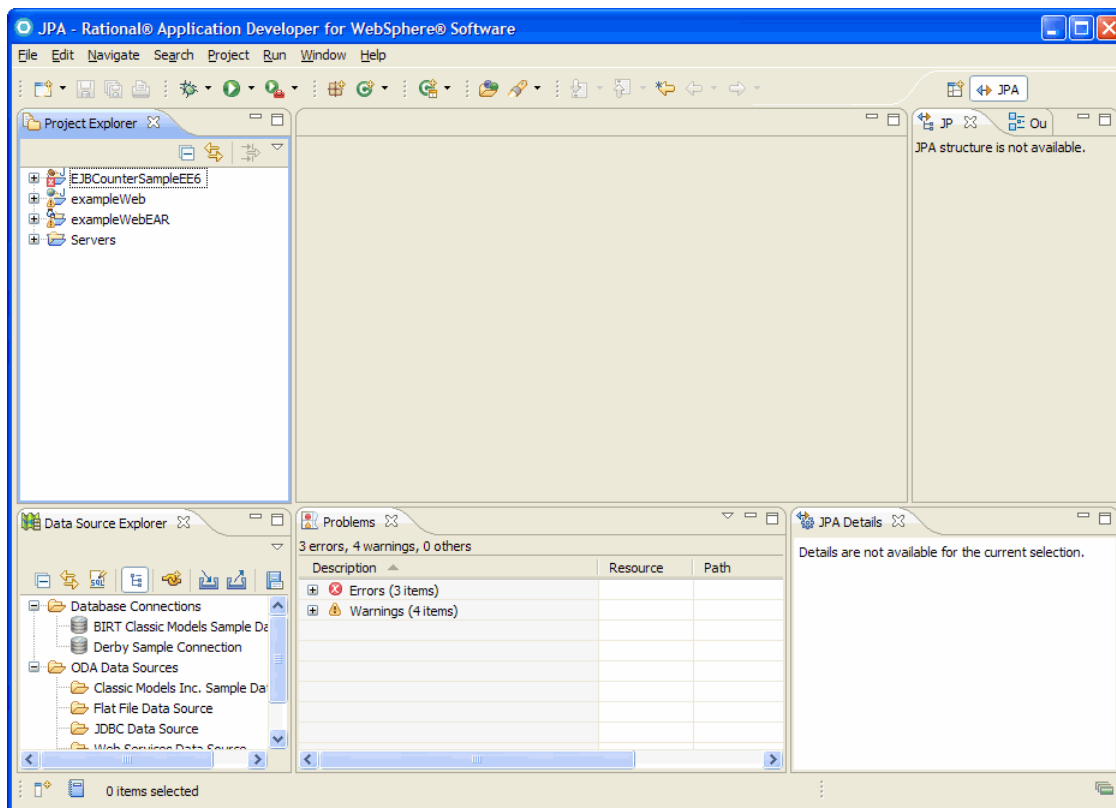


Figure 4-19 JPA perspective

The JPA perspective has the following key views:

- ▶ JPA Structure view

This view shows an outline of the structure (its attributes and mappings) of the entity that is currently selected or open in the editor.

- ▶ JPA Details view

The JPA Details view (Figure 4-20) shows the persistence information for the currently selected entity and various tabs, depending on whether the selection is on entity, attribute, or `orm.xml`.

You can work with JPA properties in either the JPA Details view or the Annotations view, so that you do not need to keep both views open at once. For clarity, the Annotations view distinguishes between implied and explicit annotation attributes.

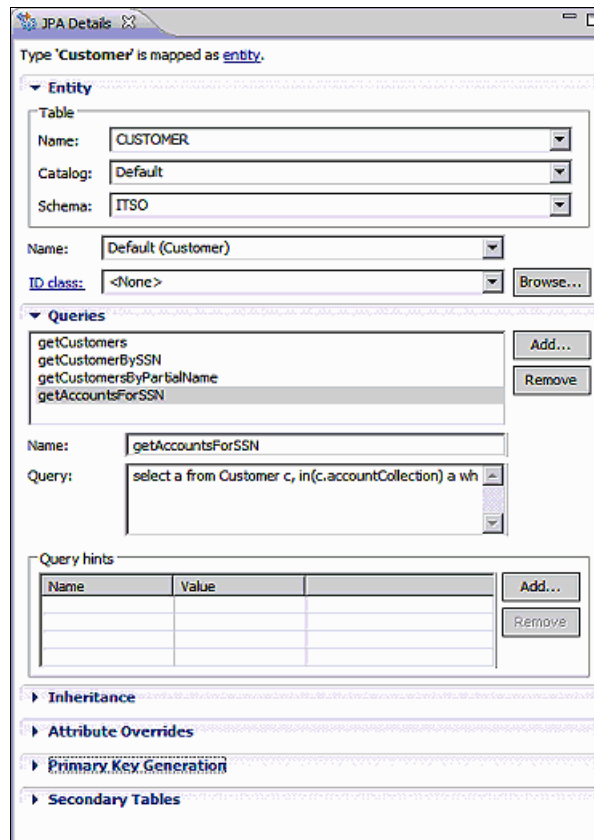


Figure 4-20 JPA Details view

For more details about working with JPA, see Chapter 10, “Persistence using the Java Persistence API” on page 443.

4.3.12 Modeling perspective

Rational Application Developer provides facilities to allow architects and software designers to create Unified Modeling Language (UML) diagrams, including class diagrams, sequence diagrams, and topic diagrams. These diagrams need to be built in the Modeling perspective, which includes views and commands to make the design process easier.

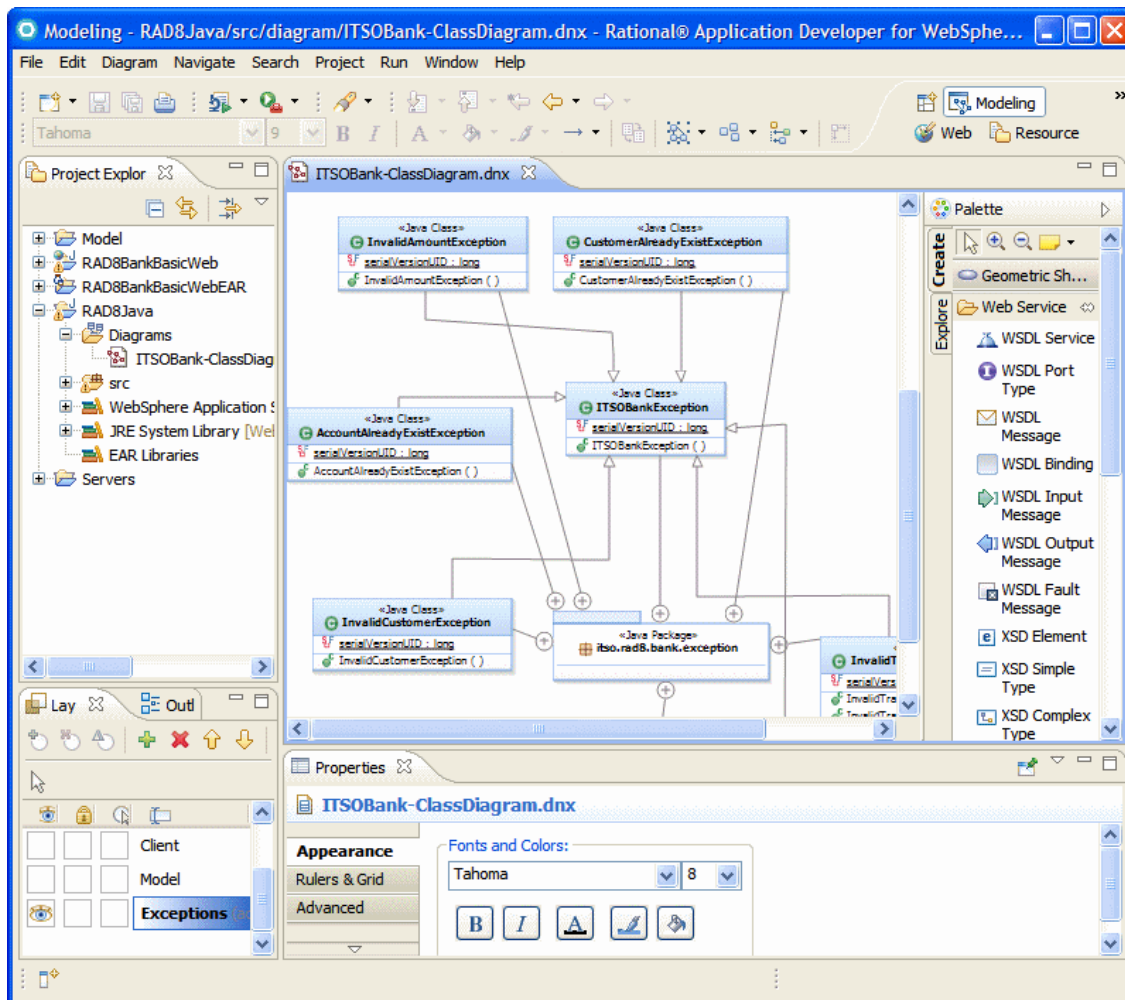


Figure 4-21 Modeling perspective

The Modeling perspective (Figure 4-21 on page 123) contains the Layers view. In the Layers view, each element in a UML diagram can be assigned to a *layer*, which is a grouping of UML elements on the diagram. When viewing a UML layer, it is possible to filter each layer so that the diagram only shows the pertinent elements.

The Project Explorer, Outline, Properties, and Palette views are also relevant to the Modeling perspective. We discussed these views in previous sections of this chapter.

For more details about working with UML, see Chapter 6, “Unified Modeling Language” on page 173.

4.3.13 Plug-in Development perspective

The ability to write extra features and plug-ins is an important part of the philosophy of the Eclipse framework. Using this perspective (Figure 4-22 on page 125), you can develop your own Rational Application Developer or Eclipse tools.

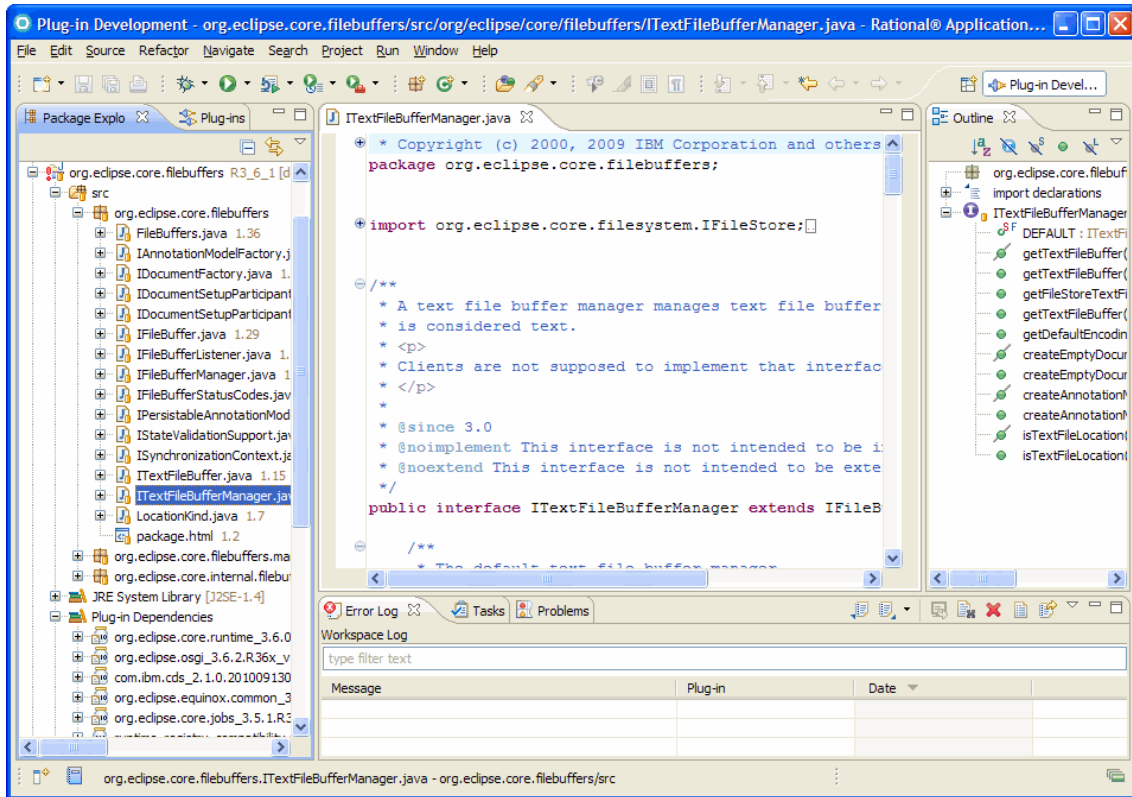


Figure 4-22 Plug-in Development perspective

The Plug-in Development perspective (Figure 4-22) includes the following views:

- ▶ Plug-ins view
 - This view shows the combined list of workspace and external plug-ins.
- ▶ Error Log view
 - This view shows the error log for the software development platform, allowing a plug-in developer to diagnose problems with plug-in code.

This perspective includes the Package Explorer, Outline, Tasks, and Problems views, which we described earlier in this chapter.

To learn about plug-in development for Rational Application Developer or Eclipse, see the IBM Redbooks publication *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework*, SG24-6302, or *The Java Developer's Guide to Eclipse-Second Edition*, by D'Anjou et al., which you can download from the following website:

<http://jdg2e.com>

4.3.14 Profiling and Logging perspective

The Profiling and Logging perspective (Figure 4-23) provides several views for working with logs and for profiling applications. One of the key views is the *Profiling Monitor* view. This view shows the process that can be controlled by the profiling features of Rational Application Developer. Performance and statistical data can be collected from processes using this feature and displayed in various specialized views and editors.

The screenshot shows the 'Profiling and Logging' perspective in Rational Application Developer. The main window is titled 'Execution Statistics - itso.rad8.bank.client.BankClient at RL3EPD16 [PID: 5272]'. It displays a 'Session summary' table with the following data:

Package	<Base Time (seconds)	Average Base T...	Cumulative Tim...	Calls
itso.rad8.bank.model	0.704773	0.001456	0.704773	484
Debit	0.428286	0.071381	0.428484	6
toString() java.lang.Strin	0.428111	0.107028	0.428221	4
process(java.math.BigDe	0.000152	0.000152	0.000176	1
Debit(java.math.BigDecin	0.000022	0.000022	0.000087	1
Account	0.147181	0.000651	0.700892	226
processTransaction(java.	0.105331	0.035110	0.230812	3
toString() java.lang.Strin	0.038932	0.001770	0.468721	22
getTransactions() java.u	0.001115	0.000021	0.001115	53
getAccountNumber() jav	0.000659	0.000010	0.000659	64
Account(java.lann.Strin	0.000534	0.000038	0.000864	14

Below the table, there are tabs for 'Session summary', 'Execution Statistics', 'Call Tree', 'Method Invocation Details', and 'Method Invocation'. At the bottom, a 'Console' view shows the application output:

```
<terminated> BankClient [Java Application] java.exe (12 October 2010 12:09:16 PM)

Customer: xxx-xx-xxxx Mr Napoli Juan
[Account 1: --> Current balance: $12500.00
Transactions:
1. itso.rad8.bank.model.Credit@3a873a87
```

Figure 4-23 Profiling perspective

In addition, several editors are available for viewing the results of profiling, which depends on which information was selected for monitoring. For example, the Execution Statistics view from Figure 4-23 on page 126 shows the number of calls and amount of time spent in each method of the Debit and Account classes. For more details about these views and the techniques required to use them, see Chapter 27, “Profiling applications” on page 1419.

4.3.15 Report Design perspective

The Report Design perspective (Figure 4-24) features the Data Explorer, Resource Explorer, and Property Editor views.

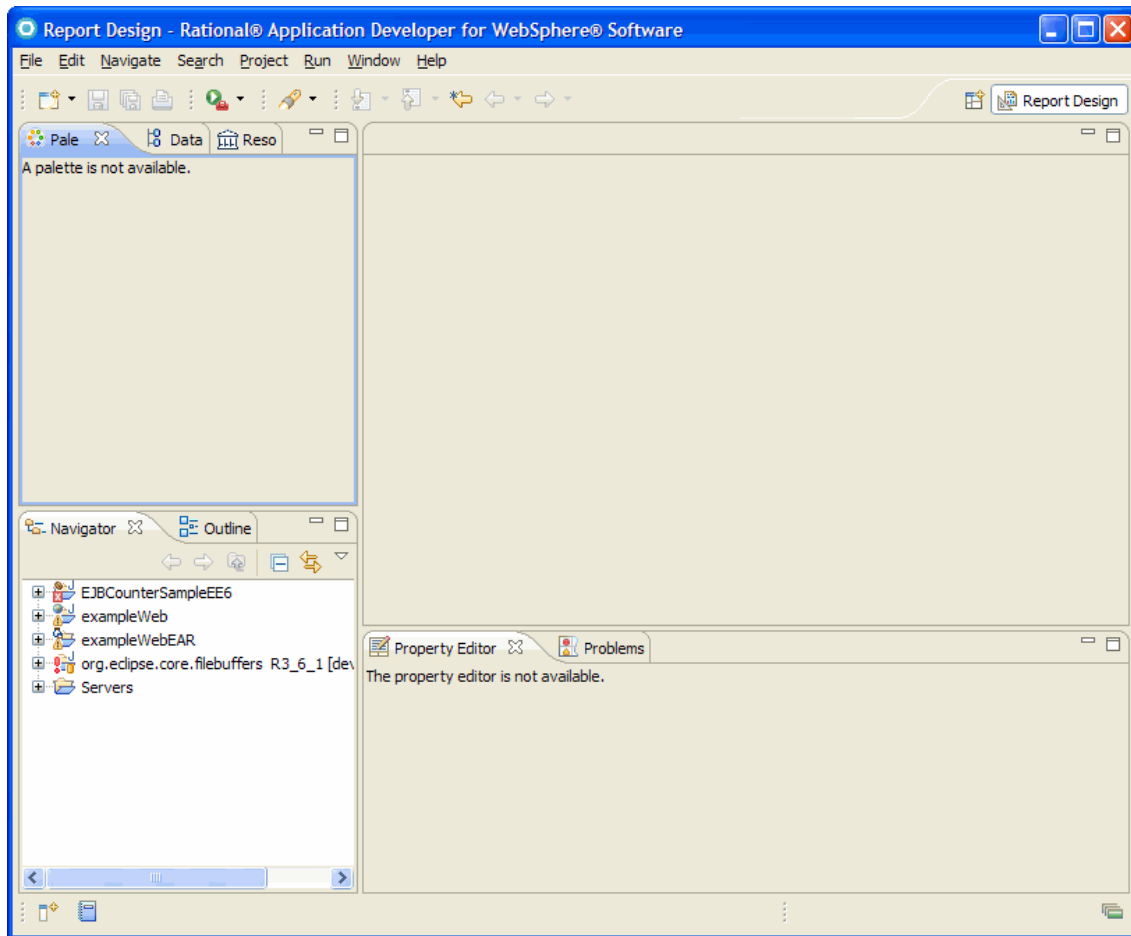


Figure 4-24 Report Design perspective

This perspective has the following key views:

- ▶ Data Explorer view

The Data Explorer view shows the data sources, query result sets, and other elements that are used by a report.

- ▶ Resource Explorer view

The Resource Explorer view shows reusable objects and shared content that can be included in reports.

- ▶ Property Editor view

The Property Editor view shows commonly used properties in a designed layout. The standard Properties view, although not shown by default, shows all available properties.

4.3.16 Resource perspective

The Resource perspective is a simple perspective (Figure 4-25 on page 129). By default, this perspective contains only the Navigator view, Outline view, Tasks view, and an editor area. You can use this perspective to view the underlying files and folders present for a project without any overlaid information or extraneous views. All views in this perspective are available in other perspectives and are described in previous sections in this chapter.

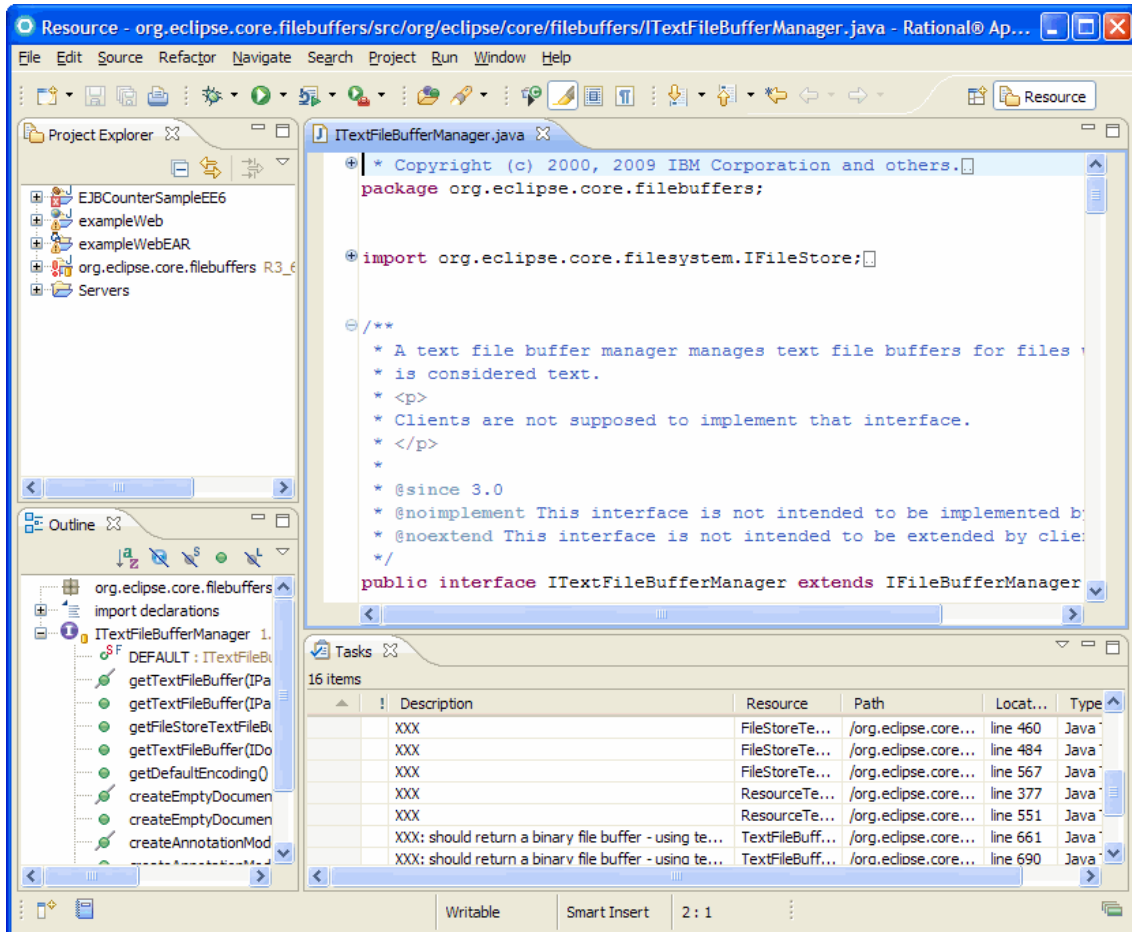


Figure 4-25 Resource perspective

4.3.17 Team Synchronizing perspective

The Team Synchronizing perspective enables the user to synchronize the resources in the workspace with resources held on an SCM repository system. This perspective is used with CVS and ClearCase repositories, plus any other source code repository that might run as an additional plug-in to Rational Application Developer.

Figure 4-26 shows a typical layout when working in the Team Synchronizing perspective.

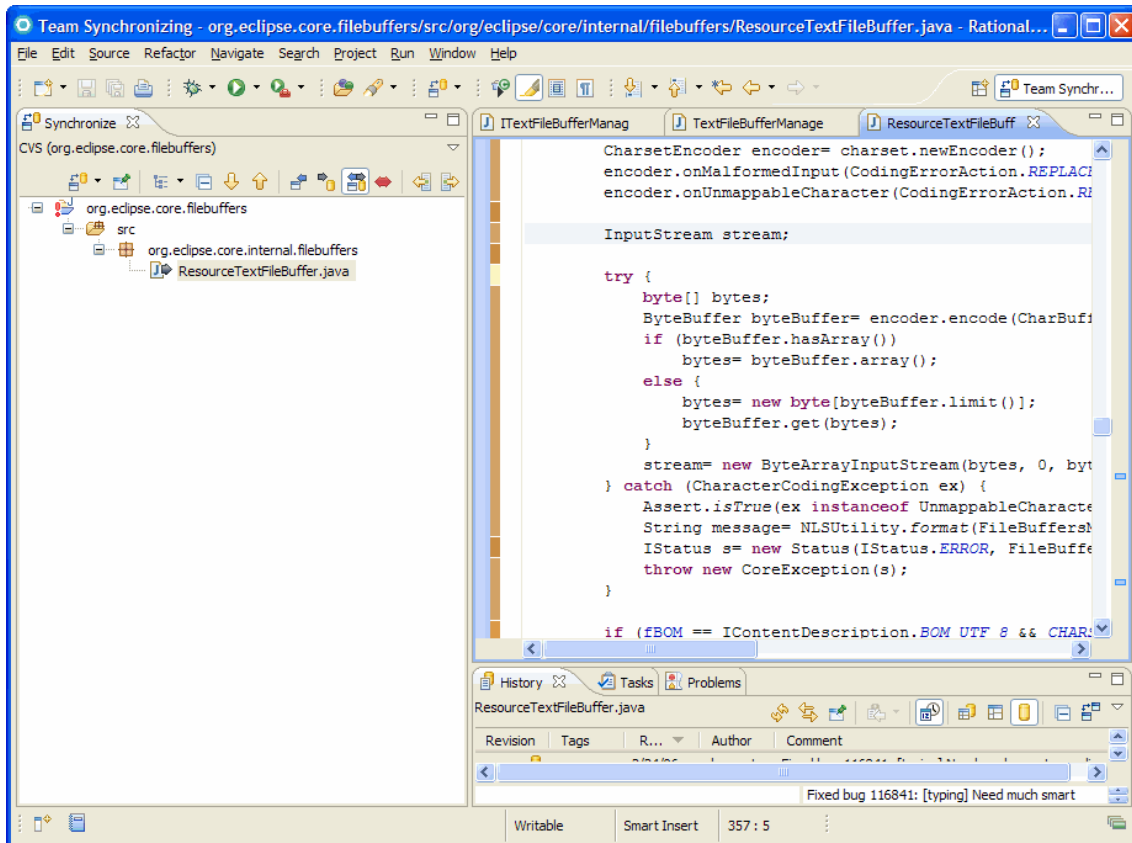


Figure 4-26 Synchronizing resources using the Team Synchronizing perspective

The following views are important when working in this perspective:

- ▶ Synchronize view

For any resource that is under source control, you can select **Team** → **Synchronize**, which prompts you to move to the Team Synchronizing perspective and show the Synchronize view. It shows the list of synchronization items that result from the analysis of the differences between the local and repository versions of your projects. Double-clicking an item opens the comparison view to help you in completing the synchronization.

- ▶ Comparison editor

This editor is displayed in the main editor area and shows a line-by-line comparison of two revisions of the same source code.

Also present in the Team Synchronizing perspective is the History view to show the revision history of a given resource file and the Tasks and Problems view. For more details about these views and how to use them, see Chapter 29, “Concurrent Versions System (CVS) integration” on page 1533.

4.3.18 Test perspective

The Test perspective (Figure 4-27) provides a framework for defining and executing test cases and test suites. The focus here is on running the tests and examining the results rather than building the JUnit tests themselves. Building JUnit tests often involves writing complex Java code and is best done in the Java perspective.

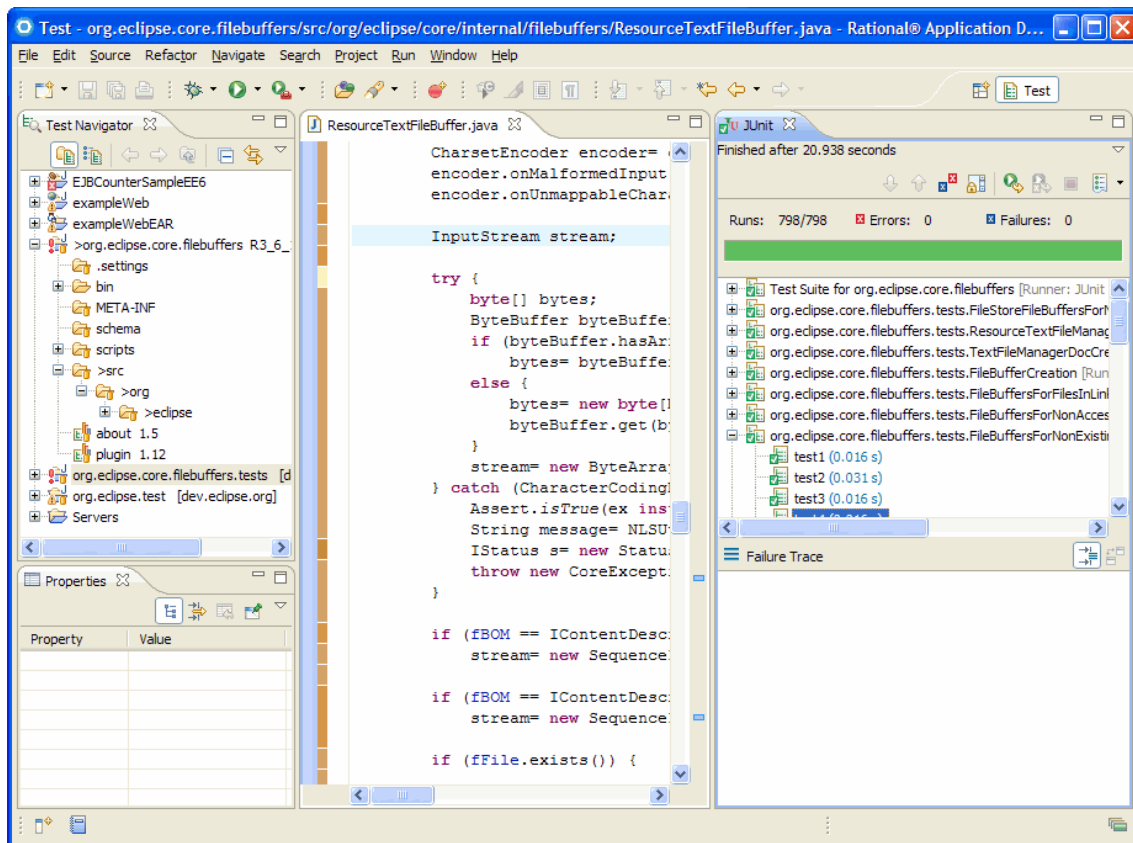




Figure 4-27 Test perspective

This perspective has the following key views:

▶ Test Navigator view

This view is the main navigator view for browsing and editing test suites and reviewing test results. It has two main options to structure the display of these resources:

- The Show the resource test navigator () option shows the resources based on the file system, with the test suites displayed at the bottom.
- The Show the logical test navigator () option shows the resources arranged by test suites, source code, and test results.

▶ Test Log view

If the user clicks a test result, this view is shown in the main editor area showing the date, time, and result of the test.

▶ Test editor

This editor shows a summary of a test suite and its contained tests.

The Tasks, Properties, and Outline views are also present and useful when working in the Test perspective. We explain these views in previous sections of this chapter.

For more information about component testing, see Chapter 26, “Testing using JUnit” on page 1365.

4.3.19 Web perspective

Web developers can use the web to build and edit web resources, such as Java servlets, JSP files, HTML pages, style sheets, and images, as well as the deployment descriptor `web.xml`. Figure 4-28 on page 133 shows a typical layout when developing in this perspective.

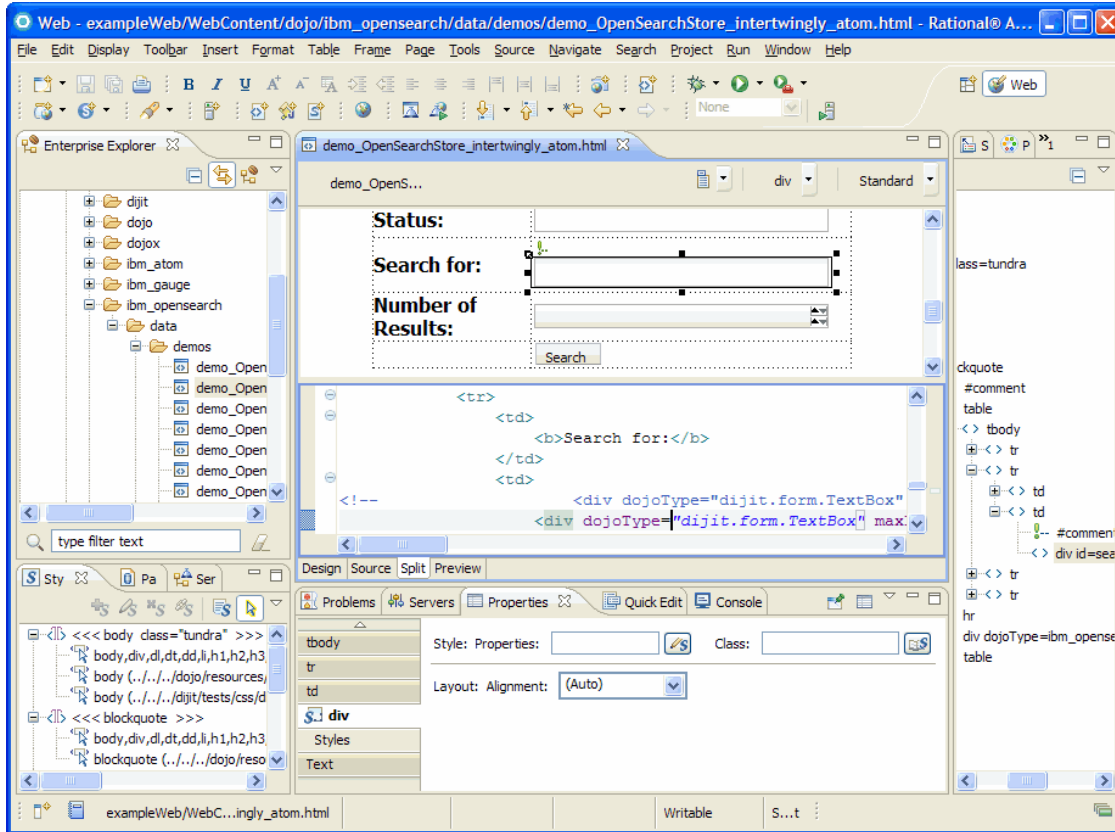


Figure 4-28 Web perspective

The Web perspective contains the following views and editors:

- ▶ Page Designer (editor)

With Page Designer, you can edit HTML files, JSP files, and their embedded Java and JavaScript sources. It provides four tabs offering separate ways for you to work with the file that you are editing. This editor is synchronized with the Outline and Properties views, so that the selected HTML or JSP element is always displayed in these views.

- Design

The Design page of Page Designer is the “what you see is what you get” or WYSIWYG mode for editing HTML and JSP files. As you edit in the Design page, your work reflects the layout and style of the web pages you build without the added complexity of source tagging syntax, navigation, and debugging. Although all tasks can also be done in the Source page,

the Design view allows most operations to be done more efficiently and without requiring a detailed knowledge of HTML syntax.

- Source

From the Source page, you can view and work with a file's source code directly.

- Split

The Split page combines the Source page and either the Design page or the Preview page in a split view. Changes that you make in one part of the split page can immediately be seen in the other part of the split page. You can split the page horizontally or vertically.

- Preview

The Preview page shows how the current page is likely to look when viewed in a web browser. JSP shown in this view contain only static HTML output.

- ▶ Web Diagram Editor

Use the Web Diagram Editor (Figure 4-29 on page 135) to design and construct the logic of a web application. From within the Web Diagram Editor, you can configure your web application by creating a navigation structure, adding data to pages, and creating actions and connections. You can use the palette to add visual representations called *nodes* for web pages, web projects, connections, and JavaServer Faces (JSF) and Struts resources (if these facets are added to your web project).

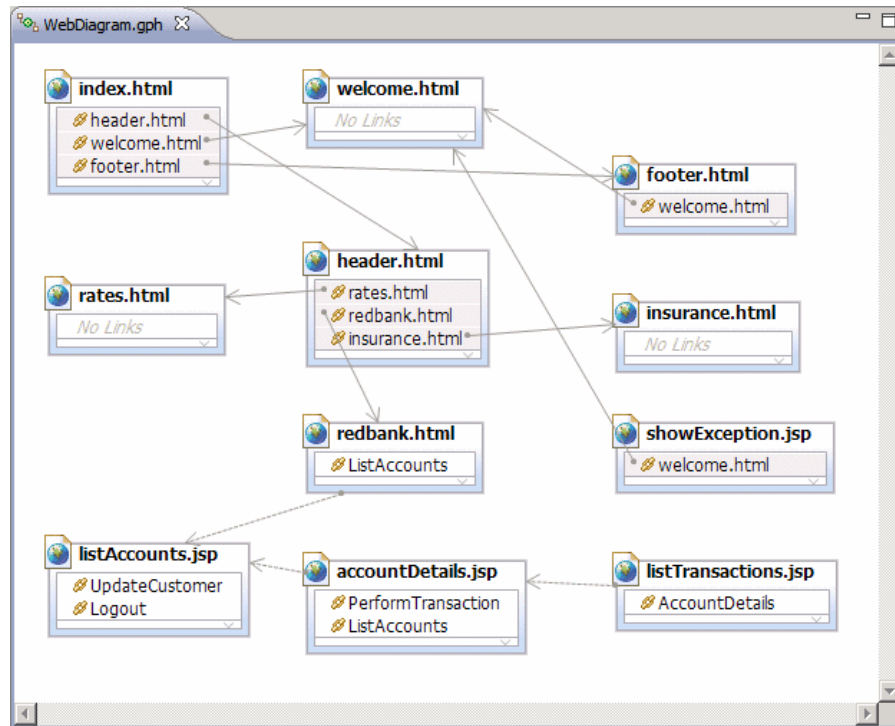


Figure 4-29 Web Diagram Editor

► Page Data view

With this view, you can manage data from a variety of sources, such as session EJBs, JavaBeans, Service Data Objects, JPA objects, and web services, which can be configured and dropped onto a JSP page.

► Services view

This view lists all of the services available to all of your projects, including web services and RPC Adapter services. This view does not require you to open a web page in the editor to view the services that are specific to that particular page.

► Styles view

This view provides guided editing for cascading style sheets and individual style definitions for HTML elements.

► Thumbnails view

This view shows thumbnails of the images in the selected project, folder, or file. This view is synchronized with the Enterprise Explorer view and will show thumbnail versions of all the files in the selected directory. There is also a filter

option so that users can restrict the files shown to images, HTML files, and so forth.

- ▶ Quick Edit view

With this view, you can edit small bits of JavaScript code, including adding and editing actions assigned to tags. This view is synchronized with the element selected in the Page Designer. You can also drag items from the Snippets view to the Quick Edit view.

- ▶ Palette view

This view contains expandable drawers of drag-and-drop objects. With this view, you can drag objects, such as tables or form buttons, to the Design or Source page of the Page Designer.

The Enterprise Explorer, Outline, Properties, Servers, Console, Problems, and Snippets views are also present in the Web perspective and have already been discussed in this chapter.

For more information about developing JSP and other web application components in the Web perspective, see Chapter 18, “Developing web applications using JavaServer Pages (JSP) and servlets” on page 981.

4.3.20 XML perspective

Rational Application Developer contains special tools to facilitate the process of developing XML-based applications. In particular, there are tools to help define and verify XML schemas. Figure 4-30 on page 137 shows a typical screen capture of work in the XML perspective.

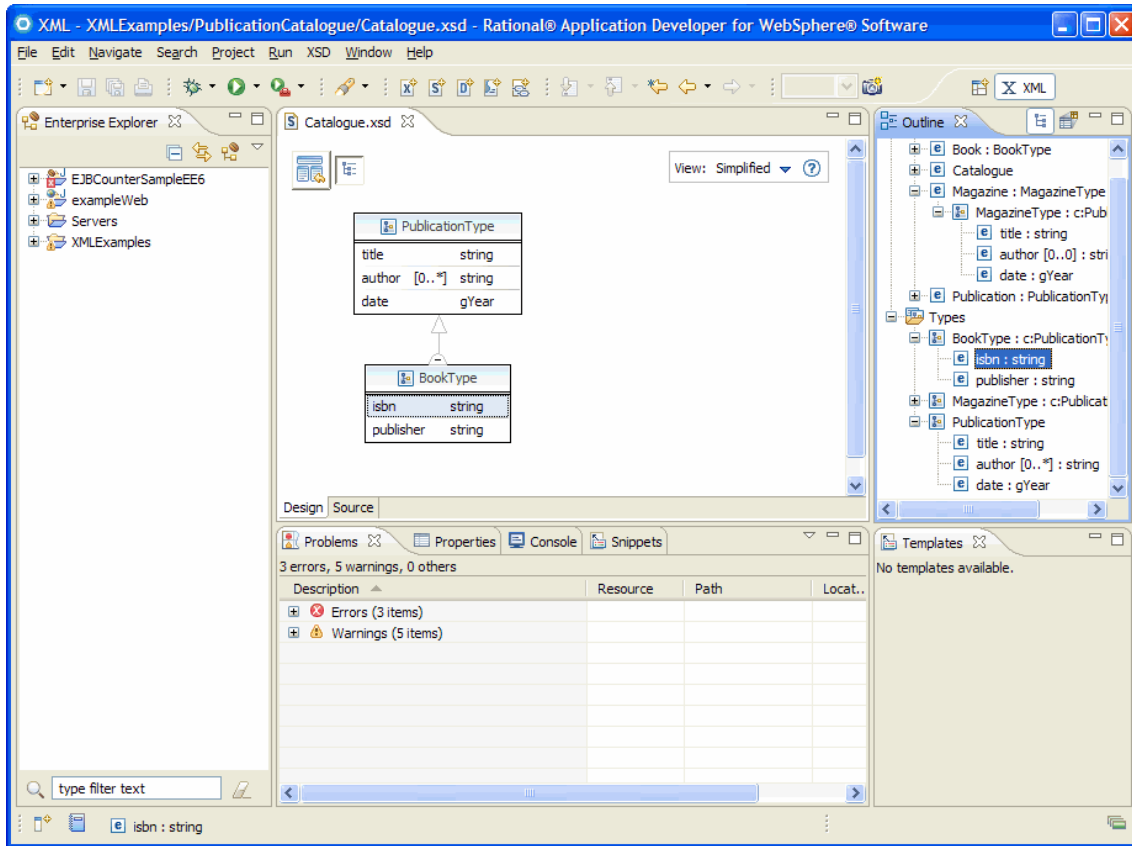


Figure 4-30 XML perspective

The following views are the main views of this perspective:

- ▶ XML Editor

Use this editor to work with XML files. The Design tab includes several menu options to speed up the process of simple functions, such as adding elements and editing namespaces. The Source tab allows you to work with the underlying source code.

- ▶ XML Schema Editor

Use this editor to work with XML Schema Definition Language (XSD) or XML schema definitions. It includes a graphical view of the schema, which you can zoom into or zoom out of, and the ability to export the schema view as an image.

The Enterprise Explorer, Outline, Properties, Problems, Snippets, and Templates views are also present in the XML perspective and have already been discussed in this chapter.

For more information about developing applications and using XML and XSD files, see Chapter 8, “Developing XML applications” on page 331.

4.3.21 Progress view

The Progress view shows the progress of operations running in the background.

As Rational Application Developer performs a task that takes a substantial amount of time, a prompt might appear during its execution (Figure 4-31).

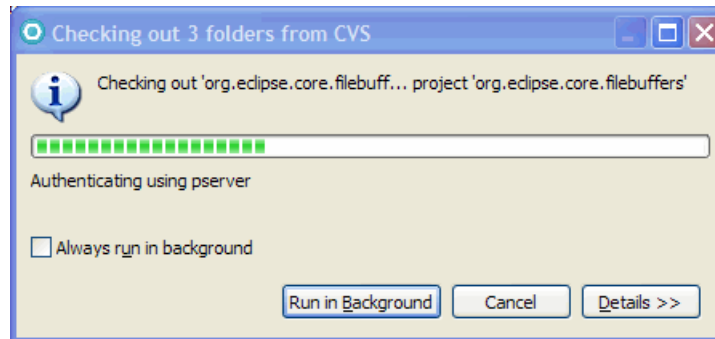


Figure 4-31 Run in background option

The user can either wait until the dialog is dismissed by the completed operation or click **Run in Background** and have the task continue in the background. If the second option is selected, the operation might take longer to complete overall, but the user is free to carry out other tasks in the meantime. Examples of tasks that might be worth running in the background are publishing and launching an enterprise application, checking a large project in to or out of CVS, or rebuilding a complex set of projects. If you clicked “Run in Background”, you can use the Progress view to review the status of the running task by clicking the icon in the lower right of the workspace. This icon is only displayed when processes are running in the background.

Certain processes do not prompt the user with a window and instead automatically run in the background when they are initiated. In these cases, the Progress view can be accessed in the same way. By default, the progress view appears minimized and is represented as an icon in the lower right area of the workbench (Figure 4-32 on page 139).

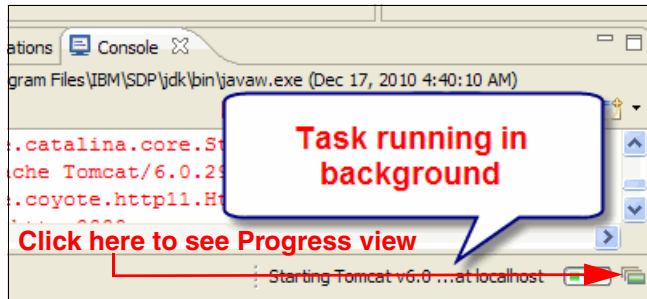


Figure 4-32 Process information in status bar

Clicking the symbol will show the status of the operations in the Progress view. Double-clicking the neighboring status text will also open the Progress view.

If the user becomes concerned about the time that the deployment process is taking, the user can open the Progress view, review the processes that are running, and if necessary, stop the processes (Figure 4-33).

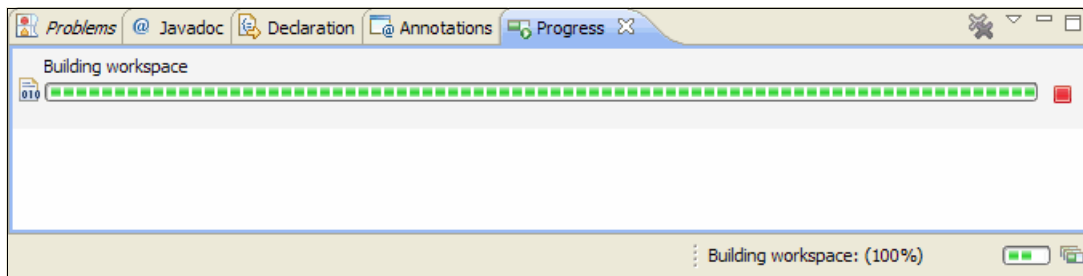


Figure 4-33 Progress view

4.4 Summary

In this chapter, we described the perspectives available within Rational Application Developer and the associated main views. In the next parts of this book, we demonstrate the use of most of these perspectives for various development scenarios.



Projects

In this chapter, we provide an overview of the types of projects that can be created with Rational Application Developer, and the major features of each project type.

Because many of the available project types are used when constructing Java Enterprise Edition 6 (Java EE 6) applications, we start with a review of the major features of the Java EE 6 platform, including the packaging of project code for deployment to an application server.

We describe techniques for the manipulation of projects, including project creation and deletion, followed by a section listing the project wizards that are provided by Rational Application Developer for the creation of new projects. Finally, we discuss the sample projects that are provided.

The chapter is organized into the following sections:

- ▶ Java Enterprise Edition 6
- ▶ Java EE 6 project types
- ▶ Project basics
- ▶ Project wizards
- ▶ Sample projects
- ▶ Summary

5.1 Java Enterprise Edition 6

The Java Enterprise Edition (Java EE) platform is used to host enterprise applications, ensuring that they can be highly available, reliable, scalable, and secure. Java EE 6 is the latest version of the Java EE platform and is supported by Rational Application Developer when the target run time selected is WebSphere Application Server V8 Beta. Previous versions of Java EE, such as Java EE 5, are also supported by WebSphere Application Server V8 Beta. If WebSphere Application Server V7 is chosen as the target run time, only support for the Java EE platforms up to Java EE 5 is available.

The Java EE 6 specification, along with many other resources relating to Java EE 6, are available at the following web address:

<http://www.oracle.com/technetwork/java/javaee/overview/index.html>

The Java EE architecture consists of a set of containers, each of which is a runtime environment that hosts specific Java EE components and provides services to those components. The details of the services provided by each container are documented in the Java EE 6 specification document available at the following web address:

<http://jcp.org/aboutJava/communityprocess/final/jsr316/index.html>

The Java EE architecture includes four containers:

- ▶ The Enterprise JavaBeans (EJB) container. This container hosts EJB components, which are typically used to provide business logic functionality with full transactional support. This container runs on the application server.
- ▶ The Web container. This container hosts web components, such as servlets and JavaServer Pages (JSP), which are executed in response to HTTP requests from a web client application, such as a web browser. This container runs on the application server.
- ▶ The Application Client container. This container hosts standard Java applications, with or without a GUI, and provides the services required for those applications to access enterprise components in an EJB container. This container runs on a client machine.
- ▶ The Applet container. This container hosts Java applets, which are GUI applications that are typically presented by a web browser. The applet container runs on a client machine under the control of a web browser.

Figure 5-1 on page 143 shows the Java EE architecture containers. The diagram includes a database that is typically used for the persistence of enterprise application data. It is not necessary to employ all of the containers in a specific enterprise application. In certain enterprise applications, only the Web container

is employed. In this case, all business logic and persistence functionality execute in the Web container along with the code that presents the user interface. In other enterprise applications, only the Web container and EJB container are employed. In this case, the user interface is presented by components in the Web container with all business logic and persistence functionality delegated to the EJB container and the EJB components that it contains. One new feature of Java EE, introduced in Java EE 6, is support for EJB components in the Web container. This feature gives developers a new option for the implementation of business logic and persistence functionality when an enterprise application with just a Web container is required.

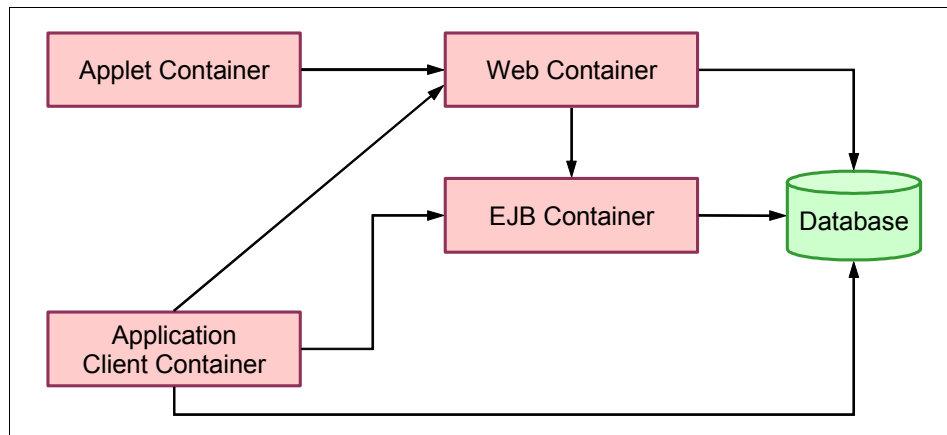


Figure 5-1 Java EE architecture containers

A Java EE enterprise application is assembled from one or more Java EE modules, and each Java EE module contains one or more enterprise application components. An optional deployment descriptor, which describes the module and the components that it contains, can also be included in the module. The following sections provide a summary of the purpose that is served by each of the modules and the types of components typically contained in the module. Subsequent sections describe the types of projects that are available in Rational Application Developer and that are used to create each module.

Figure 5-2 shows a high-level view of the module structure of a Java EE enterprise application.

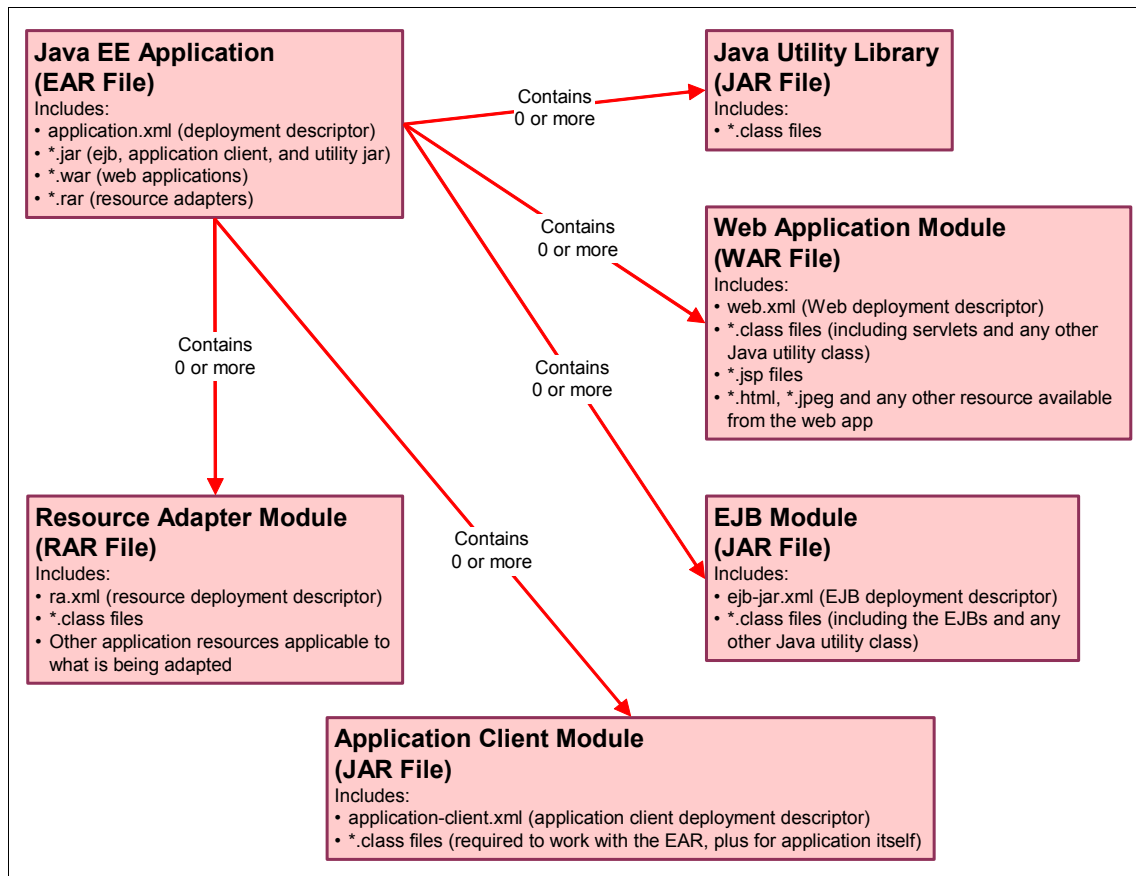


Figure 5-2 Java EE 5 module structure

5.1.1 Enterprise application modules

Enterprise application modules contain one or more of the other types of Java EE modules. They act as the highest-level enterprise application packaging unit. The modules in an enterprise application are deployed as a unit to the WebSphere Application Server. Enterprise application modules are packaged as enterprise archive (EAR) files with the .ear extension. EAR files are standard Java archive files that have a defined directory structure. An optional deployment descriptor file called `application.xml` can be included.

An enterprise application module can include zero or more of the following modules:

Web modules	WAR files with the .war extension
EJB modules	EJB JAR files with the .jar extension
Application client modules	Application client JAR files with the .jar extension
Resource adapter modules	Resource adapter archive files with the .rar extension
Utility libraries	JAR files with the .jar extension, which are shared by all the other modules that are packaged in the EAR file

5.1.2 Web modules

Web modules contain all the components that are part of a specific web application. The following items are several of the components that might be included:

- ▶ Hypertext Markup Language (HTML) web page files
- ▶ Cascading Style Sheet (CSS) files
- ▶ JavaServer Pages (JSP) web page files
- ▶ Facelets
- ▶ Compiled Java servlet classes
- ▶ Other compiled Java classes
- ▶ Image files
- ▶ Portlets (portal applications)
- ▶ iWidgets

Web modules are packaged as WAR files with the .war extension. WAR files have a defined directory structure and include an optional deployment descriptor called `web.xml`, which contains the configuration information for the web module. Each web module has a defined context root that determines the URL required to access the components present in the web module. Web modules can be packaged in an EAR for deployment to an application server or can be deployed stand-alone. Because Java EE 6 allows EJB components to be packaged in a web module, the use of a stand-alone web module is now a more versatile option.

5.1.3 EJB modules

An EJB module contains EJB components. EJB modules are packaged as JAR files with the .jar extension. EJB JAR files have a defined directory structure and include an optional deployment descriptor called `ejb-jar.xml`, which

contains configuration information for the EJB module. Alternatively, the configuration can be defined using annotations in the Java classes and interfaces of the EJB components.

5.1.4 Application client modules

An application client module contains enterprise application client code. Application client modules are packaged as JAR files with the `.jar` extension. An application client module typically includes the classes and interfaces to allow a client application to access EJB components in an EJB module. Code in an application client module can also access components in a web module. The file has a defined directory structure and includes an optional deployment descriptor called `application-client.xml`.

5.1.5 Resource adapter modules

A resource adapter (RA) module contains resource adapters. Resource adapter modules are packaged as `.rar` files. Resource adapters provide access to back-end resources using services provided by the application server. Resource adapters are often provided by vendors of Enterprise Information Systems, such as SAP and PeopleSoft, to facilitate access from Java EE 6 applications.

Resource adapter modules can be installed as stand-alone modules within the application server, so that they can be shared by several enterprise applications. They can also be included in a specific EAR file, in which case they are only available to the modules contained within that EAR file. A `.rar` file has a defined directory structure and contains an optional deployment descriptor called `ra.xml`.

5.1.6 Java utility libraries

Java utility libraries can be included in a Java EE enterprise application so that all the modules included in the application can share the code that they contain. Java utility libraries are packaged as standard Java JAR files with the `.jar` extension.

5.2 Project basics

Within Rational Application Developer, projects are contained in a workspace. A workspace maintains everything needed by the developer for building and testing a project. A project must be present in a workspace before it can be accessed and used. Many types of projects can be created as required for a specific

application. Projects are typically created or imported using one of the wizards available in Rational Application Developer. See 5.4, “Project wizards” on page 162 for a list of available project wizards.

Unless otherwise specified, projects are stored in the Rational Application Developer workspace directory. A workspace is chosen when Rational Application Developer is started, although it is also possible to switch workspaces at a later time by selecting **File** → **Switch Workspace**.

5.2.1 Creating a new project

Development on a new application is usually started by creating one or more projects. Plan the required projects in advance and then use the relevant project wizards to create a skeleton set of projects for the application under construction. You can also open existing projects if they are to be used as part of the current application.

To create a new project using the Enterprise Application Project wizard, follow these steps:

1. Start this wizard by selecting **File** → **New** → **Project**.
2. In the New Project window (Figure 5-3 on page 148), select **Java EE** → **Enterprise Application Project** and click **Next**.

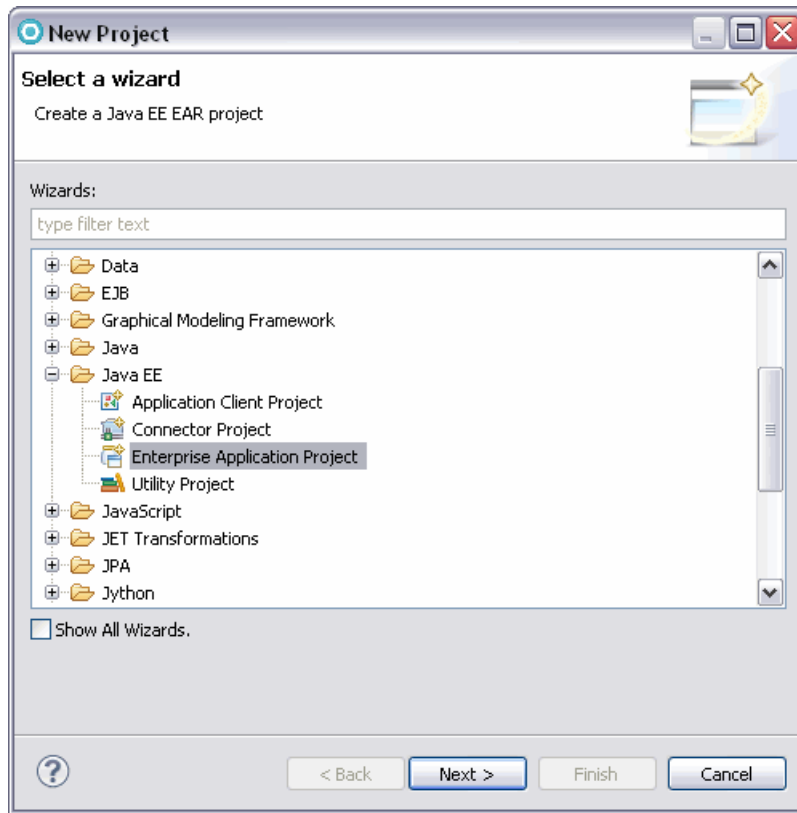


Figure 5-3 New Project: Select a wizard window with Enterprise Application Project selected

3. Figure 5-4 shows the New EAR Application Project: EAR Application Project window, which is the first window of the Enterprise Application Project wizard.

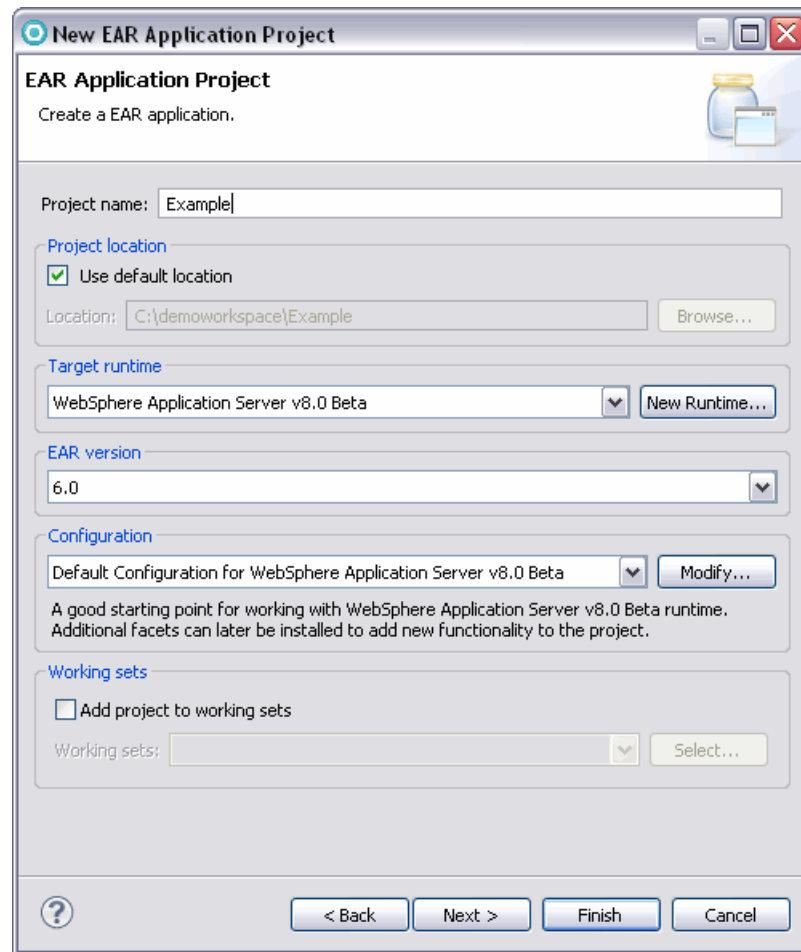


Figure 5-4 New EAR Application Project: EAR Application Project window

In this window, you can specify the following details:

- Project Name. In this field, you enter the project name, which is Example in this case.
- Project location. By default, projects are stored in a subdirectory that is created for the project in the workspace directory. By clearing the **Use default location** option, you can specify another location.
- Target Runtime. An enterprise application project is targeted to run on an Application Server. With this option, you can configure the target runtime

environment. In this case, WebSphere Application Server V8.0 Beta has been selected.

- **EAR Version.** An enterprise application can be created for a specific version of Java EE, such as 1.4, 5, or 6. In this case, we select EAR Version **6.0** from the list. The versions that are available depend on the Target Runtime selected. With WebSphere Application Server V8.0 Beta as the Target Runtime, versions up to the latest version, 6.0, are available.
- **Configuration.** This list contains the saved configurations that have been created for previous enterprise application projects. A configuration includes a specific set of facets and their specific versions. Make sure that all similar projects use the same configuration. An existing configuration can be chosen or <custom> can be selected. When <custom> is selected, you can specify a configuration. Click **Modify** to modify a configuration.

If you click **Modify**, the Project Facets window (Figure 5-5 on page 151) opens. In this window, you can customize the facets and versions of a facet that will be available in the new project. The set of facets selected is a project configuration and determines the capabilities supported by the project. The validity of facet selections is determined by the target run time selected, the module version selected, and the currently selected facets.

In this case, the EAR, WebSphere Application (Co-existence), and WebSphere (Extended) facets are selected by default, and the iWidgets, JavaScript, SCA, and WebSphere 8.0 Beta SCA facets are available for selection.

You can save a selected set of facets as a configuration so that it can be used for subsequent projects. Configurations can be created and saved that are known to be valid for a specific target run time and Java EE module version.

You can also use the Project Facets window when creating new Dynamic Web, EJB, and Connector projects, because they are Java EE project types, but the facets listed in each case are applicable to the type of project being created.

Several predefined configurations are available in Rational Application Developer, many of which are covered in 5.3.3, “Dynamic web project” on page 159. The configuration of a project can be modified after it has been created by selecting **Project Facets** from the project Properties window and adding and removing facets from the list that is presented.

Click **Cancel** to return the New EAR Application Project: EAR Application Project window, Figure 5-4 on page 149.

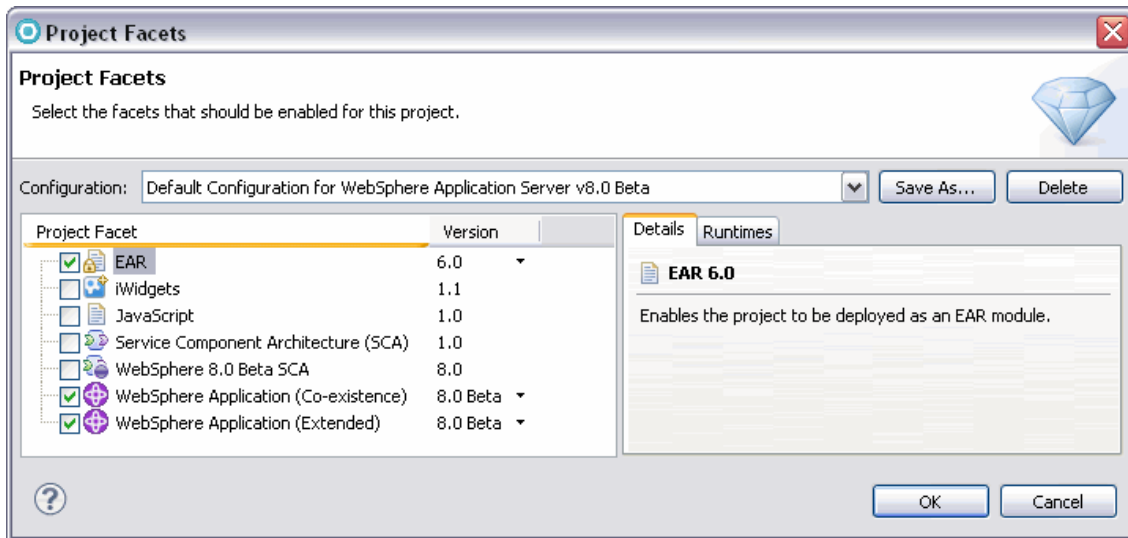


Figure 5-5 Project Facets window

- For the Working sets section back on the New EAR Application Project: EAR Application Project window (Figure 5-4 on page 149), Rational Application Developer uses working sets to group resources, such as files, for viewing or filtering operations. Rational Application Developer has many predefined working sets, including EAR Projects, JPA Projects, and EJB Projects. Resources in a new enterprise application project are automatically included in the EAR Project working set and all user-defined working sets that are of type EAR Project. By default, “Add project to working sets” is not selected. If “Add project to working sets” is selected, you can click Select to open the Select Working Sets window, as shown in Figure 5-6 on page 152. Any existing working sets of type Resource can be selected. The most generic type of working set is the Resource type. If no working sets of this type are available, new working sets can be created by clicking New. Working sets of type Resource are the only type of working set that can be created here. In this case, one working set of type Resource is available, which is called Brians Projects, and it has not been selected. You can create working sets of any required type elsewhere in Rational Application Developer. If you clicked Select, click Cancel in the Select Working Sets window to return to the New EAR Application Project: EAR Application Project window, as shown in Figure 5-4 on page 149.

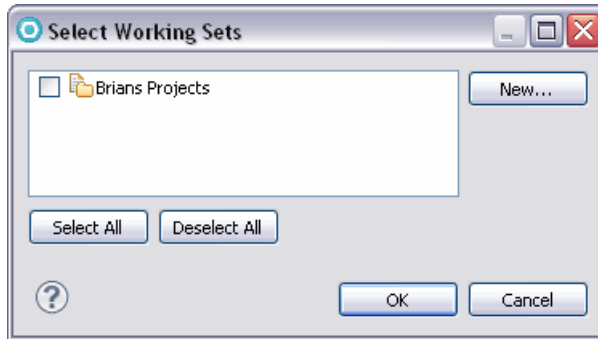


Figure 5-6 Select Working Sets window

5. Clicking **Next** in the New EAR Application Project: EAR Application Project window shows the final window in the Enterprise Application Project wizard. In this window (Figure 5-7), you can select other projects that are to be part of this new enterprise application. Check boxes are included for all the Java, EJB, Web, and Application Client projects in the current workspace. If a check box is selected, the project is added to the project references for the new enterprise application project.

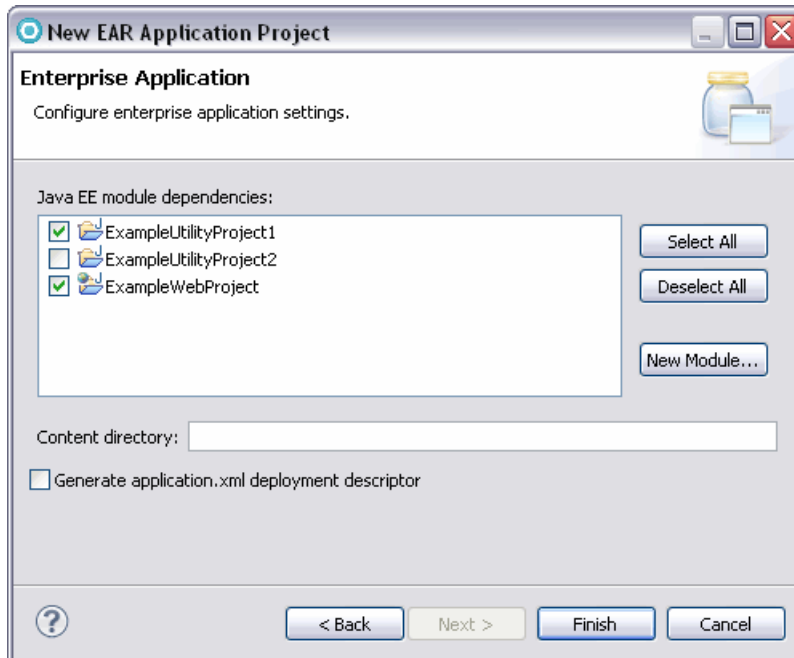


Figure 5-7 New EAR Application Project: Enterprise Application window

6. In this Enterprise Application window, as shown in Figure 5-7 on page 152, you can specify the following information:
- Content Directory. This option specifies the folder within the Enterprise Application project under which the contents will be stored. You can leave this box empty, meaning that all contents will be stored under the root directory.
 - New Module. This button provides the ability to automatically create projects to be referenced by the new Enterprise Application project. Clicking New Module opens the window in Figure 5-8.
 - Java EE module dependencies. Because the current workspace used here contains three existing projects, three modules are listed in the Java EE module dependencies list box. The modules listed can be selected or deselected as required, all can be selected, or all can be deselected by using the Select All and Deselect All buttons. The selections made indicate that there is a dependency on the ExampleWebProject and ExampleUtilityProject1 projects, because both have been selected. There is no dependency on ExampleUtilityProject2, because it is not selected.
 - Generate Deployment Descriptor. Optionally, check this box to generate the application.xml deployment descriptor file.

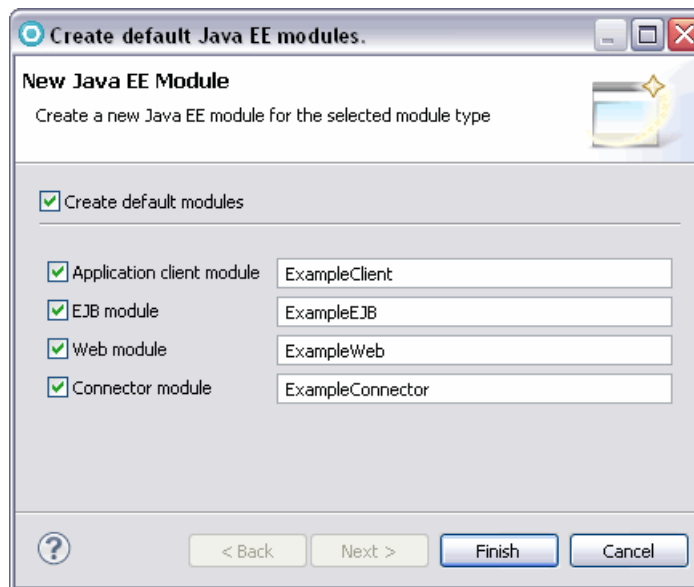


Figure 5-8 Create default Java EE modules window

7. The Create default Java EE modules window, Figure 5-8 on page 153, opens when you click **New Module** on the New EAR Application Project: Enterprise Application window. Choose either of the following actions:
 - Click Cancel if you decide not to create any default Java EE modules.
 - Select the check boxes for the projects that you want to create, change the names if desired, and click Finish.
8. In the New EAR Application Project wizard, New EAR Application Project: Enterprise Application window, click **Finish**, and the new project and any associated projects are created.

If the project that you created is associated with a particular perspective, in this case with the Java EE perspective, but you currently have a separate perspective selected, Rational Application Developer prompts you to decide whether you want to switch over to the relevant perspective (Figure 5-9).

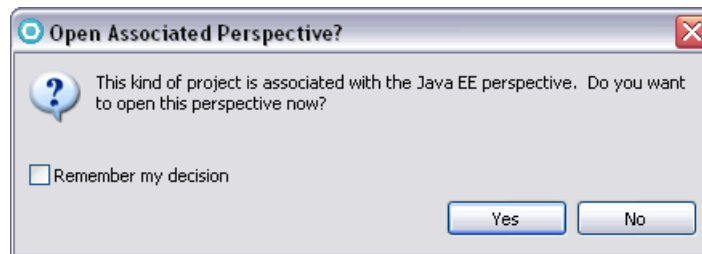


Figure 5-9 Open Associated Perspective message

5.2.2 Project properties

To make changes to the properties of a project, right-click the project and select **Properties**. Figure 5-10 shows the Properties window for an Enterprise Application project.



Figure 5-10 Project properties for an Enterprise Application project

In the Properties window, you can edit most project attributes. Each type of project has separate options available.

The following options are a few of the available options for an Enterprise Application project:

- ▶ **Deployment Assembly.** This option shows the packaging structure for the Enterprise Application project.
- ▶ **Project Facets.** You select this option to view the facets that are available for this project and to have the opportunity to add or remove facets.
- ▶ **Project References.** You use this option to configure project dependencies.
- ▶ **Server.** You use this option to specify the default application server to use when running this application.
- ▶ **Validation.** You use this option to indicate whether you want to run any non-default validation tools, and if so, which ones to run after making changes.

5.2.3 Deleting projects

To delete a project from the workspace, right-click the project and select **Delete**. When deleting projects from a workspace, Rational Application Developer gives you the option to also delete the project contents on disk.

Figure 5-11 shows the Delete Resources window that opens when deleting a project. The default setting only removes the project from the workspace and leaves the project files on disk intact. Select the “Delete project contents on disk (cannot be undone)” check box if you want to remove the project completely.

Deletion: Deleting a project from disk is permanent, and the project cannot be opened again.

Refactoring: Depending on the projects being deleted, part of the optional refactoring might result in other projects being modified. These changes can be previewed by clicking the Preview button that is shown in Figure 5-11.

If a project is only removed from the workspace, you can import it later by selecting **File** → **Import** → **General** → **Existing Projects Into Workspace**. A project that has been deleted from the workspace takes up no memory and is not examined during the build process. Deleting projects from the workspace can improve the performance of Rational Application Developer.

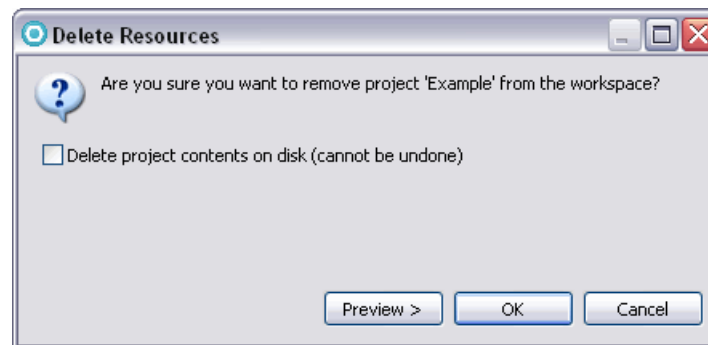


Figure 5-11 Project Delete Resources window

5.2.4 Transferring projects between workspaces

Projects can be transferred between workspaces as *archive files*. An archive file is a compressed file that is used to encapsulate a project. To create an archive file for any project, select **File** → **Export** → **General** → **Archive File**, and specify

which projects to export and to which location. Any number of projects present in the workspace can be exported to the archive file. Several options are available when creating the archive, such as archive file type of zip or tar and whether the file content needs to be compressed. To import projects, which are stored as an archive file, into another workspace, select **File** → **Import** → **General** → **Existing Projects Into Workspace**, and choose **Select archive file** rather than “Select root directory”. When exporting and importing a project, the project interrelationships are transferred, but not the referenced projects. Therefore, it might be necessary to export all the related projects to ensure that everything is available in the other workspace.

5.2.5 Closing projects

You can close projects that are in a workspace. Closing a project locks it so that it cannot be edited or referenced from another project. To close a project, select either Close Project or Close Unrelated Projects from the Explorer context menu. Closed projects are still visible in the workspace, but they cannot be expanded.

Closing unnecessary projects can speed up compilation times, because the underlying application builders only have to check for resources in open projects. You can reopen a closed project by right-clicking it, and selecting Open Project.

5.3 Java EE 6 project types

Rational Application Developer complies with the Java EE 6 specifications for the development of enterprise applications.

Module packaging into files, as described in 5.1, “Java Enterprise Edition 6” on page 142, is only applied by Rational Application Developer when a Java EE project is exported or deployed.

While working within Rational Application Developer, only the projects present in the workspace are edited. The relationships between the enterprise application projects, and the modules that they contain, are managed by Rational Application Developer, and are applied on export or deployment to produce a properly packaged EAR file.

Figure 5-12 shows the arrangement of projects and their associated outputs. This figure relates to Figure 5-2 on page 144, where the relationships between various Java EE modules are reflected in the Rational Application Developer project references.

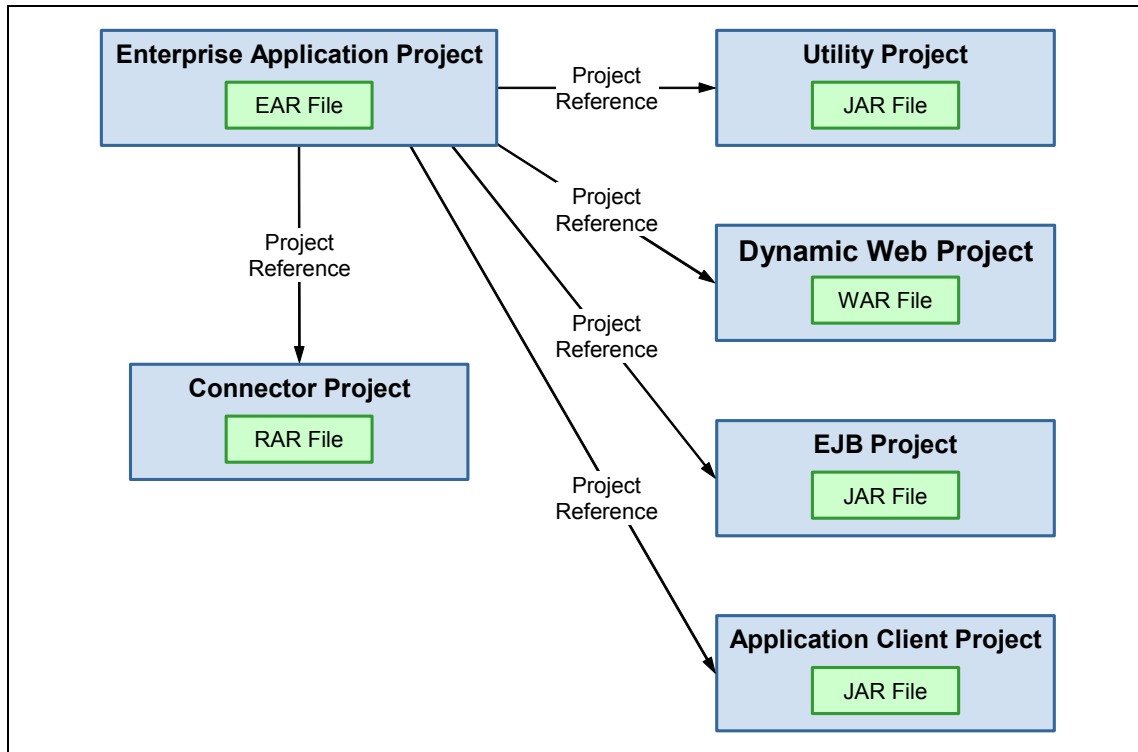


Figure 5-12 Java EE projects in Rational Application Developer

The following sections briefly describe the creation of each of the project types that are shown in Figure 5-12, using the wizard that is available for each project type.

5.3.1 Enterprise application projects

Enterprise application projects contain the resources needed for enterprise applications and can contain references to a combination of web projects, EJB projects, application client projects, resource adapter projects, and utility library projects. You can specify the relationships when creating a new enterprise application project through the wizard, as previously shown, or through the project properties. We have already covered the wizard used to create an

enterprise application project in detail in 5.2.1, “Creating a new project” on page 147.

5.3.2 Application client project

Application client projects contain the resources needed for application client modules. An application client module is used to contain a fully functioning client Java application (non-web-based) that connects to and uses the resources in an enterprise application and an application server. By holding a reference to the associated enterprise application, it shares information, such as the Java Naming and Directory Interface (JNDI) references to EJB components and to data sources.

The New Application Client Project wizard allows the project location, target run time, application client module version, project configuration, EAR membership, and working sets to be specified. Clicking the configuration Modify button allows the facets for the project to be defined.

By default, the facets Application Client Module, Java, WebSphere Application Client (Co-existence), and WebSphere Application Client (Extended) are selected. The iWidgets, JavaScript, JDBC Mediator, JPA, SCA, and WebSphere 8.0 Beta Service Component Architecture (SCA) facets are available for selection. By default, the wizard also creates a Java Main class for the Application client project.

For more information about developing application clients, see Chapter 13, “Developing Java Platform, Enterprise Edition (Java EE) application clients” on page 649.

5.3.3 Dynamic web project

A dynamic web project contains the resources needed for a Java EE web application, such as JSP, Java servlets, and HTML pages as well as additional file types.

The New Dynamic Web Project wizard allows the project location, target run time, dynamic web module version, project configuration, EAR membership, and working sets to be specified. Clicking the configuration Modify button allows you to define the facets for the project.

With the default configuration selected, the dynamic web module, Java, WebSphere Application Client (Co-existence), and WebSphere Application Client (Extended) facets are selected. JSP, Struts, or JavaServer Faces capabilities are not included, by default.

Several other configurations are available in addition to the default configuration, but the list of configurations depends on the target run time and dynamic web module version selected. With the target run time set to WebSphere Application Server V8.0 Beta and the Dynamic web module version set to 3.0, the following additional configurations are available:

- ▶ IBM JAX-RS configuration
- ▶ JavaServer Faces V2.0 project
- ▶ Minimal configuration
- ▶ Minimal configuration for WebSphere Application Server

If the JavaServer Faces V2.0 Project configuration is selected, the Dynamic Web Module, Java, JavaServer faces, WebSphere Application Client (Co-existence), and WebSphere Application Client (Extended), which ensures that the web module can be used for JavaServer Faces development, are selected.

A large number of facets are available for Dynamic Web projects, including:

- ▶ Struts
- ▶ JavaServer Pages Standard Tag Library (JSTL)
- ▶ Dojo Toolkit
- ▶ Java Persistence API (JPA)
- ▶ JAX-RS

For more information about developing dynamic web applications, see Chapter 18, “Developing web applications using JavaServer Pages (JSP) and servlets” on page 981.

5.3.4 EJB project

EJB projects contain the resources for EJB applications. These resources include the classes and interfaces for the EJB components, the deployment descriptor for the EJB module, IBM extensions, bindings files, and files describing the mapping between entity beans in the project and relational database resources.

The New EJB Project wizard allows the project location, target run time, EJB module version, project configuration, EAR membership, and working sets to be specified. Clicking the configuration Modify button allows the facets for the project to be defined.

In the case of a target run time of WebSphere Application Server V8.0 Beta and an EJB module version of 3.1, two predefined configurations are available in addition to the default configuration:

- ▶ Minimal Configuration
- ▶ Minimal Configuration for WebSphere Application Server

With the default configuration selected, the EJB Module, Java, and WebSphere EJB (Extended) facets are selected. With the minimal configuration selected only, the EJB Module and Java facets are selected.

The wizard allows an EJB Client JAR to be created, which includes all the resources needed by client code to access the EJB module-like interfaces.

For more information about developing EJB, see Chapter 12, “Developing Enterprise JavaBeans (EJB) applications” on page 577.

5.3.5 Connector project

A connector project contains the resources that are required for a Java EE resource adapter.

The New Connector Project wizard allows the project location, target run time, Java EE Connector Architecture (JCA) module version, project configuration, EAR membership, and working sets to be specified. Clicking the configuration Modify button allows the facets for the project to be defined.

The default configuration selects the Java and JCA Module facets.

5.3.6 Utility project

A utility project is a Java project that contains Java packages and Java code as .class files.

The New Java Utility Module wizard allows the project location, target run time, project configuration, EAR membership, and working sets to be specified. Clicking the configuration Modify button allows the facets for the project to be defined.

The default configuration selects the Java and Utility Module facets.

In the case of a target run time of WebSphere Application Server V8.0 Beta, four predefined configurations are available in addition to the default configuration:

- ▶ Minimal configuration
- ▶ Minimal JPA 1.0 configuration
- ▶ Minimal JPA 2.0 configuration
- ▶ Minimal configuration for WebSphere Application Server

If the Minimal JPA 2.0 configuration is chosen, the Java, JPA, WebSphere Application Client (Co-existence) and Utility Module facets are selected, which

ensures that the utility module has the capability of being a container for JPA entities.

5.4 Project wizards

The following list describes a selection of the project wizards that you can use to create projects within Rational Application Developer. The project category is shown in brackets after the project name. To invoke a wizard, select **File** → **New** → **Project** and select the appropriate project wizard. A wizard prompts you for the required information as appropriate for the type of project. To ensure that all project types are available, select **File** → **Preferences** to open the preferences window and then choose **General** → **Capabilities**. Click **Enable All** and then click **OK** to ensure that all the Rational Application Developer installed capabilities are enabled and all project types are listed.

Each wizard creates a project of the specified type with the files, folders, supporting libraries, and references to support the project. However, after the project has been created, it is still possible to change aspects of a project through the project properties:

- ▶ **Project (General)**. This wizard is used for the simplest projects that only contain a collection of files and folders. It has no associated project builder configuration and is useful for creating a project that has no application code, for example, a project to store XML or XML Schema Definition (XSD) files, or to store application configuration information.
- ▶ **Faceted Project (General)**. This wizard allows a project to be created using a specific pre-existing configuration or using a selection of facets selected when the wizard is executed. The desired project configuration can be selected from a list of currently available project configurations, including those that are predefined in Rational Application Developer and any custom configurations created by the user. New configurations can also be created that are specific to a user's needs.
- ▶ **Report Project (Business Intelligence and Reporting Tools)**. BIRT is an initiative that provides an open source reporting system in Java. This wizard creates a report project that can combine database information or content from XML into report templates.

BIRT: To learn more about BIRT, visit the following web address:

<http://eclipse.org/birt/phoenix/project>

- ▶ **ODA Designer Plug-In Project (Business Intelligence and Reporting Tools)**. Open Data Access (ODA) is a data access framework that can be used to

access both standard data sources, such as those sources accessible using Java Database Connectivity (JDBC), as well as custom data sources. BIRT uses ODA. When working with ODA, you can create an ODA driver and associated GUI. An ODA Designer Plug-in project is used when creating the GUI for an ODA driver.

- ▶ ODA Runtime Driver Plug-in Project (Business Intelligence and Reporting Tools). This type of project is used to create an ODA driver run time. Together, an ODA Designer Plug-in project and an ODA Runtime Driver Plug-in project are used to produce a usable ODA driver.
- ▶ Projects from Concurrent Versions System (CVS). This wizard guides you through the creation of a new project by checking out an existing project within CVS. With the wizard, you can check out a complete project from CVS or create a new project as part of the check-out process.
- ▶ Data Design Project (Data). This wizard creates a project to store data design artifacts, including data design models and SQL statements.
- ▶ Data Development Project (Data). This wizard creates a project that stores a connection to a given database. From within this type of project, you can create resources to interrogate and manipulate the associated database. Initially, the wizard creates folders to store SQL scripts and stored procedures.
- ▶ Existing Rational Application Developer 6.x Data Definition Project (Data). The tooling that supports database definitions has changed since Rational Application Developer V6.0.x and V5.1.2. Therefore, any data project that contains database definitions or other database objects from previous versions of Rational Application Developer must be migrated to work with Rational Application Developer V8.0. This wizard takes a project folder in the old format and migrates it to Rational Application Developer V8.0.
- ▶ EJB Project (EJB). This wizard guides you through the process of creating a project suitable for containing EJB components. This procedure also creates an empty EJB deployment descriptor and can associate the EJB project with a new or existing enterprise application project.
- ▶ Java Project (Java). This simple wizard is used to create a Java application project. The wizard allows the Java Runtime Environment (JRE), project folder layout, and class path, including project dependencies, to be specified.
- ▶ Java Project from Existing Ant Buildfile (Java). It is possible to export the build settings of a project as an Ant file (select **File** → **Export** → **General** → **Ant Buildfiles**). With an Ant build file, this wizard can be used to create a new project based on the instructions contained within it.
- ▶ Application Client Project (Java EE). This wizard guides you through the creation of a new Application Client project. The wizard allows the project to

be associated with a new or existing enterprise application project. You can also define the desired project configuration and facets.

- ▶ Connector Project (Java EE). This wizard guides you through the creation of a Java EE connector project, which includes specifying the associated enterprise application project, as well as the configuration and set of facets applicable to this type of project.
- ▶ Enterprise Application Project (Java EE). This wizard creates a new EAR project. We describe this wizard in detail in 5.2.1, “Creating a new project” on page 147, because you create a new EAR project for many project types, such as EJB and JSF projects.
- ▶ Utility Project (Java EE). This wizard assists in the construction of a Java utility library project, which is associated with an Enterprise Application project. Code present in a Java utility library that is present in a Java EE application is shared between the modules present in the application.
- ▶ JavaScript Project (JavaScript). This wizard allows you to create a project to contain JavaScript files. The wizard allows JavaScript page support, such as the ECMA 3 browser library to be defined, as well as the project folder layout. JavaScript embedded in HTML and JSP is supported in Rational Application Developer web projects. A JavaScript project allows the creation of stand-alone JavaScript source files that can be referenced as external JavaScript by web project files, such as HTML and JSP files. Stand-alone JavaScript files normally have the extension `.js`.
- ▶ JET Transformation Project (JET Transformations). This wizard allows a Java Emitter Template (JET) transformation project to be created. JET is the automatic generation of code, such as Java source code and XML from predefined templates. The wizard allows JET project settings and transformation properties to be configured, as well as allowing a new transformation to be extended from a pre-existing one.
- ▶ JET Project with Exemplar Authoring (JET Transformations). Exemplar authoring is capturing leading practices from previous software designs so that they can be applied to new designs. This wizard allows the creation of a JET project used to create exemplar JET templates. Exemplar JET templates transform elements from an input design to elements in an output design through the application of a design pattern that encapsulates best practices.
- ▶ JPA Project (JPA). This wizard creates a Java Persistence API (JPA) project. In the past, JPA was defined within the Java EE specification for Enterprise JavaBeans 3.0. With Java EE 6 and JPA 2.0, the JPA specification is defined separately from Java EE specification for Enterprise JavaBeans 3.1. JPA is the Java EE 6 standardized object-relational mapping framework.
- ▶ Jython Project (Jython). This wizard creates an empty project for developing Jython resources. Jython is the latest version of JPython, an implementation

of the Python language, coded in Java, which can be executed on the Java virtual machine (JVM). Jython is one of the supported scripting languages when administering the WebSphere Application Server.

- ▶ OSGi Bundle Project (OSGi). OSGi is the packaging of software modules and applications so that they can be used in a standardized manner as components over a wide range of network-connected computing devices. This wizard creates a project used to develop OSGi bundles. OSGi bundles contain a collection of OSGi components.
- ▶ OSGi Composite Bundle Project (OSGi). This wizard creates a project that can be used to create an OSGi composite bundle. An OSGi composite bundle is a container for a collection of constituent OSGi bundles.
- ▶ OSGi Application Project (OSGi). An OSGi Application project is used to group OSGi bundles into an application that provides specific business logic functionality. The application acts as a facade for the services offered by the bundles with services externally visible only when the application is configured to export those services. The wizard allows the OSGi bundles, to be included in the application, to be selected when creating a new application.
- ▶ Feature Patch, Feature Project, Fragment Project, Plug-in from Existing JAR Archives, Plug-in Project, and Update Site Project (Plug-in Development). These wizards assist in the creation of Eclipse plug-ins, features, and fragments, which can enhance existing Rational Application Developer perspectives or create entirely new Rational Application Developer perspectives. The Rational Application Developer Help system has a section that describes how to use these wizards, and the Eclipse marketplace home page at the following address has information about many already built plug-ins and tutorials about building new plug-ins:
<http://marketplace.eclipse.org/>
- ▶ Portal Project (Portal). This wizard is used to develop a specific type of Java EE web application called a portal application. The interface presented by a portal application is an aggregate of other portal applications, as well as portlets. Portlet applications need a portal server target run time, such as WebSphere Portal V6.1. The wizard allows the target run time to be chosen, as well as a default theme that defines the way the portlet looks and its layout. In addition, you can choose a default *skin*, which is the border around the components aggregated by the portlet.
- ▶ Portlet Project (Portal). This wizard creates a project that is used to develop portlets that can be used in a portal project. *Portlets* are components that form part of a web user interface. During execution of the wizard, as well as selecting the target run time, you can request that an initial portlet is created with a specific configuration and features. Throughout the wizard, many other settings are available, such as the portlet content type and mode and security settings concerned with single sign-on.

- ▶ SCA Project (Service Component Architecture). This wizard allows you to create a new Service Component Architecture (SCA) project. SCA applies service-oriented architecture (SOA) concepts to the development of software components. The wizard allows you to choose the target run time and facet configuration, as well as the types of implementation that can be used for the SCA components, such as Java and EJB.
- ▶ SIP Project (SIP). The Session Initiation Protocol (SIP) is an extension to the Java EE servlet API that is intended for telecommunications applications using technologies, such as Voice over IP (VOIP). This wizard creates a web project with the appropriate facets selected to allow the construction of SIP applications.
- ▶ Dynamic Web Project (Web). This wizard creates a project for a web application, which can include JSP, servlets, and other dynamic content. We describe this type of project in 5.3.3, “Dynamic web project” on page 159. Notice that you define a dynamic web project to create a JavaServer Faces 2.0 project. To create a JavaServer Faces V1.x project, the project type Faces 1.x Component Library Project is available.
- ▶ Faces 1.x Component Library Project (Web). This wizard creates a dynamic web project that is configured to allow the creation of a JavaServer Faces 1.x component library.
- ▶ iWidget Project (Web). An *iWidget* is a reusable component that can be used in web applications. iWidgets adhere to an IBM-defined specification and run inside an iWidget-compliant container. WebSphere Application Server V8 Beta and other IBM products, provide an iWidget container. When executing the wizard, you are asked to choose either a Web Technologies (Ajax, HTML, CSS, and so on) or Web and Java EE technologies (Ajax, HTML, CSS, JSP, servlets, and so on) iWidget project type. The Web Technologies iWidget project type is a static web project and the Web and Java EE Technologies iWidget project type is a dynamic web project.
- ▶ Library Definitions Project (Web). This wizard is used to create a library of components that can be used in web applications. A project of this type can contain custom JSP and JSF tags, as well as files and other resources. After the project is created, resources can be added, such as a Faces Library Definition or a JSP Library Definition, to include the JSF and JSP tags in the library.
- ▶ Static Web Project (Web). This wizard creates a project for a website, incorporating elements, such as images, HTML, CSS, and JavaScript.
- ▶ Web Fragment Project (Web). This wizard is used to create a project that contains a portion or fragment of a complete web application. Web Fragments is a new feature included in Java EE 6 and supported by WebSphere Application Server V8.0 Beta. Web Fragment projects can be included in an existing web application in a pluggable manner without any changes being


made to the configuration of the existing web application. They are, in effect, web application utility libraries containing web application components. A Web Fragment project is packaged as a JAR file with the necessary deployment information contained in a file called `web-fragment.xml`. An article that looks in detail at Web Fragments is available on the IBM developerWorks site at the following address:

http://www.ibm.com/developerworks/wikis/download/attachments/140051369/Web+fragment_GA.pdf?version=1

5.5 Sample projects

Rational Application Developer provides a wide range of sample applications that can help you to explore the features provided by the software development platform and the types of projects that can be created.

You can access the samples in two ways:

- ▶ To access samples from the Rational Application Developer Welcome window, choose **Help** → **Welcome**. On the Welcome window, you can click the **Samples** icon (a circle with an orange ball, blue cube, and green pyramid ()). The Samples icon opens a Samples page that links to the samples that are present in the Rational Application Developer Help system.
- ▶ To access samples directly from the Rational Application Developer Help System, choose **Help** → **Help Contents**.

5.5.1 Help system samples

You can select the Help System samples from a hierarchical list in the leftmost pane of the Help window (Figure 5-13 on page 168).

The samples are arranged in categories starting with EJB and ending with OSGi.

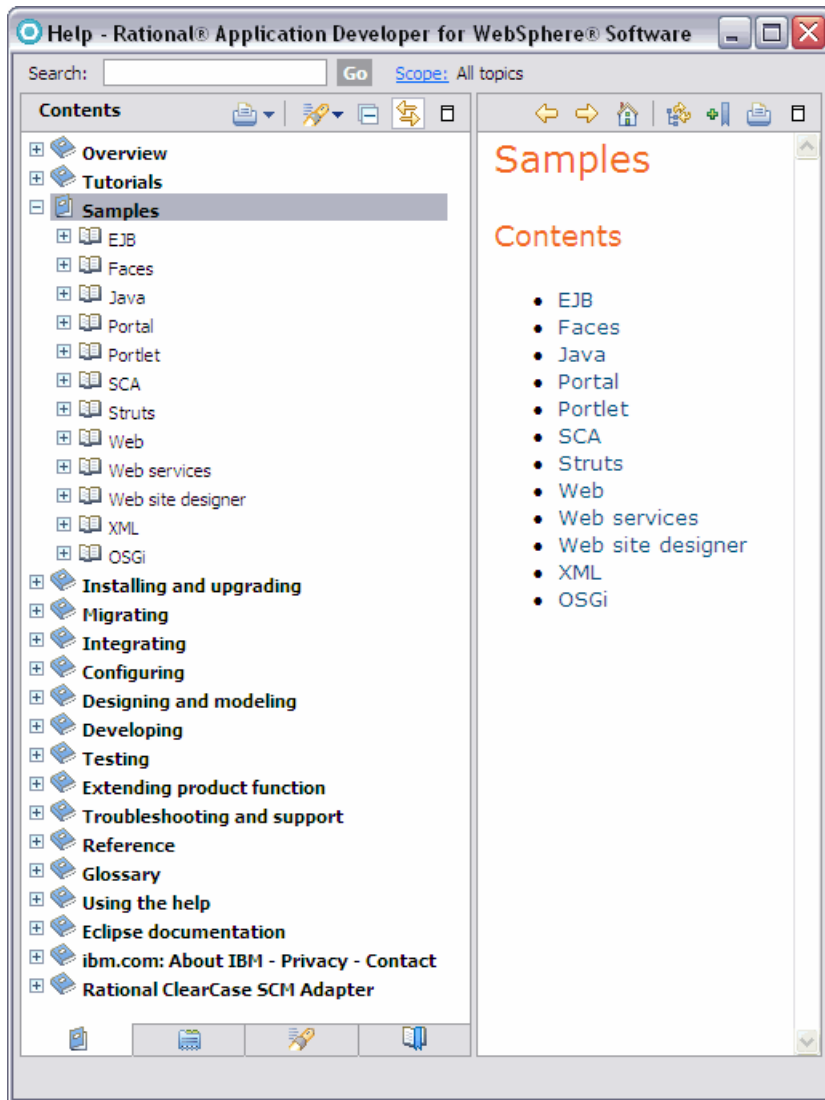


Figure 5-13 Help System samples

For example, the Stock widget application is one of the iWidget samples. The starting page for this sample provides an introduction and links to setting up the sample, getting the sample code, running the sample, and references for further information.

Figure 5-14 shows the starting page for the Stock widget sample application.



Figure 5-14 Stock widget sample

The **Import the sample** link, as shown on Figure 5-14 on page 169, when clicked, presents an import window that allows you to select the sample projects to be imported (Figure 5-15). Clicking **Finish** imports the sample projects.

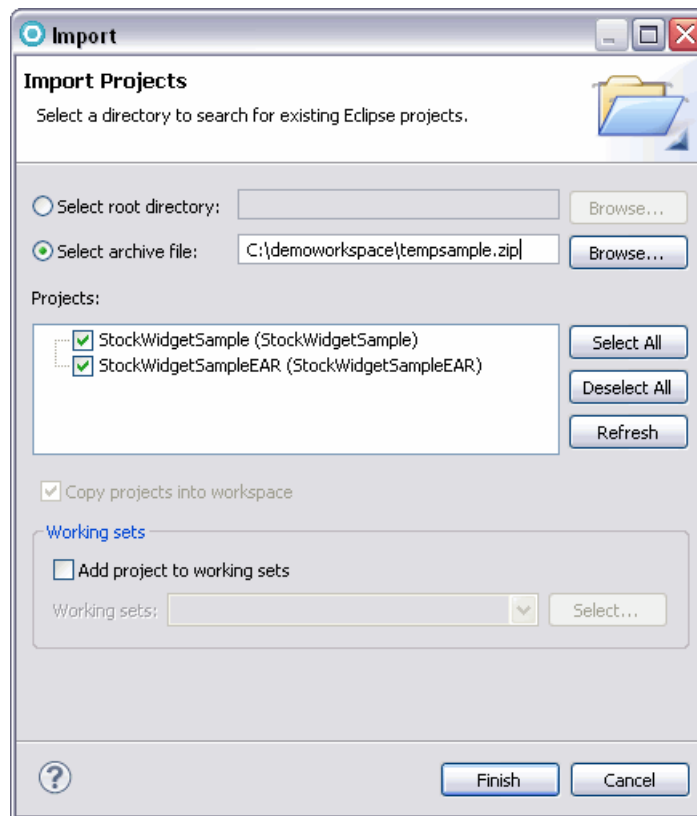


Figure 5-15 Import stock widget sample

You can now run, modify, and experiment with the imported code as required.

5.5.2 Example projects wizard

An additional way to access sample projects is through the New Project window. Rational Application Developer provides several example project wizards that can be used to add sample projects to a workspace. Select **New** → **Project** → **Examples**, and choose a sample project (Figure 5-16 on page 171).

An example project is provided for editing and validating XML files. Running the wizard adds the example project to the workspace and shows an entry from Rational Application Developer Help describing the sample.

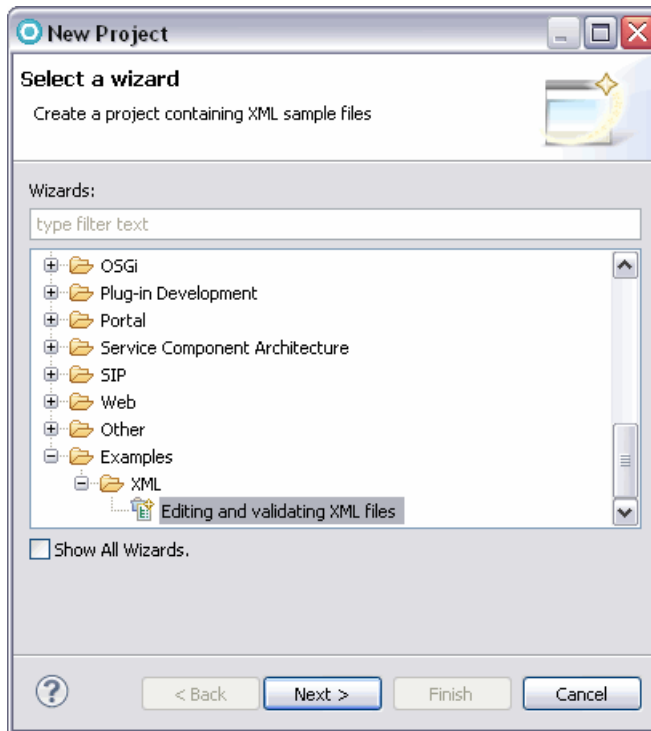


Figure 5-16 Example project in the New Project window

5.6 Summary

In this chapter, we discussed the major types of projects that can be created with Rational Application Developer, in particular, those projects that are used in the development of Java EE applications. We also presented the techniques for handling projects within a Rational Application Developer workspace, looked at the range of wizards available for the creation of projects, and provided an introduction to the samples that are supplied with Rational Application Developer.

In the remaining chapters of this book, we discuss the use of each project type in Rational Application Developer and describe the specific features that are available in each project type when building various types of applications.



Unified Modeling Language

The Unified Modeling Language (UML), an Object Management Group (OMG) standard, is used by the majority of people involved in modern software development. UML defines a graphical notation for the visual representation of a wide range of the artifacts that are created during the software development process. The visual modeling capabilities of UML range from the functionality expected of a system to the classes and components from which a system is constructed to the servers and systems on which the components are deployed.

Rational Application Developer provides visual UML tooling that, although it does not support the full capabilities of UML, is appropriate for those involved in the design and coding of software applications and components. If full UML support is required, it is provided by products, such as Rational Software Architect and Rational Software Modeler.

The chapter is organized into the following sections:

- ▶ Overview
- ▶ Constructing and visualizing applications with UML
- ▶ Working with UML class diagrams
- ▶ Describing interactions with UML sequence diagrams
- ▶ More information about UML

6.1 Overview

Rational Application Developer provides features that developers can use to visually develop and represent software development artifacts, such as Java classes, interfaces, Enterprise JavaBean (EJB) components, and web services, in UML. Rational Application Developer provides a customizable UML 2-based modeling tool that integrates tightly with the development environment. The discussion of UML in this chapter focuses on this tool.

6.2 Constructing and visualizing applications with UML

Rational Application Developer provides UML visual editing support to simplify the development of complex Java applications. Developers can create and modify Java classes and interfaces visually using class diagrams. Review of the structure of an application, by viewing the relationships between the various elements that comprise the application, is facilitated using Rational Application Developer *Browse* and *Topic* diagrams. Model elements, such as classes and packages, are synchronized automatically with their corresponding source code, allowing developers the freedom to choose to edit the model or the source code as required.

The code visualization capabilities of Rational Application Developer provide diagrams that enable developers to view existing code from various perspectives. Unlike the diagrams offered in Rational Software Modeler or Rational Software Architect, these visualizations are of actual code only. This means that full UML 2 modeling is not possible using Rational Application Developer. The UML support is present only to provide a way to visualize and understand the code or to allow the editing of code from its visual representation in a model. This is, in fact, a common way in which UML is typically employed.

Visual editing offers developers the ability to produce code without explicitly typing the code into a text editor. A palette is used to drag modeling elements, such as classes and interfaces, to a diagram. In the case of classes, you can edit them visually, for example, by adding operations and attributes or by defining their relationships with other classes.

Rational Application Developer supports the following types of UML diagrams:

- ▶ Class diagrams

Class diagrams present the static structure of an application. They show visually the classes and interfaces from which the application is composed, their internal structure, and the relationships which exist between them. When visually representing the static view of an application, many class diagrams

are created by the developer as required, and a single class diagram typically presents a subset of all the classes and interfaces present in an application. Class diagrams are created within a project and exist permanently in that project until deleted.

► Sequence diagrams

Sequence diagrams present the dynamic structure of an application. They show the interactions between objects present in an executing application. Objects present in an executing application interact through the exchange of messages, and the sequencing of this message exchange is an important aspect of any application. In the case of Java applications, the most basic type of messaging between objects is the method call. Sequence diagrams visually represent objects and their lifelines and the messages that they exchange. They also provide information about the sequencing of messages in time. Sequence diagrams are created within a project and exist permanently in that project until deleted.

► Browse diagrams

Browse diagrams are specific to Rational Application Developer. They are not a new type of diagram, merely a facility provided within Rational Application Developer for the creation of diagrams. A browse diagram exists temporarily and is not editable. With this type of diagram, a developer can explore the details of an application through its underlying elements and relationships. Browse diagrams are not a permanent part of a model; they are created as needed to allow the exploration of a model.

A browse diagram provides a view of a chosen context element. Context element exploration takes place in a similar way to the way in which web pages are viewed in a web browser when navigating a website. You cannot add or modify individual diagram elements or save a browse diagram in a project. However, you can convert a browse diagram to a UML class diagram or save it as an image file for use elsewhere.

► Topic diagrams

Topic diagrams share many of the features of browse diagrams except that they are generated through the execution of a query on the application model and remain permanently in a project when created. You can customize the underlying query, open multiple topic diagrams at the same time, and save them away for further use. Each time that a topic diagram is opened, the query is executed, and the diagram is populated. They are invaluable when discovering the architecture of an existing application.

► Static method sequence diagrams

Static method sequence diagrams are a form of topic diagrams. They are non-editable diagrams that visually represent and explore the chronological sequence of messages between instances of Java elements in an interaction.

You can create a static sequence diagram view of a method (operation), including signatures, in Java classes and interfaces to illustrate the logic inside that operation.

6.2.1 UML visualization capabilities

All of these diagrams help developers to understand and document code. To provide further documentation, you can also generate Javadoc HTML documentation that contains UML diagram images. See 7.8.4, “Generating the Javadoc with diagrams automatically” on page 303.

The UML visualization tools are applicable not only to Java classes but also to other types of artifacts, such as web services and EJBs. Rational Application Developer also supports data visualization using UML or Information Engineering notation.

Figure 6-1 on page 177 provides an overview of the workspace that you might see when using the UML visualization capabilities:

- ▶ The center area is the UML editor. This editor is used to display and modify the elements present in the model.
- ▶ A palette is built into the editor and is used to drag elements to the editor work area. The items that are displayed in the palette are specific to the type of project that is being visualized. The palette is only available when the diagram is editable. The palette is not displayed for topic and browse diagrams.
- ▶ The Outline view enables you to see, in miniature, the whole diagram currently being viewed with the area of the diagram you have zoomed in on highlighted. This view can be useful for finding your way around a complex diagram, because you can left-click the area of the outline view that is highlighted and drag it around to see a separate zoomed-in area. You can also change the outline view to show a tree of all the elements that are present in the current diagram.
- ▶ The Properties view enables you to review or change any property that is related to a selected diagram or a diagram element.
- ▶ You can drag project elements from the Package Explorer view directly to the editor work area to add these items to the diagram. You can, for example, drag a Java class from the Explorer to the editor work area where it will be rendered as a UML class in the diagram. If relationships exist between the Java class you have dragged, such as an association with another class, this relationship will be rendered as well (only when the related class is already present on the diagram, or the related class is among the dragged classes).

The diagram, shown in Figure 6-1, was created by dragging the DepositCommand class, TransferCommand class, and Command interface to the editor work area. In this case, the TransferCommand and DepositCommand classes implement the Command interface, and as you can see, UML implements the relationships that have also been rendered in the diagram.

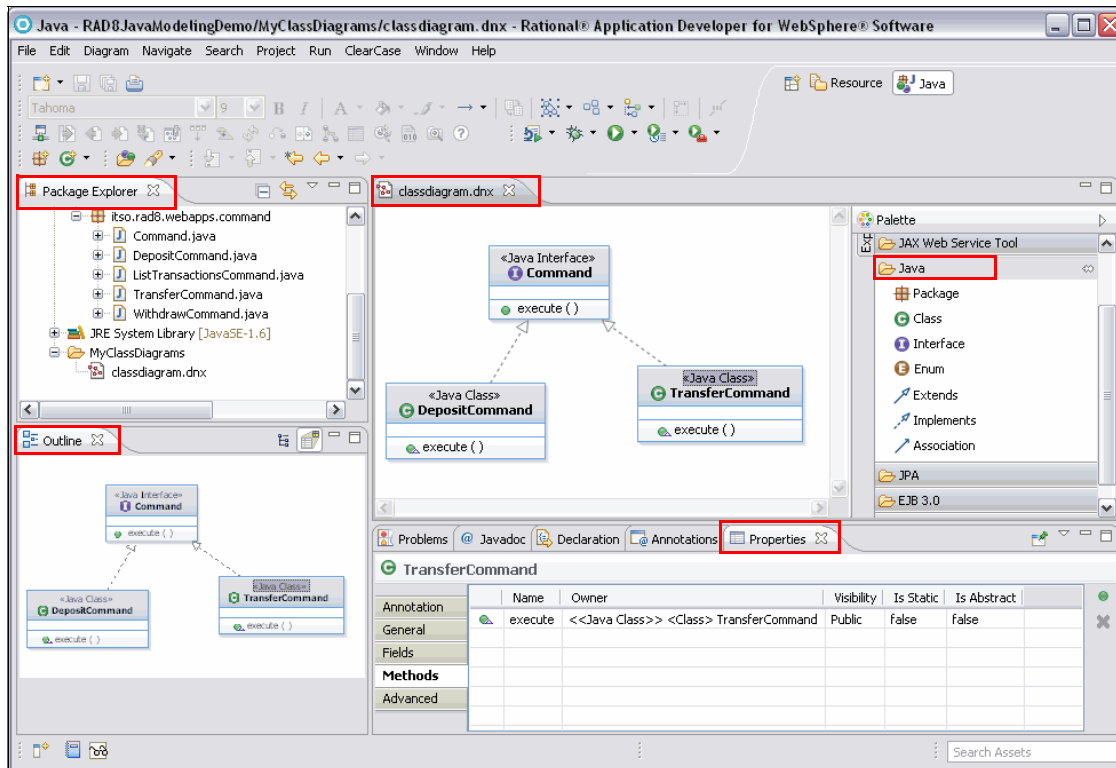


Figure 6-1 Example workspace when using the UML visualization capabilities

6.2.2 Unified Modeling Language

A *model* is a description of a system from a particular perspective, omitting irrelevant details so that the characteristics of interest are seen more clearly. That is, a model is a simplification of reality. The more complex a system is, the more important that it is modeled. Models are useful for problem solving and understanding, communicating with team members and stakeholders, preparing documentation, and designing applications. Modeling promotes a better understanding of requirements, cleaner designs, and more maintainable applications.

UML is a standardized language for modeling the various aspects of an application. You can use this language to visualize, specify, construct, and document the artifacts of an application. UML models are constructed using three kinds of building blocks: elements, relationships, and diagrams.

Elements

Elements are an abstraction of the structural or behavioral features of the system being modeled. Each element type has specific semantics and gives meaning to any diagram in which it is included. UML defines the following kinds of elements:

Structural elements This type of element is used to model the static parts of a system. Examples of this type of element are interfaces, classes, components, and actors.

Behavioral elements This type of element models the dynamic parts of a system. They are typically found in UML interaction diagrams and in other diagram types. Examples of this type of element are objects, messages, activities, and decisions.

Grouping or organizational elements

This type of element is used to group together other elements into a meaningful set. An example of a grouping element is the package.

Annotational elements

This type of element is used to comment and describe a model. Examples of this type of element are notes and constraints.

Relationships

Relationships are used to document the semantic ties that exist between model elements. UML relationships fall into the following commonly used categories:

► Dependency relationships

This type of relationship is used to indicate that changes to a specific model element can affect another model element. For example, consider a Bank class that depends on an Account class. An operation that can be called on an object of the Bank class might take as a parameter a reference to an object of the Account class. The Account object has been created elsewhere, but the Bank object uses it and therefore depends on it. After the Account object has been used, the Bank object does not retain its reference to it. A dependency relationship therefore exists between the Account class and the Bank class.

► Association relationships

This type of relationship indicates that instances of a specific model element are connected to instances of another model element. For example, a

Customer class might have an association with an Account class. When an object of the Customer class obtains a reference to an object of the Account class, it retains it and can interact with the Account object whenever required. If the classes are implemented in Java, then typically the Customer class includes an instance variable to hold the reference to an Account object.

Several types of association relationships can be used, depending on how tightly connected the modeling elements are. For example, consider a relationship between a Car and Engine class. In this case, the association is stronger than in the previous Customer and Account example. One of the stronger types of association, such as aggregation or even composition, might therefore be used in the model. In the case of composition, the connection between the classes is so strong that the lifetimes of the objects are bound together. The object of one class never exists without an object of the other class, and when one is deleted so is the other.

- ▶ Generalization relationships

This type of relationship is used to indicate that a specific model element is a generalization or specialization of another model element. Generalization relationships are used to show inheritance between model elements. If Java is used to implement a UML class element and that element has a generalization relationship with another class in a model, the Java `extends` keyword is used in the source code to establish this relationship. For example, an Account class might be a generalization of a SavingsAccount class. Another way to say this is that the SavingsAccount is a specialization of the Account class, or that the SavingsAccount class inherits from the Account class.

- ▶ Realization relationships

This type of relationship is used to indicate that a specific model element provides a specification that another model element implements. Realization relationships are typically used between an interface and the class that implements it. The interface defines operations, and the class implements the operations by providing the method behind each operation. In Java, this maps to the `implements` keyword. For example, consider a Command interface and a DepositCommand class. A realization relationship exists in the model between the DepositCommand class and the Command interface. This means that the DepositCommand class implements the Command interface.

Diagrams

A UML diagram provides a visual representation of an aspect of a system. A UML diagram illustrates the aspects of a system that can be described visually, such as relationships, behavior, structure, and functionality. Depending on the content of a diagram, it can provide information about the design and architecture of a system from the lowest level to the highest level. UML provides

thirteen types of diagrams with which you can capture, communicate, and document all aspects of an application.

The individual diagrams can be categorized into the following main types. Each type represents a separate view of an application.

- ▶ **Static**

Diagrams of this type show the static aspects of a system. This includes the units from which the application is constructed (classes, for example) and how the units relate to each other. This type of diagram does not show changes that occur in the system over time. Examples of this type of diagram are the component diagram, class diagram, and deployment diagram.

- ▶ **Dynamic**

Diagrams of this type show the dynamic aspects of a system. They document how an application responds to requests or otherwise evolves over time by showing the collaborations that take place between objects and the changes to the internal states of objects. Objects in a system achieve nothing unless they interact or collaborate. Examples of this type of diagram are the sequence diagram and communication diagram.

- ▶ **Functional**

Diagrams of this type show the functional requirements of a system. Examples of this type of diagram are the use case diagram.

You can find additional information about UML at the following web address:

<http://www-01.ibm.com/software/rational/uml>

6.3 Working with UML class diagrams

A UML class diagram is a diagram that provides a static view of an application. It shows part or all of the components or elements in an application and the relationships between them, such as inheritance and association. You can use class diagrams to visually represent and develop Java applications and Java EE EJB applications. Rational Application Developer also allows Web Services Description Language (WSDL) elements, such as WSDL services, port types, and messages to be shown on class diagrams. In Rational Application Developer, enhanced support is provided for UML visualization of EJB 3.x applications.

The content of a class diagram is stored in a file with a .dtx extension. The UML class diagram editor consists of an editor window that shows the current class

diagram and a palette that contains individual drawers containing the elements that can be added to a class diagram.

6.3.1 Creating class diagrams

A new class diagram is created using the New Class Diagram wizard. You can start this wizard directly from the menu in Rational Application Developer. To create a class diagram from the menu, select **File** → **New** → **Other** → **Modeling** → **Class Diagram**. After the wizard starts, you can enter the name for your class diagram and specify the folder where you want to store the class diagram file (Figure 6-2).

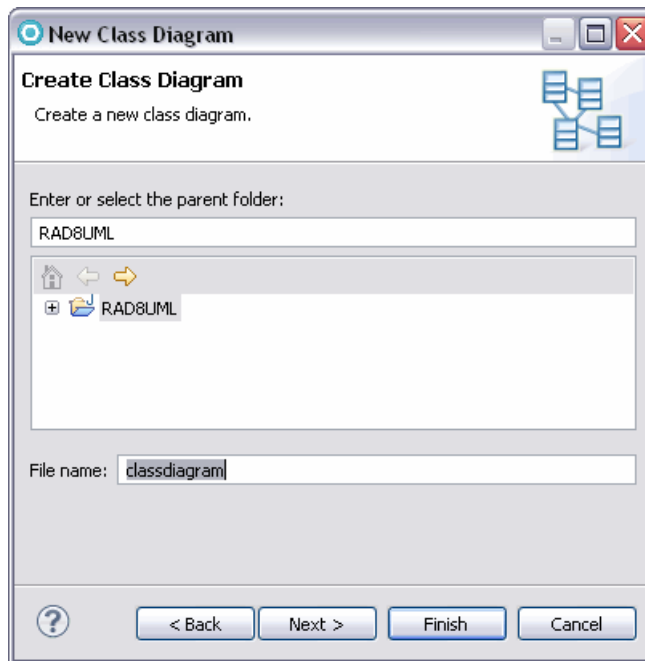


Figure 6-2 New Class Diagram wizard: Create Class Diagram window

Clicking **Finish** will create a class diagram with default modeling capabilities for the current project type, in this case, Java. Clicking **Next** will show the New Class Diagram wizard: Specify Class Capabilities window (Figure 6-3 on page 182). This window allows the capabilities of the class diagram to be selected and determines what can be used to model. The drawers available in the palette are also determined by what is selected here. In this case, only the Java palette drawer will be present, because, as can be seen in Figure 6-3 on page 182, Java Modeling is the only capability selected. These capabilities can also be

configured globally on the preference page at **Windows** → **Preferences** → **General** → **Capabilities**.

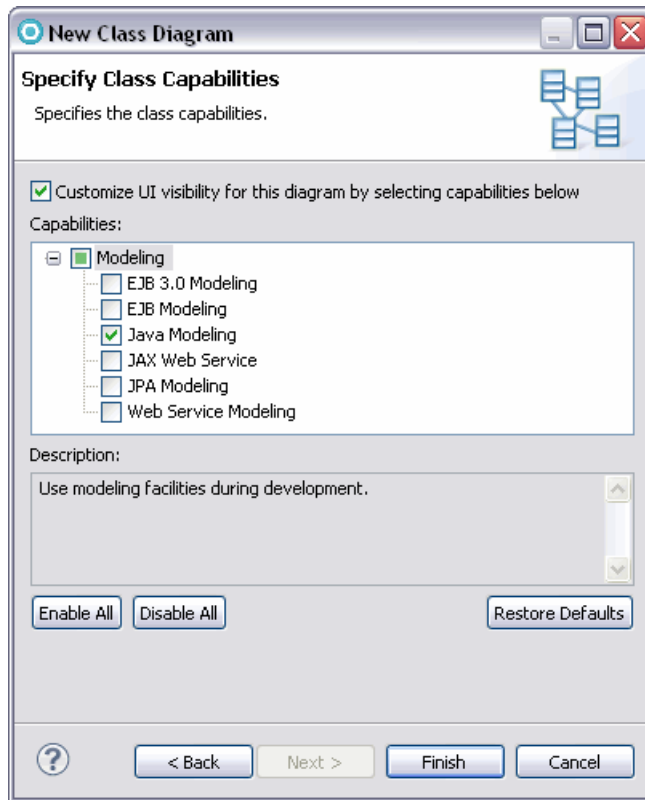


Figure 6-3 New Class Diagram: Specify Class Capabilities

When you click **Finish**, the new class diagram is created and opens for editing with the associated palette on the right side, as shown in Figure 6-4 on page 183.

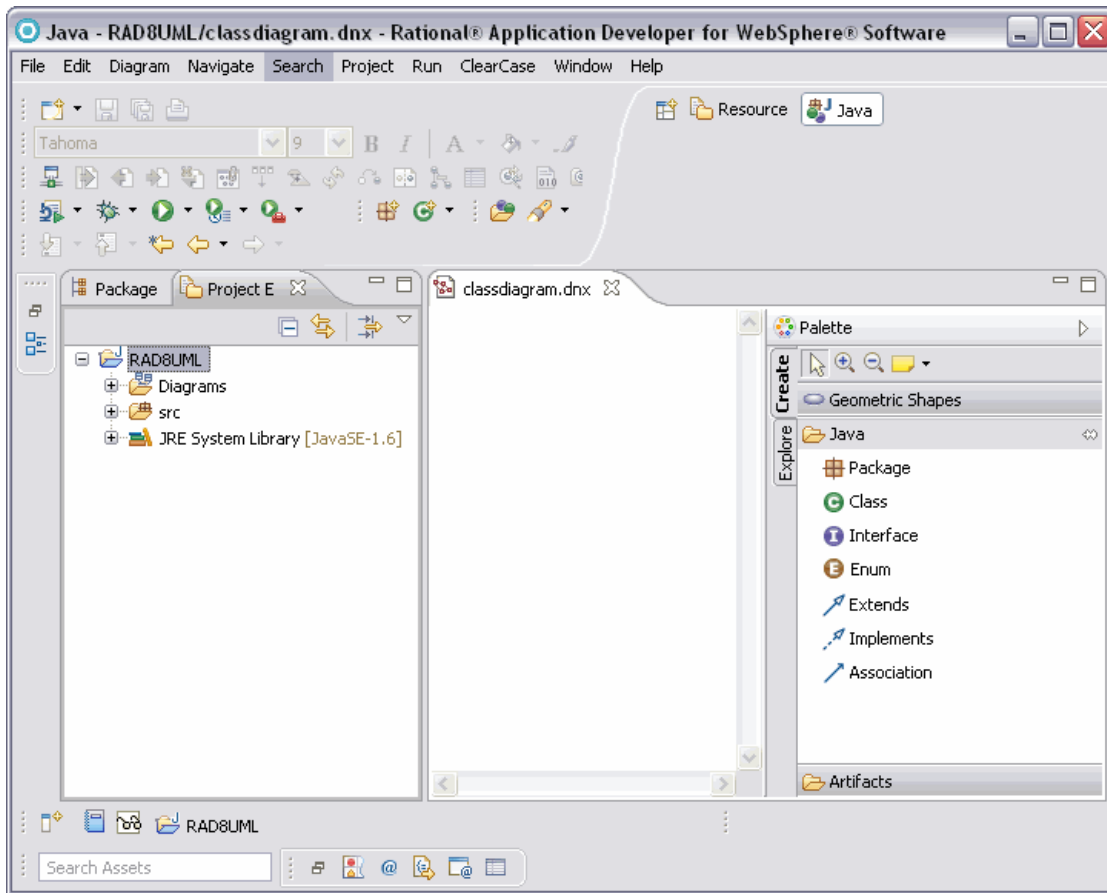


Figure 6-4 A new class diagram

Alternatively, you can create a UML class diagram from existing source elements within a project, including packages, classes, interfaces, and EJB components. In the Project Explorer, right-click the desired source element or elements and select **Visualize** → **Add to New Diagram File** → **Class Diagram**. In a similar way, you can add elements to an existing class diagram.

You can create as many class diagrams as you want to depict various aspects of your application.

6.3.2 Creating, editing, and viewing Java elements by using UML class diagrams

When working with class diagrams, developers can create, edit, and delete Java elements, such as packages, classes, interfaces, and enum types, to allow the visual development of Java application code.

To draw a class diagram, you select the desired elements from the palette and drag them to the class diagram editor window. Then the appropriate wizard opens. For example, the New Java Class wizard opens when you drag a Java class from the palette, which guides you in the creation of the new element. Alternatively, elements can be created directly in the Explorer and placed on the diagram later.

Figure 6-5 shows a class as it is seen when added to a class diagram. The class is rendered as specified in the UML standard.

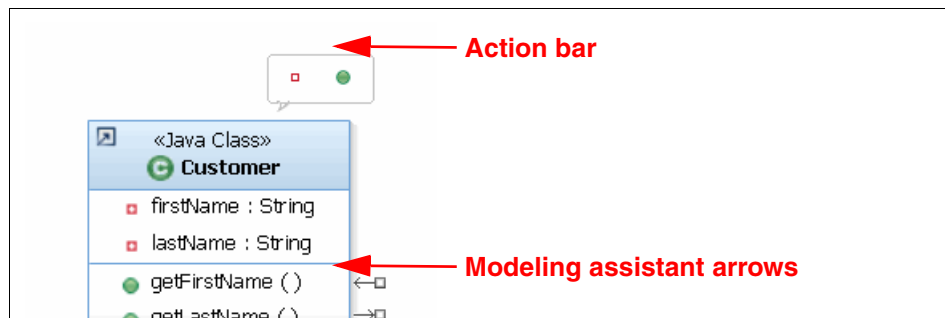


Figure 6-5 A Java class with action bar and modeling assistant arrows visible

In this case, three compartments are visible:

- ▶ The upper compartment or name compartment contains the class name and, if required, a stereotype. A *stereotype* is a string, which is surrounded by guillemets (angled brackets), that indicates the element more precisely. In this case, we have a class, but more precisely, it is a Java class.
- ▶ The middle compartment is the attribute compartment and contains the attributes present in the class.
- ▶ The lower compartment is the operation compartment and contains the operations present in the class.

To show or hide individual compartments, right-click the class and select **Filters** → **Show/Hide Compartment**. Additionally, if the class is annotated, the annotations are shown in a separate compartment over the attribute compartment.

Also, when you hover the mouse cursor over a class, the action bar and the modeling assistant arrows are displayed:

- ▶ The *action bar* is an icon-based context menu that provides quick access to commands that allow you to edit a diagram element. In the case of a Java class, you can add fields and methods to the class. The actions, available on the action bar, are also available when you right-click the class in the diagram and select the **Add Java** option.
- ▶ With the *modeling assistant*, you can create and view relationships between the selected element and other elements, such as packages, classes, or interfaces. One modeling assistant arrow points towards the element and the other points away. The arrow pointing toward the element is for incoming relationships. Thus, when creating a relationship where the selected element is the target, you use the incoming arrow. Similarly, the arrow pointing away from the element is for outgoing relationships and is used in a similar way.

To create a relationship from one Java element to another element, follow these steps:

1. Move the mouse cursor over the source element so that the modeling assistant is available and click the small box at the end of the outgoing arrow.
2. Drag the connector that is displayed and drop it on the desired element or on an empty space inside the diagram if you want to create a new element.
3. Select the required relationship type from the context menu that opens when the connector is dropped (Figure 6-6).

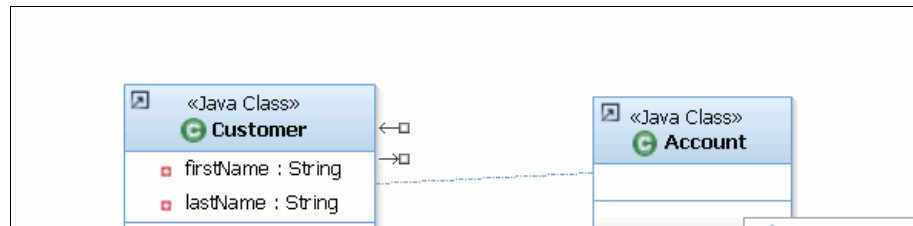


Figure 6-6 Using the modeling assistant to create a relationship

You can create incoming relationships in the same way by using the incoming arrow. Alternatively, you can select the desired relationship in the tool palette and place it on the individual elements.

With the modeling assistant, you can also view related elements that are based on a specific relationship. The elements that can be viewed are those that already exist in the model but are not currently shown on the class diagram. To view related elements that are based on a specific relationship, double-click the small box at the end of the outgoing or incoming arrow and select the desired

relationship from the resulting context menu, as shown in Figure 6-7. This action is equivalent to selecting **Filters** → **Show Related Elements** from the element's context menu.

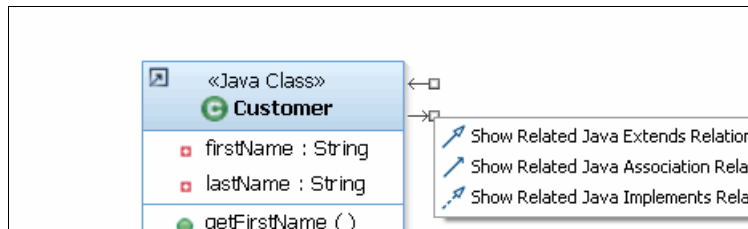


Figure 6-7 Viewing related Java elements using the modeling assistant

The context menu for a class includes several additional editing options:

- ▶ **Delete from Diagram.** This option removes the visual representation of the selected element or elements from the diagram. It does not delete the element from the project. When you delete Java elements from a class diagram, the underlying associations remain intact.
- ▶ **Delete from Project.** This option removes the visual representation of the selected element or elements from the diagram and also removes the element from the project. It is wise to be careful when using this option, because deleted elements cannot be retrieved. If the element is shown in diagrams other than the one from which it is being deleted, it is still shown on these diagrams but is rendered with an X inside a circle to indicate that it is no longer present in the project and hence cannot be edited. The element can however be deleted from the diagram. For all the diagram references to be removed on the deletion of a project element, a preference can be configured at **Windows** → **Preferences** → **Modeling** → **Java**.
- ▶ **Format.** This option changes the properties of the selected element that govern its appearance and location in a diagram. Modifying these properties only changes the appearance of this specific rendition of the element, it does not affect how the element is rendered elsewhere on the diagram or in another diagram. Several of the properties can be configured globally using the modeling preferences window.
- ▶ **Filters.** By using this option, you can manage the visual representation of elements in UML class diagrams. This option only affects what is visible on a diagram when the element is rendered, irrespective of what the element actually contains. For example, UML allows a class to have operations but allows the operations to be hidden, if required, when the class is drawn on a class diagram. With the filters option, you can show or hide attributes and operations, determine if operation signatures are displayed, or specify if the fully qualified names of individual classes are shown.

- ▶ Filters → Show Related Elements. This option is a particular function of the Filters option and requires its own explanation, because it is so useful. Show Related Elements helps developers to query for related elements in a diagram. As shown in Figure 6-8, in the Show Related Elements in Diagram window, you can select from a set of predefined queries. By clicking **Details**, you can view and change the actual queried relationships, along with other settings related to the selected query. In Figure 6-8, all of the Java relationship types have been selected.
- ▶ Refactor and Source. These options provide the same functionality to change and edit the underlying Java code as they do when invoked on the class directly in the Explorer.

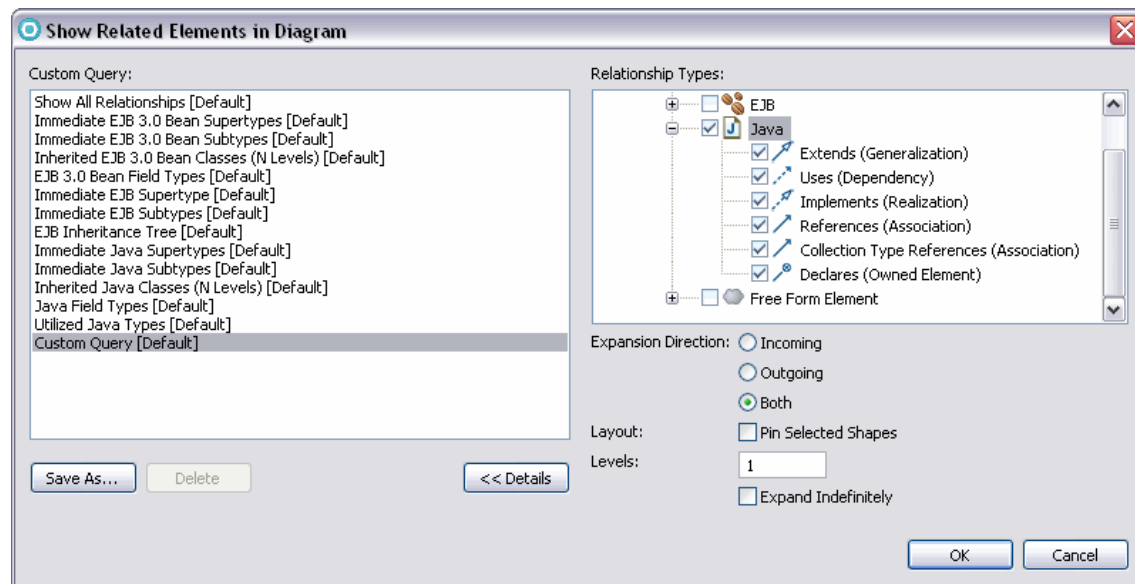


Figure 6-8 Show Related Elements in Diagram window

Figure 6-9 shows a class diagram with elements and the relationships between them.

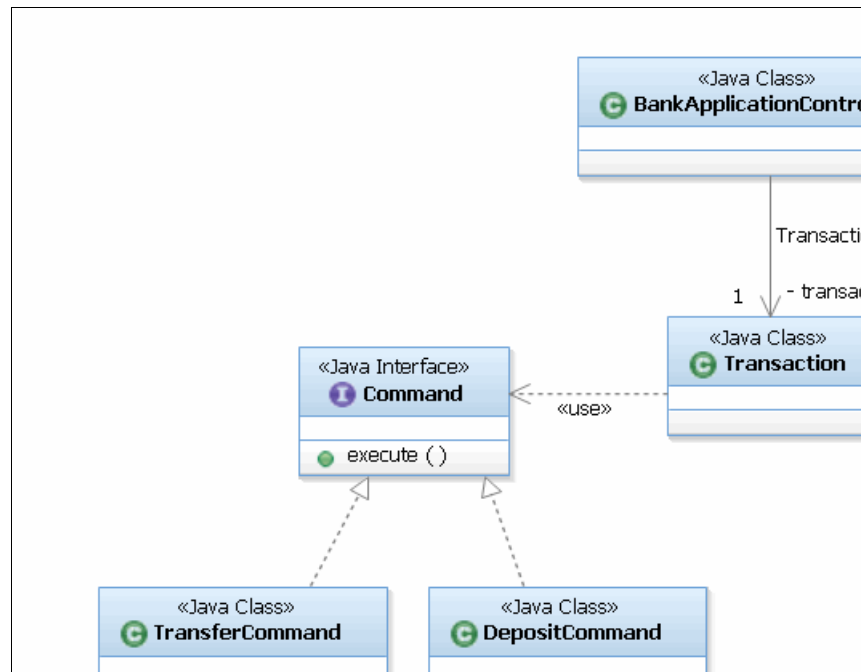


Figure 6-9 Class diagram showing various UML elements

6.3.3 Creating, editing, and viewing EJB components within UML class diagrams

By using class diagrams, developers can also visually represent and develop EJB components in EJB applications. Developers can create class diagrams and populate them with existing EJB components to allow the business tier architecture to be documented and understood. They can also use class diagrams to develop new EJB components, including EJB relationships, such as inheritance and association, and to configure the security aspects of bean access, such as security roles and method permissions.

Rational Application Developer supports EJB 3.x, and class diagrams can be drawn showing EJB 3.0 and EJB 3.1 beans. The support for drawing class diagrams showing EJB 2.1 and earlier beans is still supported, but we do not discuss it here. EJB 3.1 is part of Java EE 6 and requires that the target run time is set to WebSphere Application Server V8 Beta.

To use the EJB class diagram capabilities, a class diagram must be created within the context of an EJB project. The palette can then be used to create and edit as before, but now with the addition of an EJB drawer. Alternatively, you can create an EJB in the Enterprise Explorer view and place it on a diagram later.

Figure 6-10 shows the graphical representation of an EJB 3.1 Singleton session bean.

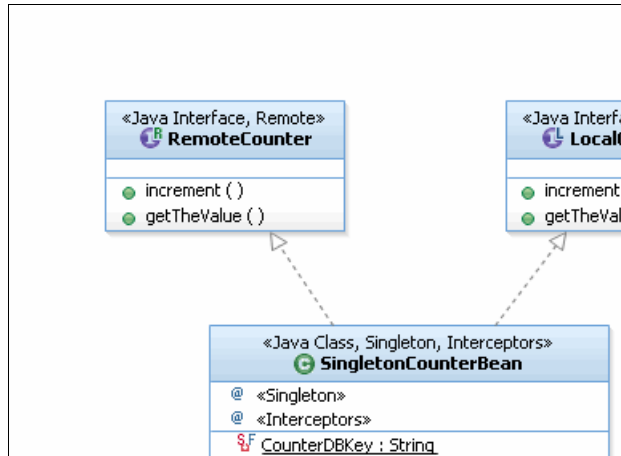


Figure 6-10 Visualization of an EJB 3.1 Singleton session bean

To visualize an EJB, you drag it from the Enterprise Explorer view to the class diagram editor. EJB components are rendered as a class that is stereotyped as <<Java Class>> with the appropriate stereotype for the type of bean. In this case, the other stereotypes are <<Singleton>> and <<Interceptor>>, because we have an EJB 3.1 Singleton stateless session bean that is also an interceptor. An EJB exposes its functionality to clients through either a remote interface, a local interface, or both. With EJB 3.1, a bean can also have no interface and be only accessible locally. The session bean that is shown in Figure 6-10 provides both a local interface and a remote interface. The local interface is shown on the class diagram as a class stereotyped as <<Java Interface>> and <<Local>>. The remote interface is shown on the class diagram as a class stereotyped as <<Java Interface>> and <<Remote>>.

EJB components in an EJB 3.x project can employ Java Persistence API (JPA) entities to provide data persistence rather than EJB 2.1 entity beans. Figure 6-11 shows an EJB 3.1 session bean and a JPA entity. The JPA entity is shown in the diagram as a class with appropriate stereotypes.

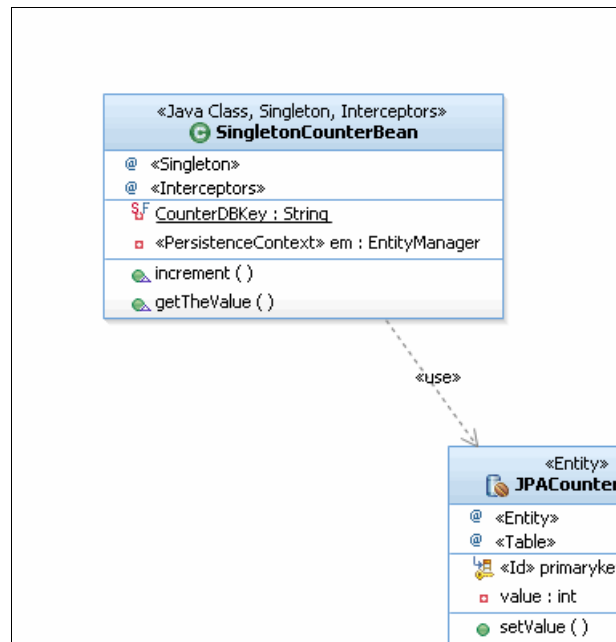


Figure 6-11 Class diagram showing an EJB 3.0 session bean and a JPA entity

Relationships between EJB components

You can use a class diagram to create relationships between EJB components. The palette supports two kinds of relationships:

- ▶ EJB inheritance
- ▶ EJB reference

EJB inheritance is a standard Java generalization relationship between two EJB classes. An *EJB reference* relationship is shown on the class diagram as an association and is implemented in the source code as an EJB 3.x reference.

Figure 6-12 on page 191 shows the following EJB 3.1 session beans with relationships:

- ▶ An inheritance relationship exists between TestSessionBean1 and BaseSessionBean, shown in the diagram as a UML generalization arrow.
- ▶ The TestSessionBean1 has an EJB reference to the no-interface session bean TestSessionBean2, shown in the diagram as a directed UML association

between the two beans. In addition, TestSessionBean1 has an EJB reference to the remote interface of TestSessionBean3 shown in the diagram as a use dependency between TestSessionBean1 and TestSessionBean3Remote and the attribute testSessionBean3Remote in TestSessionBean1.

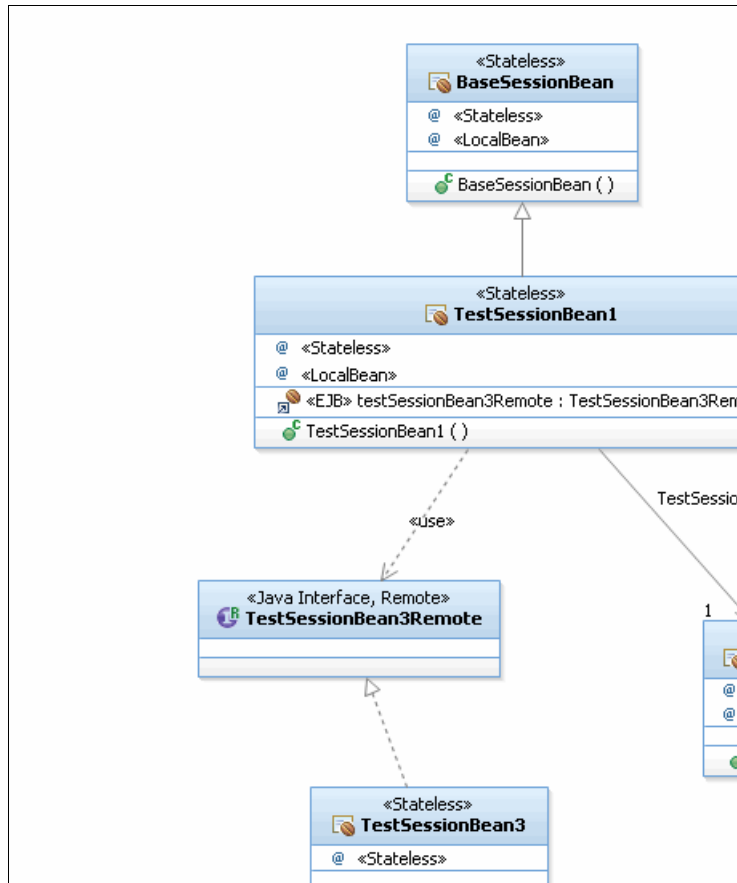


Figure 6-12 Class diagram showing EJB 3.1 beans with relationships

Previously, we looked at the Filters option, which is available when we right-click a class in a class diagram. This feature works for EJB 3.x classes. Figure 6-13 on page 192 shows that a user-defined query has been created with all the EJB relationship types selected for the currently selected EJB. All of the relationships for this EJB will be shown on the diagram when **OK** is clicked.

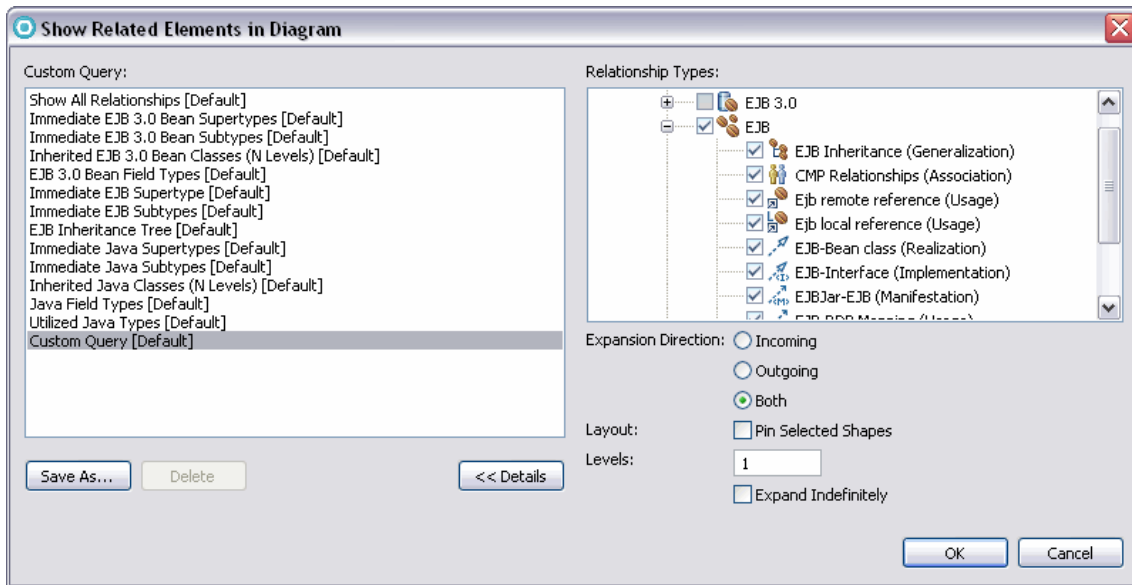


Figure 6-13 Show Related Elements in Diagram window for an EJB session bean

As well as selecting the separate types of relationships to include in the query, the expansion direction can be selected. If you select **Incoming**, all elements are shown that are related to the selected element. If you want to see all the elements with which the selected element has a relationship, select **Outgoing**. Any changes made to the queries can be saved for future use.

From the context menu, several other options are available to edit a selected EJB or to change its appearance. Most of these features have been described previously, so the following discussion focuses only on topics that are specific to EJB. Several of these features are exposed as wizards.

Security roles and method permissions

You can use UML class diagrams to visually manage EJB security. This includes creating security roles and configuring method permissions. We only discuss the support for configuring security for EJB 3.x beans using security annotations here.

An EJB 3.x security configuration involves the creation of required security roles and the definition of the EJB method security permissions. We do not discuss the linking of security to roles defined in the container, which is typically done using annotations as explained in the following steps:

1. To create a security role for a specific EJB, right-click the bean and select **Add EJB 3.0 → Security → Declare Roles**.

2. In the Declare the Roles window (left in Figure 6-14), perform these steps:
 - a. Click **Add** and enter the name of a security role, for example, User, and click **Finish**.
 - b. Click **Finish** to complete the process.

The annotation `@DeclareRoles(value="User")` is added to the source code for the EJB, and the EJB shown in the class diagram is updated with the stereotype `<<DeclareRoles>>` to indicate that a security role is now present (right in Figure 6-14).

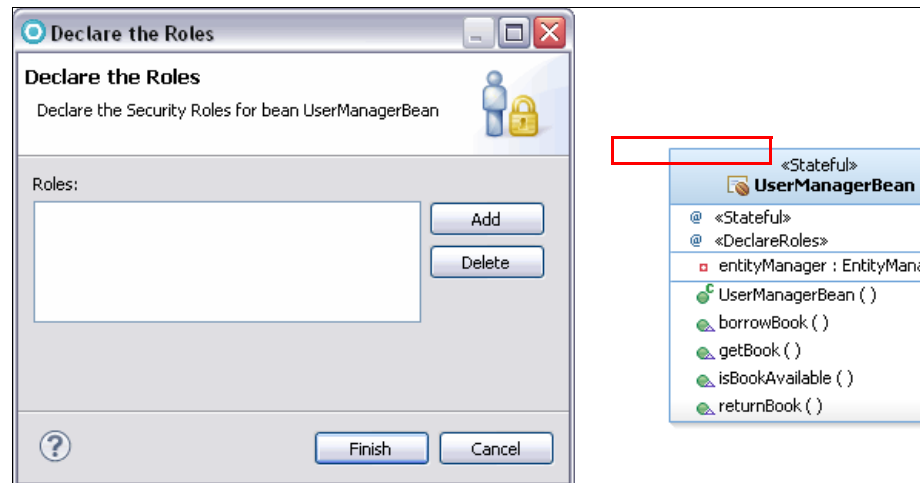


Figure 6-14 Declare the Roles window and diagram `<<DeclareRoles>>` stereotype

3. To define method permissions for an EJB 3.x session bean, follow these steps:
 - a. Select a bean method in the class diagram, for example, `getBook`, right-click, and select **Add EJB 3.0** → **Security** → **Set Allowed Roles**.
 - b. In the Set Allowed Roles window, select the roles permitted to execute the method.

If the role `User` is chosen, the annotation `@RolesAllowed(value="User")` is added to the `getBook` method in the source file for the bean and the method in the class diagram is updated with `<<RolesAllowed>>`.

Figure 6-15 on page 194 shows an EJB with method permissions set for the `getBook` method.

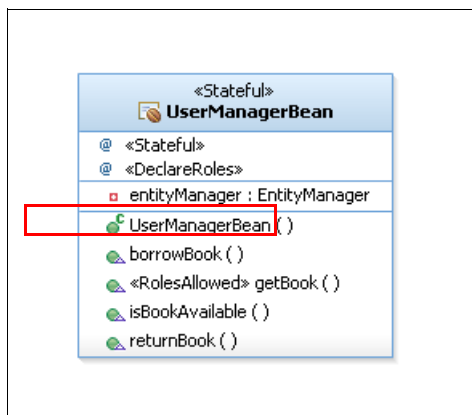


Figure 6-15 EJB 3.x session bean with method permissions

6.3.4 Creating, editing, and viewing WSDL elements within UML class diagrams

With Rational Application Developer, developers can represent and create WSDL and XML Schema Definition (XSD) elements as well as JAX-WS Web Services using UML class diagrams.

To use this feature, you must first enable the Web Service Development capability:

1. Select **Window** → **Preferences**.
2. In the Preferences window, expand the **General** node to access the **Capabilities** page. In the Capabilities page, click **Advanced**.
3. In the window that opens, expand the **Web Service Developer** node and select **Web Service Development**.

Figure 6-16 on page 195 shows the graphical representation of a WSDL service. To visualize a WSDL service, select its WSDL file from the Enterprise Explorer view and drag it to a class diagram. Alternatively, right-click a WSDL file and select **Visualize** → **Add to New Diagram File** → **Class Diagram**.

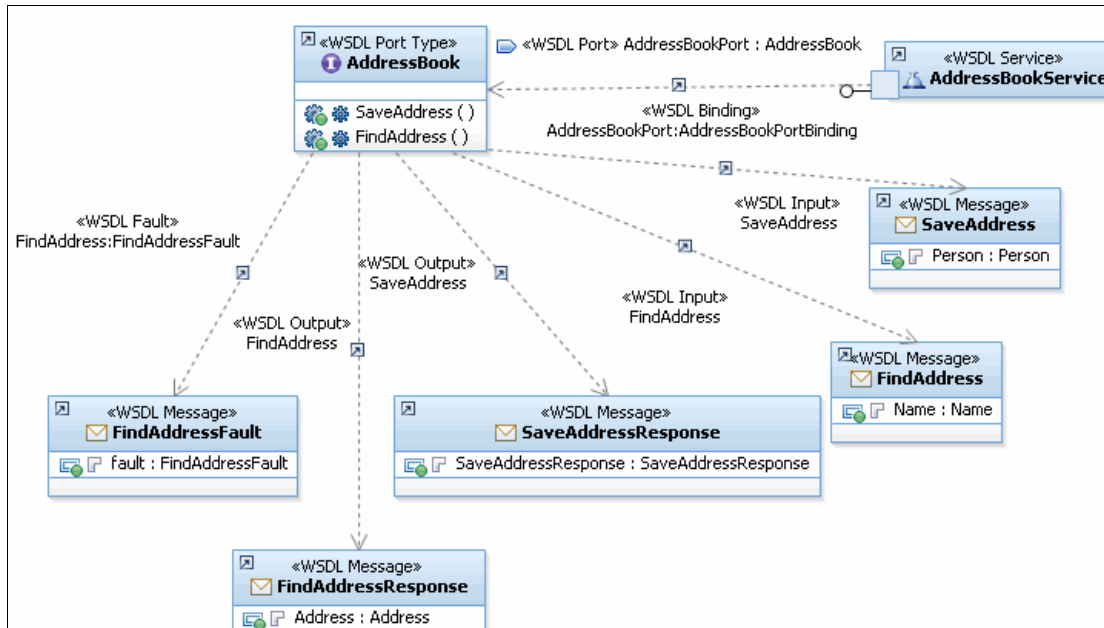


Figure 6-16 Graphical representation of a web service

By default, the external view of a service is shown. If you want to switch to the compressed view, right-click the service, select **Filter**, and clear **Show External View**.

Similar to EJB, WSDL services are displayed as UML classes on a class diagram with appropriate stereotypes:

- ▶ The individual ports of a service are depicted as small squares on the side of the class. The class is stereotyped as <<WSDL Service>>. The functionality provided by a port is exposed by a port type.
- ▶ A port type is displayed as a UML interface that is modeled using the *lollipop* notation. In Figure 6-16, you can see the port type explicitly displayed as an interface being linked to its port. This link is realized as a dependency with stereotype <<WSDL Binding>>. It describes the binding being used for this port type. A port type references messages that describe that type of data being communicated with clients.
- ▶ A message consists of one or more parts that are linked to types. Types are defined by XSD elements. Figure 6-16 shows that messages are displayed as UML classes with the <<WSDL Message>> stereotype. XSD elements are also displayed as UML classes, although none are shown in Figure 6-16.

In this section, we provide a sample scenario that explains how you can use the UML class diagram editor to create a new web service from scratch. We use the tools provided by the tool palette to create the various elements of a web service, such as services, ports, port types, operations, and messages.

Creation of the service involves the following steps:

1. Creating a WSDL service
2. Adding ports to a WSDL service
3. Creating WSDL port types and operations
4. Creating WSDL messages and parts
5. Creating XSD types and editing WSDL message part types
6. Adding messages to WSDL operations
7. Creating bindings between WSDL ports and port types

Each step is documented in detail in the following sections.

Creating a WSDL service

If you select the WSDL Service element in the tool palette and drop it on an empty space inside a class diagram, the New WSDL Service wizard starts to create a new WSDL service along with a port, as shown in Figure 6-17.



Figure 6-17 New WSDL Service wizard

To begin, you must specify the WSDL file that will contain the service. You can either click **Browse** to select an existing file or you can click **Create New** to start

the New WSDL File wizard. If you create a new WSDL file, on the Options page, clear **Create WSDL Skeleton**, because you will create these elements later in the next tasks. Finally, provide a name for the service and port and click **Finish**.

Figure 6-18 shows the result of this task. Similar to an EJB, a WSDL service is displayed as a UML class but with the stereotype <<WSDL Service>>. The port is depicted as a small square on the side of the class. The external view of the component is shown. To switch to the compressed view, open the context menu and clear **Show External View** from the Filter submenu. In this case, the port is not visible.



Figure 6-18 Visualization of a WSDL service component

Adding ports to a WSDL service

A WSDL service consists of one or more individual ports. A port describes an endpoint of a WSDL service that can be accessed by clients. It contains the properties: name, binding, and address. The name property provides a unique name across all the ports defined within the enclosing WSDL file. The binding property references a specific binding. And, the address property contains the network address of the port. Adding a port is not required for our scenario, because a port has already been created and added to the service in the previous task.

To add a port to a WSDL service if one is required:

1. Right-click the service and select **Add WSDL → Port**.
2. In the Port wizard (Figure 6-19 on page 198), enter the name of the port.
Optional: You can specify a binding and protocol.
3. Click **Finish**.

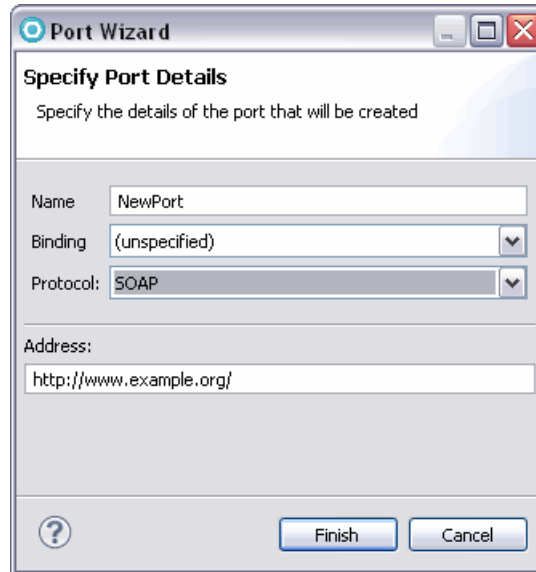


Figure 6-19 Port wizard

After you create a port, you can use the Properties view to review or change any property of the port:

1. Right-click the square representing the desired port and select **Properties**.
2. On the top of the Properties view, select **General**.
3. On the General page (Figure 6-20), enter a new name and address and select a binding and protocol.

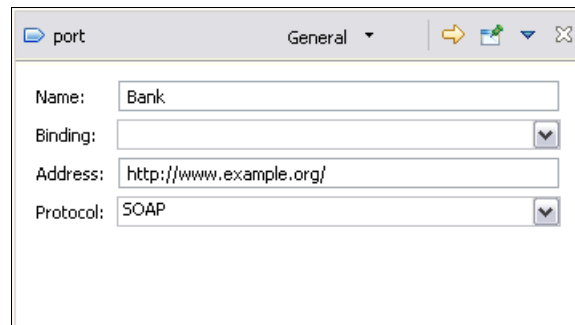


Figure 6-20 Port properties shown in the Properties view

Creating WSDL port types and operations

A port type describes the behavior of a port. It defines individual operations that can be performed and the messages that are involved. An operation is an abstract description of an action supported by a service. It provides a unique name and the expected inputs and outputs. It might also contain a fault element that describes any error data that the operation might return.

You can create a new port type together with an operation with the help of the New WSDL Port Type wizard (Figure 6-21). You can start this wizard either by dragging a WSDL Port Type from the palette to the class diagram or by right-clicking in the diagram and selecting **Add WSDL** → **Port Type**.

First, you must specify the WSDL file to contain the port type. A port type is not restricted to be in the same WSDL file as the enclosing WSDL service. As described previously, you can click **Browse** to select an existing WSDL file or click **Create New** to create a new file. Next you must provide the port type name and operation name and then click **Finish**.

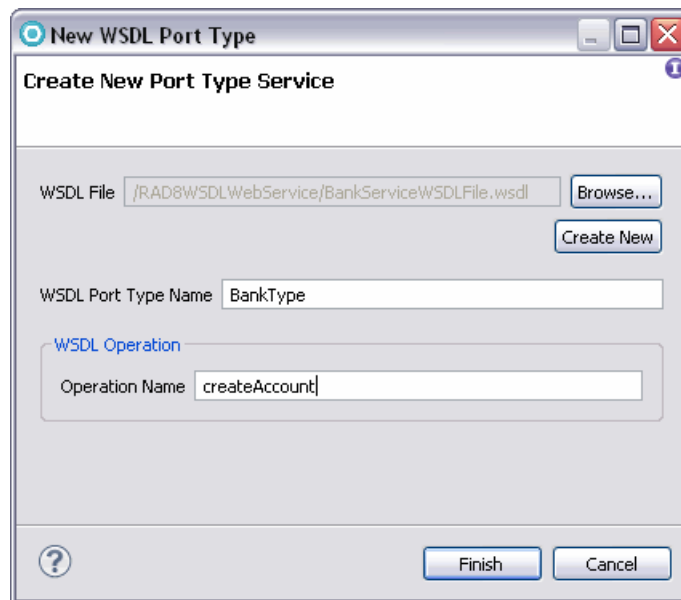


Figure 6-21 New WSDL Port Type wizard

Figure 6-22 on page 200 shows the result. A port type is visualized in the diagram by using an interface with the stereotype <<WSDL Port Type>>. You can add further operations by right-clicking the port type, and selecting **Add WSDL** → **Operation**.

We have not created a connection between this port type and the port. We do this in the last task.



Figure 6-22 Class diagram representation of a port type

Creating WSDL messages and parts

Messages are used by operations to describe the data that is being communicated with clients. An operation can have an input, output, and a fault message. A message is composed of one or several parts and each part is linked to a type. The individual parts of a message can be compared to the parameters of a method call in the Java language.

To create a new message along with a part, follow these steps:

1. Select the WSDL Message tool in the tool palette and drag it to the diagram.
2. In the New WSDL Message window (Figure 6-23), specify the WSDL file to contain the message. WSDL services or port types and messages are top-level objects that can be defined in a separate WSDL file. As described previously, you can either browse to select an existing file or create a new one. Finally, enter the message name and part name and click **Finish**.

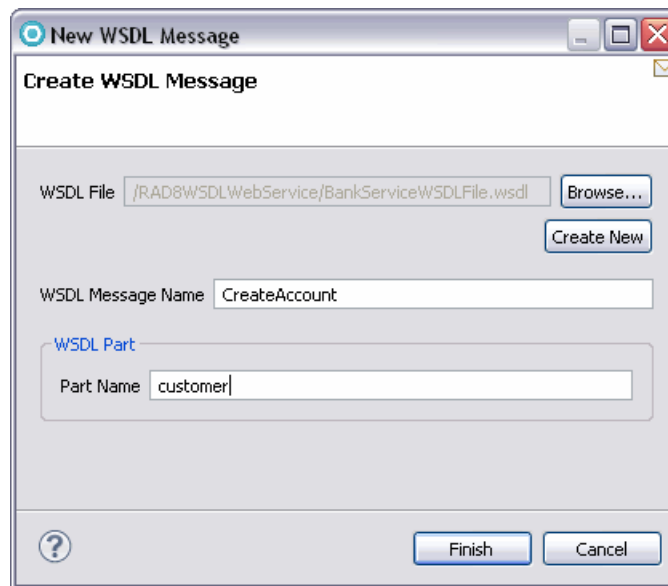


Figure 6-23 New WSDL Message wizard

Figure 6-24 shows the result. The new message is displayed by using the UML class notation with the stereotype <<WSDL Message>>. If you want to add any further parts to this message, right-click the class and select **Add WSDL** → **Part**.

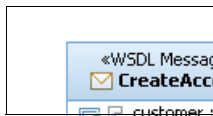


Figure 6-24 Representation of a WSDL message in a class diagram

The CreateAccount message is associated with the input-element of the createAccount operation in the next task. Before you proceed, create a second message CreateAccountResponse along with a part named account. This message will be associated with the output-element of this operation.

Creating XSD types and editing WSDL message part types

The WSDL standard recommends the use of XSD to define the type of a WSDL message part. You can use the class diagram to create and edit the required XSD objects, such as XSD elements, simple types, or complex types. If you right-click in the class diagram and select **Add WSDL**, you can select the required item from the submenu. Alternatively, you can drag the item from the palette to the class diagram. The wizard allows type definitions to be either placed inside a WSDL file, if the *In-line Schema* option is selected or in a separate schema file if the option is not selected. Choosing a separate schema file allows the Rational Application Developer schema editor to be used to edit the XSD types. It is easier to edit XSD types using the schema editor than it is from a UML class diagram.

If the XSD element added to a class diagram is a complex type, you can add new elements to it by right-clicking the complex type and selecting **Add XSD** → **Add New Element**. A new element is created within the selected complex type with a type string. You can then set or change the type of an existing XSD element. In the diagram editor, right-click an XSD element or an element within a complex type and select **Set XSD Type**. The window that opens shows a list of available types that you can select. To change the name of the element, you must edit the WSDL file directly.

After you create an XSD element, you can as easily delete it from the diagram. If you want to delete it permanently, you must edit the WSDL file directly.

To review or change a type of a part of a WSDL message, select the part to view its properties in the Properties view. Then select the **General** tab (Figure 6-25 on page 202). In the Type field, you can select the desired type. Select **Browse** to choose a type from the full list of available types, including user-defined types.

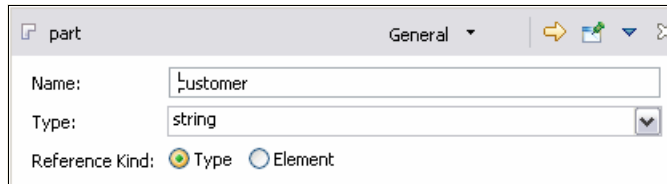


Figure 6-25 Properties view showing the properties of a WSDL message part

To proceed with this exercise, create two XSD complex types, Account and Customer, and add the elements amount, firstName, and lastName, as shown in Figure 6-26. It does not matter if these are defined in an external schema file or in-line in the current WSDL file.

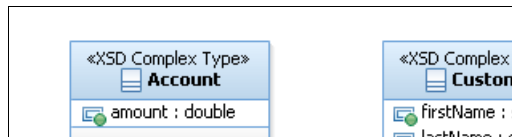


Figure 6-26 XSD complex types with elements

Finally, link these types to the parts of the messages that you have created, as shown in Figure 6-27.

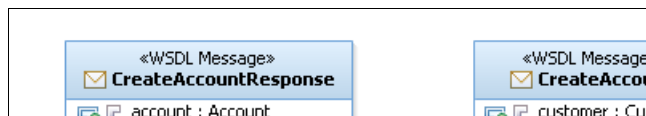


Figure 6-27 Using XSD complex types as the type for WSDL messages parts

Adding messages to WSDL operations

An operation can reference three types of messages to describe the data that is communicated with clients. These types of messages are input message, output message, and fault message.

To add a message to an operation, select either WSDL Input Message, WSDL Output Message, or WSDL Fault Message in the palette. Click the port type to which you want to add the message, and drag the cursor from the port type to the message that you want to add. In the window, select the desired operation and click **Finish** (Figure 6-28 on page 203). Alternatively, you can use the modeling assistant to do this task.

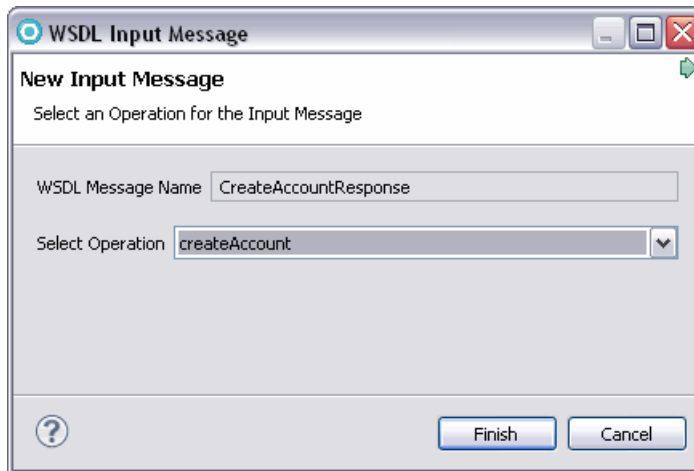


Figure 6-28 WSDL Input Message wizard

To proceed with the exercise, create a WSDL input message from the createAccount operation to the CreateAccount message. Then create a WSDL output message from the same operation to the CreateAccountResponse message (Figure 6-29).

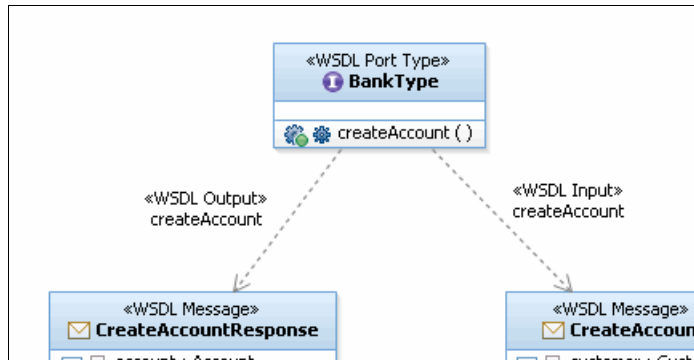


Figure 6-29 Class diagram showing a port type with messages

Creating bindings between WSDL ports and port types

A *binding* is used to link a port type to a port. The class diagram editor offers you several ways to do this. For example, you can use the WSDL Binding Creation tool in the palette:

1. Select the WSDL Binding Creation tool and click the port (shown as a small square on the side of a service). Then drag the cursor to the port type.

A new binding is created between the port and the port type. As shown in Figure 6-30, this binding is modeled as a dependency with the stereotype <<WSDL Binding>> between the two elements. The lollipop notation (line with a circle at the end) represents the interface that is provided by the port.

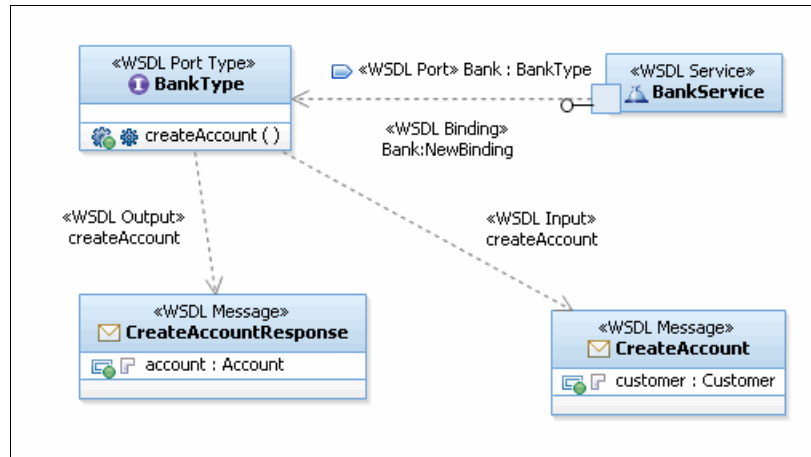


Figure 6-30 Class diagram showing a binding between a port and its port type

2. Generate the content for this binding:
 - a. Select the dependency arrow representing the binding and open the **Properties** view.
 - b. On the General page, click **Generate Binding Content** and complete the wizard (Figure 6-31).

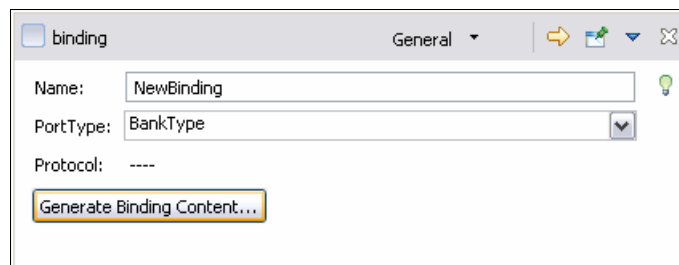


Figure 6-31 Properties view showing binding properties

After you create the entire web service, you can directly create an implementation of the web service within the diagram editor. Right-click a **WSDL Service** component and select **Implement Web Service**, which starts the Web Service wizard that guides you through the process. You can also use the

diagram editor to create a web service client for a given web service. To do this, right-click a **WSDL Service** component and select **Consume Web Service**.

6.3.5 Class diagram preferences

With Rational Application Developer, you can review and edit default settings or preferences that affect the appearance and behavior of UML class diagrams and their content. These preferences are organized under the Modeling node within the Preferences window (select **Window** → **Preferences**).

Before you create a new UML class diagram, you can set the default global preferences for attributes and operations, such as visibility styles, showing or hiding attributes and operations, showing or hiding operation signatures, and showing or hiding parent names of classifiers. Most configuration settings are present under the following Preferences window nodes:

- ▶ UML diagrams

With the UML diagrams node and the nodes beneath it, such as `Class` and `Component`, you can specify several preferences regarding the style, fonts, and colors that are displayed in UML diagrams when they are created. You can change the default settings for showing or hiding attributes, operations, operation signatures, or parent names of classifiers. You can also specify which compartments are shown, by default, when a new UML element is created.
- ▶ Java

You can use this node and the nodes beneath it to specify settings that deal with Java code when it is used in a UML model. One example is the configuration of corresponding wizards to use when new fields or methods are created within a class diagram.

 - The settings for the configuration of wizards are under *Field and Method Creation*. You can also specify the default values to apply to these wizards.
 - *Show Related Elements Filters* provides the option to filter out binary Java types when the Show Related Elements action is executed. Binary Java types are types that are not defined in the workspace, but that are available to the workspace through referenced JAR libraries.
- ▶ EJB and EJB 3.0

You can use the EJB and EJB 3.0 preferences nodes to specify if newly generated or created EJB and EJB 3.0 components are visualized in a selected class diagram. If you select this option and a diagram is not selected while creating or generating a new bean, this bean is opened in a *default* class diagram.

- ▶ Web Service

With the Web Service node, you can change the default settings for visually representing existing WSDL elements. You can specify if the external or compressed view of an existing WSDL element is shown. Furthermore, you can specify which WSDL components are visually represented in class diagrams. To show or hide WSDL components, select or clear the corresponding check boxes on this page.

6.4 Exploring relationships in applications

Rational Application Developer provides browse and topic diagrams that can be used to explore and navigate through an application and to view the details of its elements and relationships. They are designed to assist developers in understanding and documenting existing code by quickly creating UML representations of an existing application.

6.4.1 Browse diagrams

A *browse diagram* is a structural diagram that provides a view of a context element, such as a class, a package, or an EJB. It is a temporary read-only diagram that provides the capability to explore the structure and relationships of the given context element. You can view the element details, including attributes, methods, and relationships to other elements, and you are able to navigate to those elements. Browse diagrams can be applied to various elements, including Java classes and EJB components, but excluded are all elements related to web services.

You can create a browse diagram from any source element or its representation within a class diagram. To create a browse diagram, right-click the desired element and select **Visualize** → **Explore in Browse Diagram**. A browse diagram is created and shown in the corresponding diagram editor. The diagram editor consists of a panel displaying the selected element along with its relationships and a toolbar. Because a browse diagram is not editable, the palette and the modeling assistant are not available. Depending on the elements shown, the diagram is displayed either using the radial or generalization tree layout type. The radial layout type shows the selected element in the center of the diagram, whereas the generalization tree layout type organizes the general classes at the top of the diagram and the subclasses at the bottom.

The browse diagram acts in a similar manner to a web browser for your code. It provides a history and navigation, and you can customize the UML relationships that you want to see. The toolbar located at the top of the browse diagram shows

the context element that you are currently browsing. At any one time, there is only one browse diagram open. When you browse another element, it is displayed in the same diagram, replacing the previous element.

Figure 6-32 shows an example browse diagram with the Java class `DatabaseManager` as the context element. You can see all the attributes and methods declared by this class. In this case, the dependency filter button is the only one highlighted, and so elements involved in a dependency relationship with `DatabaseManager` are shown as well. You can see that `AccountDAO` and `CustomerDAO` both depend on `DatabaseManager`.

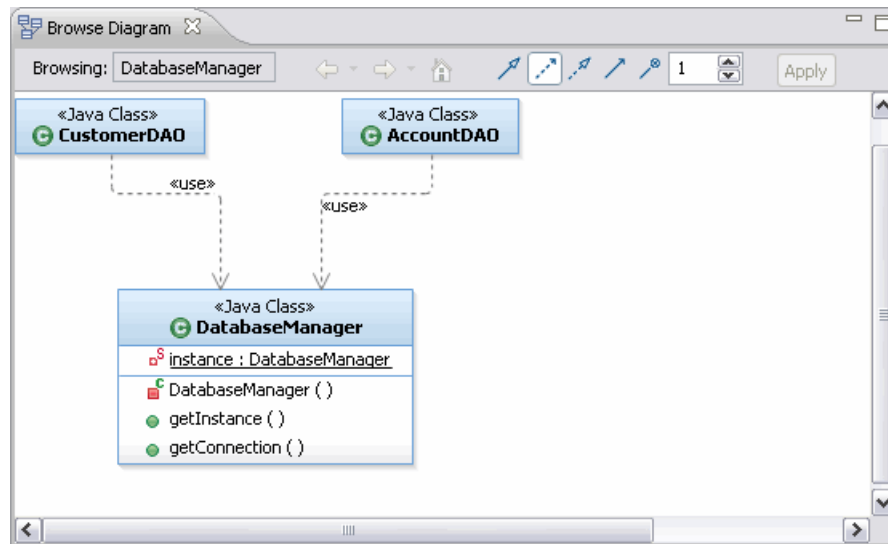


Figure 6-32 Browse diagram example

The browse diagram retains the history of elements that you have viewed so far. You can use the two arrow buttons provided in the toolbar to navigate backward or forward to browse previously viewed elements.

When you click the **Home** icon (🏠), the first element in the history is displayed. Furthermore, the toolbar contains a list of filter icons that can be used to filter the types of relationships that are shown along with the context element. Separate filters are available, depending on the type of element that you are currently browsing, for example, Java class or EJB. To enable or disable a filter, click the appropriate icon and then click **Apply**.

You can also change the number of levels of relationships that are shown for the context element. The default value is one. To change this value, specify a number and click **Apply**.

In Figure 6-32, the **Home** icon and the two arrow icons are disabled, so the current element is the first element in the browse diagram history.

If you want to explore the details of a diagram element, double-click it. This element becomes the new context element. When you right-click a diagram element, the **Navigate** submenu provides several options, such as opening the Java source of a diagram element.

A browse diagram cannot be changed or saved, but Rational Application Developer lets you save any browse diagram view as a diagram file that is fully editable. Right-click an empty space inside a browse diagram and select **File** → **Save as Diagram File**. If you want to use a browse diagram as part of permanent documentation, you can save a browse diagram view as an image file using **File** → **Save as Image File**.

6.4.2 Topic diagrams

Topic diagrams provide another way to create structural diagrams from the code in your application. They are used to quickly create a query-based view of relationships between existing elements in your application. These queries are called *topics* and represent commonly required views of your code, such as showing the super type or subtypes of a given class. Topic diagrams are applicable to various elements, such as Java classes, EJB components, or WSDL files. Similar to browse diagrams, topic diagrams are not editable, but you can save them as editable UML diagrams or as images and share them with other team members.

A new topic diagram of an application element is created by the Topic Diagram wizard. To start this wizard, right-click the desired element in the Enterprise Explorer view and select **Visualize** → **Add to New Diagram File** → **Topic Diagram**. After the wizard has started, on the Topic Diagram Location page, enter or select the parent folder and provide a name for the file, as shown in Figure 6-33 on page 209. Then click **Next**.

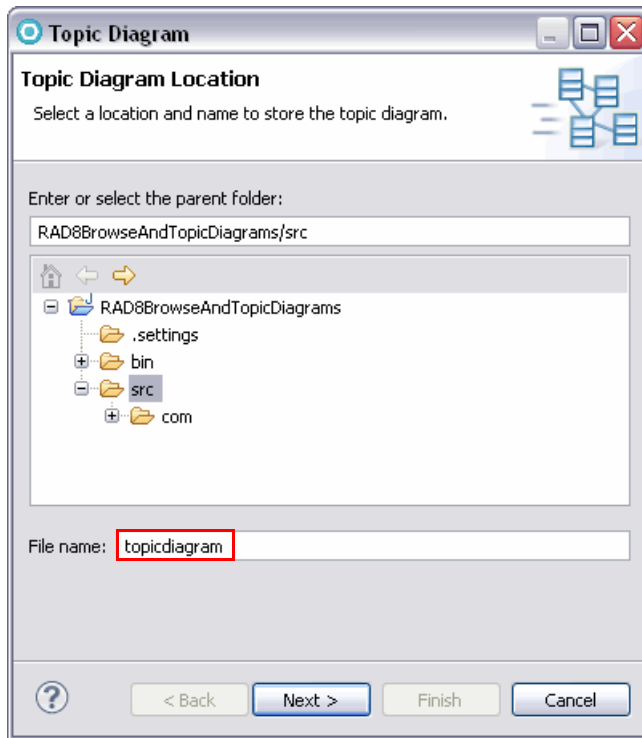


Figure 6-33 Topic Diagram wizard (part 1 of 3)

The Topics page shown in Figure 6-34 provides a list of standard topics Rational Application Developer can create. Select a predefined query and click **Finish**. A new topic diagram is created based on the default values that are associated with the selected topic.

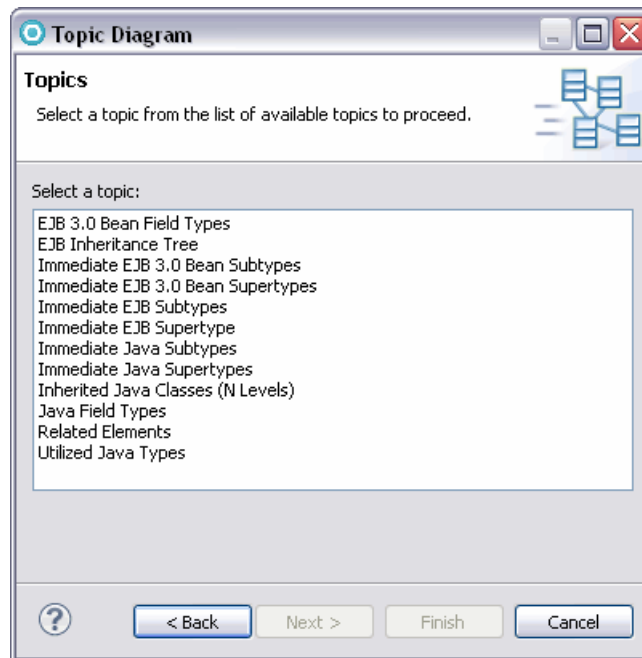


Figure 6-34 Topic Diagram wizard (part 2 of 3)

If you want to review or change these values, click **Next** instead. The Related Elements page (Figure 6-35 on page 211) opens. This page shows the details of the previous selected topic and allows you to change these values.

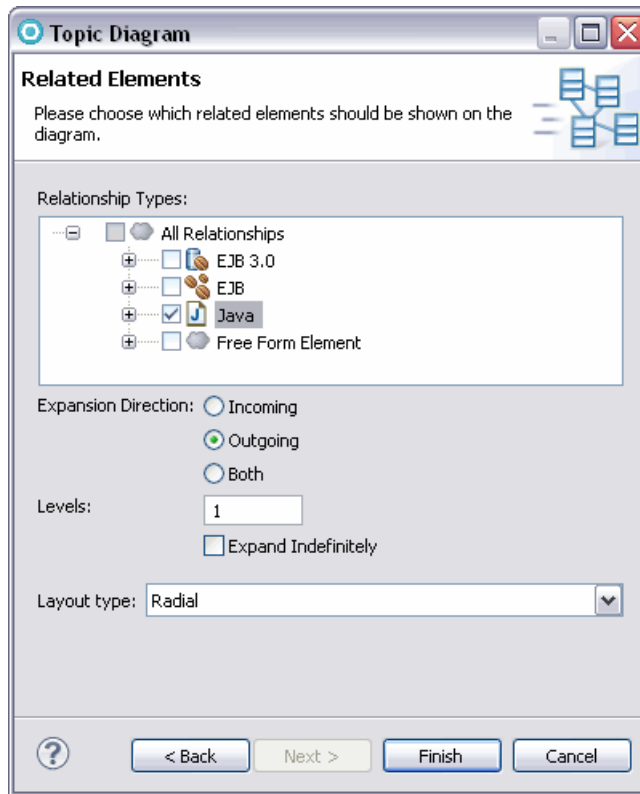


Figure 6-35 Topic Diagram wizard (part 3 of 3)

You can select various types of relationships to include in the query along with the expansion direction:

- ▶ You can select **Incoming** to see all elements that are shown that are related to the context element.
- ▶ You can select **Outgoing** to see all elements with which the context element has a relationship.
- ▶ You can further specify the number of levels of relationships to query and the layout type for the diagram. The possible values are **Default** and **Radial**. These values map to the generalization and radial tree layout types described previously.

After a topic diagram is created, you can review or change the underlying query. To do this, right-click the empty space inside the topic diagram and select **Customize Query**.

Similar to browse diagrams, topic diagrams are not editable, so the tool palette and the modeling assistant are not available. You can save a topic diagram as an editable diagram or as an image as we did previously with browse diagrams. To do this, right-click the empty space in a topic diagram and select either **File** → **Save as Diagram File** or **File** → **Save as Image File**.

The query and the context element that you have specified are persisted in the topic diagram. Every time that you open a topic diagram, the underlying elements are queried and the diagram is automatically populated with the current results. If you make changes to the underlying elements when a topic diagram is already open, the diagram might not represent the current status of the elements until you refresh the diagram manually. To do this, right-click the empty space in the topic diagram and select **Refresh**.

6.5 Describing interactions with UML sequence diagrams

Rational Application Developer provides the capability to develop and manage sequence diagrams. A sequence diagram is an interaction diagram that can be used to describe the dynamic behavior of a system. It depicts the sequence of messages that are sent between objects in a certain interaction or scenario.

You can use sequence diagrams at various stages during the development process:

- ▶ Within the analysis phase, you can use a sequence diagram to describe the realization of a use case that is a use case scenario.
- ▶ Within the design phase, you can refine sequence diagrams more to show how a system accomplishes an interaction and, in this case, to show the objects of actual design classes interacting.

A sequence diagram consists of a group of objects, their associated lifelines, and the messages that these objects exchange over time during the interaction. In this context, the term *object* does not necessarily refer to software objects instantiated from a class. An object represents any structural component defined by UML.

Figure 6-36 on page 213 provides a sample sequence diagram. It describes the scenario where a customer wants to withdraw cash from an automated teller machine (ATM). A sequence diagram has a two-dimensional nature. The horizontal axis shows each of the objects that are involved in an interaction. The vertical axis shows the lifelines, the messages exchanged, and the sequence of creation and destruction of the objects.

Most objects that are displayed in a sequence diagram are in existence for the duration of the entire interaction, so their lifelines are placed at the top of the diagram. Objects can be created or destroyed during an interaction. In this case, their lifelines start or end with the receipt of a corresponding message to create or destroy them.

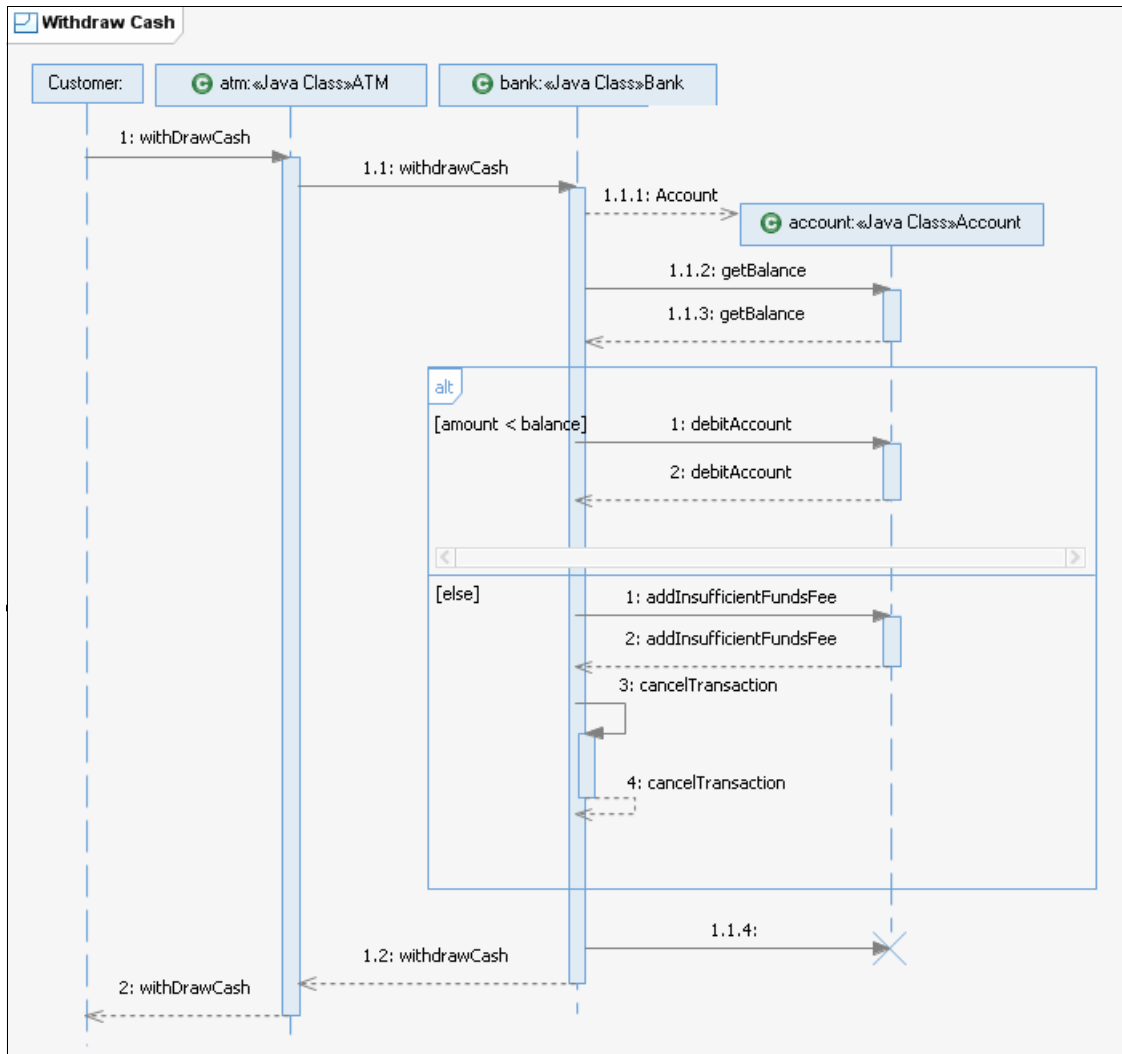


Figure 6-36 Sample sequence diagram

The major focus of Rational Application Developer when using sequence diagrams is to document and visualize the dynamic behavior of a system, rather than to develop source code. The tool enables developers to create, edit, and

delete the various elements of a sequence diagram, such as lifelines, messages, and combined fragments in a visual manner. In contrast to a class diagram, the elements of a sequence diagram do not need to be related to existing elements, such as classes or interfaces. Therefore, changes that are made in a sequence diagram do not affect any code. Rational Application Developer does, however, allow existing classes, present in source code, to be used as the type of the lifelines shown on the sequence diagram. In this case, messages between lifelines can be selected from existing methods on the class or new methods created as required. Any new methods created when adding a message result in a new method being added to the class source code.

In Figure 6-36 on page 213, the object identified as atm is of class ATM, and ATM is a Java class with associated source code. The object identified as customer does not have an associated Java class. Sequence diagrams can be drawn showing objects where every object class is defined and the sequence diagram is a dynamic view of existing code. Diagrams can also be drawn with objects, none of which has a defined class, and the sequence diagram has been drawn for a purpose other than visualizing existing code. Diagrams can also contain a mixture of both as is the case with Figure 6-36 on page 213.

Rational Application Developer has two types of sequence diagrams. The first type of sequence diagram is the type previously described. The second type of sequence diagram is referred to as a *Static Method Sequence Diagram*. This non-editable diagram is used to visualize the flow of messages between existing Java objects in an executing application.

6.5.1 Creating sequence diagrams

To create a new sequence diagram, you must use the New Sequence Diagram wizard. You can start this wizard either by selecting **File** → **New** → **Other** → **Modeling** → **Sequence Diagram** directly from the top menu of Rational Application Developer or from within the Explorer from the context menu of any resource, such as projects or packages.

You can also create a new sequence diagram populated with existing classes or interfaces. Any Java class or interface can be used on a sequence diagram, including servlet classes and EJB classes and interfaces. To do this in the Explorer, right-click the desired source element and select **Visualize** → **Add to New Diagram File** → **Sequence Diagram**.

After the wizard starts, you provide a name for the file that will be created to contain the content of the diagram, and you specify the parent folder to store this file. Clicking **Finish** completes the process and creates a new sequence diagram.

A sequence diagram has a corresponding diagram editor and palette on the right side, offering tools that can be used to add new elements to the diagram, such as lifelines, messages, or combined fragments. Also, there are two items in the tool palette where a solid triangle is shown right next to the item. When you click this triangle, a context menu is displayed that allows you to select another tool from this category, which, for example, is the case for Synchronous Message.

A sequence diagram is enclosed in a frame. A diagram frame provides a visual border and enables the diagram to be easily reused in another context. The frame is depicted as a rectangle with a notched descriptor box in the top left corner that provides a place for the diagram name. If you want to change the name, select this box and enter the new name.

6.5.2 Creating lifelines

A *lifeline* represents the existence of an object involved in an interaction over a period of time. A lifeline is depicted as a rectangle representing the object involved in the interaction, which contains the object's name, type, and stereotype with a vertical dashed line beneath indicating the progress of time.

Figure 6-37 on page 216 shows several examples of possible lifelines for objects of a class called `Customer`.

Important: The terminology that is used with sequence diagrams differs in UML 2 when compared with earlier versions of UML.

Also, text in a lifeline object box does not have to be underlined although it can be rendered that way, if required.

Note the following explanation of Figure 6-37 on page 216:

- ▶ The first lifeline represents an instance of the `Customer` Java class, and the instance is named `customer`.
- ▶ The second lifeline represents an anonymous instance of the `Customer` Java class. The instance has no name. It can also represent an object with a type but with the name hidden.
- ▶ The third lifeline represents an object named `customer` with a type but the type is hidden.
- ▶ The last lifeline represents an object named `customer` with no allocated type.

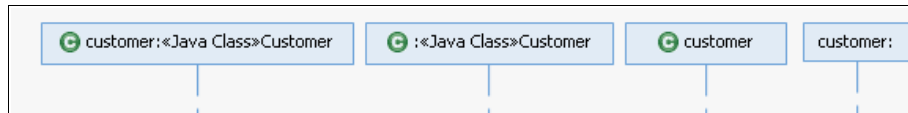


Figure 6-37 Various representations of lifelines on a sequence diagram

To add a lifeline from an existing Java class or interface to a sequence diagram, select the desired element in the Explorer view and drag it on an empty place in the diagram. This action creates a new lifeline and places it at the top of the diagram, aligned horizontally with the other lifelines. If you drag another class over the top of an existing lifeline on the sequence diagram, the class of the lifeline changes to the new class.

You can also use the tool palette to create a new lifeline. Select **Lifeline** in the palette and drop it on an empty space inside the diagram, which creates a lifeline representing an object whose type is not specified but with a default name. After a lifeline is created, you can change the name and type of the object that it represents.

If you want to change the name, select the lifeline's shape and enter the new name. If you want to change the type and a class or interface is available in the Explorer, select the desired class or interface in the Explorer and drag it on the lifeline's shape. You can also use the Properties view to review or change any property of a given lifeline.

By default, a lifeline is shown as a rectangle containing the lifeline name, type, and stereotype. If you right-click a lifeline, the **Filters** submenu provides several options to change the lifeline's appearance.

6.5.3 Creating messages

A message describes the communication that occurs between two lifelines. A message is sent from a source lifeline to a target lifeline to initiate an action or behavior, such as invoking an operation on the target, or the creation or destruction of a lifeline. The target lifeline often responds with a further message to indicate that it has finished processing.

A message is visualized as a labeled arrow that originates from the source lifeline and ends at the target lifeline. The message is sent by the source and received by the target and the arrow points from source to target. The label is used to identify the message. It contains either a name or an operation signature if the message is used to call an operation. The label also contains a sequence number that indicates the ordering of the message within the sequence of messages.

To create a message between two lifelines, hover the mouse pointer over the source lifeline so that the modeling assistant is available. Click the small box at the end of the outgoing arrow and drag the resulting connector on the desired target lifeline. In the context menu that is displayed when you drop the connector on the target, click the desired message type, such as synchronous or asynchronous, and either enter a name or select an operation from the list. You can only select an operation if the target already has available operations.

You can also use the tool palette to create a message. Select the desired message type by clicking the solid triangle next to the Message category. Then click the source lifeline and drag the cursor to the target lifeline. The following message types are available:

- ▶ By selecting **Create Message** from the palette, the source lifeline can create a new lifeline. The new lifeline starts when it receives this message. The symbol at the head of this lifeline is shown at the same level as the message that caused the creation (Figure 6-38). The message itself is visualized as a dashed line with an open arrowhead. This type of message is used to highlight that a new object is created during an interaction. One of the existing constructor operations in the CustomerTO class has been chosen for the message name and the message has been configured to show its signature to make it clear which constructor is being used.

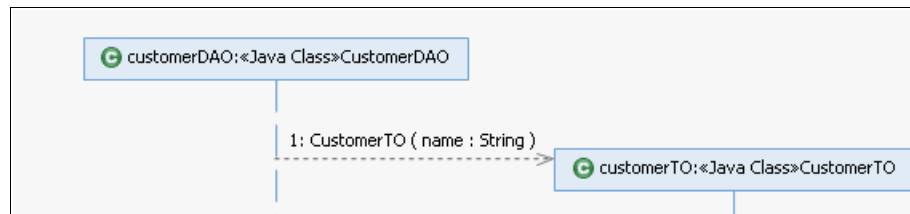


Figure 6-38 Sending a create message to create a new lifeline

- ▶ In contrast, a *destroy message* enables a lifeline to delete an existing lifeline. The target lifeline is terminated at that point when it receives the message. The end of the lifeline is denoted using the stop notation, which is a large X (Figure 6-39 on page 218). A destroy message is drawn in a similar way to a create message. You can use this type of message to describe that an object is destroyed during an interaction. After a lifeline has been destroyed, it cannot be the target of any messages. In this example, the message has not been given a name because the CustomerTO class does not contain an appropriate operation that can be called.



Figure 6-39 Destroying a lifeline during an interaction

- ▶ A *synchronous message* enables the source lifeline to invoke an operation provided by the target lifeline. The source lifeline continues and can send more messages only after it receives a response from the target lifeline. When you create a synchronous message, Rational Application Developer places three elements in the diagram (Figure 6-40):
 - A solid line representing a synchronous operation invocation
 - A dashed line representing the return message
 - A thin rectangle called an *activation bar* or *execution occurrence* representing the behavior performed

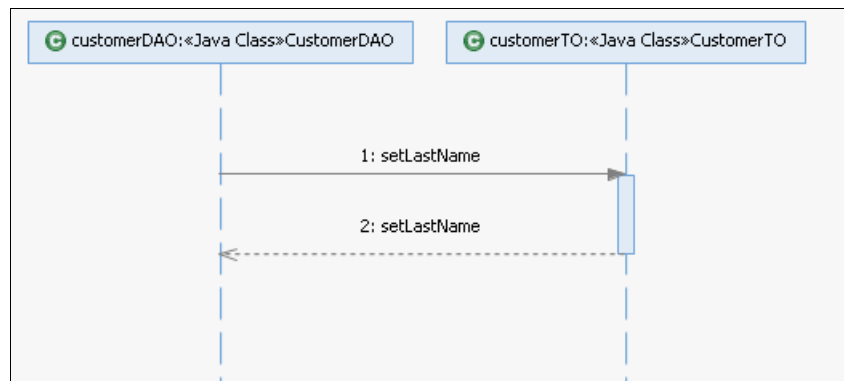


Figure 6-40 Synchronous message invocation

By default, only the operation's name is shown. If you want to see the full operation signature, right-click the message arrow and select **Filters** → **Show Signature**.

- ▶ An *asynchronous message* allows the source lifeline to invoke an operation provided by the target lifeline. The source lifeline can then continue and send more messages without waiting. When an asynchronous message is sent, the source does not have to wait until the target processes it. An asynchronous message is drawn similarly to a synchronous message, but the line is drawn with an open arrowhead, and the response is omitted. You can

also send another asynchronous message, the *asynchronous signal message*, which is a special form of a message that is not associated with a particular operation.

6.5.4 Creating combined fragments

UML 2 introduced the concept of combined fragments to support conditional and looping constructs, such as if-then-else statements, or to enable parts of an interaction to be reused.

Combined fragments are frames that encompass portions of a sequence diagram or provide reference to other diagrams.

A combined fragment is represented by a rectangle that comprises one or more lifelines. Its behavior is defined by an interaction operator that is drawn as a notched descriptor box in the upper-left corner of the combined fragment. For example, the alternative interaction operator (*alt*) acts like an if-then-else statement and is shown in Figure 6-41.

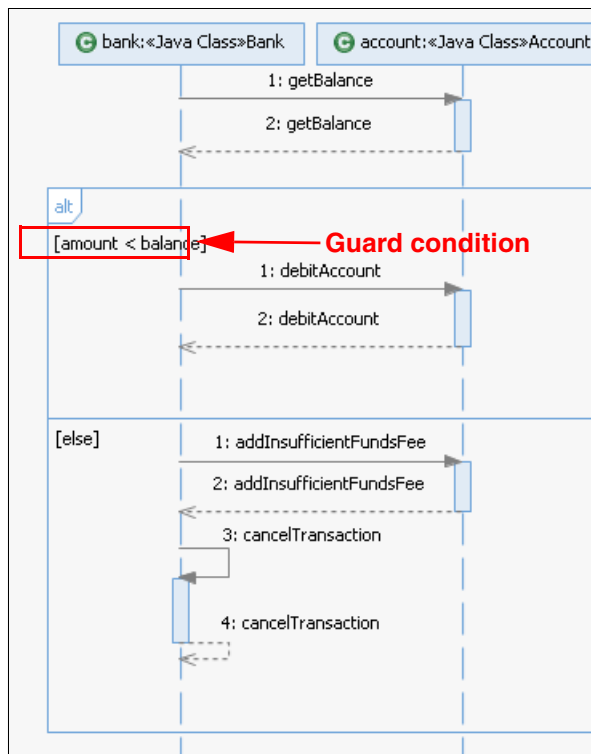


Figure 6-41 Sequence diagram with an alternative combined fragment

UML 2 provides many other interaction operators for use with combined fragments. Depending on its type, a combined fragment can have one or more interaction operands. Each interaction operand represents a fragment of the interaction with an optional *guard condition*. The interaction operand is executed only if the guard condition is true at run time. The absence of a guard condition means that the combined fragment is always executed. The guard condition is displayed as plain text enclosed within two square brackets. A combined fragment separates the contained interaction operands with a horizontal line between each operand within the frame of the combined fragment. When the combined fragment contains only one operand, the line is unnecessary.

Figure 6-41 on page 219 shows a fragment being used in a withdraw cash interaction. The combined fragment is used to model an alternative flow in the interaction. Because of the guard condition `[amount<balance]`, if the account balance is greater than the amount of money that the customer wants to withdraw, the first interaction operand is executed. This means the interaction debits the account. Otherwise, the `[else]` guard forces the second interaction operand to be executed. An insufficient fund fee is added to the account and the transaction is canceled.

To create a combined fragment, you must first select the desired fragment in the palette. Click the solid triangle next to the **Combined Fragment** category and select the desired fragment from the available fragments. Then click the left mouse button within an empty place in the diagram and drag the combined fragment across the lifelines that you want to include in it. When you release the mouse button, the Add Covered Lifelines window opens, in which you can select the individual lifelines to be covered by the combined fragment. Each lifeline is represented by a check box, and each of them is selected, by default. When you click **OK**, a new combined fragment along with one or two interaction operands is created.

Figure 6-42 shows a newly created alternative combined fragment with two empty interaction operands. If you want to specify a guard condition for an interaction operand, select the corresponding brackets and enter the text. You can create messages between the individual lifelines covered by the combined fragment in the same way as described previously. Notice how the sequence numbers of the individual messages change within an interaction operand. You can also nest other combined fragments within an existing combined fragment.

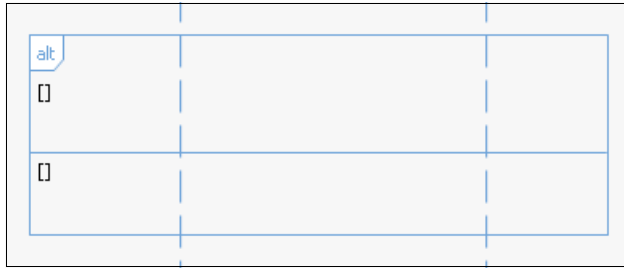


Figure 6-42 Empty combined fragment with two interaction operands

After a combined fragment has been created, it is not possible to change its type, or more precisely, its interaction operator. But if you right-click a combined fragment, the context menu allows you to add new interaction operands if you select **Add Interaction Operand** or to add and remove lifelines from the selected element if you select **Covered Lifelines**.

When you create an interaction operand, it is displayed in an expanded state. By clicking the small triangle at the top of the interaction operand, you can collapse it to hide the entire operand and its associated messages. From the context menu of an operand, several options are available to remove or reposition the selected operand. Further, you can add a guard condition to the operand or add a new interaction operand if the enclosing combined fragment allows multiple operands.

6.5.5 Creating references to external diagrams

UML 2 provides the capability to reuse interactions that are defined in another context. This function provides you the ability to create complex sequence diagrams from smaller and simpler interactions. A reference to another diagram is modeled using the *Interaction Use* element. Like a combined fragment, an Interaction Use element is represented by a frame. The operator `ref` is placed inside the descriptor box in the upper-left corner, and the name of the sequence diagram being referenced is placed inside the frame's content area along with any parameters for the sequence diagram.

An interaction use can encompass a single activation bar or several lifelines. Follow these steps to create a reference to another sequence diagram:

1. Select **Interaction Use** in the palette and place the cursor on an empty space inside the source sequence diagram.
2. When you drop the cursor, at the prompt, choose the current lifelines that will be covered.

3. In the Add Covered Lifelines window that opens, select the lifelines to encompass by the Interaction Use element and click **OK**.

In Figure 6-43 on page 222, the Interaction Use element references an interaction called `sequencediagram7`, which provides further information about how the `withdrawCash` operation is realized.

Diagram in Figure 6-43: At the time of writing, it was not possible to reference another diagram. It was only possible to provide a simple name.

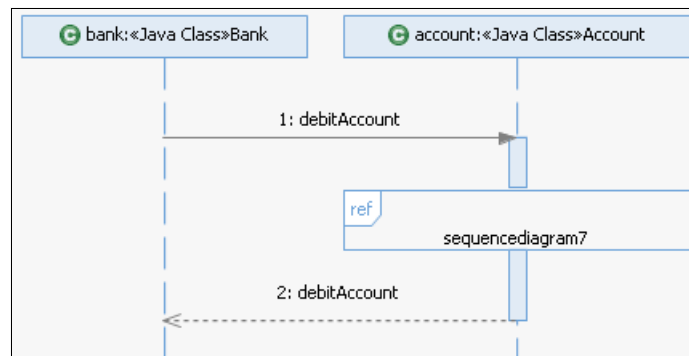


Figure 6-43 Creating a reference to another diagram

6.5.6 Exploring Java methods with static method sequence diagrams

With the static method sequence diagram feature that is provided by Rational Application Developer, developers can visualize a Java method. Existing Java code can quickly be rendered in a sequence diagram to visually examine the behavior of an application. A static method sequence diagram from a Java method provides the full view of the entire method call sequence.

To create a static method sequence diagram for a Java method, right-click the desired method in the Explorer view and select **Visualize** → **Add to New Diagram File** → **Static Method Sequence Diagram**. The diagram is created and shown in the corresponding diagram editor.

A static method sequence diagram is a topic diagram, so the diagram content is stored in a file with a `.tpx` extension. Like other topic diagrams, it is read only; the tool palette, the toolbar, and the modeling assistant are not available.

When you right-click an empty space inside the diagram, the **File** submenu provides you the options to either save the diagram as an image using **Save as**

Important: At the time of writing, Rational Application Developer failed to update the contents of a static sequence diagram when the source code was changed. One work-around is to close the diagram and the project containing the source code of the visualized method, and then reopen the project and subsequently the diagram.

6.5.7 Sequence diagram preferences

Using the Sequence and Communication node in the Preferences window, you can change default values that affect the appearance of sequence diagrams (Figure 6-45 on page 225). For example, you can specify that return messages are created automatically or to ask for message numbering to be shown.

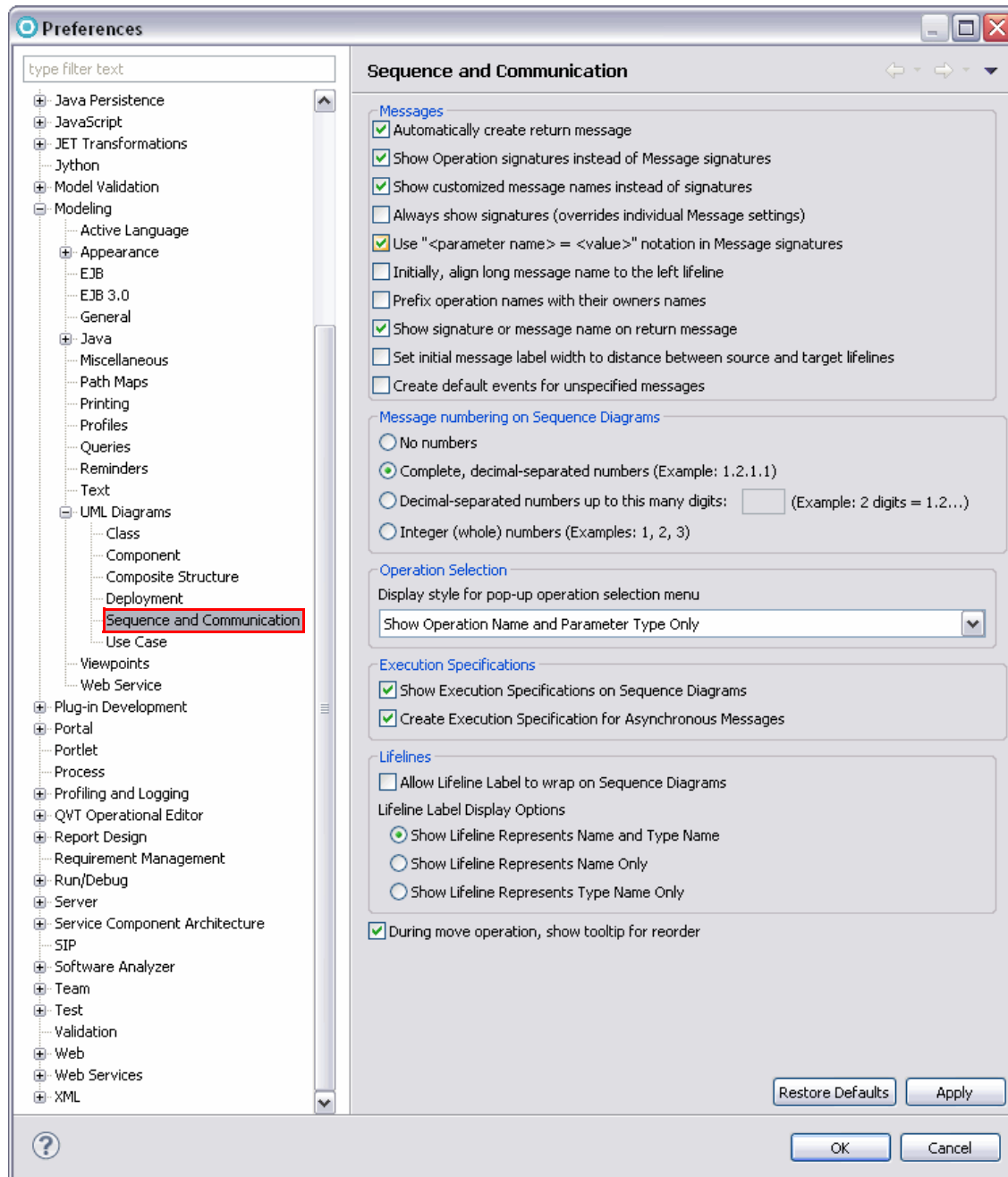


Figure 6-45 Sequence diagram preferences

6.6 More information about UML

For more information about UML, consider the following resources. These websites provide information about modeling techniques, best practices, and UML standards:

- ▶ IBM developerWorks for Rational

This website provides guidance and information that can help you implement and deepen your knowledge of Rational tools and leading practices. The site includes access to white papers, artifacts, source code, discussions, training, and other documentation:

<http://www.ibm.com/developerworks/rational/>

In particular, we highlight the following series of high-quality Rational Edge articles focusing on UML topics:

<http://www.ibm.com/developerworks/rational/library/content/RationalEdge/archives/uml.html>

- ▶ IBM Rational Software UML Resource Center

IBM Rational Software UML Resource Center is a library of UML information and resources that IBM continues to build upon and update. In addition to current news and updates about the UML, you can find UML documentation, white papers, and learning resources:

<http://www-01.ibm.com/software/rational/uml/index.html>

- ▶ Object Management Group (OMG)

The following websites provide formal specifications for UML that have been adopted by the OMG and are available in either published or downloadable form, and technical submissions on UML that have not yet been adopted:

<http://www.omg.org>

<http://www.uml.org>

- ▶ Craig Larman's home page

This website provides articles and related links about topics regarding UML:

<http://www.craiglarman.com/>



Part 2

Java and XML development

In this part, we describe the tooling and technologies provided by Rational Application Developer to develop applications using Java and XML.

This part includes the following chapters:

- ▶ Chapter 7, “Developing Java applications” on page 229
- ▶ Chapter 8, “Developing XML applications” on page 331

Sample code for download: The sample code for all the applications developed in this part is available for download at the following address:

<ftp://www.redbooks.ibm.com/redbooks/SG247835>

See Appendix C, “Additional material” on page 1877, for instructions.



Developing Java applications

In this chapter, we introduce the Java development capabilities and tooling features of IBM Rational Application Developer by developing the ITSO Bank application.

The chapter is organized into the following sections:

- ▶ Java perspectives, views, and editor overview
- ▶ Developing the ITSO Bank application
- ▶ Understanding the sample code
- ▶ Java editor and rapid application development

Java SE and Java Runtime Environment (JRE): Rational Application Developer fully supports Java Standard Edition (SE) 6.0. However, you can download, install, and use newer JRE versions in Rational Application Developer, as described in 7.7.1, “Pluggable Java Runtime Environment” on page 292.

The sample code for this chapter is in the 7835code\java folder.

7.1 Java perspectives, views, and editor overview

Within Rational Application Developer, the following predefined perspectives, which contain the views and the editor, are most commonly used when developing Java SE applications:

- ▶ Java perspective
- ▶ Java Browsing perspective
- ▶ Java Type Hierarchy perspective

We briefly introduce these perspectives and their major views in Chapter 4, “Perspectives, views, and editors” on page 91. In this section, we go deeper into the details and describe several more useful views. The highlighted areas in Figure 7-1 on page 231 indicate all perspectives and views that we discuss.

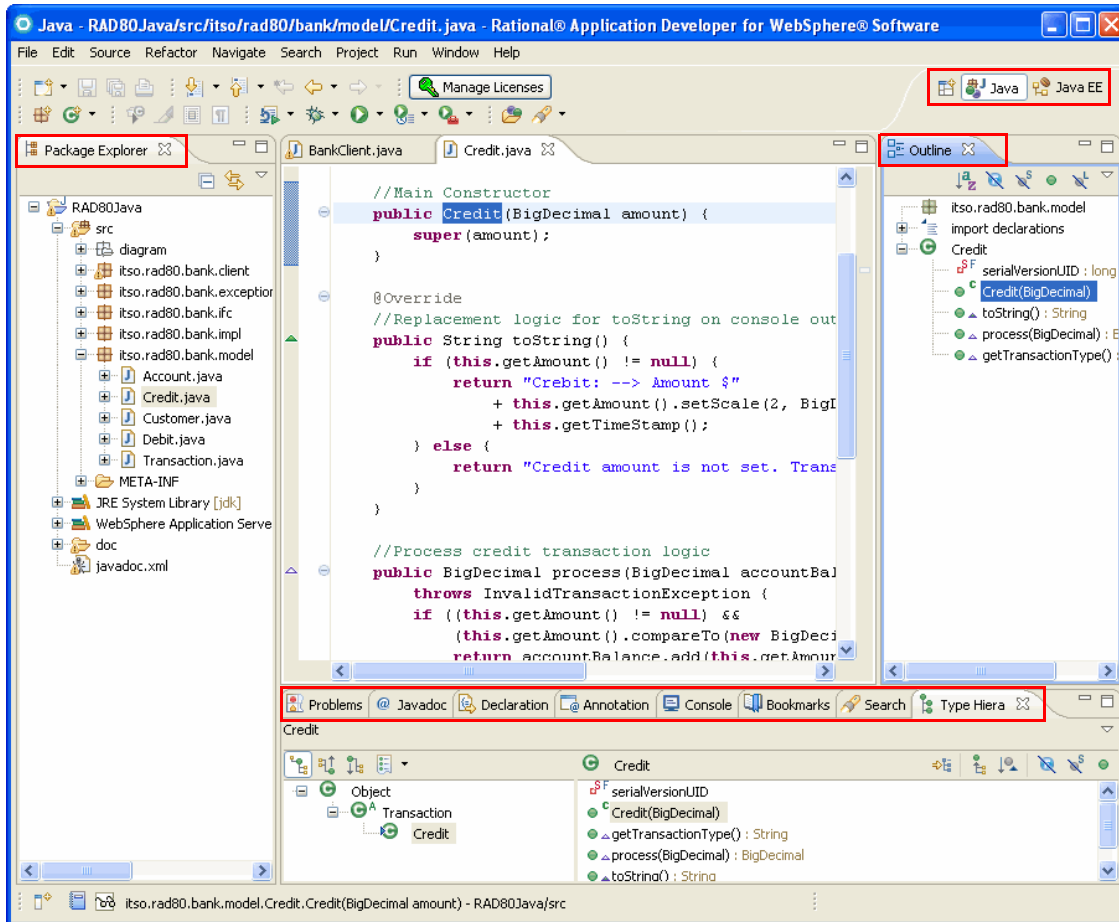


Figure 7-1 Views in the Java perspective (customized)

Customizing the perspective: Figure 8-1 shows a customized Java perspective. It is the predefined Java perspective with more useful views added. In your environment, customize the perspectives so that they fit your requirements. You can save a customized perspective by selecting **Window** → **Save Perspective As**. To return a modified perspective to a predefined or saved perspective, select **Window** → **Reset Perspective**.

7.2 Java perspective

We use the Java perspective to develop Java SE applications or utility Java projects (utility JAR files) containing code that is shared across multiple modules within an enterprise application. You can add views by selecting **Window** → **Show View**.

7.2.1 Package Explorer view

The Package Explorer view shows all projects, packages, interfaces, classes, member variables, and member methods in the workspace, as shown in Figure 7-2. With this view, you can easily navigate through the workspace.

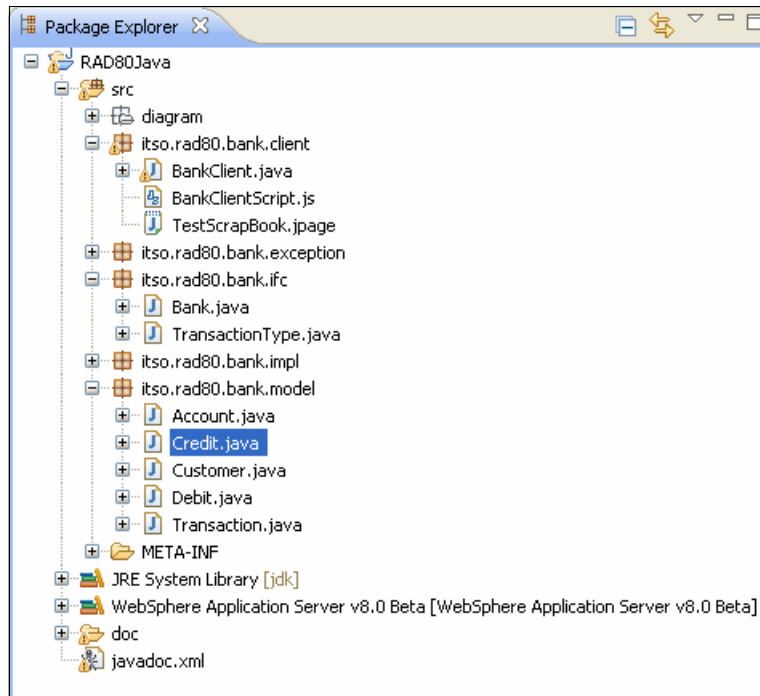


Figure 7-2 Package Explorer view

Viewing files and folders: In this view, you cannot see the generated `.class` files. To view the folders and files as they are in the file system, select **Window** → **Show View** → **Navigator** to open the Navigator view. Now you can see the source code in the `src` directory and the byte code files in the `bin` directory.

7.2.2 Hierarchy view

We use the Hierarchy view to display the type hierarchy of a selected type.

To view the hierarchy of a class type (Figure 7-3), select the class in the Package Explorer, and press F4 or right-click the class and select **Open Type Hierarchy**.

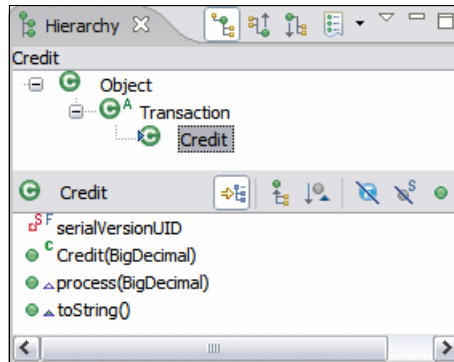


Figure 7-3 Hierarchy view for a selected class

The Hierarchy view provides the following hierarchy layouts:

- ▶ Type Hierarchy (📁): All supertypes and subtypes of the selected type are shown.
- ▶ Supertype Hierarchy (📁): Only all supertypes of the selected type are shown.
- ▶ Subtype Hierarchy (📁): Only all subtypes of the selected type are shown.

In addition, the Hierarchy view has the following other options:

- ▶ This option (🔒) locks the view and shows the members in the hierarchy. For example, use this option if you are interested in all types implementing the `toString()` method.
- ▶ This option (📁) shows all inherited members.
- ▶ This option (📁) sorts members by their defining types. Defining type is displayed before the member name.
- ▶ These options (🔍) filter the displayed members.

7.2.3 Outline view

The Outline view is useful and is the recommended way to navigate through a type that is currently opened in the Java editor. It lists all of the elements, including package, import declarations, type, fields, and methods. You can use

the icons highlighted in Figure 7-4 to sort and filter the elements that are displayed.



Figure 7-4 Outline view

7.2.4 Problems view

While editing resource files, various builders can automatically log problems, errors, or warnings in the Problems view. For example, when you save a Java source file that contains syntax errors, those syntax errors will be logged as errors, as shown in Figure 7-5 on page 235. When you double-click the problem icon (⊗), error icon (⊗), or warning icon (⚠), the editor for the associated resource automatically opens to the relevant line of code.

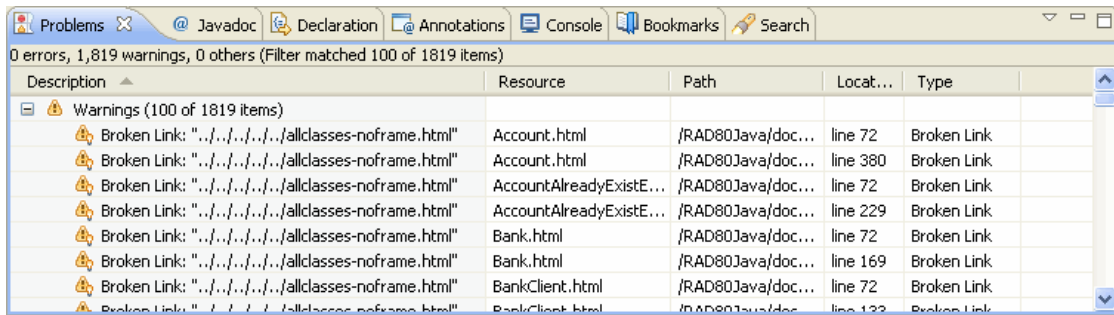


Figure 7-5 Problems view with warnings notification

Rational Application Developer provides a quick fix for certain problems. To process a quick fix, see 7.9.9, “Quick fix” on page 315.

Enabling and disabling builders: The Java builder is responsible for all Java resource files. However, in an enterprise application project, you can use other builders. Builders can be enabled and disabled for each project. Right-click the project in the Package Explorer, select **Properties** and then select **Builders**.

In the Problems view, you can filter the problems to show only specific types of problems by clicking the **View Menu** icon (☰), which opens a menu from which you can sort the content or select the Configure Contents window (Figure 7-6 on page 236).

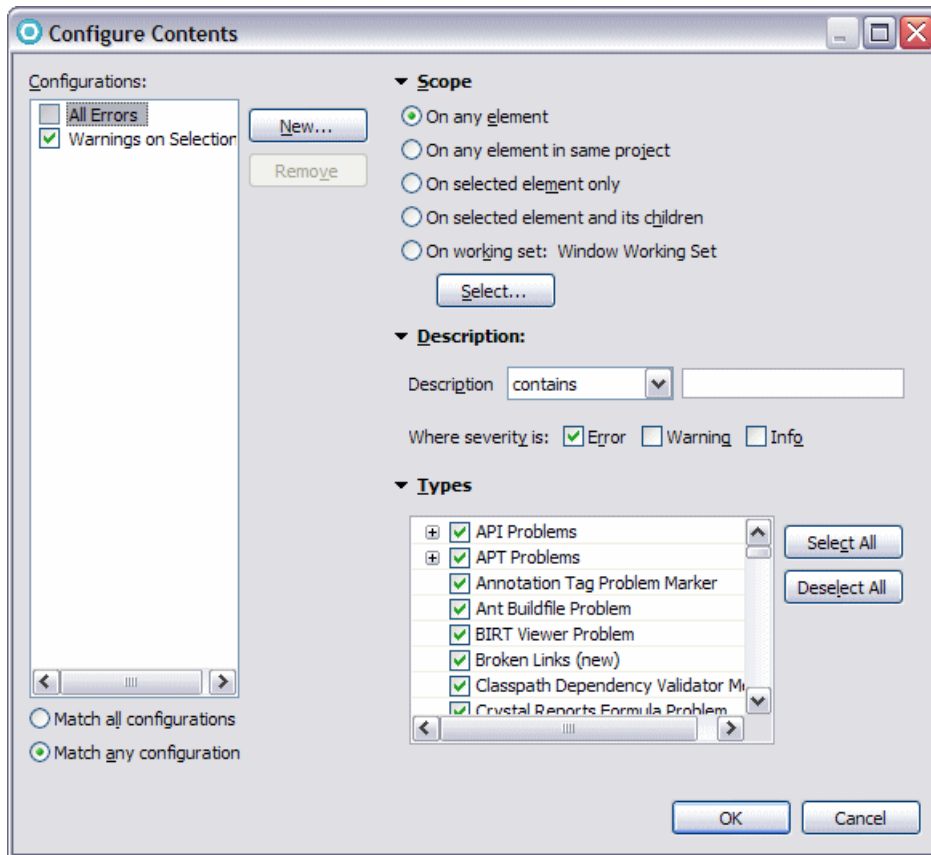

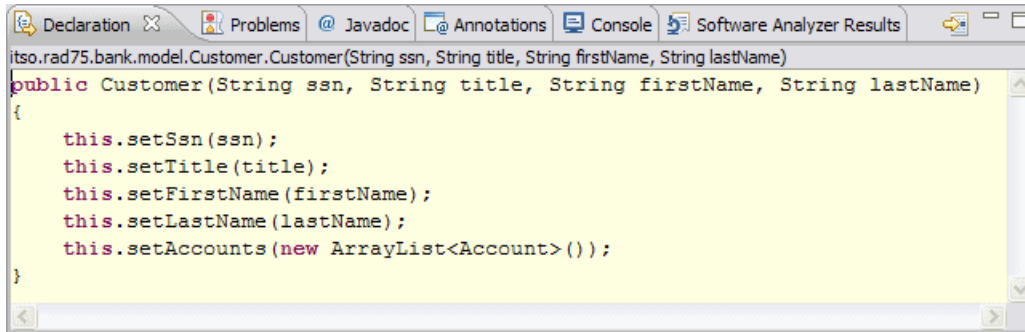


Figure 7-6 Configure Contents of Problems view

7.2.5 Declaration view

The Declaration view shows the declaration and definition of the currently selected type or element, as shown in Figure 7-7 on page 237. It is most useful to see the source code of a referenced type within your code. For example, if you reference a customer within your code and you want to see the implementation of the Customer class, select the referenced type Customer, and the source code of the Customer class is displayed in this view. Clicking the  icon directly opens the source file of the selected type in the Java editor.



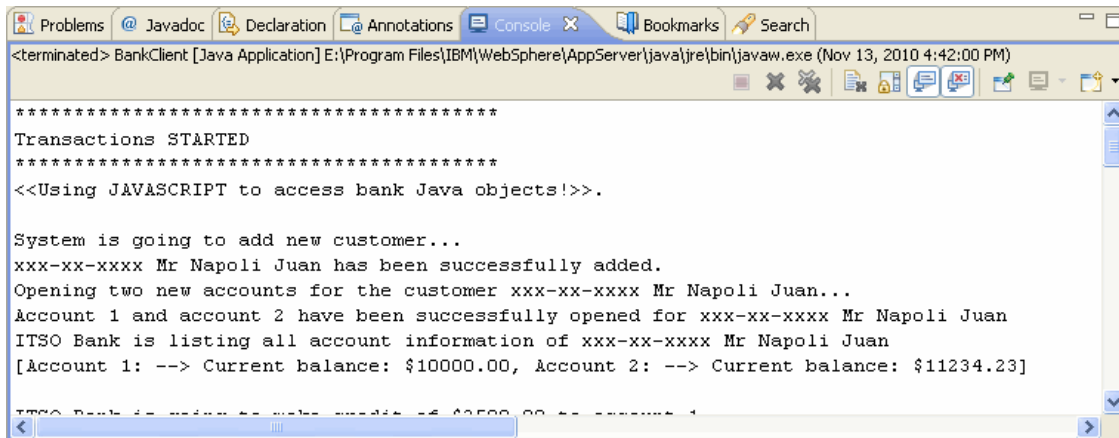
```
itso.rad75.bank.model.Customer.Customer(String ssn, String title, String firstName, String lastName)
public Customer(String ssn, String title, String firstName, String lastName)
{
    this.setSsn(ssn);
    this.setTitle(title);
    this.setFirstName(firstName);
    this.setLastName(lastName);
    this.setAccounts(new ArrayList<Account>());
}
```

Figure 7-7 Declaration view

7.2.6 Console view

In the Console view, Rational Application Developer writes all outputs of a process. In this view, you can provide keyboard inputs to the Java application while running it. Uncaught exceptions are also displayed in the console.

Figure 7-8 show you the result of a running test.










```
<terminated> BankClient [Java Application] E:\Program Files\IBM\WebSphere\AppServer\java\jre\bin\javaw.exe (Nov 13, 2010 4:42:00 PM)
*****
Transactions STARTED
*****
<<Using JAVASCRIPT to access bank Java objects!>>.

System is going to add new customer...
xxx-xx-xxxx Mr Napoli Juan has been successfully added.
Opening two new accounts for the customer xxx-xx-xxxx Mr Napoli Juan...
Account 1 and account 2 have been successfully opened for xxx-xx-xxxx Mr Napoli Juan
ITSO Bank is listing all account information of xxx-xx-xxxx Mr Napoli Juan
[Account 1: --> Current balance: $10000.00, Account 2: --> Current balance: $11234.23]
ITSO Bank is going to make credit of $2500.00 to account 1
```

Figure 7-8 Console view with standard outputs and an exception

The Console view has the following other options:

- ▶ The  icon terminates the currently running process. This button is useful for terminating a process that is running in an endless loop.
- ▶ The  and  icons remove the terminated launches from the console.
- ▶ The  icon clears the console.
- ▶ The  icon enables scroll lock in the console.
- ▶ The  icon pins the current console to remain on the top.
- ▶ The  icon shows the console when JVM logs are updated.

7.2.7 Call Hierarchy view

The Call Hierarchy view shows all callers and callees of a selected method, as shown in Figure 7-9. To view the call hierarchy of a method, select it in the Package Explorer or in the source code, and press Ctrl+Alt+H, or right-click and select **Open Call Hierarchy**.

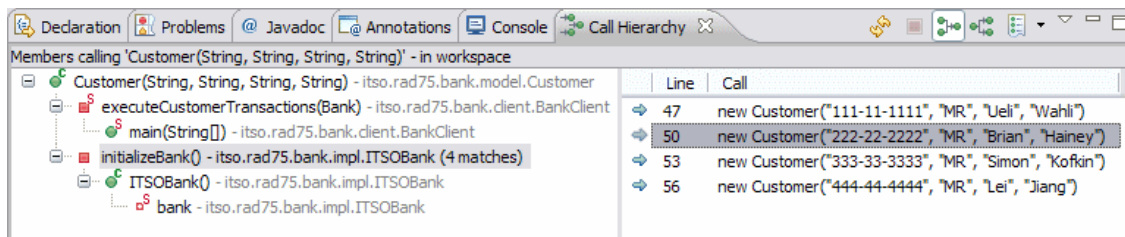




Figure 7-9 Call Hierarchy view (Callee Hierarchy)

The Call Hierarchy view provides the following hierarchy layouts:

- ▶ Caller Hierarchy (): All the members calling the selected method are shown.
- ▶ Callee Hierarchy (): All members called by the selected method are shown.

7.4 Java Type Hierarchy perspective

The Java Type Hierarchy perspective contains only the Hierarchy view, which we explain in 7.2.2, “Hierarchy view” on page 233.

7.5 Developing the ITSO Bank application

In this section, we show how you can use Rational Application Developer to develop a Java SE application. We create a Java project, including several packages, interfaces, classes, fields, and methods.

Sample code: The sample code described in this chapter can be completed by following the procedures that are documented. Alternatively, you can import the sample Java code that is provided in the `c:\7835code\java` directory.

See Appendix C, “Additional material” on page 1877, for instructions about how to download the sample code.

7.5.1 ITSO Bank application overview

The example application to work through in this section is called ITSO Bank. The banking example is deliberately oversimplified, and the exception handling is ignored to keep the example concise and relevant to our discussion.

7.5.2 Packaging structure

The ITSO Bank application contains several packages. Table 7-1 lists the packages and describes their purpose.

Table 7-1 ITSO Bank application packages

Package	Description
itso.rad80.bank.ifc	Contains the interfaces of the application
itso.rad80.bank.impl	Contains the bank implementation class
itso.rad80.bank.model	Contains all business model classes of the application
itso.rad80.bank.exception	Contains the exception classes of the application
itso.rad80.bank.client	Contains the application client that we use to run the application

7.5.3 Interfaces and classes overview

The application contains the following classes and interfaces:

Bank interface	Defines common operations that a bank might perform and typically includes customer, account, and transaction-related services.
TransactionType interface	Defines the transactions that the bank allows.
ITSOBank class	An implementation of the Bank interface.
Account class	The encapsulation of a bank account. It logs all transactions performed on it for logging and querying purposes.
Customer class	The encapsulation of a client of the bank, an account holder. A customer can have one or more accounts.
Transaction class	An abstract supertype of all transactions. A transaction is a single operation that will be performed on an account. In the example, only two transaction types exist: Debit and Credit.
Debit class	One of the two existing concrete subtypes of the Transaction class. This transaction results in an account being debited by the amount indicated.
Credit class	One of the two existing concrete subtypes of the Transaction class. It results in an account being credited by the amount indicated.
BankClient class	The executable class of the ITSO Bank application. It creates instances of ITSOBank, Customer, and Account classes, and performs transactions on the accounts.
All exception classes in the <code>itso.rad80.bank.exception</code> package	The implemented exceptions that can occur in the ITSO Bank application. <code>ITSOBankException</code> is the supertype of all ITSO Bank application exceptions.

7.5.4 Interfaces and classes structure

Table 7-2 describes the ITSO Bank interfaces structure.

Table 7-2 ITSO Bank application interfaces

Interface name	Package	Modifiers
TransactionType	itso.rad80.bank.ifc	public
Bank	itso.rad80.bank.ifc	public

Table 7-3 describes the ITSO Bank classes structure.

Table 7-3 ITSO Bank application classes

Class name	Package	Superclass	Modifiers	Interfaces
ITSOBank	itso.rad80.bank.impl	java.lang.Object	public	itso.rad80.bank.ifc. Bank
Account	itso.rad80.bank.model	java.lang.Object	public	java.io.Serializable
Customer	itso.rad80.bank.model	java.lang.Object	public	java.io.Serializable
Transaction	itso.rad80.bank.model	java.lang.Object	public abstract	java.io.Serializable
Credit	itso.rad80.bank.model	Transaction	public	
Debit	itso.rad80.bank.model	Transaction	public	
BankClient	itso.rad80.bank.client	java.lang.Object	public	
ITSOBankException	itso.rad80.bank. exception	java.lang.Exception	public	
AccountAlready ExistException	itso.rad80.bank. exception	ITSOBankException	public	
CustomerAlready ExistException	itso.rad80.bank. exception	ITSOBankException	public	
InvalidAccount Exception	itso.rad80.bank. exception	ITSOBankException	public	
InvalidAmount Exception	itso.rad80.bank. exception	ITSOBankException	public	
InvalidCustomer Exception	itso.rad80.bank. exception	ITSOBankException	public	

Class name	Package	Superclass	Modifiers	Interfaces
InvalidTransactionException	itso.rad80.bank.exception	ITSOBankException	public	

7.5.5 Interface and class fields and getter and setter methods

Table 7-4 describes the fields of the interfaces.

Table 7-4 Fields of the interfaces

Interface	Field	Type	Initial value	Visibility, modifiers
TransactionType	CREDIT	String	"CREDIT"	public static final
	DEBIT	String	"DEBIT"	public static final

Table 7-5 describes the fields of the classes. The shaded fields are the implementations of UML associations.

Table 7-5 Fields and getter and setter methods of the classes

Class	Field name	Type	Initial value	Visibility, modifiers	Methods
ITSOBank	accounts customers	Map<String,Account> Map<String, Customer>	null	private	getter: public, setter: private
	customer Accounts	Map<String, ArrayList<Account>>			
	bank	ITSOBank	new	private static	getter: public static

Class	Field name	Type	Initial value	Visibility, modifiers	Methods
Account	accountNumber	java.lang.String	null	private	getter: public, setter: private
	balance	java.math.BigDecimal			
	transactions	ArrayList<Transaction>			
Customer	ssn title firstName lastName	java.lang.String	null	private	
	accounts	ArrayList<Account>			
Transaction	timeStamp	Timestamp	null	private	
	amount	java.math.BigDecimal			
	transactionId	int	0		
AccountAlready ExistException	accountNumber	java.lang.String	null	private	
CustomerAlready ExistException	ssn	java.lang.String	null	private	
InvalidAccount Exception	accountNumber	java.lang.String	null	private	
InvalidAmount Exception	amount	java.lang.String	null	private	
InvalidCustomer Exception	ssn	java.lang.String	null	private	
Invalid Transaction Exception	transactionType	java.lang.String	null	private	
	amount	java.math.BigDecimal			
	account	Account			

7.5.6 Interface methods

Table 7-6 on page 245 describes the methods of the Bank interface.

Table 7-6 Method declarations of the Bank interface

Method name	Return type	Parameters	Exceptions
addCustomer	void	Customer customer	CustomerAlreadyExistException
closeAccountOfCustomer	void	Customer customer, Account account	InvalidAccountException, InvalidCustomerException
deposit	void	String accountNumber, BigDecimal amount	InvalidAccountException, InvalidTransactionException
getAccountsForCustomer	ArrayList<Account>	String customerSsn	InvalidCustomerException
getCustomers	Map<String, Customer>		
getTransactionsForAccount	ArrayList<Transaction>	String accountNumber	InvalidAccountException
openAccountForCustomer	void	Customer customer, Account account	InvalidCustomerException, AccountAlreadyExistException
removeCustomer	void	Customer customer	InvalidCustomerException
searchAccountByAccountNumber	Account	String accountNumber	InvalidAccountException
searchCustomerBySsn	Customer	String ssn	InvalidCustomerException
transfer	void	String debitAccountNumber, String creditAccountNumber, BigDecimal amount	InvalidAccountException, InvalidTransactionException
updateCustomer	void	String ssn, String title, String firstName, String lastName	InvalidCustomerException
withdraw	void	String accountNumber, BigDecimal amount	InvalidAccountException, InvalidTransactionException

7.5.7 Class constructors and methods

Table 7-7 describes the constructors and methods of the classes of the application.

Table 7-7 Constructors and methods of the classes of the ITSO Bank application

Method name	Modi-fiers	Type	Parameters	Exceptions
ITSOBank class				
ITSOBank	private	constructor		
addCustomer	public	void	Customer customer	CustomerAlreadyExistExcept ion
removeCustomer		void	Customer customer	InvalidCustomerException
openAccountFor Customer		void	Customer customer, Account account	InvalidCustomerException, AccountAlreadyExistExceptio n
closeAccountOf Customer		void	Customer customer, Account account	InvalidAccountException, InvalidCustomerException
searchAccountBy AccountNumber		Account	String accountNumber	InvalidAccountException
searchCustomerBy Ssn		Customer	String ssn	InvalidCustomerException
processTransactio n	private	void	String accountNumber, BigDecimal amount, String transactionType	InvalidAccountException, InvalidTransactionException

Method name	Modifiers	Type	Parameters	Exceptions
getAccountsForCustomer	public	ArrayList <Account>	String customerSsn	InvalidCustomerException
getTransactionsForAccount		ArrayList <Transaction>	String accountNumber	InvalidAccountException
updateCustomer		void	String ssn, String title, String firstName, String lastName	InvalidCustomerException
deposit		void	String accountNumber, BigDecimal amount	InvalidAccountException, InvalidTransactionException
withdraw		void	String accountNumber, BigDecimal amount	InvalidAccountException, InvalidTransactionException
transfer		void	String debitAccountNumber , String creditAccountNumber, BigDecimal amount	InvalidAccountException, InvalidTransactionException
initializeBank	private	void		
Account class				
Account	public	constructor	String accountNumber, BigDecimal balance	
processTransaction	public	void	BigDecimal amount, String transactionType	InvalidTransactionException
toString		String		
Customer class				
Customer	public	constructor	String ssn, String title, String firstName, String lastName	

Method name	Modi-fiers	Type	Parameters	Exceptions
updateCustomer	public	void	String title, String firstName, String lastName	
addAccount		void	Account account	AccountAlreadyExistExceptio n
removeAccount		void	Account account	InvalidAccountException
toString		String		
Transaction class				
Transaction	public	constructor	BigDecimal amount	
getTransactionType	public abstract	String		
process		BigDecimal	BigDecimal accountBalance	InvalidTransactionException
Credit class				
Credit	public	constructor	BigDecimal amount	
getTransactionType	public	String		
process		BigDecimal	BigDecimal accountBalance	InvalidTransactionException
toString		String		
Debit class				
Debit	public	constructor	BigDecimal amount	
getTransactionType	public	String		
process		BigDecimal	BigDecimal accountBalance	InvalidTransactionException
toString		String		
BankClient class				
main	public static	void	String[] args	

7.6 ITSO Bank application step-by-step development guide

In the following sections, we provide a step-by-step guide to develop the ITSO Bank application in Rational Application Developer.

7.6.1 Creating a Java project

Java projects are not defined in the Java SE specification. They are used as the lowest unit to organize the workspace and contain all resources needed for a Java application, such as images, source, class, and properties files.

With Rational Application Developer started, switch to the Java perspective, as described in 4.1.5, “Switching perspectives” on page 96.

To create a new Java project from the New Java Project wizard, follow these steps:

1. Follow these steps to start the New Java Project wizard:
 - a. In the New Project window, select **File** → **New** → **Project** in the workbench.
 - b. In the Select a wizard window (Figure 7-12 on page 251), select **Java Project**, or expand **Java** and select **Java Project**. Then click **Next**.

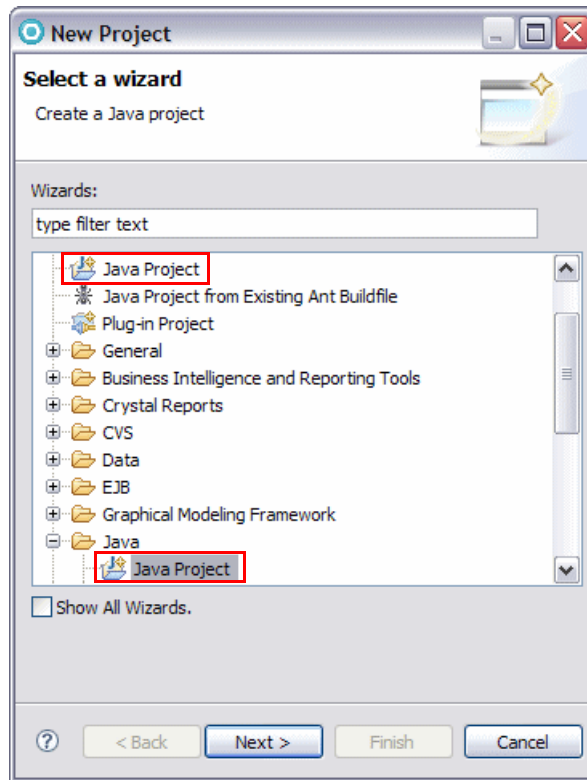



Figure 7-12 New Project window

Another way to start the New Java Project wizard is directly from the workbench. Either select **File** → **New** → **Java Project**, or click the **New Java Project** icon () in the toolbar.

2. In the New Java Project: Create a Java Project window (shown at left in Figure 7-13 on page 252), enter the project name and accept the default settings for each of the other fields:
 - a. For Project name, type RAD80Java.
 - b. For Contents, select **Use default location**.
 - c. For JRE, select **Use default JRE (Currently 'jdk')**.
 - d. For Project layout, select **Create separate source and output folders**.
 - e. Click **Next**.
3. In the New Java Project: Java Settings window (shown at right in Figure 7-13 on page 252), click **Finish** to accept the default settings for each of the tabs. On this window, you can change the build path settings for a Java project. The build class path is a list of paths visible to the compiler when building the project.

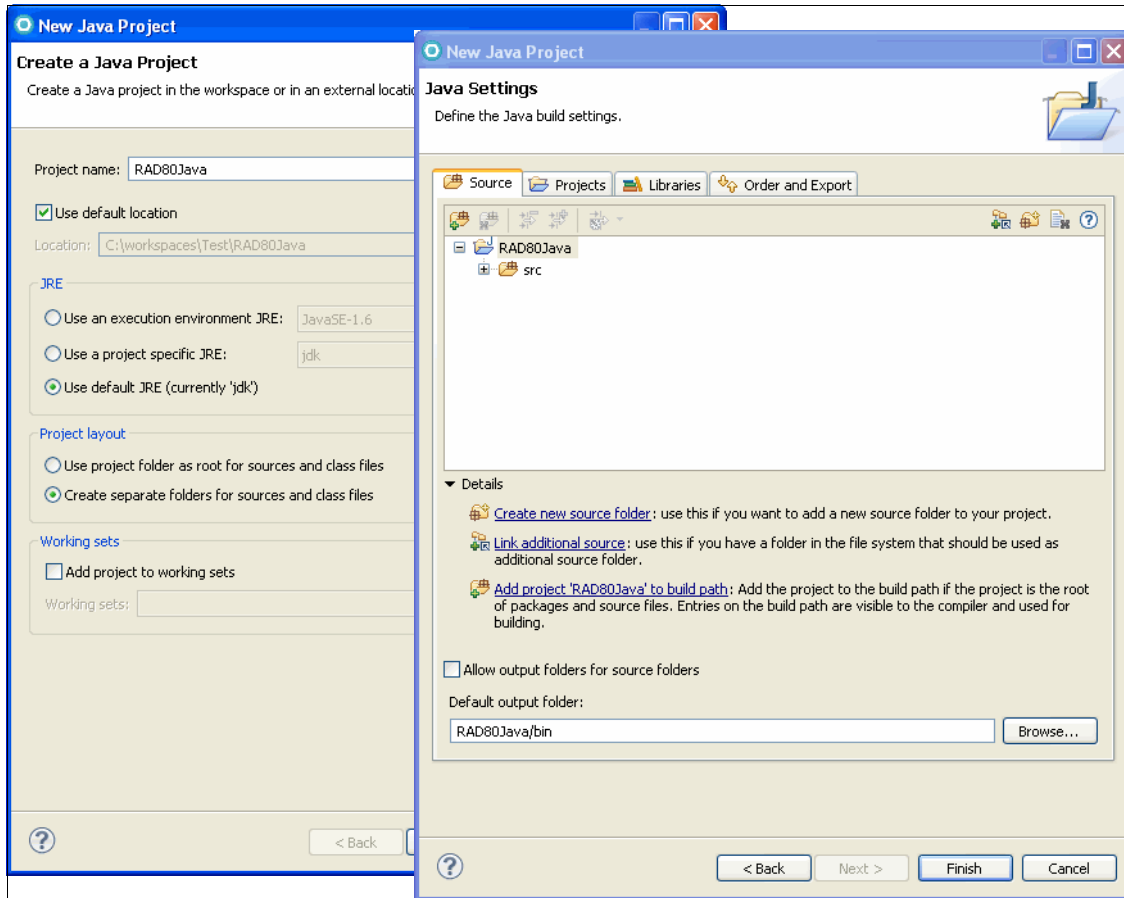


Figure 7-13 New Java Project: Create a Java Project and Java Settings windows

Table 7-8 lists the options in the New Java Project wizard window.

Table 7-8 *New Java Project: Options to create a Java Project*

Option	Description
Project name	Type a name for the new project.
Use default location	<ul style="list-style-type: none">▶ Checked: Create the new project as a folder within the workspace.▶ Unchecked: Create the project as a folder elsewhere in the file system. Click Browse to search for an existing folder.
JRE	<ul style="list-style-type: none">▶ Use default JRE: Uses the workspace default JRE and compiler compliance. Click Configure default to configure JREs.▶ Use project-specific JRE: Specify the JRE to use for the new Java project and set the matching JRE compiler compliance.▶ Use an execution environment JRE: Specify the execution environment and compiler to use for the new Java project.
Project layout	<ul style="list-style-type: none">▶ Use project folder as the root for sources and class files: The project folder is used both as source folder and as output folder for class files.▶ Create separate folders for sources and class files: Creates a source folder for Java source files and an output folder, which holds the class files of the project.
Working sets	<ul style="list-style-type: none">▶ Add project to working sets: The new project is added to the working sets shown in the Working Sets drop-down field. The drop-down field shows a list of previously selected working sets. Click Select to select working sets to which to add the new project.

Table 7-9 explains the capabilities of the New Java Project wizard window.

Table 7-9 New Java Project: Java Settings options

Tab	Description
Source	Add and remove source folders from the Java project. The compiler translates all .java files in the source folders to .class files and stores them to the output folder. The output folder is defined per project, except if a source folder specifies its own output folder. Each source folder can define an exclusion filter to specify which resources inside the folder are not visible to the compiler.
Projects	Add another project within the workspace to the build path for this new project (project dependencies).
Libraries	<p>Add libraries to the build path using one of the following options:</p> <ul style="list-style-type: none"> ▶ Add JARs: Use to navigate the workspace hierarchy and select JAR files to add to the build path. ▶ Add External JARs: Use to navigate the file system (outside the workspace) and select JAR files to add to the build path. ▶ Add Variable: Use to add class path variables to the build path. Class path variables are an indirection to JARs with the benefit of avoiding local file system paths in a class path. This is needed when projects are shared in a team. Variables can be created and edited in the Classpath Variable preference page. Select Window → Preferences → Java → Build Path → Classpath Variables. ▶ Add Library: Use to add predefined libraries, such as JUnit or Standard Widget Toolkit (SWT). ▶ Add Class Folder: Use to navigate the workspace hierarchy and select a class folder for the build path. ▶ Add External Class Folder: Use to navigate the file system (outside the workspace) and select a class folder for the build path. ▶ Migrate JAR: Use to migrate a JAR file on the build path to a newer version. If the newer version contains refactoring scripts, the refactoring stored in the script will be executed.
Order and Export	Change the build path order. You specify the search order of the items in the build path. Select an entry in the list if you want to export it. Exported entries are visible to other projects that require the new Java project being created.

7.6.2 Creating a UML class diagram

Rational Application Developer supports UML class diagrams. You can create a static visual representation of the packages, interfaces, classes, and their relationships. Rational Application Developer automatically calls the related wizard to create the Java code while adding elements to the diagram.

Creating a UML class diagram using the Class Diagram wizard

To create a UML class diagram, for our example, follow these steps:

1. Create a new folder for diagrams:
 - a. In the **RAD80Java** project in the Package Explorer, right-click the **src** folder and select **New** → **Folder** or **New** → **Other** → **General** → **Folder**.
 - b. In the window that opens, for Folder name, type **diagram** and click **Finish** to create the folder.
2. Create an empty class diagram:
 - a. In the Enterprise Explorer, right-click the **diagram** folder and select **New** → **Class Diagram** or **New** → **Other** → **Modeling** → **Class Diagram**.
 - b. In the window that opens, for the field name, type **ITSOBank-ClassDiagram** and click **Finish**.
 - c. Click **OK** to confirm that you have enabled Java Modeling.

The **ITSOBank-ClassDiagram.dnx** file is displayed in the **diagram** folder and opens in the Visualizer Class Diagram editor. Notice that the Java Drawer is open in the Palette (Figure 7-14). Rational Application Developer automatically opens the Java Drawer, by default, for a Java project.

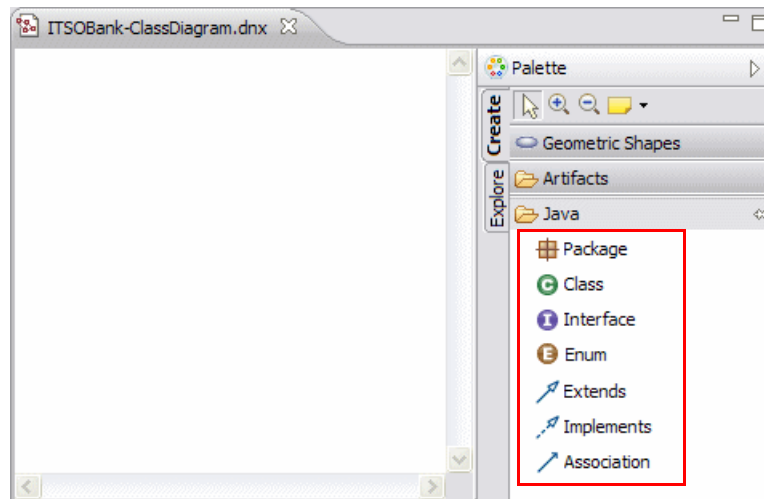




Figure 7-14 Visualizer Class Diagram editor with Java Drawer open in the Palette

Adding existing elements to a class diagram: You can also add already existing elements to a class diagram. Drag the element in the Package Explorer with the left mouse button and drop it in the class diagram by releasing the button.

7.6.3 Creating Java packages


After you create the Java project, you can add the Java packages to the project by using the New Java Package wizard. You can use either of the following options to start the New Java package wizard from the Visualizer Class Diagram editor:

- ▶ Select the  **Package** option in the Java Drawer, as shown in Figure 7-14 on page 255, and click anywhere in the class diagram editor.
- ▶ In the Java project, right-click the **src** folder and select **New** → **Package** or **New** → **Other** → **Java** → **Package**. Alternatively, click the  icon in the toolbar.

Adding a package to the class diagram: To add a package to the class diagram, drag the package to the class diagram editor, or in the Package Explorer, right-click the package and select **Visualize** → **Add to Current Diagram**.

Creating a Java package using the New Java Package wizard

To create a Java package, for our example, follow these steps:

1. In the **RAD80Java** project, select the **src** folder and click the  icon in the toolbar.
2. In the New Java Package window (Figure 7-15 on page 257), in the Name field, type `itso.rad80.bank.model` for the package name and click **Finish** to create the package.

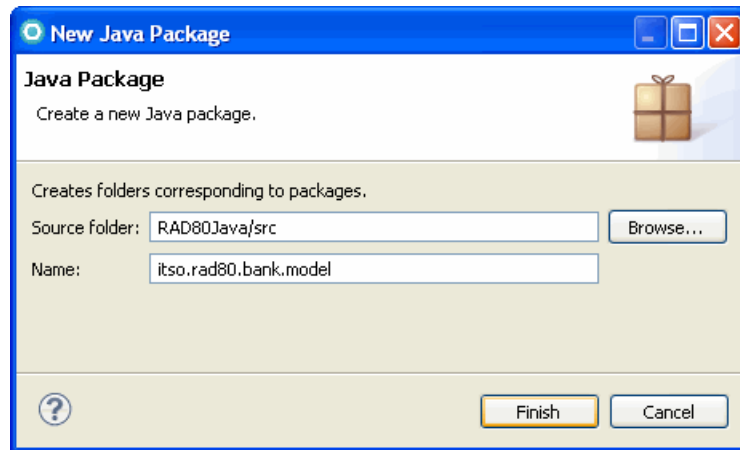


Figure 7-15 Creating a Java package

ITSOBank example: Packages

Repeat the previous steps to create the following Java packages, which are described in 7.5.2, “Packaging structure” on page 240:

- ▶ `itso.rad80.bank.model`
- ▶ `itso.rad80.bank.ifc`
- ▶ `itso.rad80.bank.impl`
- ▶ `itso.rad80.bank.exception`
- ▶ `itso.rad80.bank.client`

7.6.4 Creating Java interfaces

After the Java packages are created, you can add Java interfaces to the packages by using the New Java Interface wizard. You can start the New Java Interface wizard from the Visualizer Class Diagram editor by choosing one of the following options:

- ▶ Select the **Interface** option in the Java Drawer and click anywhere in the class diagram editor field. The New Java Interface wizard opens.
- ▶ Right-click the desired package and select **Add Java → Interface**.
- ▶ Hover the mouse over a package. When the action box opens (Figure 7-16 on page 258), click the **Add Java Interface** icon.



Figure 7-16 Action box: Adding a Java class and interface

Creating a Java interface using the New Java Interface wizard

To create Java interfaces, for our example, follow these steps:

1. Select or mouse over on the package `itso.rad80.bank.ifc` and click the **Add Java Interface** icon. The New Java Interface wizard opens.
2. In the New Java Interface window (Figure 7-17), complete the following steps:
 - a. For Source folder, type `RAD80Java/src` (default).
 - b. For Package, click **Browse** and select `itso.rad80.bank.ifc`.
 - c. For Name, type `Bank`.
 - d. Keep the defaults for all other settings.
 - e. Click **Finish** to create the Java interface.

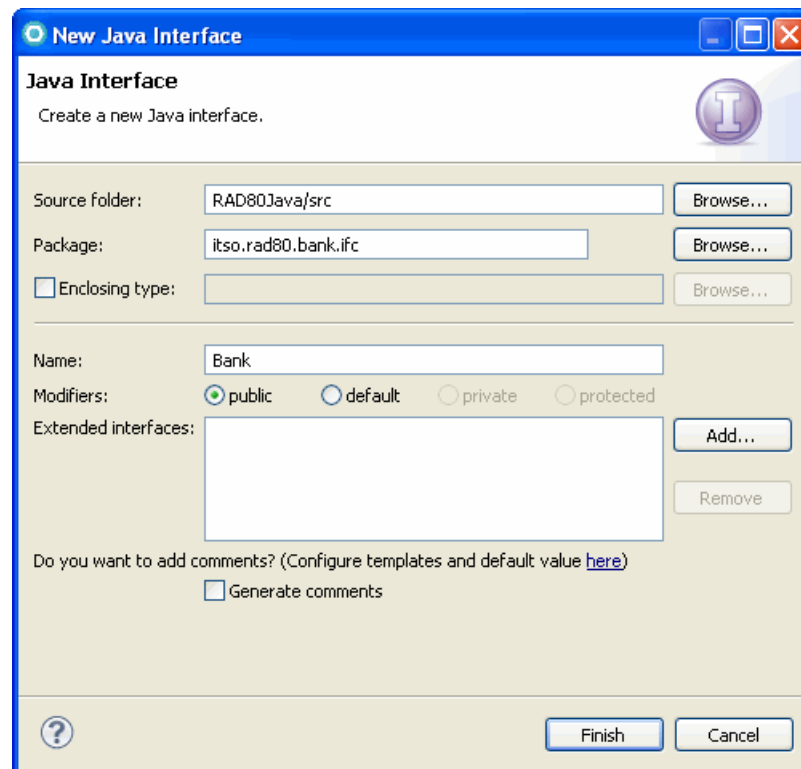


Figure 7-17 Creating a Java interface

Notice that a line is displayed between the package and the Bank interface.

Adding an interface to the class diagram: To add an interface to the class diagram, drag the interface to the class diagram editor, or click the interface in the Package Explorer and select **Visualize** → **Add to Current Diagram**.



ITSOBank example: Interfaces

Repeat the previous steps to create the following Java interfaces, which are described in 7.5.3, “Interfaces and classes overview” on page 241:

- ▶ Bank interface and `itso.rad80.bank.ifc` package
- ▶ TransactionType interface and `itso.rad80.bank.ifc` package

7.6.5 Creating Java classes

With Java packages and Java interfaces created, add the Java classes to the packages using the New Java class wizard. You can start the New Java class wizard from the Visualizer Class Diagram editor by choosing one of the following options:

- ▶ Select the  **Class** option in the Java Drawer and click anywhere in the class diagram editor. The New Java Class wizard opens.
- ▶ Right-click the appropriate package and select **Add Java** → **Class**.
- ▶ Mouse over on the package. In the action box (Figure 7-16 on page 258), click the **Add Java Class** icon (.

Creating a Java class using the New Java Class wizard

To create a Java class using the New Java Class wizard, follow these steps:

1. Select the **itso.rad80.bank.model** package and click the **Add Java Class** icon in the action box.
2. In the New Java Class window (Figure 7-18 on page 260), complete the following steps:
 - a. For Package, click **Browse** and select the **itso.rad80.bank.model** package.
 - b. For Name, type `Transaction`.
 - c. For Modifiers, select **public** (default) and **abstract**.
 - d. For Superclass, type `java.lang.Object` (default).

If you need to change the superclass, click **Browse**. In the Superclass Selection window, in the Choose a type field, type the name of the superclass and click **OK**. All matching types are listed while typing.

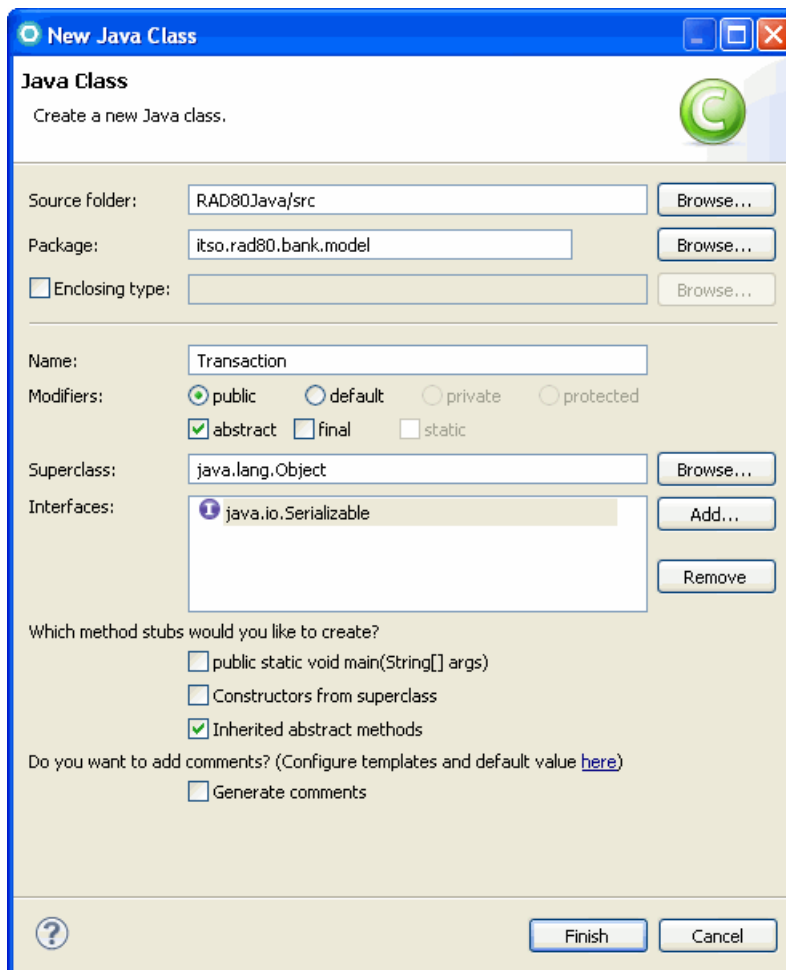


Figure 7-18 Superclass Selection window

- e. For Interfaces, click **Add**.
- f. In the Implemented Interfaces Selection window (Figure 7-19 on page 261), in the Choose interfaces field, type `Serializable`. All matching types are listed. Select the required interface **Serializable - java.io.Serializable** and click **Add**. After you add all required interfaces, click **OK**.

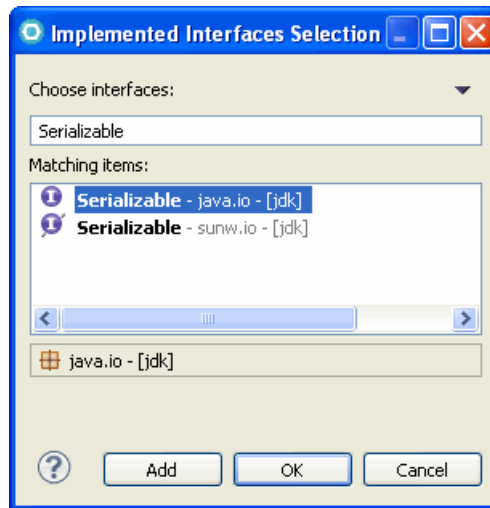


Figure 7-19 Implemented Interfaces Selection window

- g. Which method stubs do you want to create:
- `public static void main(String[] args)`: This option adds an empty main method to the class and makes the class an executable one. In the example, only the class `BankClient` must be executable. Clear is the default setting.
 - Constructors from superclass: This option copies the constructors from the superclass to the new class. Clear is the default setting.
 - Inherited abstract methods: This option adds to the new class stubs of any abstract methods from superclasses or methods of interfaces that have to be implemented. In the example, it is useful for the classes `ITSOBank`, `Credit`, and `Debit`. Ensure that this option is clear.
- h. Ensure that all check boxes are clear.
- i. For Generate comments, ensure that the check box is clear, which is the default setting.
- j. Click **Finish**.

The Java class is created. Notice that a line is displayed between the package and the Bank interface.

Adding a class to the class diagram: You can add a class to the class diagram by using the drag-and-drop method, or select **Visualize** → **Add to Current Diagram**.


ITSOBank example: Classes

Repeat the previous steps to create the following Java classes, which are described in 7.5.4, “Interfaces and classes structure” on page 242:

- ▶ Transaction class into the `itso.rad80.bank.model` package
- ▶ Customer class into the `itso.rad80.bank.model` package
- ▶ Account class into the `itso.rad80.bank.model` package
- ▶ ITSOBank into the `itso.rad80.bank.impl` package
- ▶ ITSOBankException into the `itso.rad80.bank.exception` package
- ▶ InvalidCustomerException into the `itso.rad80.bank.exception` package
- ▶ InvalidAccountException into the `itso.rad80.bank.exception` package
- ▶ InvalidTransactionException into the `itso.rad80.bank.exception` package

7.6.6 Creating Java attributes (fields) and getter and setter methods

After you have the required Java classes and interfaces, add the Java attributes (fields) to them by using the Create Java Field wizard or writing the field declaration into the interface or class body in the Java editor directly. The Create Java Field wizard is only called through the Visualizer Class Diagram editor, by choosing one of the following options:

- ▶ Move the mouse pointer anywhere over the interface or class in the diagram editor and click the **Add Java Field** icon () in the pop-up action box that is displayed, as shown in Figure 7-20.

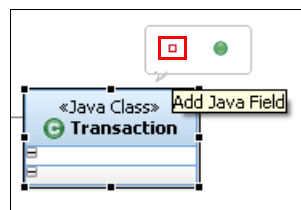


Figure 7-20 Action box: Adding a Java attribute

- ▶ Right-click the interface or class in the diagram editor and select **Add Java** → **Field**. The Create Java Field wizard opens (Figure 7-21 on page 263).

Creating Java fields using the Create Java field wizard

To create Java fields, for our example, follow these steps:

1. Select the **Transaction** class and click the **Add Java Field** icon from the action box.

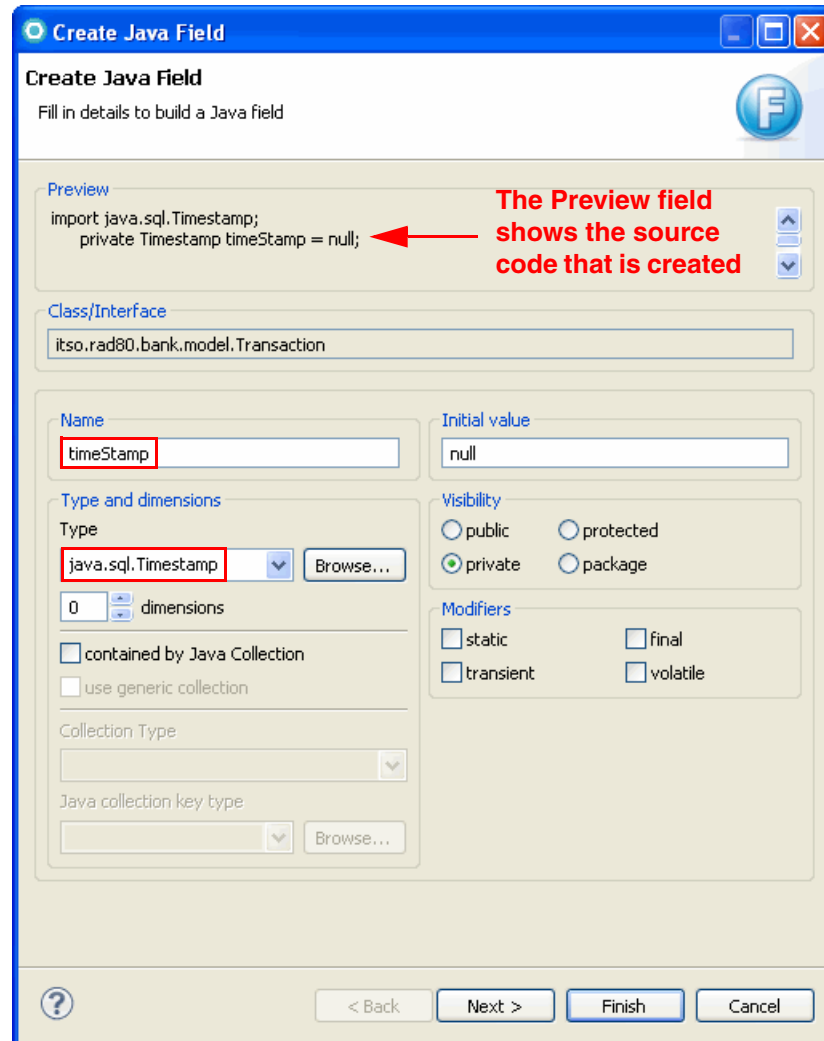


Figure 7-21 Create Java Field window

2. In the Create Java Field window (Figure 7-21), complete the following actions:
 - a. For Name, type timeStamp.

- b. For Type, click **Browse**. To pick a class or interface field, type `java.sql.Timestamp`. Click **OK**. Required import statements are added to the source code automatically.
- c. For Dimensions, keep the default of **0**. You change the value of this field when creating an array of the selected type with the selected dimension.
- d. For contained by Java Collection, ensure that the check box is clear (default).

The contained by Java Collection check box: Select the contained by Java Collection check box if the required attribute has a multiplicity higher than 1. If you select this check box, using the wizard, you can select the required Java collection class. If you select any Map class, you can select the type of the key in the Java collection key type field. Finally, you can create parameterized types by selecting the **use generic collection** check box. For more information about generic types, see the following web address:

<http://java.sun.com/developer/technicalArticles/J2SE/generics/>

- e. For Initial value, type `null`.
- f. For Visibility, ensure that **private** (default) is selected.
- g. For Modifiers, ensure that all check boxes are clear (default).
- h. Click **Finish** to create the Java field.

No attributes in the class diagram: There are two reasons why you might not see the attributes in the class diagram:

- ▶ The class diagram attribute compartment is collapsed. Select the interface or class and click the blue arrow in the compartment in the middle to expand the attribute compartment.
- ▶ The class diagram attribute compartment is filtered out. Right-click the interface or class and select **Filters** → **Show/Hide Compartment** → **Attribute Compartment**.

Creating getter and setter methods using the refactor feature

You can generate getter and setter methods for Java attributes by using the Refactor feature of Rational Application Developer. To generate getter and setter methods for a Java attribute using the Refactor feature, for our example, follow these steps:

1. Select the **Transaction** class in the Package Explorer view, right-click the **timestamp** attribute, and select **Refactor** → **Encapsulate Field** (Figure 7-22).

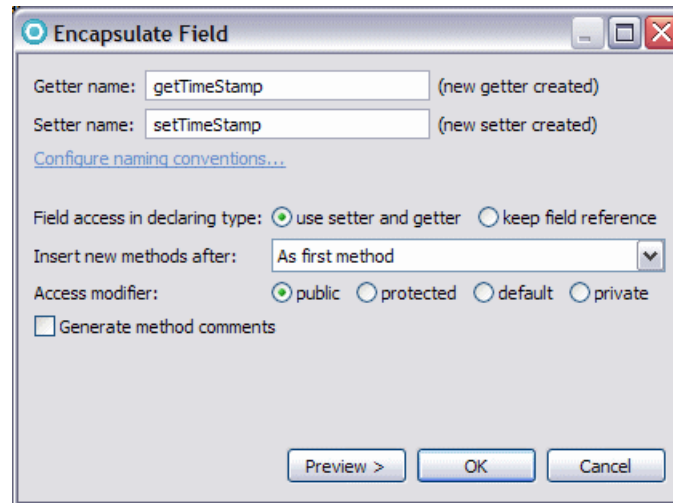


Figure 7-22 Encapsulate Field window

2. In the Encapsulate Field window, complete the following actions:
 - a. For Getter name, accept the default value **getTimeStamp**.
 - b. For Setter name, accept the default value **setTimeStamp**.
 - c. For Field access in declaring type, accept the default value of **use setter and getter**. Using the getter and setter method internally in the class to access member variables is following good programming style.
 - d. For Insert new methods after, accept the default value of **As first method**.
 - e. For Access modifier, select **public**. You can change the access modifier of the setter method later to **private** in the source code.
 - f. For Generate method comments, ensure that the check box is clear (default).
 - g. Click **OK** to generate the getter and setter methods.

Creating getter and setter methods using the source feature

You can generate getter and setter methods for Java attributes by using the source feature of Rational Application Developer. To generate getter and setter methods for a Java attribute using the source feature, for our example, follow these steps:

1. Create a field in the `Transaction` class:
 - a. For Name, type `transactionId`.
 - b. For Type, enter `int`.
 - c. For Initial value, type `0`.
2. Right-click the attribute in the diagram editor or in the Outline view and select **Source** → **Generate Getters and Setters**.

Tip: If the source code is open in the Java editor, right-click anywhere in the Java editor and select **Source** → **Generate Getters and Setters**, or select **Source** → **Generate Getters and Setters** in the menu bar.

3. In the Generate Getters and Setters window (Figure 7-23 on page 267), complete the following actions:
 - a. For Select getters and setters to create, select the defaults of **getTransactionId** and **setTransactionId(int)**.
 - b. For Insertion point, select **Last member** (default).
 - c. For Sort by, select **First getters, then setters**.
 - d. For Access modifier, ensure that the default value of **public** is selected.
You can change the access modifier of the setter method later to `private` in the source code.
 - e. For Generate method comments, ensure that the check box is not selected (default).
 - f. Click **OK** to generate the getter and setter methods.

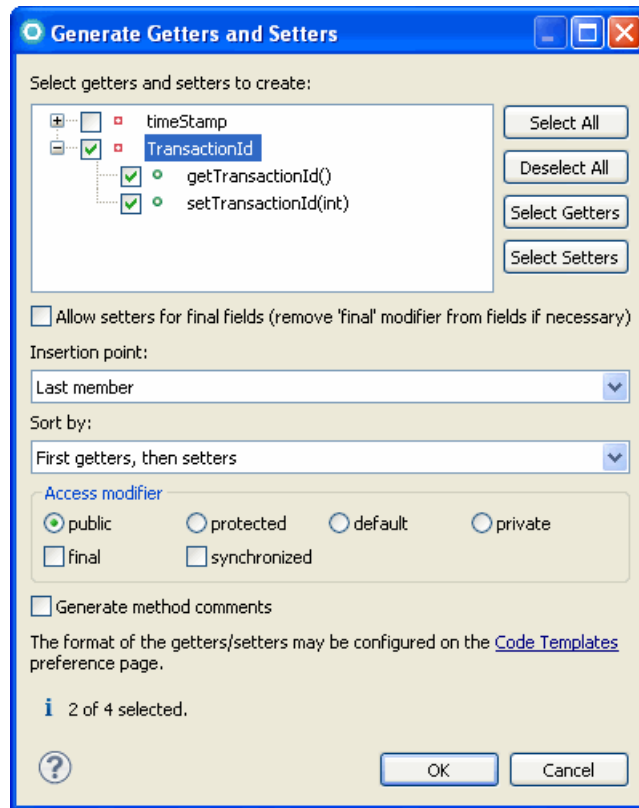


Figure 7-23 Generate Getters and Setters window


ITSOBank example: Fields and getters and setters

Repeat the previous steps to create the following fields to interfaces and to generate getters and setters for the ITSO Bank application classes. Table 7-4 on page 243 lists the fields of the interfaces, and Table 7-5 on page 243 lists the fields and getter and setter methods for the classes.

- ▶ For TransactionType interface, use the CREDIT and DEBIT—`java.lang.String` fields.
- ▶ For Customer class, use the ssn, firstName, and lastName—`java.lang.String` fields.
- ▶ Generate getters (`public`) and setters (`private`) for the Customer class.

7.6.7 Adding method declarations to an interface

You can add a method to a class or interface by either adding a Java method by using the Create Java Method wizard or by writing the method declaration into the interface or class body in the Java editor directly. The Create Java Method wizard is called only through the Visualizer Class Diagram editor:

1. Right-click the **Bank** interface in the diagram editor and select **Add Java** → **Method**. Alternatively, move the mouse pointer anywhere over the interface in the diagram editor and click  in the action bar that is over the class.
2. In the Create Java Method window (Figure 7-24 on page 269), complete the following actions:
 - a. For Name, type `searchCustomerBySsn`.
 - b. For Visibility, accept the default of **public**.
 - c. For Modifiers, ensure that all check boxes are cleared (default).

Restriction: All methods of a Java interface are `public abstract`. The `abstract` modifier can be omitted, because it is, by default, `abstract`. Therefore, there is no choice by Visibility and Modifiers when you add a method declaration to an interface. An interface never has a constructor. Therefore, the constructor check box is never active.

- d. For Type, select **void**.
- e. For Dimensions, accept the default value of **0**.
- f. For Throws, click **Add** to add an exception. To define one or more exception types to throw, type the exception class name. All matching types are listed in the Matching types field. Select the required exceptions and click **OK**. In our example, we use `itso.rad80.bank.exception.InvalidCustomerException`.

The `InvalidCustomerException` class must be created first under the `itso.rad80.bank.exception` package to be selected from the Browse Types list. Follow the instructions in “Creating a Java class using the New Java Class wizard” on page 259.
- g. For Parameters, click **Add**. In the Create Parameter window, enter the name and select the type and dimensions. We add `java.lang.String ssn`. Click **OK** to add the parameter. In this example, we do not pass any array parameters, and dimensions are always 0 (default).
- h. Click **Finish** to create the Java method.

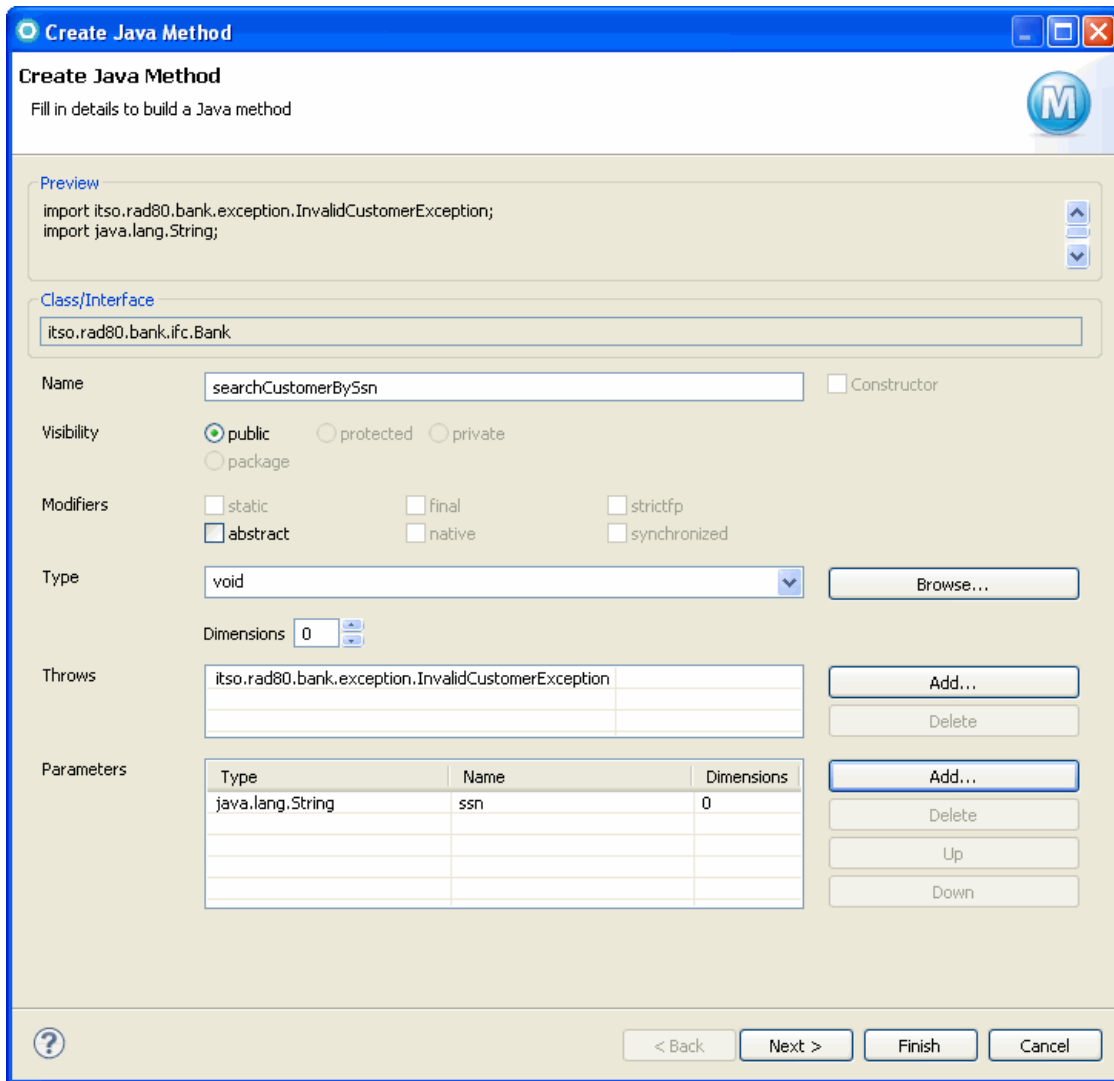


Figure 7-24 Create Java Method window

No methods in the class diagram: There are two reasons why you might not see the methods in the class diagram:

- ▶ The class diagram method compartment is collapsed. Select the interface or class and click the blue arrow in the compartment in the bottom to expand the method compartment.
- ▶ The class diagram method compartment is filtered out. Right-click the interface or class and select **Filters** → **Show/Hide Compartment** → **Method Compartment**.

ITSOBank example: Interface methods

Repeat the previous steps to create the following method declarations to the Bank interface of the ITSO Bank application. Table 7-6 on page 245 lists all method declarations that can be created.

- ▶ For `searchCustomerBySsn()`, use the `itso.rad80.bank.model.Customer` type.
- ▶ For `getCustomers()`, use the `java.util.Map` type.
- ▶ For `transfer()`, use the `void` type.

Remember:

- ▶ Set the correct parameters and exceptions to the methods that you create.
- ▶ You can also import the final code later in 7.6.10, “Implementing the classes and methods” on page 275.

7.6.8 Adding constructors and Java methods to a class

The method for adding constructors and methods to a class is the same as when you add a method declaration to an interface. See the steps in 7.6.7, “Adding method declarations to an interface” on page 268. There are no restrictions as described for the interfaces.

ITSOBank example: Class methods

Repeat the previous steps to create the following class methods for the ITSO Bank application. Remember that you must select **Constructor** when adding a constructor to a class. Table 7-7 on page 246 lists all method declarations that can be created.

- ▶ For ITSOBank class, use the `updateCustomer()` and `transfer()` methods.
- ▶ For Customer class, use the Constructor methods.
- ▶ For Transaction class, use the `getTransactionType()` methods.

Reminder: You can import the final code later, as explained in 7.6.10, “Implementing the classes and methods” on page 275.

Tip: If you want to add a method to a class that implements or overrides an existing method in an interface or a superclass, there is a much faster way to add it than by using the Create Java Method wizard. Use the source feature Override/Implement Methods:

1. Right-click the class in the diagram editor or in the Package Explorer and select **Source** → **Override/Implement Methods**.
2. In the Override/Implement Methods window (Figure 7-25), select the methods that you want to override or implement from the list of all methods that can be implemented or overridden by this class. For example, you can implement the `toString()` method in the `Transaction` class.

Click **OK** to add the method stubs to the selected class.

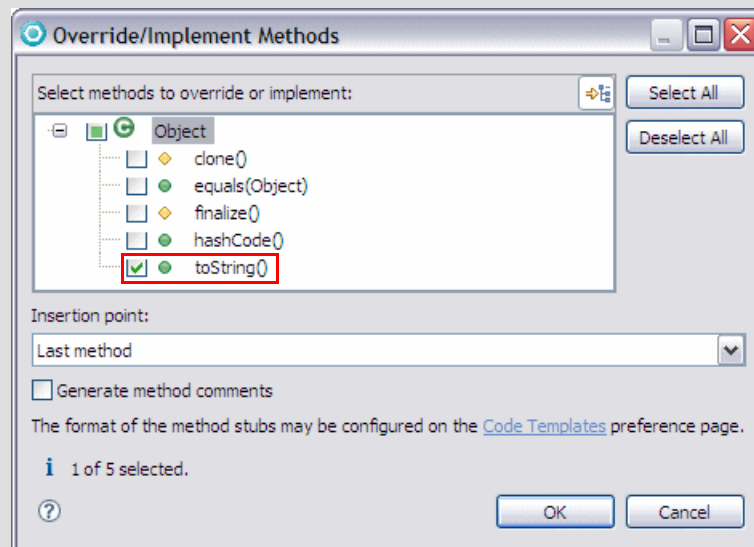


Figure 7-25 Window for Override/Implement Methods

7.6.9 Creating relationships between Java types

The classes in the ITSO Bank application have the following relationships:

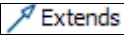
- ▶ ITSOBank remembers the customers and accounts.
- ▶ A customer knows its accounts.

- ▶ An account logs all the transactions for logging and querying purposes.

In Rational Application Developer, you can model the relationships between Java types in the Visualizer Class Diagram editor. This section includes the following topics:

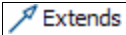
- ▶ Extends relationships
- ▶ Implements relationships
- ▶ Association relationships

Extends relationships

Extends relationships are used inside the Class diagram to represent the inheritance between Java classes. To create an *extends* relationship between existing classes, select the  option in the Java Drawer and drag the mouse with the left mouse button down from any point on the child class to the parent class.

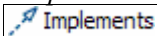
ITSOBank example: Extends relationship

The only case of inheritance in our application is between the `Transaction` class as a superclass of the `Credit` and `Debit` classes. Follow these steps to create those relationships:

1. Create the `Credit` class, as mentioned in 7.6.5, “Creating Java classes” on page 259. Select `itso.rad80.bank.model.Transaction` as the superclass.
2. Create the `Debit` class as you did with the `Credit` class, but this time, leave the default `java.lang.Object` as the superclass.
3. Select the  option in the Java Drawer, hold down the left mouse button, and drag the mouse from any point on the `Debit` class to the `Transaction` class.

A solid line with a triangular arrow is displayed from the `Credit` class and the `Debit` class to the `itso.rad80.bank.model.Transaction` class, indicating that the extends relationships were created successfully.

Implements relationships


Implements relationships are used inside the Class diagram to represent the usage of one or many Java interfaces by a Java class. To create an *implements* relationship between an existing class and an interface, select the  option in the Java Drawer, hold down the left mouse button, and drag from any point in the implementation class to the interface. The implements relationship is displayed using a dashed line with a triangular arrow pointing to the interface.

ITSOBank example: Implements relationship

An implements relationship is already in the class diagram. The `ITSOBank` class implements the `Bank` interface.

Association relationships

Association relationships are used inside the Class diagram to represent the object-level dependencies between Java classes. To create an association relationship between two classes, follow these steps:

1. Select the  **Association** option in the Java drawer.
2. Drag the mouse with the left mouse button down from any point on the Customer class to the Account class.
3. In the Create Association window, enter the following data (Figure 7-26 on page 274):
 - a. For Name, type `accounts`.
 - b. For Dimensions, accept the default value of **0**.
 - c. Select **contained by Java Collection**.
 - d. For Collection Type, select **`java.util.ArrayList`**.
 - e. Select **Use generic collection**.
 - f. For Initial value, type `null`.
 - g. For Visibility, accept the default of **private**.
 - h. For Modifiers, ensure that all check boxes are cleared (default).
 - i. Click **Finish** to create the association.

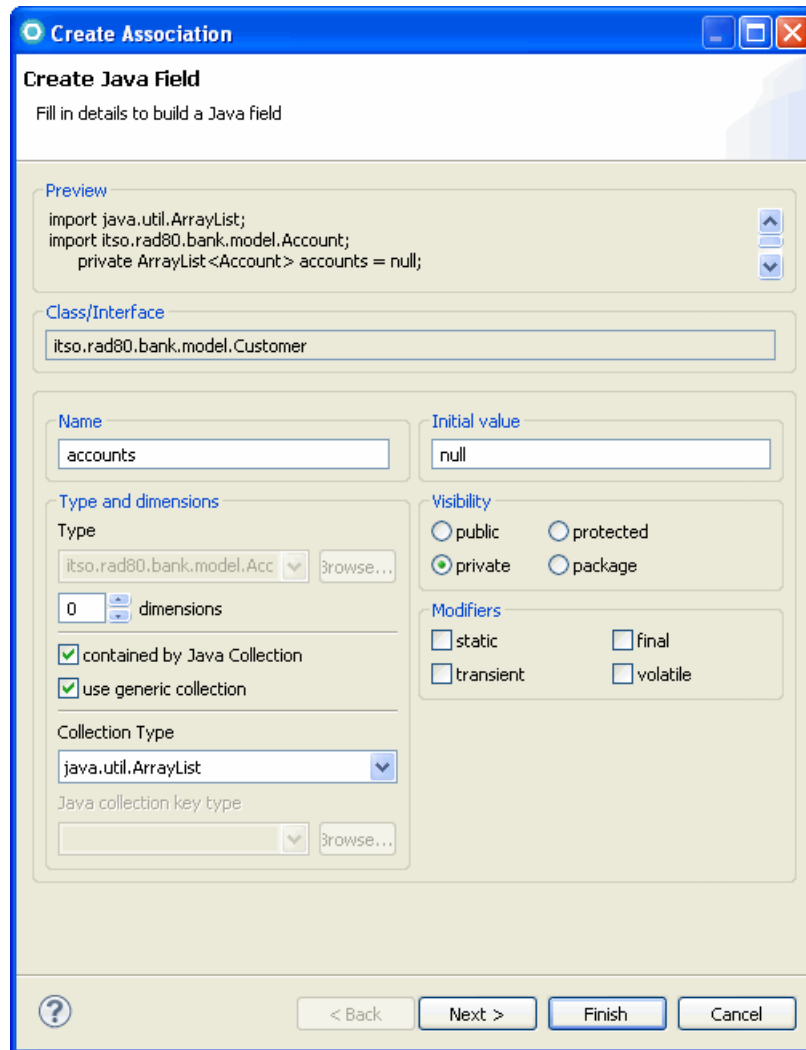


Figure 7-26 Create Association window

Association: An association can be displayed as an arrow or an attribute:

- ▶ Show as **attribute**: To display the association as an attribute, right-click the association arrow and select **Filters** → **Show As Attribute**.
- ▶ Show as **association arrow**: To display the attribute as an association, right-click the attribute and select **Filters** → **Show As Association**.

ITSOBank example: Association relationship

Repeat the previous steps to create the following association relationships of the ITSO Bank application. The associations are listed in Table 7-5 on page 243.

- ▶ For ITSOBank class, use the accounts (Map <String, Account>) field.
- ▶ For ITSOBank class, use the customers (Map <String, Customer>) field.
- ▶ For Account class, use the transactions (ArrayList<Transaction>) field.

7.6.10 Implementing the classes and methods

In the previous sections of the ITSO Bank application example, we include step-by-step approaches with the objective of demonstrating the Rational Application Developer tooling and the logical process of developing a Java application. In this section, we import all the classes with the method code.

Importing the classes

To import the classes, perform these steps:

1. Right-click the **src** folder and select **Import**.
2. Select **General** → **File System** and click **Next**.
3. Click **Browse** and navigate to the C:\7835code\java\import folder.
4. In the Import: File system window (Figure 7-27 on page 276), select the **import** folder. Expand the folder to see all classes that will be imported. They are provided in several packages.
5. Click **Finish**.

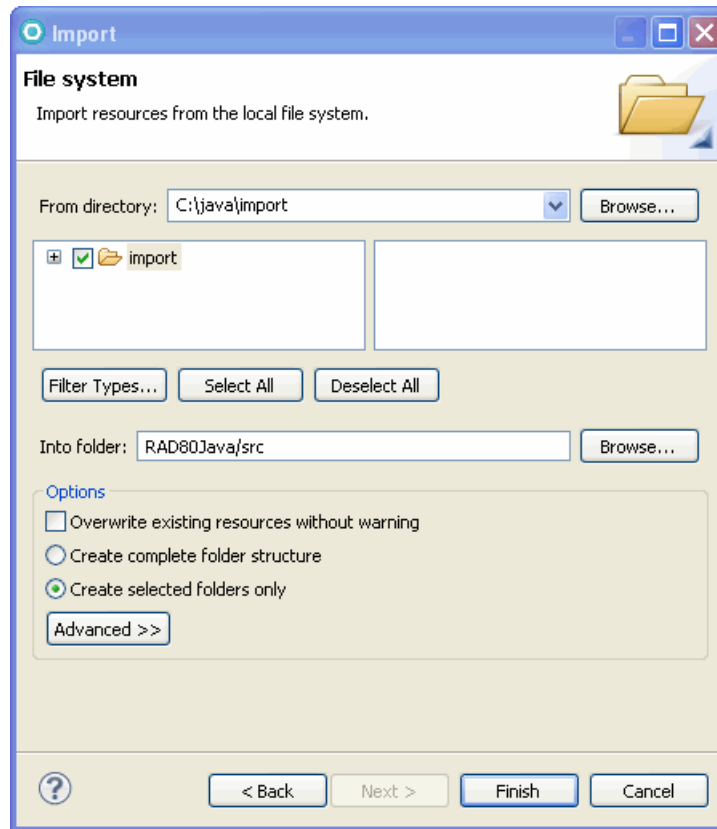


Figure 7-27 Importing the classes

6. Add all the classes to the diagram manually or import the diagram into the diagram folder from the C:\7835code\java\diagram\ITS0Bank-Diagram.dnx folder.

To change the appearance of the diagram, right-click in the diagram and select **Filters** → **Show/Hide Connector Labels** → **All** or **No connector Labels**. Alternatively, select **Filters** → **Show/Hide Relationships** and select the relationships to be displayed or hidden. For example, you can hide the many <<use>> relationships.

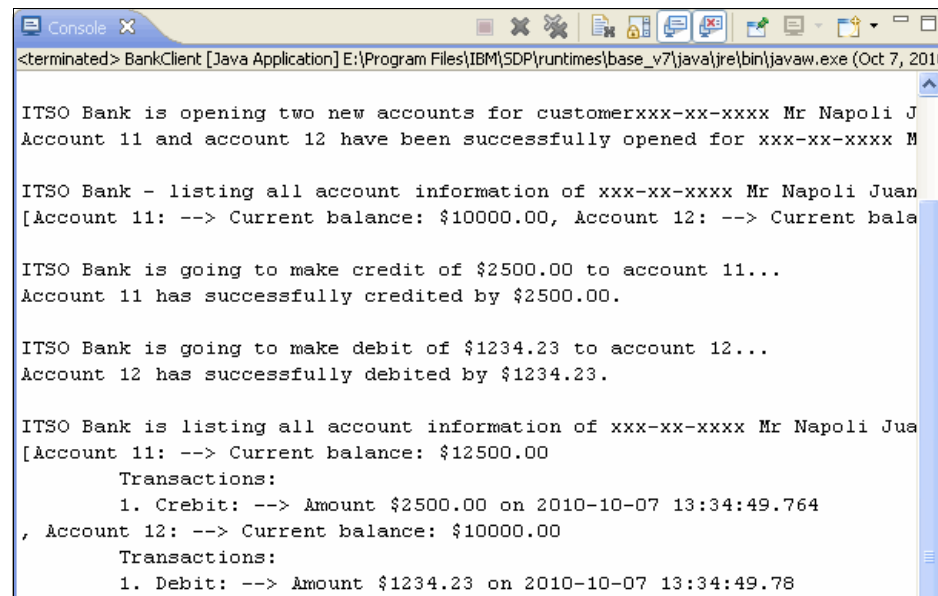
7.6.11 Running the ITSO Bank application

After you complete the ITSO Bank application, while importing all classes into your project RAD80Java, as described in the section before, you are ready to test the application. To start the application, we use a generic Java Application launch configuration that derives most of the start parameters from the Java project and the workbench preferences.

To run the ITSO Bank application, right-click the **BankClient** class in the Package Explorer and select **Run As** → **Java Application** (🟢), or click the arrow (▼) icon in the toolbar and select **Run As** → **Java Application**.

Executable class: The selected class must be executable, containing a public static void main(String[] args) method. Otherwise, the application cannot run.

You can see the output in the Console view. Figure 7-28 shows part of the output.



```
<terminated> BankClient [Java Application] E:\Program Files\IBM\SDP\runtimes\base_v7\java\jre\bin\javaw.exe (Oct 7, 2010)
ITSO Bank is opening two new accounts for customerxxx-xx-xxxx Mr Napoli J
Account 11 and account 12 have been successfully opened for xxx-xx-xxxx M

ITSO Bank - listing all account information of xxx-xx-xxxx Mr Napoli Juan
[Account 11: --> Current balance: $10000.00, Account 12: --> Current bala

ITSO Bank is going to make credit of $2500.00 to account 11...
Account 11 has successfully credited by $2500.00.

ITSO Bank is going to make debit of $1234.23 to account 12...
Account 12 has successfully debited by $1234.23.





ITSO Bank is listing all account information of xxx-xx-xxxx Mr Napoli Jua
[Account 11: --> Current balance: $12500.00
  Transactions:
    1. Crebit: --> Amount $2500.00 on 2010-10-07 13:34:49.764
, Account 12: --> Current balance: $10000.00
  Transactions:
    1. Debit: --> Amount $1234.23 on 2010-10-07 13:34:49.78
```

Figure 7-28 Console view with output of the ITSO Bank application

7.6.12 Creating a run configuration

In certain cases, you might want to override the derived parameters or specify additional arguments.

To create a run configuration, follow these steps:

1. Select **Run** → **Run Configurations** () or click the arrow () icon in the toolbar and select **Run Configurations**.
2. In the Run Configurations window (Figure 7-29), select the  Java Application option and then click the  icon to create a new configuration. Notice that we already have a configuration from running the BankClient application.

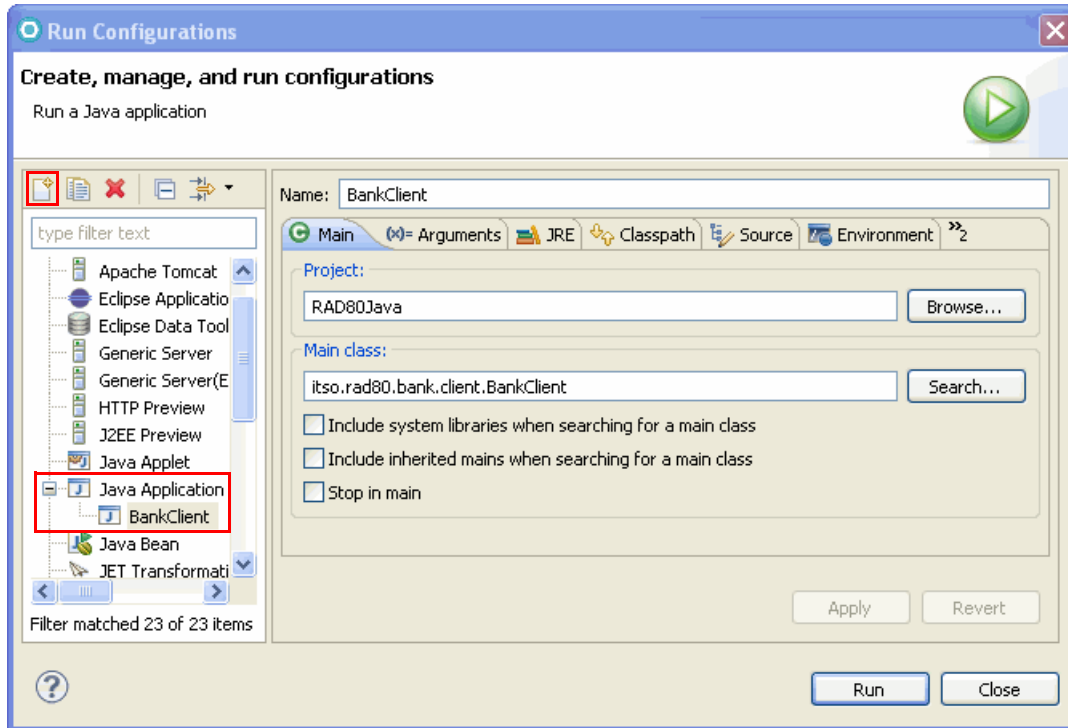


Figure 7-29 Run Configurations window

- On the Main tab, you define the class to start:
 - i. For Project, select the project containing the class to start.
 - ii. For Main class, click **Search** to see a list of all executable main classes in the project. Select the main class to be started.
 - iii. For the “Include system libraries when searching for a main class” and “Include inherited mains when searching for a main class” check boxes, you can expand the area where Rational Application Developer is searching for an executable class.
 - iv. For the Stop in main check box, the program stops in the main method whenever it is started in debug mode. You do not have to specify a

project, but by specifying a project, you can choose a default class path, source lookup path, and JRE.

- On the Arguments tab, you define the arguments to pass to the application and to the virtual machine. To add a program argument, follow these steps:
 - i. Type a value directly into the field or use a variable.
 - ii. Click **Variables** under the Program arguments field.
 - iii. Select one of the predefined variables or create your own variable by clicking **Edit Variables** and then by clicking **New**.
 - iv. Enter the name and the value for the variable and click **OK** to add it.
 - v. Click **OK** again to return to the Select a variable window. The new variable is now available in the list. Select it and click **OK** to return to the Arguments tab.

In the same way, you can also add VM arguments. You can also specify the working directory to be used by the started application.

- On the JRE tab, you define the JRE that is used to run or debug the application. You can select a JRE from the already defined JREs or define a new JRE.
- On the Classpath tab, you define the location of class files used when running or debugging an application. By default, the user and bootstrap class locations are derived from the associated project's build path. You can override these settings here.
- On the Source tab, you define the location of source files used to display source when debugging a Java application. By default, these settings are derived from the associated project's build path. You can override these settings here.
- On the Environment tab, you define the environment variable values to use when running or debugging a Java application. By default, the environment is inherited from the Eclipse run time. You can override or append to the inherited environment.
- On the Common tab, you define general information about the launch configuration. You can select to store the launch configuration in a specific file and specify which perspectives become active when the launch configuration is started.

Click **Run** to start the class.

7.6.13 Understanding the sample code

In this section, we explain the content of the sample code for the ITSO Bank solution. You can later study all the sample code imported into the project.

BankClient class

As the starting class for the sample application, the `BankClient` class creates an instance of the `ITSOBank` class and uses the `Customer`, `Account`, and `Transaction` methods to operate with the bank information. Example 7-1 shows part of the relevant Java source code in a simplified format for the `BankClient` class.

Example 7-1 BankClient class (abbreviated)

```
package itso.rad80.bank.client;
public class BankClient {
    public static void main(String[] args) {
        Bank iTSOBank = ITSOBank.getBank();
        executeCustomerTransactions(iTSOBank);
    }

    private static void executeCustomerTransactions(Bank bank)
        throws ITSOBankException {
        .....
        customer1 = new Customer("xxx-xx-xxxx", "Mr", "Juan", "Napoli");
        bank.addCustomer(oCustomer);
        (...)
    }
}
```

ITSOBank class

The `ITSOBank` class implements the logic for the interface `Bank` and contains all the business logic related to the manipulation of customers, accounts, and transactions in the `ITSOBank` application. Example 7-2 shows part of the relevant Java source code in a simplified format for the `ITSOBank` class.

Example 7-2 ITSOBank class

```
public class ITSOBank implements Bank {

    public ITSOBank() {
        this.setCustomers(new HashMap<String, Customer>());
        this.setAccounts(new HashMap<String, Account>());
        this.setCustomerAccounts(new HashMap<String,
ArrayList<Account>>());
        this.initializeBank();
    }
}
```

```

    }

    private void initializeBank() {
        Customer customer1 = new Customer("111-11-1111", "MR", "Ueli",
            "Wahli");
        this.addCustomer(customer1);
        (...)
    }
    public void updateCustomer(String ssn, String title, String
firstName,
        String lastName) throws InvalidCustomerException {
        this.searchCustomerBySsn(ssn).updateCustomer(title, firstName,
            lastName);
    }
    public void withdraw(String accountNumber, BigDecimal amount)
        throws InvalidAccountException, InvalidTransactionException {
        this.processTransaction(accountNumber, amount,
            itso.rad80.bank.ifc.TransactionType.DEBIT);
    }
    .....
}

```

Customer class

The Customer class handles the data and processes the logic of the customer entity in the ITSO Bank application. A customer can have many accounts. Therefore, it handles the relationship with the Account class. Example 7-3 shows part of the relevant Java source code in a simplified format for the Customer class.

Example 7-3 Customer class

```

package itso.rad80.bank.model;
public class Customer {
    public Customer(String ssn, String title, String firstName,
        String lastName) {
        this.setSsn(ssn);
        this.setTitle(title);
        this.setFirstName(firstName);
        this.setLastName(lastName);
        this.setAccounts(new ArrayList<Account>());
    }
    public void addAccount(Account account) throws
AccountAlreadyExistException
    {

```

```

        if (!this.getAccounts().contains(account)) {
            this.getAccounts().add(account);
            .....
        }
        .....
    }
}

```

Account class

The Account class handles the data and processes the logic of the account entity in the ITSO Bank application. On an account, many transactions can be held. Therefore, it handles the relationship with the Transaction class. Example 7-4 shows part of the relevant Java source code in simplified format for the Account class.

Example 7-4 Account class

```

package itso.rad80.bank.model;
public class Account implements Serializable {
    public Account(String accountNumber, BigDecimal balance) {
        this.setAccountNumber(accountNumber);
        this.setBalance(balance);
        this.setTransactions(new ArrayList<Transaction>());
    }
    public void processTransaction(BigDecimal amount, String
transactionType)
        throws InvalidTransactionException {
        .....
        if (TransactionType.CREDIT.equals(transactionType)) {
            transaction = new Credit(amount);
        }
        else if (TransactionType.DEBIT.equals(transactionType)) {
            transaction = new Debit(amount);
        }
        .....
    }
    .....
}

```

Transaction class

The Transaction class handles the data and processes the logic of a transaction in the ITSO Bank application. A transaction can be either the credit or debit type, which is why both inherit the transaction class structure. Example 7-5 shows part of the relevant Java source code in simplified format for the Transaction class.

Example 7-5 Transaction class

```
package itso.rad80.bank.model;
public abstract class Transaction implements Serializable {
    static int transactionCtr = 1; // to increment transactionId
    public Transaction(BigDecimal amount) {
        this.setTimeStamp(new Timestamp(System.currentTimeMillis()));
        this.setAmount(amount);
        this.setTransactionId(transactionCtr++);
    }
    public abstract String getTransactionType();
    public abstract BigDecimal process(BigDecimal accountBalance)
        throws InvalidTransactionException;
    .....
}
```

Credit class

The Credit class handles the implementation code of a transaction class for credit operations. Example 7-6 shows part of the relevant Java source code in simplified format for the Credit class.

Example 7-6 Credit class

```
package itso.rad80.bank.model;
public class Credit extends Transaction {
    public BigDecimal process(BigDecimal accountBalance)
        throws InvalidTransactionException {
        .....
        return accountBalance.add(this.getAmount());
        .....
    }
    (...)
}
```

Debit class

The Debit class is similar to the Credit class, but subtracts the amount from the balance.

7.6.14 Additional features used for Java applications

The Java editor of Rational Application Developer provides a set of useful features to develop the code. In this section, we highlight key features of Rational Application Developer when working on a Java project:

- ▶ Using scripting inside the JRE
- ▶ Analyzing the source code
- ▶ Debugging a Java application
- ▶ Using the Java scrapbook
- ▶ Pluggable Java Runtime Environment
- ▶ Exporting Java applications to a JAR file
- ▶ Running Java applications that are external to Rational Application Developer
- ▶ Importing Java resources from a JAR file into a project

7.6.15 Using scripting inside the JRE

Since the release of JRE Version 1.6, scripting code can be executed inside the virtual machine environment with the use of the classes in the `javax.script.*` native Java package. The classes included in the JRE release contain Java implementation logic for Mozilla open source Rhino and ECMAScript JavaScript language engines. However, many others, such as Ruby or Python, can be included, or you can make your own scripting interpreter.

ITSOBank example: Scripting invocation

In our application example, we create a scripting implementation, which you have already imported in “Importing the classes” on page 275. The scripting module has two components:

- ▶ The `BankClientScript.js` scripting file in the package `itso.rad80.bank.client`, which contains a similar logic as implemented by the method `executeCustomerTransactions(Bank bank)` in the `BankClient` class
- ▶ The `executeCustomerTransactionsWithScript(Bank bank)` method in the class `BankClient`, which invokes the script file and evaluates its logic

To test the scripting functionality, we must modify two lines of code in the `main(String[] args)` method of the `BankClient` class:

1. Look for the following code:

```
//Here you can switch the logic to be implemented in Java or
Scripting
executeCustomerTransactions(itSOBank);
//executeCustomerTransactionsWithScript(itSOBank);
```


2. Select the second line, right-click, and select **Source** → **Toggle Comment** to comment the line.
3. Select the third line, right-click, and select **Source** → **Toggle Comment** to uncomment it.
4. Save the changes (Ctrl+S).
5. Run the ITSOBank application, as described in 7.6.11, “Running the ITSO Bank application” on page 277.
6. Verify that the output console has no errors and shows the message <<Using JAVASCRIPT to access bank Java objects!>> (Figure 7-30).

```

<terminated> BankClient [Java Application] E:\Program Files\IBM\SDP\jdk\bin\javaw.exe (Oct 7, 2010 2:32:54 PM)

*****
Transactions STARTED
*****
<<Using JAVASCRIPT to access bank Java objects!>>.

System is going to add new customer...
xxx-xx-xxxx Mr Napoli Juan has been successfully added.
Opening two new accounts for the customer xxx-xx-xxxx Mr Napoli Juan...
Account 1 and account 2 have been successfully opened for xxx-xx-xxxx Mr Napoli Juan
ITSO Bank is listing all account information of xxx-xx-xxxx Mr Napoli Juan
[Account 1: --> Current balance: $10000.00, Account 2: --> Current balance: $11234.23]

ITSO Bank is going to make credit of $2500.00 to account 1...
Account 1 has sucessfully credited by $2500.00.

ITSO Bank is going to make debit of $1234.23 to account 2...
Account 2 has sucessfully debited by $1234.23.

ITSO Bank is listing all account information of xxx-xx-xxxx Mr Napoli Juan...
[Account 1: --> Current balance: $12500.00
  Transactions:
  1. Crebit: --> Amount $2500.00 on 2010-10-07 14:32:56.217
, Account 2: --> Current balance: $10000.00
  Transactions:
  1. Debit: --> Amount $1234.23 on 2010-10-07 14:32:56.233
1

```

Figure 7-30 Executing the BankClient with scripting

How the scripting example works

Study the code of the executeCustomerTransactionsWithScript method:

```

private static void executeCustomerTransactionsWithScript(Bank oBank)
    throws ScriptException {
    //Lookup for the scripting engine

```

```

ScriptEngineManager engineMgr = new ScriptEngineManager();
ScriptEngine engine = engineMgr.getEngineByName("ECMAScript");
//Insert the bank object in the Bindings scope
engine.put("bank", oBank);
//Execute the script
try {
    InputStream inputStream = Thread.currentThread()
        .getContextClassLoader()

.getResourcesAsStream("itso/rad80/bank/client/BankClientScript.js");
    Reader reader = new InputStreamReader(inputStream);
    engine.eval(reader);
} catch (ScriptException e) {
    throw e;
}
}

```

Note the following observations:

- ▶ The example uses the ECMAScript scripting engine.
- ▶ The bank object from the main method is inserted into the binding scope.
- ▶ The script, `BankClientScript.js`, is loaded and then evaluated by the engine.
- ▶ The script itself is similar to the `executeCustomerTransactions` method.

7.6.16 Analyzing the source code

Rational Software Analyzer is a part of the Test and Performance Tools Platform (TPTP) analysis framework. With Rational Software Analyzer, you can run a static analysis of the resources with which you are working to detect violations of rules and rule categories.

In this section, we explain how to work with the Rational Software Analyzer:

- ▶ Creating and editing a static analysis configuration
- ▶ Running a static analysis

Creating and editing a static analysis configuration

For each resource, you can create an analysis configuration that specifies the rules and rule categories that are used when analyzing the resource. A static analysis code review, for example, detects violations of specific programming rules and rule categories and generates a report in the Software Analyzer Results view (Figure 7-31 on page 287).

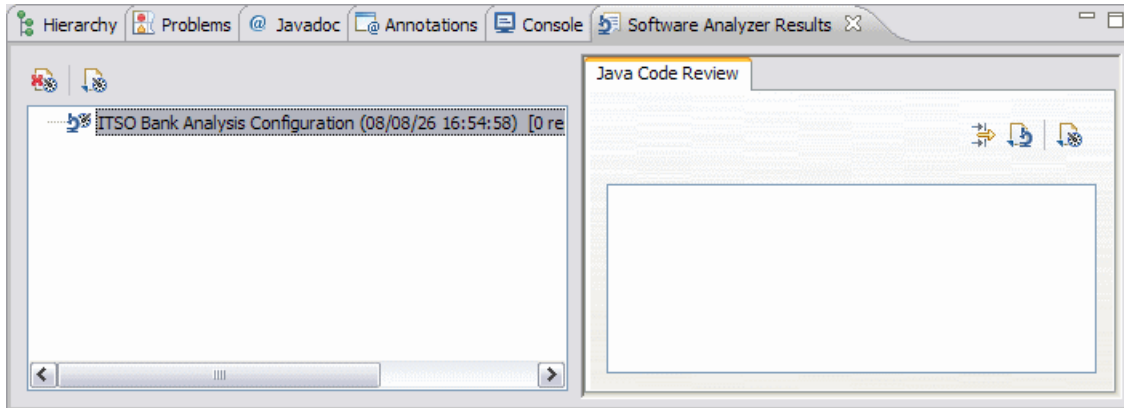





Figure 7-31 Software Analyzer Results view for a Java project

To create an analysis configuration, you must work in a perspective that supports analysis capabilities. The Java and the Debug perspectives support analysis capabilities, by default. In all other perspectives, you can add it. Select **Window** → **Customize Perspective**, click the **Commands** tab, and select **Software Analyzer**.

To create an analysis configuration, follow these steps:

1. Click **Run** → **Analysis**. Alternatively, right-click a project in the Package Explorer and select **Software Analyzer** → **Software Analyzer Configurations**, or click the arrow of the  icon in the toolbar and select **Software Analyzer Configurations**.
2. In the Software Analyzer Configurations window (Figure 7-32 on page 288), complete the following steps:
 - a. Select the  **Software Analyzer** option.
 - b. Click the **New** icon () to create a configuration.
 - c. For the name of the analysis configuration, type **ITS0 Bank Analysis Configuration**.
 - d. Set the scope of the analysis, which has the following options:
 - **Analyze entire workspace:** The rules that you select on the Rules tab are applied to all the resources in your workspace.
 - **Analyze a resource working set:** The rules that you select on the Rules tab are applied to a specific set of projects, folders, or files in your workspace.
 - **Analyze selected projects:** The rules that you select on the Rules tab are applied to the resources in the project that you select.

- Select **Analyze selected projects** and select the **RAD80Java** project.
- e. Click **Apply**.

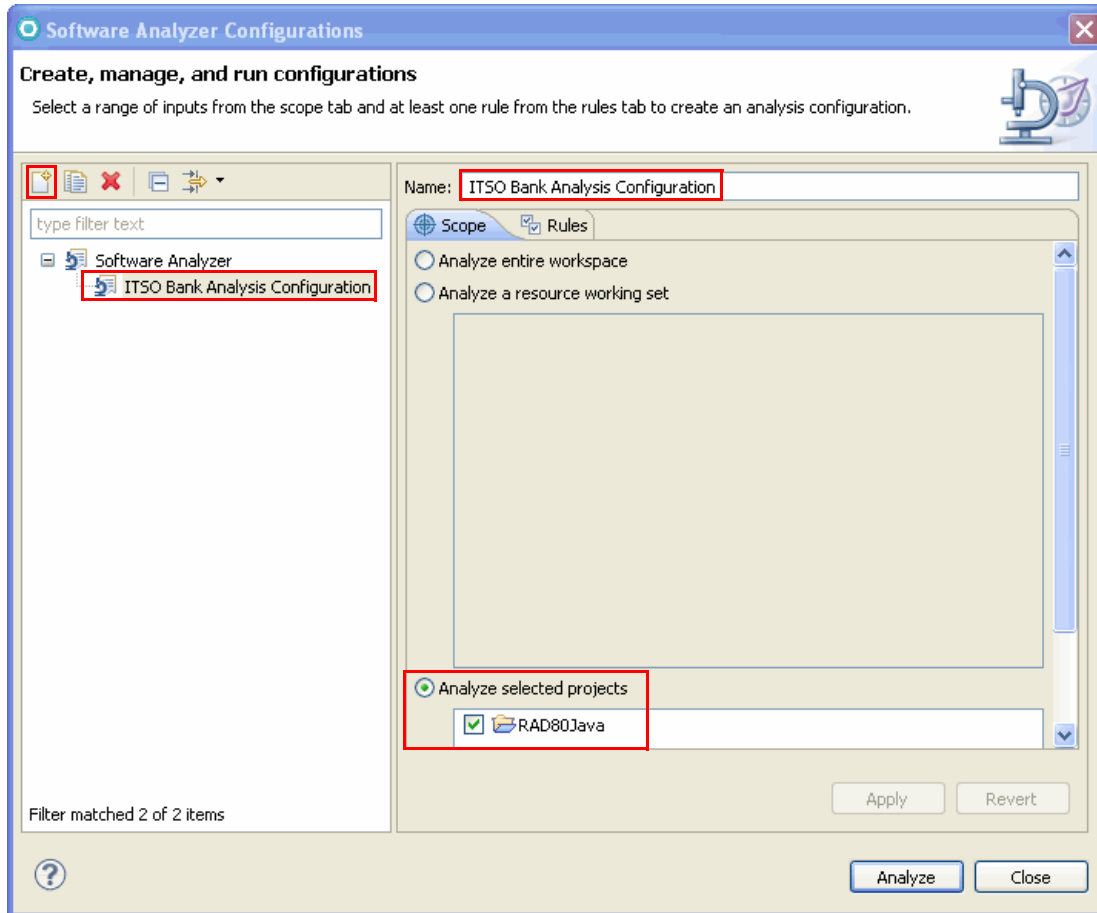


Figure 7-32 Software Analyzer Configurations: Create, manage, and run configurations

- f. Click the **Rules** tab (Figure 7-33 on page 289) to specify the rule categories, rules, or rule sets to apply during the analysis:
- For Rule Sets, select a defined rule set, for example, **Java Quick Code Review**, and click **Set** to configure the domains and rules.
 - For Analysis Domains and Rules, expand the tree and select domains and rules. For example, select **Java Code review** → **Design Principles**.

Setting a rule set selects a subset of domains and rules. In our case, several **J2SE Best Practices** rules are preselected. Expand that domain to see the selected rules and select the box for select all tests.

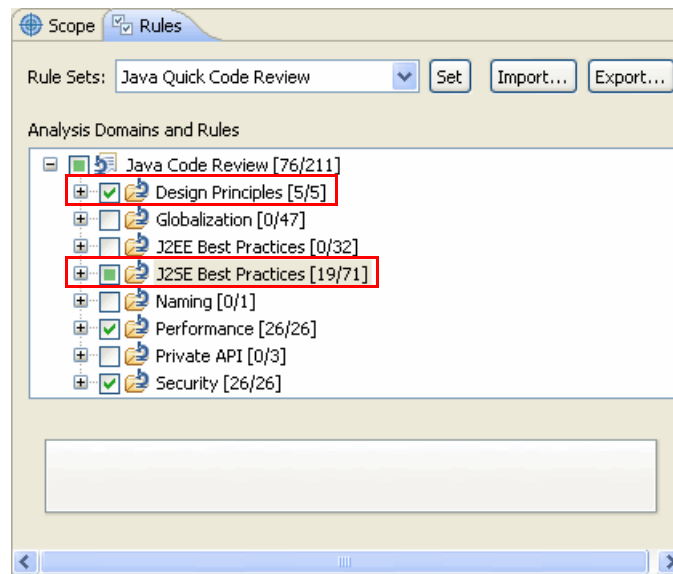


Figure 7-33 Analysis Domains and Rules

- iii. Select **Performance** and **Security**.
- iv. Click **Apply**.

Running a static analysis

You can analyze your source code using the analysis configurations that you created. To run a static analysis, select an existing configuration or create a new configuration, and click **Analyze**.

While the analysis runs, the Software Analyzer Results view opens. If your source code does not conform to the rules in the analysis configuration, the view populates with results. The results are listed in chronological order and are grouped into the same categories that you specified in the analysis configuration.

If you run the analysis for the RAD80Java project, no problems are reported. If you run the analysis for the RAD80EJB project (from Chapter 12, “Developing Enterprise JavaBeans (EJB) applications” on page 577), one problem is reported (Figure 7-34 on page 290).

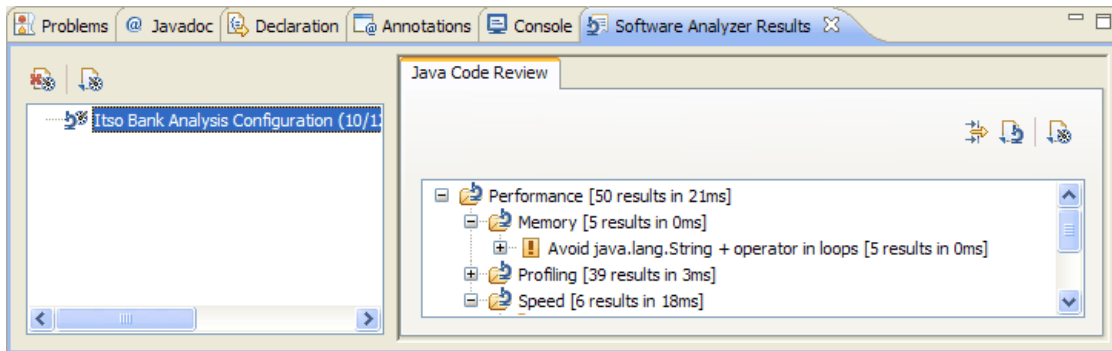


Figure 7-34 Software Analyzer Results view

Static analysis results

A *static analysis result* is a concise explanation of a rule violation. The result is a line item in the Software Analyzer Results view that shows that the resource does not comply with the rules that you applied.

A result is not necessarily a problem, mistake, or bug, but you have to evaluate each result in the list to determine what action, if any, you have to take. If the result is a problem that has a trivial solution, the author of the rule might have provided a quick fix that automatically corrects the resource.

To locate a problem, right-click an entry in the Software Analyzer Results view and select **View Result**. This action opens the Java source file with the problem code highlighted.

If a quick fix is provided, right-click an entry and select **Quick Fix**. The source code is changed, and the entry disappears from the list.

7.6.17 Debugging a Java application

For details about debugging an application, see Chapter 28, “Debugging local and remote applications” on page 1461.

7.7 Using the Java scrapbook

You can use the scrapbook feature to quickly run and test Java code without creating an executable testing class. Snippets of Java code can be entered in a scrapbook page and evaluated by selecting the code and running it.

You can add a scrapbook page to any project and package. A scrapbook page uses the `.jpage` extension to distinguish it from a normal Java source file. To create and run a scrapbook page, follow these steps:

1. Right-click a package (**itso.rad80.bank.client**) in the Package Explorer and select **New** → **Other** → **Java** → **Java Run/Debug** → **Scrapbook Page**.
2. Enter a file name (TestScrapBook) and click **Finish**. We have already imported this scrapbook.
3. When the scrapbook page opens in the Java editor, enter the snippet Java code.

Example 7-7 contains two short snippets. The first snippet is related to the ITSO Bank application and is based on the `BankClient` class `main` method. The second snippet is a simple code snippet to produce a multiplication table.

Example 7-7 Java scrapbook examples

```
// ITSO Bank Snippet
itso.rad80.bank.ifc.Bank oITSOBank =
    itso.rad80.bank.impl.ITSOBank.getBank();
System.out.println("\nITSO Bank is listing all customers status");
System.out.println(oITSOBank.getCustomers() + "\n");
for (itso.rad80.bank.model.Customer
    customer:oITSOBank.getCustomers().values())
{
    System.out.println("Customer: "+ customer);

    System.out.println(oITSOBank.getAccountsForCustomer(customer.getSsn(
    )));
}

// Multiplication Table Snippet
String line;
int result;

for (int i = 1; i <= 10; i++) {
    line ="row " + i + ": ";




    // begin inner for-loop
    for (int j = 1; j <= 10; j++) {
        result = i*j;
        line += result + " ";
    } // end inner for-loop
    System.out.println(line);
}
```

Important: All classes that are not from the `java.lang` package must be fully qualified, or you have to set import statements:

1. Right-click in the scrapbook page editor and select **Set Imports**.

2. For the example, add the following types and packages:

```
itso.rad80.bank.model.*  
itso.rad80.bank.ifc.Bank  
itso.rad80.bank.impl.ITS0Bank
```

4. To execute, display, or inspect a snippet:
- Select the code of the // Multiplication Table Snippet, right-click, and select **Execute**, or press Ctrl+U, or click the  icon in the toolbar. All output is displayed in the Console view.
 - Select the // ITS0 Bank Snippet, right-click, and select **Display**, or click the  icon in the toolbar. Again, all output is displayed in the Console view.
 - Select the // ITS0 Bank Snippet, right-click, and select **Inspect**, or press Ctrl+Shift+I, or click the  icon in the toolbar. Again, all output is displayed in the Console view. But, in addition, an expression box opens, which allows you to inspect the current variables. Pressing Ctrl+Shift+I again opens the Expression view, which you can find in the Debug perspective, as described in Chapter 28, “Debugging local and remote applications” on page 1461.

Reminder: You must select the code before you can execute, display, or inspect a snippet in the scrapbook page.

5. Click the  icon in the Console view to end the scrapbook evaluation.

7.7.1 Pluggable Java Runtime Environment

With Rational Application Developer, you can run Java projects under separate versions of the JRE. New JREs can be added to the workspace, and projects can be configured to use any of the available JREs. By default, Rational Application Developer uses and provides projects with support for IBM Java Runtime Environment V6.0.

To add another JRE to the workspace, follow these steps:

1. Select **Window** → **Preferences**, and in the Preferences window, select **Java** → **Installed JREs**.
2. Click **Add** to add a new JRE to the workspace.
3. Click **Browse** and select the home directory of the JRE that you want to use.
4. Click **OK**. The new added JRE is now available in the list. By default, the selected JRE is added to the build path of newly created Java projects.

The JRE that is used to run a program can also be selected in the Run Configurations window (Figure 7-35):

1. Select **Run** → **Run Configurations**.
2. Select an existing Java application run configuration.
3. Click the **JRE** tab, select **Alternate JRE**, and change the JRE.

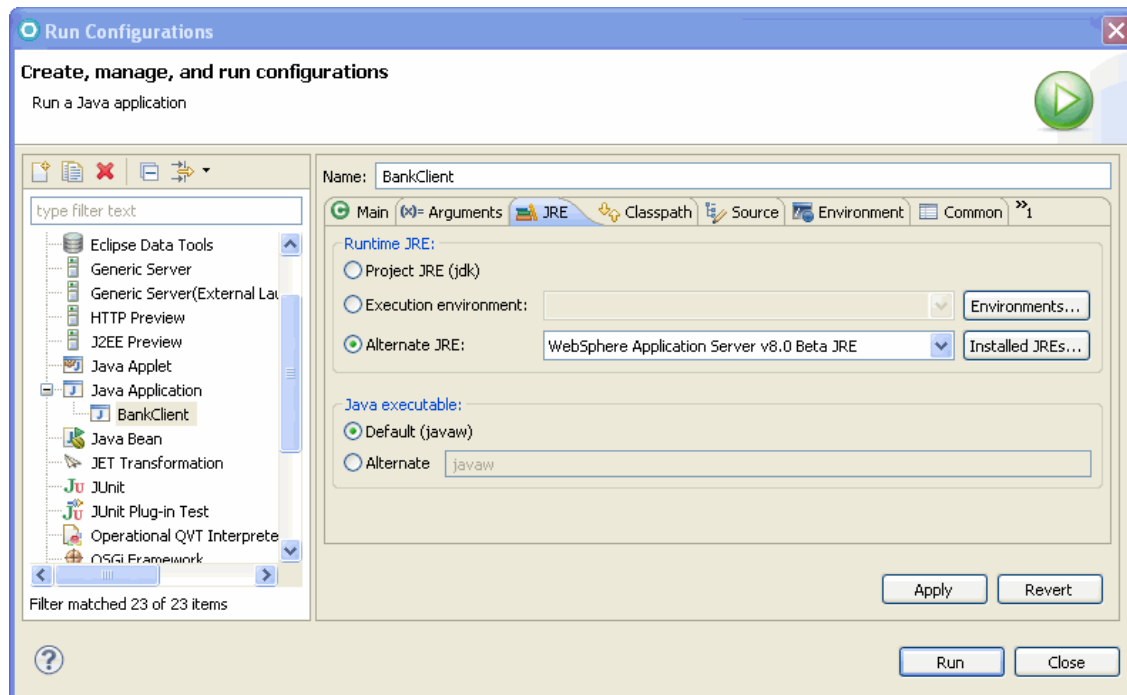


Figure 7-35 Run Configurations JRE tab

7.7.2 Exporting Java applications to a JAR file

You can export a Java application to a JAR file that can be run outside Rational Application Developer using a JRE in a Microsoft Windows command prompt. We explain how to export and run the ITSO Bank application.

To export the ITSO Bank application code to a JAR file, follow these steps:

1. Right-click the **RAD80Java** project and select **Export**.
2. In the Export window, select **Java** → **JAR file** and click **Next**.
3. In the JAR Export window (Figure 7-36 on page 295), complete the following actions:
 - a. Select the **RAD80Java** project.
 - b. Select **Export generated class files and resources** (default).
 - c. Select **Export Java source files and resources**.

Exporting the source: We select to export the source to demonstrate later how to import a JAR file into a project. It is not necessary or desirable to include Java sources in a JAR file for execution.

- d. For the JAR file, browse and select **C:\ITSOBankApplication.jar**.
- e. Select **Compress the contents of the JAR file** (default).
- f. Clear all other check boxes.

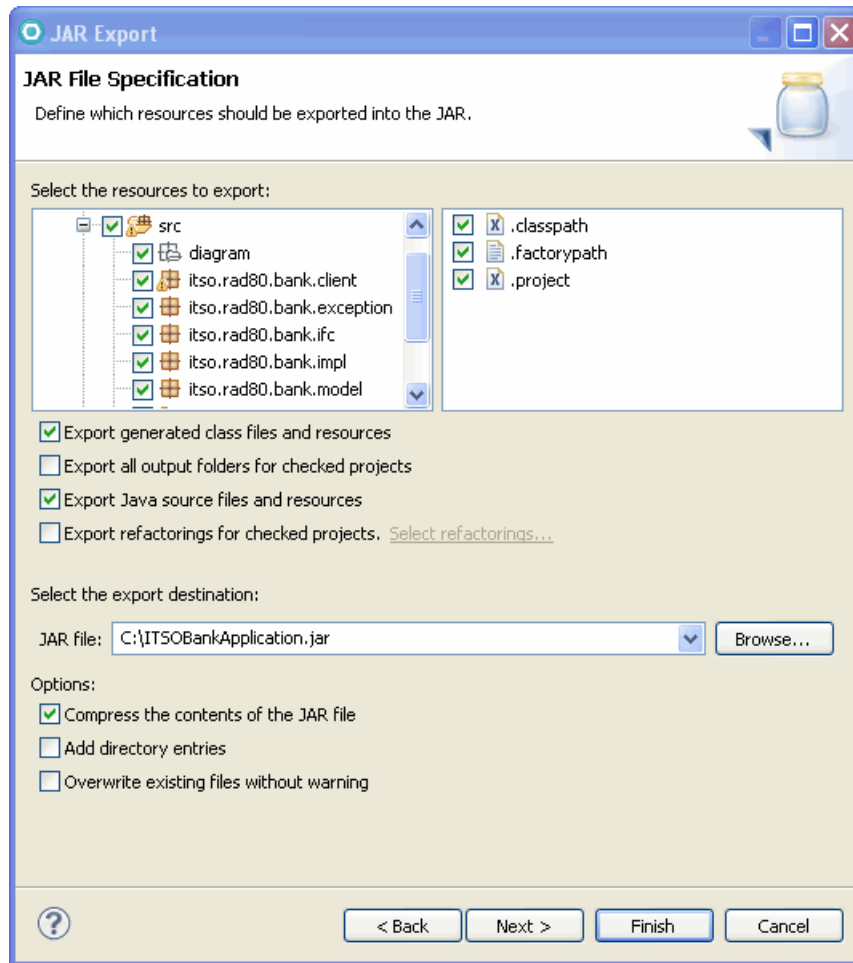


Figure 7-36 JAR Export window

4. In the JAR Packaging Options window, accept the default values and click **Next**.
5. In the JAR Manifest Specification window, for the Main class, click **Browse** and select the **BankClient** class.
6. Click **Finish** to export the entire Java project as a JAR file.
7. If a warning message window opens, click **OK**.

7.7.3 Running Java applications that are external to Rational Application Developer

After you export the Java application as a JAR file, you can run the Java application on any installed JRE on your system (at least as long as no version conflicts exist).

Adding the JRE to the path: Ensure that the JRE is set in the Microsoft Windows environment variable called PATH. You can add the JRE to the path by entering the following command in the Microsoft Windows command prompt:

```
set path=%path%;{JREInstallDirectory}\bin
set path=%path%;C:\IBM\SDP80Beta\jdk\bin
```

To run a Java application that is external to Rational Application Developer on a Microsoft Windows system, follow these steps:

1. Open a Command Prompt and navigate to the directory to which you have exported the JAR file, for example, C:\.
2. Enter the following command to run the ITSO Bank application:

```
java -jar ITSOBankApplication.jar
```

The main method of BankClient is run. Figure 7-37 on page 297 shows the results.

```
C:\>java -jar ITS0BankApplication.jar
*****
Transactions STARTED
*****
<<Using PLAIN JAVA to access bank Java objects!>>.

System is going to add new customer...
xxx-xx-xxxx Mr Napoli Juan has been successfully added.

ITS0 Bank is opening two new accounts for customer xxx-xx-xxxx Mr Napoli Juan...
Account 1 and account 2 have been successfully opened for xxx-xx-xxxx Mr Napoli
Juan.

ITS0 Bank - listing all account information of xxx-xx-xxxx Mr Napoli Juan...
[Account 1: --> Current balance: $10000.00, Account 2: --> Current balance: $112
34.23]

ITS0 Bank is going to make credit of $2500.00 to account 1...
Account 1 has successfully credited by $2500.00.

ITS0 Bank is going to make debit of $1234.23 to account 2...
Account 2 has successfully debited by $1234.23.

ITS0 Bank is listing all account information of xxx-xx-xxxx Mr Napoli Juan...
[Account 1: --> Current balance: $12500.00
  Transactions:
  1. itso.rad75.bank.model.Credit@70727072
, Account 2: --> Current balance: $10000.00
  Transactions:
  1. Debit: --> Amount $1234.23 on 2008-08-22 17:56:32.001
```

Figure 7-37 Output from running ITS0BankApplication.jar in Command Prompt

7.7.4 Importing Java resources from a JAR file into a project

You can import Java resources from a JAR file into an existing Java project in the workspace. We use the ITS0BankApplication.jar file, which we create in 7.7.2, “Exporting Java applications to a JAR file” on page 293.

To import Java resources from a JAR file into a project, follow these steps:

1. Create a Java project called RAD80JavaImport with the default options.
2. In the Package Explorer, right-click the **RAD80JavaImport** project and select **Import**.
3. In the Import window, select **General** → **Archive File** and click **Next**.
4. In the Import - Archive File window, click **Browse** and locate the JAR file (for example, c:\ITS0BankApplication.jar).
5. Clear the files .classpath and .project and the folder META-INF. These files are created when required.
6. For Into folder, select **RAD80JavaImport/src** and click **Finish**.
7. Test the imported Java project. Then select and run the BankClient class from the Package Explorer.

7.7.5 Javadoc tooling

Javadoc is a useful tool in the Java Development Kit that is used to document Java code. It generates web-based (HTML files) documentation of the packages, interfaces, classes, methods, and fields.

Rational Application Developer has a Javadoc view, which allows you to browse formatted Javadoc. In the Java perspective, the Javadoc view is context-sensitive. It only shows the Javadoc that is associated with the Java element where the cursor is currently located within the Java editor.

To demonstrate the use of Javadoc, we use the RAD80JavaImport project that we imported:

1. Open the **Javadoc** view in the Java perspective if it is not already open.
2. Open the **BankClient** class in the Java editor. Notice that, when the cursor selects a type, its Javadoc is shown in the Javadoc view.
3. Select the **BigDecimal** type and the Javadoc view changes to the documentation that is associated with `BigDecimal` (Figure 7-38).

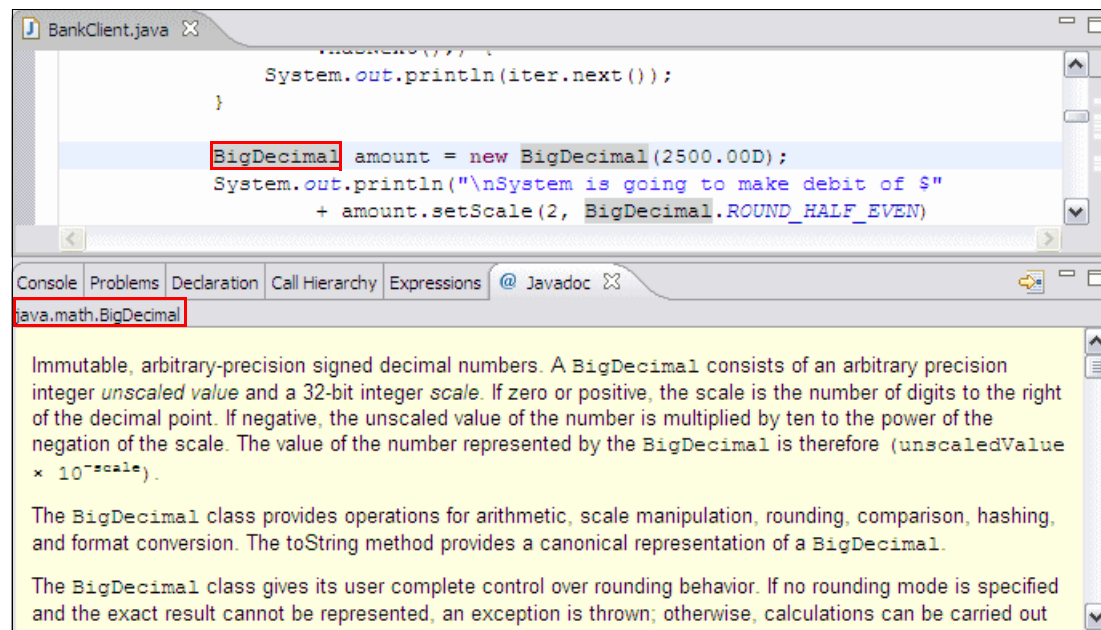


Figure 7-38 Javadoc view: Context sensitive (*BigDecimal*)

7.8 Generating the Javadoc

You can generate the Javadoc from an existing Java project. Rational Application Developer supports the following types of Javadoc generation:

- ▶ Generating the Javadoc from an existing project
- ▶ Generating the Javadoc from an Ant script
- ▶ Generating the Javadoc with diagrams from existing tags
- ▶ Generating the Javadoc with diagrams automatically

7.8.1 Generating the Javadoc from an existing project

To generate the Javadoc from an existing Java project, follow these steps:

1. Right-click the **RAD80Java** project in the Package Explorer and select **Export** → **Java** → **Javadoc**, or select **Project** → **Generate Javadoc**.
2. In the Javadoc Generation window (Figure 7-39 on page 300), complete the following actions:
 - a. Accept the predefined value for the Javadoc command.
 - b. Select **Public** for Create Javadoc for members with visibility (default).
 - c. Select **Use Standard Doclet**. Alternatively, you can specify a custom doclet with the name of the doclet and the class path to the doclet implementation.
 - d. For Destination, accept the default value **{workspaceDirectory}\RAD80Java\doc**, which optionally generates the Javadoc in the doc directory of the current project.

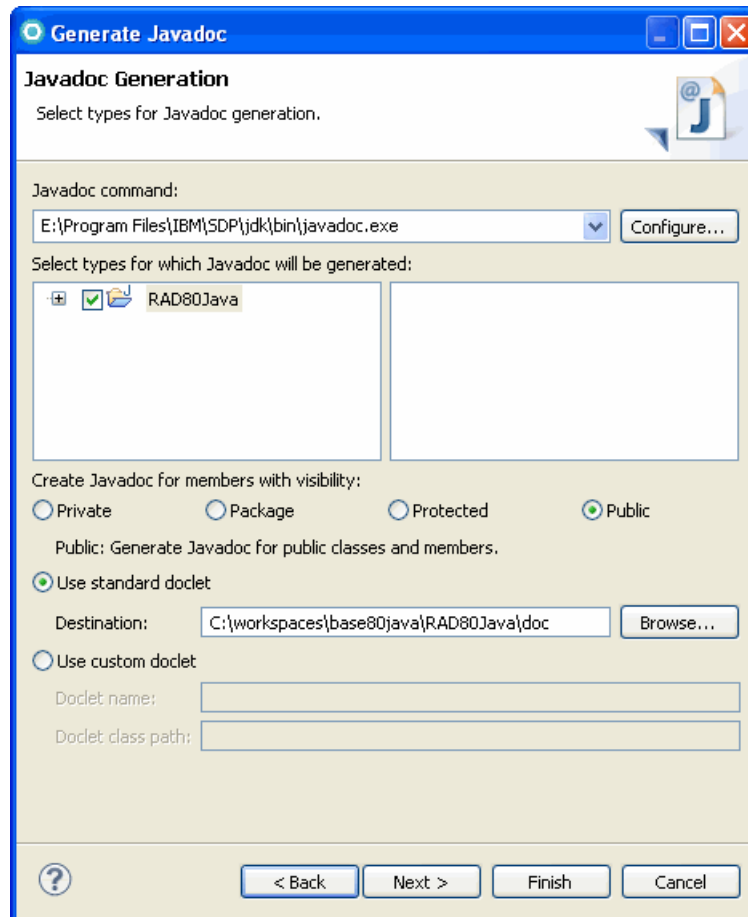


Figure 7-39 Javadoc Generation window

3. In the Configure Javadoc arguments for standard doclets window, accept the default settings and click **Next**.
4. In the Configure Javadoc arguments window, enter the following data:
 - a. For JRE source compatibility, select **1.6**, because in the project, we use generic types that are only supported by Java Development Kit (JDK) Version 1.6 and later versions.
 - b. Select **Save the settings for this Javadoc export as an Ant script** and accept the destination **{workspace}\RAD80Java\javadoc.xml**.
5. Click **Finish** to generate the Javadoc.
6. When prompted to update the Javadoc location, click **Yes to all**.

7. When prompted that the Ant file will be created, click **OK**.
8. Right-click **index.html** (in RAD80JavaImport/doc) and select **Open With** → **Web Browser** to open the Javadoc in a browser (Figure 7-40).

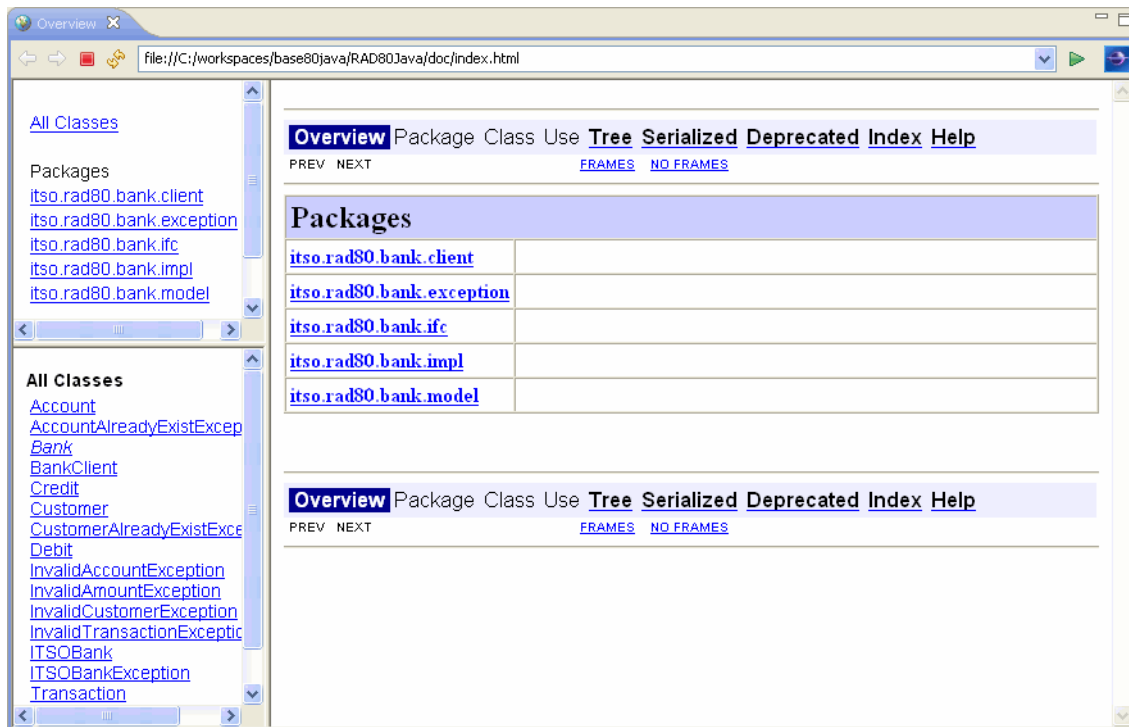


Figure 7-40 Javadoc output generated from the Javadoc wizard

7.8.2 Generating the Javadoc from an Ant script

In 7.8.1, “Generating the Javadoc from an existing project” on page 299, we selected **Save Settings for this Javadoc export as an Ant script**. Selecting this option generated the `javadoc.xml` Ant script, which can be used to invoke the Javadoc command.

To generate the Javadoc from an Ant script, right-click **javadoc.xml** in the Package Explorer and select **Run As** → **Ant Build**. The Javadoc generation process starts. If you cannot see the new generated doc folder in the project, select the project and press F5 to refresh the view.

7.8.3 Generating the Javadoc with diagrams from existing tags

Rational Application Developer enables you to embed the `@viz.diagram` tag into the Javadoc on the class, interface, or package level. The `@viz.diagram` tag assumes that the diagram being referenced is placed in the same folder as the Java file containing the `@viz.diagram` tag, and the wizard then exports that diagram into a `.gif`, `.jpg`, or `.bmp`, and embeds it into the generated Javadoc.

Example 7-8 shows the use of the `@viz.diagram` tag in the `BankClient` class.

Example 7-8 BankClient class with an @viz.diagram tag

```
package itso.rad80.bank.client;
...
/**
 * @viz.diagram ITSOBank-ClassDiagram.dnx
 */
public class BankClient {
    public static void main(String[] args) {
        try {
            ...
        }
    }
}
```

To generate the Javadoc with diagrams from existing tags, follow these steps:

1. Add the `@viz.diagram` tag to the source code, as shown in Example 7-8, and copy the `ITSOBank-ClassDiagram.dnx` file from the `diagram` folder to the `itso.rad80.bank.client` package.

Restriction: For web applications, this action results in the class diagrams being packaged into the WAR file with the compiled Java code. We found the following work-arounds:

- ▶ Manually remove these diagrams from the WAR file after exporting.
- ▶ Configure an exclusion filter for the EAR export feature.

2. Select the project in the Package Explorer and select **Project** → **Generate Javadoc with Diagrams** → **From Existing tags**.

Java Modeling: To see this action, you must enable the Java Modeling capability. Select **Window** → **Preferences** → **Advanced** → **Java** → **Java Modeling**.

3. In the Javadoc Generation window (Figure 7-39 on page 300), complete the following actions:

- a. Accept the predefined value for the Javadoc command.
 - b. Select **Public** for Create Javadoc for members with visibility (default).
 - c. Select **Use Standard Doclet**. Alternatively, you can specify a custom doclet with the name of the doclet and the class path to the doclet implementation.
 - d. For Destination, accept the default value **{workspaceDirectory}\RAD80Java\doc**, which optionally generates the Javadoc in the doc directory of the current project.
4. In the next window, click **Next**.
 5. In the Configure Javadoc arguments window, for JRE source compatibility, select **1.6**.
 6. In the Choose diagram image generation options window, accept the default settings and click **Finish**.
 7. When prompted to update the Javadoc location, click **Yes to all**.
 8. Open the Javadoc (**RAD80Java\Import\doc\index.html**) in a browser. Verify that a diagram has been added to the generated Javadoc for the `BankClient` class by selecting the **BankClient** class in the All Classes pane.

7.8.4 Generating the Javadoc with diagrams automatically

If you do not have diagrams that you want to embed in the generated Javadoc, you can let Rational Application Developer generate diagrams for you and embed them in the Javadoc.

To generate the Javadoc with diagrams automatically, follow these steps:

1. Select the project in the Package Explorer and select **Project** → **Generate Javadoc with Diagrams** → **Automatically**.
2. In the Generate Javadoc with Diagrams Automatically window (Figure 7-41 on page 304), complete the following actions:
 - a. For the Javadoc command, enter the `{JDKInstallDirectory}\bin\javadoc.exe` path.
 - b. For Diagrams, keep the default selections.
 - c. Optional: Select **Contribute diagrams and diagram tags to source** if you want the `@viz.diagram` tags to be stored in the Java sources and the generated diagrams to be stored in the packages of the Java sources.
 - d. Click **Finish** to generate the Javadoc.

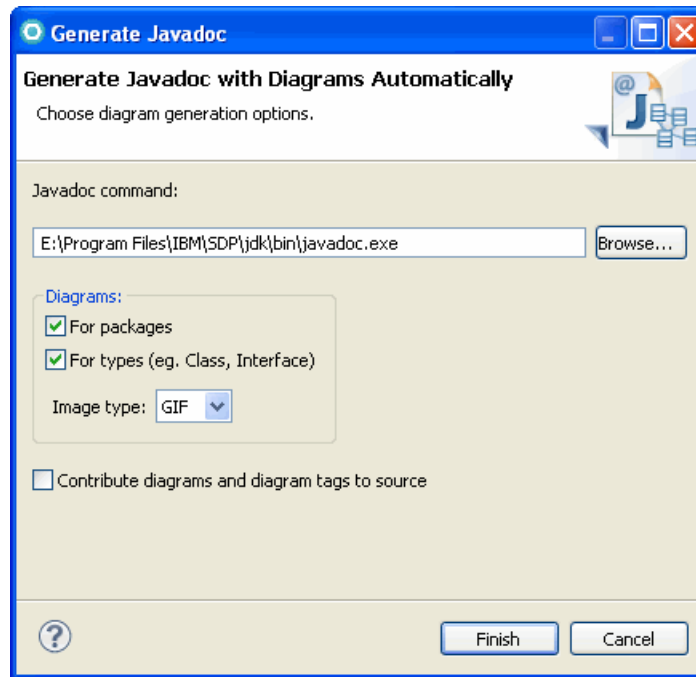


Figure 7-41 Generate Javadoc with Diagrams Automatically window

3. Open the Javadoc and browse the classes with the generated diagrams.

7.9 Java editor and rapid application development

Rational Application Developer contains several features that ease and expedite the code development process. These features are designed to make life easier for both experienced and novice Java programmers by simplifying or automating many common tasks.

This section is organized into the following topics:

- ▶ Navigating through the code
- ▶ Source folding
- ▶ Type hierarchy
- ▶ Smart insert
- ▶ Marking occurrences
- ▶ Smart compilation
- ▶ Java and file search
- ▶ Working sets

- ▶ Quick fix
- ▶ Quick assist
- ▶ Content assist
- ▶ Import generation
- ▶ Adding constructors
- ▶ Using the delegate method generator
- ▶ Refactoring

7.9.1 Navigating through the code

In this section, we highlight the use of the Outline view, Package Explorer, and bookmarks to navigate through the code.

Navigating the code by using the Outline view

The Outline view shows an outline of a structured file that is currently open in the editor area and lists structural elements. The contents of the Outline view are editor-specific.

For example, in a Java source file, the structural elements are package name, import declarations, class, fields, and methods. We use the RAD80Java project to demonstrate the use of the Outline view to navigate through the code:

1. From the Package Explorer, select and expand **RAD80Java** → **src** → **itso.rad80.bank.model**.
2. Double-click **Account.java** to open the class in the Java editor.

By selecting elements in the Outline view, you can navigate to the corresponding point in your code, which allows you to easily find method and field definitions without scrolling through the Java editor (Figure 7-42 on page 306).

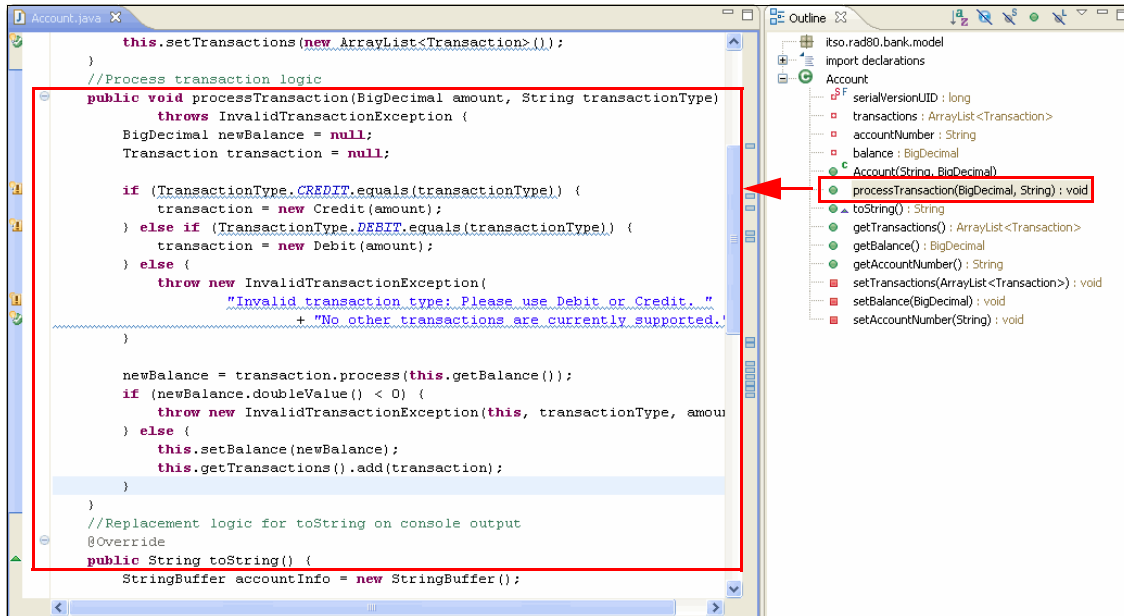


Figure 7-42 Java editor: Outline view for navigation

Navigating the code by using the Package Explorer

You can also use the Package Explorer, which is available by default in the Java perspective, for navigation. The Package Explorer provides you with a Java-specific view of the resources that are shown in the Enterprise Explorer view. The element hierarchy is derived from the project's build paths.


Navigating the code by using bookmarks

Bookmarks are another simple way to navigate to resources that you frequently use. The Bookmarks view shows all bookmarks in the workspace.

Setting a bookmark

To set a bookmark in the code, right-click in the gray sidebar to the left of the code in the Java editor and select **Add Bookmark**, or select **Edit** → **Add Bookmark**. In the Add Bookmark window, enter the name of the bookmark and click **OK**.

Viewing bookmarks

Bookmarks are indicated by the symbol  in the gray sidebar (Figure 7-43 on page 307) and are listed in the Bookmarks view (Figure 7-44 on page 307). Double-click the bookmark entry in the Bookmarks view to open the file and navigate to the line where the bookmark has been set.

```
//Process transaction logic
public void processTransaction(BigDecimal amount, String
    throws InvalidTransactionException
{
    BigDecimal newBalance = null;
    Transaction transaction = null;

    if (TransactionType.CREDIT.equals(transactionType)) {
        transaction = new Credit(amount);
    }
}
```

Figure 7-43 Java Editor with a bookmark

Showing the Bookmarks view

To see the Bookmarks view (Figure 7-44), select **Window** → **Show View** → **Other** → **General** → **Bookmarks**.

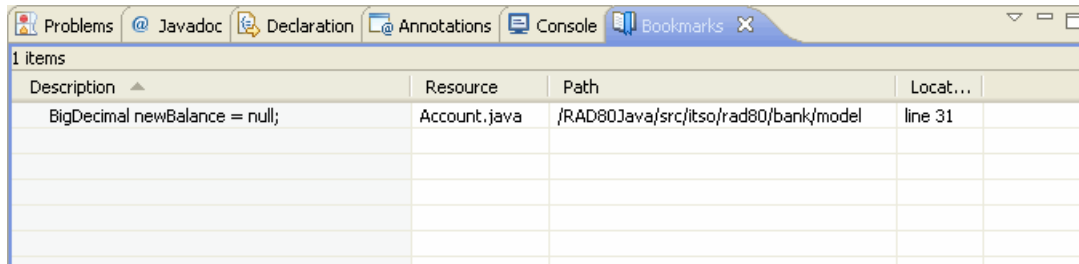



Figure 7-44 Bookmarks view

Deleting bookmarks

To remove a bookmark, right-click the bookmark symbol  in the gray sidebar and select **Remove Bookmark**. Alternatively, right-click the bookmark in the Bookmarks view and select **Delete**.

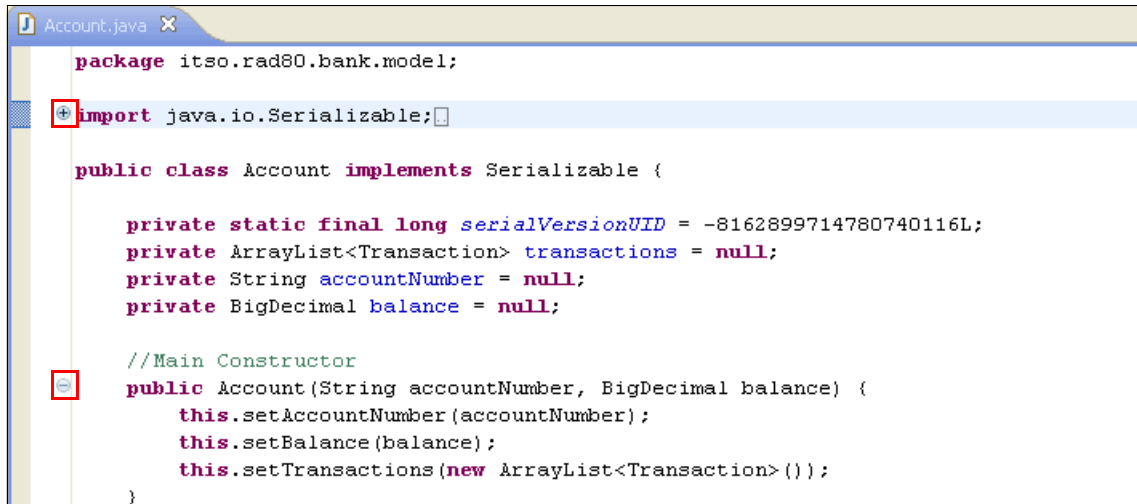
Bookmarks for a file: Bookmarks can be used in any file to provide a quick way of navigating to a specific location. They are not specific to Java code. Select the file in the Enterprise Explorer view and select **Edit** → **Add Bookmark**.

7.9.2 Source folding

Rational Application Developer folds the source of import statements, comments, types, and methods. Source folding can be configured through the Java editor preferences.

To configure the folding feature, select **Window** → **Preferences**. In the Preferences window, select **Java** → **Editor** → **Folding**.

Folded source is marked by a (+) symbol and expanded source is marked by a (-) symbol on the left side of the source code, as shown in Figure 7-45. Click the symbol to fold or expand the source code.



```
Account.java x
package itso.rad80.bank.model;

+ import java.io.Serializable;

public class Account implements Serializable {

    private static final long serialVersionUID = -8162899714780740116L;
    private ArrayList<Transaction> transactions = null;
    private String accountNumber = null;
    private BigDecimal balance = null;

    //Main Constructor
    - public Account(String accountNumber, BigDecimal balance) {
        this.setAccountNumber(accountNumber);
        this.setBalance(balance);
        this.setTransactions(new ArrayList<Transaction>());
    }
}
```

Figure 7-45 Java Editor with source folding

7.9.3 Type hierarchy

With the Java editor, you can quickly view the type hierarchy of a selected type. Select a type with the cursor and press **Ctrl+T** to display the hierarchy (Figure 7-46).

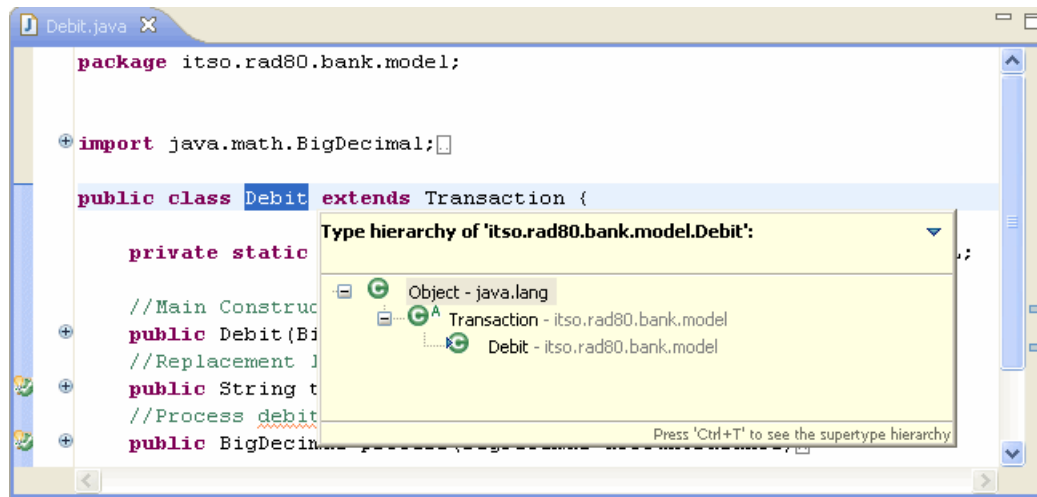


Figure 7-46 Java Editor with a quick type hierarchy view

7.9.4 Smart insert

To toggle the editor between smart insert and insert modes, press **Ctrl+Shift+Insert**. When the editor is in smart insert mode, it provides extra features that are specific to Java. For example, in smart insert mode, when you cut and paste code from a Java source to another Java source, all the needed imports are automatically added to the target Java file.


To configure the smart insert mode, select **Window** → **Preferences**. In the Preferences window, select **Java** → **Editor** → **Typing**, and study the various options.

Notice the small text at the bottom of the window as it changes from Smart Insert to Insert when you press **Ctrl+Shift+Insert**.

7.9.5 Marking occurrences

When enabled, the editor highlights all occurrences of types, methods, constants, non-constant fields, local variables, expressions throwing a declared exception, method exits, methods implementing an interface, and targets of

break and continue statements, depending on the current cursor position in the source code (Figure 7-47). For better orientation in large files, all occurrences are marked with a white line on the right side of the code.

The feature can be enabled and disabled by pressing Alt+Shift+O, or by clicking the  icon in the toolbar. To configure mark occurrences, select **Window** → **Preferences**. In the Preferences window, select **Java** → **Editor** → **Mark Occurrences**.

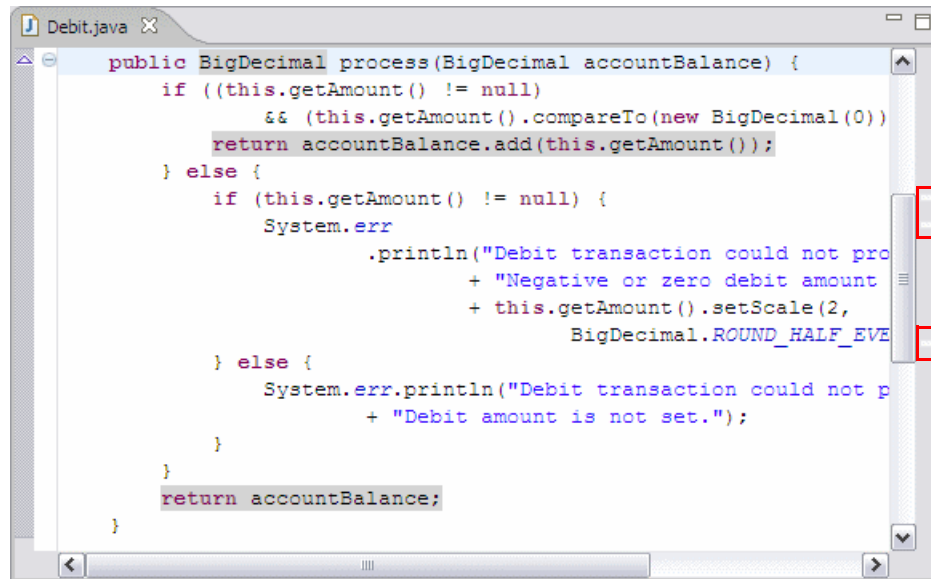


Figure 7-47 Java Editor with mark occurrences (method exits)

7.9.6 Smart compilation

The Java builder in the Rational Application Developer workbench incrementally compiles the Java code in the background as it is changed and shows any compilation errors automatically, unless you disable the automatic build feature. See Chapter 4, “Perspectives, views, and editors” on page 91, for information about enabling and disabling automatic builds and running workbench tasks in the background.

7.9.7 Java and file search


Rational Application Developer provides support for various searches. To display the Search window, click  in the toolbar, or press Ctrl+H. The Search window can be configured to display various searches by clicking **Customize**. In

Figure 7-48, the Search window has been customized to display only the Java Search and File Search tabs.

In the Search window, you can perform the following searches:

- ▶ Java searches operate on the structure of the code.
- ▶ File searches operate on the files by name or text content.
- ▶ With text searches, you can find matches inside comments and strings.

Java searches are faster, because there is an underlying indexing structure for the code.

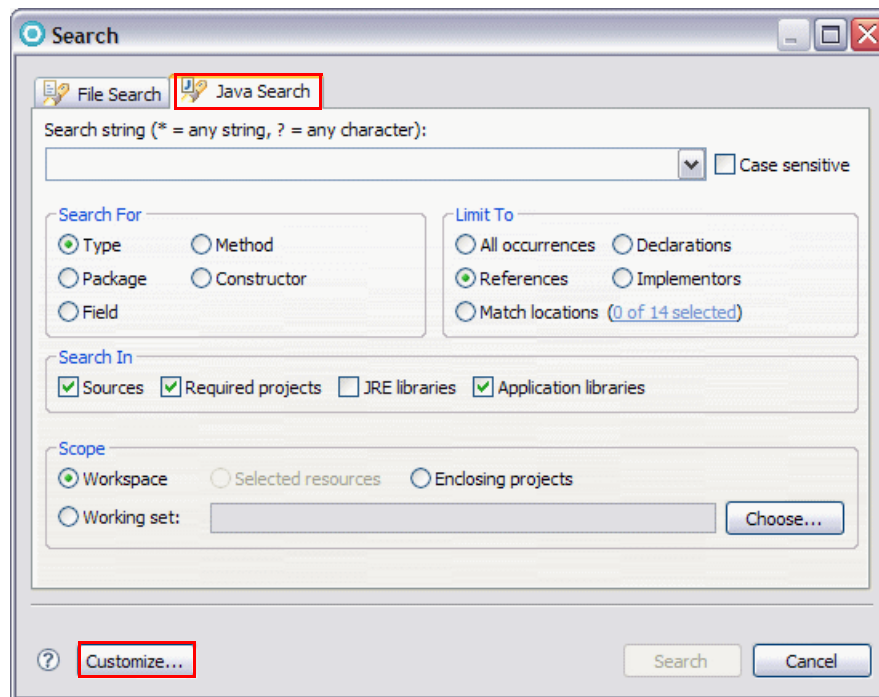



Figure 7-48 Search window [customized]

Performing a Java search from the workbench (example)

To perform a Java search from the workbench, follow these steps:

1. In the Java perspective, click the  icon in the toolbar and select **Search** → **Java**, or press Ctrl+H.
2. Click the **Java Search** tab and complete the following actions (Figure 7-49 on page 312):
 - a. For Search string, type processTransaction.
 - b. Select **Method**.

- c. Select **References**.
- d. Select **Workspace**.
- e. Click **Search**.

While searching, you can click **Cancel** at any time to stop the search. Partial results are shown. The Search view shows the search results.

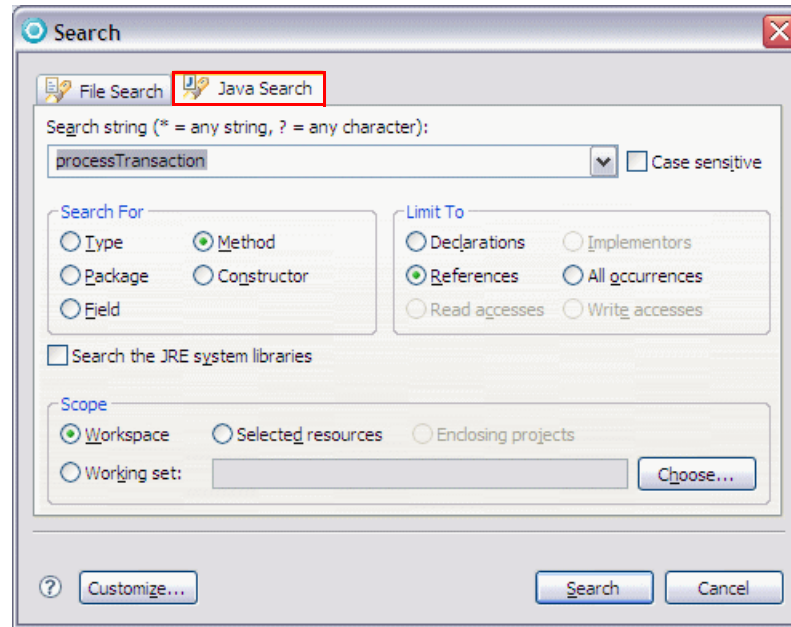


Figure 7-49 Java Search window

3. Click the down arrow (↓) or up arrow (↑) in the toolbar of the Search view to navigate to the next or previous match. If the file in which the match was found is not currently open, it is opened in the Java editor at the position of the match. Search matches are tagged with a ↗ symbol on the left side of the source code line.


Searching from a Java view or editor

You can also perform Java searches from specific views, including the Outline view, Hierarchy view, Package Explorer view, or even the Search view, or from the Java editor.

Right-click the resource for which you are looking in the view or editor and select **References** → **Workspace**, or press Ctrl+Shift+G.

Performing a file search (example)

To perform a file search, follow these steps:

1. In the Java perspective, click  in the toolbar, or select **Search** → **File**, or press Ctrl+H.
2. Click the **File Search** tab and complete the following actions (Figure 7-50):
 - a. For Containing text, enter ROUND_HALF_EVEN.
 - b. For File name patterns, use *.java (default).
 - c. Select **Workspace**.
 - d. Click **Search**.

While searching, you can click **Cancel** at any time to stop the search. Partial results will be shown. The Search view shows the search results.

Searching for all views of a file name pattern: To find all files of a given file name pattern, leave the Containing text field empty.

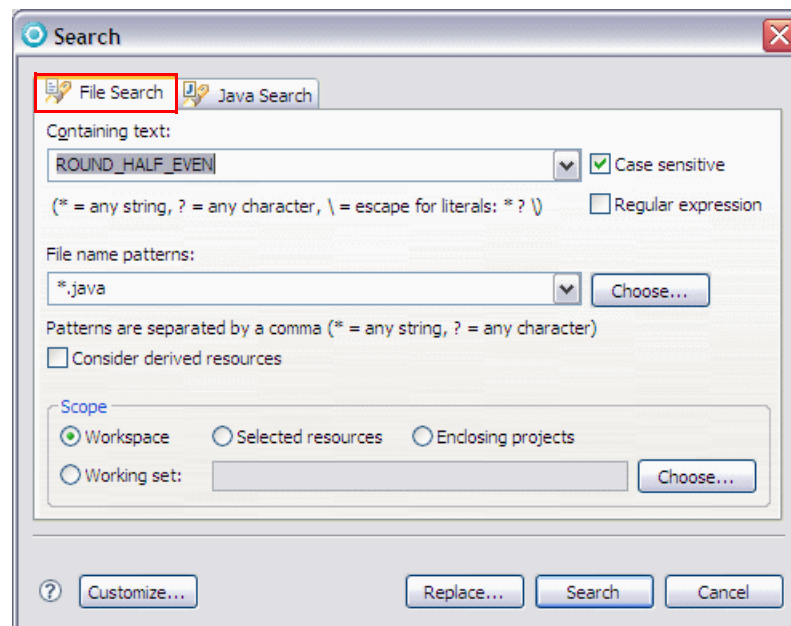




Figure 7-50 File Search window

Viewing previous search results

Click the down arrow (▼) on the right side of the  icon in the Search view toolbar and select one of the previous searches. You can clear the list by selecting **Clear History**.

7.9.8 Working sets

Working sets are used to filter resources by only including the specified resources. They are selected and defined using the view's filter selection window. We use an example to demonstrate the creation and use of a working set:

1. To open the Java Search window, click the  icon in the toolbar and select **Search** → **Java**, or press Ctrl+H.
2. In the Search string field, type `itso.rad80.bank.model.Credit`. For Scope, select **Working set** and then click **Choose**.
3. In the Select Working Set window, click **New** to create a new working set.
4. In the New Working Set window, select **Java** to indicate that the working set includes only Java resources and then click **Next**.
5. In the Java Working Set window (Figure 7-51 on page 315), select only **RAD80JavaImport** → **src** → **itso.rad80.bank.model** and click **Add**. In the Working set name field, type `EntityPackage` and click **Finish**.

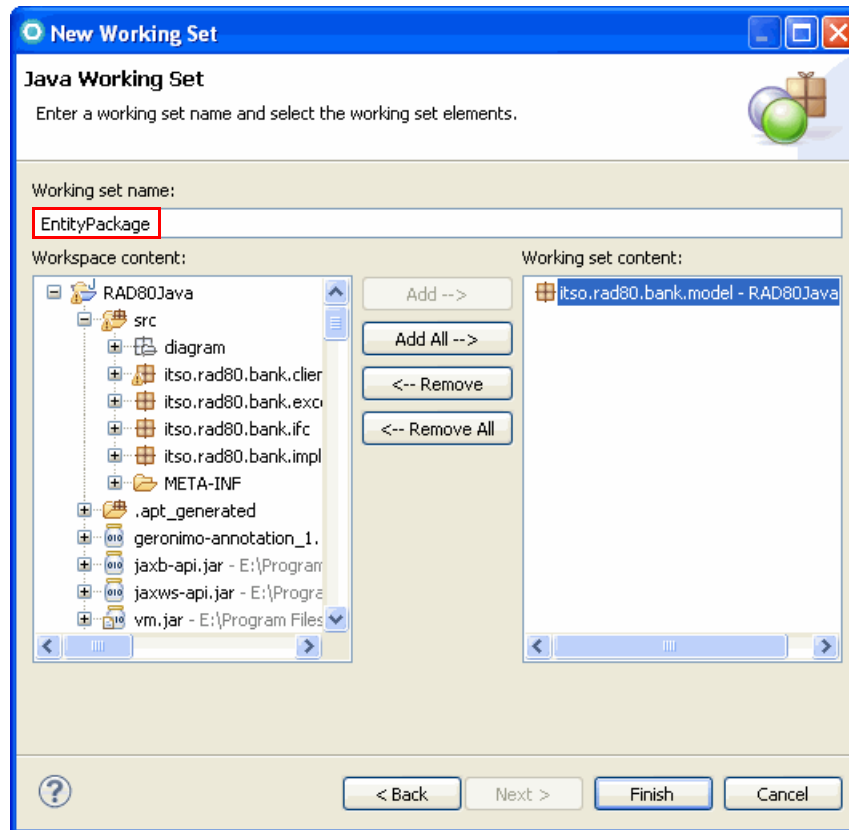


Figure 7-51 New Working Set: Java Working Set window

6. In the Select Working Sets window, select the new **EntityPackage** working set and click **OK**.






You have now created a working set named `EntityPackage` containing Java resources comprised of all the Java files in the `itso.rad80.bank.model` package.

7. Click **Search** to start the search process.

7.9.9 Quick fix

Rational Application Developer offers a quick fix for certain problems that are detected during the code compilation or static code analysis. You can process the quick fix to correct the code.

The following symbols indicate that a quick fix is available:

- ▶ Static code analysis:
 -  Quick fix for a result with a severity level of recommendation
 -  Quick fix for a result with a severity level of warning
 -  Quick fix for a result with a severity level of severe
- ▶ Code compilation:
 -  Quick fix for a warning
 -  Quick fix for an error

To process the quick fix, click the symbol. All suggestions to correct the problem are shown in an overlaid window. As soon as a suggestion is selected, a code preview is shown, so that you can see what will change (Figure 7-52). Double-click one of the suggestions to process the quick fix.

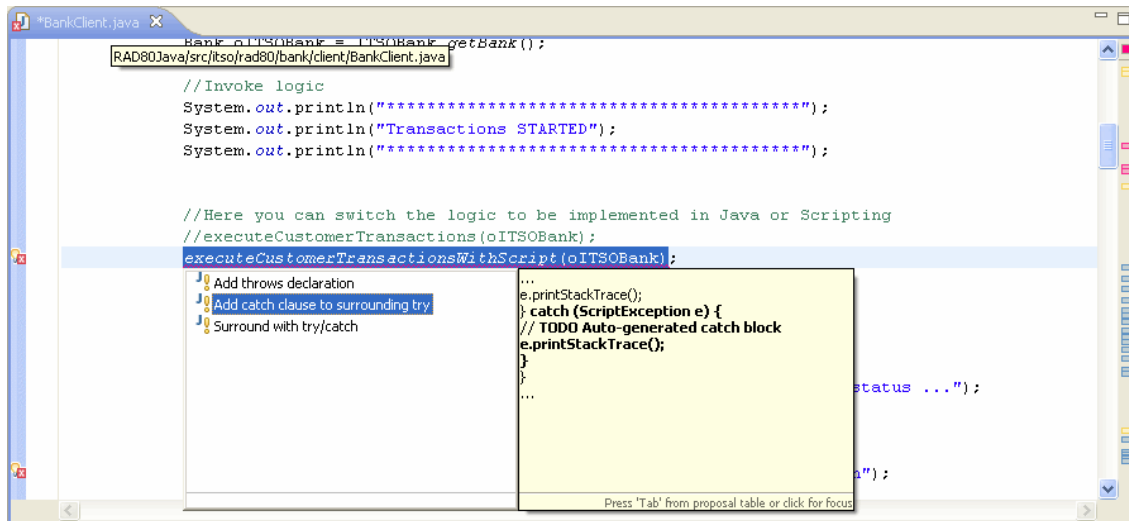



Figure 7-52 Java editor with quick fix window

7.9.10 Quick assist


Rational Application Developer supports quick assists in the Java editor to provide suggestions to complete tasks quickly. Quick assist depends on the current cursor position. When there is a suggestion and quick assist highlighting is enabled, a green light bulb icon () is displayed on the left side of the code line.

Enabling quick assist highlighting

By default, the display of the quick assist light bulb is disabled. To enable it, follow these steps:

1. Select **Window** → **Preferences**.
2. In the Preferences window, select **Java** → **Editor**.
3. Select **Light bulb for quick assists**.

Invoking quick assist

To use the quick assist feature, double-click the  icon or press Ctrl+1 to provide a list of intelligent suggestions. Select a suggestion to complete the task (Figure 7-53).

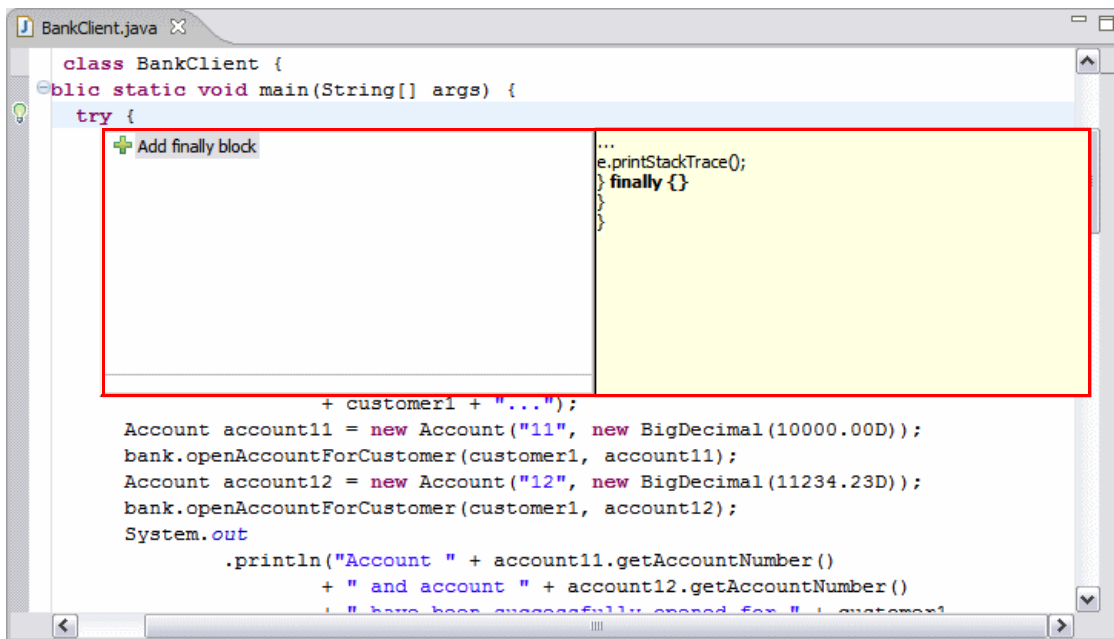


Figure 7-53 Java editor with quick assist feature

7.9.11 Content assist

The content assist feature shows possible code completions that are valid with the current context.

Content assist preferences

To configure the content assist preferences, follow these steps:

1. Select **Window** → **Preferences**.

2. In the Preferences window, select **Java** → **Editor** → **Content Assist**.
3. Modify the settings as desired and click **Apply** and **OK**.

Invoking content assist

Press Ctrl+Spacebar at any point in the Java editor to invoke the content assist. The content assist provides all of the possible code completions that are valid for the current context in an overlaid window (Figure 7-54). Double-click the desired completion, or use the arrow keys to select it, and press Enter.

Tip: If there are still too many possible completions, continue to write the code yourself and the number of suggestions becomes smaller.

You can also use content assist to insert or complete Javadoc tags.

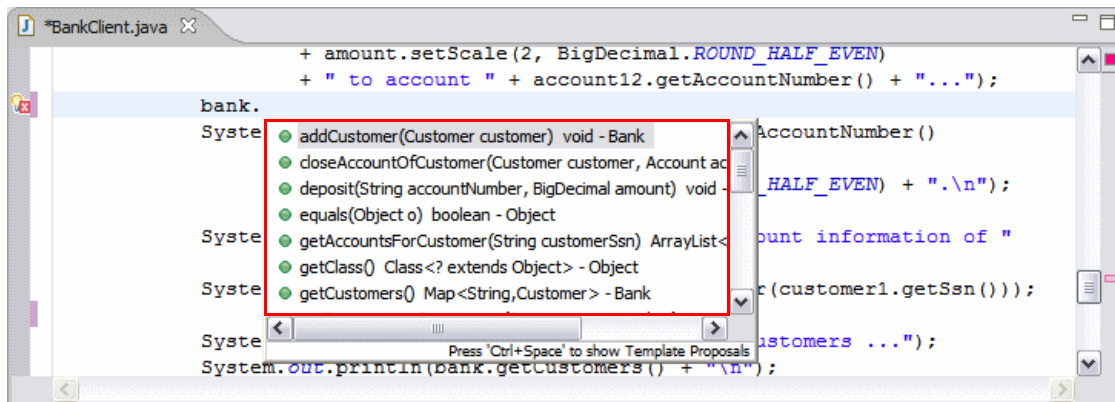


Figure 7-54 Java editor with content assist

7.9.12 Import generation

The Java editor simplifies the task of finding the correct import statements to use in the Java code.

Right-click the unknown type in the code and select **Source** → **Add Import**, or select the type and press Ctrl+Shift+M. If the type is unambiguous, the import statement is directly added. If the type exists in more than one package, a window with all the types is displayed and you can select the correct type for the import statement.

Figure 7-55 on page 319 shows an example where the selected type (BigDecimal) exists in several packages. After you have determined that the

java.math package is what you want, double-click the entry in the list or select it and click **OK** and the import statement is generated in the code.

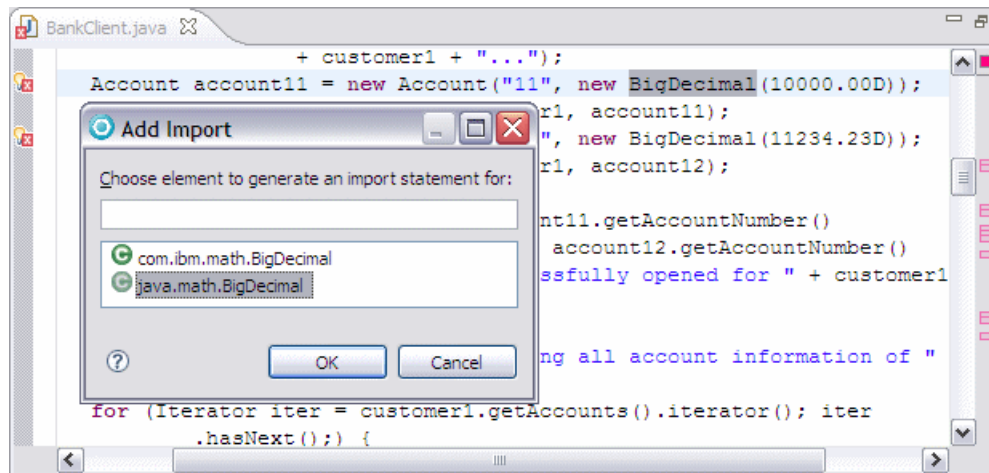


Figure 7-55 Java editor with import generation

You can also add the required import statements for the whole compilation unit. Right-click a project, package, or Java type in the Package Explorer and select **Source** → **Organize Imports**. Or select the project, package, or Java type and press Ctrl+Shift+O. The code in the compilation unit is analyzed and the appropriate import statements are added.

7.9.13 Adding constructors

By using the constructors feature, you can automatically add constructors to the open type. The following constructors can be added:

- ▶ Constructors from superclass
- ▶ Constructor using fields

Constructors from superclass

Add any or all of the constructors defined in the superclass for the currently opened type. Right-click anywhere in the Java editor and select **Source** → **Add Constructors from Superclass**. Select the constructors that you want to add to the current opened type and click **OK** (Figure 7-56 on page 320).

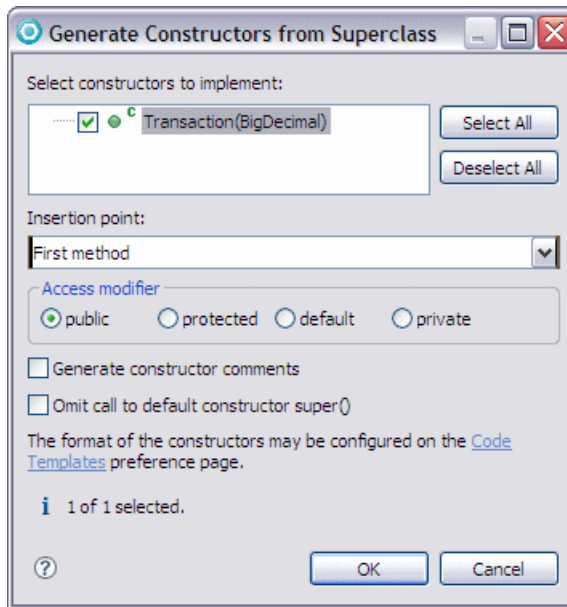


Figure 7-56 Generate Constructors from Superclass window

Constructor using fields

The Constructor using Fields option adds a constructor that initializes any or all of the defined fields of the currently opened type. Right-click anywhere in the Java editor and select **Source** → **Add Constructors using Fields**. Select the fields that you want to initialize with the constructor and click **OK** (Figure 7-57 on page 321).

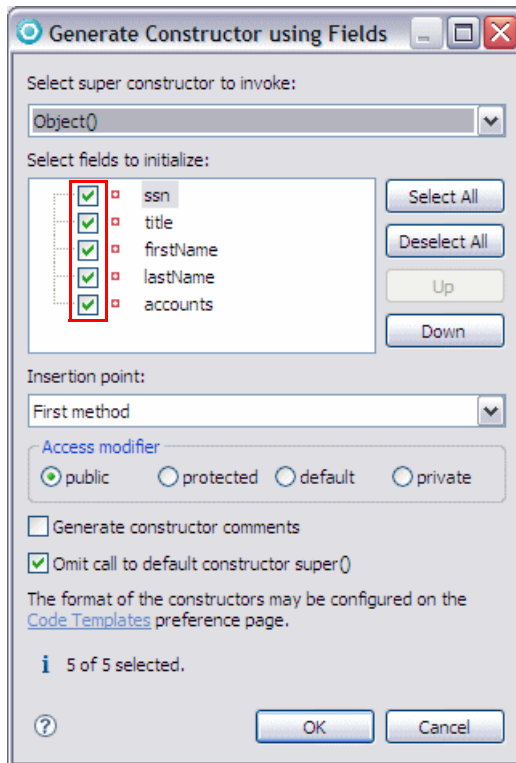


Figure 7-57 Generate Constructor using Fields window

Example 7-9 shows the constructor code that might generate with the settings that are shown in Figure 7-57.

Example 7-9 Generated constructor code

```
public Customer(String ssn, String title, String firstName, String
lastName,
                ArrayList<Account> accounts) {
    this.ssn = ssn;
    this.title = title;
    this.firstName = firstName;
    this.lastName = lastName;
    this.accounts = accounts;
}
```

7.9.14 Using the delegate method generator

Use the delegate method generator feature to delegate methods from one class to another class for better encapsulation. We use a simple example to explain this feature. A car has an engine, and a driver wants to start the car. Figure 7-58 shows that the engine is not encapsulated, the PoorDriver has to use the Car class and the Engine class to start the car.

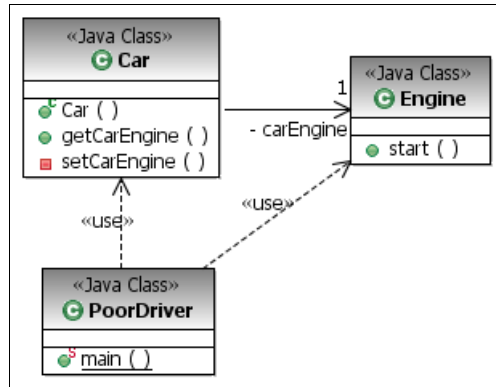


Figure 7-58 Simple car example class diagram (before method delegation)

Example 7-10 shows how the PoorDriver has to start the car.

Example 7-10 Car, Engine, and PoorDriver classes (compressed)

```
// Car class
package itso.rad80.example;
import itso.rad80.example.Engine;
public class Car {
    private Engine carEngine = null;
    public Car(Engine carEgine) {
        this.setCarEngine(carEngine);
    }
    public Engine getCarEngine() {
        return carEngine;
    }
    private void setCarEngine(Engine carEngine) {
        if (carEngine != null) {
            this.carEngine = carEngine;
        } else {
            this.carEngine = new Engine();
        }
    }
}
```

```
// Engine class
package itso.rad80.example;
public class Engine {
    public void start() {
        // code to start the engine
    }
}

// PoorDriver class
package itso.rad80.example;
import itso.rad80.example.Car;
public class PoorDriver {
    public static void main(String[] args) {
        Car myCar = new Car(null);
        /* How can I start my car?
         * Do I really have to touch the engine?
         * - Yes, there is no other way at the moment.
         */
        myCar.getCarEngine().start();
    }
}
```

To make the driver happy, we delegate the start method from the Engine class to the Car class. To delegate a method, follow these steps:

1. In the Car class, right-click the **carEngine** field and select **Source** → **Generate Delegate Methods**.
2. In the Generate Delegate Methods window (Figure 7-59), select only the **start** method and click **OK**.

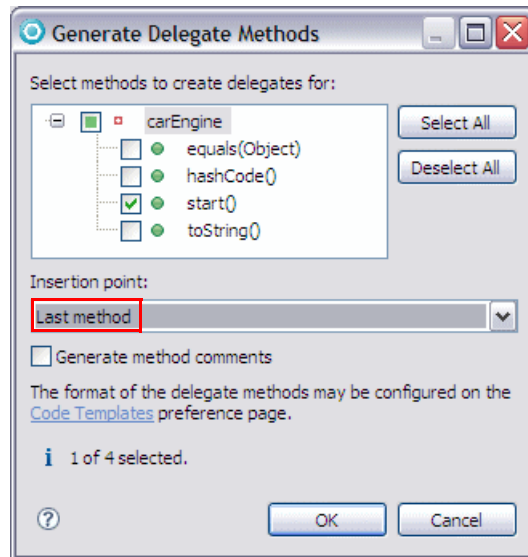


Figure 7-59 Generate Delegate Method window

The start method is added to the Car class, and code is added in the body of the method to delegate the method call to the Engine class through the carEngine attribute. Figure 7-60 on page 325 and Example 7-11 on page 325 show how the HappyDriver can start the car.

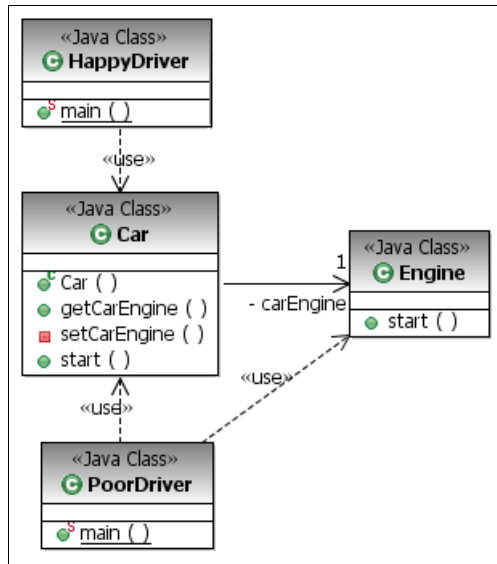


Figure 7-60 Simple car example class diagram (after method delegation)

Example 7-11 Car and HappyDriver class

```

// Car class
package itso.rad80.example;
import itso.rad80.example.Engine;
public class Car {
    private Engine carEngine = null;
    public Car(Engine carEngine) {
        this.setCarEngine(carEngine);
    }
    public Engine getCarEngine() {
        return carEngine;
    }
    private void setCarEngine(Engine carEngine) {
        if (carEngine != null) {
            this.carEngine = carEngine;
        } else {
            this.carEngine = new Engine();
        }
    }
    public void start() {
        carEngine.start();
    }
}

```

```

// HappyDriver class
package itso.rad80.example;
public class HappyDriver {
    public static void main(String[] args) {
        Car myAdvancedCar = new Car(null);
        // Start the car - I don't care about technical details
        myAdvancedCar.start();
    }
}

```

7.9.15 Refactoring

During the development of a Java application, you might need to perform tasks, such as renaming classes, moving classes between packages, and breaking out code into separate methods. These tasks are both time-consuming and error prone, because it is up to the programmer to find and update each and every reference throughout the project code. Rational Application Developer provides a list of refactor actions to automate this process.

The Java development tools of Rational Application Developer provide assistance for managing refactoring. In each refactor wizard, you can select an option:

- ▶ Refactor with preview: Click **Next** in the window to open a second window where you are notified of potential problems and are given a detailed preview of what the refactor action will do.
- ▶ Refactor without preview: Click **Finish** in the window and perform the refactor. If a stop problem is detected, refactor cancels and a list of problems is displayed.

Table 7-10 summarizes the common refactor actions.

Table 7-10 Refactor actions

Name	Function
Rename	<p>Starts the Rename Compilation Unit wizard. Renames the selected element and (if enabled) corrects all references to the elements (also in other files). It is available on methods, fields, local variables, method parameters, types, compilation units, packages, source folders, projects, and on a text selection resolving to one of these element types.</p> <p>Right-click the element and select Refactor → Rename, or select the element and press Alt+Shift+R, or select Refactor → Rename from the menu bar.</p>

Name	Function
Move	<p>Starts the Move wizard. Moves the selected elements and (if enabled) corrects all references to the elements (also in other files). Can be applied on one or more static methods, static fields, types, compilation units, packages, source folders, and projects, and on a text selection resolving to one of these element types.</p> <p>Right-click the method signature and select Refactor → Move, or select the method signature and press Alt+Shift+V, or select Refactor → Move from the menu bar.</p>
Change Method Signature	<p>Starts the Change Method Signature wizard. You can change the visibility of the method, change parameter names, parameter order, parameter types, add parameters, and change return types. The wizard updates all references to the changed method.</p> <p>Right-click the element and select Refactor → Change Method Signature, or select the element and press Alt+Shift+C, or select Refactor → Change Method Signature from the menu bar.</p>
Extract Interface	<p>Starts the Extract Interface wizard. You can create an interface from a set of methods and make the selected class implement the newly created interface.</p> <p>Right-click the class and select Refactor → Extract Interface, or select the element and select Refactor → Extract Interface from the menu bar.</p>
Push Down	<p>Starts the Push Down wizard. Moves a field or method to its subclasses. Can be applied to one or more methods from the same type or on a text selection resolving to a field or method.</p> <p>Right-click the type and select Refactor → Push Down, or select the element and select Refactor → Push Down from the menu bar.</p>
Pull Up	<p>Starts the Pull Up wizard. Moves a field or method to its superclass. Can be applied on one or more methods and fields from the same type or on a text selection resolving to a field or method.</p> <p>Right-click the type and select Refactor → Push Up, or select the element and select Refactor → Push Up from the menu bar.</p>
Extract Method	<p>Starts the Extract Method wizard. Creates a new method containing the statements or expressions currently selected, and replaces the selection with a reference to the new method.</p> <p>Right-click the statement or expression and select Refactor → Extract Method, or select it and press Alt+Shift+M, or select Refactor → Extract Method from the menu bar.</p>

Name	Function
Extract Local Variable	<p>Starts the Extract Local Variable wizard. Creates a new variable assigned to the expression currently selected and replaces the selection with a reference to the new variable.</p> <p>Right-click the expression and select Refactor → Extract Local Variable, or select it and press Alt+Shift+L, or select Refactor → Extract Local Variable from the menu bar.</p>
Extract Constant	<p>Starts the Extract Constant wizard. Creates a static final field from the selected expression and substitutes a field reference, and optionally replaces all other places where the same expression occurs.</p> <p>Right-click the expression and select Refactor → Extract Constant, or select Refactor → Extract Constant from the menu bar.</p>
Inline	<p>Starts the Inline Method wizard. Inlines local variables, non-abstract methods, or static final fields.</p> <p>Right-click the element and select Refactor → Inline, or select the element and press Alt+Shift+I, or select Refactor → Inline from the menu bar.</p>
Encapsulate Field	<p>Starts the Encapsulate Field wizard. Replaces all references to a field with getter and setter methods. Is applicable to a selected field or a text selection resolving to a field.</p> <p>Right-click the field and select Refactor → Encapsulate Field or select the element and select Refactor → Encapsulate Field from the menu bar.</p>

Refactor example (renaming a class)

In the following example of a refactor operation, we want to rename the class Transaction to BankTransaction in the RAD80Java project.

To rename the Transaction class to BankTransaction, follow these steps:

1. Right-click the **Transaction** class in the Package Explorer and select **Refactor** → **Rename**.
2. In the Rename Compilation Unit wizard (Figure 7-61 on page 329), enter the following data:
 - a. New name: BankTransaction.
 - b. Select **Update references** (default).
 - c. Clear other check boxes.
 - d. Click **Next** to see the preview and potential problems.

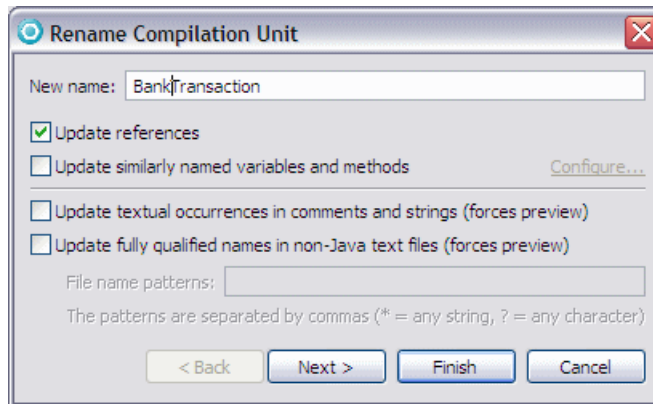


Figure 7-61 Refactor - Rename Compilation Unit wizard

3. Click **Finish** to process the rename task.

Important: If problems arise or warning messages are displayed, the Found Problems window opens. If the problems are severe, the **Continue** button is disabled and the refactor must be canceled until the problems have been corrected.

7.10 More information

For more information, use the Help feature provided by Rational Application Developer. Select **Help** → **Help Contents** → **Developing Java applications**.

Also, see the Watch and Learn tutorial called *Create a Hello World Java Application*, which you can access by selecting **Help** → **Welcome** → **Tutorials** → **Create a Hello World Java application**.

The following web resources provide further information about Eclipse and Java technology:

- ▶ Oracle Java SE page contains links to specifications, API Javadoc, and articles about Java SE:
<http://www.oracle.com/technetwork/java/javase/index.html>
- ▶ IBM developerWorks Java Technology contains Java news, downloads, CDs, and learning resources:
<http://www.ibm.com/developerworks/java/>

- ▶ Eclipse Open Source Community is the official home page of the Eclipse open source community:
<http://www.eclipse.org/>
- ▶ What's new in Eclipse 3.6:
http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.jdt.doc.user/whatsNew/jdt_whatsnew.html
- ▶ Eclipse Tips and Tricks:
http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.jdt.doc.user/tips/jdt_tips.html



Developing XML applications

In this chapter, we provide an overview of the current XML technologies. We cover the XML capabilities that are provided by Rational Application Developer:

- ▶ XML overview and associated technologies
- ▶ Rational Application Developer XML tools
- ▶ Creating an XML schema
- ▶ Generating HTML documentation from an XML schema file
- ▶ Generating an XML file from an XML schema file
- ▶ Editing an XML file
- ▶ Working with XSL transformation files
- ▶ Transforming an XML file into an HTML file
- ▶ XML mapping
- ▶ Service Data Objects and XML
- ▶ JAXB and XML

The sample code for this chapter is in the 7835code\xml folder. Refer to Appendix C, “Additional material” on page 1877.

8.1 XML overview and associated technologies

Extensible Markup Language (XML) is a subset of Standard Generalized Markup Language (SGML). Both XML and SGML are meta languages, because they allow the definition of a chosen set of elements and attributes to meet the requirements of a specific application area. Markup languages are used to annotate information so that it is easier to manipulate and understand. Markup is also used to define how information will be presented for display. One example is HTML, which is used to mark up document information so that it can be displayed by a web browser. The elements within an XML document are organized hierarchically with a single root element at the top of the hierarchy.

XML is a key part of the software infrastructure. It provides a simple and flexible means of defining, creating, and storing data. XML is used for a variety of purposes, such as systems configuration, messaging, and data storage.

The rules that define what can be present in any specific XML document are held in either a document type definition (DTD) or an XML Schema Definition (XSD). If a DTD or XSD is available, it is possible to check that an XML document is valid. If an XML document is to be usable, it must first be well-formed, which means that it must adhere to all the XML syntax rules as defined in the XML specification document. Only a well-formed XML document can be checked by using a DTD or XSD to see if it is valid. A valid XML document is guaranteed to contain only what is allowed.

For detailed information about XML, see the following web address:

<http://www.w3.org/XML/>

8.1.1 XML processors

XML is tag-based, but without predefined XML tags. Tags are the markup inserted in an XML document to define the elements from which it is composed.

XML documents follow strict syntax rules. To create, read, and update XML documents, you require an XML processor or parser. At the heart of an XML application is an XML processor that parses an XML document, allowing the document elements to be retrieved and used as required. Parsers are also responsible for checking the syntax and structure of XML documents.

In previous releases, you had to use one of two standard mechanisms for parsing your XML data:

- ▶ Documents Object Model (DOM)
- ▶ Simple application programming interface (API) for XML (SAX)

Streaming API for XML (StAX) is another, more efficient, alternative to manipulate XML data.

For detailed information about SAX, DOM, and StAX, see the following web addresses:

- ▶ SAX
<http://www.saxproject.org/>
- ▶ DOM
<http://www.w3.org/DOM/>
- ▶ StAX
<http://www.jcp.org/en/jsr/detail?id=173>

8.1.2 DTDs and XML schemas

DTDs and XML schemas are both used to describe XML document structure; however, in recent years, the acceptance of XML schemas has gained momentum. Both DTDs and XML schemas define the building blocks for XML documents: the elements, attributes, and entities.

XML schemas are more powerful than DTDs. XML schemas have the following advantages over DTDs:

- ▶ Data type definition: XML schemas can define data types for elements and attributes, and their default and fixed values. The data types can be string, decimal, integer, boolean, date, time, or duration.
- ▶ Restrictions: XML schemas can apply restrictions to elements, by stating minimum and maximum values. For example, an age element might be restricted to hold values from 1 to 90, or a string value can be restricted to only hold one value from a specific list of values in a defined allowed list, such as Fixed, Savings, or Loan. Restrictions can also be applied to characters and patterns of characters. For example, characters can be restricted to only those characters from “a” to “z” and the length of the character string can be restricted to only three letters. Another restriction can be that the string can have a range of lengths, for example, a password must be between 4 and 8 characters in length.
- ▶ Complex element: XML schemas can define complex element types. Complex types can contain simple types or other complex types. Restrictions can be applied to the sequence and frequency of the occurrence of each type. Complex types can be used in the definition of other complex types.
- ▶ XML schema documents: Unlike DTDs, you use actual XML documents. This implies that XML schema documents can be automatically checked for

validity, and authoring XML schema documents is simpler for those individuals already familiar with XML. Also, XML parsers do not have to be enhanced to provide support for DTDs. Transformation of XML schema documents can be carried out using Extensible Stylesheet Language Transformation (XSLT) documents, and they can be manipulated using the XML DOM.

For detailed information about DTD and XML schema, see the following pages at the W3C website:

- ▶ Extensible Markup Language 1.0 (Fourth Edition)
<http://www.w3.org/TR/2006/REC-xml-20060816/>
- ▶ XML Schema 1.1 status
<http://www.w3.org/XML/Schema>

8.1.3 XSL

Extensible Stylesheet Language (XSL) is a set of recommendations defined by the W3C. XSL is composed of the following W3C recommendations:

XSL Transformation An XML markup language that is used for the transformation of XML documents.

XML Path Language (XPath)

A language that is used to access or refer to parts of an XML document.

XSL-FO

An XML markup language that is used to format information for the purpose of presentation. It is similar to the way that HTML marks up information for presentation by a web browser.

XML document transformations defined using XSLT are XML documents. The elements present in the XSLT document are defined in the XSLT namespace. We discuss namespaces later in this chapter.

An XSLT transformation processor is required when transforming a document using XSLT. The processor takes as input a source XML document and an XSLT transformation document. The transformations defined in an XSLT document are used to transform the source file into the output file. XSLT uses pattern matching and templates to define the required transformation. When a pattern defined in the XSLT transformation document is found in the source document, the associated template, also defined in the XSLT transformation document, is placed in the output file.

The produced output file is typically another XML document or an HTML document.

For detailed information about XSLT, see the following web page:

<http://www.w3.org/TR/xslt.html>

For detailed information about XSLT 2.0, see the following web page:

<http://www.w3.org/TR/xslt20/>

8.1.4 XML namespaces

Namespaces are used when there is a requirement for elements and attributes of the same name to take on a separate meaning depending on the context in which they are used. For example, an element called TITLE has a separate meaning, depending on whether it is present within a PERSON element or within a BOOK element. In the case of the PERSON element, it is placed in front of a person's name, such as Mr. or Dr. In the case of a BOOK element, it is the title of the book, such as Programming Guide.

Both elements, PERSON and BOOK, must be defined in the same document, for example, in a library entry that associates a book with its author, a mechanism is required to distinguish between the two so that the correct semantics apply when the TITLE element is used in the document.

Namespaces provide this mechanism by allowing a namespace and an associated prefix to be defined. Elements that use a specific namespace prefix are said to be present in that namespace and have the meaning defined for them in the namespace. The prefix is separated from the element name by a colon character (:). In our example, TITLE is defined in two separate namespaces. One namespace pertains to elements relevant to holding information about books, and the other namespace pertains to elements storing information about people. Example start tags for the elements might be <book:TITLE> and <people:TITLE>.

For detailed information about XML namespaces, see the following web page:

<http://www.w3.org/TR/REC-xml-names/>

8.1.5 XPath

XPath is used to address parts of an XML document. The XML document is considered to be a tree of nodes and an XPath expression selects a specific node or set of nodes within the tree. You can achieve this outcome by defining the path to the node or nodes. An XPath expression, in addition, can include

instructions to manipulate values held at a specific node or set of nodes. XPath is used with XSLT, discussed previously, and with other XML technologies.

For detailed information about XPath, see the following web page:

<http://www.w3.org/TR/xpath>

For detailed information about XPath 2.0, see the following web page:

<http://www.w3.org/TR/xpath20/>

8.1.6 XML catalog

XML catalogs offer a way to manage local copies of public DTDs, schemas, or indeed any XML resource that exists outside of the referring XML instance document. An XML catalog entry contains two parts:

- ▶ A key, which represents a DTD or XML schema
- ▶ A location, which is similar to a Uniform Resource Identifier (URI), which contains information about a DTD or XML schema's location

You can place the key in an XML file. When the XML processor encounters it, it will use the XML catalog entry to find the location of the DTD or XML schema associated with the key.

8.2 Rational Application Developer XML tools

Rational Application Developer provides a comprehensive visual XML development environment. The tool set includes components for building DTDs, XML schemas, XML documents, and XSL files.

Rational Application Developer includes the following XML development tools:

- ▶ DTD editor
Used for creating, viewing, and validating Document Type Definitions (DTDs). Using the DTD editor, you can create DTDs and generate XML schema files.
- ▶ XML editor
Used for creating, viewing, and validating XML files. You can use it to create new, empty XML files, or generate them from existing DTDs or existing XML schemas. You can also use it to edit XML files, associate them with DTDs or schemas, and validate them.

- ▶ XML Schema Editor

A tool for creating, viewing, and validating XML schemas. You can use the XML Schema Editor to perform tasks, such as creating XML schema components, importing and viewing XML schemas, generating relational table definitions from XML schemas, and generating Java beans for creating XML instances of an XML schema.
- ▶ XSLT editor

Used to create new XSLT files or to edit or validate existing ones. You can use content assist and various wizards to help you create or edit the XSLT file. After you finish editing your file, you can also validate it. Additionally, you can associate an XML instance file with the XSL source file that you are editing and use it to provide guided editing when defining constructions, such as an XPath expression. In Rational Application Developer, the XSLT editor is enhanced to support XSLT 2.0. The XSLT code templates configured in preferences are available in the editor when using content assist and in the New XSLT wizard.
- ▶ XPath Expression wizard

Used to create XPath expressions. XPath expressions can be used to search through XML documents, and to extract information from the nodes, such as an element or attribute. In Rational Application Developer, it is enhanced to support XPath 2.0.
- ▶ XML Mapping editor

Used to map XML-based documents graphically by connecting elements of a source document to elements of a target document. You can extend built-in transformation functions using custom XPath expressions and XSLT templates. The mapping tool automates XSL code generation and produces a deployable transformation document based on the mapping information that you provide.
- ▶ XSL compiler

Used for compiling and integrating XSL 1.0 and 2.0 stylesheet documents into Java projects.
- ▶ XML catalog tools

XML catalog can be used for registering grammars, XSDs, and DTDs in the workspace. Rational Application Developer adds support for the Oasis XML Catalog 1.1 specification. The XML catalog tools can be found in **Preferences** → **XML** → **XML Catalog**.

You can find preferences for all the XML tools, including preferences for source editors, in **Preferences** → **XML**.

8.2.1 Creating an XML schema

You can create an XML schema for use in storing bank account information as explained in this section. The XML document structure defined in the schema is a collection of accounts. Each account has an account ID, account type, account balance, interest rate, and related customer information. The account ID is allowed to be from 6 to 8 digits long and each digit must be in the range 0 to 9. The value of the interest rate is allowed to be between 0 and 100. The telephone number is in the format (xxx) xxx-xxxx.

To create a project to hold the schema document and create an empty schema document, follow these steps:

1. Open the **XML** perspective and create a project named RAD8XMLBank:
 - a. Select **File** → **New** → **Project** → **General** → **Project**. Then click **Next**.
 - b. In the Project name field, type RAD8XMLBank. Click **Finish** and the project is displayed in the Enterprise Explorer.
 - c. Right-click the **RAD8XMLBank** project and select **New** → **Folder**.
 - d. Enter `xml` as the folder name and click **Finish**.
2. Create an XML schema:
 - a. Right-click the `xml` folder and select **New** → **Other** → **XML** → **XML schema**. Click **Next**. In the File name field, type `Accounts.xsd`. The parent folder is set to `RAD8XMLBank/xml`. Click **Finish**.

Shortcuts: Another way to start the XML Schema wizard is to use the shortcut in the toolbar. Other available shortcuts include the New XML, New DTD, New XSL, and New XML Mapping shortcuts.

- b. Make sure that the Properties view is visible. If you cannot see the Properties view, select **Window** → **Show view** → **Properties**.

Working with the Design view

With the XML editor, you can edit the XML source for the schema directly or you can use the Design view. The Design view presents the XML document in a graphical view.

To create the schema in the Design view, follow these steps:

1. Select the **Design** view. Notice *View: Simplified* in the upper-right corner. The following views are available:

Simplified The Simplified view hides many of the complicated XML schema capabilities so that you can only create XML schemas that conform to leading practice authoring patterns. XML schema elements, such as `xsd:choice`, `xsd:sequence`, `xsd:group`, and element references, are not displayed in the Simplified view. Actions in the editor that enable the creation of these elements are not available.

Detailed The Detailed view exposes the full set of XML schema capabilities so that you can create XML data structures using any authoring pattern.

We use the Detailed view for this exercise.

- From the View drop-down list (Figure 8-1), select **Detailed**. Close the pop-up message about switching view modes.

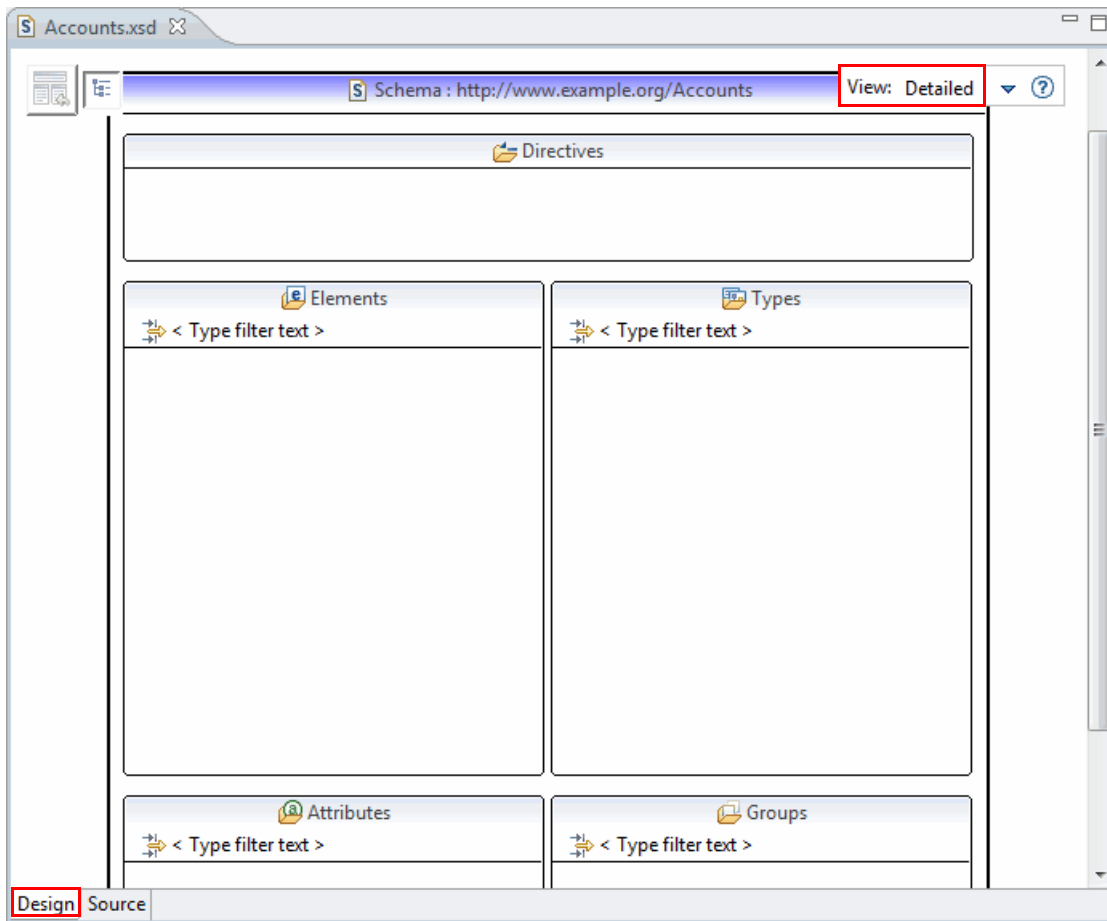



Figure 8-1 XML Schema Editor: Design view, Detailed view

Schema index view: In the Detailed view, the top-level view is called the *schema index view*. You can navigate to this view at any time while editing the schema by clicking the **Show schema index view**  icon in the upper-left corner of the Design view. In Figure 8-1, this icon is currently unavailable, because you are already viewing the schema index.

3. Change the namespace prefix and target namespace:
 - a. In the Properties view, select the **General** tab. You can see the current namespace prefix and target namespace. The values are tns and `http://www.example.org/Accounts`.
 - b. Change the Prefix to itso.
 - c. Change the Target namespace to `http://itso.rad8.xml.com`.

The `Accounts.xsd` file must contain a complex type element where you will define the account information, including account ID, account type, balance, interest, and customer information.

4. In the Design view, right-click the **Types** category and select **Add ComplexType**. Overtyping the name provided with the value `Account`.

Tip: Alternatively, you can change the type name in the Name field on the **General** tab in the Properties view. The Properties view provides many options for modifying the properties of an XML schema.

5. Right-click the **Account** complex type and select **Add Sequence**. The Design view switches from showing the schema index to only showing the `Account` complex type:
 - a. Clicking the **Show schema index view** icon returns to the schema index.
 - b. Double-clicking the **Account** complex type shows only this type in the detailed view.

Add Sequence option: In a Simplified view, you are unable to see the Add Sequence option in the context menu, because the Simplified view hides many of the more complicated XML elements.

6. Right-click the **Account** complex type and select **Add Element**.
7. Change the element name to `accountID` (Figure 8-2 on page 342).

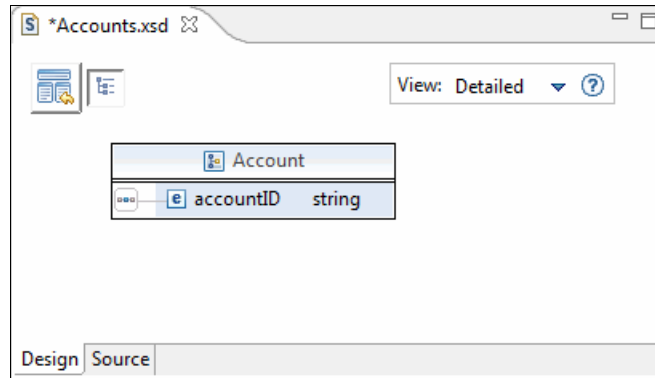


Figure 8-2 Account complex type

In our bank, the account ID is 6 - 8 characters long and takes numerical values in the range 0 - 9:

- a. Click the element **accountID**, and in the Properties view, select the **Constraints** tab.
- b. In the Specific constraint values section, select **Patterns** and click **Add**.

8. In the Regular Expression Wizard (Figure 8-3), complete these steps:
 - a. In the current regular expression field, type `[0-9]{6,8}` and click **Next**.

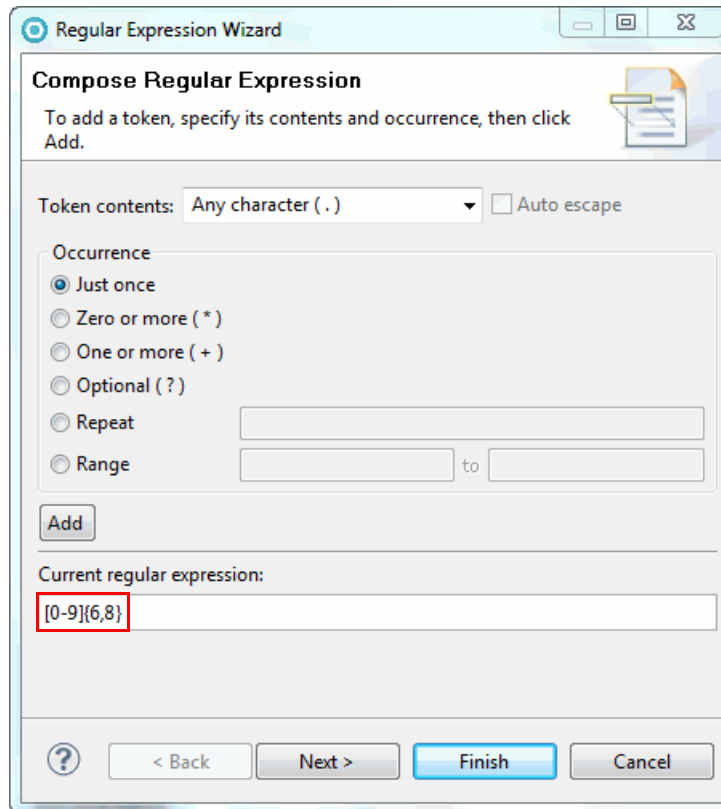


Figure 8-3 Regular Expression Wizard

- b. In the Sample text area, type 123456. The warning at the top of the dialog box is no longer displayed.
 - c. Click **Finish**.

The type for accountID changes from string to AccountIDType, because you constrained the string to create a new type.

Figure 8-4 shows the Properties view with the constraint value.

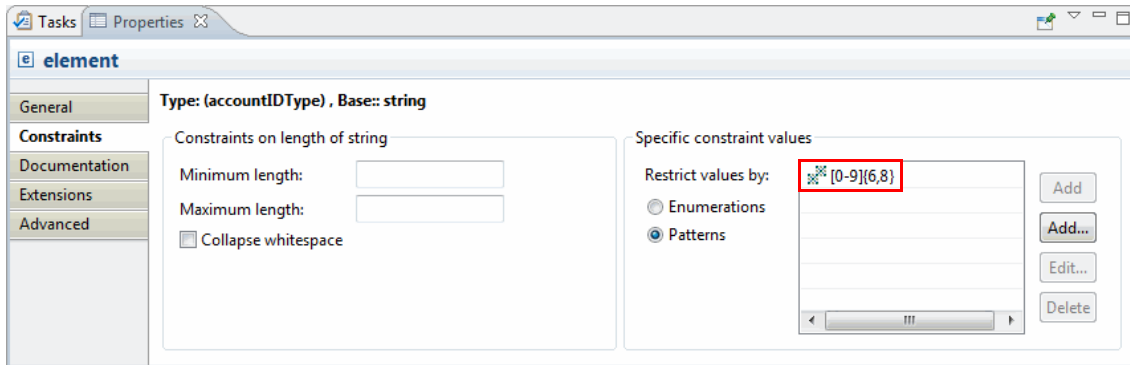


Figure 8-4 Properties view with constraint

9. In the Design view, right-click the sequence icon (☰) and select **Add Element**. Change its name to `accountType`. A bank account has three account types: Savings, Loan, and Fixed:
 - a. In the Properties view, select the **Constraints** tab. Under Specific constraint values, select **Enumerations**.
 - b. Click **Add** and enter Savings.
 - c. Click **Add** and enter Loan.
 - d. Click **Add** and enter Fixed.
10. Add the balance element:
 - a. In the Design view, right-click the sequence icon (☰) and select **Add Element**. Change its name to `balance`.
 - b. In the Properties view, on the **General** tab, select the drop-down menu for **Type**.
 - c. Select **Browse** and then select **decimal**.
11. Add the interest element:
 - a. Add an element named `interest` and set the type to **decimal** as before.
 - b. In the Properties view, select the **Constraints** tab. Set the Minimum value to **0** and Maximum value to **100**.
12. Add the customerInfo element:
 - a. Add an element named `customerInfo`.
 - b. In the Properties view, on the **General** tab, select the drop-down menu for **Type** and select **New**.

- c. In the New Type window, select **Complex Type** and the **Create as local anonymous type** check box. Click **OK** (Figure 8-5).

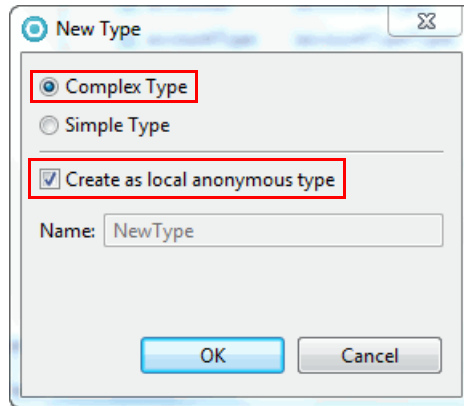


Figure 8-5 New Type window

13. Click the plus sign to the right of `customerInfo`, and a `(CustomerInfoType)` box is displayed to the right of the `Account` box.
14. Right-click (**CustomerInfoType**) and select **Add Element**. Change the name to `firstName`.
15. Add another element named `lastName`.
16. Add another element named `phoneNumber`, which holds values with a format such as (408) 555-7890:
 - a. In the Properties view, on the Constraints tab, select **Patterns**.
 - b. Click **Add**. In the Current regular expression area, type `\([0-9]{3}\) [0-9]{3}-[0-9]{4}`. Note the space between the area code and the local telephone number. Then click **Next**.
 - c. For Sample text, type (408) 555-7890 and click **Finish**.Figure 8-6 on page 346 shows the Design view.

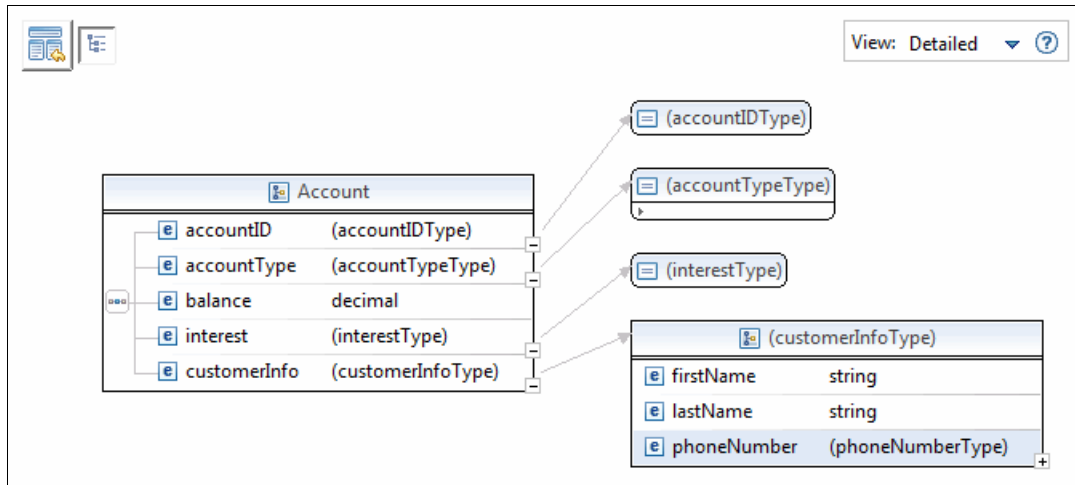



Figure 8-6 Account type complete

Tip: You can export the Design view as an image for use elsewhere by selecting **XSD** → **Export Diagram as Image**.

If this XML schema is intended to define the structure of an XML document that can actually be created, the XML schema must have a global element. All we have at present is a complex type definition. We have to define at least one element of this type.

17. Add a global element named accounts:
 - a. Click the **show schema index view** icon () in the upper-left corner.
 - b. In the Design view of the schema, right-click the **Elements** category and select **Add Element**.
 - c. Change the name to accounts.
 - d. In the Properties view, on the **General** tab, select the drop-down menu for **Type** and select **New**.
 - e. In the New Type window, select **Complex Type** and the **Create as local anonymous type** check box. Then click **OK**.
 - f. Double-click **accounts** to switch to the detailed view for the accounts element. Right-click **accountsType** and select **Add Element**.
 - g. Change the name to account.
 - h. Right-click **account** and select **Set Type** → **Browse** → **Account**.
 - i. In the Design view, make sure **account** is selected.

- j. In the Properties view, on the General tab, set Minimum Occurrence to **1** and Maximum Occurrence to **unbounded**.

Working with the Source view

Tidy up the XML source code that you created:

1. Select the **Source** tab.
2. Right-click anywhere in the source code and select **Source** → **Cleanup Document**. The cleanup dialog contains settings to update a document so that it is well-formed and formatted as **Format**, which formats either the entire document or selected elements. Select your desired options and click **OK**.
3. Save and close the file.

Example 8-1 shows the generated `Accounts.xsd` file.

Example 8-1 Accounts.xsd file

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://itso.rad8.xml.com"
  elementFormDefault="qualified"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:itso="http://itso.rad8.xml.com">
  <complexType name="Account">
    <sequence>
      <element name="accountID">
        <simpleType>
          <restriction base="string">
            <pattern value="[0-9]{6,8}"></pattern>
          </restriction>
        </simpleType>
      </element>
      <element name="accountType">
        <simpleType>
          <restriction base="string">
            <enumeration value="Savings"></enumeration>
            <enumeration value="Loan"></enumeration>
            <enumeration value="Fixed"></enumeration>
          </restriction>
        </simpleType>
      </element>
      <element name="balance" type="decimal"></element>
      <element name="interest">
        <simpleType>
          <restriction base="decimal">
            <minExclusive value="0"></minExclusive>
          </restriction>
        </simpleType>
      </element>
    </sequence>
  </complexType>
</schema>
```

```

        <maxExclusive value="100"></maxExclusive>
    </restriction>
</simpleType>
</element>
<element name="customerInfo">
    <complexType>
        <sequence>
            <element name="firstName" type="string"></element>
            <element name="lastName" type="string"></element>
            <element name="phoneNumber">
                <simpleType>
                    <restriction base="string">
                        <pattern
                            value="\([0-9]{3}\) [0-9]{3}-[0-9]{4}">
                        </pattern>
                    </restriction>
                </simpleType>
            </element>
        </sequence>
    </complexType></element>
</sequence>
</complexType>
<element name="accounts">
    <complexType>
        <sequence>
            <element name="account" type="itso:Account" minOccurs="1"
maxOccurs="unbounded"></element>
        </sequence>
    </complexType></element>
</schema>

```

Validating an XML schema

Use the XML validator:

1. Select **Window** → **Preferences**.
2. In the Preferences window (Figure 8-7 on page 349), in the left pane, select **Validation**. In the right pane, for XML Schema Validator, click the ellipsis (...) button.
3. In the Validation Filters for XML Schema Validator window (inset in Figure 8-7 on page 349), for Implementation, select **IBM XML Schema Validator** and click **OK**.

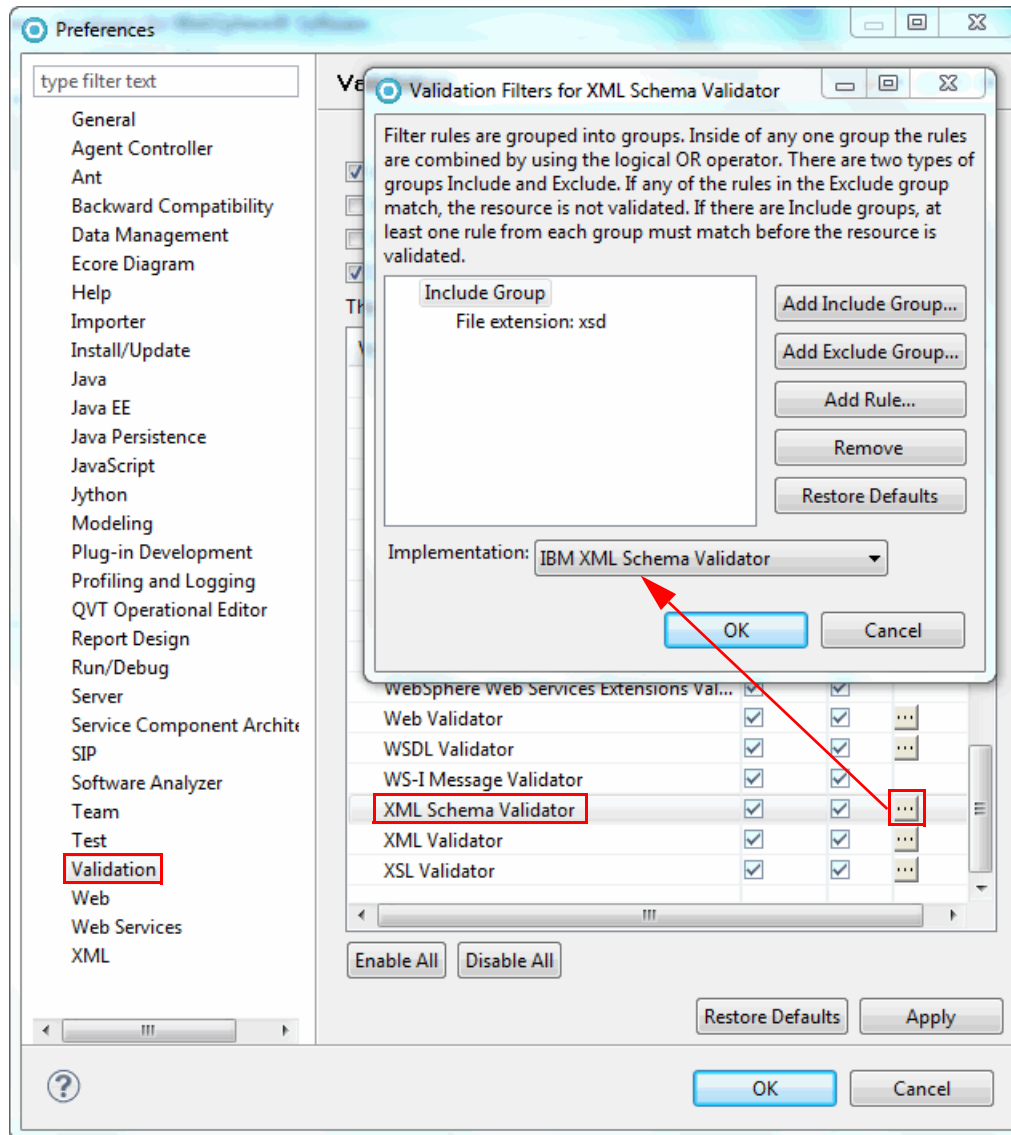


Figure 8-7 XML Schema Validator preferences

In certain cases, when you build a large Java Enterprise Edition (Java EE) project, the XSD validation process can take time. You can disable the validator at either the project level or the global level:

- ▶ Follow these steps to disable the validator at the project level:
 - a. Right-click the project and select **Properties** → **Validation**.

- b. Select **Enable project-specific settings** and clear the **XML Schema Validator** check box in the Build column.
- ▶ Perform these steps to disable the validator at the global level:
 - a. Select **Window** → **Preferences**.
 - b. In the Preferences window (Figure 8-7 on page 349), clear the **XML Schema Validator** check box in the Build column.

Running schema validation manually

The validation builder is not added to the simple projects, such as our project. To validate your XML schema, in the Enterprise Explorer, right-click **Account.xsd** and select **Validate**.

- ▶ If validation is successful, a Validation Results window opens. No errors are in the Problems view.
- ▶ If validation is unsuccessful, validation errors are displayed in the Problems view. You might have to open the view using **Window** → **Show View**. In addition, a red X is shown next to the file and in the Source view.

You will not receive any XML schema validation errors for `Accounts.xsd`, because we created it in the Design view and did not enter the XML manually.

If you want to make the document invalid so that you can see an error report, change the type of one of the elements from `decimal` to `deximal`, and execute validation again. After doing this and reading the error message, correct the error and run validation again to remove the error message.

Tip: When creating a simple project, you can add the validation builder to it to get automatic validation on resource changes, therefore removing the need for manual validation.

8.2.2 Generating HTML documentation from an XML schema file

HTML documentation generated from an XML schema file contains information about the schema, such as its name, location, and namespace, as well as details about the elements and types in the schema. This information can be useful, because it provides a summary of the content of a schema in a form that is easily readable.

To generate HTML documentation based on an XML schema file, follow these steps:

1. In the Project Explorer, right-click **Accounts.xsd** and select **Generate** → **HTML**.

2. In the XSD Documentation Generation Options window, select **Generate XSD Documentation with frames**. Selecting this option generates schema documentation that uses HTML frames. If frames are not required, select **Generate XSD Documentation without frames**. Click **Next**.
3. For the folder name, type docs and click **Finish**.

The HTML files are created in the specified location, and the generated `index.html` file opens inside Rational Application Developer. Explore the generated documentation by selecting the **Account** type. You can see the diagram and expand the underlying source code.

8.2.3 Generating an XML file from an XML schema file

Rational Application Developer can generate an XML document from an XML schema file. With this capability, a developer can gain familiarity with XML documents that are valid against a specific schema. In practice, a schema is used to validate XML documents created elsewhere. Also, an XML document generated from a schema is often called an *XML instance document* or an *instance document*. To generate an XML file from our XML schema file, follow these steps:

1. In the Enterprise Explorer, right-click **Accounts.xsd** and select **Generate** → **XML File**.
2. Accept the default name **Accounts.xml**. Click **Next**.
3. On the Select Root Element page, accept the default values for **Create first choice of required choice** and **Fill elements and attributes with data**. The XML schema that you created earlier does not have optional attributes or elements. Click **Finish**.
4. When the XML file opens in the editor, right-click the generated XML file **Account.xml** and select **Validate**. Notice the validation errors against AccountID, interest, and phoneNumber. The default values inserted by Rational Application Developer are not valid against the schema.
 - a. The XML schema specifies that the account ID is 6 - 8 digits long. Change the account ID to 123456.
 - b. Change the interest value to a valid value (5.5).
 - c. Change the telephone number to a valid (xxx) xxx-xxxx format, for example, (123) 555-7890.
 - d. Optional: Change the *firstName* and *lastName* to your name.
5. Right-click **Account.xml** and select **Validate**. You have no validation errors.

8.2.4 Editing an XML file

The XML editor enables you to directly edit XML files. There are several views that you can use to edit an XML file (Figure 8-8):

- ▶ **Source tab:** You can manually insert, edit, and delete elements and attributes in the Source view of the XML editor. To facilitate this effort, you can use content assist (Ctrl+Spacebar).
- ▶ **Design tab:** You can insert, delete, and edit elements, attributes, comments, and processing instructions in this view. We used this view previously when we created our XML schema. When editing an XML file that is not a schema, the Design view presents the document as a tree of elements with attributes rather than the format that we saw previously.
- ▶ **Outline view:** You can insert and delete element attributes, comments, and processing instructions in this view.

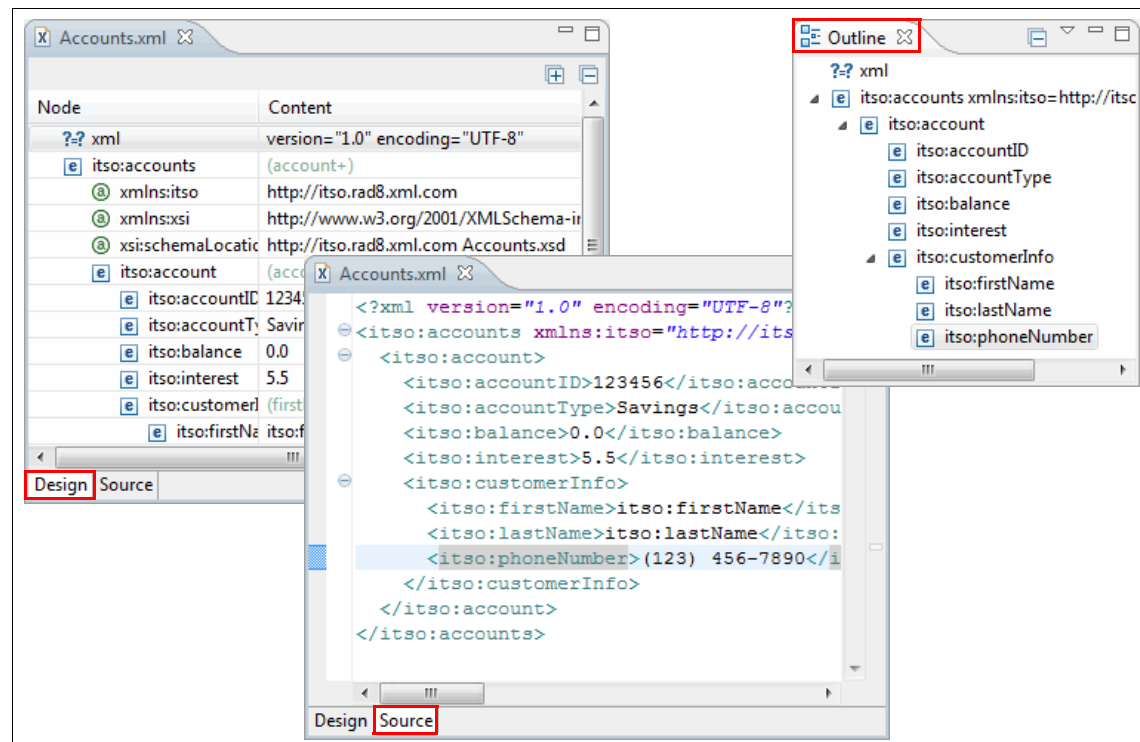


Figure 8-8 Design tab, Source tab, and Outline view

Editing on the Source tab

Working with the XML file that we generated in the previous section, follow these steps:

1. On the **Source** tab, place your cursor after the closing tag `</itso:account>`.
2. Press Ctrl+Spacebar to activate code assist. In the pop-up list (based on the context), double-click `<itso:account>`.

Content assist: Content assist works, because the document is associated with the schema that we created, and the editor can use the schema to determine what is valid content for specific locations in the document. The start tag shows how this association is specified:

```
<itso:accounts xmlns:itso="http://itso.rad8.xml.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://itso.rad8.xml.com Accounts.xsd ">
```

3. While the cursor is still between `<itso:account>` tags, press Ctrl+Spacebar and double-click `<itso:accountID>`.
4. Type a number for `accountID` between the start and end tags.
5. Repeat the same procedure to use the code assist feature to input the rest of the information, such as `accountType`, `balance`, and so forth. *All child tags are required.*
6. Define values for `accountID`, `accountType`, and `interest`.
7. After you finish typing, right-click in the XML source area and select **Source** → **Format**.

Editing on the Design tab

To edit on the Design tab, follow these steps:

1. In the Design tab, right-click **itso:accounts** and select **Add Child** → **account**.
2. Expand the **Account** element that you just created. All the child elements are created with default values. You can now edit the values of the child elements. In the Source tab, you have to add each child tag individually.

Editing in the Outline view

To edit in the Outline view, follow these steps:

1. In the Outline view, right-click **itso:accounts**. A context menu similar to the context menu in the Design view opens.
2. Save and close the file.

8.2.5 Working with XSL transformation files

An *XSL transformation file* is a style sheet that can be used to transform XML documents into other document types and to format the output. In this section, you create a simple XSL style sheet to format the XML file data into a table in an HTML file.

Creating a new XSL transformation file

To create a sample XSL transformation file, follow these steps:

1. Right-click the **xml** folder and select **New** → **Other** → **XML** → **XSL**. Click **Next**.
2. In the File name field, type `Accounts.xml` and click **Next**.
3. In the Select XML file window, expand **RAD8XMLBank/xml** and select the **Accounts.xml** file to associate the `Accounts.xml` file with the `Accounts.xml` file. Click **Finish**.

Example 8-2 shows the generated XSL file.

Example 8-2 Generated Accounts.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
</xsl:stylesheet>
```

Adding code to the XSL transformation file

The XSL editor provides help with creating content in the style sheet through the Snippets view. We use the Snippets view to add an HTML header and a table.

Snippets view: If you cannot see the Snippets view, select **Window** → **Show View** → **Other** → **General** → **Snippets**.

To add code snippets to the XSL file, follow these steps:

1. In the XSL editor, position the cursor between the `<xsl:stylesheet>` tags, immediately after `version="1.0">`.
2. Select the **Snippets** view and select the **XSL** drawer (Figure 8-9).

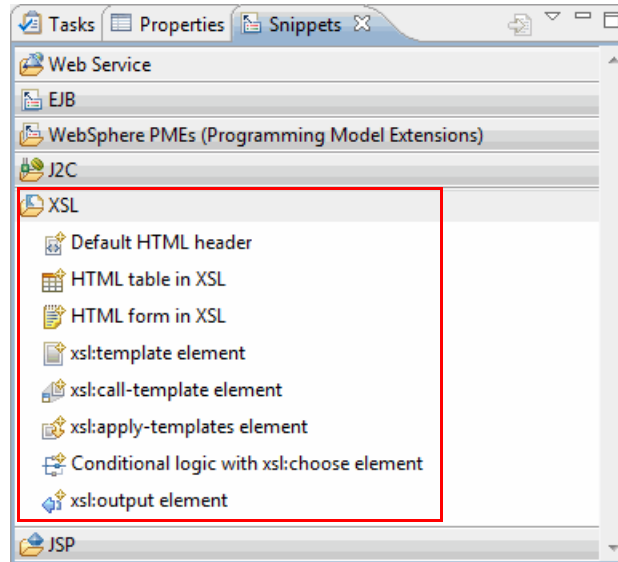


Figure 8-9 Snippets view: XSL drawer

3. Double-click **Default HTML header** to add default HTML header information to the XSL file.
4. Position the cursor after the end tag `</xsl:template>`.
5. In the XSL drawer, double-click **HTML table in XSL**. The XSL Table Wizard opens. Perform these steps:
 - a. Select **Wrap table in a template**.
 - b. Select **Include header** to indicate that you want to include a header row in the table.
 - c. Select one of the nodes on the left (for example, `its:account`) to see the generated code at the bottom (Figure 8-10 on page 356).
 - d. Click **Next**.

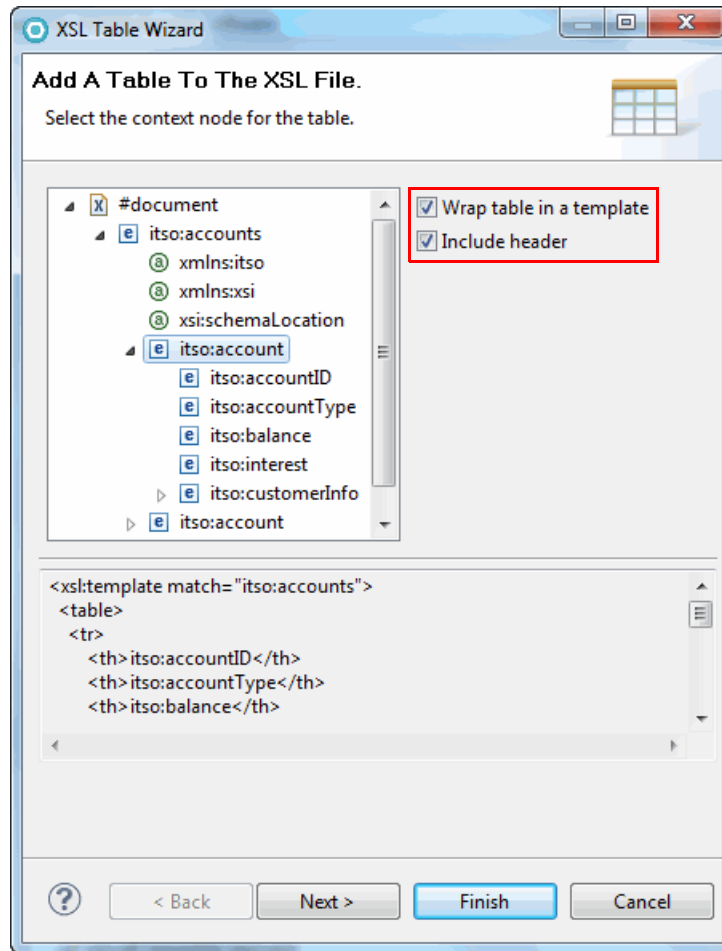


Figure 8-10 Adding a table to the XSL file in the XSL Table Wizard

6. In the final window of the wizard, specify the properties for the table. For the Border field, type 5, and for the Cell spacing field, type 10. Select a background color (**light cyan**) and a row color (**white**). Click **Finish**, and the Accounts.xsl file is completed.
7. On the Source tab, complete the following actions:
 - a. Right-click and select **Source** → **Format**.
 - b. Change the <title> to **Accounts**.
 - c. Remove the itso: prefix from the values in the table header fields, for example, <th>itso:AccountID</th>.
 - d. Save and close the file.

8. Right-click **Accounts.xml** and select **Validate**. You do not receive any validation errors or warnings.

Example 8-3 shows the generated `Accounts.xml` file.

Example 8-3 Accounts.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0" xmlns:xalan="http://xml.apache.org/xslt"
  xmlns:itso="http://itso.rad8.xml.com">
  <xsl:output method="html" encoding="UTF-8" />
  <xsl:template match="/">
    <html>
      <head>
        <title>Accounts</title>
      </head>
      <body>
        <xsl:apply-templates />
      </body>
    </html>
  </xsl:template>
  <xsl:template match="itso:accounts">
    <table bgcolor="#80ffff" border="5" cellspacing="10">
      <tr bgcolor="#ffffff">
        <th>accountID</th>
        <th>accountType</th>
        <th>balance</th>
        <th>interest</th>
        <th>customerInfo</th>
      </tr>
      <xsl:for-each select="/itso:accounts/itso:account">
        <tr bgcolor="#ffffff">
          <td>
            <xsl:value-of select="itso:accountID" />
          </td>
          <td>
            <xsl:value-of select="itso:accountType" />
          </td>
          <td>
            <xsl:value-of select="itso:balance" />
          </td>
          <td>
            <xsl:value-of select="itso:interest" />
          </td>
          <td>
```

```
        <xsl:value-of select="itso:customerInfo" />
    </td>
</tr>
</xsl:for-each>
</table>
</xsl:template>
</xsl:stylesheet>
```

Tip: You can use the XSLT debugger to detect and diagnose errors in XSL transformations. The debugger supports the debugging of both XSLT 1.0 and 2.0 stylesheets using separate XSLT processors. We discuss the XSLT debugger in Chapter 28, “Debugging local and remote applications” on page 1461.

8.2.6 Transforming an XML file into an HTML file

You can now use the XSL stylesheet file to generate an HTML file from the sample XML file:

1. In the Enterprise Explorer, hold down the Ctrl key and select both the **Accounts.xml** and **Accounts.xsl** files. Right-click and select **Run As** → **XSL Transformation**.

The resulting file name is `_Accounts_transform.html`. The file automatically opens in the Page Designer.

2. Select the **Split** tab (Figure 8-11 on page 359).

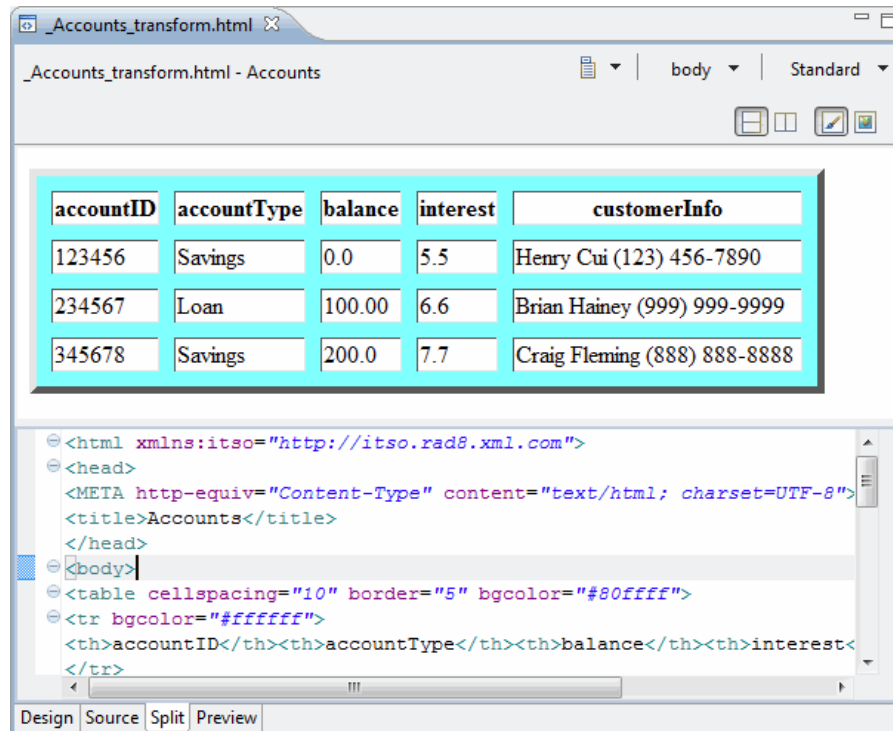


Figure 8-11 XSL stylesheet transformation result

Notice the transformation messages in the Console view:

Processing:

XSL file name:



file:///E:\Workspaces\RAD8proguide\RAD8XMLBank\xml\Accounts.xsl

XML input file name:

file:///E:\Workspaces\RAD8proguide\RAD8XMLBank\xml\Accounts.xml

Result file name:

E:\Workspaces\RAD8proguide\RAD8XMLBank\xml_Accounts_transform.html

Split tab: The Split tab in the Page Designer is a combination of the Design and Source tabs where you can view both the Design and the Source simultaneously. You can split the two views by clicking either the vertical or horizontal icon ( ) in the toolbar. Changing the source code automatically changes the design.

8.2.7 XML mapping

The XML Mapping editor is a visual data mapping tool that is designed to transform any combination of XML schema, DTD, or XML documents, and to produce a deployable transformation document. You can map XML-based documents graphically by connecting elements from a source document to elements from a target document. You can extend built-in transformation functions using custom XPath expressions and XSLT templates. The mapping tool automates XSL code generation and produces a transformation document based on the mapping information that you provide.

When mapping between a source XML file and a target XML file, many types of mapping transformation can be applied. The simplest is Move, where the values are transferred between source and target. Other mapping transformations, such as Concat, perform more complex processing on the values. Table 8-1 lists the types of mapping transformation that are available.

Table 8-1 Available mapping transformations

Option	Description
Move	This type copies data from a source to a target.
Concat	This type creates a string concatenation that allows you to retrieve data from two or more entities to link them into a single result.
Inline map	This type enables the map to call out to other maps, but other maps cannot call it. It can only be used within the current map. If the inputs and outputs are arrays, the inline map is implicitly iterated over the inputs.
Submap	This type references another map. It calls or invokes a map from this or another map file. Choosing this transform type is most effective for reuse purposes.
Substring	This type extracts information as required. For example, the substring lastname, firstname with a "," delimiter and a substring index of 0 returns the value lastname. If the substring index was changed to 1, the output is now firstname.
Group	This type takes an array or collection of data and groups it into a collection of a collection. Essentially, it is an array containing an array. Grouping is done at the field level, meaning that it is done by selecting a field of the input collection, such as "department."
Normalize	This type normalizes the input string. For example, it can be used to remove multiple occurrences of white space, such as a space, tab, or return.

Option	Description
Custom	With this type, you can enter custom code or call reference code to be used in the transform. You can extend built-in transformation functions using custom XPath expressions and XSLT templates.

In this section, we map two XML schemas. We use the mapping transformations Move, Concat, Inline map, Substring, and Custom.

Preparing for XML mapping and importing the XSD and XML files

To prepare for the XML mapping, import the provided XSD files and XML file:

Accounts.xsd	The same file created previously and the source schema for the mapping
AccountsList.xsd	An alternate representation of accounts and the target schema for the mapping
Accounts.xml	An XML file that contains sample data and that is similar to the file created previously

To import the files, follow these steps:

1. Create a new folder in the RAD8XMLBank project by right-clicking **RAD8XMLBank** and selecting **New** → **Folder**. For the folder name, type **mapping** and click **Finish**.
2. Right-click the **mapping** folder and select **Import** → **General** → **File System** and click **Next**. Click **Browse** to navigate to the `c:\7835code\xml` folder. Click **OK**. Select **AccountList.xsd** and click **Finish**.
3. Copy the `Accounts.xml` and `Accounts.xsd` files from the `xml` folder to the `mapping` folder.

Starting the XML Mapping editor

Use the XML Mapping editor to create a mapping between the two XML schemas:

1. To start the XML Mapping editor, right-click the **mapping** folder, select **New** → **Other** → **XML** → **XML Mapping**, and click **Next**.

The parent folder is set to `RAD8XMLBank/mapping`.

2. In the File name field, type `Accounts.map`. Click **Next**.

3. For Root inputs, click **Add**. Follow these steps:
 - a. In the Select a root window (Figure 8-12), from the Files of type list, select **XML Schema** and click **Browse**. Expand **RAD8XMLBank**, select **mapping** → **Accounts.xsd**, and click **OK**.
 - b. Under Global elements and types, select the **accounts** element.
 - c. Click **OK**.

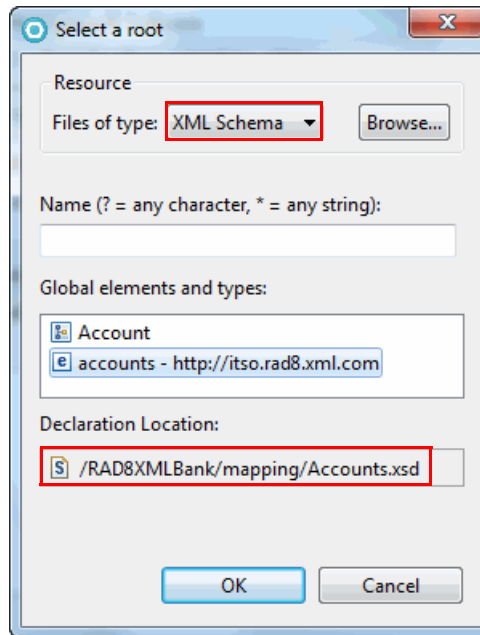


Figure 8-12 Selecting an input root for the mapping

4. For Root outputs, click **Add** and select the **AccountsList.xsd** file and the **accounts** element (same as for the Root input). Click **Next**.
5. In the New XML Mapping window (Figure 8-13 on page 363), to select a sample XML input file, click **Add**. Select **Accounts.xml** and click **OK**. Click **Finish**.

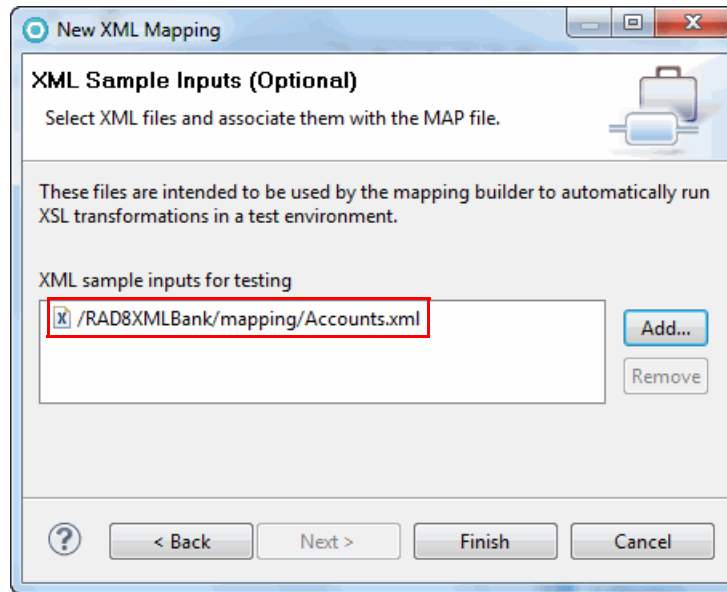


Figure 8-13 XML sample input

The XML Mapping editor opens. In the Enterprise Explorer, three new files are generated:

- ▶ `Accounts.xsl`: An XSL transformation file
- ▶ `Accounts-out.xml`: The transformation output XML file
- ▶ `Accounts.map`: The mapping file

Organizing the XML Mapping editor

One of the advantages of this tool is that you can see the changes that you make to the resulting output xml file when you work on a mapping.

Before we start editing the mapping, open **Accounts-out.xml**, and drag the editor window down to the bottom part of the mapping file (until a down arrow is displayed), so that it sits under `Accounts.map`.

In the workbench layout (Figure 8-14 on page 364), the mapping file is at the top. The resulting xml file is in the middle, and the Properties view is at the bottom.

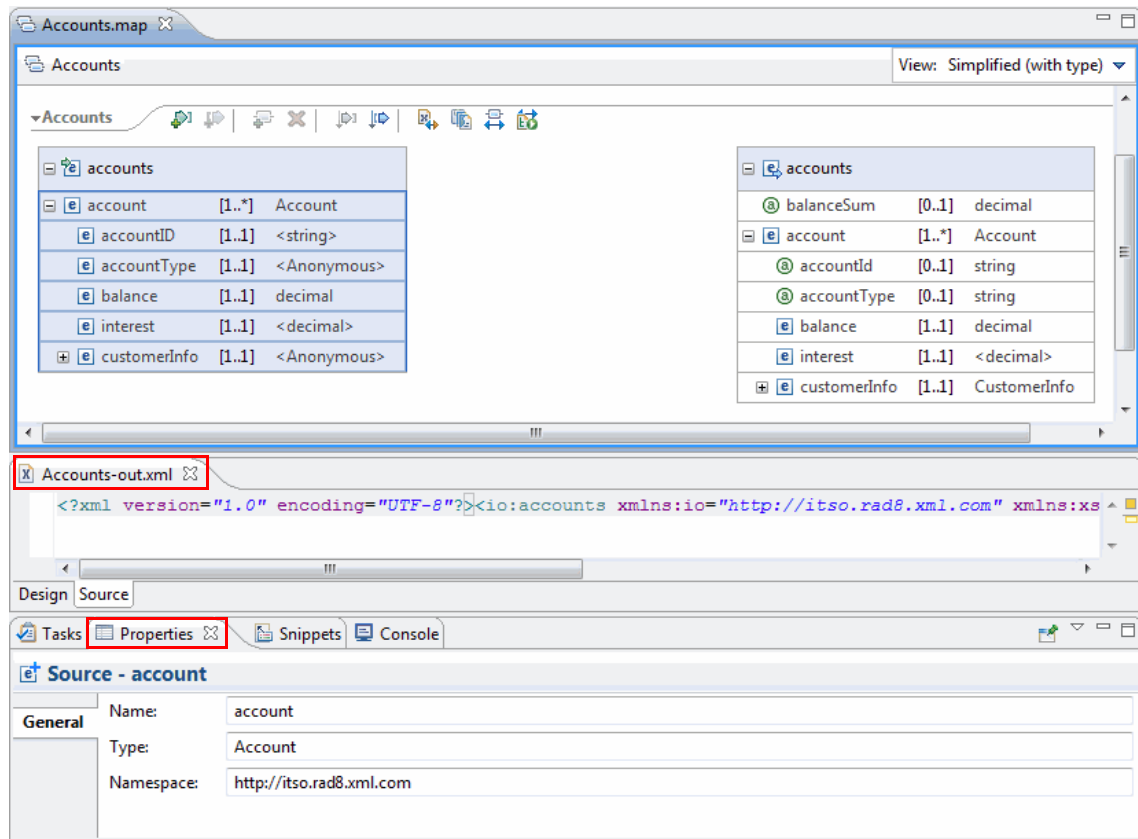


Figure 8-14 Workbench layout for the XML Mapping editor

Editing the XML mapping

Create the mapping between the two XML schemas:

1. In the XML Mapping editor (Figure 8-15 on page 365), select the **account** element from `Accounts.xsd` on the left side and drag it to the **account** element from `AccountList.xsd` on the right side.

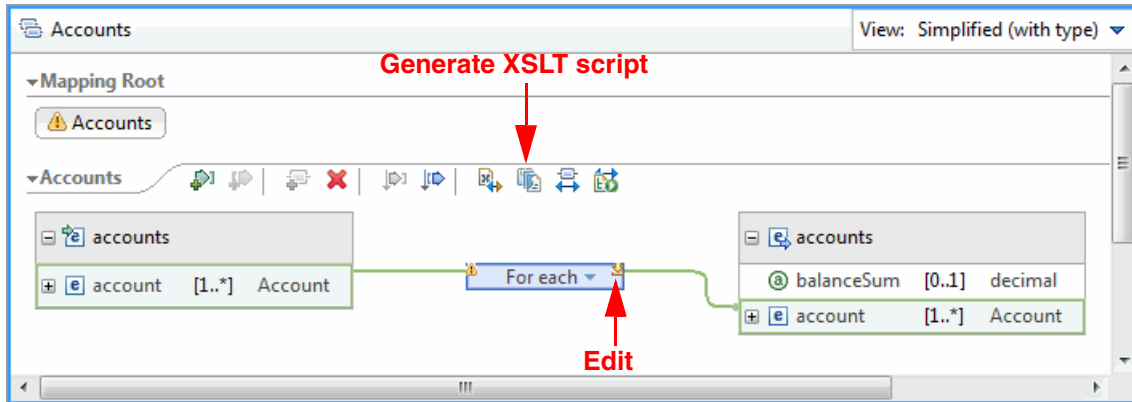


Figure 8-15 Inline map

2. Click the **Generate XSLT script** icon (highlighted in Figure 8-15). Check the Accounts-out.xml file, and you can see that it has changed.

Alternative: You can save the mapping file, and the changes are automatically reflected in the resulting XML file.

Local mapping

To perform inline mapping, follow these steps:

1. Click **Edit** in the upper-right corner of the “For each” map (highlighted in Figure 8-15).
2. In the “For each” map details view, perform the following mapping transforms:
 - a. Map the accountID, accountType, balance, and interest by dragging the elements from the left to the corresponding elements on the right. We map the element accountID to the attribute accountId.
 - b. Click **Generate XSLT script** and verify how the account information is generated in the XML output.

- c. Map the customerInfo element from left to right, which creates a local map (Figure 8-16).

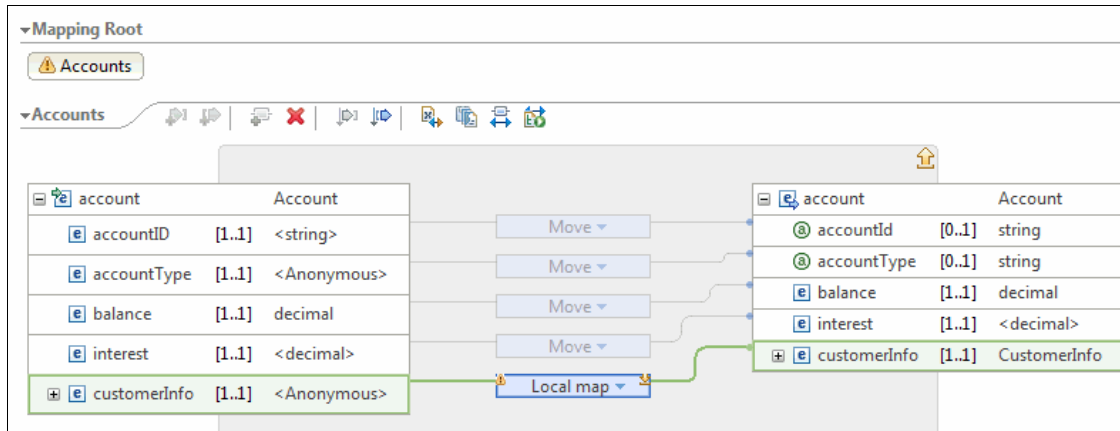


Figure 8-16 Account mapping

3. Click **Generate XSLT script** to see the change in the output XML file.

Concatenation mapping

With *concatenation mapping*, you can define mappings where a set of input values is concatenated to a single output value. In the following steps, you concatenate `firstName` and `lastName` in the source into one `Name` element in the target. Perform the following steps:

1. Click **Edit** in the upper-right corner of the `customerInfo` Local map and add the following transformations:
 - a. Select the **firstName** element and drag it to the **name** element on the right.
 - b. Select the **lastName** element and drag it to the **Move** transform box between `firstName` and `name`.

When we drag a second element to the transform type box, the transform type automatically changes to `Concat` (Figure 8-17 on page 367).

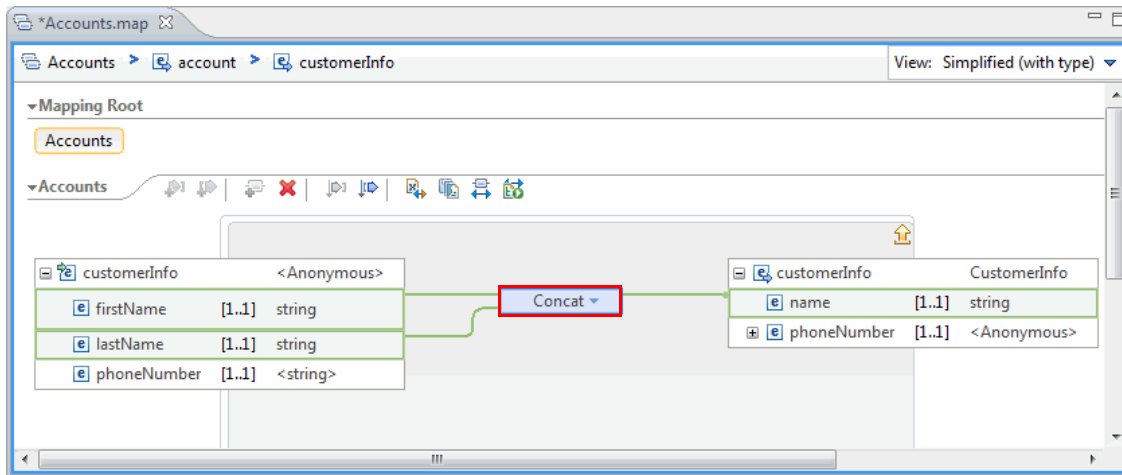



Figure 8-17 Concatenation mapping

2. Concatenate the name in the format `lastName, firstName`:
 - a. Select the **Concat** transformation, and in the Properties view, select the **Order** tab.
 - b. Select **lastName** and click the **Reorder Up** icon ()
 - c. Select the **General** tab, select **lastName**, and type a comma and a space (`,`) in the Delimiter column.
 - d. Click the **Generate XSLT script** icon. The changes are visible in the `Accounts-out.xml` file.

Substring mapping

The telephone number is stored as a single data type in the source document. To separate it into the subelements of area code and local number in the target document, follow these steps:

1. Select the **phoneNumber** element on the left and drag it to the **areaCode** element on the right. Expand the **phoneNumber** element in the target to see the **areaCode** element.
2. Click the drop-down arrow in the transformation and select **Substring** from the list.
3. Right-click the transformation and select **Show in Properties**.
4. In the Properties view, select the **General** tab and perform these steps:
 - a. In the Delimiter field, type a space. Because the telephone number format is `(xxx) xxx-xxxx`, the space must be the delimiter between the area code and local number.

- b. In the Substring index field, type 0.
5. Select the **phoneNumber** element and drag it to the **localNumber** element.
6. Change the transform type to Substring.
7. In the Properties view, Delimiter field, type a space.
8. In the Substring index field, type 1. Figure 8-18 shows the current mapping.

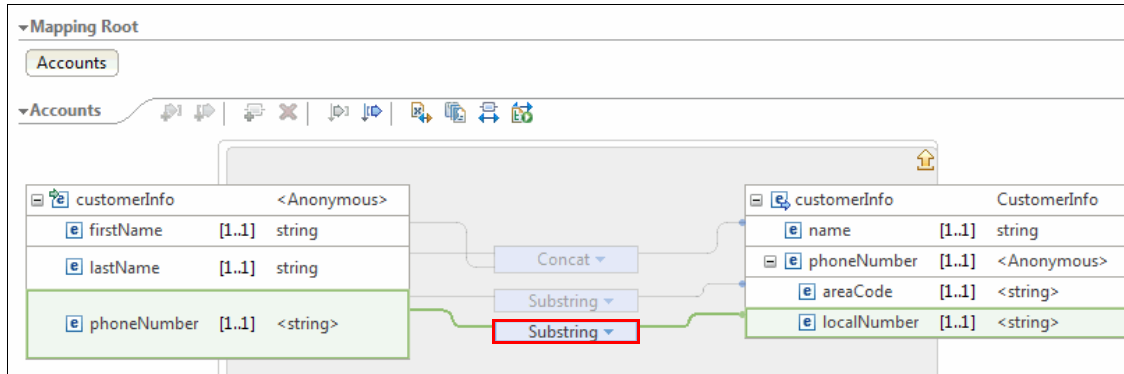



Figure 8-18 Substring mapping

9. To return to the main map, click the **Up a level** icon () in the upper-right corner of the inline map details page. Click the icon twice to return to the main map.

Calculation

We want to calculate the sum of the balance from all accounts and place it in the **balanceSum** attribute of the output document. We use an XPath expression to calculate this total:

1. Select the **accounts** element on the left and drag it to the **balanceSum** attribute on the right.
2. Click the **transform type** box and select **Custom** (Figure 8-19 on page 369).

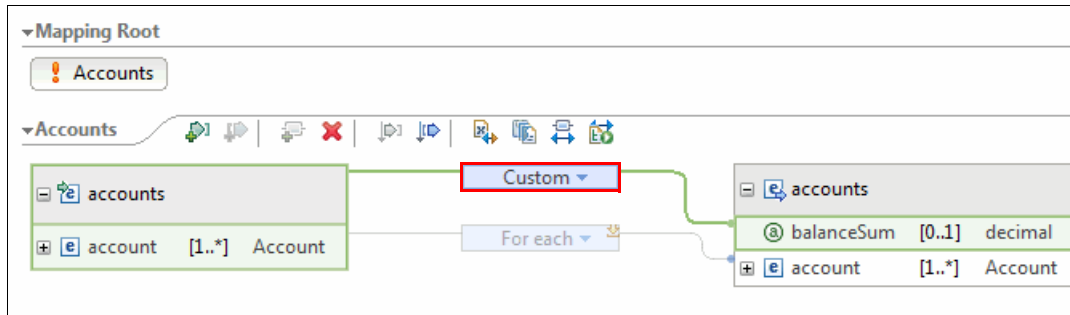


Figure 8-19 Custom mapping

3. Select the **Custom** mapping transformation. In the Properties view, select the **General** tab. For Code, select **XPath**, and for the XPath expression, type `sum(./*/io:balance)` (Figure 8-20).

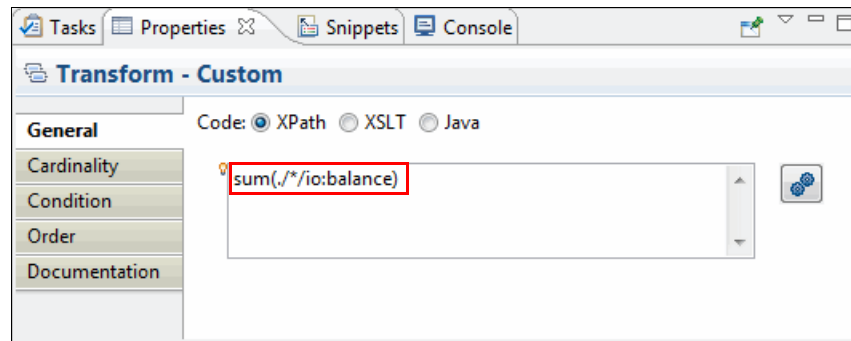


Figure 8-20 Custom mapping using XPath

4. Save the mapping file and click **Generate XSLT script**.

Example 8-4 shows the final output XML file. Notice the `balanceSum` attribute of the `accounts` element, the concatenated name, and the area code and local number.

Example 8-4 Final output showing the `balanceSum` attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<out:accounts xmlns:out="http://itso.rad8.xml.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  balanceSum="300">
  <out:account accountId="123456" accountType="Savings">
    <out:balance>0.0</out:balance>
    <out:interest>5.5</out:interest>
    <out:customerInfo>
```

```
<out:name>Cui, Henry</out:name>
<out:phoneNumber>
  <out:areaCode>(123)</out:areaCode>
  <out:localNumber>555-7890</out:localNumber>
</out:phoneNumber>
</out:customerInfo>
</out:account>
.....
<out:accounts>
```

8.3 Service Data Objects and XML

Service Data Object (SDO) is a framework for data-oriented application development, which includes an architecture and API. SDO simplifies the Java EE programming model and abstracts data in a service-oriented architecture (SOA).

SDO unifies data application development and supports data held in XML documents, incorporates Java EE patterns and leading practices, and provides uniform access to a variety of data sources.

The SDO architecture has the following core concepts:

- | | |
|--------------------|--|
| Data object | Holds a set of named properties, each of which is either of a primitive Java type, such as <code>int</code> or <code>char</code> , or a reference to another data object. The data object API provides functionality for the manipulation of these properties. |
| Data graph | Provides an envelope for data objects, and is the normal unit of transport of objects between components. Data graphs are also responsible for tracking the changes made to the graph of data objects, including inserts, deletes, and the modification of data object properties. |

Data graphs are typically constructed from data sources, such as XML files, EJBs, XML databases, relational databases, or from services, such as web services, resource adapters, and Java Message Service (JMS) messages.

Components that populate data graphs from data sources and commit changes to data graphs back to the data source are called *Data Mediator Services (DMS)*. The DMS architecture and associated APIs are outside the scope of the SDO specification.

For developers to build an XML application quickly, the XML Schema Editor supports the generation of beans from an XML schema. By using these beans, you can quickly create an instance document or load an instance document that conforms to the XML schema.

8.3.1 Generating SDOs from an XML schema

To generate beans from an XML schema, follow these steps:

1. Create a Java project to contain the beans:
 - a. Select **File** → **New** → **Project** → **Java** → **Java Project** and click **Next**.
 - b. In the Project name field, type RAD8XMLBankJava and click **Finish**.
 - c. If you are prompted to switch to the Java perspective, click **Yes**.
2. Generate the JavaBeans:
 - a. In the Enterprise Explorer, expand the **RAD8XMLBank** project, right-click **Accounts.xsd**, and select **Generate** → **Java**.
 - b. In the Generate Java window (Figure 8-21), select the **SDO Generator** and click **Next**.

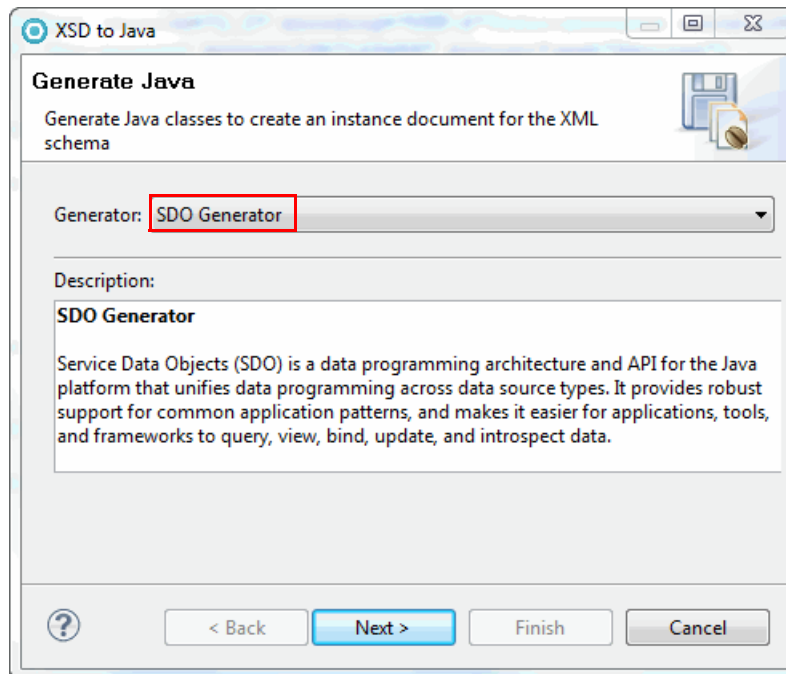


Figure 8-21 XSD to Java window

3. In the Container field, click **Browse**, select **/RAD8XMLBankJava/src**, and click **Finish**.
4. Expand the **RAD8XMLBankJava** project and study the generated packages:
 - com.xml.rad8.itso: The interfaces
 - com.xml.rad8.itso.impl: The implementation classes
 - com.xml.rad8.itso.util: The utility classes

8.3.2 Marshal SDO objects to XML

To test the generated beans, create a test class named `AccountsTest`. This class creates an instance of the `Accounts` object and serializes the instance into XML format:

1. In the Enterprise Explorer, right-click **RAD8XMLBankJava** and select **New** → **Class**.
2. For the package name, type `com.xml.rad8.itso.sdo`, and for the class name, type `AccountsTest`. Click **Finish**.
3. Complete the class with the sample code in Example 8-5. You can find the `AccountsTest.java` file in the `c:\7835code\xml` folder.

Study the `main` method first and then study the helper methods that are called from the `main` method, which are marked in bold. The call to the `save` method on the class `ItsoResourceUtil` serializes the Service Data Objects (SDO) object tree and outputs the string to the supplied file name or output stream class instance.

Example 8-5 AccountsTest program

```
package com.xml.rad8.itso.sdo;

import java.math.BigDecimal;
import com.xml.rad8.itso.*;
import com.xml.rad8.itso.util.ItsoResourceUtil;

public class AccountsTest {
    private DocumentRoot createDocumentRoot() {
        DocumentRoot documentRoot =
        ItsoFactory.eINSTANCE.createDocumentRoot();
        return documentRoot;
    }

    private AccountType createAccountType() {
        AccountType accountType =
        ItsoFactory.eINSTANCE.createAccountType();
```



```

        return accountsType;
    }

    private Account createAccount(AccountTypeType accountType,
        String accountId, BigDecimal balance, BigDecimal interest,
        CustomerInfoType customerInfo) {
        Account account = ItsoFactory.eINSTANCE.createAccount();
        account.setAccountType(accountType);
        account.setAccountID(accountId);
        account.setBalance(balance);
        account.setInterest(interest);
        account.setCustomerInfo(customerInfo);
        return account;
    }

    private CustomerInfoType createCustomerInfo(String firstName,
        String lastName, String phoneNumber) {
        CustomerInfoType customerInfo =
ItsoFactory.eINSTANCE.createCustomerInfoType();
        customerInfo.setFirstName(firstName);
        customerInfo.setLastName(lastName);
        customerInfo.setPhoneNumber(phoneNumber);
        return customerInfo;
    }

    public static void main(String args[]) throws Exception {
        AccountsTest sample = new AccountsTest();
        DocumentRoot documentRoot = sample.createDocumentRoot();

        AccountsType accountsType = sample.createAccountsType();
        CustomerInfoType customerInfo;

        customerInfo = sample.createCustomerInfo("Henry", "Cui",
            "(123) 456-7891");

        Account account =
sample.createAccount(AccountTypeType.SAVINGS,
            "123456", new
BigDecimal(20000.00),
            new BigDecimal(3.5),
customerInfo);
        accountsType.getAccount().add(account);

        customerInfo = sample.createCustomerInfo("Brian", "Hainey",
            "(408) 345-6780");
    }
}

```

```

        account = sample.createAccount(AccountTypeType.FIXED,
"123457",
        new BigDecimal(50000.00), new BigDecimal(6.0),
customerInfo);
        accountsType.getAccount().add(account);

        customerInfo = sample.createCustomerInfo("Craig", "Fleming",
"(408) 345-6789");
        account = sample.createAccount(AccountTypeType.LOAN, "123458",
        new BigDecimal(60000.00), new BigDecimal(8.0),
customerInfo);
        accountsType.getAccount().add(account);

        documentRoot.setAccounts(accountsType);

        ItsoResourceUtil.getInstance().save(documentRoot, System.out);
        ItsoResourceUtil.getInstance().save(documentRoot,
"accounts.xml");
    }
}

```

To execute the Java application, follow these steps:

1. Right-click **AccountsTest.java** and select **Run As** → **Java Application**. The XML result is displayed in the Console view and stored in the `accounts.xml` file.
2. In the Package Explorer, right-click the **RAD8XMLBankJava** project and select **Refresh**.

Example 8-6 shows the generated `accounts.xml` file.

Example 8-6 Generated accounts.xml file

```

<?xml version="1.0" encoding="UTF-8"?>
<itso:accounts xmlns:itso="http://itso.rad8.xml.com">
  <itso:account>
    <itso:accountID>123456</itso:accountID>
    <itso:accountType>Savings</itso:accountType>
    <itso:balance>20000</itso:balance>
    <itso:interest>3.5</itso:interest>
    <itso:customerInfo>
      <itso:firstName>Henry</itso:firstName>
      <itso:lastName>Cui</itso:lastName>
      <itso:phoneNumber>(123) 456-7891</itso:phoneNumber>
    </itso:customerInfo>
  </itso:account>
</itso:accounts>

```

```
</itso:account>
<itso:account>
    .....
</itso:accounts>
```

8.3.3 Unmarshal XML to an SDO data graph

In this section, we explain how to use SDO to access XML documents. You load the `accounts.xml` file into a data graph and display the content of the data graph on the console:

1. Create a new Java class:
 - a. In the Enterprise Explorer, right-click the **com.xml.rad8.itso.sdo** package and select **New** → **Class**.
 - b. For the class name, type `SDOSample`. Select **public static void main(String[] args)** so that a main method is generated.
 - c. Click **Finish**.
2. In the Enterprise Explorer, right-click **RAD8XMLBankJava** and select **Properties**.
3. In the Properties window:
 - a. Select **Java Build Path** → **Libraries**.
 - b. Click **Add External JARs** and add the **org.eclipse.emf.ecore.change_2.5.1.v20100907-1643** file, which is in the `<SDPShared>/plugins` installation folder.
 - c. Click **OK** to add the new JAR to the project.
4. Add the sample code to the main method, including `throws IOException` (Example 8-7).

Example 8-7 Code to load an XML document

```
public static void main(String[] args) throws IOException {
    System.out.println("\n--- Printing XML document to System.out
    ---");
    DocumentRoot documentRoot =
        ItsoResourceUtil.getInstance().load("Accounts.xml");
    ItsoResourceUtil.getInstance().save(documentRoot, System.out);
    System.out.println("\n\n--- Done ---");
}
```

5. Select **Source** → **Organize Imports** to add the import statements.

6. Right-click **SDOSample** and select **Run As** → **Java Application**. The `accounts.xml` file is displayed in the Console.

Navigating the SDO data graph

XPath expressions are used to obtain data from the data objects present in the data graph after the XML file is loaded. Figure 8-22 shows the data graph for the `accounts.xml` file.

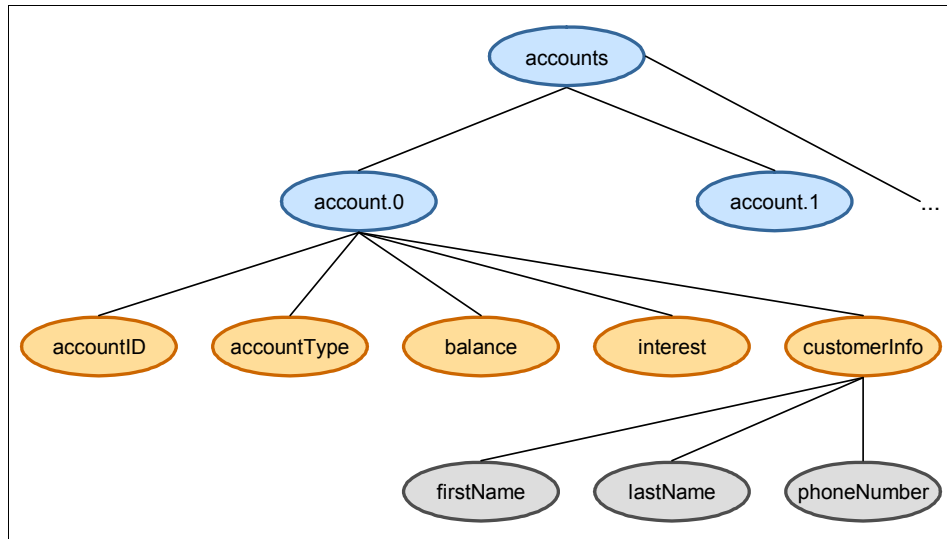


Figure 8-22 Accounts data graph

Follow these steps:

1. Add the code that is shown in Example 8-8 to the main method.

Example 8-8 Java code for navigating the SDO data graph

```
// navigating the SDO data graph
AccountsType accountsType = documentRoot.getAccounts();
DataObject accountsTypeImpl = (AccountsTypeImpl) accountsType;
DataObject account1 = accountsTypeImpl.getDataObject("account.0");
System.out.println("\n\nThe first account is: " + account1 + "\n");
DataObject account2 = accountsTypeImpl.getDataObject
    ("account[accountID = '123457']");
System.out.println("The second account is: " + account2 + "\n");
DataObject account2CustomerInfo = accountsTypeImpl.getDataObject
    ("account[accountID = '123457']/customerInfo");
System.out.println("The second account customer information is: " +
    account2CustomerInfo + "\n");
```

```
String account1CustomerName =
account1.getString("customerInfo/firstName");
System.out.println("The first account customer first name is " +
    account1CustomerName + "\n");
```

Note the following comments:

- The XPath dot notation is used to index data objects. The first object has index 0. Therefore `account.0` returns the first account data object.
 - The XPath expression `account[accountID = '123457']` returns the account data object whose account ID equals 123457.
 - `account[accountID = '123457']/customerInfo` is an XPath expression that returns a data object multiple levels beneath the root data object.
2. Right-click **SDOSample** and select **Run As** → **Java Application**. You can also select **Run** → **Run History** → **SDOSample**.

Example 8-9 shows the Console output from the SDO graph navigation code.

Example 8-9 Output from the SDO graph navigation code

```
The first account is: com.xml.rad8.itso.impl.AccountImpl@27332733
(accountID: 123456, accountType: Savings, balance: 20000, interest:
3.5)
```

```
The second account is: com.xml.rad8.itso.impl.AccountImpl@30d030d
(accountID: 123457, accountType: Fixed, balance: 50000, interest: 6)
```

```
The second account customer information is:
com.xml.rad8.itso.impl.CustomerInfoTypeImpl@7c207c2 (firstName:
Brian, lastName: Hainey, phoneNumber: (408) 345-6780)
```

```
The first account customer first name is Henry
```

Updating the SDO data graph

An SDO data graph can be modified and the modifications reflected in the source XML file that was loaded. In this example, you update the interest rate of one account, add an account, and finally delete an existing account (Example 8-10).

Example 8-10 Updating an SDO data graph

```
// updating the SDO data graph
account1.setString("interest", "10");
DataObject account3 = accountsTypeImpl.createDataObject("account");
account3.setString("accountID", "333333");
account3.set("accountType", AccountType.LOAN);
```

```
account3.setString("balance", "999999");
account3.setString("interest", "2.5");
DataObject newCustomerInfo = account3.createDataObject("customerInfo");
newCustomerInfo.setString("firstName", "Mike");
newCustomerInfo.setString("lastName", "Smith");
newCustomerInfo.setString("phoneNumber", "(201) 654-8754");
account2.delete();
System.out.println("\n--- Printing updated XML document ---");
ItsoResourceUtil.getInstance().save(documentRoot, System.out);
```

You can find the complete code listing of `SDOSample.java` in the `c:\7835code\xml` folder.

Select **Run** → **Run History** → **SDOSample**.

You can see that the interest rate for the first account (`accountID = '123456'`) has been updated, the second account (`accountID = '123457'`) has been removed, and a new account (`accountID = '333333'`) has been added. Notice that the `accounts.xml` file is not updated, because we only saved to the Console.

8.4 JAXB and XML

Java Architecture for XML Binding (JAXB) is a Java technology that provides an easy and convenient way to map Java classes and XML schema for simplified development of web services. JAXB uses the flexibility of platform-neutral XML data in Java applications to bind XML schema to Java applications without requiring extensive knowledge of XML programming. The tools included in this workbench implement JAXB 2.0 and 2.1 standards.

JAXB is an XML to Java binding technology that supports transformation between schema and Java objects and between XML instance documents and Java object instances. JAXB consists of a runtime application programming interface (API) and accompanying tools that simplify access to XML documents. JAXB also helps to build XML documents that both conform to and validate the XML schema. The application server supports the W3C XML Schema as defined in the XML Schema 1.0 Recommendation (XSD Part 1 and 2).

JAXB-annotated classes and artifacts contain all the information needed by the JAXB runtime API to process XML instance documents. The JAXB runtime API supports marshaling JAXB objects to XML and unmarshaling the XML document back to JAXB class instances. Optionally, you can use JAXB to provide XML validation to enforce both incoming and outgoing XML documents to conform to the XML constraints defined within the XML schema.

JAXB is the default data binding technology that the Java API for XML Web Services (JAX-WS) tooling uses and is the default implementation within Rational Application Developer. You can develop JAXB objects for use within JAX-WS applications. You can also use JAXB independently of JAX-WS when you want to use the XML data binding technology to manipulate XML within your Java applications.

8.4.1 Generating JAXB classes from an XML schema

To enable you to map to and from XML data and Java objects, you can use the JAXB Schema to JavaBean wizard. The wizard generates Java beans that correspond to your schema. To generate beans from a JAXB schema, follow these steps:

1. Create a Java project to contain the beans:
 - a. Select **File** → **New** → **Project** → **Java** → **Java Project** and click **Next**.
 - b. In the Project name field, type RAD8XMLBankJAXB and click **Finish**.
 - c. If you are prompted to switch to the Java perspective, click **Yes**.
2. Generate the JavaBeans:
 - a. In the Enterprise Explorer, expand the **RAD8XMLBank** project, right-click **Accounts.xsd**, and select **Generate** → **Java**.
 - b. In the Generate Java window (Figure 8-23 on page 380), select **Schema to JAXB Classes** and click **Next**.

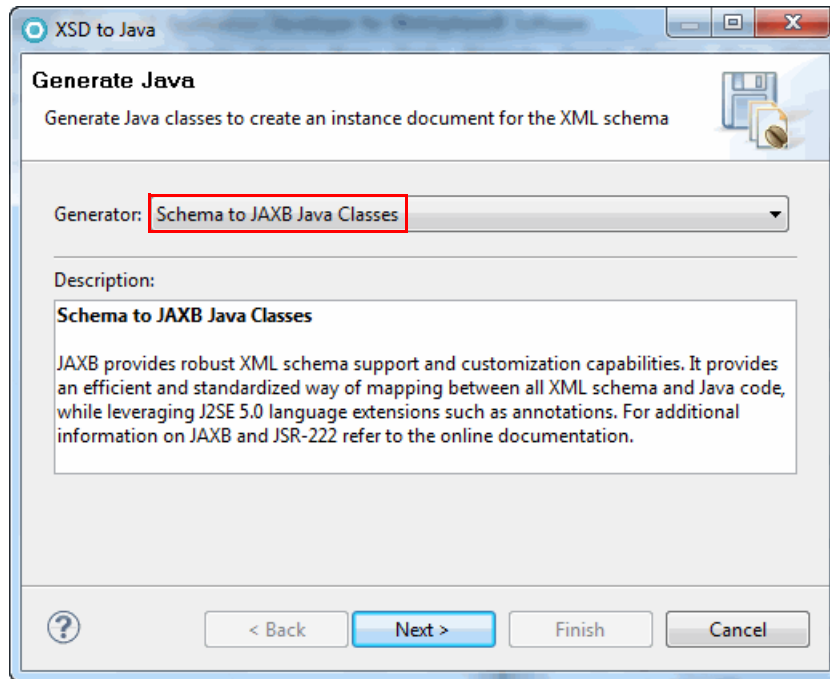


Figure 8-23 JAXB generator window

- c. In the Target Java Container field, select **/RAD8XMLBankJAXB/src**.
 - d. Select Target Package **<default-package-mapping>**.
 - e. Select JAX-WS version **2.2**.
 - f. Clear **Generate serializable JAXB classes**.
 - g. Click **Finish**.
3. Expand the **RAD8XMLBankJava** project and study the generated packages:
- `ObjectFactory.java`: Contains factory methods for each Java content interface and Java element interface generated in the `com.xml.rad8.itso` package.
 - `Accounts.java` and `Account.java`: Java object mapped from XML schema type.
 - `package-info.java`: Package-level annotations are declared inside.

8.4.2 Marshal JAXB objects to XML

After JAXB bindings exist, you can use the JAXB binding runtime API to convert XML instance documents to and from Java objects. Data stored in an XML document is accessible without the need to understand the data structure. JAXB annotated classes and artifacts contain all the information that the JAXB runtime API needs to process XML instance documents. The JAXB runtime API enables the marshaling of JAXB objects to XML and the unmarshaling of the XML document back to the JAXB object.

In this section, you use the JAXB runtime API to marshal the JAXB object instances into an XML instance document:

1. In the Enterprise Explorer, right-click **RAD8XMLBankJAXB** and select **New → Class**.
2. For the package name, type `com.xml.rad8.itso.jaxb`, and for the class name, type `Marshal`. Click **Finish**.
3. Complete the class with the sample code in Example 8-11. You can find the `Marshal.java` file in the `c:\7835code\xml` folder.

Example 8-11 Marshal.java

```
package com.xml.rad8.itso.jaxb;

import java.math.BigDecimal;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import com.xml.rad8.itso.Account;
import com.xml.rad8.itso.Account.CustomerInfo;
import com.xml.rad8.itso.Accounts;

public class Marshal {

    public static void main(String[] args) {
        try {
            // create a JAXBContext capable of handling classes
            generated into the com.xml.rad8.itso package
            JAXBContext jc =
JAXBContext.newInstance("com.xml.rad8.itso" );
            // create a Marshaller and marshal to a file
            Marshaller m = jc.createMarshaller();
            Accounts accounts = new Accounts();
            Account account1= new Account();
            account1.setAccountID("1111111");
```

```

        account1.setAccountType("Savings");
        account1.setBalance(new BigDecimal(9999999));
        CustomerInfo customerInfo= new CustomerInfo();
        customerInfo.setFirstName("Henry");
        customerInfo.setLastName("Cui");
        customerInfo.setPhoneNumber("123-555-7890");
        account1.setCustomerInfo(customerInfo);
        accounts.getAccount().add(account1);
        m.setProperty( Marshaller.JAXB_FORMATTED_OUTPUT,
Boolean.TRUE );
        m.marshal( accounts, System.out );
    } catch( JAXBException je ) {
        je.printStackTrace();
    }
}
}

```

4. Examine the code. The marshalling process involves two steps:
 - a. Instantiate your JAXB classes.
 - b. Invoke the JAXB marshaller.
5. To execute the Java application, follow these steps:
 - a. Right-click **Marshal.java** and select **Run As** → **Java Application**. The XML result is displayed in the Console view.
 - b. Example 8-12 shows the Console output from the JAXB code.

Example 8-12 Output from Marshal.java

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<accounts xmlns="http://itso.rad8.xml.com">
  <account>
    <accountID>1111111</accountID>
    <accountType>Savings</accountType>
    <balance>9999999</balance>
    <customerInfo>
      <firstName>Henry</firstName>
      <lastName>Cui</lastName>
      <phoneNumber>123-555-7890</phoneNumber>
    </customerInfo>
  </account>
</accounts>

```

8.4.3 Unmarshal the XML file to JAXB objects

In this section, you use the JAXB runtime API to unmarshal the XML into JAXB objects:

1. In the Enterprise Explorer, right-click **RAD8XMLBankJAXB** and select **New** → **Class**.
2. For the package name, type `com.xml.rad8.itso.jaxb`, and for the class name, type `Unmarshal`. Click **Finish**.
3. Complete the class with the sample code in Example 8-13. You can find the `Unmarshal.java` file in the `c:\7835code\xml` folder.

Example 8-13 Unmarshal.java

```
package com.xml.rad8.itso.jaxb;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.List;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
import com.xml.rad8.itso.Account;
import com.xml.rad8.itso.Accounts;

public class Unmarshal {
    public static void main(String[] args) {
        try {
            // Create a JAXBContext capable of handling classes
            // generated into the com.xml.rad8.itso package
            JAXBContext jc = JAXBContext.newInstance(
                "com.xml.rad8.itso" );
            // create an Unmarshaller
            Unmarshaller u = jc.createUnmarshaller();
            // unmarshal the XML document into Java object
            Accounts accounts =(Accounts)u.unmarshal( new
                FileInputStream( "Accounts.xml" ) );
            System.out.println( "Account information: " );
            // display the account information
            List<Account> accountList = accounts.getAccount();
            for (Account account: accountList){
                System.out.println("-----");
                System.out.println("Account ID: "+
                    account.getAccountID());
            }
        }
    }
}
```

```

        System.out.println("Customer name: "+
account.getCustomerInfo().getFirstName() + "
"+account.getCustomerInfo().getLastName());
        System.out.println("Account type: "+
account.getAccountType());
        System.out.println("Account balance: "+
account.getBalance());
        System.out.println("Account interest: "+
account.getInterest());
    }
    } catch( JAXBException je ) {
        je.printStackTrace();
    } catch( IOException ioe ) {
        ioe.printStackTrace();
    }
}
}

```

4. Examine the code. The unmarshalling process involves two steps:
 - a. Obtain an existing XML instance document.
 - b. Invoke the JAXB unmarshaller.
5. To execute the Java application, follow these steps:
 - a. Right-click **Unmarshal.java** and select **Run As** → **Java Application**. The XML result is displayed in the Console view.
 - b. Example 8-14 shows the Console output.

Example 8-14 Output from Unmarshal.java

```

Account information:
-----
Account ID: 123456
Customer name: Henry Cui
Account type: Savings
Account balance: 0.0
Account interest: 5.5
-----
Account ID: 234567
... ..

```

8.4.4 JAXB customization

In many cases, the default bindings generated by the JAXB generator will be sufficient to meet your needs. However, there are certain cases in which you might want to modify the default bindings. For example, you want to change the default namespace to package mapping, customize the generated class/attribute names, and solve naming conflicts.

In this section, we provide an example to customize the default binding between an XML schema component and its Java representation by adding an external binding declaration. Example 8-15 shows the external binding file. You can find the binding.xjb file in the c:\7835code\xml folder.

Example 8-15 binding.xjb

```
<jxb:bindings xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:jxb="http://java.sun.com/xml/ns/jaxb"
  xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
  version="1.0">
  <jxb:bindings schemaLocation="Accounts.xsd" node="/xs:schema">
    <jxb:schemaBindings>
      <jxb:package name="com.ibm.rad8.customization" />
    </jxb:schemaBindings>
    <jxb:bindings node="//xs:complexType[@name='Account']">
      <jxb:class name="ITS0Account">
        <jxb:javadoc>A <b>Account</b> consists of account ID,
account
        type, balance, and customer info.</jxb:javadoc>
      </jxb:class>
    </jxb:bindings>
    <jxb:bindings node="//xs:element[@name='accountID']">
      <jxb:property name="ID" />
    </jxb:bindings>
  </jxb:bindings>
</jxb:bindings>
```

Examine the binding file:

- ▶ `<package>` specifies the name of the package. This means the generated code will be put under package `com.ibm.rad8.customization`.
- ▶ `<class>` declarations are used to customize the name for a schema-derived Java interface. `node="//xs:complexType[@name='Account']"` is an XPath expression to find the `Account` type in the XML schema. The `Account` type will be mapped to `ITS0Account.java`.

- ▶ The `<javadoc>` element specifies the Javadoc tool annotations for the schema-derived Java interface.
- ▶ The `<property>` binding declaration enables you to customize the binding of an XML schema element to its Java representation as a property. The `accountID` will be mapped to the property ID in the generated JavaBean.

Perform these steps to use the external JAXB binding file:

1. Import **binding.xjb** from the `c:\7835code\xml` folder into the `xml` folder of the **RAD8XMLBank** project.
2. In the Enterprise Explorer, expand the **RAD8XMLBank** project, right-click **Accounts.xsd**, and select **Generate** → **Java**.
3. In the Generate Java window, select **Schema to JAXB Classes** and click **Next**.
4. In the Target Java Container field, select **/RAD8XMLBankJAXB/src** (Figure 8-24).
5. In the Binding Files section, click **Add** and select **binding.xjb** in the `xml` folder of your RAD8XMLBank project.

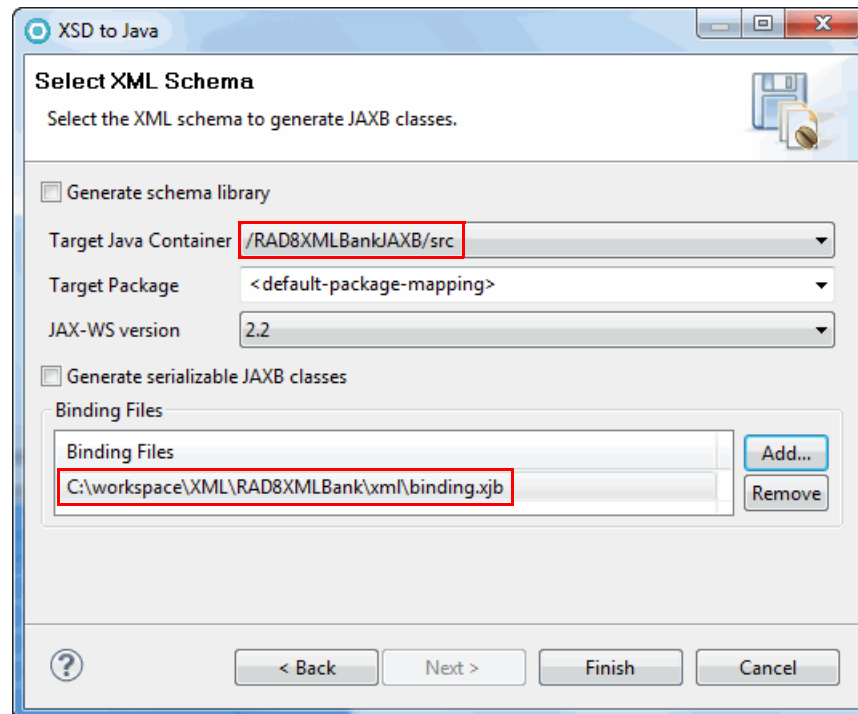


Figure 8-24 JAXB generator window

6. Click **Finish**.
7. Expand the **RAD8XMLBankJAXB** project and study the generated files:
 - a. The generated code is put into package `com.ibm.rad8.customization`.
 - b. `ITSOAccount.java` is generated, instead of `Account.java`.
 - c. Open **ITSOAccount.java**. You can see a comment is put into this class (Example 8-16).

Example 8-16 ITSOAccount.java

```
/**
 * A <b>Account</b> consists of account ID, account
 *           type, balance, and customer info.
 *
 *
 *
 */
```

- d. In `ITSOAccount.java`, the `accountID` is mapped to a Java property ID:

```
@XmlElement(name = "accountID", required = true)
protected String id;
```

8.5 Feature Pack for XML

The IBM WebSphere Application Server V7 Feature Pack for XML supports the following new or updated World Wide Web Consortium (W3C) XML standards:

- ▶ Extensible Stylesheet Language Transformations (XSLT) 2.0
Programming language that is used to transform XML into a new XML format or into another presentation-oriented format, such as HTML, XHTML, or Scalable Vector Graphics (SVG)
- ▶ XML Path Language (XPath) 2.0
Programming language that is designed to allow developers to select nodes from an XML document
- ▶ XML Query Language (XQuery) 1.0
Query language that is built with the intent of enabling access to collections of XML documents in a way that bridges the retrieval of both structured and unstructured data

The Feature Pack for XML also provides the IBM XML API to support these standards. This API invokes a runtime engine that is capable of executing XPath 2.0, XSLT 2.0, and XQuery 1.0, as well as manipulating the returned XML data.

The new XML processor (XPath 2.0, XSLT 2.0, and XQuery 1.0), which was first provided by the WebSphere Application Server V7 Feature Pack for XML, has become a core component in WebSphere Application Server V8 Beta. To learn more about the Feature Pack for XML, visit the following resources:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.xmlfep.multiplatform.doc/info/ae/ae/welcome_fepxml.html

8.6 More information

For more information about XML schemas, see the following web address:

<http://www.w3.org/XML/Schema>

For more information about XML, see the following web address:

<http://www.w3.org/XML/>

For more information about XML parsers, see the following web addresses:

- ▶ Xerces (XML parser - Apache)
<http://xml.apache.org/xerces2-j>
- ▶ Xalan (XSLT processor - Apache)
<http://xml.apache.org/xalan-j>
- ▶ JAXP (XML parser - Sun)
<http://java.sun.com/xml/jaxp>
- ▶ SAX2 (XML API)
<http://sax.sourceforge.net>

For more information about SDO, see the following web addresses:

- ▶ Introduction to Service Data Objects on developerWorks
<http://www-128.ibm.com/developerworks/java/library/j-sdo/>
- ▶ Open service-oriented architecture: SDO Resources
<http://www.osoa.org/display/Main/SDO+Resources>

For more information about JAXB specification, see the following web address:

<http://jcp.org/en/jsr/detail?id=222>



Part 3

Persistence and enterprise information system integration development

In this part, we describe the tooling and technologies provided by Rational Application Developer to develop applications using databases and the Java Persistence API (JPA). We also describe how to work with enterprise information systems.

This part includes the following chapters:

- ▶ Chapter 9, “Developing database applications” on page 393
- ▶ Chapter 10, “Persistence using the Java Persistence API” on page 443

- ▶ Chapter 11, “Developing applications to connect to enterprise information systems” on page 531

Sample code for download: The sample code for all the applications developed in this part is available for download at the following address:

<ftp://www.redbooks.ibm.com/redbooks/SG247835>

See Appendix C, “Additional material” on page 1877, for instructions.



Developing database applications

In an enterprise environment, applications that use databases are common. In this chapter, we explore technologies that are used in developing Java database applications. In this chapter, we highlight the database tooling that is provided with IBM Rational Application Developer.

This chapter is organized into the following sections:

- ▶ Introduction
- ▶ Connecting to the ITSOBANK database
- ▶ Connecting to databases
- ▶ Creating SQL statements
- ▶ Developing Java stored procedures
- ▶ Developing SQLJ applications
- ▶ Data modeling

The sample code for this chapter is in the 7835code\database folder.

9.1 Introduction

Rational Application Developer provides rich features to make it easier to work with tables, views, and filters; create and work with SQL statements; create and work with database routines (such as stored procedures and user-defined functions); and create and work with SQLJ files. You can also create, modify, and generate data models. Depending on the project requirements, you have to take certain steps to set up your work environment.

Depending on the project requirements, this chapter is written for three types of users:

- ▶ If you want to *access* databases and discover information about them, you can use the Data Source Explorer to create a connection to those databases. After you have set up connection information for a database, you can connect, refresh a connection, and browse the objects that are in the database.
- ▶ If you want to *develop* database-related activities, such as SQL queries and stored procedures, you have to create a data development project. The data development project stores your routines and other data development objects. Rational Application Developer also provides tooling to assist you to develop SQLJ applications, and offers a DB beans package to access database information without directly using the Java Database Connectivity (JDBC) interface.
- ▶ If you want to *design* your database model, you have to create a data design project to store your objects. The modeling tool assists you to build a data model, analyze the model, perform the impact analysis, and so forth.

All examples in this chapter are demonstrated against the open source embedded Derby database server. The embedded version of Derby is bundled inside Rational Application Developer, so its availability is guaranteed. These examples can be easily applied to DB2 databases.

9.2 Connecting to the ITSOBANK database

We provide two implementations of the ITSOBANK database, Derby and DB2. Follow the instructions in “Setting up the ITSOBANK database” on page 1880 to set up the database.

The ITSOBANK database has four tables: CUSTOMER, ACCOUNT, ACCOUNT_CUSTOMER, and TRANSACTIONS. The name TRANSACTION is reserved in the Derby database. Therefore, we use TRANSACTIONS.

An account can have multiple transactions, and the ACCOUNT_ID becomes the foreign key in the TRANSACT table and is related to the primary key of the ACCOUNT table.

There is a many-to-many association between customers and accounts. ACCOUNT_CUSTOMER is the junction table to turn this many-to-many relationship into a one-to-many relationship between CUSTOMER and ACCOUNT_CUSTOMER and a one-to-many relationship between ACCOUNT and ACCOUNT_CUSTOMER.

9.2.1 Connecting to databases

With Rational Application Developer, you can create a connection to the following databases:

- ▶ Cloudscape
- ▶ DB2 for Linux, UNIX®, and Windows
- ▶ DB2 UDB for i
- ▶ DB2 UDB for z/OS
- ▶ Derby
- ▶ Generic JDBC
- ▶ Informix
- ▶ MySQL
- ▶ Oracle
- ▶ SQL Server
- ▶ Sybase

For more information about supported databases, see the following website:

<http://www-01.ibm.com/support/docview.wss?rs=2042&uid=swg27019500#DB>

9.2.2 Creating a connection to the ITSOBANK database

To connect to the Derby ITSOBANK database using the New Database Connection wizard, follow these steps:

1. Stop the WebSphere Application Server V8.0 Beta if it is running and if you have accessed the ITSOBANK database for exercises in other chapters of this book, because Derby only allows one connection.
2. Select **Window** → **Open Perspective** → **Other** to open the Data perspective. In the Open Perspective window, select **Data** and click **OK**.
3. In the Data perspective, locate the Data Source Explorer view, which is typically in the lower left in the Data perspective.
4. In the Data Source Explorer, right-click **Database Connections** and select **New**.

5. In the New Connection wizard (Figure 9-1 on page 397), follow these steps:
 - a. Clear **Use default naming convention**, and for Connection Name, type ITSOBANKderby.
 - b. For Select a database manager, select **Derby**.
 - c. For JDBC driver, select **Derby 10.2 - Embedded JDBC Driver Default**.
 - d. For Database location, click **Browse** and locate **C:\7835code\database\derby\ITSOBANK**.
 - e. Leave the User name and Password fields empty, because the Derby database does not require authentication.
 - f. Select **Create database (if required)**.
 - g. Click **Test Connection**. A window opens and shows the “Connection succeeded” message. Click **OK** to close the window.
 - h. Click **Next**.

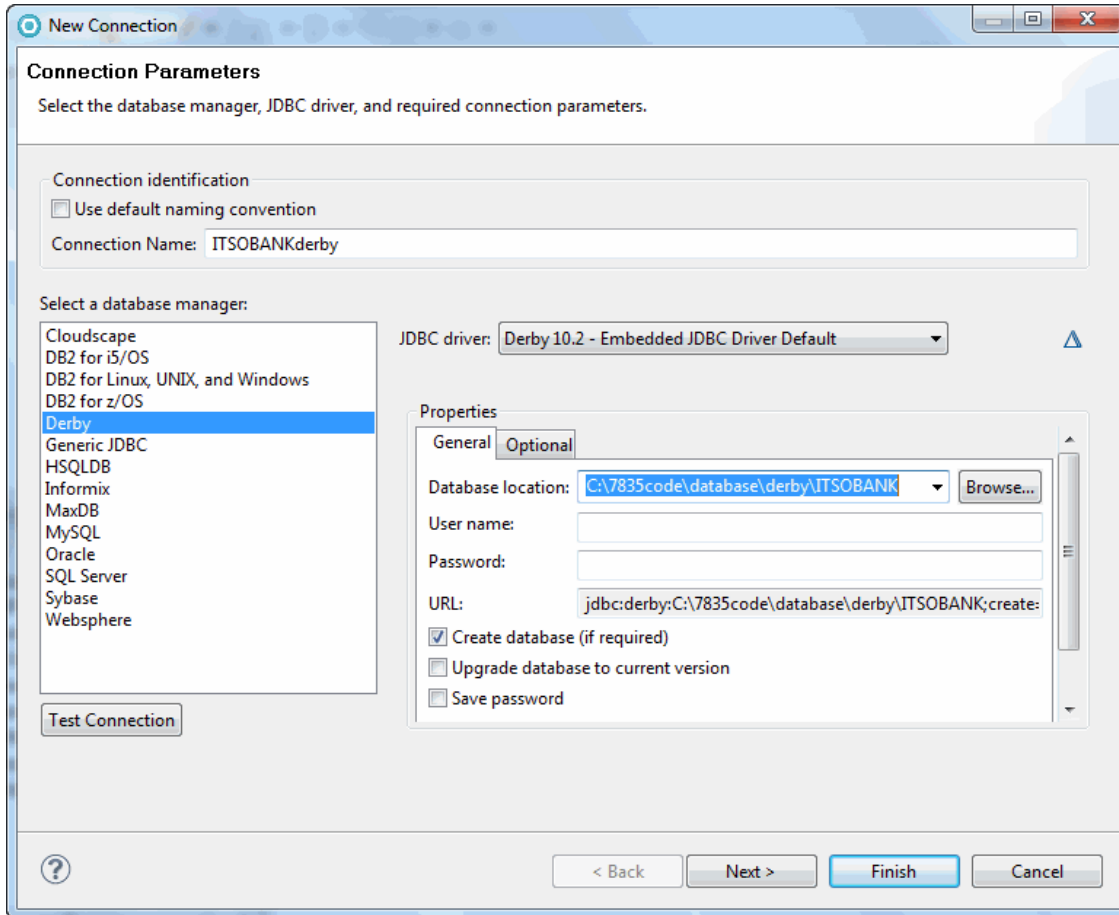


Figure 9-1 New Connection: Connection Parameters window

You can use filters to exclude data objects, such as tables, schemas, stored procedures, and user-defined functions, from the view. Only the data objects that match the filter condition are shown.

6. To see the objects in the ITSO schema, in the Filter window (Figure 9-2 on page 398), complete these actions:
 - a. Clear the **Disable filter** check box.
 - b. Select **Selection**.
 - c. Select **Include selected items**.
 - d. From the schema list, select **ITSO**.
 - e. Click **Finish**.

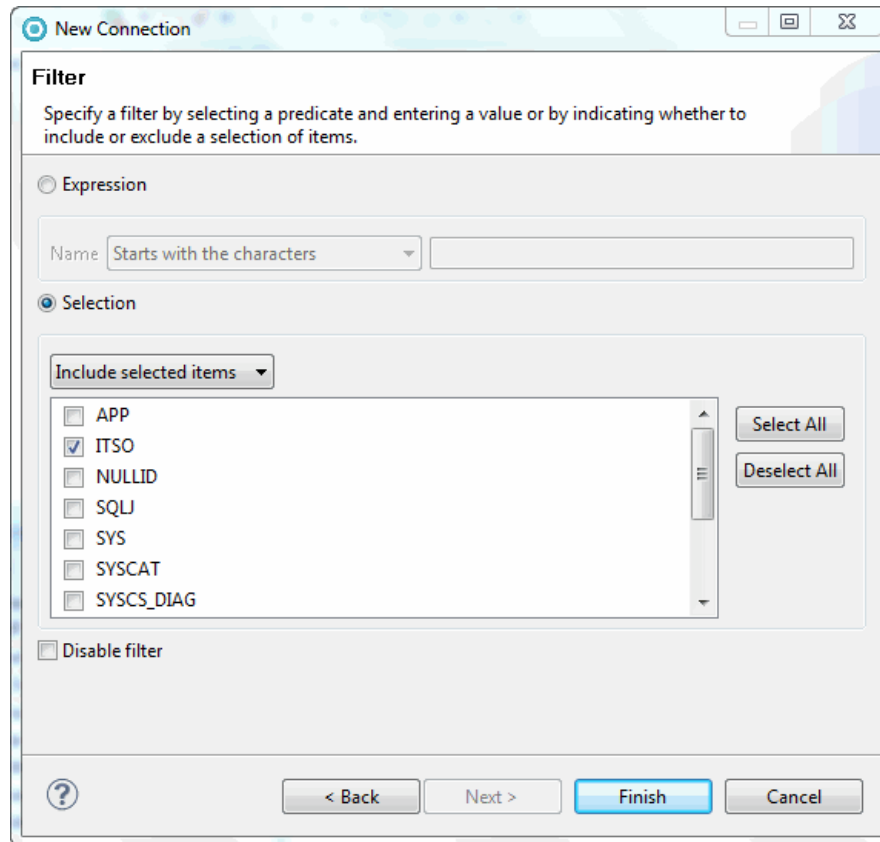


Figure 9-2 New Connection: Filter window

7. When the connection is displayed in the Data Source Explorer, expand **ITSOBANKderby [Apache Derby 10.5.1.1 ...]** → **ITSOBANK**. The Schemas

folder is marked as [Filtered]. Only one schema (ITSO) is listed, and the others are filtered (Figure 9-3).

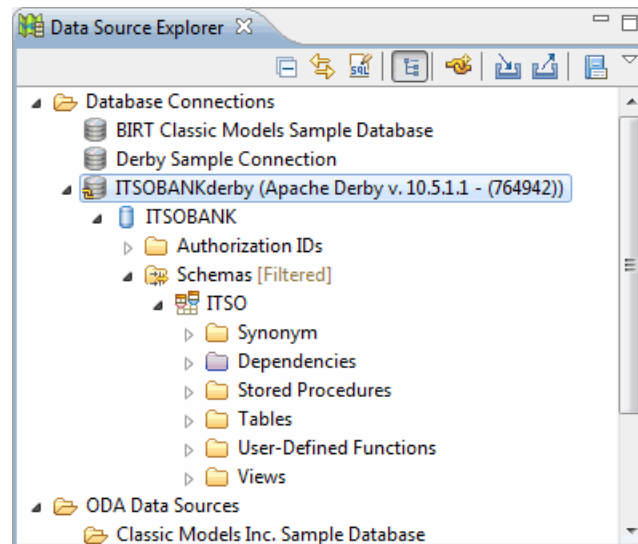


Figure 9-3 Connection with schema and tables in Data Source Explorer

With the filter framework, you can filter out the tables at a more granular level. Suppose we only want to see tables that start with the letter A. Follow these steps:

1. Expand the schema **ITSO**, right-click **Tables**, and select **Filter**.
2. In the Filter window, follow these steps:
 - a. Clear the **Disable filter** check box.
 - b. Select **Expression**.
 - c. In the Name section, select **Starts with the characters** and type A.
 - d. Click **OK**.

Now you can only see two tables in the Data Source Explorer: **ACCOUNT** and **ACCOUNT_CUSTOMER**.

We use the four tables in later sections.

To disable the filter, right-click **Tables**, select **Filters**, and select **Disable filter**. Click **OK**.

9.2.3 Browsing a database with the Data Source Explorer

The Data Source Explorer view operates similarly to a graphical directory browsing program. It provides a list of configured connection profiles. Here, you can create and manage database connections, browse data objects in a connection, modify data objects, and more.

Follow these steps to explore the Derby ITSOBANK database:

1. Expand **ITSOBANKderby (...)** → **ITSOBANK** → **Schemas** → **ITSO** → **Tables** → **CUSTOMER** (Figure 9-4).
 - a. Expand **Columns**. All the columns in the CUSTOMER table are listed. The Social Security number (SSN) is marked as a primary key.
 - b. Expand **Constraints**. PK_CUSTOMER is listed as the primary key constraint.

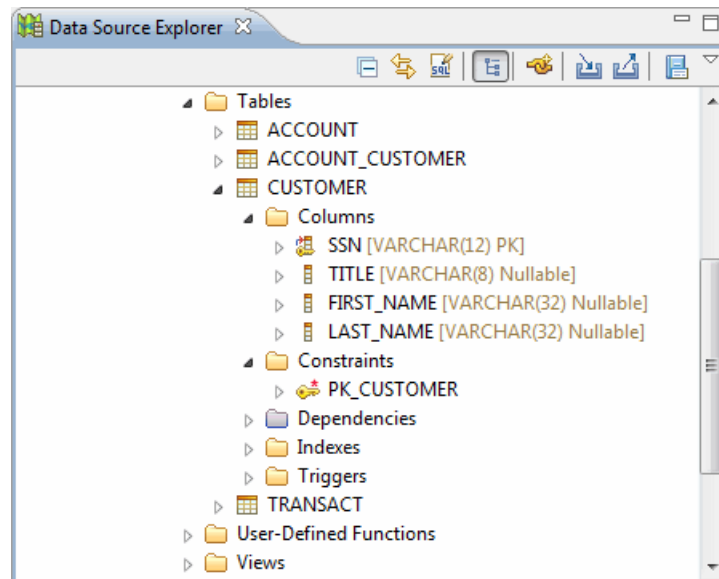


Figure 9-4 Customer table with columns

2. In the Data Source Explorer view, right-click the **Customer** table and select **Data** → **Sample Contents**. The action opens the SQL Results view (Figure 9-5 on page 401), and the status of the running result is shown as Succeeded. Highlight the **Succeeded** run and then select the **Results1** tab to see the list of customers.

Status	Operation	Date	Connection Profile	SSN	TITLE	FIRST_NAME	LAST_NAME
✓ Succeeded		Oct 5, 2010 9:46:18 PM	ITSOBANKderby	111-11-1111	Mr	Henry	Cui
				222-22-2222	Ms	Craig	Fleming
				333-33-3333	Mr	Rafael	Coutinho
				444-44-4444	Mr	Salvatore	Sollami
				555-55-5555	Mr	Brian	Hainey
				666-66-6666	Mr	Steve	Baber
				777-77-7777	Mr	Sundaragopal	Venkatraman
				888-88-8888	Mr	Lara	Ziosi
				999-99-9999	Mr	Sylvi	Lippmann
				000-00-0000	Mr	Venkata	Kumari
				000-00-1111	Mr	Martin	Keen

Figure 9-5 Sample contents of the Customer table

Editing, extracting, and loading options

When you right-click the CUSTOMER table, you have the following options:

- ▶ **Data** → **Edit**: Directly affects the contents of the target table in a spreadsheet-like interface
- ▶ **Data** → **Extract**: Extracts data into a file using a delimiter (comma, semicolon, space, tab, or vertical bar), for example:


```
"111-11-1111", "Mr", "Henry", "Cui"
"222-22-2222", "Ms", "Craig", "Fleming"
.....
```
- ▶ **Data** → **Load**: Loads data from a file that is similar to the files produced by the extract option

9.3 Creating SQL statements

You can create an SQL statement by using the SQL Builder or the SQL editor in the Data perspective.

The *SQL editor* supports any statements that can be run by the database to which you are connected. You can create single or multiple SQL statements, single or multiple XQuery statements, XQuery statements that are nested in SQL statements, and SQL statements that are nested in XQuery statements. The SQL editor provides features, such as multiple statement support, syntax highlighting, content assist, query parsing, validation, and SQL formatting.

The *SQL Builder* provides a graphical interface for creating and running SQL statements. Statements that are generated by the SQL Builder are saved in a file

with the extension `.sql`. The SQL Builder supports creating SELECT, INSERT, UPDATE, DELETE, FULLSELECT, and WITH (DB2 only) statements.

In this section, we create and run an SQL query to retrieve a customer name based on the Social Security number, and the total amount of money involved in each transaction type (credit or debit). The SQL select statement includes table aliases, table joins, a query condition, a column alias, a sort type, a database function expression, and a grouping clause.

9.3.1 Creating a Data Development project

Before you create routines or other database development objects, you must create a Data Development project to store your objects. A Data Development project is linked to one database connection in the Data Source Explorer.

A Data Development project is used to store routines and queries. You can store and develop the following types of objects in a database development project:

- ▶ SQL scripts
- ▶ DB2 and Derby stored procedures
- ▶ DB2 user-defined functions

You can also test, debug, export, and deploy these objects from a data development project. The wizards that are available in a Data Development project use the connection information that is specified for the project to help you develop objects that are targeted for that specific database.

To create a Data Development project, follow these steps:

1. In the Data perspective, Data Project Explorer, select **File** → **New** → **Data Development Project**. Alternatively, you can right-click in the Data Project Explorer and select **New** → **Data Development Project**.
2. In the New Data Development Project window, for the project name, type `RAD8DataDevelopment` and click **Next**.
3. In the Select Connection window (Figure 9-6 on page 403), from the Connections list, select **ITSOBANKderby**. Click **Finish**.

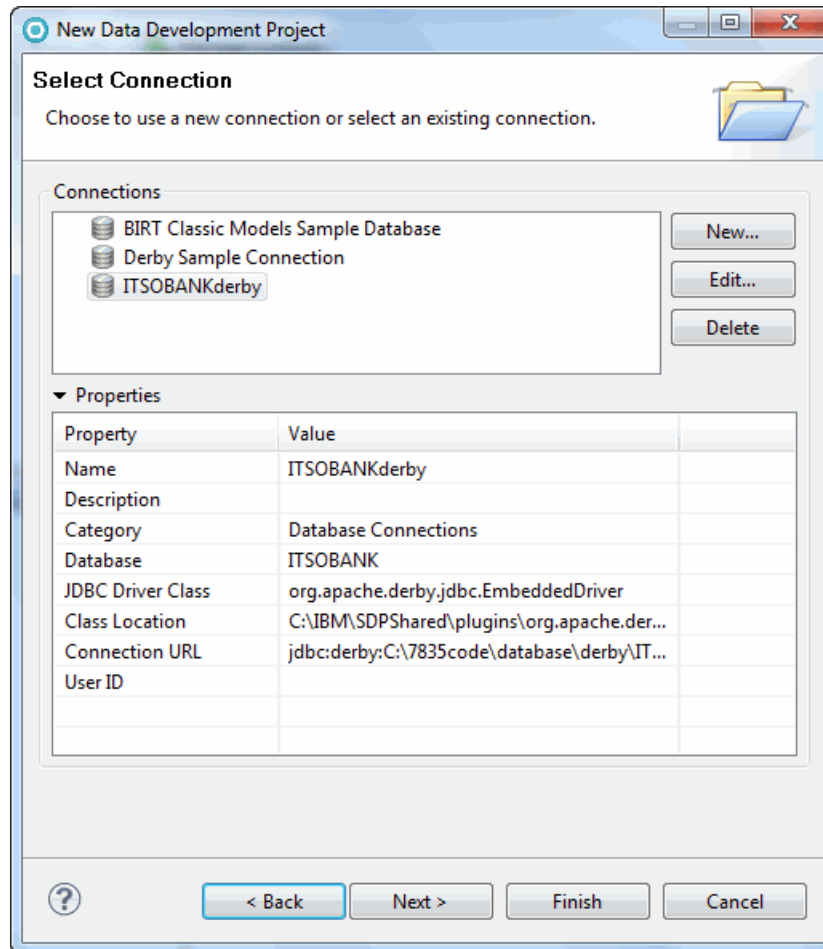


Figure 9-6 Data Development Project: Select Connection window

The Data Development project is displayed in the Data Project Explorer view.

9.3.2 Populating the transactions table

Before building the SQL query, populate the TRANSACT table with more data. Only one customer has transactions already. To load the records into this table, follow these steps:

1. In the Data Project Explorer, right-click the **RAD8DataDevelopment** project and select **Import**.
2. Expand **General** and select **File System**. Click **Next**.

3. Click **Browse** and locate the C:\7835code\database\samples directory. Select **LoadTransaction.sql** and click **Finish**. The LoadTransaction.sql file is displayed in the SQL Scripts folder.
4. Right-click **LoadTransaction.sql** and select **Run SQL**.

The results are shown in the SQL Results view. The status shows *Succeeded*.

9.3.3 Creating a select statement

To retrieve a customer name and the total amount of money involved in each transaction type (credit or debit), create a select statement:

1. In the Data Project Explorer view, in the **RAD8DataDevelopment** project, right-click the **SQL Scripts** folder and select **New** → **SQL or XQuery Script**.
2. In the Script Name and Editor window (Figure 9-7), for Name, type CustomerTransactions. For Edit using, select **SQL Query Builder**. For Statement type, select **SELECT**. Click **Finish**.

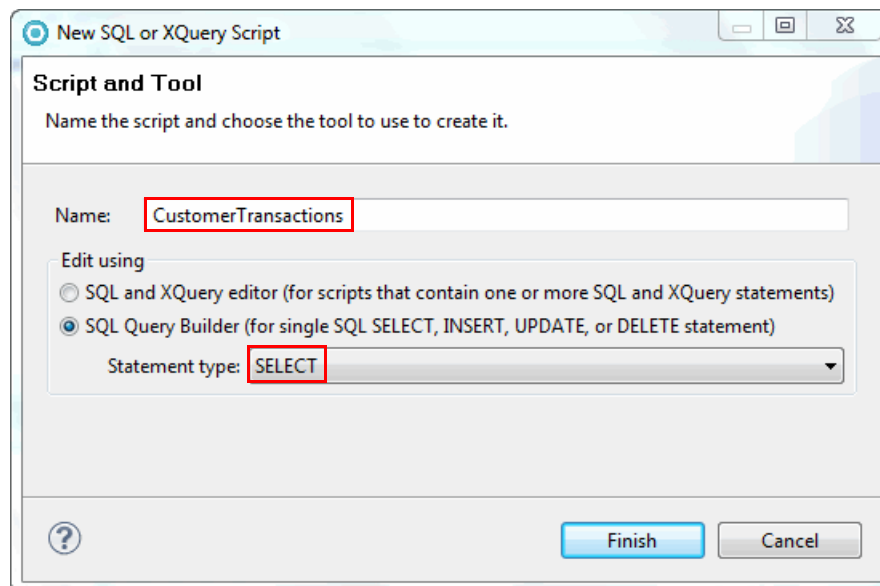


Figure 9-7 New SQL or XQuery Script editor window

The SELECT statement template is created and opens in the SQL Builder (Figure 9-8 on page 405).

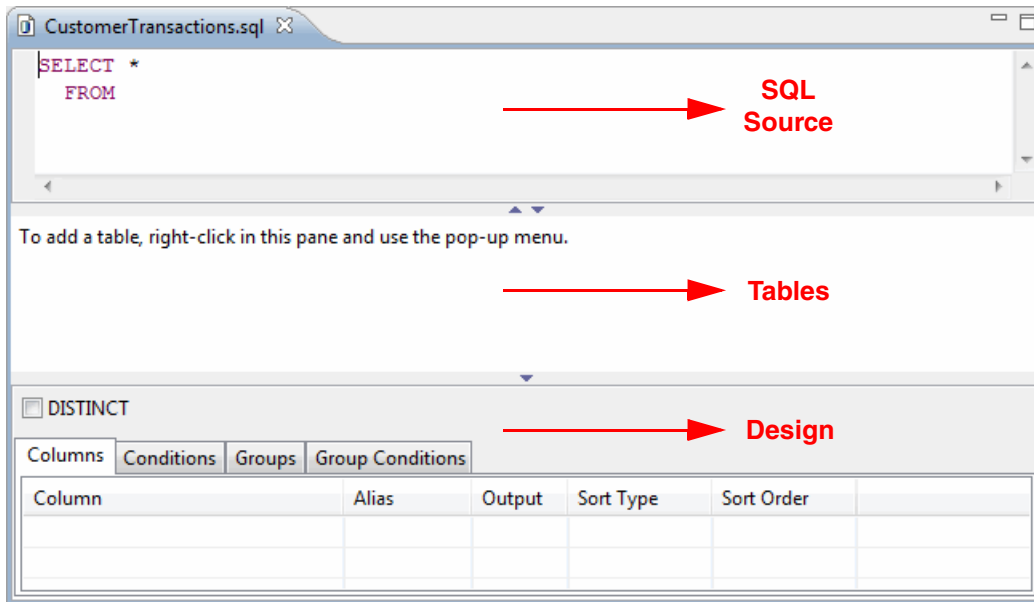


Figure 9-8 SQL Builder

Using the SQL Builder

The SQL Builder has the following sections:

- ▶ **SQL source pane:** The SQL Source pane contains the source code of the SQL statement. You can type the SQL statement in this pane, or use the features that are provided by the SQL Builder to build the statement. Content assist is available as you type and also through the pop-up menu in the SQL source pane. This pane provides content tips through the pop-up menu. A content tip shows a simple example for the type of statement that you are creating.
- ▶ **Tables pane:** The Tables pane provides a graphical representation of the table references that are used in the statement. In this pane, you can add or remove a table, give a table an alias, and include or exclude columns from the table. When you build a SELECT statement, you can also define joins between tables in this pane.
- ▶ **Design pane:** The options in the Design pane vary, depending on the type of statement that you are creating. When multiple sets of options are available, the options appear as notebook pages. For example, for a SELECT statement, the options include selecting columns, creating conditions, creating groups, and creating group conditions.

Adding tables to the statement

We add four tables to the SELECT statement for the CustomerTransactions query, because this query traverses from CUSTOMER through ACCOUNT and TRANSACT. We also create an alias for each of the tables in the SELECT statement. An *alias* is an indirect method of referencing a table so that an SQL statement can be independent of the qualified name of that table. If the table name changes, only the alias definition must be changed.

Table aliases: The aliases for the ACCOUNT, CUSTOMER, TRANSACT, and ACCOUNT_CUSTOMER tables are A, C, T, and AC respectively.

Perform these steps to add tables to the statement:

1. In the Data Source Explorer, expand **ITSOBANKderby** → **ITSOBANK** → **Schemas** → **ITSO** → **Tables**. You can see the four tables.
2. Right-click in the Tables pane and then click **Add Table**.
3. In the Table name list, expand the **ITSO** schema and select **CUSTOMER**. In the Add Table window (Figure 9-9), for the Table alias, type C and click **OK**. The CUSTOMER table is shown in the Tables pane, and the source code in the SQL Source pane shows the addition of the CUSTOMER table in the SELECT statement (Figure 9-10 on page 407).

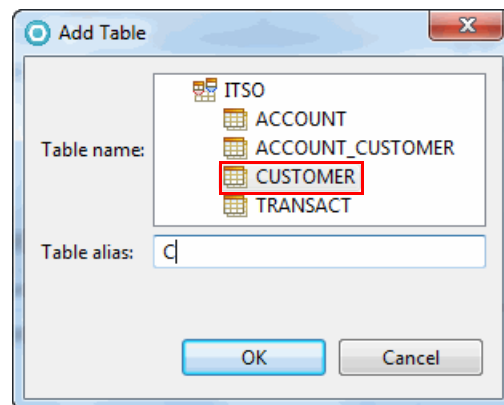


Figure 9-9 Add Table window

4. Follow the same procedure to add the ACCOUNT_CUSTOMER (alias AC), ACCOUNT (alias A), and TRANSACT (alias T) tables to the Tables pane in the SQL Builder.

5. Select a table and drag the sides to adjust the size of the displayed rectangle (Figure 9-10).

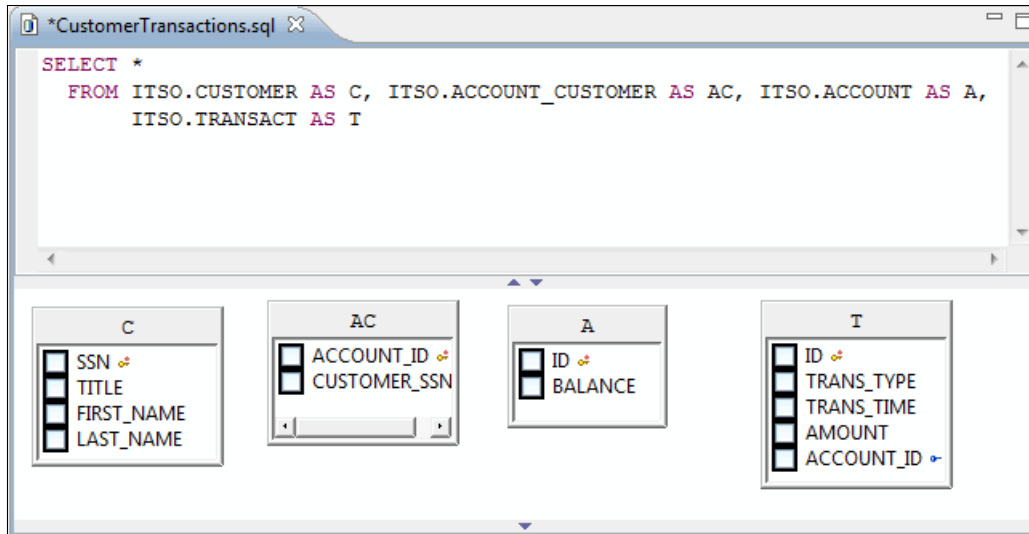


Figure 9-10 Creating a SELECT statement: Tables

Selecting columns for the result set

Add the following columns to the result set by selecting the columns in the Tables pane:

1. Select the **FIRST_NAME** and **LAST_NAME** columns in the C (CUSTOMER) table.
2. Select the **TRANS_TYPE** column in the T (TRANSACT) table.

Joining tables

You can use a join operation to retrieve data from two or more tables based on matching column values. Three joins are needed for this query:

1. Drag the cursor from the **SSN** column in the C (CUSTOMER) table to the **CUSTOMER_SSN** column in the AC (ACCOUNT_CUSTOMER) table.
2. Drag the cursor from **ACCOUNT_ID** in the AC (ACCOUNT_CUSTOMER) table to **ID** in the A (ACCOUNT) table.
3. Drag the cursor from **ID** in the A (ACCOUNT) table to **ACCOUNT_ID** in the T (TRANSACT) table.

Figure 9-11 shows the relationship lines that are drawn between the selected columns.

The screenshot shows a SQL editor window titled "*CustomerTransactions.sql". The SQL statement is:

```
SELECT C.FIRST_NAME, C.LAST_NAME, T.TRANS_TYPE
FROM
  ITSO.CUSTOMER AS C JOIN ITSO.ACCOUNT_CUSTOMER AS AC ON C.SSN = AC.CUSTOMER_SSN
  JOIN ITSO.ACCOUNT AS A ON AC.ACCOUNT_ID = A.ID
  JOIN ITSO.TRANSACTIONS AS T ON A.ID = T.ACCOUNT_ID
```

Below the SQL editor is a diagram showing the relationships between four tables: C, AC, A, and T. The columns and their relationships are as follows:

- Table C:** SSN, TITLE, FIRST_NAME, LAST_NAME. SSN is the primary key. FIRST_NAME and LAST_NAME are selected (checked).
- Table AC:** ACCOUNT_ID, CUSTOMER_SSN. ACCOUNT_ID is the primary key. CUSTOMER_SSN is a foreign key to C.SSN.
- Table A:** ID, BALANCE. ID is the primary key.
- Table T:** ID, TRANS_TYPE, TRANS_TIME, AMOUNT, ACCOUNT_ID. ID is the primary key. ACCOUNT_ID is a foreign key to A.ID. TRANS_TYPE is selected (checked).

Relationships are shown as double-headed arrows with crow's foot notation. C is linked to AC, AC is linked to A, and A is linked to T.

Below the diagram is a table with the following columns: Column, Alias, Output, Sort Type, Sort Order. The table contains the following rows:

Column	Alias	Output	Sort Type	Sort Order
C.FIRST_NAME		<input checked="" type="checkbox"/>		
C.LAST_NAME		<input checked="" type="checkbox"/>		
T.TRANS_TYPE		<input checked="" type="checkbox"/>		

Figure 9-11 Creating a SELECT statement: Columns and table joins

Adding a function expression to the result set

The fourth column for the query result set is the result of a column expression. In the following steps, you add the total amount of each transaction type, which can be calculated by using the Expression Builder wizard:

1. In the Columns tab of the Design pane, click the fourth cell (the first empty cell) in the Column column and select **Build Expression** from the drop-down list. Press Enter.
2. In the Expression Builder wizard (Figure 9-12), select **Function** and click **Next**.



Figure 9-12 Creating a SELECT statement: Expression Types window

3. In the Function Builder page, complete the following tasks (Figure 9-13):
 - a. For Select a function category, select **Aggregate**.
 - b. For Select a function, select **SUM**.
 - c. For Select a function signature, select **SUM(expression) → expression**.
 - d. In the Value column of the argument table, click the cell and select **T.AMOUNT** in the drop-down list. The preview of the function expression is displayed as SUM(AMOUNT).
 - e. Click **Finish**.

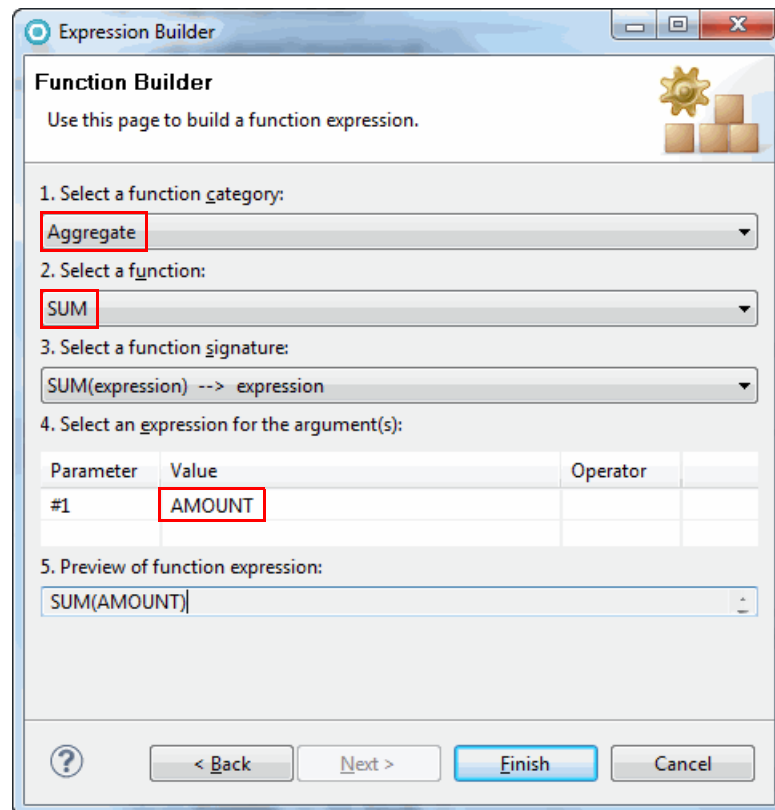


Figure 9-13 Creating a SELECT statement: Function Builder window

Adding a column alias and sort type

Add a column alias for the function column expression and sort the results:

1. In the Design pane, click the **Columns** tab.
2. In the Alias column, click the cell next to the SUM(T.AMOUNT) expression, type TotalAmount, and press Enter.

- In the Sort Type column, click the cell next to the TotalAmount alias, select **Ascending**, and press Enter.

Figure 9-14 shows the Columns page.

Column	Alias	Output	Sort Type	Sort Order
C.FIRST_NAME		<input checked="" type="checkbox"/>		
C.LAST_NAME		<input checked="" type="checkbox"/>		
T.TRANS_TYPE		<input checked="" type="checkbox"/>		
SUM(T.AMOUNT)	TotalAmount	<input checked="" type="checkbox"/>	Ascending	1

Figure 9-14 Creating a SELECT statement: Columns page

Creating a query condition

The query needs a condition so that it extracts only result rows with a given customer Social Security number. Add conditions to the query by using the Conditions page in the Design pane.

Follow these steps to create a query condition:

- In the Design pane, select the **Group Conditions** tab.
- In the first row, click the cell in the **Column** column and select **C.LAST_NAME** in the list.
- In the same row, click the cell in the **Operator** column and select the **LIKE** operator.
- In the same row, click the cell in the **Value** column and enter %u%.

A colon followed by a variable name is the SQL syntax for a host variable that will be substituted with a value when you run the query.

Figure 9-15 shows the Conditions page.

Column	Operator	Value	AND/OR
C.LAST_NAME	LIKE	'%u%'	

Figure 9-15 Creating a SELECT statement: Conditions page

Adding a GROUP BY clause

Group the query by the transaction type so that you have one sum of the amount for each type of transaction (credit or debit):

1. In the Design pane, select the **Groups** tab.
2. In the Column table, click the first row, select **T.TRANS_TYPE** in the list, and press Enter.
3. Repeat these steps for **C.FIRST_NAME** and **C.LAST_NAME**.

Figure 9-16 shows the Groups page.

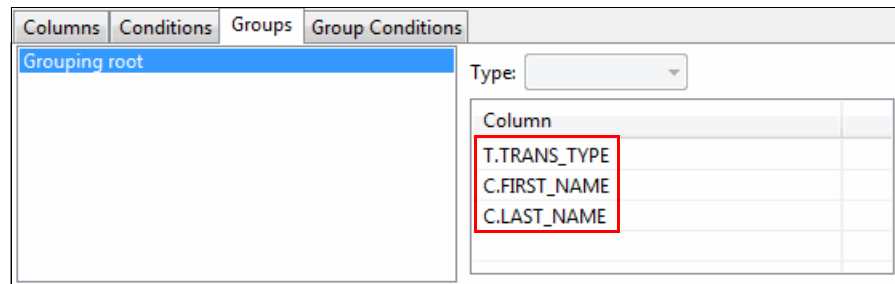


Figure 9-16 Creating a SELECT statement: Groups page

The query is now complete. Save the SELECT statement. Example 9-1 shows the SQL statement.

Example 9-1 CustomerTransactions.sql

```
SELECT C.FIRST_NAME, C.LAST_NAME, T.TRANS_TYPE, SUM(T.AMOUNT) AS
"TotalAmount"
FROM
    ITSO.CUSTOMER AS C JOIN ITSO.ACCOUNT_CUSTOMER AS AC ON C.SSN =
    AC.CUSTOMER_SSN JOIN ITSO.ACCOUNT AS A ON AC.ACCOUNT_ID = A.ID
JOIN
    ITSO.TRANSACT AS T ON A.ID = T.ACCOUNT_ID
WHERE C.LAST_NAME LIKE '%u%'
GROUP BY T.TRANS_TYPE, C.FIRST_NAME, C.LAST_NAME
ORDER BY "TotalAmount" ASC
```

9.3.4 Running the SQL query

To run the SQL query, in the Data Project Explorer, right-click **CustomerTransactions.sql** and select **Run SQL**. Click **Finish**.

Figure 9-17 shows the result. You can see that the total amount of money for each transaction type is calculated and the results are ordered by TotalAmount in ascending order.

Status	Operation	Date	Connection Profile	FIRST_NAME	LAST_NAME	TRANS_TYPE	TotalAmount
✓ Succeeded	LoadTransaction...	Oct 6, 2010 6:...	ITSOBANKderby	Venkata	Kumari	Credit	9999.99
✓ Succeeded	SELECT C.FIRST_...	Oct 6, 2010 6:...	ITSOBANKderby	Henry	Cui	Debit	368178.00
				Henry	Cui	Credit	2161003.00

Figure 9-17 Query results

9.4 Developing Java stored procedures

A *stored procedure* is a block of procedural constructs and embedded SQL statements that are stored in a database and can be called by name. Stored procedures can improve application performance and reduce database access traffic. All database access must go across the network, which, in certain cases, can result in poor performance. For each SQL statement, a database manager application must initiate a separate communication with the database.

To improve application performance, you can create stored procedures that run on a database server. A client application can then call the stored procedures to obtain the results of the SQL statements that are in the procedure. Because the stored procedure runs the SQL statements on the server for you, database performance is improved.

Stored procedures can be written as SQL procedures, or as C, COBOL, PL/I, or Java programs. In this section, we develop a Java stored procedure against the ITSOBANK Derby database to obtain the account information based on a partial customer last name. While doing so, we give a credit of \$100 to every account retrieved.

9.4.1 Creating a Java stored procedure

To create a stored procedure using the Stored Procedure wizard, follow these steps:

1. In the Data Project Explorer view, expand the **RAD8DataDevelopment** project, right-click the **Stored Procedures** folder, and select **New** → **Stored Procedure**.

2. Perform these steps in the New Stored Procedure wizard, in the Name and Language window (Figure 9-18):
 - a. Notice that **RAD8DataDevelopment** is preselected.
 - b. Type **AddCredit** for the Name.
 - c. **Java** is shown as the language.
 - d. For Java package, type **itso.bank.data**.
 - e. Select **Dynamic SQL using JDBC** and click **Next**.

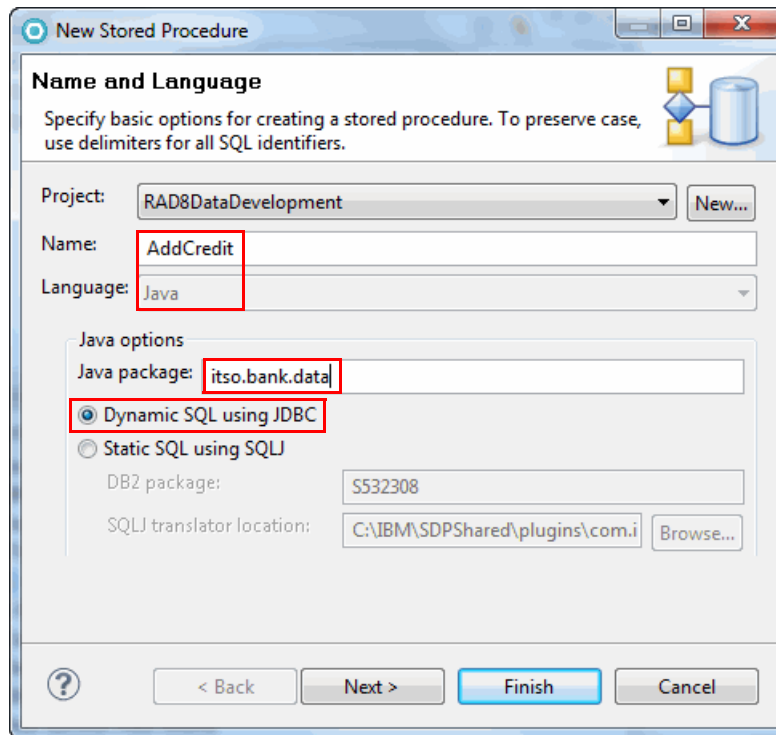


Figure 9-18 Creating a new stored procedure: Name and Language window

3. On the SQL Statements page, click **Create SQL** to start the New SQL Statement wizard that guides you through the creation of an SQL statement.
4. In the first window of the New SQL Statement wizard, keep the defaults to create a SELECT statement using the wizard and click **Next**.
5. To create the SQL statement, perform these steps on the New SQL Statement: Construct an SQL Statement window (Figure 9-19 on page 415):
 - a. On the **Tables** page, in the Available Tables list, expand the **ITSO** schema, select **ITSO.ACCOUNT**, **ITSO.ACCOUNT_CUSTOMER**, and select

ITSO.CUSTOMER. Click > to move the three tables to the Selected Tables list.

- b. Select the **Columns** tab. Expand the **CUSTOMER** table and select **FIRST_NAME** and **LAST_NAME**. Expand the **ACCOUNT** table and select **BALANCE**. Click > to move the columns to the Selected Columns list.
- c. Select the **Joins** tab. Drag the cursor from **SSN** (CUSTOMER table) to **CUSTOMER_SSN** (ACCOUNT_CUSTOMER table) and from **ID** (ACCOUNT table) to **ACCOUNT_ID** (ACCOUNT_CUSTOMER table) to create two joins.

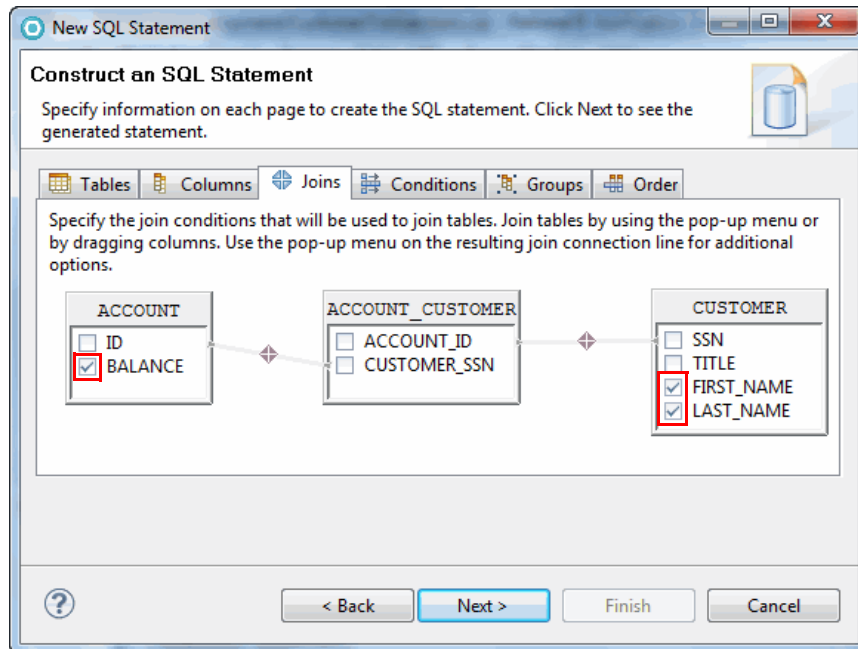


Figure 9-19 Creating a stored procedure: Joins tab

- d. Select the **Conditions** tab (Figure 9-20). In the first row, under Column, click the cell and select **CUSTOMER.LASTNAME**. In the same row, for Operator, select **LIKE**, and for Value, type **PARTIALNAME**. Click **Next**.

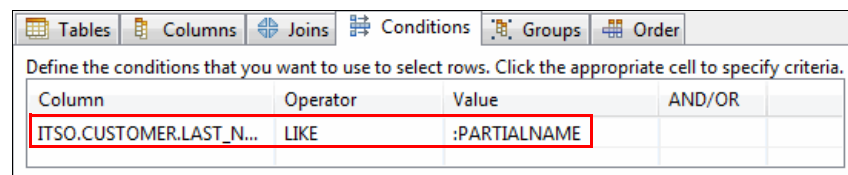


Figure 9-20 Creating a stored procedure: Conditions tab

6. In the Change the SQL Statement window, review the generated SQL statement and click **Finish** to close the New SQL Statement wizard.
7. Back in the New Stored Procedure wizard, for Result set, select **One** and click **Next**.
8. The Parameters window displays an input variable named PARTIALNAME. Accept the default settings and click **Next**.
9. In the Deploy Options window, clear **Deploy on Finish**. We deploy the stored procedure later. Click **Next**.
10. In the Code Fragments window, click **Next**.
11. Review the Summary page and click **Finish**. The stored procedure opens in the routine editor.
12. Select the **Configuration** tab in the routine editor. In the Java section, click **ADDCREDIT.java**. The generated file (Example 9-2) opens.

Example 9-2 ADDCREDIT.java

```
package itso.bank.data;

import java.sql.*; // JDBC classes

public class ADDCREDIT {
    public static void addCREDIT(java.lang.String PARTIALNAME,
        ResultSet[] rs1)
        throws SQLException, Exception {
        // Get connection to the database
        Connection con =
        DriverManager.getConnection("jdbc:default:connection");
        PreparedStatement stmt = null;
        boolean bFlag;
        String sql;

        sql = "SELECT ITSO.CUSTOMER.FIRST_NAME,
        ITSO.CUSTOMER.LAST_NAME, ITSO.ACCOUNT.BALANCE"
            + " FROM"
            + "          ITSO.ACCOUNT JOIN ITSO.CUSTOMER JOIN
        ITSO.ACCOUNT_CUSTOMER ON ITSO.CUSTOMER.SSN =
        ITSO.ACCOUNT_CUSTOMER.CUSTOMER_SSN ON ITSO.ACCOUNT.ID =
        ITSO.ACCOUNT_CUSTOMER.ACCOUNT_ID"
            + " WHERE ITSO.CUSTOMER.LAST_NAME LIKE ?";
        stmt = con.prepareStatement(sql);
        stmt.setString(1, PARTIALNAME);
        bFlag = stmt.execute();
        rs1[0] = stmt.getResultSet();
    }
}
```

```
}  
}
```

13. We give a \$100 credit to the selected accounts. Add the code in Example 9-3 under the `rs1[0] = stmt.getResultSet()` statement.

Example 9-3 Snippet to give \$100 credit to each account

```
String sql2 = "UPDATE ITSO.ACCOUNT SET BALANCE = (BALANCE +  
100)"  
    + " WHERE ID IN " +  
    "(SELECT ITSO.ACCOUNT.ID FROM ITSO.ACCOUNT"  
    + " JOIN ITSO.ACCOUNT_CUSTOMER"  
    + " ON ITSO.ACCOUNT.ID =  
ITSO.ACCOUNT_CUSTOMER.ACCOUNT_ID"  
    + " JOIN ITSO.CUSTOMER ON  
ITSO.ACCOUNT_CUSTOMER.CUSTOMER_SSN ="  
    + " ITSO.CUSTOMER.SSN"  
    + " WHERE ITSO.CUSTOMER.LAST_NAME LIKE ?)";  
stmt = con.prepareStatement(sql2);  
stmt.setString( 1, PARTIALNAME );  
stmt.executeUpdate();
```

You can also find the modified `AddCredit.java` file in the `C:\7835code\database\samples` directory.

9.4.2 Deploying a Java stored procedure

A stored procedure must be deployed to the database where it is stored in the catalog, ready for execution. Follow these steps to deploy a Java stored procedure to a database:

1. In the Data Project Explorer, expand **RAD8DataDevelopment** → **Stored Procedures**, right-click **ADDCREDIT**, and select **Deploy**.
2. In the Deploy Routines wizard (Figure 9-21 on page 418), for Target schema, type **ITSO** and click **Finish**.

You see a “Succeeded” build status in the SQL Results view.

3. In the Data Source Explorer, expand **ITSOBANK** → **Schemas** → **ITSO**, right-click **Stored Procedures**, and select **Refresh**. You can see that **ADDCREDIT** has been added to the Stored Procedures folder.

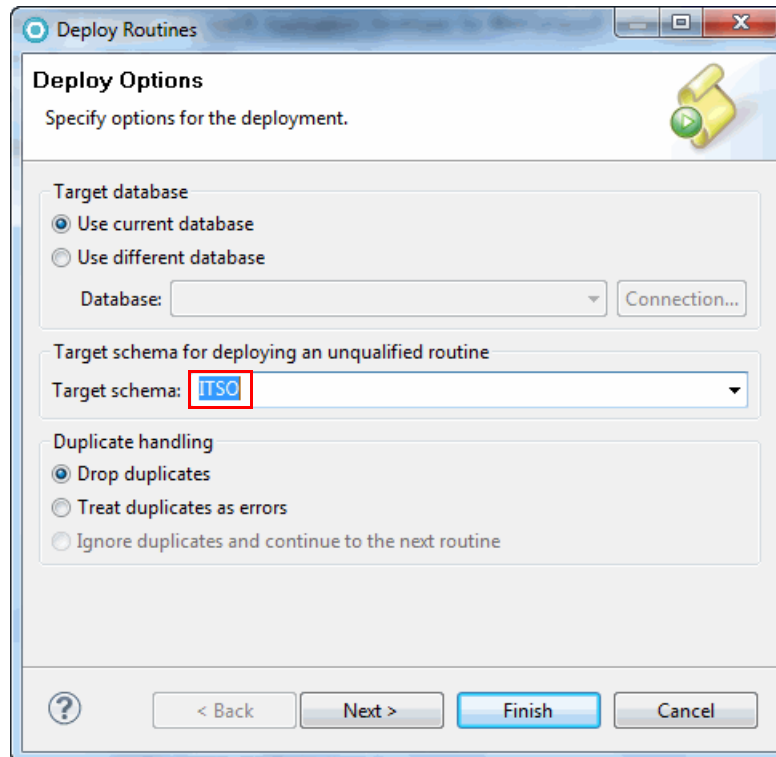


Figure 9-21 Deploy Routines

9.4.3 Running the stored procedure

Rational Application Developer provides a test facility for testing the Java stored procedures. Perform these steps to run the stored procedure:

1. In the Data Project Explorer, right-click the stored procedure **ADDCREDIT** and select **Run**.
2. In the Specify Parameter Values window, in the Value field, type `C%` in the cell and press Enter. `PARTIALNAME LIKE 'C%'` retrieves only customers with a last name that starts with C.
3. Click **OK**.

Figure 9-22 on page 419 shows the result. You now see that \$100 has been added to the related accounts and that the balances are updated.

Status	Operation	Date	Connection	Parameter	Result1	
				FIRST_NAME	LAST_NAME	BALANCE
✓ Succeeded	Deploy ITSO.AD...	Oct 7, 2010 6:...	ITSOBANKde	1 Henry	Cui	12545.67
✓ Succeeded	Run ITSO.ADDC...	Oct 7, 2010 6:...	ITSOBANKde	2 Henry	Cui	6743.21
				3 Henry	Cui	298.76
				4 Rafael	Coutinho	10076.52
				5 Rafael	Coutinho	768.79
				6 Rafael	Coutinho	221.56

Total 6 records shown

Figure 9-22 Stored procedure results

9.5 Developing SQLJ applications

With SQLJ, you can embed SQL statements into Java programs. SQLJ is an ANSI standard developed by a consortium of leading providers of database and application server software.

The SQLJ translator translates an SQLJ source file into a standard Java source file plus an SQLJ serialized profile that encapsulates information about static SQL in the SQLJ source. The translator converts SQLJ clauses to standard Java statements by replacing the embedded SQL statements with calls to the SQLJ runtime library. An SQLJ customization script binds the SQLJ profile to the database, producing one or more database packages. The Java file is compiled and run (with the packages) on the database. The SQLJ runtime environment consists of an SQLJ runtime library that is implemented in pure Java. The SQLJ runtime library calls the JDBC driver for the target database.

SQLJ provides better performance by using static SQL. SQLJ generally requires fewer lines of code than JDBC to perform the same tasks. The SQLJ translator checks the syntax of SQL statements during translation. SQLJ uses database connections to type-check static SQL code. With SQLJ, you can embed Java variables in SQL statements. SQLJ provides strong typing of query output and return parameters and allows type-checking on calls. SQLJ provides static package-level security with compile-time encapsulation of database authorization.

Using the SQLJ wizard that ships with Rational Application Developer, you can perform the following actions:

- ▶ Name an SQLJ file and specify its package and source folder.

- ▶ Specify advanced project properties, such as additional JAR files, to add to the project class path, translation options, and whether to use long package names.
- ▶ Select an existing SQL SELECT statement, or construct and test a new SQL SELECT statement.
- ▶ Specify information for connecting to the database at run time.

In this section, we create an SQLJ application to retrieve the customer and the associated account information.

9.5.1 Creating SQLJ files

You can create SQLJ files by using the New SQLJ File wizard. The SQLJ support is automatically added to the project when you use this wizard.

Create a Java project named RAD8SQLJ and then create the SQLJ file in this project:

1. Open the Java perspective, select **File** → **New** → **Java Project**. For the Project name, type RAD8SQLJ and click **Finish**.
2. Select **File** → **New** → **Other** → **Data** → **SQLJ Applications** → **SQLJ File** and click **Next**.
3. In the SQLJ File window (Figure 9-23), for Package, enter `itso.bank.data.sqlj`. For Name, enter `CustomerAccountInfo`. Then click **Next**.

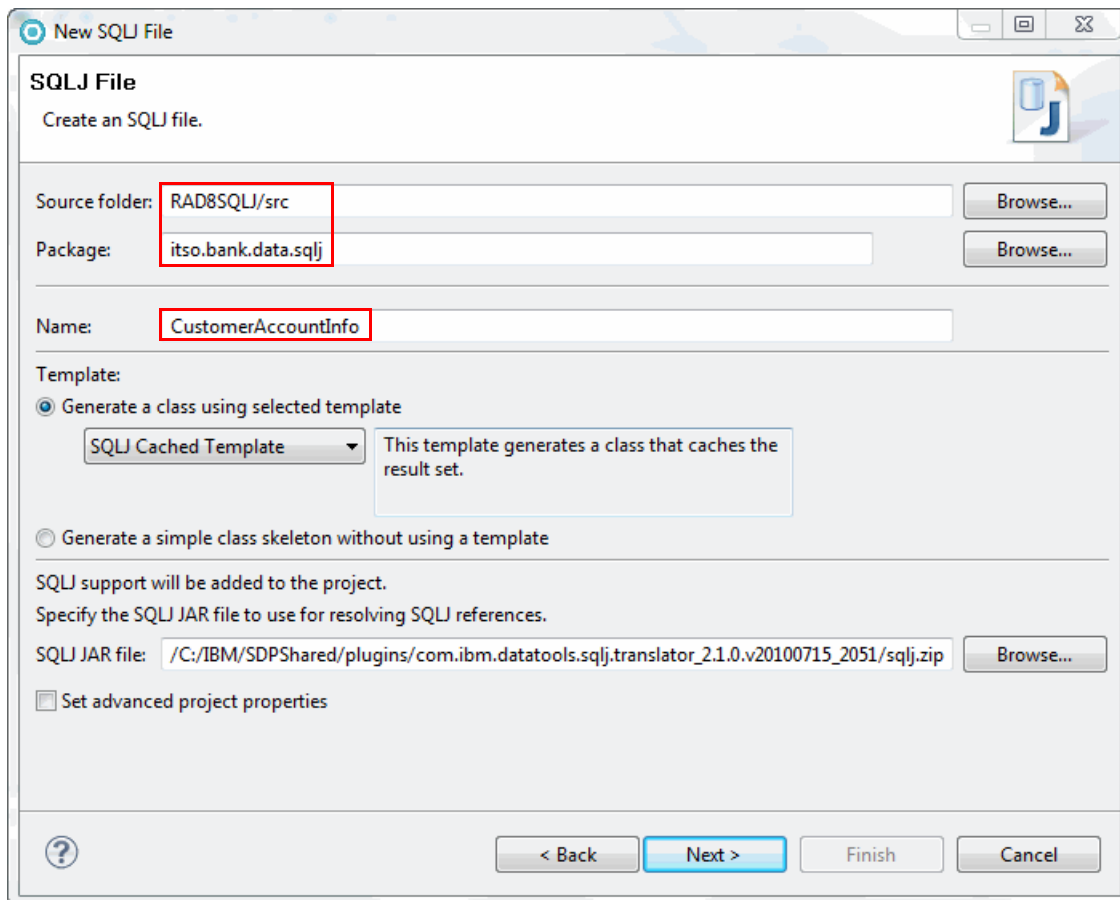


Figure 9-23 New SQLJ File window

4. In the Select an Existing Statement Saved in Your Workspace window, click **Next**. We create a new SQL statement.
5. In the Specify SQL Statement Information window, select **Be guided through creating an SQL statement** and click **Next**.
6. In the Select Connection window, select the **ITSOBANKderby** connection that you created in the previous section. Click **Reconnect** to reconnect to the database if it is disconnected. Click **Next**.
7. In the Construct an SQL Statement window, follow these steps:
 - a. On the **Tables** tab, for Available Tables, expand the **ITSO** schema and select the **CUSTOMER**, **ACCOUNT**, and **ACCOUNT_CUSTOMER** tables. Click > to move these three tables to the Selected Tables list.
 - b. Select the **Columns** tab. In the Available columns list, under the CUSTOMER table, select **TITLE**, **FIRST_NAME**, and **LAST_NAME**. Under the ACCOUNT table, select **ID** and **BALANCE**. Then click > to move these columns to the selected Columns list (Figure 9-24).

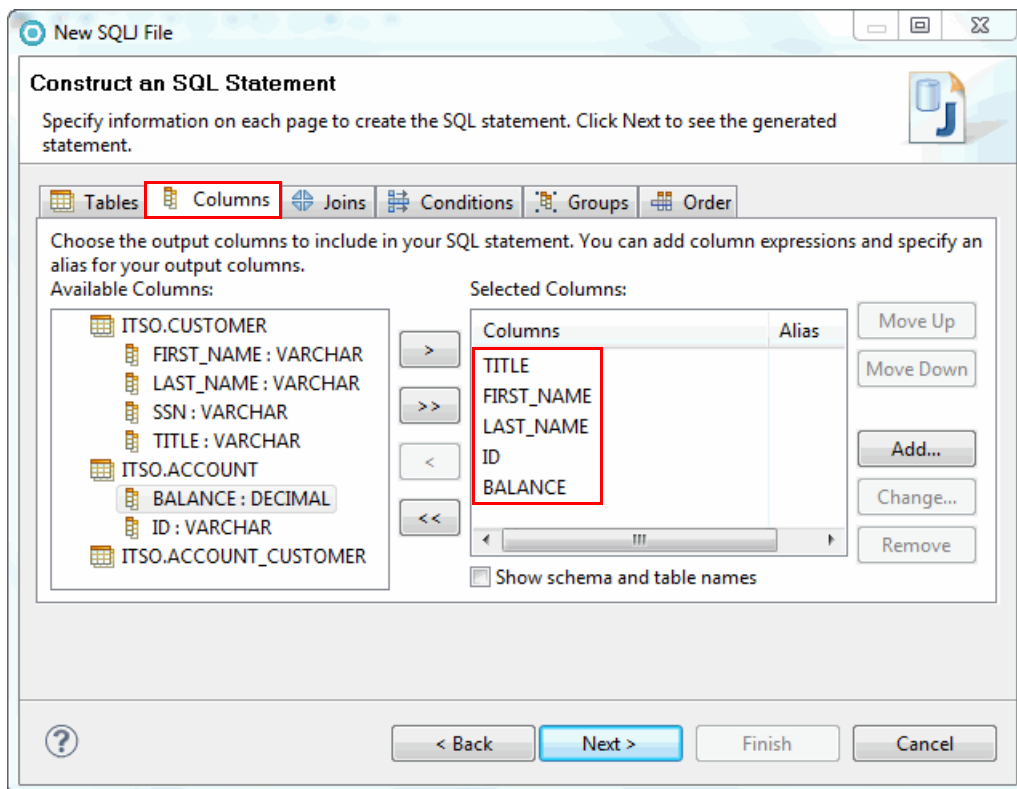


Figure 9-24 Selecting the output columns

- c. Select the **Joins** tab (Figure 9-19 on page 415). Drag the cursor from **CUSTOMER.SSN** to **CUSTOMER_SSN** and from **ACCOUNT.ID** to **ACCOUNT_ID**.
 - d. Select the **Conditions** tab. In the first row, click the cell in the Column and select **ACCOUNT.BALANCE**. In the same row, for Operator, select **>=**, and for Value, type **:BALANCE**.
 - e. Select the **Order** tab. Under the **ACCOUNT** table, select **BALANCE** and click **>**. For Sort order, select **DESC**. The results are listed with the highest balance first.
 - f. Click **Next**.
8. In the Change the SQL Statement window, review the generated SQL statement:

```

SELECT ITSO.CUSTOMER.TITLE, ITSO.CUSTOMER.FIRST_NAME,
ITSO.CUSTOMER.LAST_NAME, ITSO.ACCOUNT.ID, ITSO.ACCOUNT.BALANCE
  FROM ITSO.CUSTOMER JOIN ITSO.ACCOUNT_CUSTOMER ON ITSO.CUSTOMER.SSN
=
      ITSO.ACCOUNT_CUSTOMER.CUSTOMER_SSN JOIN ITSO.ACCOUNT ON
      ITSO.ACCOUNT_CUSTOMER.ACCOUNT_ID = ITSO.ACCOUNT.ID
WHERE ITSO.ACCOUNT.BALANCE >= :BALANCE
ORDER BY BALANCE DESC

```

Click **Parse** and then click **Next**.

9. In the Specify Runtime Database Connection Information window (Figure 9-25), select **Use DriverManager connection**. Derby does not use authentication. Select **Variables inside of method**. Leave the user ID as **its0** and do not enter a password. Click **Finish**.

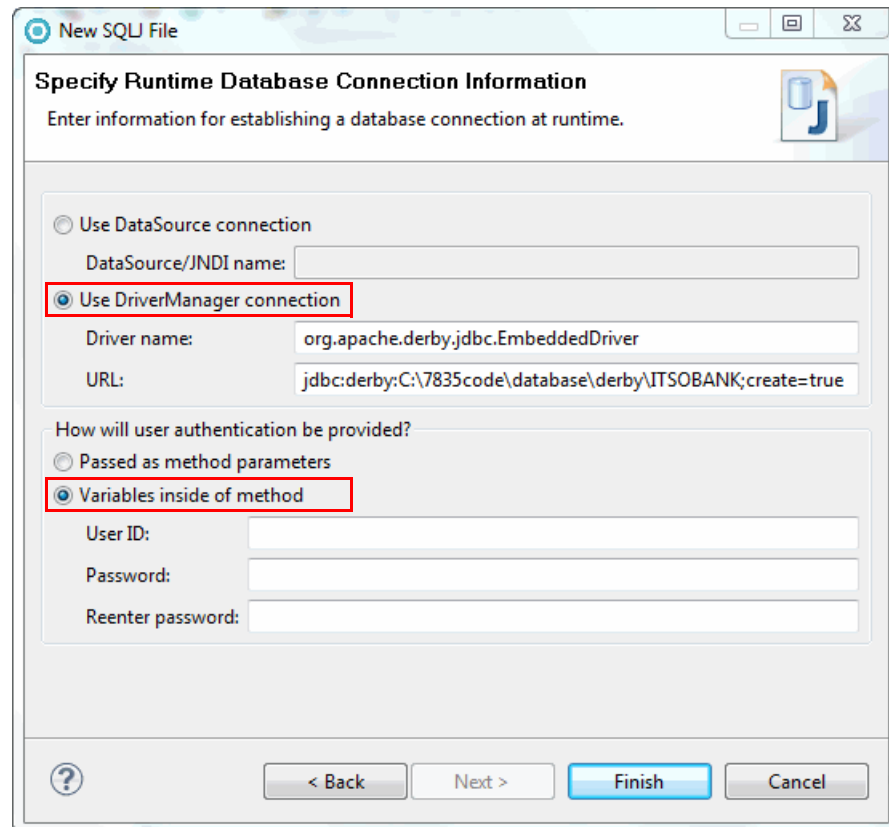


Figure 9-25 Specify Runtime Database Connection Information window

The SQLJ file is generated.

9.5.2 Examining the generated SQLJ file

Before testing the SQLJ program, we examine the generated SQLJ file:

- ▶ The establishConnection method creates the database connection.
- ▶ The execute method executes the SQL query and stores the result set in a cache. Example 9-4 on page 425 shows the SQLJ statement that is embedded in this method.

Example 9-4 Embedded SQLJ

```
#sql [ctx] cursor1 = {SELECT ITSO.CUSTOMER.TITLE,  
    ITSO.CUSTOMER.FIRST_NAME, ITSO.CUSTOMER.LAST_NAME,  
    ITSO.ACCOUNT.ID,  
    ITSO.ACCOUNT.BALANCE FROM ITSO.CUSTOMER JOIN  
    ITSO.ACCOUNT_CUSTOMER  
    ON ITSO.CUSTOMER.SSN = ITSO.ACCOUNT_CUSTOMER.CUSTOMER_SSN  
JOIN  
    ITSO.ACCOUNT ON ITSO.ACCOUNT_CUSTOMER.ACCOUNT_ID =  
    ITSO.ACCOUNT.ID  
    WHERE ITSO.ACCOUNT.BALANCE >= :BALANCE ORDER BY BALANCE  
DESC};
```

- ▶ The next method moves to the next row of the result set if another row exists.
- ▶ The close method commits changes and closes the connection.
- ▶ The corresponding setter and getter methods for the table fields in the database are also generated. You can use the getter methods to retrieve the columns in a row.

9.5.3 Testing the SQLJ program

To create a test program to invoke the SQLJ program, follow these steps:

1. In the Package Explorer, right-click the package **itso.bank.data.sqlj** and select **New** → **Class**.
2. For Class name, type **TestSQLJ**. Select **public static void main(String[] args)** and click **Finish**.
3. Copy and paste the code that is shown in Example 9-5 to **TestSQLJ**. You can find the **TestSQLJ.java** code in **C:\7835code\database\samples**.

Example 9-5 TestSQLJ.java

```
package itso.bank.data.sqlj;  
import java.math.BigDecimal;  
  
public class TestSQLJ {  
  
    public static void main(String[] args) {  
        try {  
            CustomerAccountInfo info = new CustomerAccountInfo();  
            info.execute(new BigDecimal(10000)); // minimum balance  
displayed  
            while (info.next()) {
```

```

        System.out.println("Customer name: " +
info.getCUSTOMER_TITLE()
        + " " + info.getCUSTOMER_FIRST_NAME() + " "
        + info.getCUSTOMER_LAST_NAME());
        System.out.println("Account ID: " +
info.getACCOUNT_ID() +
        " Balance: " +
info.getACCOUNT_BALANCE());

System.out.println("-----");
    }
    info.close();
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

4. Add the Derby JDBC driver library to the project build path:
 - a. Right-click project **RAD8SQLJ** and select **Properties**.
 - b. On the **Java Build Path, Libraries** tab, click **Add External JARs** and select **derby.jar**, which is in `<WebSphere Application Server v8_HOME>\derby\lib`. Click **OK**.

Derby accepts only one database connection at a time.
5. Switch to the Data perspective. In the Data Source Explorer, right-click the **ITSOBANK** connection and select **Disconnect**.
6. In the Java perspective, right-click **TestSQLJ.java** and select **Run As** → **Java Application**.

Example 9-6 shows how the result is displayed in the console.

Example 9-6 Result as displayed in the console

```

Retrieve some data from the database.
Customer name: Mr Brian Hainey
Account ID: 005-555002 Balance: 72213.41
-----
Customer name: Mr Venkata Kumari
Account ID: 000-000001 Balance: 66666.66
-----
Customer name: Ms Craig Fleming
Account ID: 002-222001 Balance: 65484.23
-----
Customer name: Mr Salvatore Sollami

```

```
Account ID: 004-444003 Balance: 23156.46
-----
Customer name: Mr Henry Cui
Account ID: 001-111001 Balance: 12645.67
-----
Customer name: Mr Rafael Coutinho
Account ID: 003-333001 Balance: 10176.52
-----
Customer name: Mr Steve Baber
Account ID: 006-666003 Balance: 10000.00
-----
```

Error message: The following exception message means that the Derby database is locked by another connection:

```
Failed to start database 'C:/7835code/database/derby/ITSOBANK'
```

In the Data Source Explorer, check whether the ITSOBANK connection is still active. If it is active, disconnect the connection, or restart the workbench.

9.6 Data modeling

Rational Application Developer provides tools to create, modify, and generate Data Definition Language (DDL) for data models. At any time when you are building a data model, you can analyze the model to verify that it is compliant with the defined constraints. If you make changes to the data model, Rational Application Developer provides tooling to compare the changed data model with the original data model. You can also perform an impact analysis to determine how the changes might affect other objects.

A physical data model is a database-specific model that represents relational data objects (for example, tables, columns, primary keys, and foreign keys) and their relationships. A physical data model can be used to generate DDL statements, which can then be deployed to a database server.

In the workbench, you can create and modify data models by using the Data Project Explorer, the Properties view, or a diagram of the model. You can also analyze models and generate DDL.

In this section, we create the physical model from a template, create tables using the data diagram, and deploy the physical model to the database. This section includes the following tasks:

- ▶ Creating a Data Design project
- ▶ Creating a physical data model
- ▶ Modeling with diagrams
- ▶ Generating DDL from a physical data model and deploying

9.6.1 Creating a Data Design project

Before you create data models or other data design objects, you must create a Data Design project to store your objects.

A Data Design project is primarily used to store modeling objects. You can store the following types of objects in a Data Design project:

- ▶ Logical data models
- ▶ Physical data models
- ▶ Domain models
- ▶ Glossary models
- ▶ SQL scripts, including DDL scripts

Perform these steps to create a Data Design project:

1. In the Data perspective, select **File** → **New** → **Data Design Project**.
2. In the New Data Design Project wizard, in the Project Name field, type RAD8DataDesign and click **Finish**.

The Data Design project is displayed in the Data Project Explorer view (Figure 9-26).

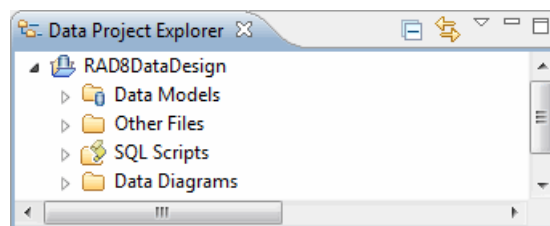


Figure 9-26 Data Project Explorer: Data Design project layout

9.6.2 Creating a physical data model

A *physical data model* is a database-specific model that represents relational data objects (for example, tables, columns, and primary and foreign keys) and their relationships. You can use a physical data model to generate DDL statements, which can then be deployed to a database server.

Using the data tooling, you can create a physical data model in several ways:

- ▶ Create a blank physical model by using a wizard.
- ▶ Create a physical model from a template by using a wizard.
- ▶ Reverse engineer a physical model from a database or a DDL file by using a wizard or by dragging data objects from the Data Source Explorer.
- ▶ Import a physical data model file from the file system.

In this section, we show you two ways to create the physical data model. First, we create a physical data model by reverse engineering the model from an existing database, **ITSOBANK**. Then we create a new physical data model from a template and deploy this new model to the database.

Creating a physical data model using reverse engineering

To create a physical data model by reverse engineering an existing database schema, follow these steps:

1. In the Data Source Explorer, right-click **ITSOBANKDerby** and select **Connect**.
2. In the Data Project Explorer, right-click **RAD8DataDesign** and select **New** → **Physical Data Model**.

3. In the New Physical Data Model wizard (Figure 9-27), complete the following actions:
 - a. For File name, type **ITSOBANK_Reverse**.
 - b. For Database, select **Derby**, and for Version, select **10.5**.
 - c. Select **Create from reverse engineering**.
 - d. Click **Next**.

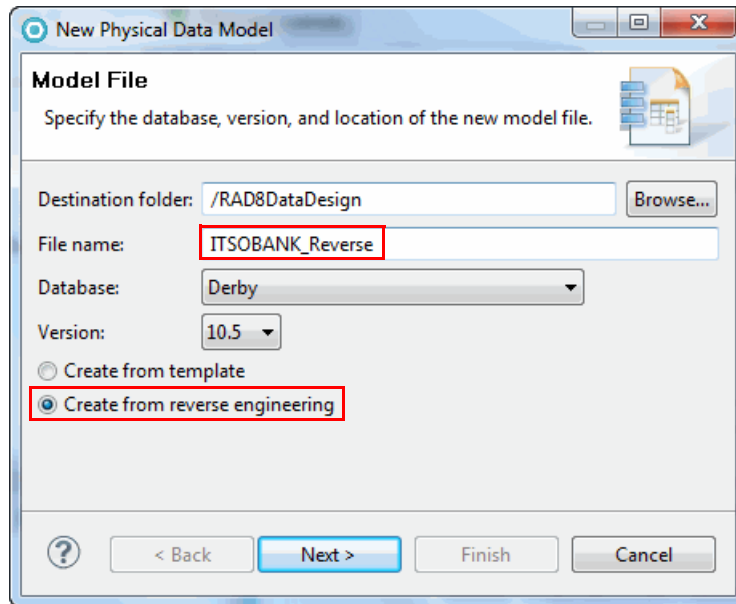


Figure 9-27 New Physical Data Model: Create from reverse engineering

4. In the Select connection window, select **ITSOBANKderby** from the existing connections list. Click **Next**.
5. In the Schema window, select **ITSO** and click **Next**.
6. In the Database Elements window, select **Tables** and click **Next**.

7. In the Options window (Figure 9-28), under Generate diagrams, select **Overview**. Click **Finish**.

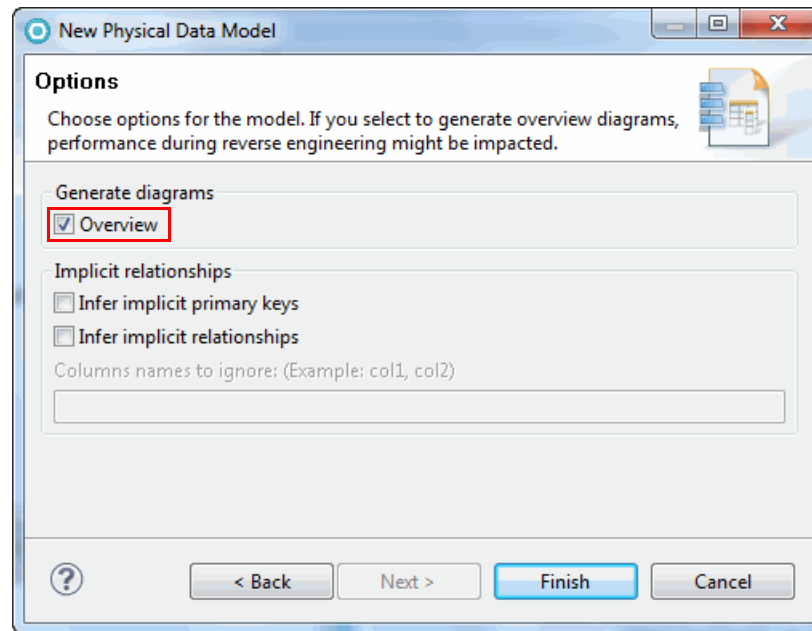


Figure 9-28 New Physical Data Model: Options window

The physical model is created and added to the Data Models folder. The overview diagram is added to the Data Diagrams folder (Figure 9-29).

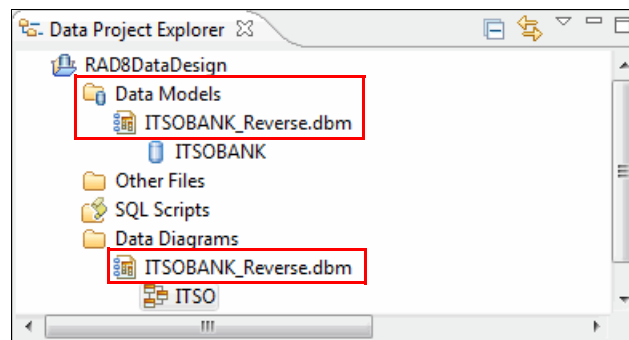


Figure 9-29 Data Design Project with physical model and diagram

The Physical Data Model editor is open on the ITSOBANK_Reverse model. Descriptive information can be added. Do not close the physical model, which must be open to open the diagram.

- Open the **ITSO** overview diagram in the Data Diagrams folder. It contains all the tables that are in the schema. You can move the tables to get a better diagram (Figure 9-30).

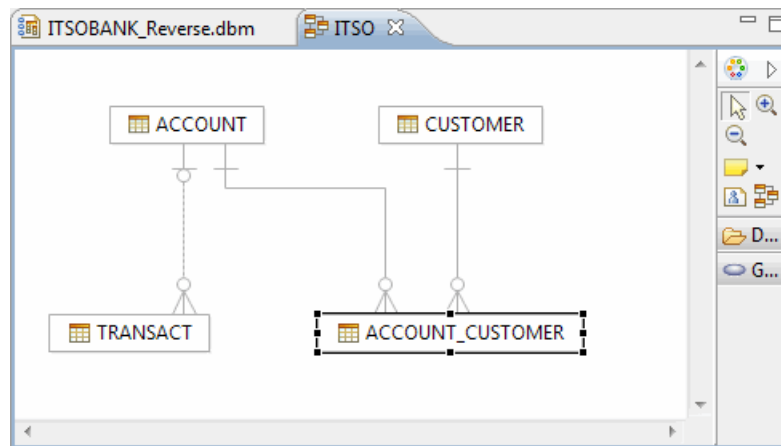


Figure 9-30 Overview diagram

- Select a table in the diagram and then look at the Properties view to see the columns and relationships.
- Select a relationship (line) in the diagram and look at the Properties view to see the cardinality (Details tab). The cardinality is also displayed visually in the diagram.
- Save and close the ITSO diagram and the ITSOBANK_Reverse.dbm model.

Creating a physical data model from a template

To create a physical data model from a template, follow these steps:

- Right-click **RAD8DataDesign** and select **New** → **Physical Data Model**.
- In the New Physical Data Model wizard (Figure 9-27 on page 430), complete the following actions:
 - For File name, type `Bank_model`.
 - For Database, select **Derby**, and for Version, select **10.5**.
 - Select **Create from template**.
 - Click **Finish**.

The physical model is created and displayed in the Data Models folder. The data diagram for the schema opens in the diagram editor.

9.6.3 Modeling with diagrams

You can use data diagrams to visualize and edit objects that are in data projects. *Data diagrams* are a view representation of an underlying data model. You can create diagrams that contain only a subset of model objects that are of interest.

In this section, we create a schema named RAD8Bank in the physical data model. Under this schema, we create two tables: ACCOUNT and TRANSACT. We add a foreign key relationship between the ACCOUNT and TRANSACT tables.

1. In the Data Project Explorer, select **RAD8DataDesign** → **Data Models** → **Bank_model.dbm** → **Database** → **Schema**. In the Properties view, change the schema name from Schema to RAD8Bank.
2. Follow these steps to add a table:
 - a. In the diagram editor, select the **Data** drawer in the palette, and in the Data drawer, select **Table**.
 - b. Click the empty area in the data diagram. A new table is added to the diagram.
 - c. Overtyping the table name with ACCOUNT.
3. Hover the mouse over the **ACCOUNT** table in the diagram. You see four icons outside of the table. Click the **Add Key** icon (highlighted in Figure 9-31).

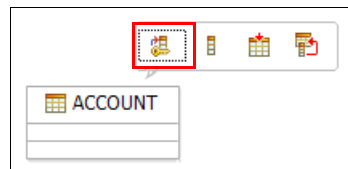


Figure 9-31 Add key, column, index, and trigger icons

4. Overtyping the name with ID (or change the name in the Properties view on the General tab).

5. Select the ID column, and in the Properties view, on the **Type** tab, change the Data type to **VARCHAR**. For Length, enter 16 (Figure 9-32).

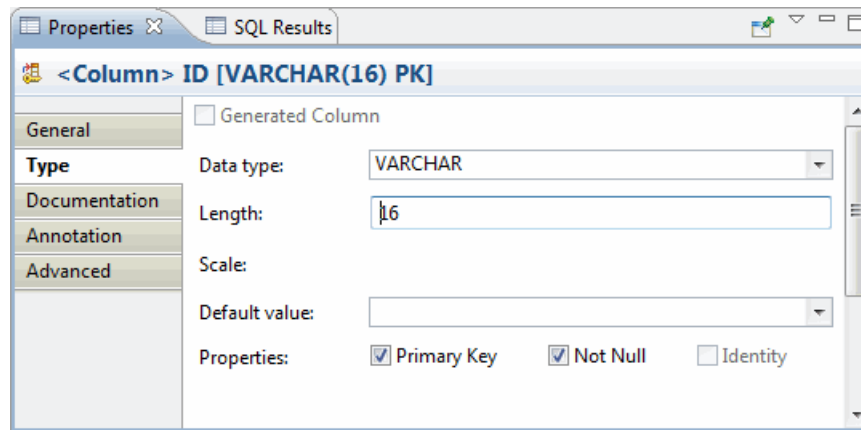


Figure 9-32 Editing the key

6. Hover the mouse over the **ACCOUNT** table and click the **Add Column** icon.
7. In the Properties view, change Name to BALANCE and the Data type to **DECIMAL**. For Precision, type 8, and for Scale, type 2. Select **Not Null**.
8. Follow the same procedure to create the TRANSACT table:
 - Key: ID VARCHAR(250) NOT NULL
 - Columns:
 - TRANS_TYPE VARCHAR(32) NOT NULL
 - TRANS_TIME TIMESTAMP NOT NULL
 - AMOUNT DECIMAL(8,2) NOT NULL
 - ACCOUNT_ID VARCHAR(16)

Figure 9-33 shows the data diagram.

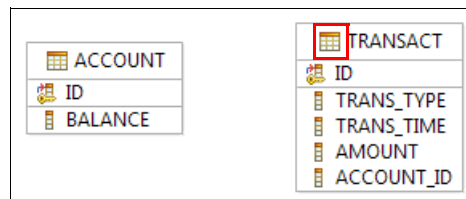


Figure 9-33 Data diagram with two tables

9. Hover the mouse over the **ACCOUNT** table object in the diagram. You see two arrows outside the table, pointing in opposite directions.

Use the arrow that points away from the **ACCOUNT** table (representing a relationship from parent to child) to create a relationship between the **ACCOUNT** table and the **TRANSACTION** table.

10. Drag the arrow that points away from the **ACCOUNT** table to the **TRANSACTION** table. In the menu that opens, select **Create Non-Identifying Optional FK Relationship** (Figure 9-34).

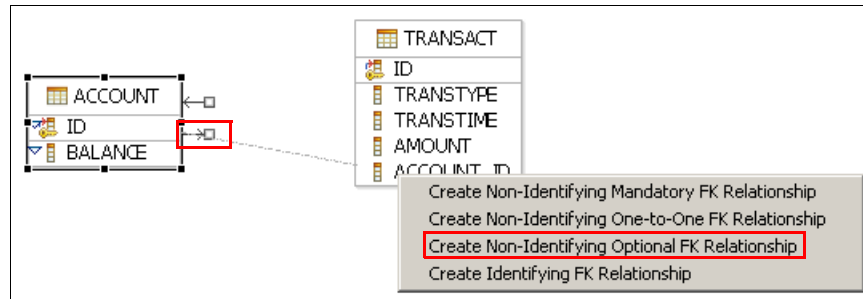


Figure 9-34 Adding a relationship between tables

11. In the Migrate Key Option window (Figure 9-35), select **Use the existing child attribute/column** and click **OK**.

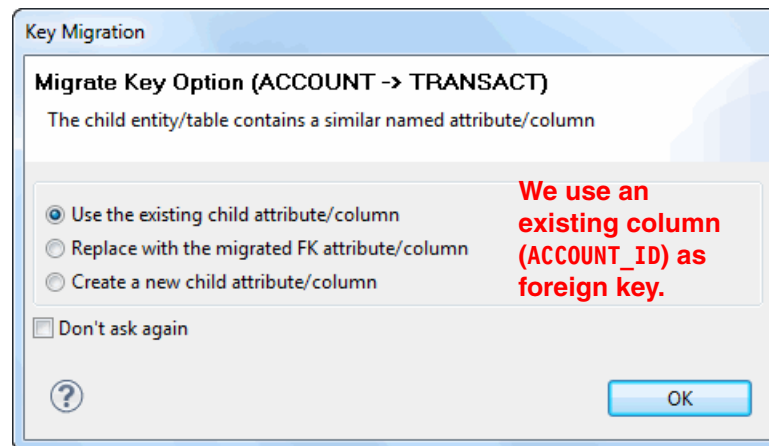


Figure 9-35 Key migration window

12. Select the foreign key relationship that you just created. In the Properties view, select the **Details** page.
13. Click the ellipsis button (...) next to the Key Columns field. In the window that opens, select the **ACCOUNT_ID** check box and clear the **ID** check box. Click **OK**.

14. Add information to the relationship properties to identify the roles of each table in the relationship (Figure 9-36):
 - a. In the Inverse Verb Phrase field, type transaction.
 - b. In the Verb Phrase field, type account.
 - c. Set the cardinality as * and 0..1.

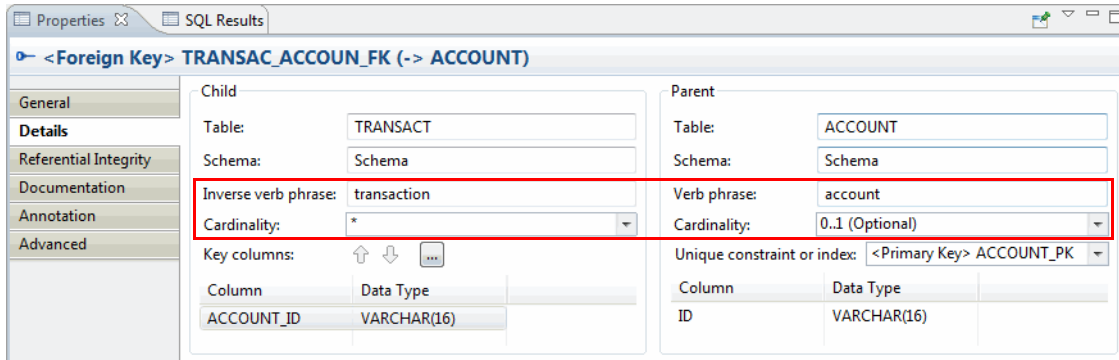


Figure 9-36 Relationship Details page

15. Save but do not close the diagram. Figure 9-37 shows the relationship with the verbs account and transaction in the diagram.

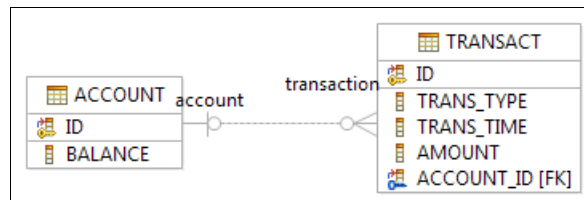


Figure 9-37 Relationship with verbs

9.6.4 Generating DDL from a physical data model and deploying

To generate the DDL and run the generated DDL in the Derby database, follow these steps:

1. In the Data Project Explorer, expand **Data Models** → **Bank_model.dbm** → **Database** and right-click **RAD8Bank** → **Generate DDL**.

- In the Generate DDL: Options window (Figure 9-38), select **Fully qualified name** and **CREATE statements** and click **Next**.

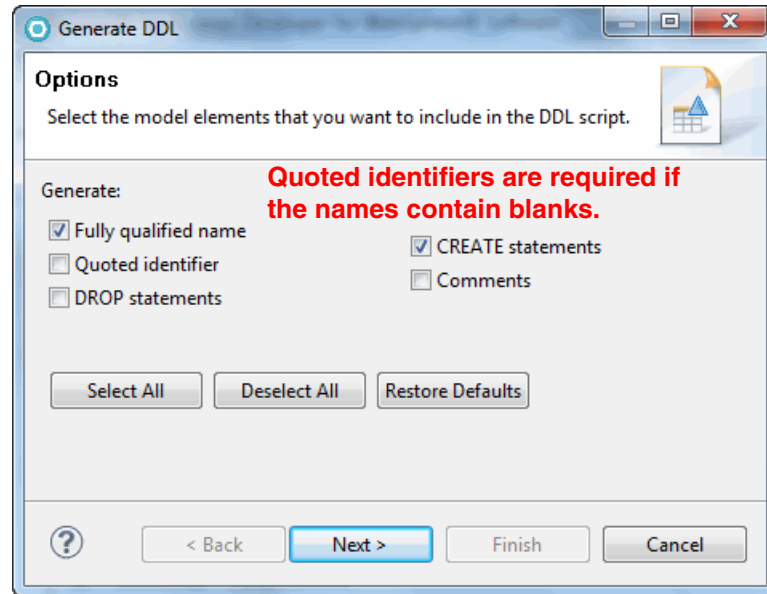


Figure 9-38 Generate DDL window

- In the Objects window, accept all the default settings and click **Next**.

4. In the Save and Run DDL window (Figure 9-39), for File name, type `rad8bank.sql`, review the DDL, select **Run DDL on server**, and click **Next**.

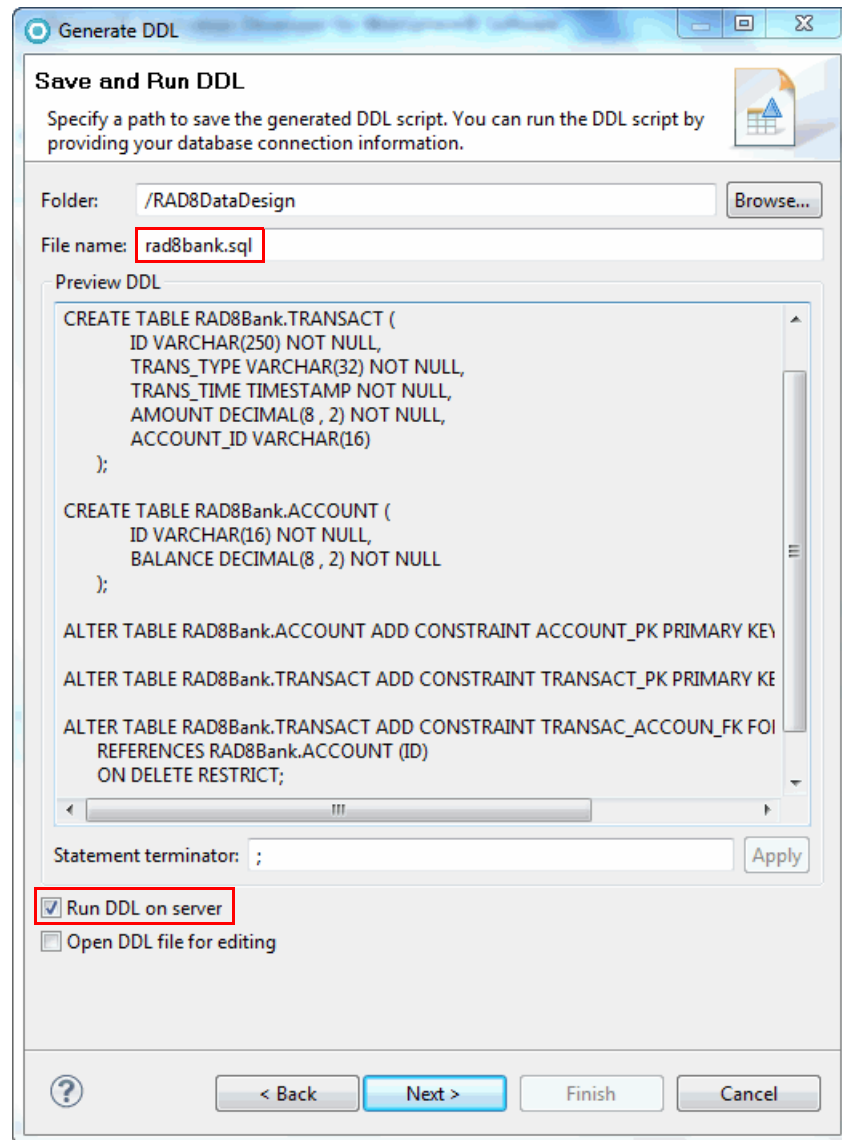


Figure 9-39 Save and Run DDL window

5. In the Select connection window, select **ITSOBANKDerby** and click **Next**.
6. In the Summary window, click **Finish**. The SQL script is created and stored in the SQL Scripts folder.

7. Open the **rad8bank.sql** file to review the DDL.
8. In the Data Source Explorer, right-click the **ITSOBank** connection and select **Refresh**. The RAD8BANK schema is displayed.

Tip: If you cannot see the RAD8BANK schema, you can right-click **Schemas** and select **Properties**. Then make sure that both **ITSO** and **RAD8BANK** are selected. The RAD8BANK schema with two tables is now visible.

9.6.5 Analyzing the data model

The Analyze Model wizard analyzes a data model to ensure that it meets certain specifications. Model analysis helps to ensure model integrity and helps to improve model quality by providing design suggestions and best practices.

Perform these steps to analyze the RAD8Bank schema in the physical data model:

1. In the Data Project Explorer, right-click the schema **RAD8Bank** (by selecting **Data Models** → **Bank_model.dbm** → **Database**) and select **Analyze Model**.

- In the Analyze Model window (Figure 9-40), select the items in the list to see the rules that are checked. Click **Apply** if you made any changes and then click **Finish**.

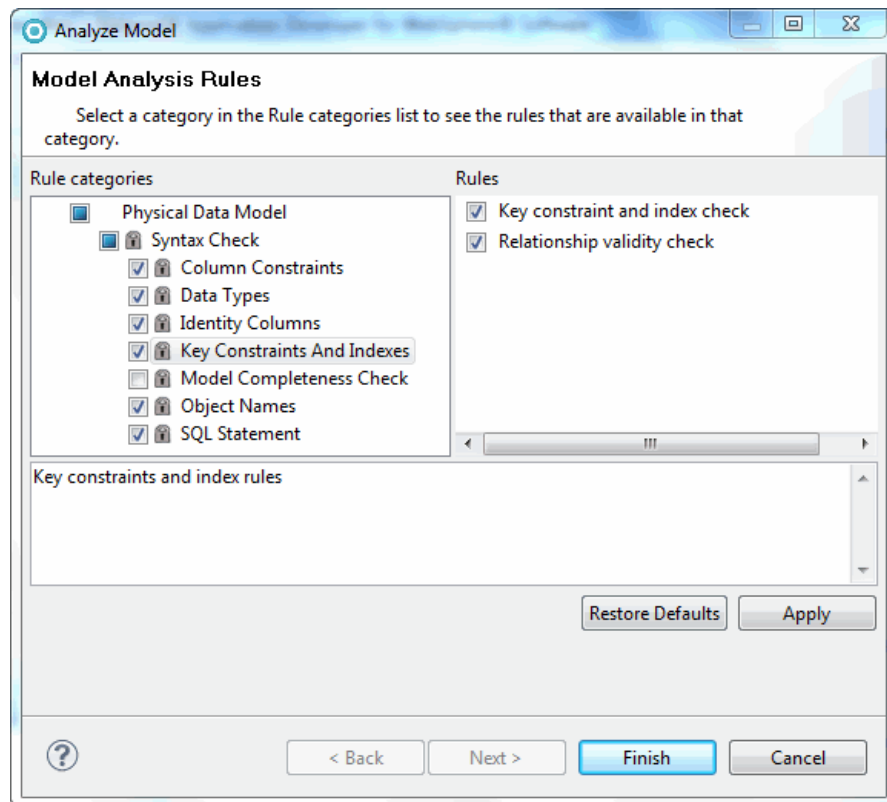


Figure 9-40 Analyze Model window

The following result is displayed in the console:

```
Validation - 0 error(s), 0 warning(s), 0 informational message(s).
```

- Close all the open files.

9.7 More information

For more information about JDBC, see the following pages on the Oracle Sun Developer Network:

- Documentation

<http://java.sun.com/products/jdbc/overview.html>

- ▶ Java SE Technologies: Database
<http://java.sun.com/javase/technologies/database/>
- ▶ JDBC Data Access API
<http://developers.sun.com/product/jdbc/drivers>

For more information about SQLJ, see the following web resources:

- ▶ InfoWorld Java World: “SQLJ: The ‘open sesame’ of Java database applications”
<http://www.javaworld.com/javaworld/jw-05-1999/jw-05-sqlj.html>
- ▶ O’Reilly ON Java
<http://www.onjava.com/pub/st/27>



Persistence using the Java Persistence API

In this chapter, we create the Java Persistence API (JPA) entities that coordinate and mediate access with the ITSOBANK database. We can use either the Derby or the DB2 database to create the matching JPA entities (Customer, Account, and Transaction). We have a choice between the *bottom-up* scenario and the *top-down* scenario. To connect the entity model and any of the two databases, we use a Java Naming and Directory Interface (JNDI) data source in the server.

To illustrate the JPA tooling, we create a JPA project and JPA entities. We use the Derby database. We add inheritance to the entity model by introducing the Credit and Debit subclasses of the Transaction entity.

The chapter is organized into the following sections:

- ▶ Introducing the Java Persistence API
- ▶ Creating a JPA project
- ▶ Creating JPA entities
- ▶ Creating a JPA Manager Bean
- ▶ Visualizing JPA entities
- ▶ Testing JPA entities
- ▶ Preparing the entities for deployment in the server
- ▶ More information

The sample code for this chapter is in the `\7835codesolution\jpa` folder.

10.1 Introducing the Java Persistence API

The JPA defines the management of persistence and object-relational mapping within Java Enterprise Edition (Java EE) and Java Standard Edition (Java SE) environments. The JPA represents a simplification of the persistence programming model. In the past, JPA was defined within the Java EE specification for Enterprise JavaBeans (EJB) 3.0. With JPA 2.0, the JPA specification is defined separately in *Java Specification Request (JSR) 317: Java Persistence API, Version 2.0*.

JPA combines the best features from previous persistence mechanisms, such as Java Database Connectivity (JDBC) APIs, Object Relational Mapping (ORM) frameworks, and Java Data Objects (JDO). Creating entities under JPA is as simple as creating serializable classes. JPA supports the large data sets, data consistency, concurrent use, and query capabilities of JDBC. Like object-relational software and object databases, JPA allows the use of advanced object-oriented concepts, such as inheritance.

The JPA implementation does not mandate that you migrate existing applications. Existing EJB 2.x Container Manager Persistence applications continue to execute without changes.

With the JPA tools in Rational Application Developer, you can use wizards to create and automatically initialize mappings. You can create new database tables from existing entity classes (top-down mapping). In the other scenario, you create new entity beans from existing database tables (bottom-up mapping). You can also use the tools to create mappings between existing database tables and entity beans (meet-in-the-middle mapping), where names or other attributes differ. For flexibility in designing your data access application, you can choose from a range of mapping types. You can create mappings from several types of Java classes, and you can specify entity inheritance with several options for database design.

Details of the JPA specification JSR 317 and certain JPA 2.0 enhancements are described in the following sections:

- ▶ JPA entity object
- ▶ Object-rational mapping
- ▶ Entity inheritance
- ▶ Persistence units
- ▶ Entity Manager
- ▶ JPA Manager Bean
- ▶ Java Persistence Query Language
- ▶ Criteria API
- ▶ Persistence provider

- ▶ JPA 2.0 enhancements

10.1.1 JPA entity object

An *entity object* is a simple Java class that represents a row in a database table. Entities can be concrete classes or abstract classes. They maintain states by using properties or fields.

A JPA entity object is a Java object that must match the following rules:

- ▶ It is a plain old Java object (POJO) that does not have to implement any particular interface or extend a special class.
- ▶ The class must not be declared final, and no methods or persistent instance variables must be declared final.
- ▶ The entity class must have a no-argument constructor that is public or protected. The entity class can have other constructors as well.
- ▶ The class must either be annotated with the `@Entity` annotation or specified in the `orm.xml` JPA mapping file. We use annotations in our examples.
- ▶ The class must define one or more attributes that are used to identify, in an unambiguous way, an instance of that class (they correspond to the primary key in the mapped relational table).
- ▶ Both abstract and concrete classes can be entities, and entities can extend non-entity classes.

A simple entity object example

Example 10-1 shows a simple Customer entity with a few fields.

Example 10-1 Simple entity class with annotations

```
package itso.bank.entities;

@Entity
public class Customer implements java.io.Serializable {

    @Id
    private String ssn;

    private String title;
    private String firstName;
    private String lastName;

    public String getSsn() { return this.ssn; }
    public void setSsn(String ssn) { this.ssn = ssn; }
```

```
    // more getter and setter methods  
}
```

Within this example, you notice that the class is conforming to the JavaBean specification. The `@Entity` annotation identifies this Java class as an entity. The `@Id` annotation is used to identify the property that corresponds to the primary key in the mapped table. In the following section, we show the various types of `@Id` annotation.

Using @Id annotation

The `@Id` annotation offers the simplest mechanism to define the mapping to the primary key. You can associate the `@Id` annotation to fields and properties of these types:

- ▶ Primitive Java types and their wrapper classes
- ▶ Arrays of primitive or wrapper types
- ▶ Strings: `java.lang.String`
- ▶ Large numeric types: `java.math.BigDecimal` or `java.math.BigInteger`
- ▶ Temporal types: `java.util.Date` or `java.sql.Date`

We discourage the use of floating point types, such as `float` and `double`, and their wrapper classes for decimal data, because you can have rounding errors and the result of using the equals (=) operator is unreliable in these cases. Use `BigDecimal` instead.

The `@Id` annotation fits well in scenarios where a natural primary key is available, or when database designers use surrogate primary keys (typically, an integer) that has no descriptive value and is not derived from any application data in the database.

Composite keys are useful when the primary key of the corresponding database table consists of more than one column. Composite keys can be defined by the `@IdClass` or `@EmbeddedId` annotation. The `@IdClass` annotation is used to model a composite key.

The `@EmbeddedId` annotation is used in conjunction with the `@Embeddable` annotation to move the definition of a composite key inside the entity. The `@Embeddable` annotation is used to model persistent objects that have no identity of their own, because they are nested inside another entity.

10.1.2 Object-rational mapping

Before we explain object-rational mapping in detail and the entity relationships, we review how the concept of relationships is defined in object-oriented and

relational database management system (RDBMS) environments, as shown in Table 10-1.

Table 10-1 Relationship concept in two separate environments

Java/JPA	RDBMS
A <i>relationship</i> is a reference from one object to another. Relationships are defined through object references (pointers) from a source object to the target object.	Relationships are defined through foreign keys.
If a relationship involves a collection of other objects, a collection or array type is used to hold the contents of the relationship.	Collections are either defined by the target objects having a foreign key back to the source object primary key, or by having an intermediate join table to store the relationships.
Relationships are always unidirectional. If a source object references a target object, it is not guaranteed that the target object also has a relationship to the source object.	Relationships are defined through foreign keys and queries, so that the inverse query always exists.

Mapping the table and columns

To specify the mapping of the entity to a database table, we use the `@Table` and `@Column` annotations. Example 10-2 shows the declaration for these annotations, which are highlighted.

Example 10-2 Entity with mapping to a database table

```
@Entity
@Table (schema="ITS0", name="CUSTOMER")
public class Customer implements java.io.Serializable {

    @Id
    @Column (name="SSN")
    private String ssn;
    @Column (name="LAST_NAME")
    private String lastName;
    private String title;
    private String firstName;
    .....
}
```

Note the following points:

- ▶ The `@Table` annotation provides information related to the table and schema to which the entity corresponds.
- ▶ The `@Column` annotation provides information related to which column is mapped by an entity property. By default, properties are mapped to columns with the same name, and the `@Column` annotation is used when the property and column names differ.

Persistence mechanisms: Entities support two types of persistence mechanisms:

- ▶ Field-based persistence: The entity properties must be declared as public or protected, and annotations are added on properties.
- ▶ Property-based persistence: You must provide getter/setter methods, and annotations are added on these methods.

See also the annotation `@AccessType` with possible values `AccessType.FIELD` and `AccessType.PROPERTY`, which are applicable at the level of the entity or of a specific attribute.

Mapping through annotation relationships

JPA defines the following relationships:

- ▶ One-to-one
- ▶ Many-to-one
- ▶ One-to-many
- ▶ Many-to-many

One-to-one relationship

In a one-to-one relationship, each entity instance is related to a single instance of another entity. The `@OneToOne` annotation is used to define this single value association, for example, a `Customer` is related to a `CustomerRecord`, as shown in Example 10-3.

Example 10-3 @OneToOne annotation to define a single value association

```
@Entity
@Table (schema="ITS0", name="CUSTOMER")
public class Customer {

    @OneToOne
    @JoinColumn(
        name="CUSTREC_ID", unique=true, nullable=false, updatable=false)
    public CustomerRecord getCustomerRecord() {
        return customerRecord;
    }
}
```

```
    ....  
}
```

In many situations, the target entity of the one-to-one has a relationship back to the source entity, but this is not required. In our example, `CustomerRecord` can have a reference back to the `Customer`. When this is the case, we call it a *bidirectional one-to-one relationship*.

There are two rules for bidirectional one-to-one associations:

- ▶ The `@JoinColumn` annotation must be specified in the entity that is mapped to the table containing the join column, or the owner of the relationship.
- ▶ The `mappedBy` element must be specified in the `@OneToOne` annotation in the entity that does not define a join column, that is, the inverse side of the relationship.

Many-to-one and one-to-many relationships

The many-to-one mapping is used to represent a single valued reference to a Java object, allowing multiple source objects to reference the same target object. In Java, a single pointer stored in an attribute represents the mapping between the source and target objects. Relational database tables implement these mappings using foreign keys.

The one-to-many mapping is used to represent the relationship between a single source object and a collection of target objects. This relationship is usually represented in Java with a collection of target objects. A new feature in JPA 2.0 is the ability to map a one-to-many relationship unidirectionally, using the annotation `@JoinColumn`. In JPA 1.0, you were only able to achieve a one-to-many relationship unidirectionally by using `@JoinTable`. In Open JPA 1.0, you were only able to use the proprietary annotation `org.apache.openjpa.persistence.jdbc.ElementJoinColumn`.

For example, an `Account` entity object can be associated with many `Transact` entity objects using `@OneToMany` and `@ManyToOne` annotations, as shown in Example 10-4.

Example 10-4 @OneToMany and @ManyToOne annotations

```
@Entity  
@Table (schema="ITSO", name="ACCOUNT")  
public class Account implements Serializable {  
    @Id  
    private String id;  
    private BigDecimal balance;  
  
    @OneToMany (mappedBy="account")
```

```

private List<Transaction> transacts;

    ....
}

=====
@Entity
@Table (schema="ITSO", name="TRANSACTION")
public class Transaction implements Serializable {
    @Id
    private String id;
    .....

    @ManyToOne
    @JoinColumn (name="ACCOUNT_ID")
    private Account account;
    ....
}

```

Using the @JoinColumn annotation

In the database, a *relationship mapping* means that one table has a reference to another table. The database term for a column that refers to a key (usually the primary key) in another table is a *foreign key column*. In JPA, we call them *join columns*, and the @JoinColumn annotation is used to configure these types of columns.

No @JoinColumn?: If you do not specify @JoinColumn, a default column name is assumed. The algorithm that is used to build the name is based on a combination of both the source and target entities. It is the name of the relationship attribute in the Transaction source entity (the account attribute), plus an underscore character (_), plus the name of the primary key column of the target Account entity (the id attribute).

Therefore, a foreign key named ACCOUNT_ID is expected inside the TRANSACTION table. If this is not applicable, you must use @JoinColumn to override this automatic behavior.

The @JoinColumn annotation also applies to one-to-one relationships.

Many-to-many relationship

When entity A references multiple B entities, and other B entities might reference several of the same As, we call this a *many-to-many relationship* between A and

B. To implement a many-to-many relationship, a distinct join table, which is called an *association table*, must map the many-to-many relationship.

For example, a Customer entity object can be associated with many Account entity objects, and an Account entity object can be associated with many Customer entity objects, as shown in Example 10-5.

Example 10-5 Associating Customer and Account entity objects

```
@Entity
@Table (schema="ITSO", name="CUSTOMER")
public class Customer implements Serializable {
    .....

    @ManyToMany (mappedBy="customers")
    private List<Account> accounts;
    .....
}
=====
@Entity
@Table (schema="ITSO", name="ACCOUNT")
public class Account implements Serializable {
    .....

    @ManyToMany
    @JoinTable (name="ACCOUNT_CUSTOMER", schema="ITSO",
        joinColumns=@JoinColumn (name="ACCOUNT_ID"),
        inverseJoinColumns=@JoinColumn (name="CUSTOMER_SSN"))
    private List<Customer> customers;
    ....
}
```

The `@JoinTable` annotation is used to specify the table and columns in the database that associate customers with accounts. The entity that specifies the `@JoinTable` is the owner of the relationship. Therefore, in this case, the Account entity is the owner of the relationship with the Customer entity.

The join column pointing to the owning side is described in the `joinColumns` element. The join column pointing to the inverse side is specified by the `inverseJoinColumns` element.

Foreign keys: Neither the CUSTOMER table nor the ACCOUNT table contains a foreign key. The foreign keys are in the association table. Therefore, the Customer or the Account entity can be defined as the owning entity.

Fetch modes

When an Entity Manager retrieves an entity from the underlying database, it can use two types of fetch strategies:

- ▶ Eager mode: When you retrieve an entity from the Entity Manager or by using a query, you are guaranteed that all of its fields (with relationships, too) are populated with data store data.
- ▶ Lazy mode: This is a hint to the JPA run time that you want to defer the loading of the field until you access it. Lazy loading is completely transparent; when you attempt to read the field for the first time, the JPA run time loads the value from the data store and populates the field automatically.

```
@OneToMany(mappedBy="accounts", fetch=FetchType.LAZY)  
private List<Transaction> transacts;
```

Lazy mode is the default for 1:m and m:m relationships, so the specification is optional in those cases.

Performance impact of eager mode: The use of eager mode can greatly affect the performance of your application, especially if your entities have many and recursive relationships, because all of the entity will be loaded at one time.

If the subsequent entity has been read by the Entity Manager and is detached and sent over the network to another layer, ensure that all the entity attributes have been read from the underlying data store or that the receiver does not require related entities.

Object-relational mapping through orm.xml

The object-relational mapping of an entity can be done through the use of annotations, as described in “Mapping through annotation relationships” on page 448. As an alternative, you can specify the same information in an external file, the `orm.xml` file. This file must be packaged in the `META-INF` directory of the persistence module or in a separate file packaged as a resource and defined in the `persistence.xml` file with the `mapping-file` element. Example 10-6 shows an extract of an `orm.xml` file that defines the `Account` entity.

Example 10-6 Extract of an `orm.xml` file to define an entity mapping

```
<entity-mappings .....>  
  <entity class="itso.bank.entity.Account" metadata-complete="true"  
    name="Account">  
    <description>Account of ITSO Bank</description>  
    <table name="ACCOUNT" schema="ITSO"></table>  
    <attributes>  
      <id name="accountNumber">
```



```
        <column name="id"/>
    </id>
    <basic name="balance"></basic>
    <one-to-many name="transacts"></one-to-many>
</attributes>
</entity>
</entity-mappings>
```

To define or edit the object-relational mappings for JPA entity beans in the `orm.xml` file, you use the Object Relational Mapping XML Editor. For this purpose, you right-click the `orm.xml` file of the JPA project that you want to edit in the Package Explorer view and select **Open with** → **Object Relational Mapping XML Editor**.

All new mapping enhancements can be defined within the `orm.xml` file as well.

Overrides: The mapping information defined in the `orm.xml` file automatically overrides both the default JPA behavior and any mappings that are defined using annotations.

10.1.3 Entity inheritance

The term *impedance mismatch* describes the difficulties in bridging the object and relational environments. In regard to inheritance, unfortunately, there is no natural and efficient way to represent an inheritance relationship in a relational database.

JPA introduces the following strategies to support inheritance:

▶ Single table

This strategy maps all classes in the hierarchy to the base class table. This means that the table contains the superset of all the data in the class hierarchy. For an example of single table inheritance, see 10.6.8, “Adding inheritance” on page 519.

▶ Joined tables

With this strategy, the top-level entry in the entity hierarchy is mapped to a table that contains columns common to all the entities, and each of the other entities down the hierarchy is mapped to a table that contain only columns specific to that entity.

- ▶ Table per class

With this strategy, both the superclass and subclasses are stored in their own tables, and no relationship exists between any of the tables. Therefore, all the entity data is stored in its own tables.

10.1.4 Persistence units

A persistence unit defines a set of entity classes that is managed by one *EntityManager* instance in an application. This set of entity classes represents the data contained within a single data store. A persistence unit consists of the declarative metadata that describes the relationship of entity class objects to a relational database. The *EntityManagerFactory* uses this data to create a persistence context that can be accessed through the *EntityManager*.

Persistence units are defined in the `persistence.xml` configuration file. This file must be packaged in the `META-INF` directory of the persistence module.

Example 10-7 shows an extract of a `persistence.xml` file.

Example 10-7 Extract of a persistence.xml file

```
<persistence version="2.0" .....>
  <persistence-unit name="RAD8JPA" transaction-type="JTA">
    <jta-data-source>jdbc/itsobank</jta-data-source>
    <class>itso.bank.entities.Account</class>
    <class>itso.bank.entities.Customer</class>
    <class>itso.bank.entities.Transaction</class>
  </persistence-unit>
</persistence>
```

The `persistence.xml` file describes the details of the persistence units in your JPA project. A persistence unit contains a list of entity beans. To edit the `persistence.xml` file, you use the Persistence XML Editor. Therefore, you right-click the `persistence.xml` file of the JPA project that you want to edit in the Package Explorer view and select **Open with** → **Persistence XML Editor**. Table 10-2 on page 455 shows the persistence unit details, which can be changed.

Table 10-2 Persistence unit details

Attribute	Description
Name	The name of the persistence unit. This attribute is required.
Description	To represent the description of the persistence unit.
Provider	Name of the <code>javax.persistence.spi.PersistenceProvider</code> class interface for the persistence provider.
Transaction type	Choose if it is a Java transaction API (JTA) of a non-JTA data source.
JTA data source	Define the global JNDI name of the JTA data source.
Non-JTA data source	Define the global JNDI name of a non-JTA data source.
Exclude unlisted classes	This attribute is <code>False</code> by default. When this is set to <code>True</code> , classes, which are not listed in the <code>persistence.xml</code> file, will be excluded from the persistence unit.
JAR file	The name of JAR files containing entities for this persistence unit.
Mapping file	If you use a custom mapping file, list it here. The <code>orm.xml</code> file for mapping is read automatically and is not to be added here.

JPA 2.0 also provides the service to synchronize persistent classes in the `persistence.xml` file. To do this, right-click the **persistence.xml** file of the JPA project and select **JPA Tools** → **Synchronize classes**. As result of this feature, the persistent classes in your JPA project are automatically discovered and added to the persistence unit in the `persistence.xml` file.

10.1.5 Entity Manager

Entities cannot persist themselves on the relational database. Annotations are used only to declare a POJO as an entity or to define its mapping and relationships with the corresponding tables on the relational database.

JPA has defined the `EntityManager` interface for this purpose to let applications manage and search for entities in the relational database. The `EntityManager` primary definition includes the following elements:

- ▶ It is an object that manages a set of entities defined by a persistence unit.
- ▶ Each `EntityManager` instance is associated with a *persistence context*.
- ▶ An API manages the life cycle of entity instances. Table 10-3 on page 456 shows the major operations that can be performed by an `EntityManager`.

Table 10-3 Entity Manager operations

Operation	Description
persist	This operation results in the insertion of a new entity instance into the database. It saves the persistent state of the entity and any owned relationship references. The entity instance becomes managed.
find	This operation obtains a managed entity instance with a given persistent identity, which will be the primary key. In case the object is not found, null will be returned.
remove	This operation deletes a managed entity with the given persistent identity from the database.
merge	The state of a detached entity gets merged into a managed copy of the entity. The managed entity that is returned has a separate Java identity than the detached entity.
refresh	This operation reloads the entity state from the database.
lock	This operation sets the lock mode for an entity object contained in the persistence context.
flush	This operation forces synchronization with the database.
contains	This operation determines if an entity is contained by the current persistence context.
createQuery	This operation creates a query instance using dynamic Java Persistence Query Language (JPQL).
createNamedQuery	This operation creates an instance of a predefined query.
createNativeQuery	This operation creates an instance of an SQL query.
getCriteriaBuilder	The return value is the CriteriaBuilder interface to create a CriteriaQuery. This operation is new for the JPA Criteria API.

The `EntityManager` tracks all entity objects within a persistence context for changes and updates made, and flushes these changes to the database. After a persistence context is closed, all managed entity object instances become detached from the persistence context and its associated `EntityManager`, and are no longer managed. An entity object instance is either managed (attached) by an `EntityManager` or unmanaged (detached).

Managed and unmanaged entities: An entity object instance is either *managed* (attached) by an `EntityManager` or *unmanaged* (detached):

- ▶ When an entity is *attached* to an Entity Manager, the manager monitors any changes to the entity and synchronizes them with the database whenever the Entity Manager decides to flush its state.
- ▶ When an entity is *detached*, and therefore is no longer associated with a persistence context, it is unmanaged, and its state changes are not tracked by the Entity Manager and synchronized with the database.

Container-managed Entity Manager

One way to use an Entity Manager in a Java EE environment is with a container-managed Entity Manager. In this mode, the container is responsible for the opening and closing of the Entity Manager and thus the life cycle of the persistence context. A container-managed Entity Manager is also responsible for transaction boundaries. A container-managed Entity Manager is obtained in an application through dependency injection or through JNDI lookup, and the container manages interactions with the Entity Manager factory transparently to the application. A container-managed Entity Manager requires the use of a JTA transaction, because its persistence context will automatically be propagated with the current JTA transaction, and the Entity Manager references that are mapped to the same persistence unit will provide access to this same persistence context within the JTA transaction. This propagation of persistence context by the Java EE container means that the application does not have to pass references to the Entity Manager instances from one component to another.

Container-managed persistence contexts can be defined to have one of two scopes:

- ▶ Transaction persistence scope
- ▶ Extended persistence scope

Application-managed Entity Manager

Using an application-managed Entity Manager allows you to control the `EntityManager` in application code. When using an application-managed Entity Manager, note the following issues:

- ▶ With application-managed Entity Managers, the persistence context is not propagated to application components, and the life cycle of Entity Manager instances is managed by the application. This means that the persistence context is not propagated with the JTA transaction across `EntityManager` instances in a particular persistence unit.
- ▶ The `EntityManager`, and its associated persistence context, is created and destroyed explicitly by the application.

You typically use this type of Entity Manager in two scenarios:

- ▶ In Java SE environments, where you want to access a persistence context that is stand-alone, and not propagated along with the JTA transaction across the EntityManager references for the given persistence unit
- ▶ Inside a Java EE container, when you want to gain fine-grained control over the EntityManager life cycle

10.1.6 JPA Manager Bean

JPA Manager Beans are service beans, which act as facades or controllers over a particular JPA entity. They encapsulate and abstract all of the data access code for creating, updating, deleting, and displaying information from your database using JPA entities. JPA Manager Beans map in a one-to-one relationship to a JPA entity. For the Account entity, you can create a JPA Manager Bean named AccountManager, which contains all of the data access logic needed to work with the Account entity.

JPA Manager Beans are a programming model that is ideal for use in two-tier web environments. JPA Manager Beans act in a role that is similar to the role of a session bean in an EJB environment. That means all of the business logic related to an entity is performed by the JPA Manager Bean.

JPA entities do not need to reside in the same project as the JPA Manager Beans. For example, your JPA entities can exist in a JPA Utility project, and you can generate JPA Manager Beans for those entities inside of a web project.

To generate a JPA Manager Bean, you have to perform the following steps:

1. Right-click the JPA project or JPA enabled project and select **JPA Tools** → **Add JPA Manager Beans**.
2. The JPA Manager Bean Wizard opens and shows the Available JPA Entities. Select the entities for which to create manager beans.
 - a. You have the additional functionality to create or edit a JPA entity by using **Create New JPA Entities** or **Edit Selected Entities**.
 - b. Select all JPA entities with **Select All** or select at least one of the listed JPA entities.
3. Click **Next**.
4. In the next window, configure the tasks for each entity:
 - a. Check the task **Named Queries**. To define or edit a named query, you have to launch the Entity Configuration Wizard.

- b. In the **Other** tab, review the possible ways of generating JPA Managers:
 - i. Select **I want to manage the persistence unit myself** if you run your application outside of a managed environment, for example, a plain JSP or a Java application.
 - ii. Select **I want the container to inject the persistence unit into my beans** to generate JPA Manager Beans with annotations that use the Web container to inject resources and manage the beans. This option allows you to run your application in a managed resource, such as a JavaServer Faces managed bean.

5. Click **Finish**.

The created JPA Manager Beans are located in the `controller` package and can be used for your purposes. We use the generated JPA Manager Beans, for example, for the development of web services, as shown in Chapter 14, “Developing web services applications” on page 681.

The JPA Manager Beans can be generated in various ways depending on whether they need to be used in a Java SE application or in a Java EE container, and if they have to be exposed as RPC Adapter services or used in JSF applications. For more information about the meaning of the options in the Other task, see this website:

<http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/com.ibm.etools.webtoolscore.doc/topics/tjpaconfigmgrbeanother.html>

10.1.7 Java Persistence Query Language

You use the Java Persistence Query Language (JPQL) to define searches against persistent entities independently of the mechanism used to store those entities. JPQL is *portable* and not constrained to any particular data store.

The JPQL is an extension of the EJB query language and is designed to combine the syntax and simple query semantics of SQL with the expressiveness of an object-oriented expression language. The following steps show how JPQL is used in JPA:

1. The application creates an instance of the `javax.persistence.EntityManager` interface.
2. The `EntityManager` creates an instance of the `javax.persistence.Query` interface through its public methods, for example, `createNamedQuery`.
3. The `Query` instance executes a query to read or update entities.

Example 10-8 on page 460 shows a simple query that retrieves all the `Customer` entities from the database.

Example 10-8 Create query getAllCustomers

```
EntityManager em = ...
Query q = em.createQuery("SELECT c FROM Customer c");
List<Customer> results = (List<Customer>)q.getResultList();
```

A JPQL query has an internal namespace declared in the from clause of the query. Arbitrary identifiers are assigned to entities so that they can be referenced elsewhere in the query. In the query in Example 10-8, the identifier *c* is assigned to the Customer entity.

The where condition is used to express a logical condition. Example 10-9 shows how to define a where condition.

Example 10-9 Create query getCustomerBySSN

```
EntityManager em = ...
Query q = em.createQuery("SELECT c FROM Customer c
                        where c.ssn= '111-11-1111'");
List<Customer> results = (List<Customer>)q.getResultList();
```

JPA 2.0 supports new retrieving definitions according to the resultList, which will be created from the query. These are definitions, such as retrieving the first result and specifying the maximum size of the resultList.

Query types

Query instances are created using the methods exposed by the EntityManager interface. Table 10-4 describes the methods and their use.

Table 10-4 Creating a query instance

Method name	Description
createQuery(String qlString)	Create an instance of Query for executing a JPQL statement.
createNamedQuery (String name)	Create an instance of Query for executing a named query (in the JPQL or in native SQL).
createNativeQuery (String sqlString)	Create an instance of Query for executing a native SQL statement, for example, for update or delete.
createNativeQuery (String sqlString, Class resultClass)	Create an instance of Query for executing a native SQL query that retrieves a single entity type.

Method name	Description
createNativeQuery (String sqlString, String resultSetMapping)	Create an instance of Query for executing a native SQL query statement that retrieves a result set with multiple entity instances.
New methods as a result of the new Criteria API	
createQuery (CriteriaQuery<T>, criteriaQuery)	Create an instance of TypedQuery for executing a criteria query.
createQuery (Class<T> resultClass)	Create a CriteriaQuery object with the specified result type.
createQuery (String sqlString, Class<T> resultClass)	Create an instance of TypedQuery for executing a JPQL statement. The select list of the query must contain only a single item, which must be assignable to the type specified by the resultClass argument.
createNamedQuery (String sqlString, Class<T> resultClass)	Create an instance of TypedQuery for executing a JPQL named query.

As a result of the new Criteria API, described in 10.1.8, “Criteria API” on page 464, there are new methods provided by the `EntityManager` interface.

Table 10-4 on page 460 includes these new methods to have a complete overview of methods provided by the `EntityManager` interface. They are separated in the second part of the table.

Operators

JPQL provides several operators. The following operators are most often used:

- ▶ Logical operators: NOT, AND, OR
- ▶ Relational operators: =, >, >=, <, <=, <>, [NOT] BETWEEN, [NOT] LIKE, [NOT] IN, IS [NOT] NULL, IS [NOT] EMPTY, [NOT] MEMBER [OF]
- ▶ Arithmetic operators: +, -, /, *

Within JPA 2.0, the JPQL is enhanced with CASE expressions NULLIF and COALESCE.

Named queries

JPQL defines two types of queries:

- ▶ Dynamic queries

These queries are created at run time.

- ▶ Named queries

These queries are intended to be used in contexts where the same query is invoked several times. Their major benefits include the improved reusability of the code, minor maintenance effort, and finally, better performance because they are evaluated one time.

Defining a named query

Named queries are defined by using the `@NamedQuery` annotation. Example 10-10 shows the definition of a named query.

Example 10-10 NamedQuery with positional parameter

```
@Entity
@Table (schema="ITSO", name="CUSTOMER")
@NamedQuery(name="getCustomerBySSN",
            query="select c from Customer c where c.ssn = ?1")
public class Customer implements Serializable {
    ...
}
```

The name attribute is used to identify the named query uniquely. The query attribute defines the query. We can see how this syntax resembles the syntax used in JDBC code with `jdbc.sql.PreparedStatement` statements.

Instead of a positional parameter (`?1`), the same named query can be expressed by using a named parameter, as shown in Example 10-11.

Example 10-11 NamedQuery with named parameter

```
@NamedQuery(name="getCustomerBySSN",
            query="select c from Customer c where c.ssn = :ssn")
```

Completing a named query

Named queries must have all their parameters specified before being executed. The `javax.persistence.Query` interface exposes two methods:

- ▶ `public void setParameter (int position, Object value)`
- ▶ `public void setParameter (String paramName, Object value)`

Example 10-12 on page 463 show a complete example that uses a named query.

Example 10-12 Use of named query

```
EntityManager em = ...
Query q = em.createNamedQuery("getCustomerBySSN");
q.setParameter(1, "111-11-1111");
//q.setParameter("ssn", "111-11-1111"); // for named parameter
List<Customer> results = (List<Customer>)q.getResultList();
```

Defining multiple named queries

If an entity has more than one named query, the named queries are placed inside an `@NamedQueries` annotation, which accepts an array of one or more `@NamedQuery` annotations. Example 10-13 shows the definition of queries from the Customer table.

Example 10-13 Define multiple named queries

```
@NamedQueries({
    @NamedQuery(name="getCustomers",
        query="select c from Customer c"),
    @NamedQuery(name="getCustomerBySSN",
        query="select c from Customer c where c.ssn =:ssn"),
    @NamedQuery(name="getAccountsBySSN",
        query="select a from Customer c,
                in(c.accounts) a
                where c.ssn =:ssn order by a.accountNumber")
})
```

Relationship navigation

Relations between objects can be traversed by using a Java-like syntax. This syntax is shown in Example 10-14.

Example 10-14 Use relationship

```
SELECT t FROM Transaction t WHERE t.account.id = '001-111001'
```

There are other ways to use queries to traverse relationships. For example, if the Account entity has a property called `transacts` that is annotated as an `@OneToMany` relationship, this query retrieves all the Transaction instances of one Account. Example 10-15 shows this use.

Example 10-15 Named query for OneToMany relationship

```
@NamedQuery(name="getTransactionsByID",
    query="select t from Account a,
    in(a.transacts) t where a.id =:aid order by t.transTime")
```

10.1.8 Criteria API

The JPA *Criteria API* is a new feature in Version 2.0 of JPA. The defined queries can be verified for syntactical correctness at compile time. Furthermore, the JPA Criteria API is used to define queries through the construction of object-based query definition objects. JQPL, which is described in 10.1.7, “Java Persistence Query Language” on page 459, uses the string-based approach. With this string-based approach, it is not possible to verify the syntactical correctness at compile time.

The syntax of the Criteria API is designed to allow the construction of an object-based query “graph”, whose nodes correspond to the semantic query elements. The definition in this section corresponds to the “Criteria API” chapter in the JPA specification *JSR 317: Java Persistence API, Version 2.0*.

A criteria query is constructed through the creation and modification of a `javax.persistence.criteria.CriteriaQuery` object. To construct these `CriteriaQuery` objects, you use the `CriteriaBuilder` interface. The `CriteriaBuilder` implementation is accessed through the `getCriteriaBuilder` method of the `EntityManager` or `EntityManagerFactory` interface. Example 10-16 shows the definition of a `CriteriaBuilder`.

Example 10-16 Create CriteriaBuilder

```
EntityManager em = ...;
CriteriaBuilder cb = em.getCriteriaBuilder();
```

A `CriteriaQuery` object is created by the method `createQuery` or the `createTupleQuery` method of the `CriteriaBuilder` interface. A `CriteriaQuery` object is typed according to its expected result type when the `CriteriaQuery` object is created. The creation of a `CriteriaQuery` object, with the result type `Customer`, is shown in Example 10-17.

Example 10-17 Create CriteriaQuery

```
CriteriaBuilder cb = ...
CriteriaQuery<Customer> q = cb.createQuery(Customer.class);
```

A criteria query is executed by passing the `CriteriaQuery` object to the `createQuery` method of the `EntityManager` interface to create a `TypedQuery` object, which can then be passed to one of the query execution methods of the `TypedQuery` interface.

The preceding description shows how to create the most important elements of the Criteria API, which are the `CriteriaBuilder` and the `CriteriaQuery` classes. The `CriteriaBuilder` provides an entry point into the API and delivers various factory

methods for constructing queries. The `CriteriaQuery` class is a container for holding and assembling query elements.

Example 10-18 shows the creation of the `CriteriaQuery` for the query to select one certain customer with `ssn = "111-11-1111"`, as defined before with JPQL in the example in “Completing a named query” on page 462.

Example 10-18 CriteriaQuery for getCustomerBySSN

```
//select c from Customer c where c.ssn = "111-11-1111";
CriteriaBuilder criteriaBuilder = entityManager.getCriteriaBuilder();
CriteriaQuery<Account> criteriaQuery =
    criteriaBuilder.createQuery(Customer.class);
Root<Customer> root = criteriaQuery.from(Customer.class);
Predicate predicate =
    criteriaBuilder.gt(root.get(Customer_.ssn), "111-11-1111");
criteriaQuery.where(predicate);
TypedQuery<Account> typedQuery =
    entityManager.createQuery(criteriaQuery);
List<Customer> resultList = typedQuery.getResultList();
assertTrue(resultList.size() == 1);
```

To specify the parameters, the `Predicate` object is used to store the parameters in this object. The `Predicate` object is the parameter for the method `where` of the `CriteriaQuery` object, as shown with `criteriaQuery.where(predicate)`. The method `createQuery` with the parameter `CriteriaQuery` of the `EntityManager` object is used to create the `TypedQuery` object, which delivers the `resultList`.

Canonical metamodel classes: For the creation of the `Predicate` object, the canonical metamodel class of the `Customer` class, `Customer_` is used. These canonical metamodel classes are produced for every managed class in your persistence unit.

The metamodel class can either be generated by means of an annotation processor, or they can be accessed dynamically when using the `javax.persistence.metamodel.Metamodel` interface. Therefore, the `EntityManager` provides the method `getMetamodel`. For more details of the metamodel, see 6.2 “Metamodel” in the JPA specification *JSR 317: Java Persistence API, Version 2.0*.

Rational Application Developer provides the support for generating the canonical metamodel. For enabling an automatic generation, you need to set the source folder in the Canonical metamodel pane of the Java Persistence properties area.

To configure the source folder for the metadata of the Criteria API, perform the following steps (possible after creation of JPA project, defined in 10.2, “Creating a JPA project” on page 469):

1. Right-click the **RAD8JPA** project and choose **Properties**.
2. Choose **Java Persistence** in the navigation section of the Properties for the RAD8JPA window.
3. In the Canonical metamodel (JPA 2.0) pane, set **.apt_generated** as the Source folder and click **Apply**.
4. Click **OK**.

A detailed description of JPA Criteria API examples is given in 8.2.2 “Samples” in *Getting Started with the Feature Pack for OSGi Applications and JPA 2.0*, SG24-7911.

For more information, refer to Chapter 6, “Criteria API”, in *JSR 317: Java Persistence API, Version 2.0* and Chapter 8, “JPA Criteria API”, in *Getting Started with the Feature Pack for OSGi Applications and JPA 2.0*, SG24-7911.

10.1.9 Persistence provider

Persistence providers are implementations of the JPA specification and can be deployed in the Java EE-compliant application server that supports JPA persistence.

WebSphere Application Server has two built-in JPA persistence providers:

- ▶ JPA for WebSphere Application Server persistence provider
- ▶ Apache OpenJPA persistence provider

If an explicit provider element is not specified in the persistence unit definitions, the application server uses the default persistence provider, which is the JPA for WebSphere Application Server persistence provider.

JPA for WebSphere Application Server persistence provider

While built from the Apache OpenJPA persistence provider, the JPA for WebSphere Application Server persistence provider contains the following enhancements:

- ▶ Statement batching support
- ▶ Version ID generation
- ▶ ObjectGrid cache plug-in support
- ▶ WebSphere product-specific commands and scripts
- ▶ Translated message files

Apache OpenJPA persistence provider

WebSphere Application Server provides the Apache OpenJPA persistence provider to support the open source implementation of JPA and allows for easy migration of existing OpenJPA applications to the application server's solution for JPA.

For further information, see the information center documentation:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.jpafep.multipatform.doc/info/ae/ae/tejb_defjpadatasource.html

OpenJPA applications: WebSphere Application Server's JPA 2.0 solution is built on OpenJPA, but all OpenJPA functions, extensions, and configurations are unaffected by the WebSphere Application Server extensions, independent of the chosen provider. You do not need to make changes to OpenJPA applications to use these applications in WebSphere Application Server.

10.1.10 JPA 2.0 enhancements

This section provides a summary of the enhancements that come with Version 2.0 of JPA. Several of these features were already present as provider-specific extensions in JPA 1.x. With JPA 2.0, they are portable between providers. The following enhancements are available:

- ▶ Bean validation

JPA 2.0 covers the optional integration of Bean Validation. In section 3.6 of *JSR 317: Java Persistence API, Version 2.0*, the support of Bean Validation in JPA 2.0 is defined in detail.
- ▶ Criteria API

As discussed in 10.1.8, “Criteria API” on page 464, the base for the Criteria API is a metamodel generated from the application's domain entities. The metamodel is used at development time for defining queries, which later allow the compiler to perform syntax checks during compile time.
- ▶ Access type

With JPA 2.0, the restriction that access types within an entity or even an entity hierarchy cannot be mixed is removed. The new `@Access` annotation allows you to have a combination of both field-based and property-based access types within the same entity.
- ▶ Extended map

With JPA 2.0, the support for the mapping has been enhanced. It is possible to contain any combination of basic types, embeddables, or entities as keys or

values. Therefore, the new annotations `@MapKeyColumn`, `@MapKeyClass`, and `@MapKeyJoinColumn` are defined.

► Orphan removal

The persistence provider can optionally manage the relationships between parent and child entities automatically in one-to-one and one-to-many relationships. When the `orphanRemoval` attribute is set to `true`, there is no need to specify `cascade=REMOVE` on the same relation because it is implied by the `orphanRemoval` attribute.

► Derived identity

The relationship definitions for the primary key have this new functionality. This functionality allows the developer to directly define the identity either by an attribute of another related entity or even as a composition of attributes of both the source entity and the related entity.

► Nested embedding

The limitation that only basic relationships are allowed in an embeddable, nested embeddables, or embeddables holding entities has been removed, and now nested embeddables, as well as relations to other entities, are supported.

► New collection mappings

The collection is enhanced to hold, in addition to entities, embeddables and primitives. Therefore, the new annotations `@ElementCollection` and `@CollectionTable` are defined.

► Unidirectional one-to-many mapping

In JPA 1.x, the directly unidirectional one-to-many relationship was not supported. In JPA 2.0, this problem is resolved and eliminates the limitation with a new annotation `@JoinColumn`, as described in “Using the `@JoinColumn` annotation” on page 450.

► Ordered list mapping

In one-to-many and many-to-many relationships, as well as on element collections, the new annotation `@OrderColumn` is used that the persistence provider automatically manages the order of the list.

► Pessimistic logging

In JPA 1.x, only optimistic locking is supported. Now the service for pessimistic locking scenarios is provided. The locking modes `PESSIMISTIC_READ`, `PESSIMISTIC_WRITE`, and `PESSIMISTIC_FORCE_INCREMENT` have been added in JPA 2.0.

- ▶ Standard properties

With JPA 1.x, properties related to the `persistence.xml` configuration file are vendor-specific. With JPA 2.0, the most common properties have now been standardized:

- `javax.persistence.jdbc.driver`
- `javax.persistence.jdbc.url`
- `javax.persistence.jdbc.user`
- `javax.persistence.jdbc.password`

- ▶ API enhancements

Various changes are available for the API in several areas, such as `EntityManager`, `EntityManagerFactory`, queries, and cache issues. For detailed information, see section 3.2.13 “API enhancements” in *Getting Started with the Feature Pack for OSGi Applications and JPA 2.0*, SG24-7911.

- ▶ JPQL enhancements

As mentioned in 10.1.7, “Java Persistence Query Language” on page 459, there is a lot of updated functionality in the JPQL, such as the support for one or more entity types in a query and new syntax support for the CASE function.

For additional information, see Chapter 3, “Introduction to JPA 2.0”, in *Getting Started with the Feature Pack for OSGi Applications and JPA 2.0*, SG24-7911.

10.2 Creating a JPA project

There are multiple ways to create a JPA project:

- ▶ Create a new JPA project
- ▶ Adding JPA support to an existing project
- ▶ Converting a Java project to a JPA project

We discuss these options in this section.

10.2.1 Setting up the ITSOBANK database

The JPA entities are based on the ITSOBANK database. Therefore, we must define a database connection within Rational Application Developer that the mapping tools use to extract schema information from the database.

See “Setting up the ITSOBANK database” on page 1880 for instructions about how to create the ITSOBANK database. For the JPA entities, we can either use the DB2 or Derby database. For simplicity, we use the built-in Derby database in this

chapter. Additionally, you have to create a connection to the database ITSOBANK, as described in 9.2.2, “Creating a connection to the ITSOBANK database” on page 395.

JPA Tools can automatically create a data source in the enhanced EAR. You can use the context menu that is available on the JPA project (**JPA Tools** → **Configure project for JDBC Deployment**) or as described at this website:

<http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/com.ibm.servertools.doc/topics/tjpaautv7.html>

10.2.2 Create a new JPA project

To create a new JPA project, you open the Java EE perspective and perform these steps:

1. Within the Enterprise Explorer view, right-click and select **New** → **Project**.
2. In the New Project wizard, select **JPA** → **JPA Project** and click **Next**.
3. In the New JPA Project wizard, in the JPA Project window, as shown in Figure 10-1 on page 471, define the project details:
 - a. For Project name, type RAD8JPA.
 - b. Select **Use default location** (selected by default) for Project location.
 - c. For Target Runtime, select **WebSphere Application Server v8.0 Beta**.
 - d. For Configuration, select **Minimal JPA 2.0 Configuration**.
 - e. Clear **Add project to an EAR**.

In case you plan to use this JPA project later within the EJB project, we suggest that you add the JPA project to an EAR file by selecting “Add project to an EAR”.
 - f. Clear **Add project to working sets**.
 - g. Click **Next**.

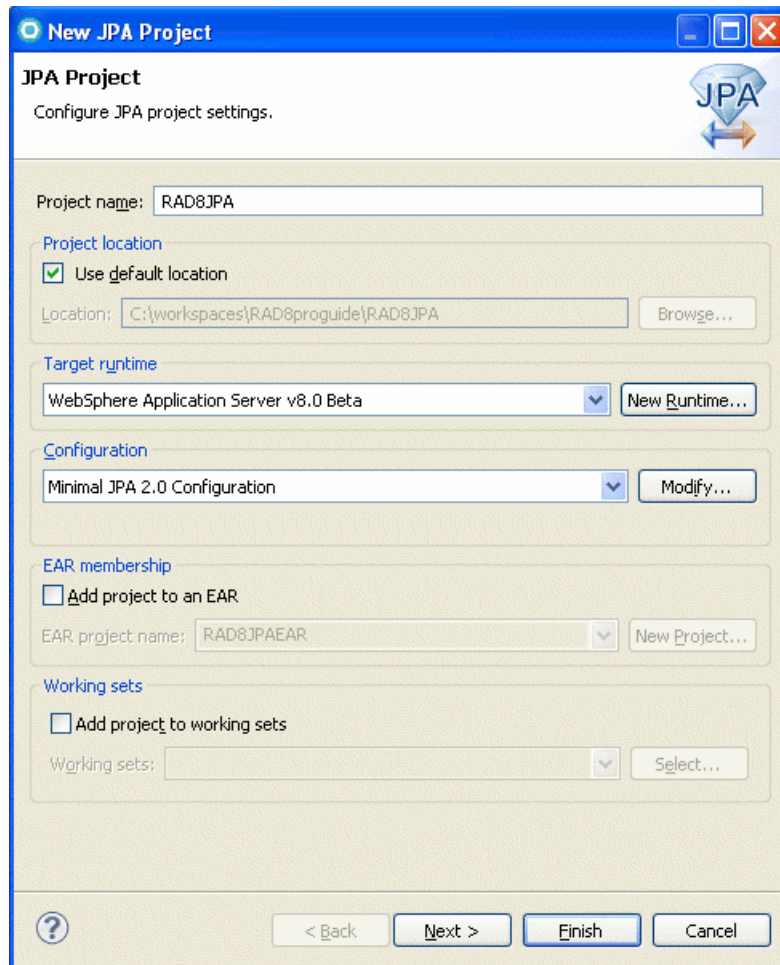


Figure 10-1 New JPA Project wizard

4. Define your src folder in the next window:
 - a. The default folder is src. You can create an additional folder by using Add Folder.
 - b. Click **Next** without adding a new folder.
5. In the New JPA Project wizard in the JPA Facet window, as shown in Figure 10-2 on page 473, define the database details:
 - a. For Platform, select **RAD JPA 2.0 Platform**.
 - b. For JPA implementation, select Type **Library Provided by Target Runtime**.

- c. For Connection, select **ITSOBANKderby**. If the connection is not active, click **Connect**.
- d. Clear **Add driver library to build path**.
- e. Select **Override default schema from connection** and select Schema **ITSO**.
- f. For persistent class management, select **Discover annotated classes automatically**.
- g. Select **Create mapping file (orm.xml)**.
- h. Click **Finish**.

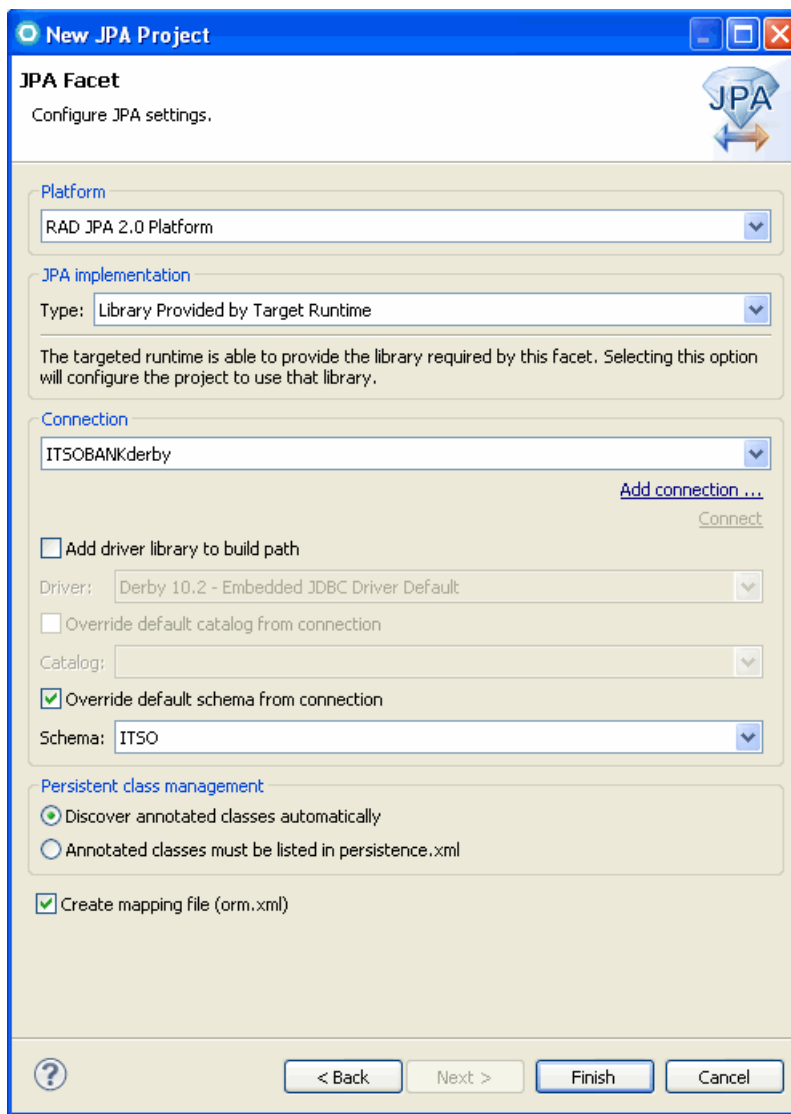


Figure 10-2 JPA Facet window

6. When prompted to switch to the JPA perspective, click **Yes**.

The RAD&JPA project is created. The Technology Quickstart page opens with the title “Help for JPA application development”.

The following files are created in the src/META-INF folder:

- ▶ An empty orm.xml file that can be used for explicit mapping of entities to database tables and to define the schema name
- ▶ A persistence.xml file that defines a persistence unit, in our case RAD8JPA, as shown in Example 10-19

Example 10-19 Persistence-unit name definition

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
  xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="RAD8JPA">
    </persistence-unit>
</persistence>
```

This persistence.xml file will hold the persistent entity classes when the files are created.

10.2.3 Adding JPA support to an existing project

If you have an existing project, you can add JPA support to this faceted project by adding the Java Persistence project facet. It is necessary to have a faceted project, because you cannot add JPA support to a plain Java project. In case you have a Java Project, you must use the functionality to convert a Java project to a JPA project, which is described in 10.2.4, “Converting a Java project to a JPA project” on page 475.

A dynamic web project and an EJB project are faceted projects, for example. A project facet is a specific unit of functionality that you can add to a project when that functionality is required. When a project facet is added to a project, it can add natures, builders, class path entries, and resources to a project, depending on the characteristics of the particular project.

To add JPA support to your existing project, you select this project in the Package Explorer view and perform the following steps:

1. Click **Project** → **Properties** in the main menu, and the Properties page opens.
2. Click the **Project Facets** property.
3. In the list of project facets, select **JPA** and Version **2.0** and then click **Apply**. The Java Persistence 2.0 facet is added to your project.

4. Click **OK**, and the Properties page closes.

Within a web project, the `persistence.xml` file is created in the newly created folder for JPA Content. The `persistence.xml` file describes the definition of the persistence unit with the name of your selected project, as shown in Example 10-19 on page 474.

10.2.4 Converting a Java project to a JPA project

You can convert a plain Java project to a JPA project. You enable a Java project for JPA with the following steps. First, you have to prepare your class that you want to convert to a JPA entity bean:

1. Open the Java source file with the Java editor.
2. Add one of the following JPA annotations before the class declaration:
 - `@Entity`
 - `@Embeddable`
 - `@MappedSuperclass`

When you have added one of the annotations, you can see a problem in this line and a light bulb beside the annotation that you typed. You have to perform the following steps to resolve this problem:

1. Click the **Quick Fix** icon or press `Ctrl+1` to view the suggestion to resolve this problem.
2. Select the first suggestion and click **Add WebSphere Application Server JPA support**.
3. The Add WebSphere Application Server JPA support page opens, as shown in Figure 10-3 on page 476, and you have to select the runtime environment and the version of JPA:
 - a. To define the Targeted Runtime, select **WebSphere Application Server v8.0 Beta**.
 - b. For Java Persistence Version, select **2.0**.
4. Click **OK** and the JPA support is added to your project.

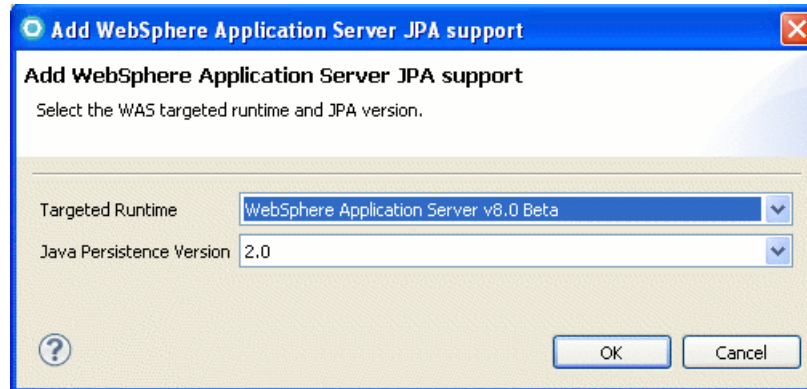


Figure 10-3 Add WebSphere Application Server JPA support window

The update of your project is complete, and the appropriate import statement for `javax.persistence.Entity` is added to the class file. Additionally, the `persistence.xml` file is created, and the class file is added to this `persistence.xml` file. Your project is converted to a JPA project and needs further configuration steps.

Your class file still shows a problem from adding the annotation `@Entity`, because an entity needs the definition of a primary key attribute. To resolve this problem, you can add the `@Id` annotation to one of your existing properties or create a new property as a definition of the primary key. This behavior and the definition of `@Id` annotations are described in “Using `@Id` annotation” on page 446.

Switch to the JPA Development perspective to finish creating the new JPA entity bean, and create additional entities as needed. To change the perspective in the main menu, select **Window** → **Open Perspective** → **JPA**.

10.3 Creating JPA entities

In this section, we develop the JPA entities in multiple ways:

- ▶ Creating a new JPA entity with the wizard
- ▶ Creating a JPA entity when adding persistence to a POJO
- ▶ Generating JPA entities from database tables

Additionally, we show how to use JPA tools to generate data definition language (DDL) files for 10.3.3, “Generating database tables from JPA entities” on page 483.

10.3.1 Creating a new JPA entity with the wizard

To create a JPA entity with the wizard, you select the source folder `/RAD8JPA/scr` and right-click. Then you perform the following steps:

1. Select **New** → **Other**.
2. In the New wizard, select **JPA** → **Entity** and click **Next**.
3. In the New JPA Entity wizard, in the JPA Project window, as shown in Figure 10-4 on page 478, define the entity's details:
 - a. For Project name, select `RAD8JPA`.
 - b. For Source Folder, use default value `/RAD8JPA/scr`, as already defined.
 - c. For Java Package, enter `itso.bank.entities`.
 - d. For Class name, type `Customer`.
 - e. For Inheritance, select **Entity**.
 - f. For XML entity mappings, select **Add to entity mappings in XML**. The mapping file `META-INF/orm.xml` is defined.
 - g. Click **Next**.

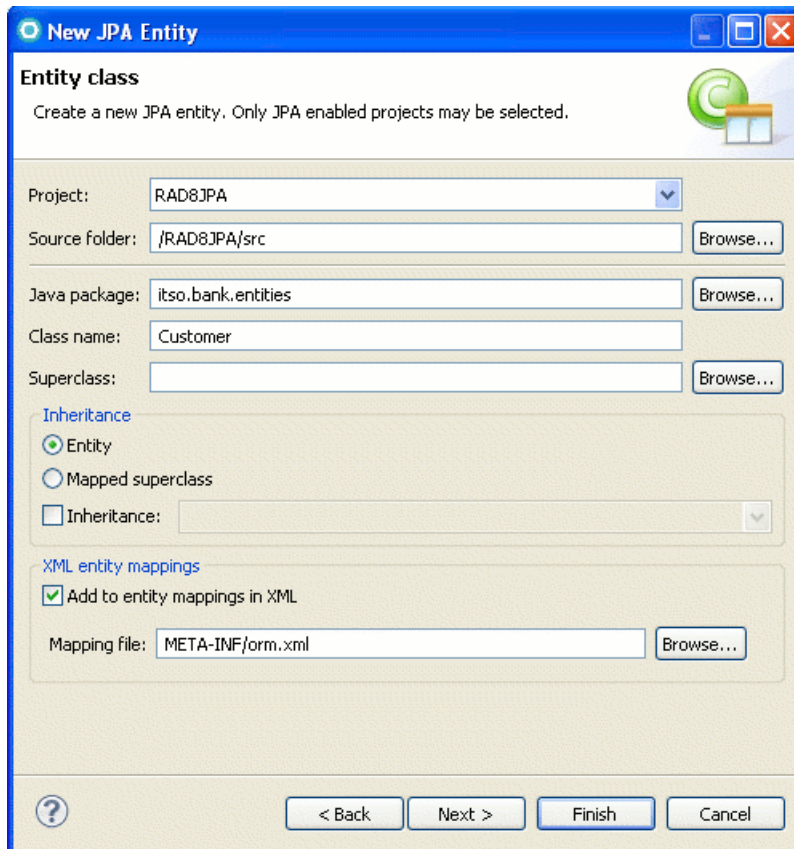


Figure 10-4 New JPA Entity wizard

4. In the New JPA Entity wizard in the Entity Properties window, as shown in Figure 10-5 on page 479, define the entity's properties:
 - a. For Entity name, the value is predefined with Customer, which is the value of the Class name defined before.
 - b. For Table name, select **Use default**. The Table name is same as the Entity name Customer.
 - c. For Entity fields, define new fields using **Add**:
 - i. Type `ssn` for the name and Integer as Type.
 - ii. For at least one field that you define here, you have to select it as **Key** in the first row. Select `ssn` as Key.
 - iii. You can define as many fields as you want within this step.
 - d. For Access type, select **Field** and click **Next**.

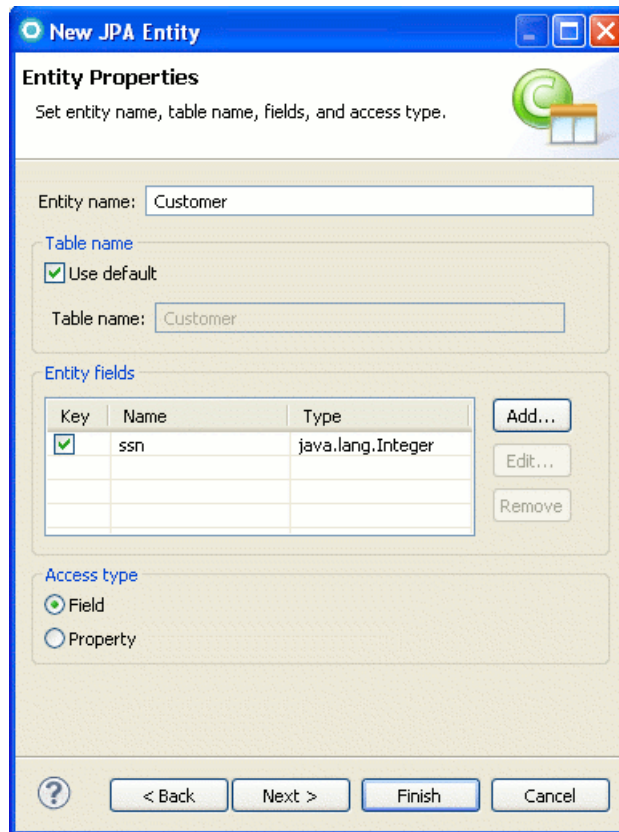


Figure 10-5 JPA Entity Properties window

5. In the New JPA Entity wizard in the Select Class Diagram for Visualization window, select **Add bean to Class Diagram**:
 - You can select the default **classdiagram.dnx** file or another already defined class diagram of your project.
 - You can define a new diagram by using **New**.
6. Click **Finish**.

If you see problems in your newly created JPA entity, select **Project** → **Clean** to start a rebuild from scratch of your project. After this process, your new JPA entity is available without problems.

The Customer JPA entity is created and has to be mapped with the following steps:

1. Open the JPA perspective by clicking **Window** → **Open Perspective** → **JPA**.

2. Double-click the **Customer** class in the Package Explorer view to show the Customer class name in the JPA Structure view.
3. In the JPA Details view, click **here** within the following sentence:
“Type 'Customer' is not mapped, click here to change mapping type.”
4. Select **Entity** in the list of Mapping items and click **OK**.

As result of these steps, in the Customer class, the @Entity annotation is defined over your class definition.

You can use these newly created JPA entities for top-down mapping. Using the JPA tools, you can generate DDL files for creating database tables from entity beans. We describe this method in 10.3.3, “Generating database tables from JPA entities” on page 483.

10.3.2 Creating a JPA entity when adding persistence to a POJO

In this section, we create a JPA entity when adding persistence to a POJO. Therefore, two scenarios are possible. For the first scenario, we assume that a Java object Account already exists in your JPA project. You can use the JPA project, which we created in 10.3.1, “Creating a new JPA entity with the wizard” on page 477, to define the Customer JPA entity. Perform the following steps to create an Account JPA entity and to add persistence to the POJO Account:

1. Open the JPA perspective by clicking **Window** → **Open Perspective** → **JPA**.
2. Right-click the **orm.xml** file in the JPA project in the Package Explorer view and select **Open with** → **Object Relational Mapping XML Editor**.
3. Select **Entity Mappings** in the Overview pane, as shown in Figure 10-6 on page 481.

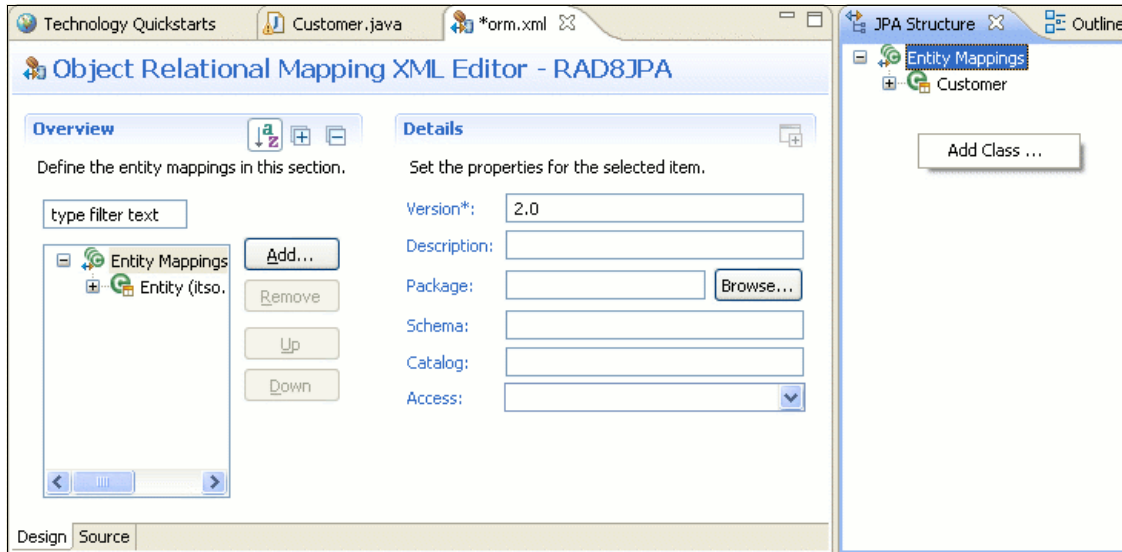


Figure 10-6 Object Relational Mapping XML Editor: Add Class

4. Right-click in the JPA Structure view to get the Add Class option, as shown in Figure 10-6.
5. Click **Add Class**.
6. The Add Class window opens, as shown in Figure 10-7, and you have to define the Account class. You have two options:
 - You type Account or use **Browse** to get a view to select your Account class.
 - For Map as, select **Entity**.
 Click **OK**.

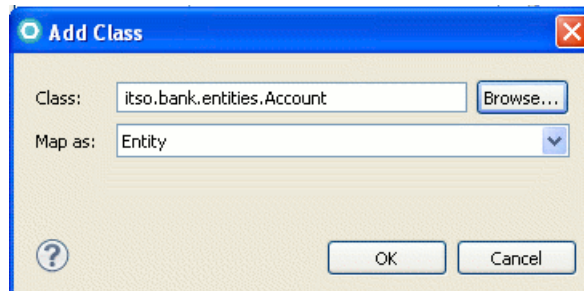


Figure 10-7 Add Class Account

As a result, you see the Account class in the JPA Structure view. There is one problem to resolve, because the Account class has no primary key defined so far. Therefore, perform the following steps:

1. In the JPA Structure view, right-click the property **id** of your added class Account, as shown in Figure 10-8.
2. Click **Add Attribute to XML and Map**. Follow these steps:
 - a. The Add Attribute window opens. Select Map as **ID**.
 - b. Click **OK**.

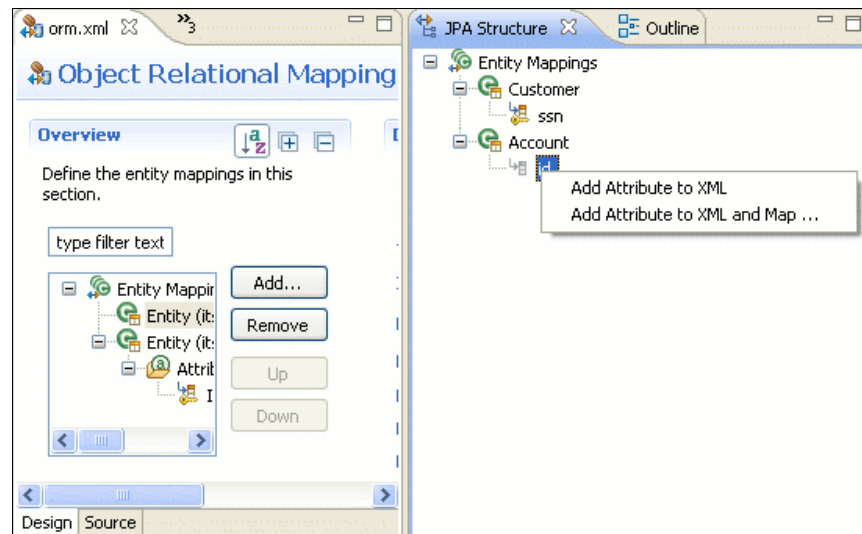


Figure 10-8 Object Relational Mapping XML Editor: Add Attribute

The already existing POJO Account is now a JPA entity.

For the second scenario, see the description in 10.2.4, “Converting a Java project to a JPA project” on page 475, when you add persistence to a plain Java project, while adding persistence to a plain Java class. If your project is already JPA-enabled and you perform the first steps, as defined in 10.2.4, “Converting a Java project to a JPA project” on page 475:

- ▶ Open the Java source file, in this case, your Account class with the Java editor.
- ▶ Add the `@Entity` JPA annotation before the class declaration.

After you have added the annotation, you can see a problem in this line and a light bulb beside the annotation you typed.

You have to perform the following steps to resolve this problem:

1. Click the **Quick Fix** icon or press Ctrl+1 to view the suggestion to resolve this problem.
2. Select the first suggestion and click **Import 'Entity' (javax.persistence)**.

Because you have created an entity with these steps, you have to define a primary key by adding the @Id annotation to your existing property id. This @Id annotation needs to be imported, as well. To organize imports, select **Source** → **Organize Imports**, or press Ctrl+Shift+O.

10.3.3 Generating database tables from JPA entities

In the top-down mapping approach, you start with entity beans and use them to create your database tables. You start from nothing with the entity definitions and the object relational mappings and then you derive database schemas from that data. If you use this approach, you are most likely concerned with creating the architecture of your object model and then writing your entity classes. These entity classes eventually drive the creation of your database model. If you are using a top-down mapping of the object model to the relational model, develop the entity classes and then use the JPA tools' DDL generation capability to create the database tables that are based on the entity classes.

The process of mapping database tables top down from JPA entity beans requires several steps. Therefore, we assume that the following prerequisites exist:

- ▶ A JPA project or enabled JPA support in an appropriate project exists.
- ▶ Created JPA entities exist. You can use the entities that were created in 10.3.1, “Creating a new JPA entity with the wizard” on page 477 and 10.3.2, “Creating a JPA entity when adding persistence to a POJO” on page 480 as a basis to create a DDL file for a JPA project.

To generate the DDL, right-click the JPA project in the Package Explorer view and select **JPA Tools** → **Generate Tables from**.

In the META-INF directory of your JPA project, the file named Table.ddl is created. You can use the Table.ddl file to generate the tables for your entity beans in the database.

10.3.4 Generating JPA entities from database tables

This bottom-up mapping was provided in Rational Application Developer V7.5 and has been improved in Rational Application Developer V8.0. We generate the JPA entities from the ITS0BANK database into the plain JPA project. We use these

entities for additional steps, such as 10.3.5, “Adding business logic” on page 492 and 10.3.6, “Adding named queries” on page 494. To create the entities, perform the following steps:

1. In the Project Explorer, right-click the **RAD8JPA** project and select **JPA Tools** → **Generate Entities from Tables**.
2. In the Generate Custom Entities wizard in the Select Tables window, as shown in Figure 10-9, define the connection, schema, and tables:
 - a. For Connection, select **ITSOBANKderby**.
 - b. For Schema, select **ITSO**.
 - c. To select the four tables, click **Select All**.
 - d. Select **Update class list in persistence.xml** so that the generated classes are added to the file and click **Next**.

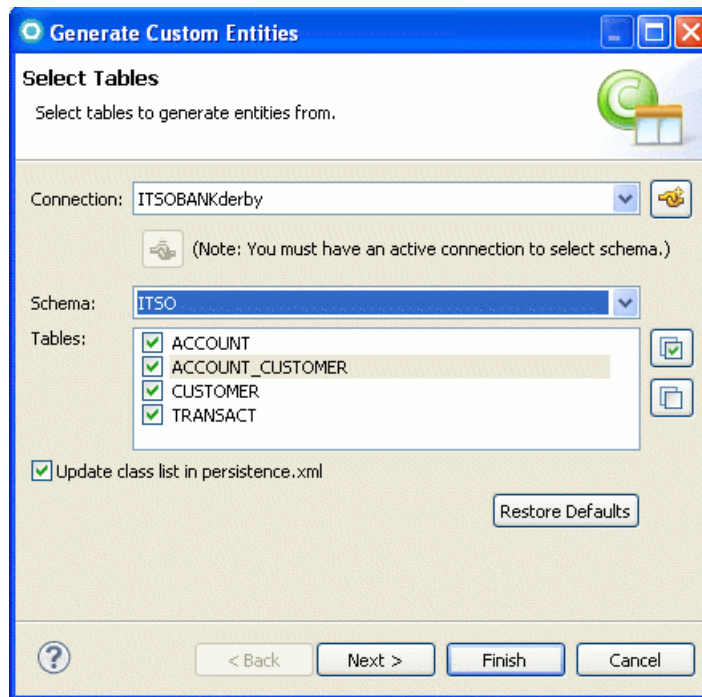


Figure 10-9 Generate Custom Entities: Select Tables window

3. In the Generate Custom Entities wizard in the Table Associations window, two associations are defined, as shown in Figure 10-10. Do not change the definition. Click **Next**.

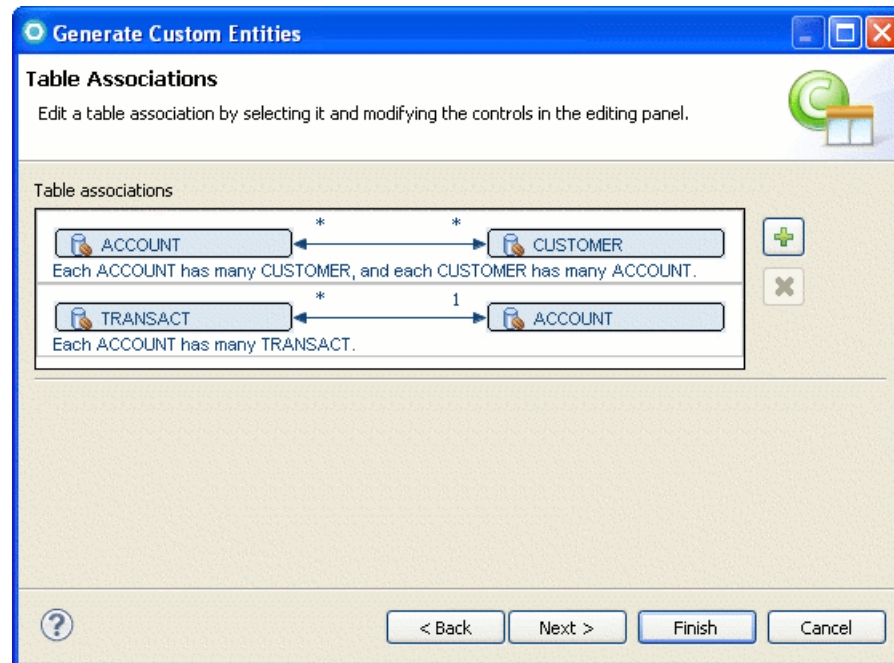


Figure 10-10 Generate Custom Entities: Table Associations window

4. In the Generate Custom Entities wizard in the Customize Default Entity Generation window, as shown in Figure 10-11 on page 486, we define the table mapping and the package name:
 - a. For the Table mapping definition, select Key generator **none**.
 - b. For Entity access, select **Field**.
 - c. For Associations fetch, select **Default**.
 - d. For Collection properties type, select **java.util.List**.
 - e. Clear **Always generate optional JPA annotations and DDL parameters**.
 - f. For Source folder, select **RAD8JPA/src**.
 - g. For Package type, select **itso.bank.entities** and click **Next**.

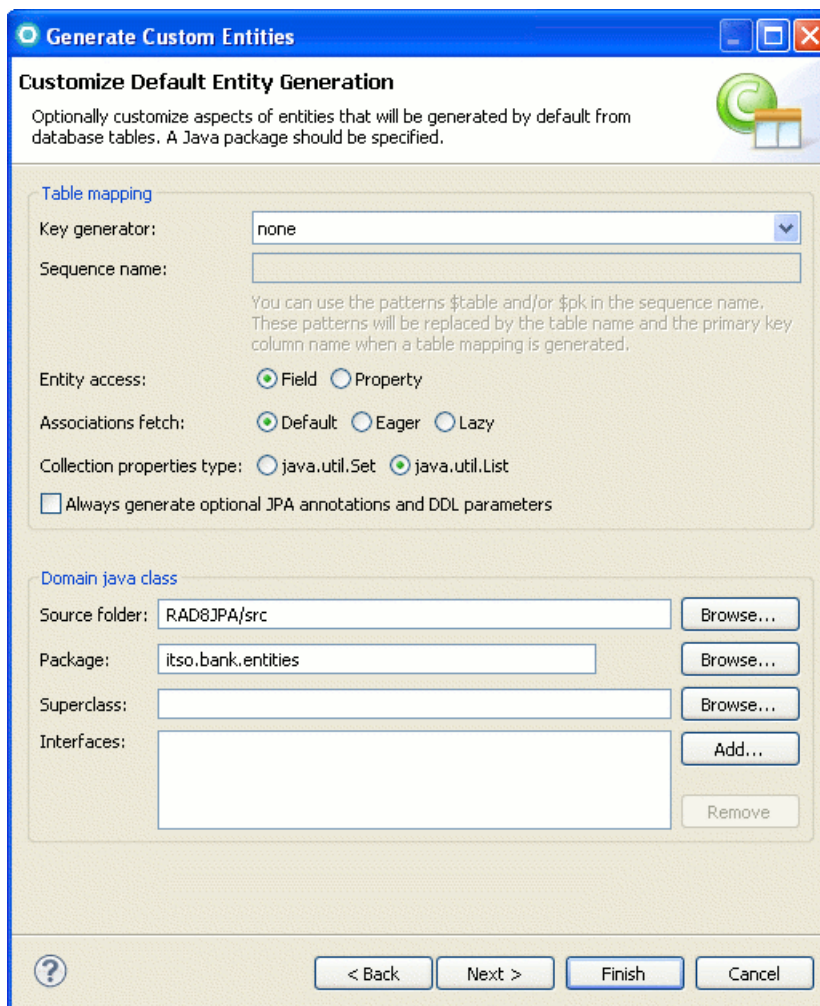


Figure 10-11 Generate Custom Entities: Customize Default Entity Generation

5. In the Generate Custom Entities wizard in the Customize Individual Entities window, as shown in Figure 10-12 on page 487, define the class name:
 - a. Select **TRANSACT** in the Tables and columns pane.
 - b. Class name is defined, by default, with TRANSACT. Overwrite the class name and type Transaction. You only need to change the name of this class.
 - c. Key generator is defined by the default, **none**, because we defined none in the Default Entity Generation window.

- d. For Entity access, select **Field**, because we defined **Field** in the Default Entity Generation window.

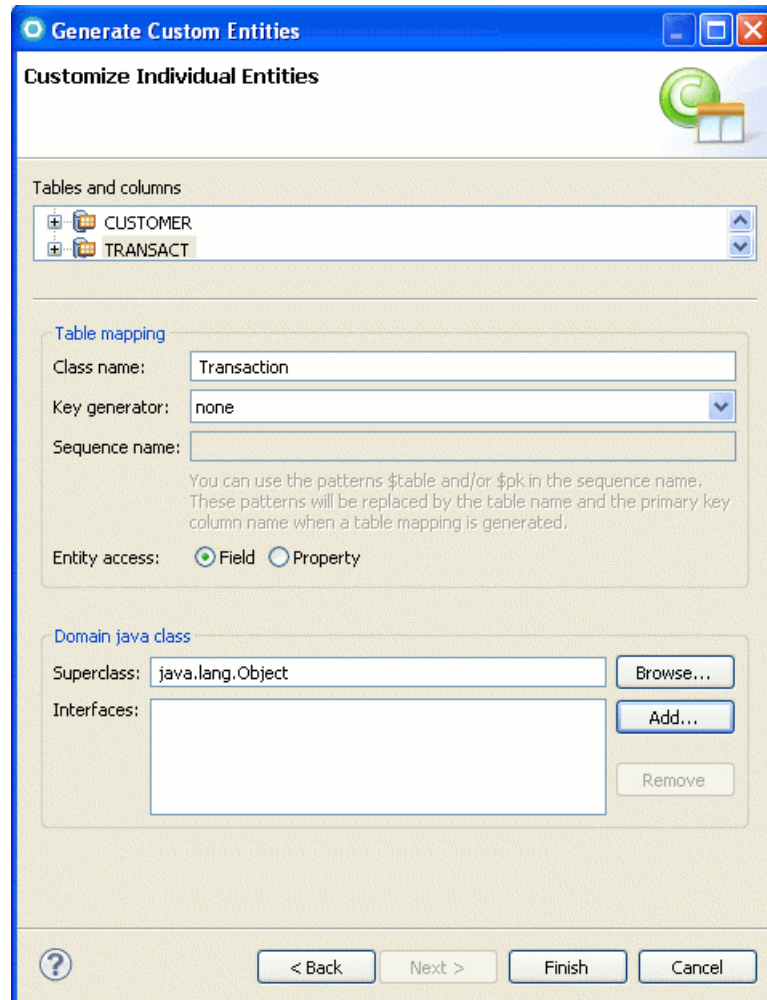


Figure 10-12 Generate Custom Entities: Customize Individual Entities window

6. Click **Finish**.

The `itso.bank.entities` package with three classes is generated, and the three classes are added to the `persistence.xml` file.

Generated JPA entities

Let us study the generated entities.

Account entity

Example 10-20 shows an extract of the Account class.

Example 10-20 Account entity

```
package itso.bank.entities;

import java.io.Serializable;
import java.math.BigDecimal;
import java.util.List;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.OneToOne;

@Entity
public class Account implements Serializable {
    @Id
    private String id;

    private BigDecimal balance;

    @OneToMany(mappedBy="account")
    private List<Transaction> transacts;

    @ManyToMany
    @JoinTable(
        joinColumns=@JoinColumn(name="ACCOUNT_ID"),
        inverseJoinColumns=@JoinColumn(name="CUSTOMER_SSN"))
    private List<Customer> customers;

    .....
    // constructor, getter, setter methods
```

Note the following points:

- ▶ The `@Entity` annotation defined the class as an entity.
- ▶ The `@Id` annotation defines `id` as the primary key.
- ▶ The `@OneToMany` annotation defines the 1:m relationship with `Transaction`. The mapping is defined in the `Transaction` entity. A `List<Transaction>` field holds the related instances.

- ▶ The `@ManyToMany` and `@JoinTables` annotations define the m:m relationship with `Customer`, including the two join columns. A `List<Customer>` field holds the related instances. We have to add the name of the relationship table (`ITSO.ACCOUNT_CUSTOMER`). You will add this name in “Completing the entity classes” on page 490.

Customer entity

Example 10-21 shows an extract of the `Customer` class.

Example 10-21 Customer entity

```
import .....;

@Entity
public class Customer implements Serializable {
    @Id
    private String ssn;

    private String title;

    @Column(name="FIRST_NAME")
    private String firstName;

    @Column(name="LAST_NAME")
    private String lastName;

    @ManyToMany(mappedBy="customers")
    private List<Account> accounts;

    .....
    // constructor, getter, setter methods
```

Note the following points:

- ▶ The `@Entity` annotation defined the class as an entity.
- ▶ The `@Id` annotation defines `ssn` as the primary key.
- ▶ The `@Column` annotation maps the fields to a table column if the names do not match. By convention, column names with underscores (`FIRST_NAME`) create good Java field names (`firstName`).
- ▶ The `@ManyToMany` annotation defines the m:m relationship with `Account`. The mapping is defined in the `Account` entity.

Transaction entity

Example 10-22 on page 490 shows an extract of the `Transaction` class.

Example 10-22 Transaction entity

```
@Entity
@Table(name="TRANSACT")
public class Transaction implements Serializable {
    @Id
    private String id;
    @Column(name="TRANS_TYPE")
    private String transType;

    @Column(name="TRANS_TIME")
    private Timestamp transTime;

    private BigDecimal amount;

    @ManyToOne
    private Account account;
    .....
    // constructor, getter, setter methods
```

Note the following points:

- ▶ The @Entity annotation defined the class as an entity.
- ▶ The @Table annotation defines the mapping to the TRANSACT table. The schema name (ITS0) is missing. The other two classes have no @Table annotation, because the entity name is identical to the table name.
- ▶ The @Id annotation defines id as the primary key.
- ▶ The @Column annotation maps the fields to a table column if the names do not match.
- ▶ The @ManyToOne annotation defines the m:1 relationship with Account. The mapping defaults to account_id as the column name.

Completing the entity classes

The generated entities are missing the table mapping, such as ITS0.CUSTOMER. Without explicit mapping, default table names are assumed, and these default names have the current user ID as the schema name. Make these changes:

- ▶ For the Account entity, we add the @Table annotation to the entity. Within the many-to-many relationship, we add the schema to the @JoinTable annotation, as shown in Example 10-23.

Example 10-23 Complete Account class with @Table annotation

```
@Entity
@Table (schema="ITS0", name="ACCOUNT")
```

```

public class Account implements Serializable {
    .....
    @ManyToOne
    @JoinTable(name="ACCOUNT_CUSTOMER", schema="ITSO",
        joinColumns=@JoinColumn(name="ACCOUNT_ID"),
        inverseJoinColumns=@JoinColumn(name="CUSTOMER_SSN"))
    private List<Customer> customers;

```

- ▶ For the Customer entity, we add the @Table annotation, as shown in Example 10-24.

Example 10-24 Complete Customer class with @Table annotation

```

@Entity
@Table(schema="ITSO", name="CUSTOMER")
public class Customer implements Serializable {
    ...
}

```

- ▶ For the Transaction entity, we add the schema to the @Table annotation, and we add the @ForeignKey annotation to the relationship with Account, as shown in Example 10-25.

Example 10-25 Complete Transaction class with schema in @Table annotation

```

@Entity
@Table(schema="ITSO", name="TRANSACTION")
public class Transaction implements Serializable {
    .....
    @ManyToOne
    @ForeignKey
    private Account account;
    ...
}

```

Resolve the missing import statement by selecting **Source** → **Organize Imports**.

Verifying the persistence.xml file

Open the **persistence.xml** file and verify that the three classes have been added, as shown in Example 10-26.

Example 10-26 persistence.xml with new classes

```

<persistence version="2.0" .....>
  <persistence-unit name="RAD&JPA">

```

```
<class>
  itso.bank.entities.Account</class>
<class>
  itso.bank.entities.Customer</class>
<class>
  itso.bank.entities.Transaction</class>
</persistence-unit>
</persistence>
```

10.3.5 Adding business logic

We want to add business logic, so that an account balance can only be changed through a deposit or withdrawal of funds. In addition, a deposit or withdrawal must create a transaction record.

Transaction class

To create transaction records, define the possible values for transaction type (credit and debit) and add a constructor with parameters:

1. Define two constants in the Transaction class:

```
public static final String DEBIT = "Debit";
public static final String CREDIT = "Credit";
```

2. Add a constructor that sets the id to a universally unique identifier (UUID) and the transTime to the current time stamp, as shown in Example 10-27.

Example 10-27 Transaction constructor

```
public Transaction(String transType, BigDecimal amount) {
    super();
    setId(java.util.UUID.randomUUID().toString());
    setTransType(transType);
    setAmount(amount);
    setTransTime( new Timestamp(System.currentTimeMillis()) );
}
```

Key for transaction objects: Transaction objects must have a unique key, and the time stamp (transTime field) is not unique on fast machines. Therefore, we create a UUID using a Java utility class.

Account class

First, we change the constructor to initialize the balance:

1. Open the **Account** class and change the code of the constructor:

```
public Account() {
    super();
    setBalance(new BigDecimal(0.00));
}
```

2. Make the setBalance method private, so that it cannot be used by clients:

```
private void setBalance(BigDecimal balance) {
    this.balance = balance;
}
```

3. Add a processTransaction method that performs the credit or debit of funds and creates a transaction instance, as shown in Example 10-28.

Example 10-28 Processing credit and debit transactions

```
public Transaction processTransaction
    (BigDecimal amount, String transactionType) throws
Exception {
    if (Transaction.CREDIT.equals(transactionType)) {
        balance = balance.add(amount);
    } else if (Transaction.DEBIT.equals(transactionType)) {
        if (balance.compareTo(amount) < 0)
            throw new Exception("Not enough funds for DEBIT of " +
amount);
        balance = balance.subtract(amount);
    } else
        throw new Exception("Invalid transaction type");
    Transaction transaction = new Transaction(transactionType,
amount);
    transaction.setAccount(this);
    return transaction;
}
```

Notice that the method verifies that enough funds are available for withdrawal (debit). After adjusting the balance, a transaction instance is created, and the account is set into the new transaction.

4. Resolve the missing imports by selecting **Source** → **Organize Imports** or pressing Ctrl+Shift+O.

10.3.6 Adding named queries

Named queries provide additional functionality, such as retrieving all the instances of a class, or retrieving related instances by following a relationship and sorting the results. When following the relationships through the generated collection (`List`), the related instances can be in any order.

Named queries are based on the entity attributes, not on the column names in the mapped table.

Customer class

For the `Customer` class, to retrieve all customers, retrieve a customer by Social Security number (SSN), retrieve a customer by partial last name, and to retrieve the list of accounts of a customer sorted by the account ID, follow these steps:

1. Open the **Customer** class.
2. Add an `@NamedQueries` annotation that defines four named queries for the `Customer`, as shown in Example 10-29.

Example 10-29 Named queries for Customer

```
@Entity
@Table (schema="ITS0", name="CUSTOMER")
@NamedQueries({
    @NamedQuery(name="getCustomers",
        query="select c from Customer c"),
    @NamedQuery(name="getCustomerBySSN",
        query="select c from Customer c
            where c.ssn =:ssn"),
    @NamedQuery(name="getCustomersByPartialName",
        query="select c from Customer c
            where c.lastName like :name"),
    @NamedQuery(name="getAccountsForSSN",
        query="select a from Customer c, in(c.accounts) a
            where c.ssn =:ssn order by a.id")
})
public class Customer implements Serializable {
    ...
}
```

The four queries enable these functions:

- | | |
|-------------------------|---|
| getCustomers | Retrieve all customers |
| getCustomerBySSN | Retrieve the customer for a given SSN, as an alternative to the Entity Manager <code>find</code> method |

getCustomerByPartialName

Retrieve a list of customers by partial last name

getAccountsForSSN

Retrieve the accounts of one customer

The last query illustrates how to follow a relationship (accounts) from a customer to the related accounts and sort them by their ID.

3. Always organize imports. Select **Source** → **Organize Imports** or press Ctrl+Shift+O.

Account class

In the Account class, add two named queries to retrieve the accounts for a customer and to retrieve the transactions for an account. Both queries follow the relationships defined in the Account class. Follow these steps:

1. Open the **Account** class.
2. Add two named queries for Account, as shown in Example 10-30.

Example 10-30 Named queries for Account

```
@Entity
@Table (schema="ITSO", name="ACCOUNT")
@NamedQueries({
    @NamedQuery(name="getAccountsBySSN",
        query="select a from Account a, in(a.customers) c
            where c.ssn =:ssn order by a.id"),
    @NamedQuery(name="getTransactionsByID",
        query="select t from Account a, in(a.transacts) t
            where a.id =:aid order by t.transTime")
})
public class Account implements Serializable {
    ...
}
```

We use either the getAccountsForSSN query defined in the Customer class, or the getAccountsBySSN query defined in the Account class, to retrieve the accounts of a customer.

10.4 Creating a JPA Manager Bean

To create the JPA Manager Beans, we use the JPA project, which includes the entities, created by the bottom-up method described in 10.3.4, “Generating JPA entities from database tables” on page 483.

Perform the following steps to generate one JPA Manager Bean for each JPA entity Account, Customer, and Transaction:

1. In the Project Explorer, right-click the **RAD8JPA** project and select **JPA Tools** → **Add JPA Manager beans**.
2. In the JPA Manager Bean Wizard window, as shown in Figure 10-13, the list of Available JPA Entities is given. Click **Select All** to define the list of account, customer, and transaction entities to use for JPA Manager Bean generation. Click **Next**.

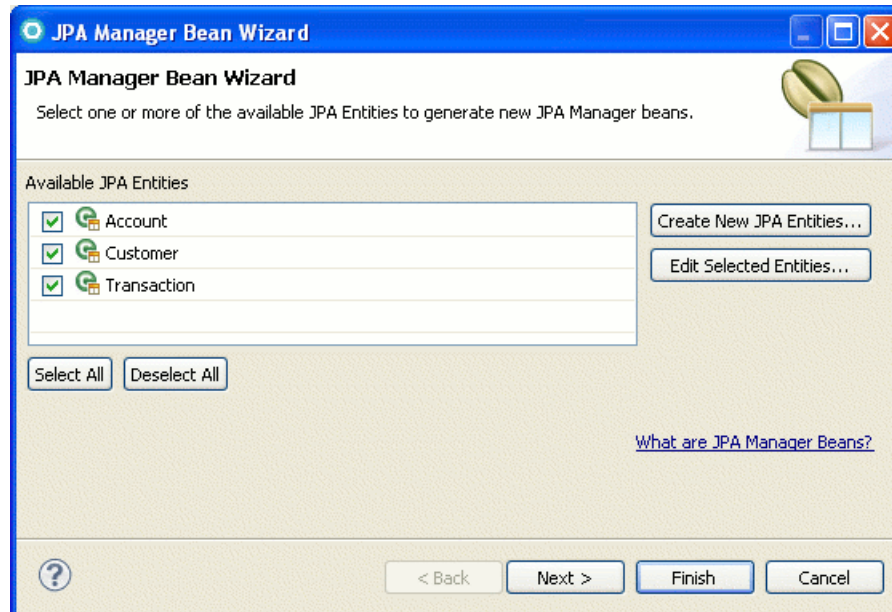


Figure 10-13 JPA Manager Bean Wizard

3. In the JPA Manager Bean Wizard in the Tasks window, as shown in Figure 10-14 on page 497, you can define Query methods for each JPA entity:
 - a. The previously defined NamedQueries are visible for the Account and Customer entities.
 - b. The query methods can be Edit or Remove, or you can Add a new method.

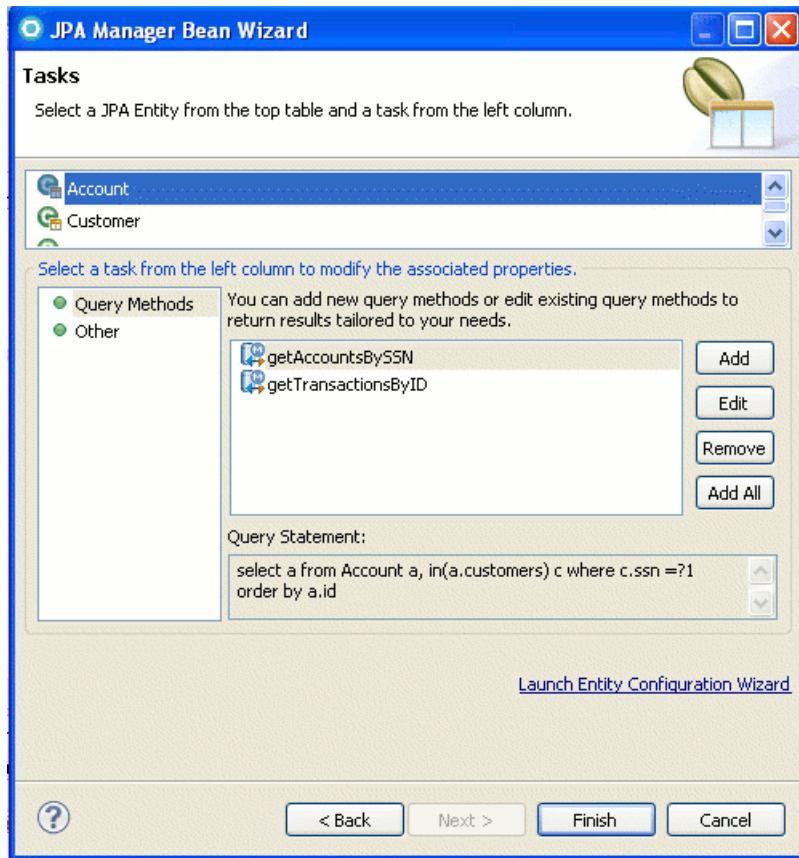


Figure 10-14 JPA Manager Bean Wizard: Tasks

4. In the Other tab, accept the default choice: **I want to manage the persistence unit myself**, as shown in Figure 10-15 on page 498.

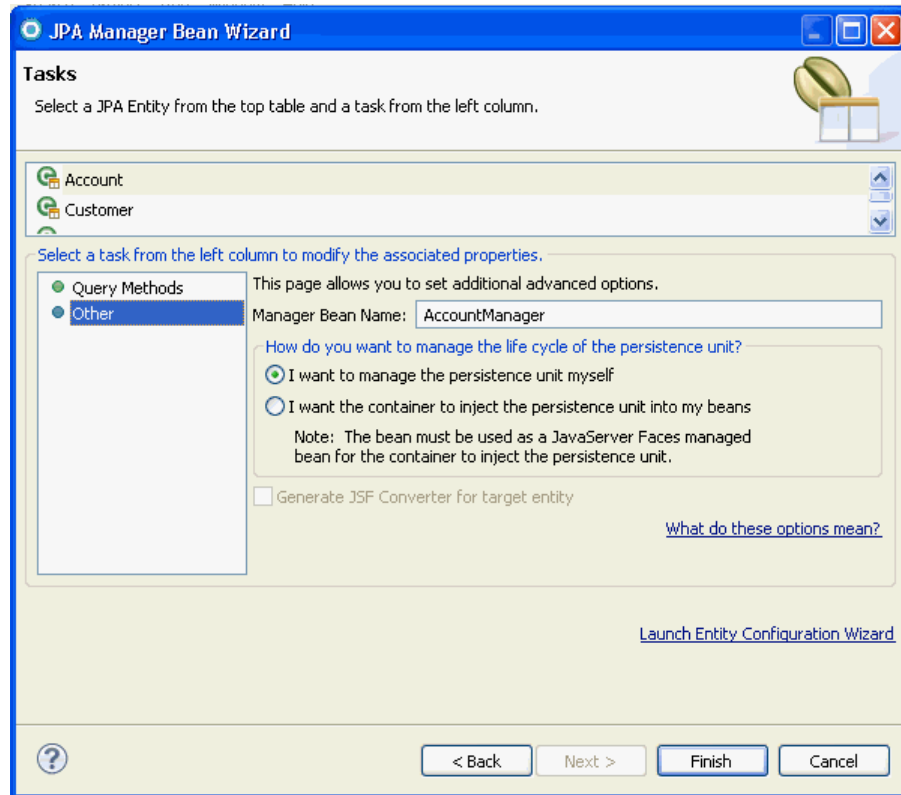


Figure 10-15 JPA Manager Tasks: Other

5. Click **Finish** without any change to the tasks.

The `itso.bank.entities.controller` package, with three classes `AccountManager`, `CustomerManager`, and `TransactionManager`, is generated.

Be aware that the return value of method `getTransactionsByID` in `AccountManager` has generated as `Account`. You have to change this value to `Transaction`. The same change is necessary for method `getAccountsForSSN` in `CustomerManager`. You have to change the return value from `Customer` to `Account`.

10.5 Visualizing JPA entities

A Unified Modeling Language (UML) diagram can visualize JPA entities and their relationships. You can create entities from the diagram, or you can use the UML diagram to visualize existing entities.

To create a UML class diagram from the generated JPA entities:

1. Right-click the **RAD8JPA** project and select **New** → **Class Diagram**.
2. Accept the default name (classdiagram) and click **Finish**.
3. When prompted to enable modeling capabilities, click **OK**.
4. When the class diagram opens, expand the Palette and JPA drawer, as shown in Figure 10-16.

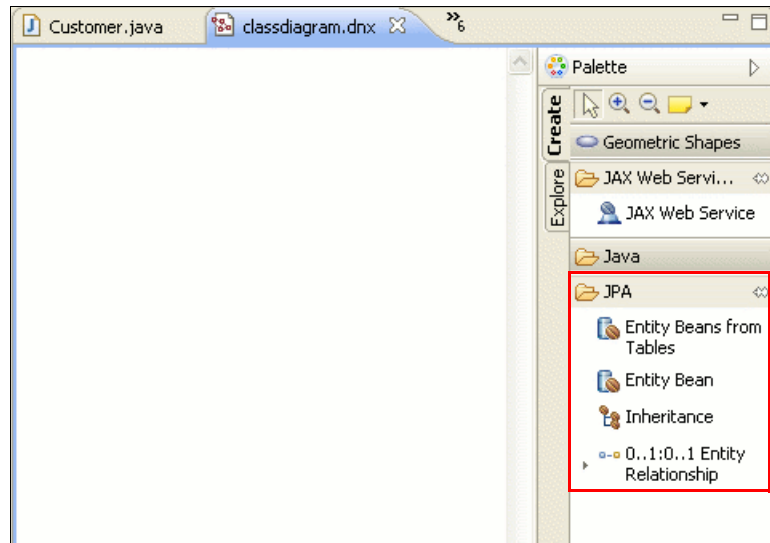


Figure 10-16 Class diagram with JPA Palette

5. Drag the **Customer** class to the diagram, as shown in Figure 10-17 on page 500.

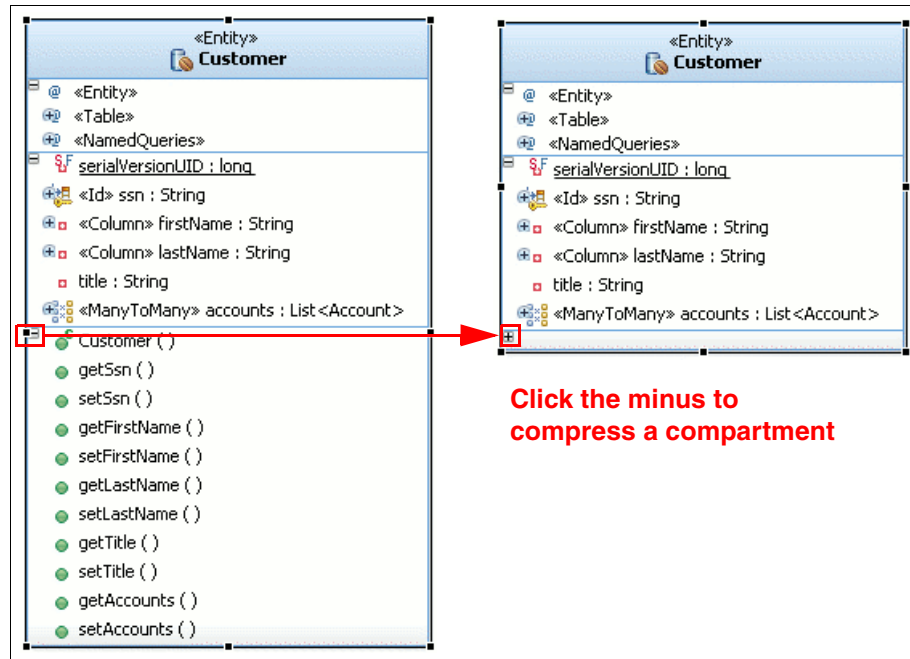


Figure 10-17 Customer class visualized

- Drag the **Account** and **Transaction** classes to the diagram. Select and drag the **<<use>>** arrows to separate them, as shown in Figure 10-18.

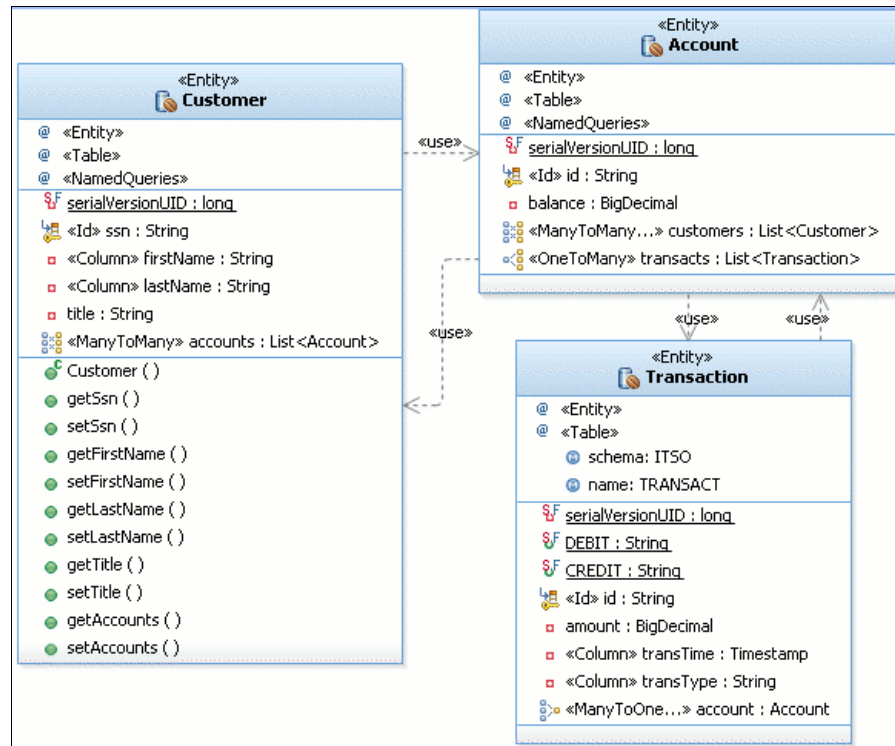


Figure 10-18 JPA class diagram

- Save and close the diagram.

10.6 Testing JPA entities

One benefit of JPA entities over container-managed persistence (CMP) beans is that they can be tested outside of a WebSphere Application Server using a Java class with a `main` method. JPA entities can also be tested using the JUnit framework. See Chapter 26, “Testing using JUnit” on page 1365, for more information about JUnit. Additionally, the Universal Test Client (UTC) can test the entities. Therefore, the UTC uses the WebSphere Application Server JPA containers to test the entities. You have to click the JPA Explorer in the UTC and define the Persistence unit name, such as `RAD8JPA` for our example, as shown in Figure 10-19 on page 502.

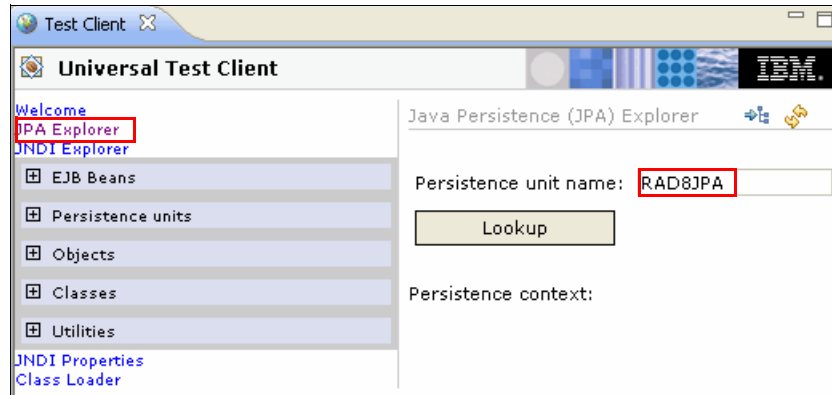


Figure 10-19 Universal Test Client: JPA Explorer to test JPA entities

In this section, we describe the approach to test the entities outside of a WebSphere Application Server. Therefore, we create an independent project called RAD8JPATest that links to the JPA project RAD8JPA. To run JPA outside of the server, we must use the OpenJPA implementation and not the JPA implementation of the server. This method requires making modifications to the persistence.xml file, as defined in 10.6.4, “Setting up the persistence.xml file” on page 508.

10.6.1 Creating the Java project for entity testing

To test the JPA entities, we use a simple Java project. Therefore, perform the following steps to create it:

1. In the Java perspective, right-click in the Package Explorer and select **New** → **Java Project**.
2. In the Create a Java Project window, for Project name, type RAD8JPATest. Accept all the defaults and click **Next**.
3. In the Java Settings window, select the **Projects** tab and click **Add**. Select the **RAD8JPA** project and click **OK**.
4. Click **Finish**.
5. When prompted to switch to the Java perspective, click **Yes**.

10.6.2 Creating a Java class for entity testing

To create an EntityTester class with a main method, follow these steps:

1. Right-click the **RAD8JPATest** project and select **New** → **Class**.

2. In the Java Class window, follow these steps:
 - a. For Package, type `itso.bank.entities.test`.
 - b. For Name, type `EntityTester`.
 - c. For Which method stubs would you like to create, select **public static void main(String[] args)**.
 - d. Click **Finish**.

The `EntityTester` class opens in the editor.

10.6.3 Setting up the build path for OpenJPA

Because this class runs outside of the server, you must add the required JPA and server libraries to the build path:

1. Right-click the **RAD8JPATest** project and select **Properties**.
2. In the Properties window, in the left navigation pane, select **Java Build Path**. In the right pane, select the **Source** tab. Change the output folder from `RAD8JPATest/bin` to `RAD8JPATest/src`, as shown in Figure 10-20 on page 504. Without this change, your created `persistence.xml` file cannot be found.

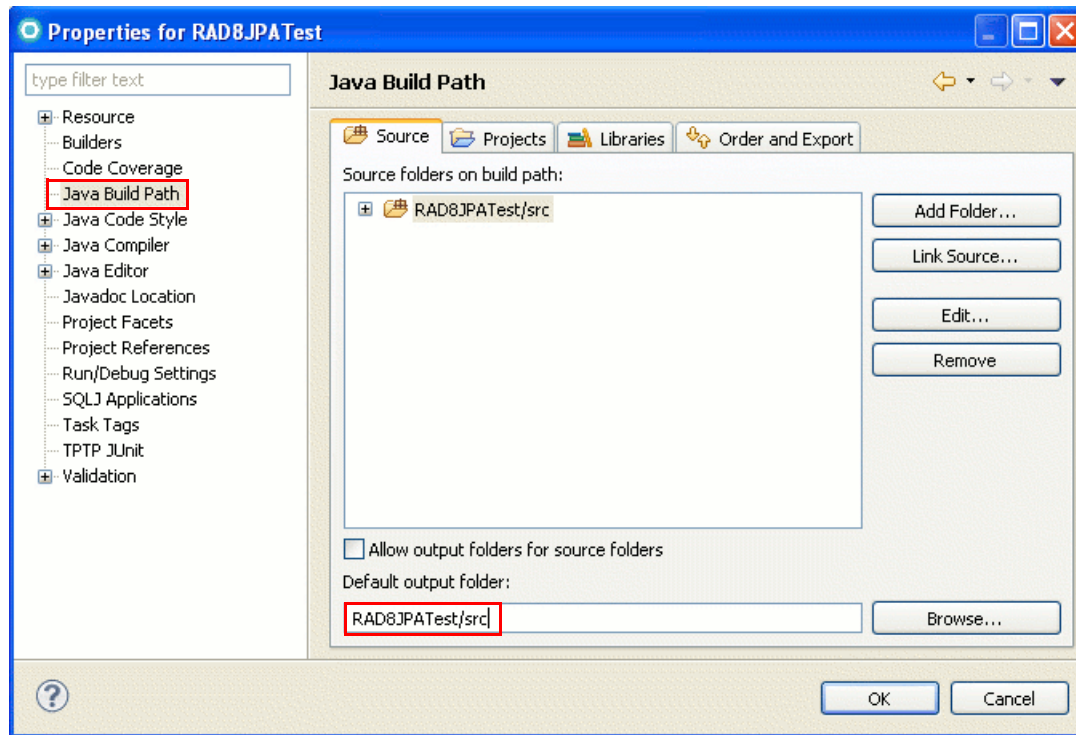


Figure 10-20 Java Build Path of Java project: Source tab

3. Select the **Libraries** tab, as shown in Figure 10-21, and click **Add Variable**.

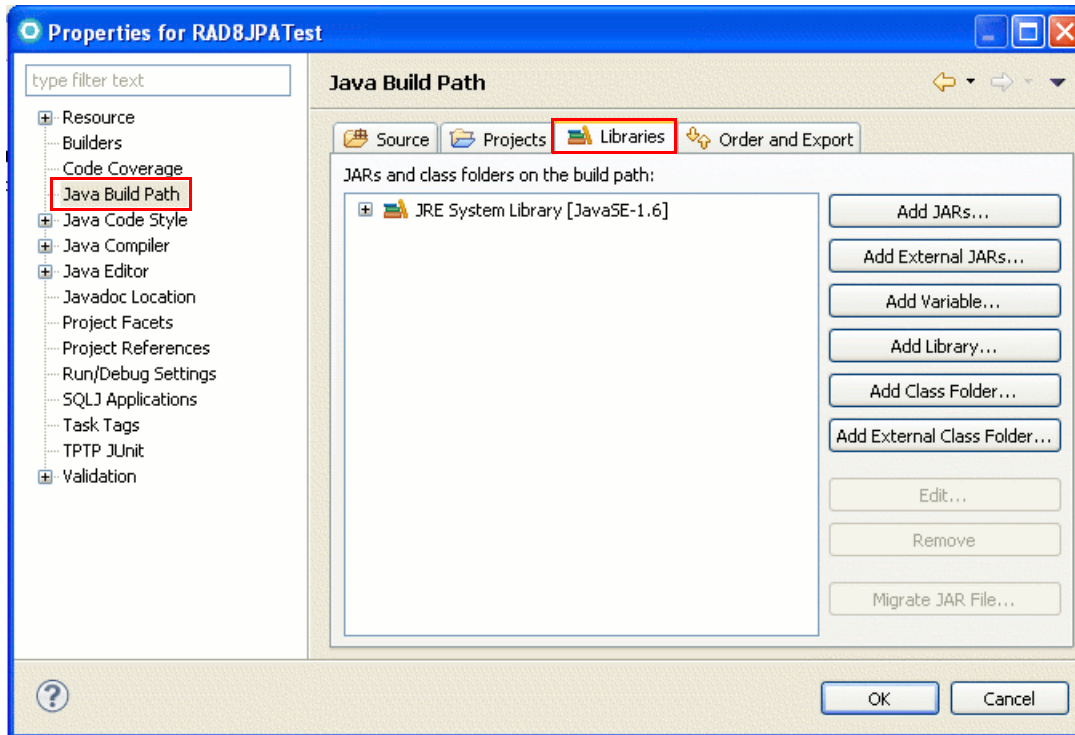


Figure 10-21 Java Build Path of Java project: Libraries tab

4. In the New variable Classpath Entry window that is shown in Figure 10-22, select **ECLIPSE_HOME** and click **Extend**.

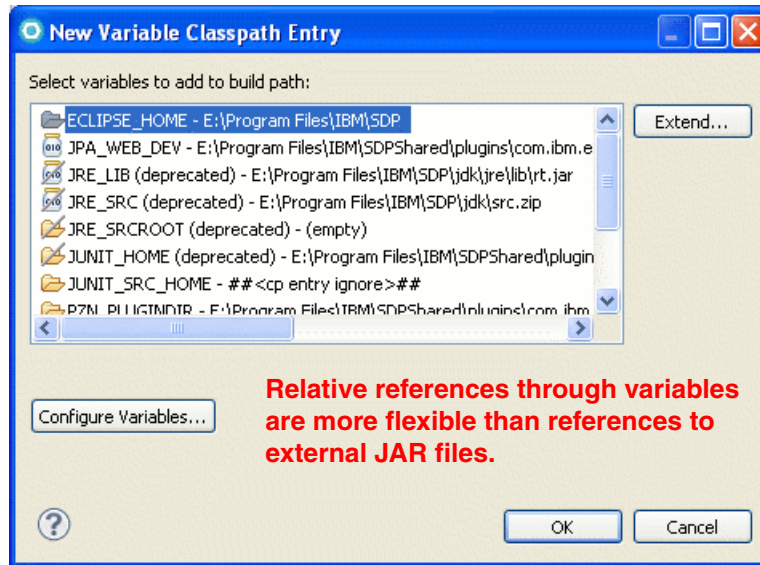


Figure 10-22 Extending a variable

5. In the Variable Extension window that is shown in Figure 10-23, expand **runtimes** → **base_v8_stub** → **lib**, select **j2ee.jar**, and click **OK**.

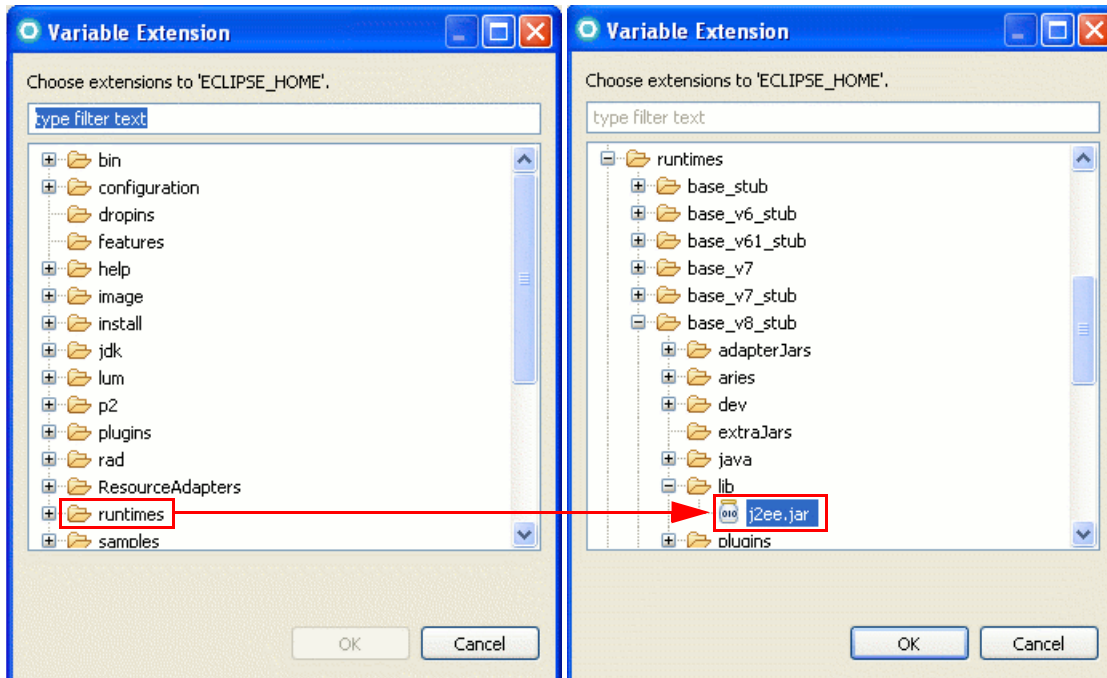


Figure 10-23 Selecting a runtime JAR file

- e. If you have no stand-alone WebSphere Application Server installed, you have to repeat this sequence by extending the *ECLIPSE_HOME* variable and select the following JAR files:
- `runtimes/base_v7/derby/lib/derby.jar`
 - `runtimes/base_v8_stub/plugins/com.ibm.ffdc.jar`
 - `runtimes/base_v8_stub/plugins/com.ibm.ws.jp.a.jar`
 - `runtimes/base_v8_stub/plugins/com.ibm.ws.prereq.commons-collecti ons.jar`

Licensing: Make sure that you check the license condition before you use these JAR files:

- ▶ `runtimes/base_v7/derby/lib/derby.jar`
- ▶ `runtimes/base_v8_stub/plugins/com.ibm.ffdc.jar`
- ▶ `runtimes/base_v8_stub/plugins/com.ibm.ws.jpa.jar`
- ▶ `runtimes/base_v8_stub/plugins/com.ibm.ws.prereq.commons-collections.jar`

If you define these JAR files, you have to skip step 4. Step 4 describes how to add the external `com.ibm.ws.jpa.thinclient_8.0.0.jar`, which is included in the WebSphere Application Server environment.

6. Click **Add External JARs** in your already selected the **Libraries** tab. Select the `com.ibm.ws.jpa.thinclient_8.0.0.jar` library, which is available in your WebSphere Application Server installation folder, for example, `C:\Programs\IBM\WebSphere\AppServer\runtimes`. Click **Open**.

Now the Libraries tab shows the additional JAR files, as shown in Figure 10-24.

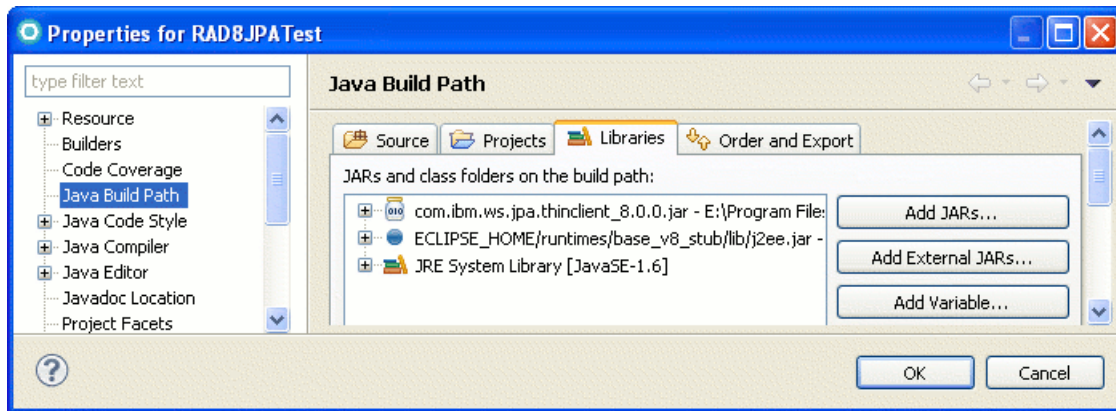


Figure 10-24 Library tab with extra JAR files

7. Click **OK** to close the Properties window and save changes.

10.6.4 Setting up the `persistence.xml` file

The `persistence.xml` file is used to configure the OpenJPA implementation. Because we want to use the RAD8JPA project later in the WebSphere Application

Server server, we do not want to change that file. We can create a similar file in the RAD8JPATest project, so that it overwrites the file in the RAD8JPA project:

1. Right-click the **src** folder in the RAD8JPATest project and select **New** → **Folder**. For Folder name, type META-INF and click **Finish**.
2. Copy the persistence.xml file from the RAD8JPA project to the META-INF folder of the RAD8JPATest project.
3. Open the **persistence.xml** file.
4. In the editor, which is shown in Example 10-31, complete the following actions:
 - a. Add transaction-type to the `<persistence-unit>` tag.
 - b. Remove the `<jta-data-source>` tag.
 - c. Add a `<provider>` tag.
 - d. Add four properties to set up the connection to the ITSOBANK database and for logging.

Example 10-31 Persistence.xml file for OpenJPA

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence .....
<persistence-unit name="RAD8JPA"
transaction-type="RESOURCE_LOCAL"

<provider>org.apache.openjpa.persistence.PersistenceProviderImpl
  </provider>
  <class>itso.bank.entities.Account</class>
  <class>itso.bank.entities.Customer</class>
  <class>itso.bank.entities.Transaction</class>
  <properties>
    <property name="openjpa.ConnectionURL"
      value="jdbc:derby:C:\7835code\database\derby\ITSOBANK"
    />
    <property name="openjpa.ConnectionDriverName"
      value="org.apache.derby.jdbc.EmbeddedDriver" />
    <property name="openjpa.ConnectionUserName" value="itso"
  />
    <property name="openjpa.Log" value="none" />
  </properties>
</persistence-unit>
</persistence>
```

You can copy and paste the code from the 7835code\jpa\test\persistence.xml file.

DB2 for JPA entries: To use DB2 for the JPA entities, add the following JAR files to the Libraries page:

- ▶ <SQLLIB-HOME>/java/db2jcc.jar
- ▶ <SQLLIB-HOME>/java/db2jcc_license_cu.jarr

Change the JPA properties in the persistence.xml file:

```
<property name="openjpa.ConnectionURL"
           value="jdbc:db2://localhost:50000/ITS0BANK" />
<property name="openjpa.ConnectionDriverName"
           value="com.ibm.db2.jcc.DB2Driver" />
<property name="openjpa.ConnectionUserName" value="db2admin" />
<property name="openjpa.ConnectionPassword" value="<password>" />
<property name="openjpa.Log" value="none" />
```

10.6.5 Creating the test

To test the JPA entities, complete the main method in the EntityTester class, as shown in Example 10-32:

1. Copy and paste the code from the 7835code\jpa\test\EntityTester.java file.
2. Select **Source** → **Organize Imports**.
3. Resolve the following items:

```
java.math.BigDecimal
javax.persistence.Query
itso.bank.entities.Transaction
java.util.List
```

Example 10-32 Testing JPA entities using a Java program

```
public class EntityTester {

    static EntityManager em;

    public static void main(String[] args) {
        String customerId = "111-11-1111";
        if (args.length > 0) customerId = args[0];
        System.out.println("Entity Testing");
        System.out.println("\nCreating EntityManager");
        em = Persistence.createEntityManagerFactory("RAD8JPA")
            .createEntityManager();
    }
}
```

```

        System.out.println("RAD8JPA EntityManager successfully
created\n");
        em.getTransaction().begin();

        System.out.println("\nAll customers: ");
        Query query1 = em.createNamedQuery("getCustomers");
        List<Customer> custList1 = query1.getResultList();
        for (Customer cust : custList1) {
            System.out.println(cust.getSsn() + " " + cust.getTitle() +
" "
                                + cust.getFirstName() + " " +
cust.getLastName());
        }
        System.out.println("\nCustomers by partial name: a");
        Query query2 =
em.createNamedQuery("getCustomersByPartialName");
        query2.setParameter(name, "%a%");
        List<Customer> custList2 = query2.getResultList();
        for (Customer cust : custList2) {
            System.out.println(cust.getSsn() + " " + cust.getTitle() +
" "
                                + cust.getFirstName() + " " +
cust.getLastName());
        }

        System.out.println("\nRetrieve one customer: " + customerId);
        Customer cust = em.find(Customer.class, customerId);
        System.out.println(cust.getSsn() + " " + cust.getTitle() + " "
                                + cust.getFirstName() + " " +
cust.getLastName());

        List<Account> acctSet = cust.getAccounts();
        System.out.println("Customer has " + acctSet.size() + "
accounts");
        for (Account account : acctSet) {
            System.out.println("Account: " + account.getId() + "
balance "
                                + account.getBalance());
        }
        System.out.println
            ("\nRetrieve customer accounts sorted using named
query:");
        Query query3 = em.createNamedQuery("getAccountsBySSN");
        query3.setParameter(ssn, cust.getSsn());
        List<Account> acctList = query3.getResultList();

```

```

        for (Account account : acctList) {
            System.out.println("Account: " + account.getId() + "
balance "
                                + account.getBalance());
        }

        System.out.println("\nPerform transactions on one account: "
);
        Account account = acctList.get(0);
        System.out.println("Account: " + account.getId() + " balance "
                                + account.getBalance());

        Transaction tx = null;
        try {
            BigDecimal balance = account.getBalance();
            tx = account.processTransaction(new BigDecimal(100.00),
"Credit");
            em.persist(tx); // make insert persistent
            System.out.println("Tx created: " + tx.getAccount().getId() +
" "
                                + tx.getTransType() + " " + tx.getAmount() +
" "
                                + tx.getTransTime() + " id " + tx.getId());
            tx = account.processTransaction(new BigDecimal(50.00),
"Debit");
            em.persist(tx);
            System.out.println("Tx created: " + tx.getAccount().getId() +
" "
                                + tx.getTransType() + " " + tx.getAmount() +
" "
                                + tx.getTransTime() + " id " +
tx.getId());
            tx = account.processTransaction(balance.add(new
                                BigDecimal(200.00)),
"Debit");
            em.persist(tx);
        } catch (Exception e) {
            System.out.println("Transaction failed: " +
e.getMessage());
        }

        em.flush(); // make inserts persistent in the DB
        em.refresh(account); // retrieve account again to access
                                transactions

```

```

        System.out.println("\nAccount: " + account.getId() + " balance
"
                                +
account.getBalance());

        Query query4 = em.createNamedQuery("getTransactionsByID");
        query4.setParameter(aid, account.getId());
        List<Transaction> transList = query4.getResultList();
        System.out.println("Account has " + transList.size()
            +" transactions");
        for (Transaction tran : transList) {
            System.out.println("Transaction: " + tran.getTransType() +
" "
                + tran.getAmount() + " " + tran.getTransTime() + " id "
                + tran.getId());
        }
        em.getTransaction().commit ();
    }
}

```

Understanding the entity testing code

In the following sequence, we demonstrate how to work with entities:

1. We require an Entity Manager (em) for the RAD8JPA persistence unit:

```

static EntityManager em;
.....
em = Persistence.createEntityManagerFactory("RAD8JPA")
                .createEntityManager();

```

2. We start a transaction (this step is not required for read-only access):

```
em.getTransaction().begin();
```

3. We retrieve all the customers with the `getCustomers` named query. A named query with multiple results returns a list:

```

Query query1 = em.createNamedQuery("getCustomers");
List<Customer> custList1 = query1.getResultList();

```

4. We use the Java EE 5 support for iterating through a list:

```
for (Customer cust : custList1) { .... }
```

5. We use the `getCustomersByPartialName` named query to retrieve customers with the letter a in the last name. This query illustrates how to set a parameter in the query:

```

Query query2 = em.createNamedQuery("getCustomersByPartialName");
query2.setParameter(name, "%a%");

```

6. We use the accounts relationship in the Customer class to list the related accounts. When we list the accounts, they are in any order:

```
List<Account> acctSet = cust.getAccounts();
```

7. We use the getAccountsBySSN named query to retrieve the related accounts in sorted order:

```
Query query3 = em.createNamedQuery("getAccountsBySSN");
    query3.setParameter(ssn, cust.getSsn());
    List<Account> acctList = query3.getResultList();
```

8. We process the transactions on one account. The last transaction fails, because the amount is larger than the balance:

```
tx = account.processTransaction(new BigDecimal(100.00), "Credit");
tx = account.processTransaction(new BigDecimal(50.00), "Debit");
tx = account.processTransaction( balance.add(new
                                BigDecimal(200.00)), "Debit");
```

9. We have to persist each new transaction:

```
em.persist(tx);
```

10. We want to retrieve the account and see all the transactions. The flush method writes the updates to the database, and the refresh method refreshes the account in memory:

```
em.flush();           // make inserts persistent in the DB
em.refresh(account); // retrieve account again to access transactions
```

11. We list the transactions of the account in time stamp sequence using the getTransactionsByID named query. The returned transactions transacts are in sequence:

```
Query query4 = em.createNamedQuery("getTransactionsByID");
    query4.setParameter(aid, account.getId());
    List<Transaction> transList = query4.getResultList();
```

12. We commit all the changes:

```
em.getTransaction().commit();
```

10.6.6 Running the JPA entity test

To run the test, follow these steps:

1. Make sure that the ITS0BANKderby connection is disconnected (with Embedded Derby, you can only have one active connection to a database). You can verify that the ITS0BANKderby connection is disconnected in the JPA perspective, Data Source Explorer. If the connection is active, right-click the connection and select **Disconnect**.

2. Right-click the **EntityTester** class and select **Run As** → **Java Application**.
3. You receive error messages in the Console that the `Customer` class cannot be found.
4. Select **Run** → **Run Configurations**. You can find the **EntityTester** configuration under Java Application.
5. Select the **Arguments** tab.
6. For Program arguments, type `333-33-3333` to work with the SSN of an existing customer.
7. For VM arguments, type (use the installation directory) the following line:

```
-javaagent:C:/IBM/RAD8/runtimes/base_v8_stub/plugins/com.ibm.ws.jpa.jar
```

Java agents: OpenJPA includes a Java agent for automatically enhancing persistent classes as they are loaded into the Java virtual machine (JVM). Java agents are classes that are invoked prior to your application's `main` method. The OpenJPA agent uses JVM hooks to intercept all class loading to enhance classes that have persistence metadata before the JVM loads them.

Figure 10-25 on page 516 shows the run configuration arguments.

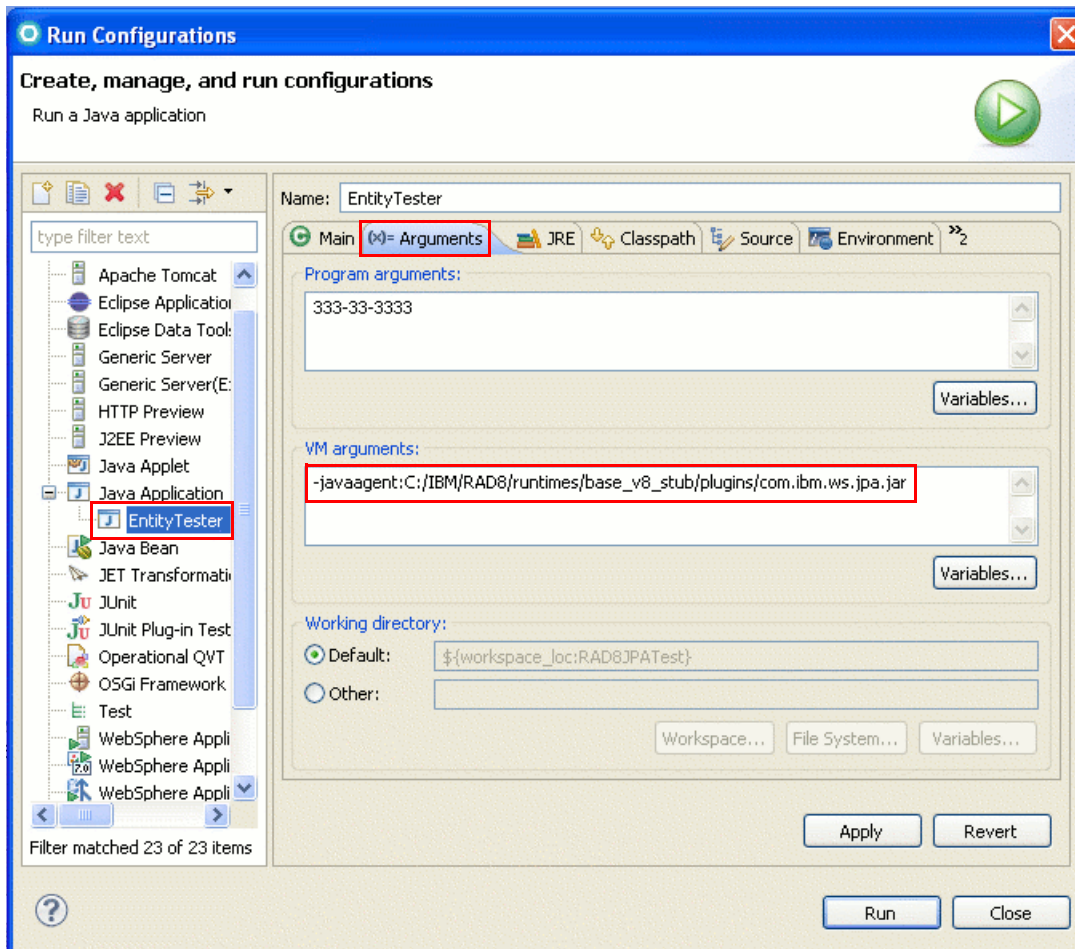


Figure 10-25 Run configuration arguments

8. Click **Apply** and then click **Run**. The Console shows the output (Example 10-33).

Example 10-33 Sample output of entity tester

Entity Testing

Creating EntityManager
 RAD8JPA EntityManager successfully created

All customers:
 111-11-1111 Mr Henry Cui
 222-22-2222 Mr Craig Fleming

333-33-3333 Mr Rafael Coutinho
444-44-4444 Mr Salvatore Sollami
555-55-5555 Mr Brian Hainey
666-66-6666 Mr Steve Baber
777-77-7777 Mr Sundaragopal Venkatraman
888-88-8888 Mrs Lara Ziosi
999-99-9999 Mrs Sylvi Lippmann
000-00-0000 Mrs Venkata Kumari
000-00-1111 Mr Martin Keen

Customers by partial name: a
444-44-4444 Mr Salvatore Sollami
555-55-5555 Mr Brian Hainey
666-66-6666 Mr Steve Baber
777-77-7777 Mr Sundaragopal Venkatraman
999-99-9999 Mrs Sylvi Lippmann
000-00-0000 Mrs Venkata Kumari

Retrieve one customer: 333-33-3333
333-33-3333 Mr Rafael Coutinho
Customer has 3 accounts
Account: 003-333001 balance 10176.52
Account: 003-333002 balance 568.79
Account: 003-333003 balance 21.56

Retrieve customer accounts sorted using named query:
Account: 003-333001 balance 10176.52
Account: 003-333002 balance 568.79
Account: 003-333003 balance 21.56

Perform transactions on one account:
Account: 003-333001 balance 10176.52
Tx created: 003-333001 Credit 100 2010-10-15 07:22:14.656 id
94edf041-ccbc-4fec-b6e2-99231c5c88bb
Tx created: 003-333001 Debit 50 2010-10-15 07:22:14.671 id
0338af0f-1c27-4da0-b441-a56b91d4bc94
Transaction failed: Not enough funds for DEBIT of 10376.52

Account: 003-333001 balance 10226.52
Account has 14 transactions
Transaction: Debit 50.00 2010-10-12 13:35:55.89 id
55231b9f-facf-42e0-b924-ff685423b104
Transaction: Credit 100.00 2010-10-12 13:35:55.89 id
c0b5ff85-6a47-49bf-8084-9592620e6a47

```
Transaction: Credit 100.00 2010-10-12 13:36:48.281 id
f8054776-973e-4766-9df0-0d61cfcc7433
Transaction: Debit 50.00 2010-10-12 13:36:48.296 id
9640dad3-55ce-4ac5-a029-61cc2b8d687c
Transaction: Debit 50.00 2010-10-12 13:38:11.171 id
dalea997-5e48-4f8a-830e-ad7cf68f0892
Transaction: Credit 100.00 2010-10-12 13:38:11.171 id
80762c74-b8a4-44ca-b0a3-cbe346e9af0f
Transaction: Credit 100.00 2010-10-13 08:13:18.359 id
782a4dec-3619-4325-9ee9-27247cb243e8
Transaction: Debit 50.00 2010-10-13 08:13:18.375 id
99385ec6-1134-45b2-92e4-91c2617edf85
Transaction: Debit 50.00 2010-10-13 08:34:17.265 id
c65a3228-879e-46fe-8599-5a16feeb12a9
Transaction: Credit 100.00 2010-10-13 08:34:17.265 id
d7ab6d70-756c-4bfc-a934-ce426f156c2b
Transaction: Debit 50.00 2010-10-13 09:49:43.718 id
c710cdc6-280a-4f9d-97ce-fe5cc9f5a7f6
Transaction: Credit 100.00 2010-10-13 09:49:43.718 id
978fac18-4aaf-4147-ba30-ed5f7c83ea07
Transaction: Credit 100 2010-10-15 07:22:14.656 id
94edf041-ccbc-4fec-b6e2-99231c5c88bb
Transaction: Debit 50 2010-10-15 07:22:14.671 id
0338af0f-1c27-4da0-b441-a56b91d4bc94
```

10.6.7 Displaying the SQL statements

You can configure the OpenJPA properties so that the SQL statements issued against the database are displayed:

1. Open the **persistence.xml** file in project RAD8JPATest.
2. Change the `openjpa.Log` property to the value `SQL=TRACE`:
`<property name="openjpa.Log" value="SQL=TRACE" />`
3. Rerun the test by selecting **Run** → **Run History** → **Entity Tester**. You can see the SQL statements:

– All customers:

```
SELECT t0.ssn, t0.FIRST_NAME, t0.LAST_NAME, t0.title FROM
ITSO.CUSTOMER t0
```

- Customers by partial last name:


```
SELECT t0.ssn, t0.FIRST_NAME, t0.LAST_NAME, t0.title FROM
ITSO.CUSTOMER t0 WHERE (t0.LAST_NAME LIKE ? ESCAPE '\')
[params=(String) %a%]
```
- Accounts of a customer:


```
SELECT t1.id, t1.balance FROM ITSO.ACCOUNT_CUSTOMER t0 INNER JOIN
ITSO.ACCOUNT t1 ON t0.ACCOUNT_ID = t1.id WHERE t0.CUSTOMER_SSN =
? [params=(String) 333-33-3333]
```
- Accounts of a customer sorted:


```
SELECT t0.id, t0.balance FROM ITSO.ACCOUNT t0 INNER JOIN
ITSO.ACCOUNT_CUSTOMER t1 ON t0.id = t1.ACCOUNT_ID INNER JOIN
ITSO.CUSTOMER t2 ON t1.CUSTOMER_SSN = t2.ssn WHERE
(t1.CUSTOMER_SSN = ?) ORDER BY t0.id ASC [params=(String)
333-33-3333]
```
- Perform a transaction:


```
INSERT INTO ITSO.TRANSACT (id, amount, TRANS_TIME, TRANS_TYPE,
ACCOUNT_ID) VALUES (?, ?, ?, ?, ?) [params=(String)
c77b2cb3-a4a6-4db3-bb27-22dec71b8bb2, (BigDecimal) 50,
(Timestamp) 2010-10-14 16:12:49.406, (String) Debit, (String)
003-999000777]
```
- Update the account balance after the transactions:


```
UPDATE ITSO.ACCOUNT SET balance = ? WHERE id = ?
[params=(BigDecimal) 10026.52, (String) 003-999000777]
```
- Transactions of an account:


```
SELECT t1.id, t1.ACCOUNT_ID, t1.amount, t1.TRANS_TIME,
t1.TRANS_TYPE FROM ITSO.ACCOUNT t0 INNER JOIN ITSO.TRANSACT t1 ON
t0.id = t1.ACCOUNT_ID WHERE (t0.id = ?) ORDER BY t1.TRANS_TIME
ASC [params=(String) 003-999000777]
```

To deactivate the trace, replace the value "SQL=TRACE" with the value "none".

10.6.8 Adding inheritance

Two types of transactions, credit and debit, are specified in the database table by the `TRANS_TYPE` column, which became the `transType` field in the `Transaction` class. In this section, we define two subclasses of `Transaction`, `Credit` and `Debit`. We use single table inheritance, by mapping all three classes to one table. The `TRANS_TYPE` column becomes the discriminator column, and the `transType` field is deleted in the `Transaction` class.

Changing the Transaction class for inheritance

Inheritance is defined through three annotations:

- @Inheritance** Defines that inheritance is present
- @DiscriminatorColumn** Defines the discriminator column
- @DiscriminatorValue** Defines the value for each class

To define inheritance in the Transaction class, follow these steps:

1. Open the **Transaction** class.
2. Add the annotations **@Inheritance** and **@DiscriminatorColumn** and make the class **abstract**, because there are no Transaction instances, only Credit and Debit, as highlighted in Example 10-34.

Example 10-34 Set transaction abstract

```
@Entity
@Table(schema="ITS0", name="TRANSACTION")
@Inheritance
@DiscriminatorColumn(name="TRANS_TYPE",
    discriminatorType=DiscriminatorType.STRING, length=32)
public abstract class Transaction implements Serializable { ... }
```

The transType field is not part of the instances, after the matching column is used as the discriminator column.

3. Remove or comment the transType field:

```
//@Column(name="TRANS_TYPE")
//private String transType;
```

4. Change the getTransType method to abstract (we still want to provide the transType value to clients) and remove the setter:

```
public abstract String getTransType();
// public void setTransType(String transType) { ..... }
```

5. Remove transType from the constructor, as shown in Example 10-35.

Example 10-35 Modify Transaction constructor

```
public Transaction(String transType, BigDecimal amount) {
    super();
    setId(java.util.UUID.randomUUID().toString());
setTransType(transType);
    setAmount(amount);
    setTransTime( new Timestamp(System.currentTimeMillis()) );
}
```

Because you have changed the `Transaction` class, there is a problem in your `TransactionManager` for the method `getNewTransaction()`, because it cannot instantiate the type `Transaction` anymore. You delete the class `TransactionManager` and create manager beans for `Credit` and `Debit`. We describe the creation of these manager beans in “Creating Credit and Debit manager beans” on page 524. First, you have to create the new classes `Credit` and `Debit`, as described in following sections.

Adding the Credit subclass

The `Credit` class is a subclass of the `Transaction` class and is mapped to the same `ITSO.TRANSACT` table. To add the `Credit` subclass, follow these steps:

1. Create a `Credit` class in the `itso.bank.entities` package as a subclass of `Transaction`, as shown in Figure 10-26 on page 522:
 - a. Set the superclass to `itso.bank.entities.Transaction`.
 - b. Select **Constructor from superclass** and **Inherited abstract methods**.
 - c. Click **Finish**.

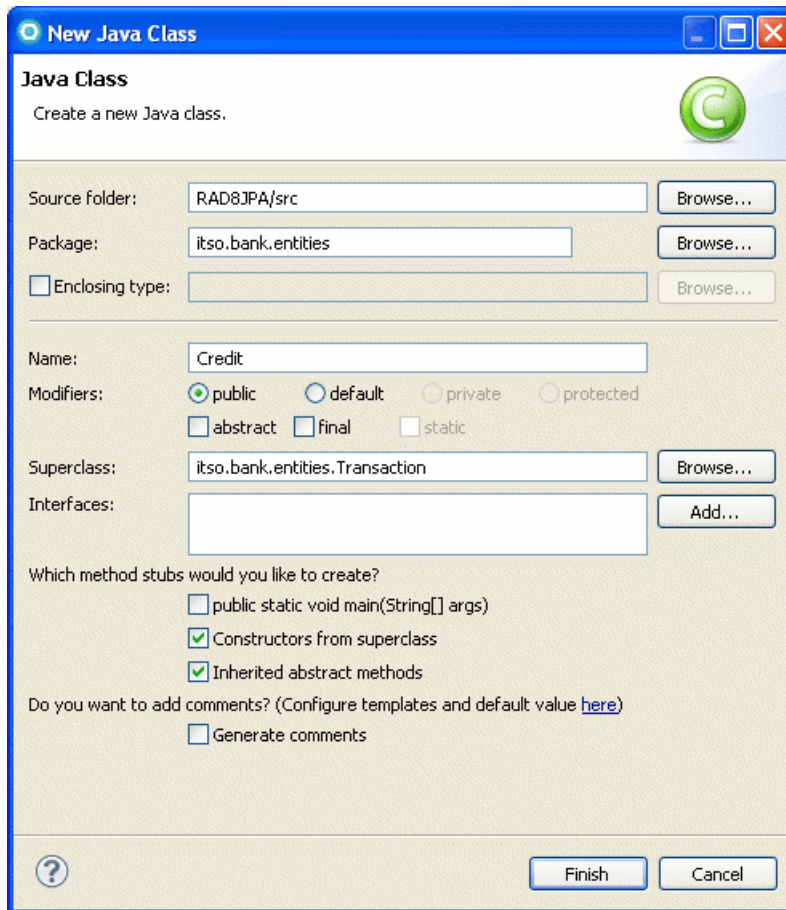


Figure 10-26 Creating the Credit class

2. Complete the code with annotations and implement the `getTransType` method, as highlighted in Example 10-36.

Example 10-36 Credit class

```

@Entity
@Inheritance
@DiscriminatorValue("Credit")
public class Credit extends Transaction {
    public Credit() {
        super(new BigDecimal(0.00));
    }
    public Credit(BigDecimal amount) { super(amount); }
    @Override

```

```
public String getTransType() {  
    return Transaction.CREDIT;  
}  
}
```

3. To resolve the warning that the class does not implement a serialVersionID, click the warning marker and select **Add default serial version ID**.

Adding the Debit subclass

Repeat the sequence, as described in the previous section, and define the Debit subclass, as shown in Example 10-37.

Example 10-37 Debit class

```
@Entity  
@Inheritance  
@DiscriminatorValue("Debit")  
public class Debit extends Transaction {  
    private static final long serialVersionUID = 1L; //generated  
  
    public Debit() {  
        super(new BigDecimal(0.00));  
    }  
    public Debit(BigDecimal amount) {  
        super(amount);  
    }  
    @Override  
    public String getTransType() {  
        return Transaction.DEBIT;  
    }  
}
```

Adding the Credit and Debit class to the persistence unit

Both new classes have to be added to the persistence unit, because as result of adding the Credit and Debit class, you have two problems in your project. The message is: "The class "Credit" is mapped, but is not included in any persistence unit." To resolve these problems, right-click the **persistence.xml** file of the JPA project and select **JPA Tools** → **Synchronize classes**. As result of this feature, the new persistent classes in your JPA project are automatically discovered and added to the persistence unit in the **persistence.xml** file.

Creating Credit and Debit manager beans

Perform the steps that are described in 10.4, “Creating a JPA Manager Bean” on page 495 and right-click the **RAD8JPA** project and select **JPA Tools** → **Add JPA Manager beans**.

This time, you select the classes `Credit` and `Debit`, and the `CreditManager` and `DebitManager` are created in the `itso.bank.entities.controller` package.

Changing the Account class to process transactions

The `processTransaction` method in the `Account` class shows an error, because we create a `Transaction` instance. Now we must change the method to create either a `Credit` or a `Debit` instance:

1. Open the **Account** class.
2. Change the `processTransaction` method, as shown in Example 10-38.

Example 10-38 Processing credit or debit transactions

```
public Transaction processTransaction(BigDecimal amount,
                                       String transactionType) throws
Exception {
    Transaction transaction = null;
    if (Transaction.CREDIT.equals(transactionType)) {
        balance = balance.add(amount);
        transaction = new Credit(amount);
    } else if (Transaction.DEBIT.equals(transactionType)) {
        if (balance.compareTo(amount) < 0)
            throw new Exception("Not enough funds for DEBIT of " +
amount);
        balance = balance.subtract(amount);
        transaction = new Debit(amount);
    } else throw new Exception("Invalid transaction type");
    transaction.setAccount(this);
    return transaction;
}
```

Adding toString methods for printing

To facilitate the output when testing, especially with the Universal Test Client, add `toString` methods to the classes:

1. Open the **Account** class and add the `toString` method:

```
public String toString() {
    return "Account: " + getId() + " balance " + getBalance();
}
```


2. Open the **Customer** class and add the toString method:

```
public String toString() {
    return "Customer: " + getSsn() + " " + getTitle() + " "
        + getFirstName() + " " + getLastName();
}
```

3. Open the **Transaction** class and add the toString method:

```
public String toString() {
    return getTransType() + ": " + getAmount() + " at "
        + getTransTime() + " (" + getAccount().getId() + ")";
}
```

Testing inheritance

The EntityTester class shows no errors, and we can run the test unchanged by selecting **Run** → **Run History** → **EntityTester**. However, if you run the test, make sure that it works as before.

Adding inheritance to the class diagram

You can add the two subclasses to the class diagram by dragging them to the diagram. Then rearrange the diagram for better visibility, as shown in Figure 10-27.

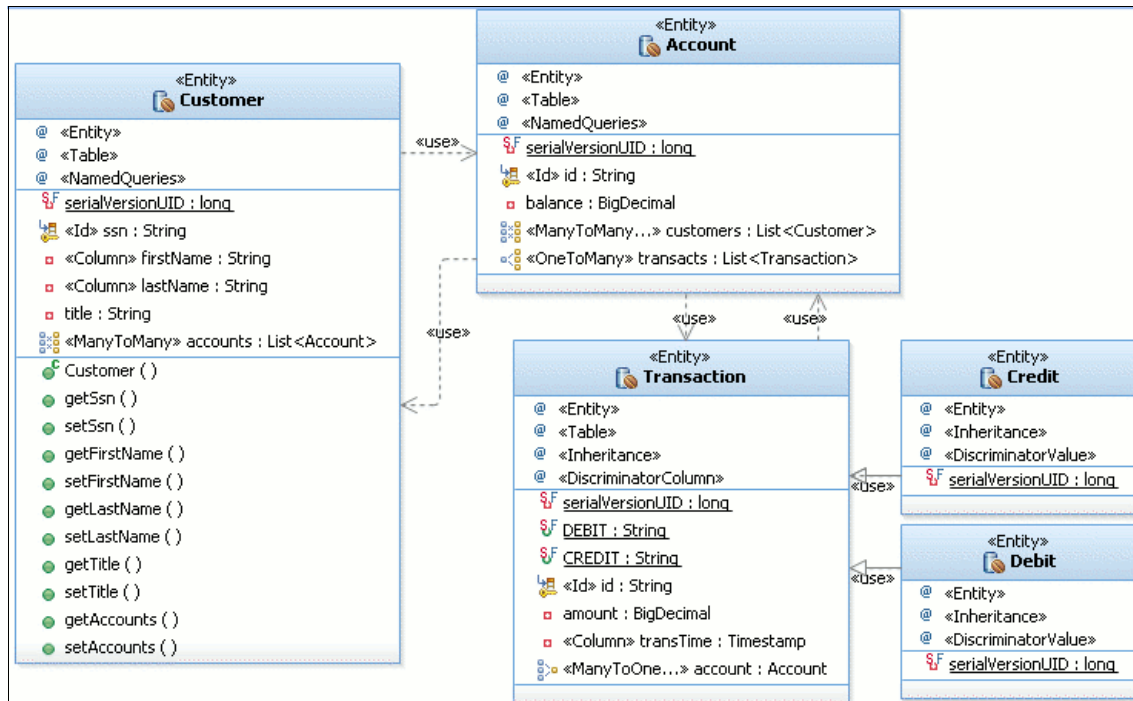


Figure 10-27 Complete class diagram of JPA entities

10.7 Preparing the entities for deployment in the server

Later in this book, we access the JPA entities from EJB 3.1 session beans. To access the JPA entities, we must configure the server with a data source for the ITSOBANK database. We must also configure the persistence.xml file to specify the JNDI name of the data source. The data source of the ITSOBANK database is defined in “Configuring the data source in WebSphere Application Server” on page 1882:

- ▶ For a user with DB2, we define data sources for both Derby and DB2. The JNDI names are *jdbc/itsobank* and *jdbc/itsobankdb2*.
- ▶ We use *jdbc/itsobank* for the JPA entities. By changing the JNDI names in the server, we can run with either database without changing the application code.

Alternative: For testing purposes, you can configure the data source in the WebSphere Enhanced EAR editor of an enterprise application. We describe this technique in 23.8.1, “Creating a data source in the Enhanced EAR editor” on page 1262.

Configuration in persistence.xml

In order for JPA to use the correct database at run time, we have to add the JNDI name of the data source and the transaction type to the `persistence.xml` file of the RAD8JPA project, as shown in Example 10-39.

Example 10-39 Define JNDI name in persistence.xml

```
<persistence .....
```

```
  <persistence-unit name="RAD8JPA" transaction-type="JTA">
```

```
    <jta-data-source>jdbc/itsobank</jta-data-source>
```

```
    <class>
```

```
      .....
```

Configuration in orm.xml

Additionally, we have to define the ITS0 schema in our `orm.xml` file. Perform the following steps:

1. Open **orm.xml** with Object Relational Mapping XML Editor.
2. Select **Entity Mappings** in the Overview pane and click **Add**.
3. In the Add Item window, select **Persistence Unit Metadata**, as shown in Figure 10-28 on page 528, and click **OK**.

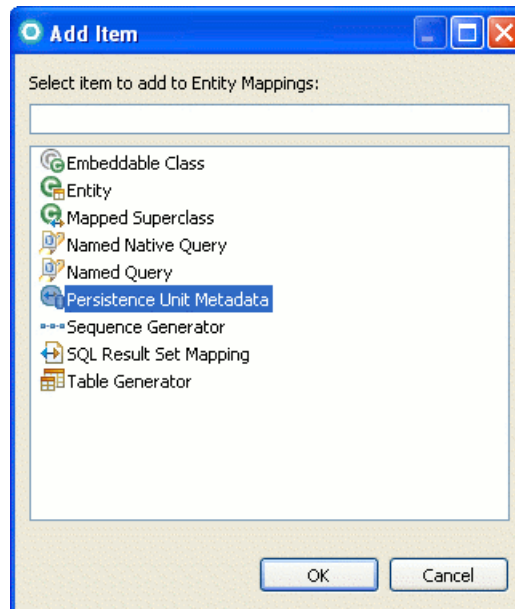


Figure 10-28 ORM XML Editor: Add Item

4. Select your added item, **Persistence Unit Metadata**, under Entity Mappings in the Overview pane, and click **Add**.
5. In the Add Item window, select **Persistence Unit Defaults** and click **OK**.
6. Select your added item, **Persistence Unit Defaults**, under Persistence Unit Metadata in the Overview pane and define the schema in the Details pane. Type ITS0 for Schema, as shown in Figure 10-29 on page 529.

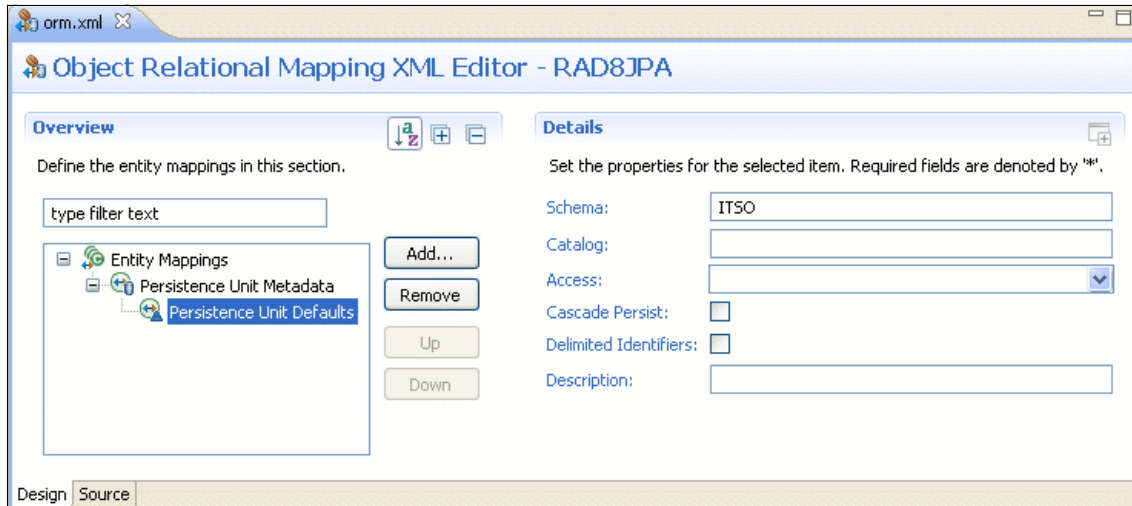


Figure 10-29 Object Relational Mapping XML Editor: Schema ITSO

7. Save and close your `orm.xml` file.

Now your JPA project is ready to use for deployment in the server.

10.8 More information

- ▶ The JPA 2.0 is documented separately from the EJB 3.1 specification and is available in *JSR 317: Java Persistence API, Version 2.0* at the following address:
 - <http://jcp.org/en/jsr/summary?id=317>
- ▶ Refer to *Getting Started with the Feature Pack for OSGi Applications and JPA 2.0*, SG24-7911, for information about the Feature Pack for JPA 2.0.
- ▶ Refer to the WebSphere Application Server V8 Beta Information Center for detailed descriptions about JPA application development:
 - http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.jpa.doc/topics/c_jpa.html
 - <http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/com.ibm.servertools.doc/topics/tjpaautv7.html>
 - <http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/com.ibm.ertools.webtoolscore.doc/topics/tjpaconfigmgrbeanother.html>



Developing applications to connect to enterprise information systems

Java EE Connector Architecture (JCA) plays a key role in the integration of applications and data using open standards. In this chapter, we introduce JCA and demonstrate by example how to access the operations and data on enterprise information systems (EIS), such as CICS, IMS, SAP, Siebel, PeopleSoft, JD Edwards, and Oracle, within the Java EE platform.

In addition, we explain how to develop Java Enterprise Edition (EE) applications using Java EE Connector tools within Rational Application Developer.

The chapter is organized into the following sections:

- ▶ Introduction to Java EE Connector Architecture
- ▶ Application development for EIS
- ▶ Sample application overview
- ▶ CICS outbound scenario
- ▶ CICS channel outbound scenario
- ▶ SAP outbound scenario
- ▶ Monitoring inbound events for resource adapters
- ▶ More information

11.1 Introduction to Java EE Connector Architecture

JCA is a standard architecture for connecting enterprise information systems (EISs), such as CICS, IMS, SAP, Siebel, PeopleSoft, JD Edwards, and Oracle. JCA standardizes the way that Java EE application components, Java EE-compliant application servers, and EIS resources interact with each other. Resource adapters and application servers implement the contract defined in the JCA specification. Resource adapters run in the context of the application server and enable Java 2 Platform, Enterprise Edition (J2EE) application components to interact with the EIS using a common client interface. JCA-compliant application servers can support any JCA-compliant resource.

Figure 11-1 shows the Java EE component connected to an EIS through the JCA resource adapter.

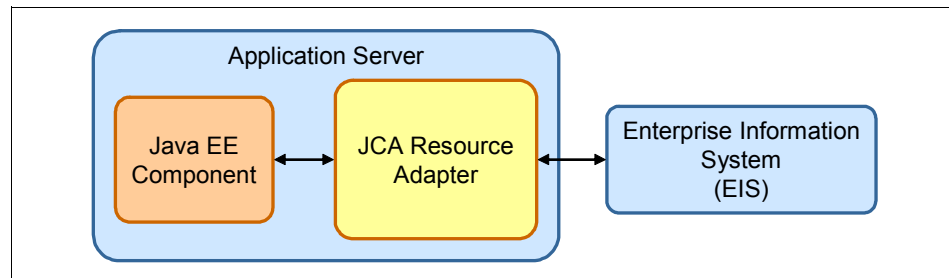


Figure 11-1 Java EE component connecting to EIS through JCA resource adapter

JCA was developed under the Java Community Process as Java Specification Request (JSR) 16 (JCA 1.0) and JSR 112 (JCA 1.5), which is the current version.

11.1.1 System contracts

In this section, we present the major components of the JCA architecture. JCA defines a standard set of system-level contracts between the Java EE application server and a resource adapter. The application server and the resource adapter connect and interact with each other by using the system contracts.

The JCA specification defines the following types of system contracts:

Connection management contract

Allows the application server to create a physical connection to the EIS. It also provides a mechanism for the application server to manage connection pooling.

Transaction management contract

Provides transactional support that allows the EIS to participate in a transaction. Transactions can be managed by the application server's transaction manager with multiple EISs and other resources as participants.

Security contract

Allows application components in an application server to access the EIS securely. The security contract is an extension of the connection management contract implemented by adding Java Authentication and Authorization Service (JAAS) into connection management interfaces.

Life-cycle management contract

Allows the application server to manage the life cycle of the resource adapter. It provides a mechanism for the application server to start and shut down an instance of the resource adapter.

Work management contract

Allows the resource adapter to submit work to the application server for execution. The application server dispatches a thread to handle the work. This contract is optional.

Transaction inflow contract

Allows the EIS to propagate a transaction through the resource adapter to the application server.

Message inflow contract

Allows the resource adapter to pass synchronous or asynchronous inbound messages to message endpoints on the application server.

Figure 11-2 on page 534 shows the integration between the EIS, application server, and application component. These components are bound together by using JCA contracts.

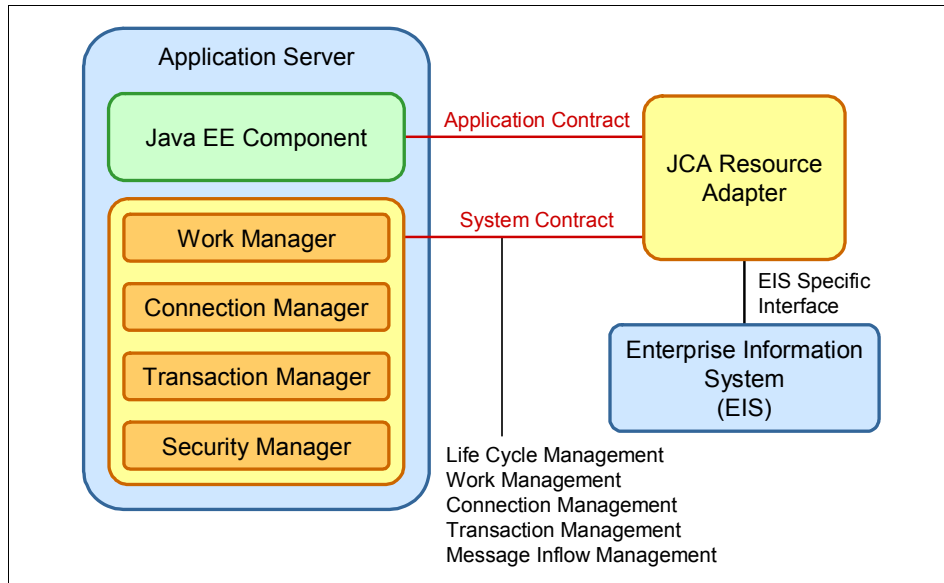


Figure 11-2 System contract, application server, and resource adapter integration

To learn more about JCA Specification, visit the JCA specification website:

<http://java.sun.com/j2ee/connector/download.html>

11.1.2 Resource adapter

To achieve a standard system-level pluggability between application servers and EISs, the JCA architecture defines a standard set of system-level contracts between an application server and EIS as discussed in 11.1.1, “System contracts” on page 532. The JCA resource adapter implements the EIS side of these system-level contracts.

A JCA resource adapter is a system-level software driver used by an application server or an application client to connect to an EIS. By plugging into an application server, the resource adapter collaborates with the server to provide the underlying mechanisms, transactions, security, and connection pooling mechanisms. A JCA resource adapter is used within the address space of the application server.

Rational Application Developer includes the following JCA resource adapters along with JEE 6 adapters:

- ▶ CICS ECI adapter 8.0.0.0
- ▶ CICS ECI XA adapter 8.0.0.0

- ▶ IMS Transaction Manager (TM) resource adapter 10.4
- ▶ IMS TM resource adapter 11.2

JCA resource adapters support two-way communication between the Java EE components and an EIS:

- ▶ *Outbound communication* is initiated by a J2EE component, which acts as a client to access an EIS.
- ▶ *Inbound communication* is initiated by the EIS to notify a Java EE component, which subscribed for events from that EIS. Inbound communications are performed asynchronously using the messaging infrastructure that is provided by the hosting application server as message providers.

Java EE components that use the resource adapter can co-reside with the adapter on the same application server or operate remotely.

11.1.3 Common Client Interface

The Common Client Interface (CCI) is a standard API that allows application components and Enterprise Application Integration (EAI) frameworks to interact with the resource adapter. It provides a standard way for application components to invoke functions on an EIS and get the returned results. The CCI is intended for use by EAI and enterprise tools vendors. Therefore, WebSphere adapters use CCI for outbound communication with an EIS.

11.1.4 WebSphere adapters

The WebSphere adapter portfolio is a new generation of adapters based on the Java EE Platform, Enterprise Edition standard. A WebSphere adapter implements the JCA specification 1.5. Also known as resource adapters or JCA adapters, WebSphere adapters enable managed, bidirectional connectivity and data exchange between a number of EIS resources, including PeopleSoft, SAP, Siebel, JD Edwards, and Oracle.

Rational Application Developer includes the following versions of WebSphere adapters:

- ▶ IBM WebSphere adapter for JD Edwards EnterpriseOne 6.2.0.x_IF01
- ▶ IBM WebSphere adapter for JD Edwards EnterpriseOne 7.0.0.0_IF02
- ▶ IBM WebSphere adapter for Oracle 6.2.0.x
- ▶ IBM WebSphere adapter for Oracle e-business suite 7.0.0.0_IF02
- ▶ IBM WebSphere adapter for PeopleSoft Enterprise 6.2.0.x
- ▶ IBM WebSphere adapter for PeopleSoft Enterprise 7.0.0.0_IF03

- ▶ IBM WebSphere adapter for Sap Software 6.2.0.x
- ▶ IBM WebSphere adapter for Sap Software 7.0.0.0_IF04
- ▶ IBM WebSphere adapter for Sap Software with transaction support 6.2.0.x
- ▶ IBM WebSphere adapter for Sap Software with transaction support 7.0.0.0_IF04
- ▶ IBM WebSphere adapter for Siebel Business Applications 6.2.0.x
- ▶ IBM WebSphere adapter for Siebel Business Applications 7.0.0.0_IF03

When developing a custom JCA-compliant resource adapter, you can choose to develop either the WebSphere type of resource adapter or the base JCA type of resource adapter. WebSphere adapters are fully compliant with the JCA 1.5 specification and contain IBM extensions.

If you choose to develop an IBM WebSphere type of resource adapter, you can use the services provided by the adapter foundation classes. You can extend the generically implemented system contract classes to fit the needs of your custom adapter. Your custom adapter can also use the built-in utility APIs to handle common adapter tasks. Using adapter foundation classes significantly reduces your development time and effort to create a custom adapter.

More information: For more information about custom adapter development, see *WebSphere Adapter Development*, SG24-6387.

11.2 Application development for EIS

Rational Application Developer simplifies application development for EIS by providing wizard-based tools and a list of ready to use adapters. This section introduces these tools and their capabilities.

With the Java EE Connector tools, you can create Java EE applications running on WebSphere Application Server to access operations and data on EIS. Java EE Connector (J2C) tools offer the following qualities of service (QoS) that can be provided by an application server:

- ▶ Security credential management
- ▶ Connection pooling
- ▶ Transaction management

These qualities of service are provided by means of system-level contracts between a resource adapter provided by the connector (for example, CICS Transaction Gateway or IMS Connect) and the application server.

11.2.1 Importers

IMS or CICS external call interface (ECI) transactions are often written in COBOL, C, or PL/I. For a Java application to access these transactions through J2C resource adapters, the data must be imported and mapped to Java data structures. The importers are tools that deliver this data mapping. Three importers are available for you to use in your application: C Importer, COBOL Importer, and PL/I Importer. After you import a COBOL, C, or PL/I file into a project, you can work with this data as you might with any data construct.

11.2.2 J2C wizards

Rational Application Developer provides J2C wizards with which you can create J2C applications, either as stand-alone programs or as added functions to existing applications. These wizards offer the following benefits:

- ▶ Dynamically import your selected resource adapter
- ▶ Help you to set the connection properties to connect to the EIS servers
- ▶ Guide you through the file importing and data mapping steps
- ▶ Facilitate the creation of Java classes and methods to access the transformed source data

A typical J2C application consists of a J2C JavaBean with one or more methods that call EIS functions. For CICS and IMS, the input and outputs to these functions are data binding classes that are created by the CICS/IMS Java Data Binding Wizard. When you have created a J2C JavaBean, you then can create web pages, Enterprise JavaBeans (EJB), or a web service for the JavaBean.

To use the J2C wizard within Rational Application Developer, follow these steps:

1. Switch to the **Java EE** perspective.
2. Select **File** → **New** → **Other** → **J2C** and select the J2C wizard that you want to start (Figure 11-3).

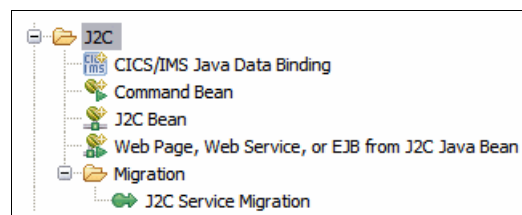


Figure 11-3 J2C wizards

The following wizards are available:

- ▶ CICS or IMS Java Data Binding: You can create the data binding classes on their own. These classes are used in J2C methods that invoke CICS or IMS functions.
- ▶ Command Bean: You can use this wizard (optionally) to expose selected methods as a command bean.
- ▶ J2C Bean: You can use this wizard to create a JavaBean that communicates with an EIS through JCA.
- ▶ Web page, web service, or EJB from J2C JavaBean: You can use this wizard to create a Java EE resource that wraps the functionality provided by a J2C JavaBean. For example, you can create JavaServer Pages (JSP) to deploy the J2C bean on WebSphere Application Server. The Java EE resource types available with this wizard are Simple JSP, Faces Web Page, EJB, and web service.
- ▶ J2C Service Migration: You can use this wizard to migrate JCA applications that were created in WebSphere Studio Application Developer Integration Edition applications into Rational Application Developer projects.

More information: For more information about the J2C wizard, see the Rational Application Developer Information Center:

<http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp>

11.3 Sample application overview

In this section, we show three sample applications that connect to separate types of EISs (CICS and SAP) and illustrate outbound communication. We provide the following examples:

- ▶ CICS outbound scenario
- ▶ CICS channel outbound scenario
- ▶ SAP outbound scenario

11.4 CICS outbound scenario

In this example, we expose a COBOL program as an EJB 3.1 session bean that is invoked by a JavaServer Faces (JSF) JSP.

The product documentation contains related samples and tutorials that are used as the starting point for this scenario:

- ▶ Tutorials → Java → Create a J2C application for a CICS transaction with the same input and output
- ▶ Samples → Technology Samples → Java → J2C Samples → CICS adapter samples → Same input and outputs

11.4.1 Prerequisites

You must configure the CICS server and the CICS Transaction Gateway for this example to work. This configuration is beyond the scope of this book. Obtain the parameters required to connect to the CICS Transaction Gateway from your CICS administrator. These parameters typically include the following types:

- ▶ URL
- ▶ Server name
- ▶ Port
- ▶ User name
- ▶ Password

The sample COBOL program, `taderc9.cbl`, is in the `<RAD_Install-SDPShared>/plugins/com.ibm.j2c.cheatsheet.content_7.0.110.v20100921-2345/Samples/CICS/taderc99` directory.

This program must be installed on the CICS server. You must be able to query it from a CICS Terminal (Figure 11-4) using the command:

```
CEMT INQ PROG(taderc99)
```

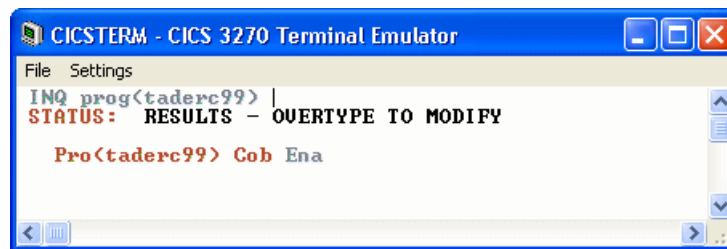


Figure 11-4 CICS Terminal showing the installed COBOL program `taderc99.cbl`

11.4.2 Creating the Java data binding class

With the CICS/IMS Java Data Binding wizard, you can create a class or set of classes that map to COBOL, to C, or to PL/I data structures:

1. Select from the menu **File** → **New** → **Other**.

2. In the Select a wizard window, click **J2C** → **CICS/IMS Java Data Binding** and click **Next**.
3. In the Data Import window, for Choose mapping, select **COBOL to Java**. Click **Browse** and locate the taderc99.cbl file in the <RAD_Install_SDPShared>\plugins\com.ibm.j2c.cheatsheet.content_7.0.10.v20100921-2345/Samples/CICS/taderc99 directory.

Example 11-1 shows the COBOL DFHCOMMAREA structure in taderc99.

Example 11-1 DFHCOMMAREA of COBOL program

```
01 DFHCOMMAREA.  
    02 CustomerNumber    PIC X(5).  
    02 FirstName         PIC A(15).  
    02 LastName          PIC A(25).  
    02 Street            PIC X(20).  
    02 City              PIC A(20).  
    02 Country           PIC A(10).  
    02 Phone             PIC X(15).  
    02 PostalCode        PIC X(7).
```

Click **Next**.

4. In the Importer window, set values for the platform, code page, and other properties. For Data Structures, the default value **DFHCOMMAREA** is preselected. Click **Next**.
5. In the Saving Properties window, complete the following steps:
 - a. For Project Name, select **Taderc99**. (Click **New** and create a new Java project.)
 - b. For Package Name, type `sample.cics.data`.
 - c. For ClassName, change the default DFHCOMMAREA to `CustomerInfo`.
 - d. Click **Finish**.

Review the generated Java file `CustomerInfo` class. Notice that it implements `javax.resource.cci.Record`. It also exposes getters and setters for all the fields of DFHCOMMAREA (for example, `getFirstName` and `setFirstName`).

11.4.3 Creating the J2C bean

Follow these steps to create the J2C bean:

1. Select from the menu **File** → **New** → **Other**.
2. In the Select a wizard window, click **J2C** → **J2C Bean** and click **Next**.

3. Select the resource adapter **CICS** → **ECIResourceAdapter (IBM: 8.0.0.0)** and click **Next** (Figure 11-5).

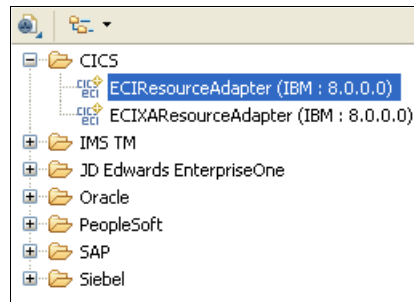


Figure 11-5 Selection of the resource adapter

This action imports the

<RAD_Install>/ResourceAdapters/cics15/cicseci8000.rar file.

4. In the Connector Import window, for Target Server, select **WebSphere Application Server 8.0** and click **Next**.
5. In Connection Properties (Figure 11-6 on page 542), complete these actions:
 - a. Clear **Managed Connection**.
 - b. Select **Non-Managed Connection**.
 - c. Enter the connection details, which are typically the Connection URL, Server name, Port number, User name, and Password, as provided by your CICS administrator.
 - d. Click **Next**.

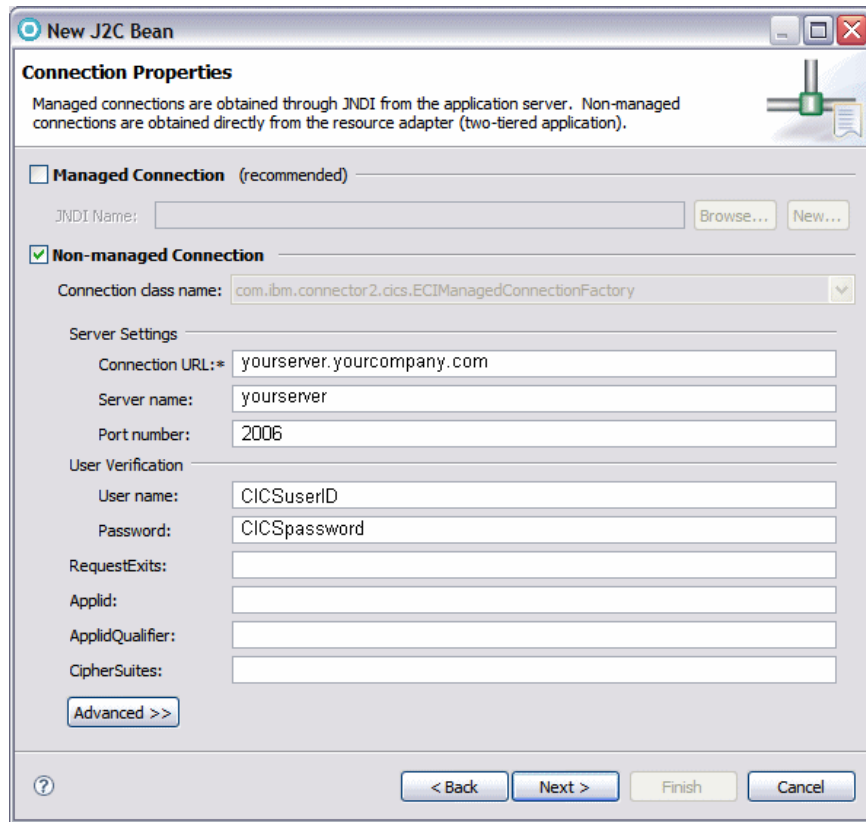


Figure 11-6 Connection Properties

6. In the J2C Java Bean Output Properties window, complete the following steps:
 - a. For Project Name, enter Taderc99.
 - b. For Package name, enter sample.cics.
 - c. For Interface name, type Customer.
 - d. For Implementation Name, select **CustomerImpl** (automatically filled).
 - e. Click **Next**.
7. In the Java Methods window, click **Add**.
8. In the Java Method window (Figure 11-7 on page 543), complete the following steps:
 - a. For Name, type getCustomer.
 - b. For Input type, click **Browse** and type cust to locate the CustomerInfo class in the sample.cics.data package. The value becomes /Taderc99/src/sample/cics/data/CustomerInfo.java.

- c. Select **Use the input type for output**.
- d. Click **Finish**.

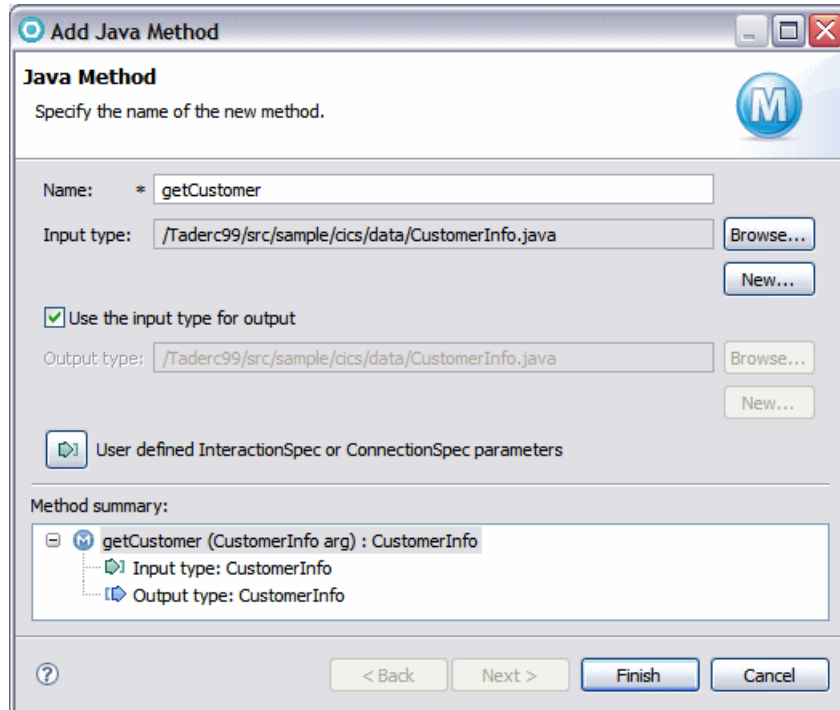


Figure 11-7 Adding a new Java method to a J2C JavaBean

- Back in the Java Methods window (Figure 11-8), in the InteractionSpec properties section, for Function name, enter `taderc99` (this value must match the name of the CICS program). You can see that the method `getCustomer` is listed. Click **Next**.

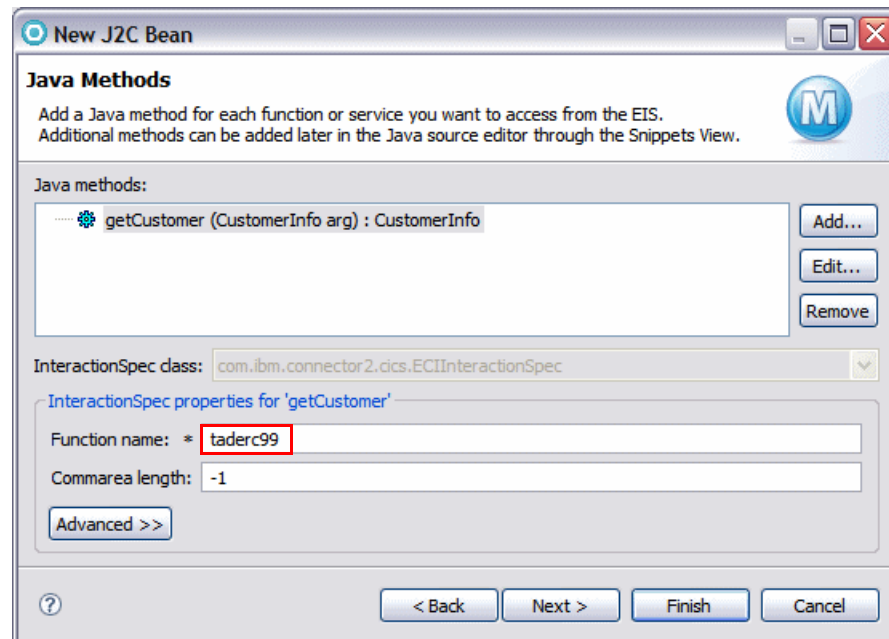


Figure 11-8 InteractionSpec properties

- On the Deployment Information window, click **Finish**.

A Customer interface and a CustomerImpl class are generated in the Taderc99 project. In addition, the `cicseci8000` project is created. This project contains a plug-in (`j2c_plugin.xml`) and a CICS resource adapter XML file (`ra.xml`).

11.4.4 Deploying the J2C bean as an EJB 3.0 session bean

From the J2C bean window, generate a session EJB:

- Select **File** → **New** → **Other**.
- In the Select a wizard window, select **J2C** → **Web page, Web Service or EJB from J2C Java bean** and click **Next**.
- By default, the J2C bean implementation gets populated in the J2C Java bean selection window (Figure 11-9 on page 545). If it does not, click **Browse** and

type cust to locate the **CustomerImpl** class
(\Taderc99\src\sample\cics\CustomerImpl.java). Click **Next**.

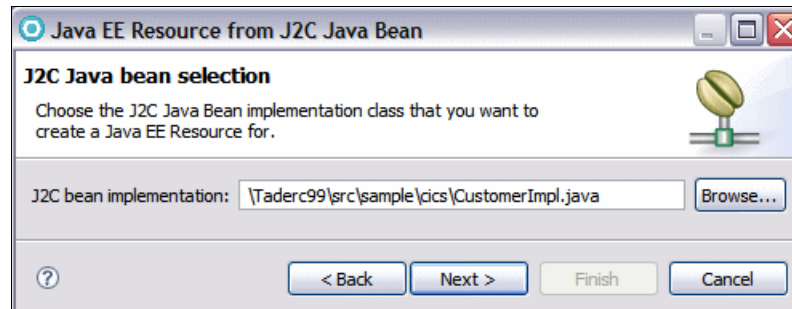


Figure 11-9 Generate an EJB from a J2C bean

4. In the Deployment Information window, select **EJB**. The target project containing the J2C bean will be transformed into an EJB 3.0 project, and the J2C bean class will be annotated as a session bean. Click **Next**.
5. In the Enterprise bean creation window (Figure 11-10), for the EAR project name, click **New** and create a new EAR project by typing Taderc99Ear. For Java Naming and Directory Interface (JNDI) name, type j2c/taderc99. Accept the other values. Click **Finish**.

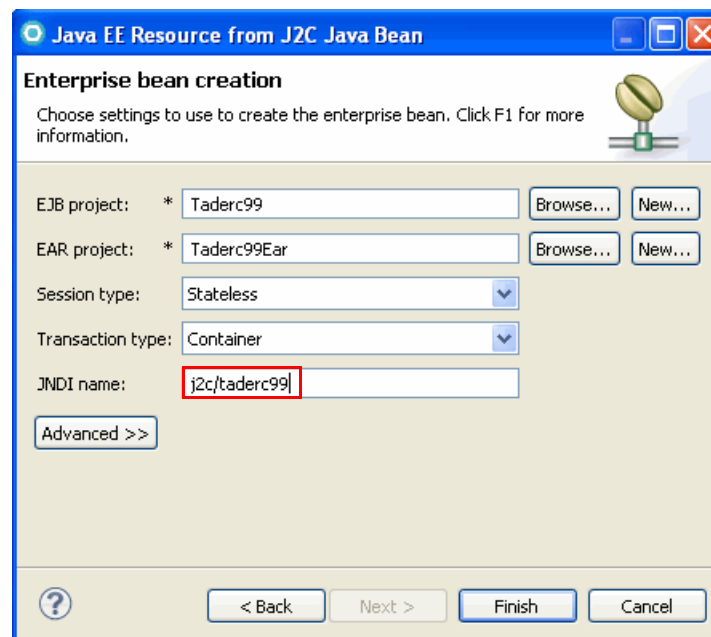


Figure 11-10 Generate an EJB from a J2C bean

6. Open the **CustomerImpl** class and verify that the `@Stateless` annotation was added:

```
@Stateless(mappedName="j2c/taderc99")
public class CustomerImpl implements sample.cics.Customer {
```

11.4.5 Generating a JSF client

Now generate a JSF client to test the EJB and the access to the CICS system.

Creating a web project and enterprise application

To create a web project and enterprise application, follow these steps:

1. Create a new dynamic web project:
 - a. For Project name, type **Taderc99Web**.
 - b. For Target Runtime, select **WebSphere Application Server v8.0 Beta**.
 - c. For Dynamic Web Module version, select **3.0**.
 - d. For Configuration, modify **JavaServer Faces v2.0 Project**.
 - e. For EAR membership, select **Taderc99Ear**.
 - f. Click **Finish**.
2. Add the **Taderc99** and **cicseci8000** projects to the Java EE Module dependencies list of the EAR project and of the web project:
 - a. Right-click **Taderc99Ear** and select **Properties**. In the Project references window, select **Taderc99** and **cicseci8000** and click **OK**.
 - b. Right-click **Taderc99Web** and select **Properties**. In the Project references window, select **Taderc99** and **cicseci8000** and click **OK**.

Creating a web page

Add a new web page called **CustomerPage**:

1. Right-click **WebContent** (in **Taderc99Web**) and select **New** → **Web Page**.
2. For the Name, type **CustomerPage**. Select **Basic Templates** → **Facelet** and click **Finish**.

Creating a data component

To create a data component, complete these steps:

1. Look at the Page Data view. If you do not see a **Services** folder, click the **Create a new data component** icon (). In the New data component window, select **Services** and click **OK**.

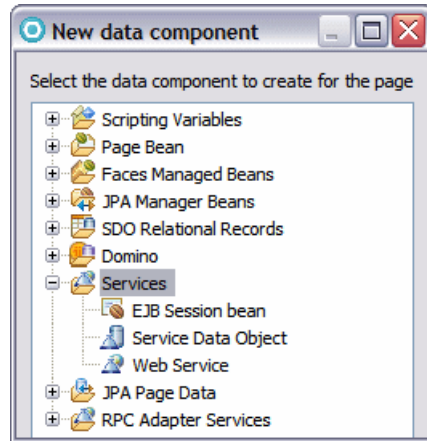


Figure 11-11 New data component window

2. In the Page Data view, right-click **Services** and select **New** → **EJB Session bean**.
3. In the Add Session Bean window:
 - a. Click **Add** to open the Add EJB Reference window.
 - b. In the EJB Reference window (Figure 11-12 on page 548), select **Taderc99EAR** → **Taderc99** → **CustomerImpl**.
The Name becomes `ejb/CustomerImpl`.
 - c. For RefType, select **Local**.
 - d. Click **Finish**.

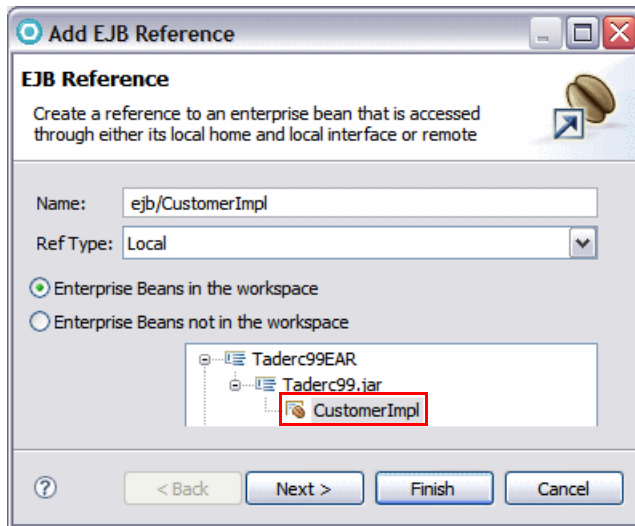


Figure 11-12 Creation of an EJB reference in the web project

4. In the Add Session Bean window, in which the `ejb/CustomerImpl` service has been added and the `getCustomer` method is selected (Figure 11-13), click **Finish**.

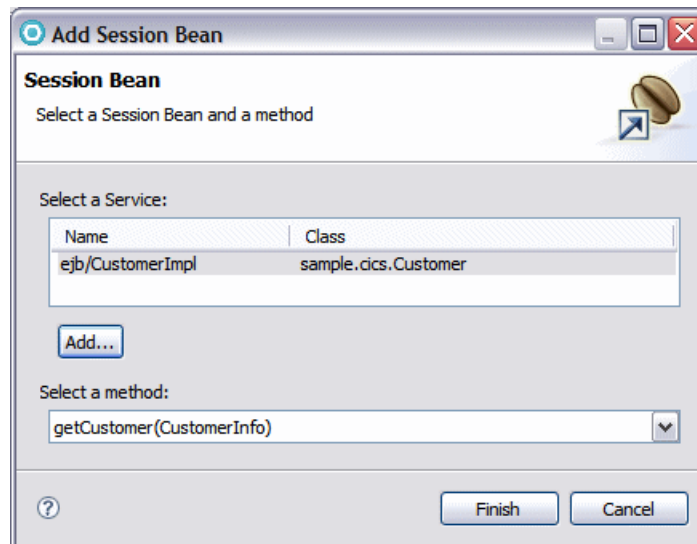


Figure 11-13 Add Session Bean window

In the Page Data view (Figure 11-14), you can see the following items:

- ▶ A `customer_GetCustomer` method with an input `Param Bean` and an output `Result Bean`, both of type `sample.cics.data.CustomerInfo`
- ▶ An action `customer_GetCustomer.doAction`

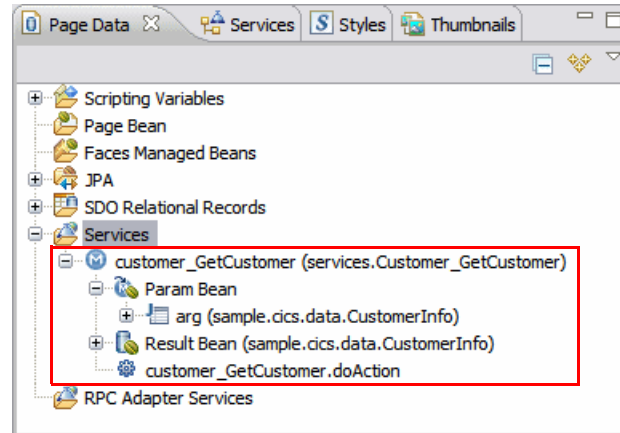


Figure 11-14 Page Data view with session bean

Creating the web page content

To create the web page content, follow these steps:

1. Expand **customer_GetCustomer** → **Param Bean** → **arg**.
2. Select **customerNumber** and drag it to the editor of the CustomerPage web page.
3. When prompted for Configuring Data Controls, complete the following steps:
 - a. Select **Inputting data**.
 - b. Change the label to Customer number.
 - c. Click **Options** and verify that a Submit button is created.
 - d. Click **Finish**, and an Input Control and Button are added to the page.
4. Drag the action **customer_GetCustomer.doAction** from the Page Data view to the **Submit** button.
5. Drag the **Result Bean** from the Page Data view to the editor. Drop it under the **Submit** button.
6. When prompted for Configuring Data Controls, complete the following steps:
 - a. Select the fields that were present in DFHCOMMAREA (starting with customerNumber).
 - b. Optional: Tailor the labels and the sequence.

- c. Click **Finish**.
7. Save and close the web page.

11.4.6 Running the JSF client

To run the JSF client, follow these steps:

1. Start WebSphere Application Server V8.0 Beta if it is not already started.
2. Add the **Taderc99EAR** enterprise application to the server.
3. Right-click **CustomerPage.xhtml** and select **Run As** → **Run on Server**.
4. When prompted, select **WebSphere Application Server v8.0 Beta**.
5. Enter the customer number 44444 and click **Submit**.

The EIS system is invoked, and the customer information is returned (Figure 11-15).

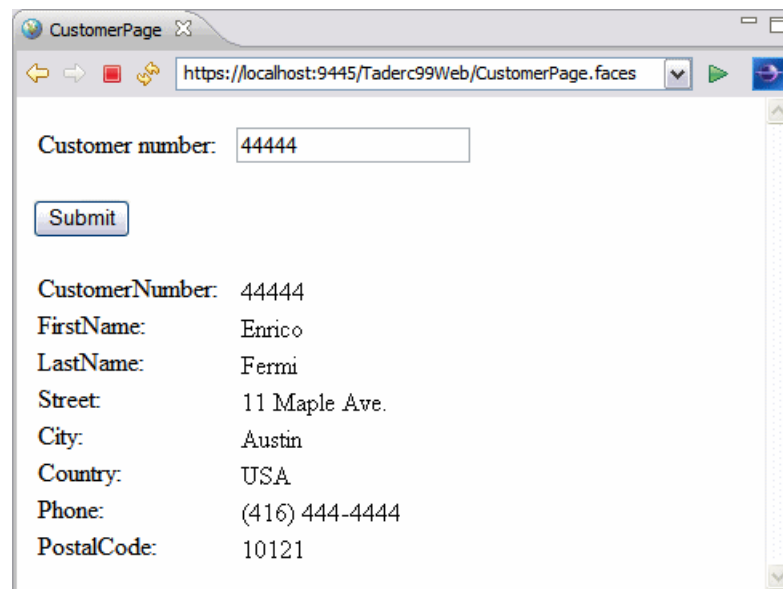


Figure 11-15 JSF output from calling a CICS system

11.5 CICS channel outbound scenario

In this example, we show how to create and execute the sample in the product help under **Samples** → **Java** → **J2C Samples** → **CICS adapter samples** → **Same input and outputs**.

We recreate this sample so that a JAX-WS web service can invoke it.

11.5.1 Creating the Java data binding for the channel and containers

Follow these steps to create the sample:

1. Select **New** → **Other**.
2. In the Select a wizard window, select **J2C** → **CICS/IMS Java Data Binding**.
3. In the Data Import window (Figure 11-16), for Choose Mapping, select **COBOL CICS Channel to Java**. You see a message: “‘Containers’ cannot be empty.” Next to the Containers area, click **New**.

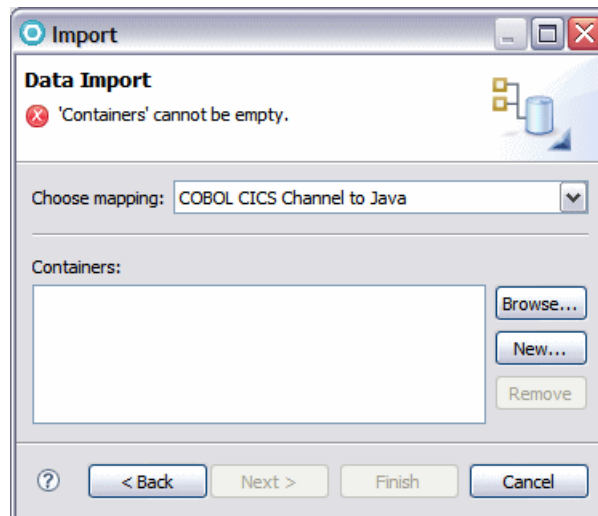


Figure 11-16 Selecting the mapping COBOL CICS Channel to Java

4. In the new Data Import: Specify data import configuration properties window, for Import File, browse to the sample COBOL file `<SDPShared_dir>\plugins\com.ibm.j2c.cheatsheet.content_7.0.110.v20100921-2345\Samples\CICS32K\ec03.ccp`. Then click **Next**.
5. In the Importer: Select a communication data structure window (Figure 11-17 on page 552), select **DATECONTAINER**, **TIMECONTAINER**, **INPUTCONTAINER**, **OUTPUTCONTAINER**, and **LENGTHCONTAINER**. Then click **Finish**.

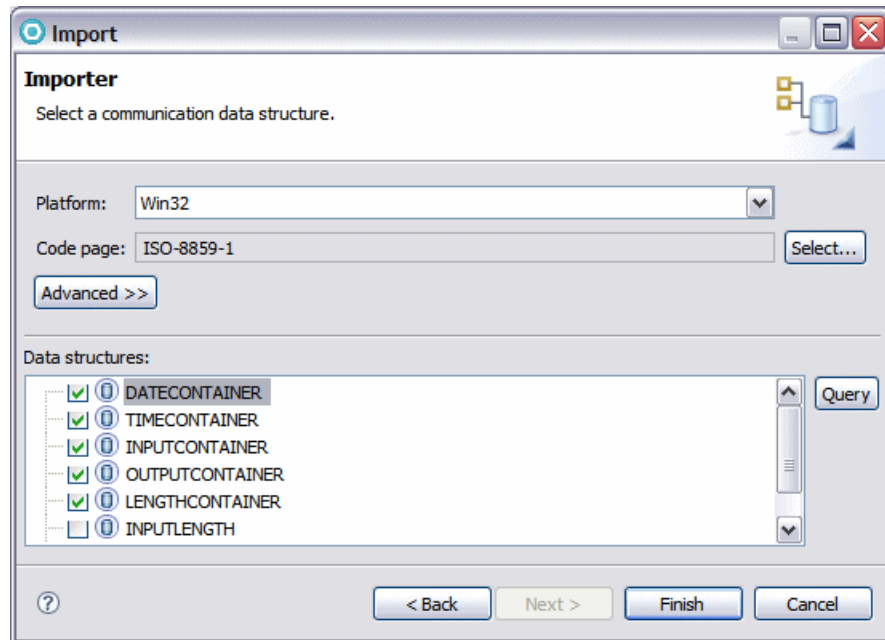


Figure 11-17 Discovery of the containers defined in the COBOL file

Other data structures are listed in the window. We made the selection to create Java beans for the five containers that are defined in the COBOL file (Example 11-2).

Example 11-2 Definition of containers in the COBOL file ec03.cpp

```
* Container names
   01 DATECONTAINER      PIC X(16) VALUE 'CurrentDate'.
   01 TIMECONTAINER      PIC X(16) VALUE 'CurrentTime'.
   01 INPUTCONTAINER     PIC X(16) VALUE 'InputData'.
   01 OUTPUTCONTAINER    PIC X(16) VALUE 'OutputMessage'.
   01 LENGTHCONTAINER    PIC X(16) VALUE 'InputDataLength'.
```

6. In the Data Import window, which opens again, showing the five containers listed, click **Next**.
7. In the Saving Properties window (Figure 11-18 on page 553), complete the following steps:
 - a. For Project Name, click **New**.
 - b. In the New Source Project window:
 - i. Select **Java project** and click **Next**.
 - ii. For Name, type **CICSChannel** and click **Finish**.

- c. Back in the Saving Properties window, for Package Name, type `sample.cics.data`.
- d. For Class Name, type `EC03ChannelRecord`.
- e. For Channel Name, type `InputRecord`.

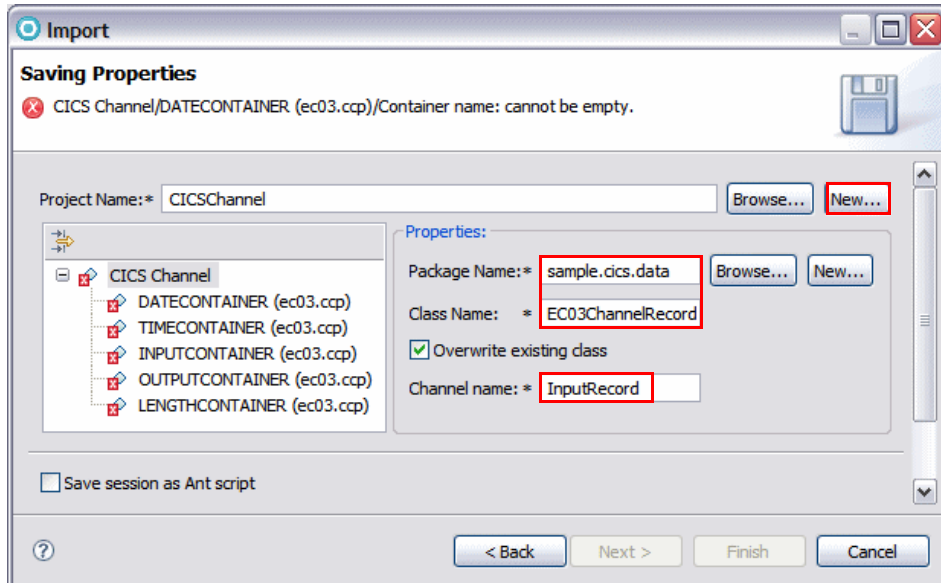


Figure 11-18 Defining the CICS Channel name and related class name

We can use an arbitrary channel name, because the COBOL file expects to receive the channel name as input (Example 11-3).

Example 11-3 Channel name is expected in input in ec03.ccp

```

* Get name of channel
  EXEC CICS ASSIGN CHANNEL(CHANNELNAME)
  END-EXEC.

* If no channel passed in, terminate with abend code NOCH
  IF CHANNELNAME = SPACES THEN
    EXEC CICS ABEND ABCODE('NOCH') NODUMP
    END-EXEC

  END-IF.

```

- f. Select the **DATECONTAINER** (Figure 11-19 on page 554) and enter the following values:
 - i. For Package Name, accept **sample.cics.data**.
 - ii. For Class Name, type `DateContainer`.

- iii. For Container name, type CURRENTDATE.
- iv. For Container type, select **CHAR**.

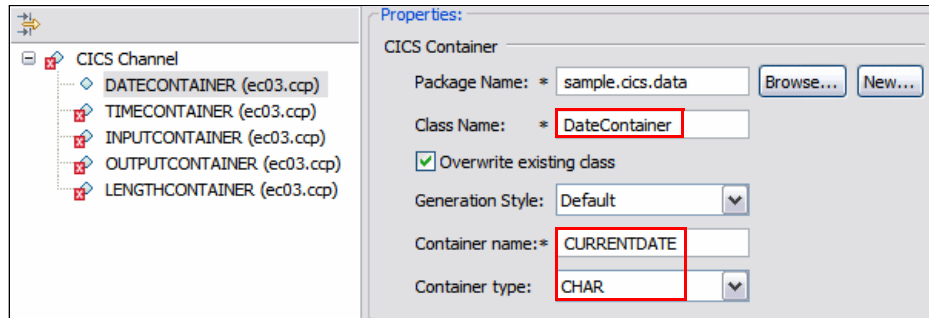


Figure 11-19 Defining the DATECONTAINER

- g. Repeat this step for all of the other containers:
 - i. Select **TIMECONTAINER**. For Class Name, type TimeContainer, for Container name, type CURRENTTIME, and for Container type, select **CHAR**.
 - ii. Select **INPUTCONTAINER**. For Class Name, type InputContainer, for Container name, type INPUTDATA, and for Container type, select **CHAR**.
 - iii. Select **OUTPUTCONTAINER**. For Class Name, type OutputContainer, for Container name, type OUTPUTMESSAGE, and for Container type, select **CHAR**.
 - iv. Select **LENGTHCONTAINER**. For Class Name, type LengthContainer, for Container name, type INPUTDATALENGTH, and for Container type, select **CHAR**.

You can choose the class names arbitrarily, but the names of the containers must match those containers that are defined in the COBOL file (Example 11-2 on page 552).

The window no longer shows any error messages.

- h. Click **Finish**.

At this point, the CICSChannel connector project is generated with a EC03ChannelRecord and five container classes in the sample.cics.data package.

- 8. Close the editor of the EC03ChannelRecord class.

11.5.2 Creating the J2C bean that accesses the channel

To create the J2C bean, follow these steps:

1. Select **File** → **New** → **Other**.
2. In the Select a wizard window, expand **J2C** → **J2C Bean** and click **Next**.
3. Select the resource adapter **CICS** → **ECIResourceAdapter (IBM: 8.0.0.0)** → **cicseci8000** and click **Next**.
4. In the Connection Properties window (same as Figure 11-6 on page 542), perform these tasks:
 - a. Clear **Managed Connection**.
 - b. Select **Non-Managed Connection**.
 - c. Enter the connection details provided by your CICS administrator, typically, Connection URL, Server name, Port number, User name, and Password.
 - d. Click **Next**.
5. In the J2C Java Bean Output Properties window (Figure 11-20), complete the following actions:
 - a. For Project name, accept CICSChannel.
 - b. For Package name, type sample.cics.
 - c. For Interface name, type Ec03.

The implementation name is now set to Ec03Impl. Click **Next**.

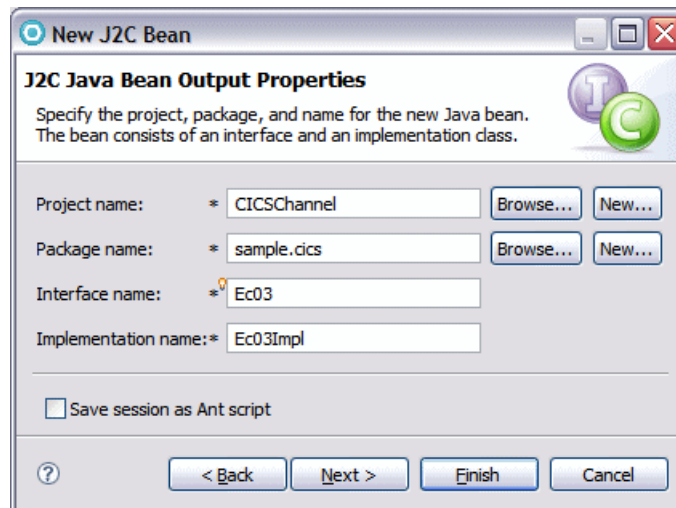


Figure 11-20 J2C Java Bean Output Properties for CICSChannel

6. In the Java Methods window, complete the following actions:
 - a. Click **Add**.
 - b. In the Java Method window (Figure 11-21), complete these steps:
 - i. For Name, type `invoke`.
 - ii. For Input type, click **Browse** and select **EC03ChannelRecord**.
 - iii. Click **Finish**.

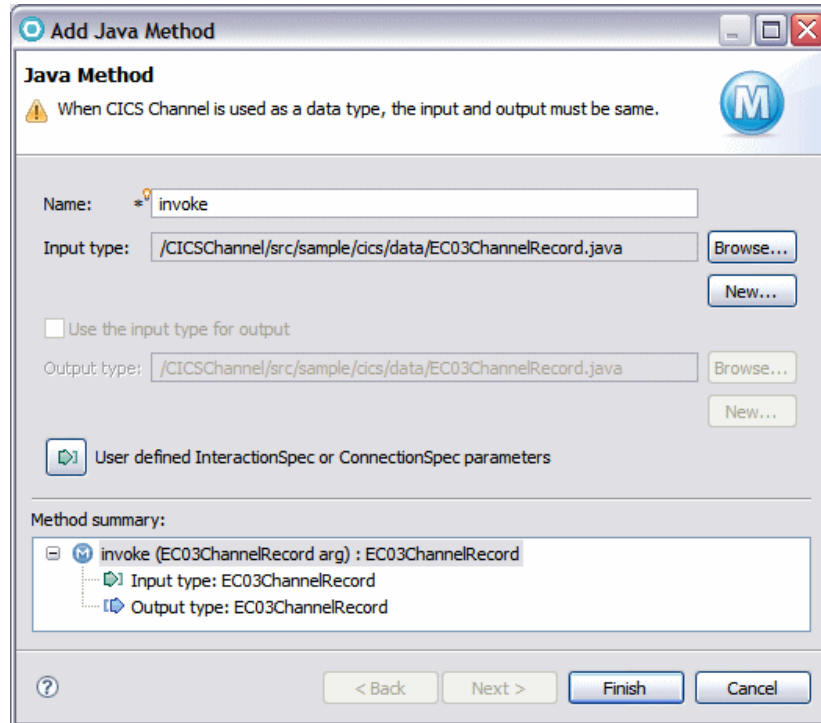


Figure 11-21 Adding a Java method with a channel record as input and output

The Java Methods window (Figure 11-22 on page 557) now lists the `invoke` method.

- c. In the InteractionSpec properties, for Function name, type `EC03` (it must match the name of the CICS program).
- d. Click **Next**.

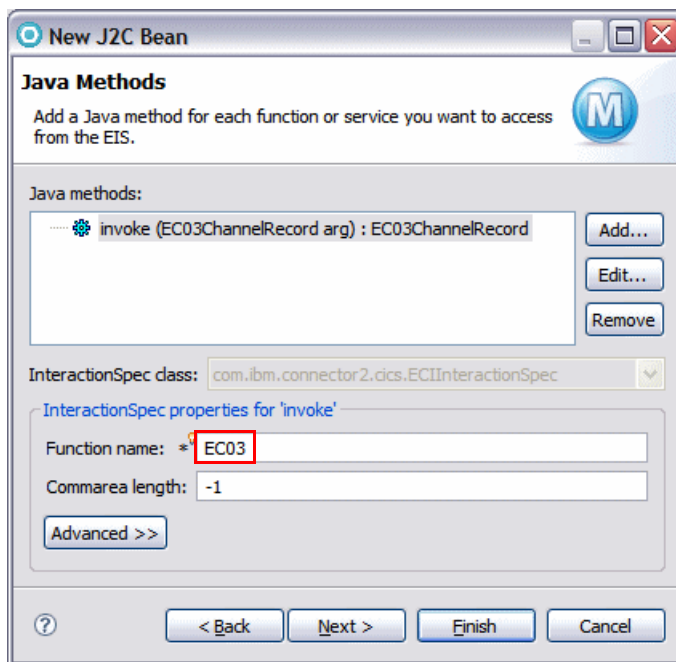


Figure 11-22 Specifying the function name in the InteractionSpec (COBOL)

7. Select the **Create a Web page, Web Service, or EJB from the J2C bean** check box on the New J2C Bean window.

11.5.3 Developing a web service to invoke the COBOL program

In the Deployment Information window (Figure 11-23 on page 558), complete the following actions:

1. Select the **Web Service** radio button.

The **Learn more** link toward the bottom of the window points to a product help page. This page states that the code generated by the J2C bean tool does not allow for serialization. Therefore, the bottom-up approach for web service creation is not readily available.

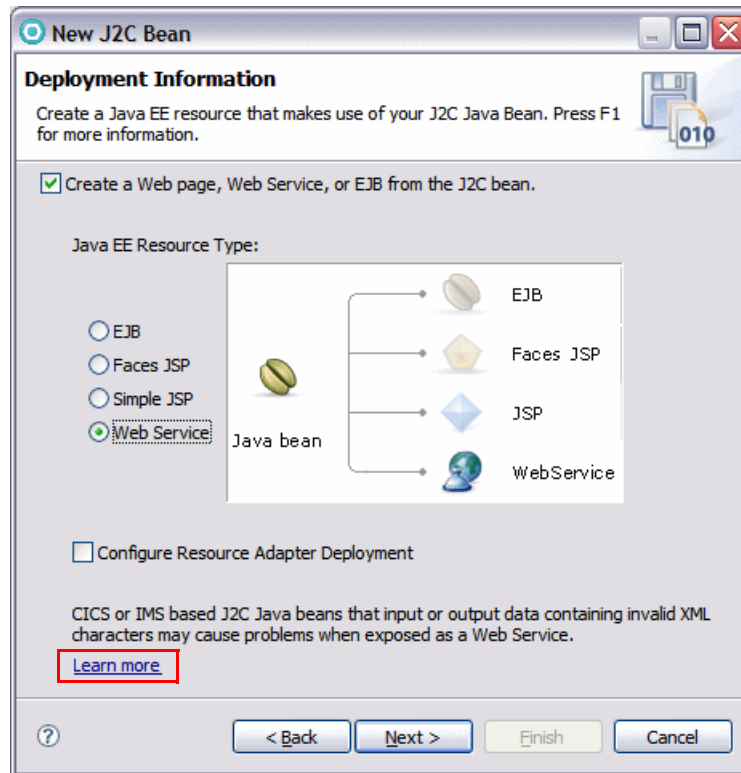


Figure 11-23 Link to online help in the Web Service wizard

2. Click **Next**.
3. In the Web Service Creation window, for the web project, click **New** and create a web project named CICSChannelWeb, as shown in Figure 11-24 on page 559.

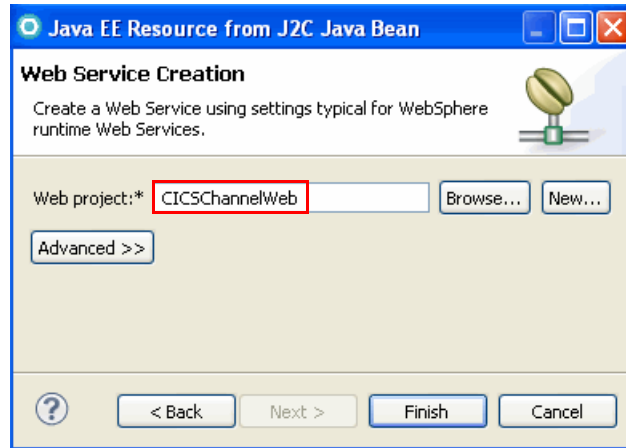


Figure 11-24

4. Click **Finish**. The Ec03 interface and the Ec03Impl class are generated into the CICSCChannel project.

Using an alternative approach

In this simple case, create a JavaBean wrapper that encapsulates the more complex types so that you can run a bottom-up code generation:

1. In the CICSCChannel project, add an Ec03Wrapper class with the code shown in Example 11-4.

Example 11-4 Ec03Wrapper.java

```
package sample.cics;

import javax.resource.ResourceException;

import sample.cics.data.EC03ChannelRecord;
import sample.cics.data.InputContainer;

public class Ec03Wrapper {

    public String invoke(String in) throws ResourceException{
        Ec03Impl test = new Ec03Impl();
        InputContainer inputContainer = new InputContainer();
        inputContainer.setInputContainer_inputcontainer(in);
        EC03ChannelRecord inputRecord = new EC03ChannelRecord();
        inputRecord.setInputContainer(inputContainer);
        EC03ChannelRecord output = test.invoke(inputRecord);
        return output.toString();
    }
}
```

```
}  
  
}
```

2. Create a new dynamic web project called `CICSChannelWeb` associated with the EAR `CICSChannelEAR`. (Use the default module Version 3.0 and the v8.0 target server.)
3. In the Enterprise Explorer, expand **CICSChannelEAR**, right-click **Modules**, and select **Modify**.
4. Select **CICSChannel** (listed as Utility JAR) and **cicseci8000** (listed as Module). Click **OK**.
5. Start WebSphere Application Server V8.0 Beta if it is not already running.
6. Right-click **Ec03Wrapper.java** (in `CICSChannel`) and select **Web Services** → **Create Web service**.

7. In the Web Service wizard (Figure 11-25), accept all the defaults and click **Next**.

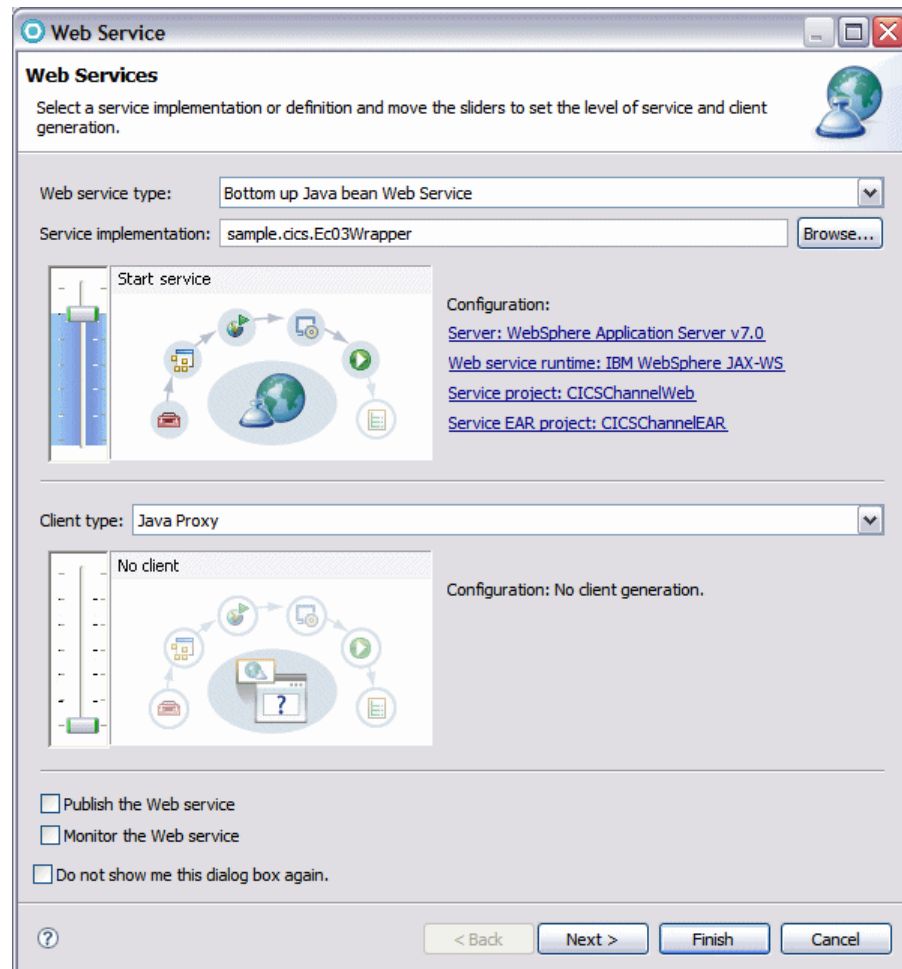


Figure 11-25 Web Service wizard

8. In the next window, select **Generate WSDL file into the project** (to see the generated WSDL) and **Generate Web Service Deployment Descriptor**.
9. In the next window, accept all the default settings.
10. When you reach the last window in the wizard, click **Finish**.

In the Enterprise Explorer, CICSChannelWeb project (Figure 11-26 on page 562), you see the following information:

- ▶ The Services node with Ec03WrapperService

- ▶ The WSDL file (Ec03WrapperService.wsdl) under WEB-INF/wsdl, with an associated XML schema (Ec03WrapperService_schema1.xsd)
- ▶ The web service deployment descriptor (webservices.xml)

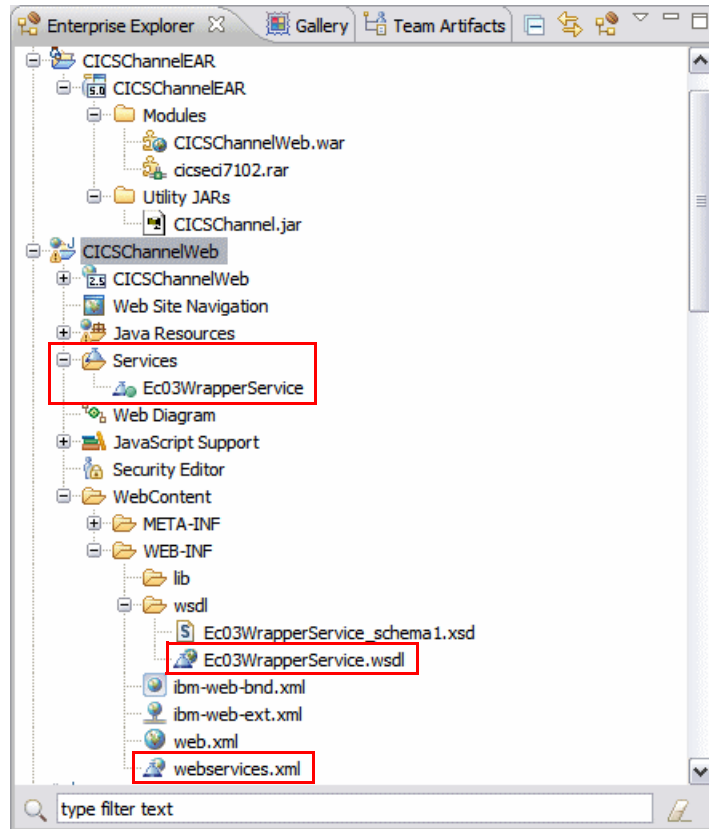


Figure 11-26 Enterprise Explorer view after generating a bottom-up web service

The CICSCChannelEAR enterprise application is deployed to the server.

11.5.4 Testing the web service with CICS access

To test the generated code, we use the Web Services Explorer:

1. Right-click **Ec03WrapperService.wsdl** and select **Web Services** → **Test with Generic Services Client**.
2. In the GSC Explorer (Figure 11-27 on page 563), complete the following steps:
 - a. Select **Edit Data** → **Message** → **Form**.

- b. For the arg parameter, select **OUTPUTCONTAINER** → **recordName**.
 - c. Type Hello as the value.
 - d. Click **Invoke**.
3. The web service is invoked.

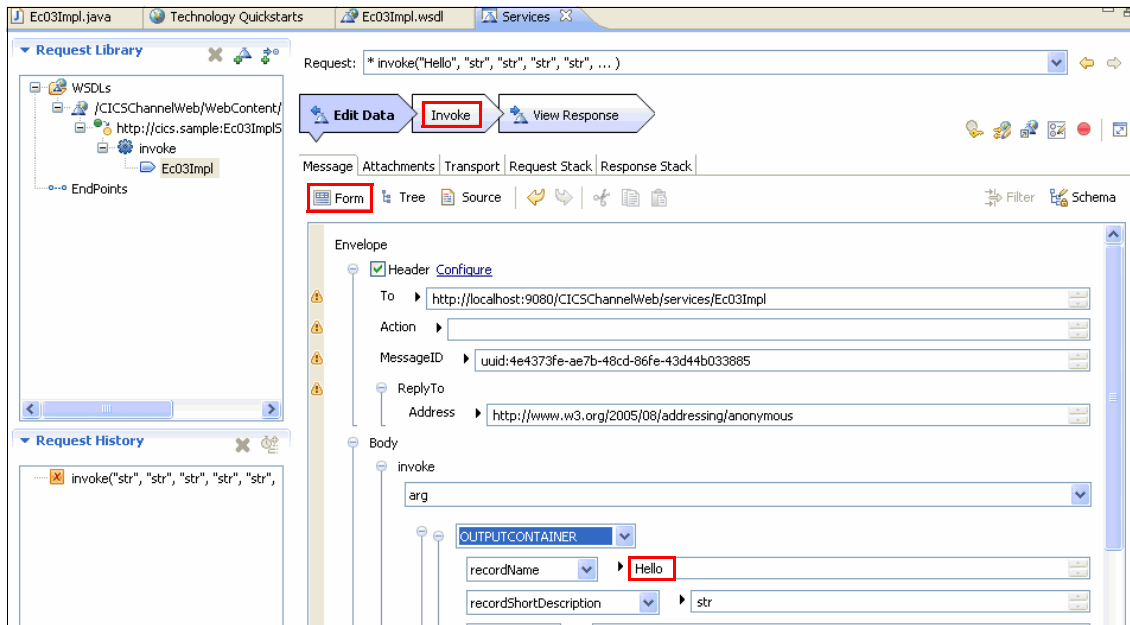


Figure 11-27 GSC Explorer

4. Click **Source** in the Status pane to see the SOAP messages. Example 11-5 shows the SOAP message result.

Example 11-5 Web service response (formatted for readability)

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Body>
<invokeResponse xmlns:ns2="http://cics.sample/">
<return>
sample.cics.data.EC03ChannelRecord@4f474f47
sample.cics.data.OutputContainer@5e905e90 Input data was:
Hello
sample.cics.data.InputContainer@5f005f00 Hello
sample.cics.data.LengthContainer@5f255f25 Buffer Size: 4
bytes
```

```
00000010 00000000 00000000 00000000 |.....|
</return>
</invokeResponse>
</soapenv:Body>
</soapenv:Envelope>
```

11.6 SAP outbound scenario

This sample demonstrates how to use the WebSphere SAP resource adapter to create and retrieve information about an SAP system.

11.6.1 Required software and configuration

To complete the SAP adapter sample in this chapter, you must have the following software installed:

- ▶ IBM Rational Application Developer V8.0 (required)
- ▶ J2EE Connector (J2C) tools

To install the J2C tools, follow these steps:

- a. Open the Installation Manager. Click **Modify** and click **Next**.
 - b. In the Modify Packages page, select **IBM Software Deliver Platform** and click **Next**.
 - c. In the Features list, select **IBM Rational Application Developer for WebSphere Software 8.0** and click **Next**.
 - d. On the Install Packages page, select **Java EE Connector (J2C) Tools** → **Java EE Connector (J2C) Tools** → **WebSphere Adapters**.
 - e. Click **Install**.
- ▶ The following files:
 - The sapjco.jar file
 - The librfr32.dll file
 - The sapjcorfc.dll file

Obtain these files from your SAP server administrator and add them to the following directories:

- Copy the sapjco.jar file to the <WAS_DIR>\lib directory.
- Copy the librfr32.dll file to the <WAS_DIR>\bin and <WAS_DIR>\java\jre\bin directories.

- Copy the sapjcorfc.dll file to the <WAS_DIR>\bin and <WAS_DIR>\java\jre\bin directories.

Note that <WAS_DIR> = <RAD_HOME>\runtimes\base_v8.

11.6.2 Creating a connector project and J2C beans

To create the SAP connector project and related J2C beans for creating and retrieving information to and from the SAP system, follow these steps:

1. Open the Java EE perspective.
2. In the workbench, select **File** → **New** → **Other**.
3. In the Select a wizard window, click **J2C** → **J2C Bean**.
4. In the New J2C Bean window (Figure 11-28), select **IBM WebSphere Adapter for SAP Software** and click **Next**.

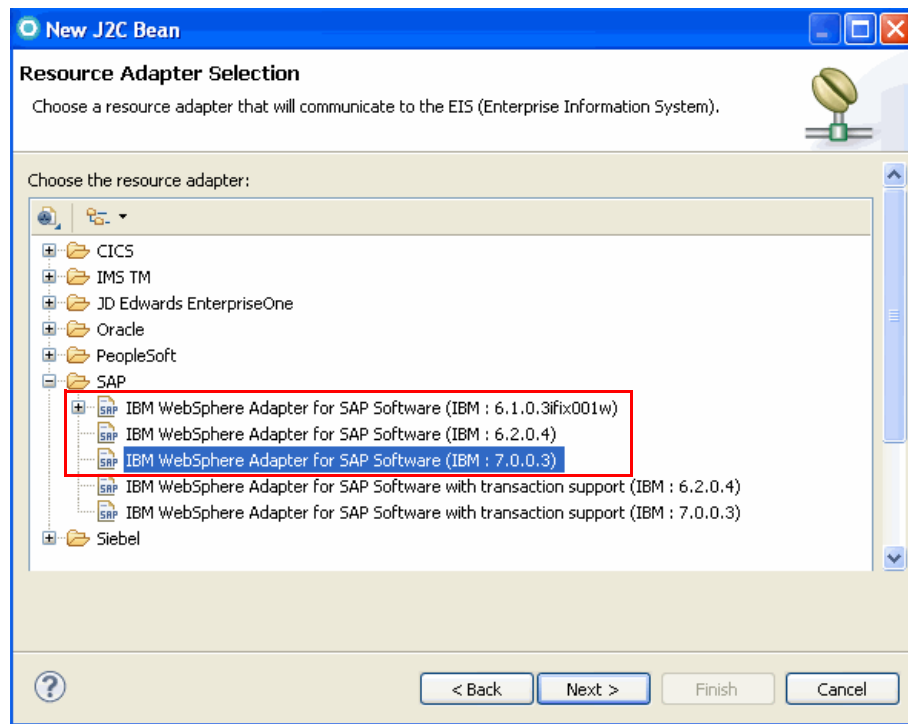


Figure 11-28 New J2C Bean: Resource Adapter Selection

5. In the Connector Import window, complete the following actions:
 - a. In the Connector project field, type CWYAP_SAPAdapter.
 - b. For the Target server, select **WebSphere Application Server v8.0 Beta**.
 - c. Click **Next**.
6. In the Connector Settings window, for each file that is required to access the SAP server, click **Browse**. Navigate to the **sapjco.jar**, **librfc32.dll**, and **sapjcorfc.dll** files that we copied to the server folders. Click **Next**.
7. In the Adapter Style window (Figure 11-29), select **Outbound** and click **Next**.

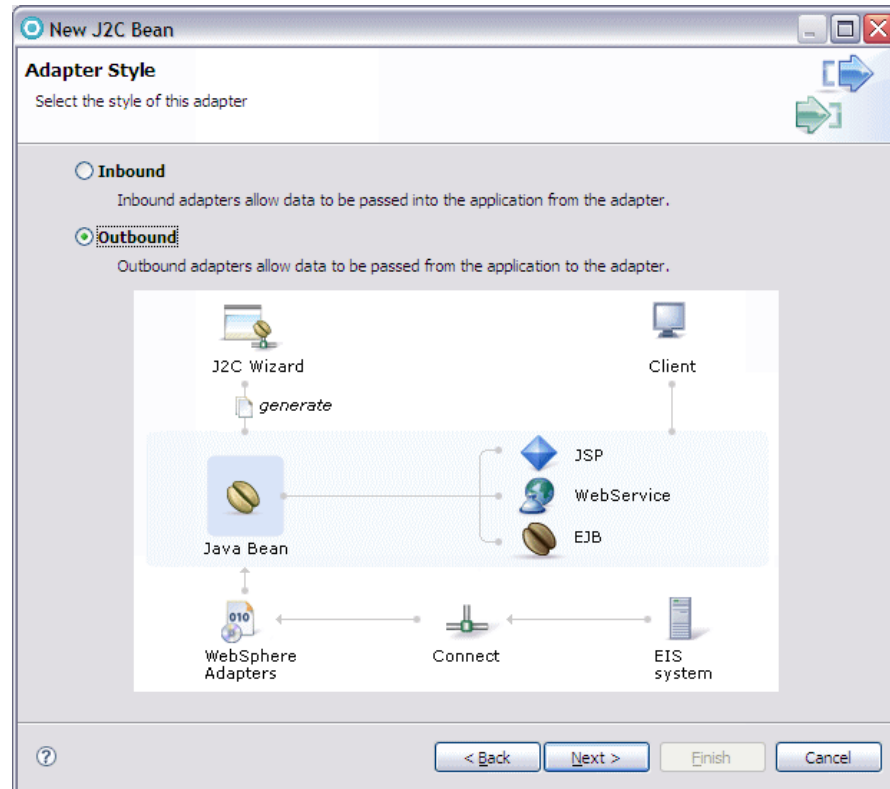


Figure 11-29 New J2C Bean: Adapter Style

8. In the Discovery Configuration window (Figure 11-30), enter your SAP server connection information. For this sample, for the SAP interface name, select **BAPI**. Click **Next**. The wizard retrieves the objects that were discovered by the query.

New J2C Bean

Discovery Configuration

The wizard can now guide you through discovery of objects to communicate with. Specify settings to begin discovery.

Connection Configuration

SAP system connection information

Host name: * 9.26.247.94

System number: 00

Client: 001

Language code: EN (English)

Code page: 1100


User name: * CHENELL

Password: * *****

SAP interface name: BAPI

Specify the level of the logging desired

Figure 11-30 New J2C Bean: Discovery Configuration

9. In the Object Discovery and Selection window (the left window in Figure 11-31), complete the following actions:
 - a. Under Objects discovered by query, select **RFC**.
 - b. Click the **Create or edit filter** () button.
 - c. In the Filter Properties for 'RFC' window (the right window in Figure 11-31), in the Find objects with this pattern field, type `BAPI_CUSTOMER_*` and click **OK**.

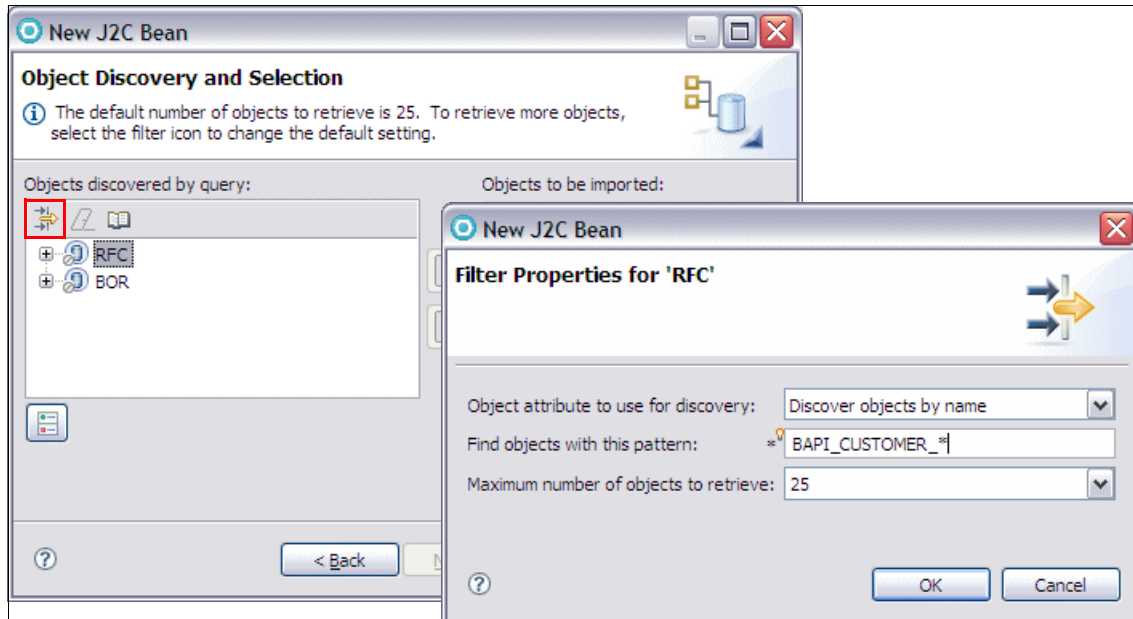



Figure 11-31 New J2C Bean: Object Discovery and Selection

10. In the Object Discovery and Selection window (Figure 11-32), complete the following steps:
- Expand **RFC (filtered)**, select **BAPI_CUSTOMER_CREATEFROMDATA1** and **BAPI_CUSTOMER_GETDETAIL**.
 - Click the  button to add them to the Objects to be imported box. Click **Next**.

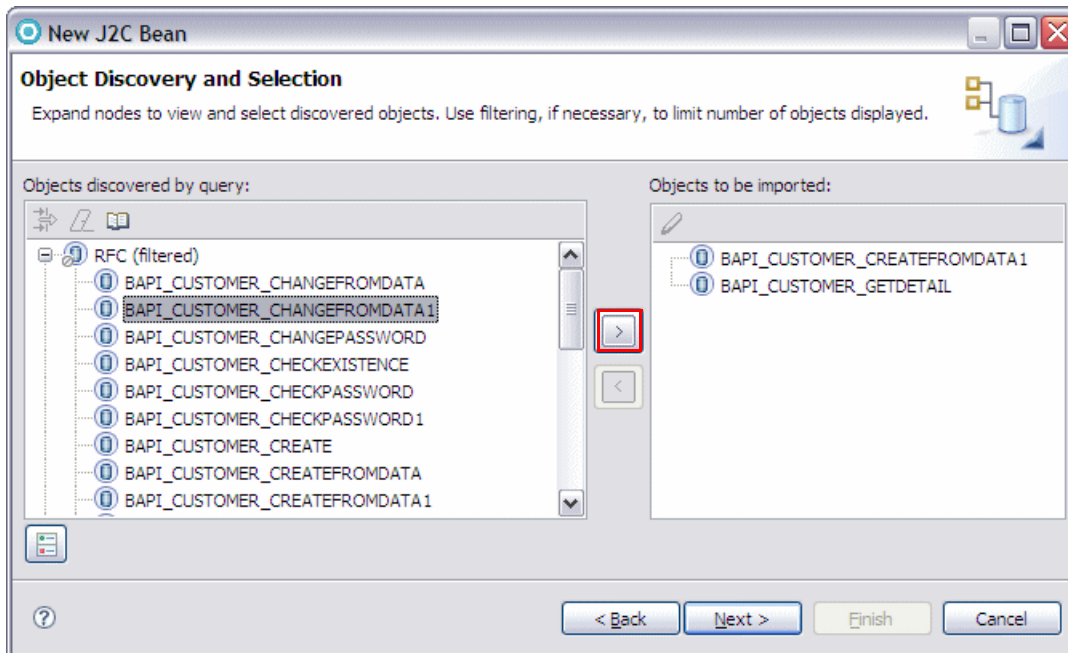


Figure 11-32 New J2C Bean: Object Discovery and Selection

- In the Configuration Parameters window, accept the defaults and click **OK**. Click **Next**.
11. In the Configure Composite Properties window, complete the following steps:
- In the Business objects name for service operations field, type RAD80BAPI.
 - Click **Add**.
 - In the Add Value window, select **Create** and click **OK**.
 - For the RFC function for selected operation, select **BAPI_CUSTOMER_CREATEFROMDATA1**.

- e. Repeat the previous steps, but in the Add value window, select **Retrieve**. For RFC function for selected operation, select **BAPI_CUSTOMER_GETDETAIL**.
- f. Click **Next**.
12. In the J2C Bean Creation and Deployment Configuration window, in the Project name field, click **New** to create a new Java project that contains the generated J2C beans.
13. In the New Source Project Creation window, select **Java project** and click **Next**.
14. In the Create a Java project window, for Project name, type RAD80SAP, accept the defaults, and click **Finish**.
15. In the J2C Bean Creation and Deployment Configuration window, complete the following actions:
 - a. For Package Name, type `itso.rad80.babi`.
 - b. For Interface Name, type `Customer`.
 - c. For Implementation Name, type `CustomerImpl`.
 - d. Clear **Managed Connection** and select **Non-managed Connection**.
 - e. Click **Finish**.

After finishing with the J2C Wizard, the following projects are in the workspace:

- ▶ CWYAP_SAPAdapter: SAP Adapter Project
- ▶ RAD80SAP: Java Project that holds the generated J2C beans

To test the generated J2C beans, generate a simple web application using J2C wizards.

11.6.3 Generating the sample web application

To generate the simple web application, follow these steps:

1. In the workbench, select **File** → **New** → **Other**.
2. In the Select a wizard window, expand **J2C** → **Web Page, Web Service, or EJB from J2C Java Bean** and click **Next**.
3. In the Java EE Resource from J2C Bean window, for J2C Bean implementation, click **Browse** and select **CustomerImpl**. Click **Next**.
4. In the Deployment Information window (Figure 11-33 on page 571), for Java EE Resource Type, select **Simple JSP** and click **Next**.

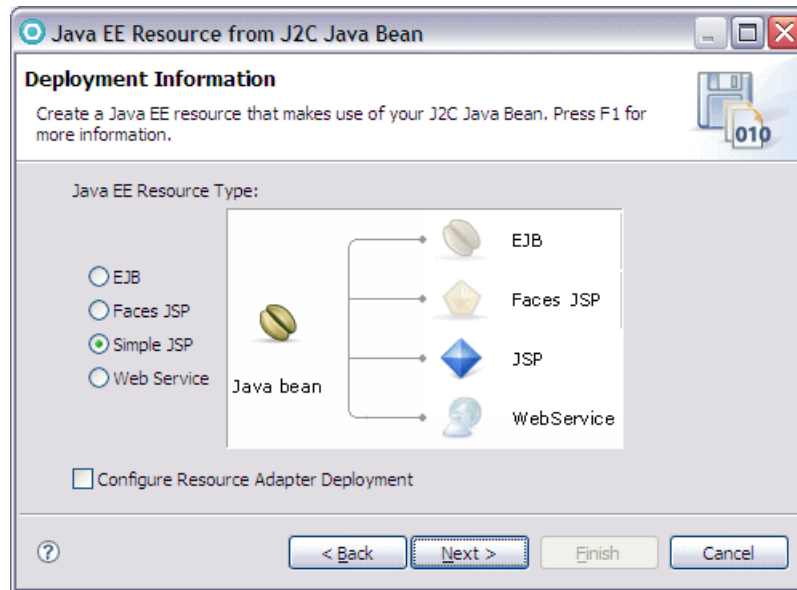


Figure 11-33 Deployment Information

5. In the Simple JSP Creation window, in the Web project field, click **New** to create a new web project.
6. In the Dynamic Web Project window, for the Project name, type RAD80SAPTestWeb, and for the EAR Project Name, type RAD80SAPTestWebEAR. For the Target Runtime, ensure that **WebSphere Application Server v8.0 Beta** is selected. Click **Finish**.
7. In the Simple JSP Creation window, for JSP Folder, type Samp1eJSP. Click **Finish**.

After you complete the steps in the Simple JSP Creation wizard, the following projects are created in your workspace:

- RAD80SAPTestWeb
- RAD80SAPTestWebEAR

The RAD80SAPTestWebEAR project has to reference the CWYAP_SAPAdapter project.

8. Configure this dependency by adding the CWYAP_SAPAdapter as a dependency to the EAR project:
 - a. In the Enterprise Explorer, right-click **RAD80SAPTestWebEAR** and select **Properties**. Select **Java EE Module Dependencies** and select **CWYAP_SAPAdapter**. Click **OK**.

- b. Right-click **RAD80SAPTestWeb** and select **Properties**. Select **Java EE Module Dependencies** and select **CWYAP_SAPAdapter.rar**. Click **OK**.

11.6.4 Testing the web application

To test the sample web application, follow these steps:

1. Expand **RAD80SAPTestWeb** → **WebContent** → **SampleJSP**, right-click **TestClient.jsp**, and select **Run As** → **Run on Server**. Select the server (**WebSphere Application Server v8.0 Beta**) and click **Finish**.

The application opens in a web browser.

2. From the Methods Pane, click **createSapRAD80BAPIWrapper**. Scroll down the page for Inputs, enter the input values for **sapBapiCustomerCreatefromdata1Input**, and click **Invoke**.

The response from the SAP system is displayed in the Results pane.

3. From the Methods Pane, click **retrieveSapRAD80BAPIWrapper**. Scroll down the page for **sapBapiCustomerGetdetail**, enter an input value for **customerToBeRequired**, and click **Invoke**.

The response from the SAP system is displayed in the Results pane.

11.7 Monitoring inbound events for resource adapters

The adapter supports monitoring inbound events from the Oracle database, in addition to the other events that you monitor using WebSphere Business Monitor or WebSphere Business Events.

11.7.1 Monitoring inbound events using WebSphere Business Monitor

You can use Rational Application Developer V8 for WebSphere Software and Adapter for Oracle E-Business Suite to send inbound events to WebSphere Application Server Common Event Infrastructure (CEI), where they are accessible to WebSphere Business Monitor.

For more information, you can visit this website:

http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.wsadapters.rad.jca_oracleebiz.doc/shared/csha_mon_inevents_wbm.html

11.7.2 Monitoring inbound events using WebSphere Business Events

You can use Rational Application Developer and the Adapter for Oracle E-Business Suite to send inbound events to the WebSphere Application Server JMS topic, where they are accessible to WebSphere Business Events.

For more information, you can visit:

http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.wsadapters.rad.jca_oracleebiz.doc/shared/csha_mon_inevents_wbe.html

11.8 More information

See the following sections in the product help for more information:

- ▶ Developing → Developing Data Access Applications → Connecting to enterprise information systems
- ▶ Tutorials → Watch and learn → Create a J2C application for a CICS transaction with the same input and output
- ▶ Tutorials → Do and learn → Create a J2C application for a CICS transaction containing multiple possible outputs
- ▶ Samples → Technology Samples → Java → J2C Samples
- ▶ Cheat Sheets → J2C Java Bean

In addition, see *Revealed! The Next Generation of Distributed CICS*, SG24-7185, and the following sources of information:

- ▶ Generating a J2C bean using the J2C Tools in Rational Application Developer V7.0:
http://www.ibm.com/developerworks/rational/library/06/1212_nigu1/
- ▶ Create a J2C application for an Information Management System (IMS) phone book transaction using IMS Resource Adapter:
<http://www.ibm.com/developerworks/rational/library/08/dw-r-j2cimsresource/>
- ▶ Working with J2C Ant Scripts in Rational Application Developer V7:
http://www.ibm.com/developerworks/rational/library/06/1205_ho-benedek/

Enterprise and service-oriented architecture (SOA) application development

In this part, we describe the tooling and technologies provided by Rational Application Developer to develop enterprise applications.

This part includes the following chapters:

- ▶ Chapter 12, “Developing Enterprise JavaBeans (EJB) applications” on page 577
- ▶ Chapter 13, “Developing Java Platform, Enterprise Edition (Java EE) application clients” on page 649

- ▶ Chapter 14, “Developing web services applications” on page 681
- ▶ Chapter 15, “Developing Open Services Gateway initiative (OSGi) applications” on page 837
- ▶ Chapter 16, “Developing Service Component Architecture (SCA) applications” on page 885
- ▶ Chapter 17, “Developing Modern Batch jobs on computing grids” on page 957

Sample code for download: The sample code for all the applications developed in this part is available for download at the following address:

<ftp://www.redbooks.ibm.com/redbooks/SG247835>

See Appendix C, “Additional material” on page 1877, for instructions.



Developing Enterprise JavaBeans (EJB) applications

In this chapter, we introduce Enterprise JavaBeans (EJB) and demonstrate by example how to create, maintain, and test EJB components. We explain how to develop session beans and describe the relationships between the session beans and the Java Persistence API (JPA) entity beans. Then we integrate the EJB with a front-end web application for the sample application. We include examples for creating, developing, and testing the EJB using Rational Application Developer.

The chapter is organized into the following sections:

- ▶ Introduction to Enterprise JavaBeans
- ▶ Developing an EJB module
- ▶ Testing the session EJB and the JPA entities
- ▶ Invoking EJBs from web applications
- ▶ More information

The sample code for this chapter is in the 7835code\ejb folder.

12.1 Introduction to Enterprise JavaBeans

EJB is an architecture for server-side component-based distributed applications written in Java. Details of the EJB 3.1 specification, EJB components and services, and new features in Rational Application Developer are described in the following sections:

- ▶ EJB 3.1 specification
- ▶ EJB component types
- ▶ EJB services and annotations
- ▶ EJB 3.1 application packaging
- ▶ EJB 3.1 Lite
- ▶ EJB 3.1 features in Rational Application Developer

12.1.1 EJB 3.1 specification

The EJB 3.1 specification is defined in *Java Specification Request (JSR) 318: Enterprise JavaBeans 3.1*. As described in 10.1, “Introducing the Java Persistence API” on page 444, the JPA 2.0 specification and EJB 3.1 specification are separate. The specification of JPA 1.x was included in the EJB 3.0 specification. We describe the usage of the EJB 3.x specification, with focus on EJB 3.1 in this chapter.

EJB 3.1 simplified model

Many publications discuss the complexities and differences between the old EJB 2.x programming model and the new EJB 3.x. For this reason, in this book, we focus on the new programming model. To overcome the limitations of the EJB 2.x, the new specification introduces a new simplified model with the following features:

- ▶ Entity EJBs are now JPA entities, plain old Java objects (POJO) that expose regular business interfaces, as plain old Java interface (POJI), and there is no requirement for home interfaces.
- ▶ The requirement for specific interfaces and deployment descriptors has been removed (deployment descriptor information can be replaced by annotations).
- ▶ A completely new persistence model, which is based on the JPA standard (see Chapter 10, “Persistence using the Java Persistence API” on page 443), replaces EJB 2.x entity beans.
- ▶ An interceptor facility is used to invoke user methods at the invocation of business methods or at life-cycle events.
- ▶ Default values are used whenever possible (“configuration by exception” approach).

- ▶ Requirements for the use of checked exceptions have been reduced.
- ▶ EJB 3.1 Lite, as a minimal subset of the full EJB 3.1 API, offers the major functionality of EJB 3.1 API, as described in detail in 12.1.5, “EJB 3.1 Lite” on page 600.

Figure 12-1 shows how the model of Java 2 Platform, Enterprise Edition (J2EE) 1.4 has been completely reworked with the introduction of the EJB 3.x specification. With the EJB 3.1 specification, this model has been updated with new features, summarized in 12.1.6, “EJB 3.1 features in Rational Application Developer” on page 601.

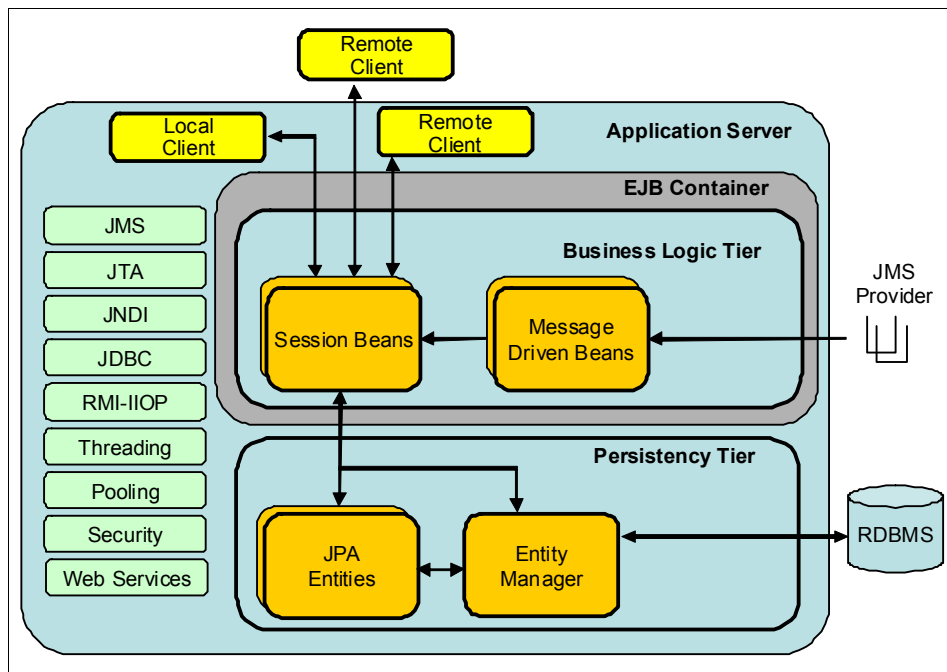


Figure 12-1 EJB 3.1 architecture

12.1.2 EJB component types

EJB 3.1 has the following component types of EJBs:

- ▶ Session beans: stateless
- ▶ Session beans: stateful
- ▶ Session beans: Singleton bean
- ▶ Message-driven EJB (MDB)

This section defines several EJBs.

Session beans

There are several kinds of session beans, the stateless and stateful EJB, and as a new feature, the definition of Singleton session beans. In this section, we describe these tasks:

- ▶ Defining a stateless session bean in EJB 3.1
- ▶ Defining a stateful session bean in EJB 3.1
- ▶ Defining a Singleton session bean in EJB 3.1

Additionally, we show the life-cycle events and leading practices for developing session beans.

Defining a stateless session bean in EJB 3.1

Stateless session EJBs have always been used to model a task being performed for client code that invokes it. They implement the business logic or rules of a system, and provide the coordination of those activities between beans, such as a banking service that allows for a transfer between accounts.

A stateless session bean is generally used for business logic that spans a single request and therefore cannot retain the client-specific state among calls.

Because a stateless session bean does not maintain a conversational state, all the data exchanged between the client and the EJB has to be passed either as input parameters, or as a return value, declared on the business method interface.

To declare a session stateless bean, add the `@Stateless` annotation to a POJO as shown in Example 12-1.

Example 12-1 Definition of a stateless session bean

@Stateless

```
public class MyFirstSessionBean implements MyBusinessInterface {  
  
    // business methods according to MyBusinessInterface  
    .....  
}
```

Note the following points in Example 12-1:

- ▶ `MyFirstSessionBean` is a POJO that exposes a POJI, in this case, `MyBusinessInterface`. This interface is available to clients to invoke the EJB business methods. For EJB 3.1, a business interface is not required.
- ▶ The `@Stateless` annotation indicates to the container that the given bean is a stateless session bean so that the proper life-cycle and runtime semantics can be enforced.

- By default, this session bean is accessed through a local interface.

This is all the information that you need to set up a session EJB. There are no special classes to extend and no interfaces to implement.

Figure 12-2 shows the simple model of EJB 3.1.

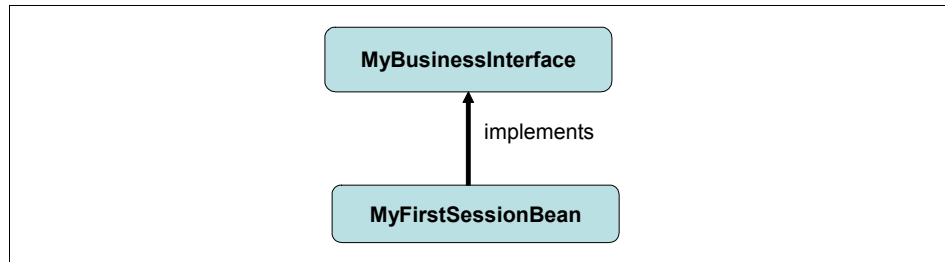


Figure 12-2 EJB is a POJO exposing a POJI

If we want to expose the same bean on the remote interface, we use the `@Remote` annotation, as shown in Example 12-2.

Example 12-2 Defining a remote interface for stateless session bean

```
@Remote(MyRemoteBusinessInterface.class)
@Stateless
public class MyBean implements MyRemoteBusinessInterface {
    // ejb methods
    .....
}
```

Tip: If the session bean implements only one interface, you can use the `@Remote` annotation without a class name.

Defining a stateful session bean in EJB 3.1

Stateful session EJBs are typically used to model a task or business process that spans multiple client requests. Therefore, a stateful session bean retains its state on behalf of an individual client. The client has to store the handling of the stateful EJB, so that it always accesses the same EJB instance. Using the same approach that we adopted before, to define a stateful session EJB, you have to declare a POJO with the annotation `@Stateful`, as shown in Example 12-3.

Example 12-3 Defining a stateful session bean

```
@Stateful
public class SecondSessionBean implements MyBusinessStatefulInterface {
    // ejb methods
```

```
.....  
}
```

The `@Stateful` annotation indicates to the container that the given bean is a stateful session bean so that the proper life-cycle and runtime semantics can be enforced.

Defining a Singleton session bean in EJB 3.1

The definition of a Singleton session bean is a new feature of EJB 3.1. The definition of the Singleton pattern is associated with the defined design pattern in software engineering. You define a session bean as a Singleton to restrict the instantiation of this class to only one object. For example, the object, which coordinates actions across the system, can be defined as a Singleton, because only this object is responsible for the coordination. Therefore, you define the annotation `@Singleton` in front of the class declaration, as shown in Example 12-4. The new annotation `@LocalBean` is described in “Business interfaces” on page 583.

Example 12-4 Defining a Singleton session bean

```
@Startup  
@Singleton  
@LocalBean  
public class MySingletonBean{  
  
    // ejb methods  
    .....  
}
```

A Singleton session bean offers you the opportunity to initialize objects at application start-up. This functionality can replace proprietary WebSphere Application Server start-up beans. Therefore, you define the new annotation `@Startup` additionally in front of the class declaration, as shown in Example 12-4. As result, you have the initialization at the application start-up instead of the first invocation by the client code.

The concurrency in Singleton session beans is either defined as container-managed concurrency (CMC) or bean-managed concurrency (BMC). In case no annotation for the concurrency is specified in front of the class, the default value is CMC. The further default value for CMC is `@Lock(WRITE)`. If you want to define a class or method associated with a shared lock, use the annotation `@Lock(READ)`.

To define BMC, use the annotation `@ConcurrencyManagement` (`ConcurrencyManagementType.BEAN`). After it has been defined as BMC, the

container allows full concurrent access to the Singleton session bean. Furthermore, you can define that concurrency is not allowed. Therefore, use the annotation `@ConcurrencyManagement` (`ConcurrencyManagementType.CONCURRENCY_NOT_ALLOWED`).

For detailed information about Singleton session beans, see section 3.4.7.3 “Singleton Session Beans” in *JSR 318: Enterprise JavaBeans 3.1*.

Business interfaces

EJBs can expose various business interfaces, because the EJB can be accessed from either a local or remote client. Therefore, place common behavior to both local and remote interfaces in a superinterface, as shown in Figure 12-3.

You have to ensure the following aspects:

- ▶ A business interface cannot be both a local and a remote business interface of the bean.
- ▶ If a bean class implements a single interface, that interface is assumed to be the business interface of the bean. This business interface is a *local* interface, unless the interface is designated as a remote business interface by use of the `@Remote` annotation or by means of the deployment descriptor.

This approach provides flexibility during the design phase, because you can decide which methods are visible to local and remote clients.

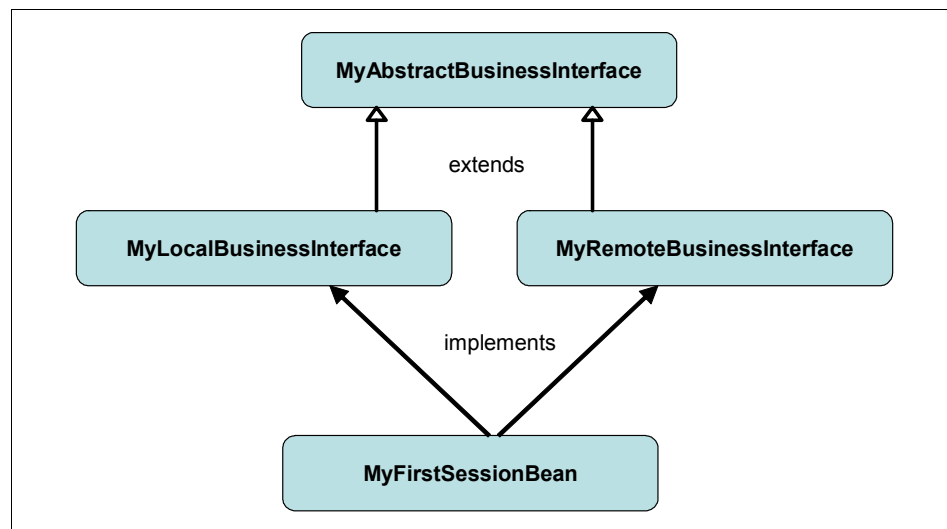


Figure 12-3 How to organize the EJB component interfaces

Using these guidelines, the first EJB is refactored, as shown in Example 12-5.

Example 12-5 Implementing local and remote interface

```
@Stateless
public class MyFirstSessionBean
    implements MyLocalBusinessInterface, MyRemoteBusinessInterface {

    // implementation of methods declared in MyLocalBusinessInterface
    ....

    // implementation of methods declared in MyRemoteBusinessInterface
    ....
}
```

The MyLocalBusinessInterface is declared as an interface with an @Local annotation, and the MyRemoteBusinessInterface is declared as an interface with the @Remote annotation, as shown in Example 12-6.

Example 12-6 Defining local and remote interface

```
@Local
public interface MyLocalBusinessInterface
    extends MyAbstractBusinessInterface {

    // methods declared in MyLocalBusinessInterface
    .....
}
```

=====

```
@Remote
public interface MyRemoteBusinessInterface
    extends MyAbstractBusinessInterface {

    // methods declared in MyRemoteBusinessInterface
    .....
}
```

Another technique to define the business interfaces exposed either as local or remote is to specify @Local or @Remote annotations with the full class name that implements these interfaces, as shown in Example 12-7.

Example 12-7 Defining full class interfaces

```
@Stateless
@Local(MyLocalBusinessInterface.class)
@Remote(MyRemoteBusinessInterface.class)
```

```
public class MyFirstSessionBean implements MyLocalBusinessInterface,
                                           MyRemoteBusinessInterface {

    // implementation of methods declared in MyLocalBusinessInterface
    ....
    // implementation of methods declared in MyRemoteBusinessInterface
    ....
}
```

You can declare any exceptions on the business interface, but be aware of the following rules:

- ▶ Do not use `RemoteException`.
- ▶ Any runtime exception thrown by the container is wrapped into an `EJBException`.

As a new feature of EJB 3.1, you can define a session bean without a local business interface. Therefore, the local view of a session bean can be accessed without the definition of a local business interface.

As shown in Figure 12-4 on page 586, there is a new check box available, to select and define that no interface has to be created.

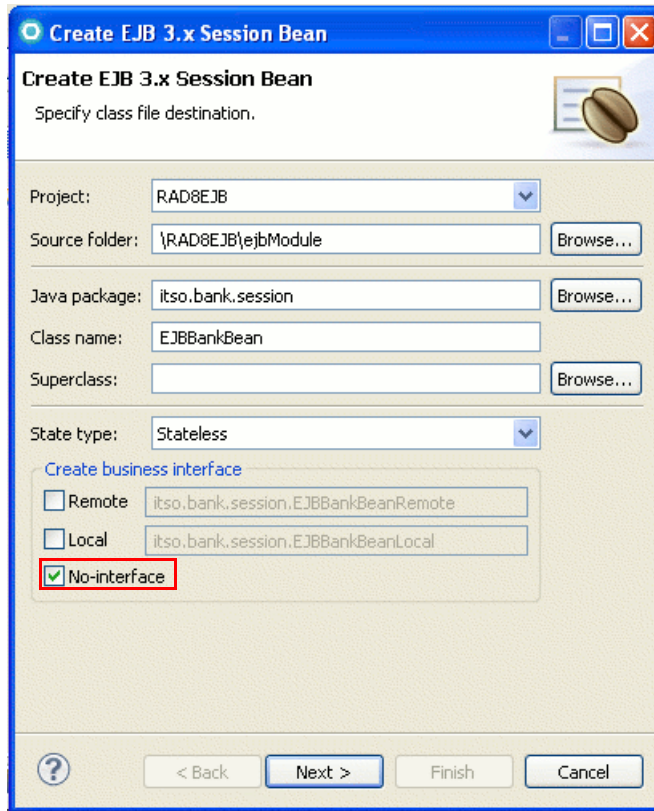


Figure 12-4 Creating a session bean without an interface

If you select No-interface in the Create EJB 3.x Session Bean wizard, the `@LocalBean` annotation will be generated for your session bean, as shown in Example 12-8.

Example 12-8 Session bean with No-interface view

```

package itso.bank.session;
import javax.ejb.LocalBean;
import javax.ejb.Stateless;

/**
 * Session Bean implementation class EJBBankBean
 */
@Stateless
@LocalBean
public class EJBBankBean{
/**

```

```
* Default constructor.  
*/  
    public EJBBankBean() {  
        // TODO Auto-generated constructor stub  
    }  
    ...  
}
```

For detailed information, see section 3.4.4 “Session Bean’s No-Interface View” in the *JSR 318: Enterprise JavaBeans 3.1* specification.

Life-cycle events

Another powerful use of annotations is to *mark* callback methods for session bean life-cycle events.

EJB 2.1 and prior releases required the implementation of several life-cycle methods, such as `ejbPassivate`, `ejbActivate`, `ejbLoad`, and `ejbStore`, for every EJB, even if you did not need these methods.

Life-cycle methods: As we use POJOs in EJB 3.x, the implementation of these life-cycle methods is optional. The container invokes any callback method if you implement it in the EJB.

The life cycle of a session bean can be categorized into several phases or events. The most obvious two events of a bean life cycle are the creation and destruction for stateless session beans.

After the container creates an instance of a session bean, the container performs any *dependency injection* (described in the following section) and then invokes the method annotated with `@PostConstruct` (if there is one).

The client obtains a reference to a session bean and invokes a business method.

Life cycle of a stateless session bean: The life cycle of a stateless session bean is independent of when a client obtains a reference to it. For example, the container might give a reference to the client, but not create the bean instance until later, when a method is invoked on the reference. In another example, the container might create several instances at start-up and match them with references later.

At the end of the life cycle, the EJB container calls the method annotated with `@PreDestroy` (if there is one). The bean instance is ready for garbage collection.

Example 12-9 shows a stateless session bean with the two callback methods.

Example 12-9 Stateless session bean with two callback methods

```
@Stateless
public class MyStatelessBean implements MyBusinessLogic {
// .. bean business method

    @PostConstruct
    public void initialize() {
        // initialize the resources uses by the bean
    }

    @PreDestroy
    public void cleanup() {
        // deallocates the resources uses by the bean
    }
}
```

All stateless and stateful session EJBs go through these two phases.

In addition, stateful session beans go through the passivation and activation cycle. An instance of a stateful bean is bound to a specific client, and therefore, it cannot be reused among various requests. The EJB container has to manage the amount of available physical resources, and might decide to deactivate, or *passivate*, the bean by moving it from memory to secondary storage.

In correspondence with this more complex life cycle, we have further callback methods, specific to stateful session beans:

- ▶ The EJB container invokes the method annotated with `@PrePassivate`, immediately before passivating it.
- ▶ If a client invokes a business method on the bean while it is in the passive stage, the EJB container activates the bean by calling the method annotated with `@PostActivate` and then moves it to the ready stage.

At the end of the life cycle, the client explicitly invokes a method annotated with `@Remove`, and the EJB container calls the callback method annotated `@PreDestroy`. Developers can explicitly invoke only the life-cycle method annotated with `@Remove`. The other methods are invoked automatically by the EJB container.

Stateful session beans: Because a stateful bean is bound to a particular client, it is a leading practice to correctly design stateful session beans to minimize their footprints inside the EJB container. Also, it is a leading practice to correctly unallocate it at the end of its life cycle, by invoking the method annotated with `@Remove`.

Stateful session beans have a *timeout* value. If the stateful session bean has not been used in the timeout period, it is marked inactive and is eligible for automatic deletion by the EJB container. Of course, it is still a leading practice for applications to remove the bean when the client is finished with it, rather than relying on the timeout mechanism.

Leading practices for developing session EJB

As a leading practice, EJB 3.x developers follow these guidelines:

- ▶ Each session bean has to be a POJO, the class must be concrete, and it must have a no-argument constructor. If the no-argument constructor is not present, the compiler inserts a default constructor.
- ▶ If the business interface is annotated as `@Remote`, all the values passed through the interface must implement `java.io.Serializable`. Typically, the declared parameters are defined as `serializable`, but this is not required as long as the actual values passed are serializable.
- ▶ A session EJB can subclass a POJO, but cannot subclass another session EJB.

Message-driven EJB

MDBs are used for the processing of asynchronous Java Message Service (JMS) messages within JEE-based applications. MDBs are invoked by the container on the arrival of a message.

In this way, MDBs can be thought of as another interaction mechanism for invoking EJB, but unlike session beans, the container is responsible for invoking them when a message is received, not a client or another bean.

To define an MDB in EJB 3.x, you must declare a POJO with the `@MessageDriven` annotation, as shown in Example 12-10.

Example 12-10 Declaring a POJO to define an MDB in EJB

```
@MessageDriven(activationConfig = {  
    @ActivationConfigProperty(propertyName="destinationType",  
        propertyValue="javax.jms.Queue"),  
    @ActivationConfigProperty(propertyName="destination",  
        propertyValue="queue/myQueue")  
})
```

```

    })
    public class MyMessageBean implements javax.jms.MessageListener {

        public void onMessage(javax.msg.Message inMsg) {
            //implement the onMessage method to handle the incoming message
            ....
        }
    }
}

```

Note the following features of Example 12-10 on page 589:

- ▶ In EJB 3.x, the MDB class is annotated with the `@MessageDriven` annotation, which specifies a set of activation configuration parameters. These parameters are unique to the particular Java EE Connector Architecture (JCA) 1.5 adapter that is used to drive the MDB. Certain adapters have configuration parameters that let you specify the destination queue of the MDB. If not, the destination name must be specified using a `<message-destination>` entry in the XML binding file.
- ▶ The bean class has to implement the `javax.jms.MessageListener` interface, which defines only one method, `onMessage`. When a message arrives in the queue monitored by this MDB, the container calls the `onMessage` method of the bean class and passes the incoming message as the parameter.
- ▶ Furthermore, the `activationConfig` property of the `@MessageDriven` annotation provides messaging system-specific configuration information.

12.1.3 EJB services and annotations

The use of annotations is important to define EJB services:

- ▶ Interceptors
- ▶ Dependency injection
- ▶ Asynchronous invocations
- ▶ EJB timer service
- ▶ Web services

We describe the definitions of these services, while using annotations, in this section. Additionally, we provide the description of using deployment descriptors and the description of the new features of Portable JNDI name and Embedded Container API in this section.

Interceptors

The EJB 3.x specification defines the ability to apply custom-made interceptors to the business methods of session and MDB beans. Interceptors take the form

of methods annotated with the `@AroundInvoke` annotation, as shown in Example 12-11.

Example 12-11 Applying an interceptor

```
@Stateless
public class MySessionBean implements MyBusinessInterface {

    @Interceptors(LoggerInterceptor.class)
    public Customer getCustomer(String ssn) {
        ...
    }
    ...
}

public class LoggerInterceptor {
    @AroundInvoke
    public Object logMethodEntry(InvocationContext invocationContext)
        throws Exception {
        System.out.println("Entering method: "
            + invocationContext.getMethod().getName());
        Object result = invocationContext.proceed();
        // could have more logic here
        return result;
    }
}
```

Note the following points for Example 12-11:

- ▶ The `@Interceptors` annotation is used to identify the session bean method where the interceptor will be applied.
- ▶ The `LoggerInterceptor` interceptor class defines a method (`logMethodEntry`) annotated with `@AroundInvoke`.
- ▶ The `logMethodEntry` method contains the advisor logic, in this case, it logs the invoked method name, and invokes the `proceed` method on the `InvocationContext` interface to advise the container to proceed with the execution of the business method.

The implementation of the interceptor in EJB 3.x differs from the analogous implementation of the aspect-oriented programming (AOP) paradigm that you can find in frameworks, such as Spring or AspectJ, because EJB 3.x does not support *before* or *after* advisors, only *around* interceptors.

However, *around* interceptors can act as *before* interceptors, *after* interceptors, or both. Interceptor code before the `invocationContext.proceed` call is run

before the EJB method, and interceptor code after that call is run after the EJB method.

A common use of interceptors is to provide preliminary checks, such as validation, security, and so forth, before the invocation of business logic tasks, and therefore, they can throw exceptions. Because the interceptor is called together with the session bean code at run time, these potential exceptions are sent directly to the invoking client.

In Example 12-11 on page 591, we have seen an interceptor applied on a specific method. Alternatively, the `@Interceptors` annotation can be applied at the class level. In this case, the interceptor will be called for every method.

Furthermore, the `@Interceptors` annotation accepts a list of classes, so that multiple interceptors can be applied to the same object.

Default interceptor: To give further flexibility, EJB 3.x introduces the concept of a default interceptor that can be applied on every session or MDB contained inside the same EJB module. A default interceptor cannot be specified using an annotation. Instead, define it inside the deployment descriptor of the EJB module.

Interceptors run in the following execution order:

- ▶ Default interceptor
- ▶ Class interceptors
- ▶ Method interceptors

To disable the invocation of a default interceptor or a class interceptor on a specific method, you can use the `@ExcludeDefaultInterceptors` and `@ExcludeClassInterceptors` annotations, respectively.

Dependency injection

The new specification introduces a powerful mechanism for obtaining Java EE resources, such as Java Database Connectivity (JDBC) data source, JMS factories and queues, and EJB references to inject them into EJB, entities, or EJB clients.

The EJB 3.x specification adopts a *dependency injection* (DI) pattern, which is one of the best ways to implement loosely coupled applications. It is much easier to use and more elegant than older approaches, such as dependency lookup through Java Naming and Directory Interface (JNDI) or container callbacks.

The implementation of dependency injection in the EJB 3.x specification is based on annotations or XML descriptor entries, which allow you to inject dependencies on fields or setter methods.

Instead of complicated XML EJB references or resource references, you can use the @EJB and @Resource annotations to set the value of a field or to call a setter method within your beans with anything registered within JNDI. With these annotations, you can inject EJB references and resource references, such as data sources and JMS factories.

In this section, we show the most common uses of dependency injection in EJB 3.x, such as the @EJB annotation and @Resource annotation.

@EJB annotation

The @EJB annotation is used for injecting session beans into a client. This injection is only possible within managed environments, such as another EJB, or a servlet. We cannot inject an EJB into a JavaServer Faces (JSF)-managed bean or Struts action.

The @EJB annotation has the following optional parameters:

name	Specifies the JNDI name that is used to bind the injected EJB in the environment naming context (<code>java:comp/env</code>).
beanInterface	Specifies the business interface to be used to access the EJB. By default, the business interface to be used is taken from the Java type of the field into which the EJB is injected. However, if the field is a supertype of the business interface, or if method-based injection is used rather than field-based injection, the <code>beanInterface</code> parameter is typically required, because the specific interface type to be used might be ambiguous without the additional information provided by this parameter.
beanName	Specifies a <i>hint</i> to the system of the <code>ejb-name</code> of the target EJB that must be injected. It is analogous to the <code><ejb-link></code> stanza that can be added to an <code><ejb-ref></code> or <code><ejb-local-ref></code> stanza in the XML descriptor.

Example 12-12 shows the code to access a session bean from a Java servlet.

Example 12-12 Injecting an EJB reference inside a servlet

```
import javax.ejb.EJB;
public class TestServlet extends javax.servlet.http.HttpServlet
    implements javax.servlet.Servlet {

    // inject the remote business interface
```

```

@EJB(beanInterface=MyRemoteBusinessInterface.class)
MyAbstractBusinessInterface serviceProvider;

protected void doGet(HttpServletRequest request,
                    HttpServletResponse response)
                    throws ServletException, IOException {
    // call ejb method
    serviceProvider.myBusinessMethod();
    .....
}
}

```

Note the following points regarding Example 12-12 on page 593:

- ▶ We specified the `beanInterface` attribute, because the EJB exposes two business interfaces (`MyRemoteBusinessInterface` and `MyLocalBusinessInterface`).
- ▶ If the EJB exposes only one interface, you are not required to specify this attribute. However, it can be useful to make the client code more readable.

Special notes for stateful EJB injection:

- ▶ Because a servlet is a multi-thread object, you cannot use dependency injection, but you must explicitly look up the EJB through the JNDI.
- ▶ You can safely inject a stateful EJB inside another session EJB (stateless or stateful), because a session EJB instance is guaranteed to be executed by only a single thread at a time.

@Resource annotation

The `@Resource` annotation is the major annotation that can be used to inject resources in a managed component. Therefore, two techniques exist: the field technique and the setter injection technique. In the following section, we show the most commonly used scenarios of this annotation.

Example 12-13 shows how to inject a typical resource, such as a data source inside a session bean using the field injection technique. A data source (`jdbc/datasource`) is injected inside a property that is used in a business method.

Example 12-13 Field injection technique for a data source

```

@Stateless
public class CustomerSessionBean implements CustomerServiceInterface {

    @Resource (name="jdbc/datasource")

```

```

private DataSource ds;
public void businessMethod1() {
    java.sql.Connection c=null;
    try {
        c = ds.getConnection();
        // .. use the connection
    } catch (java.sql.SQLException e) {
        // ... manage the exception
    } finally {
        // close the connection
        if(c!=null) {
            try { c.close(); } catch (SQLException e) { }
        }
    }
}
}
}
}

```

The @Resource annotation has the following optional parameters:

name	Specifies the component-specific internal name, which is the resource reference name, within the java:comp/env namespace. It does not specify the global JNDI name of the resource being injected.
type	Specifies the resource manager connection factory type.
authenticationType	Specifies whether the container or the bean is to perform authentication.
shareable	Specifies whether resource connections are shareable.
mappedName	Specifies a product-specific name to which the resource must be mapped. WebSphere does not make any use of mappedName.
description	Description.

Another technique is to inject a setter method. The setter injection technique is based on JavaBean property naming conventions, as shown in Example 12-14.

Example 12-14 Setter injection technique for a data source

```

@Stateless
public class CustomerSessionBean implements CustomerServiceInterface {

    private Datasource ds;

    @Resource (name="jdbc/dataSource")
    public void setDatasource(DataSource datasource) {

```

```

        this.ds = datasource;
    }
    ...
    public void businessMethod1() {
        ...
    }
}

```

Note the following points regarding Example 12-13 on page 594 and Example 12-14 on page 595:

- ▶ We directly used the data source inside the session bean, which is not a good practice. Instead, place the JDBC code in specific components, such as data access objects.
- ▶ Use the setter injection technique, which gives more flexibility:
 - You can put initialization code inside the setter method.
 - The session bean is set up to be easily tested as a stand-alone component.

In addition, note the following use of the @Resource annotation:

- ▶ To obtain a reference to the EJB session context, as shown in Example 12-15

Example 12-15 Resource reference to session context

```

@Stateless
public class CustomerSessionBean implements CustomerServiceInterface
{
    ....
    @Resource javax.ejb.SessionContext ctx;
}

```

- ▶ To obtain the value of an environment variable, which is configured inside the deployment descriptor with env-entry, as shown in Example 12-16

Example 12-16 Resource reference to environment variable

```

@Stateless
public class CustomerSessionBean implements CustomerServiceInterface
{
    ....
    @Resource String myEnvironmentVariable;
}

```

For detailed information, see section 4.3.2 “Dependency Injection” in *JSR 318: Enterprise JavaBeans 3.1*.

Asynchronous invocations

All session bean invocations, regardless of which view, the Remote, Local, or the no-interface views, are synchronous by default. As a new feature of the EJB 3.1 specification, you can define a bean class or a method as asynchronous, while using the annotation `@Asynchronous`. This approach to define a method as asynchronous avoids the behavior that one request blocks for the duration of the invocation until the process is completed. This is a result of the synchronous approach. In case a request invokes an asynchronous method, the container returns control back to the client immediately and continues processing the invocation on another thread. Therefore, the asynchronous method has to return either `void` or `Future<T>`.

For detailed information, see section 3.4.8 “Asynchronous Invocations” in *JSR 318: Enterprise JavaBeans 3.1*.

EJB timer service

The EJB timer service is a container-managed service for scheduled callbacks. The definition of time-based events with the timer service is a new feature of EJB 3.1. Therefore, the method `getTimerService` exists, which returns the `javax.ejb.TimerService` interface. This method can be used by stateless and Singleton session beans. Stateful session beans cannot be timed objects. Time-based events can be calendar-based-scheduled, at a specific time, after a specific past duration, or for specific circular intervals.

To define timers to be created automatically by the container, use the `@Schedule` and `@Schedules` annotations. Example 12-17 shows how to define a timer method for every second of every minute of every hour of every day.

Example 12-17 Timer service definition

```
@Schedule(dayOfWeek="*",hour="*",minute="*",second="*")
public void calledEverySecond(){
    System.out.println("Called every second");
}
```

The definition of the attributes of the `@Schedule` annotation can be modified as well using the Attributes view in Rational Application Developer, as shown in Figure 12-5 on page 598.

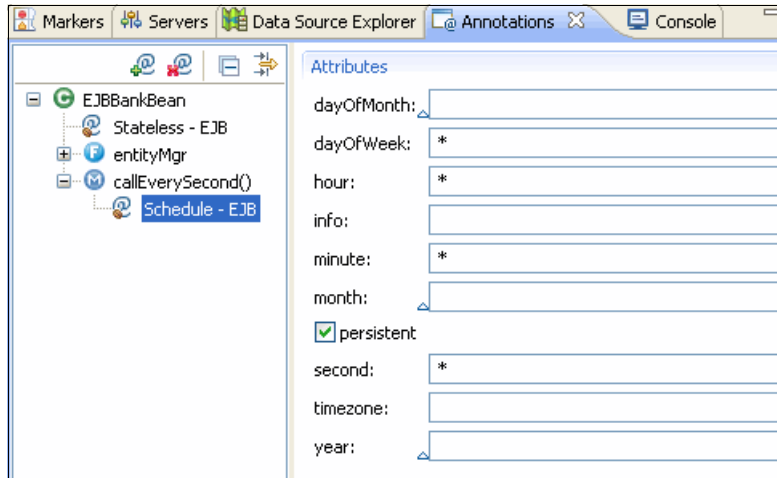


Figure 12-5 Annotation @Schedule attribute view

For detailed information about the EJB timer service, see Chapter 18, “Timer Service”, in *JSR 318: Enterprise JavaBeans 3.1*, or refer to this website:

http://publib.boulder.ibm.com/infocenter/wasinfo/beta/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tejb_timerserviceejb_enh.html

Web services

Chapter 14, “Developing web services applications” on page 681, explains how to expose EJB 3.1 beans as web services by using the `@WebService` annotation. In that same chapter, we show how to implement a web service from an EJB 3.1 session bean.

Portable JNDI name

As a new feature defined in the Java EE 6 specification, a standardized global JNDI namespace is defined. These portable JNDI names are defined with the following syntax:

```
java:global[/<app-name>]/<module-name>/<bean-name>[!<fully-qualified-interface-name>]
```

The parameters consist of the following content:

<app-name>

Name of the application. Because a session bean is packaged within an EAR file, this field is an optional value. It defaults to the name of the EAR file, just without the .ear file extension.

<module-name>	Name of the module in which the session bean is packaged. The value of the name defaults to the base name of the archive without the file extension. This archive can be a stand-alone JAR file or a WAR file.
<bean-name>	Name of the session bean.
<fully-qualified-interface-name>	Fully qualified name of each defined business interface. Because a session bean can have no interface, which means it has only a no-interface view, this field is an optional value.

For detailed information, see section 4.4 “Global JNDI Access” in *JSR 318: Enterprise JavaBeans 3.1*.

Embedded Container API

Defining an embedded container is a new feature in EJB 3.1. An embedded container provides the same managed environment as the Java EE runtime container. The services for injection, access to a component environment, and container-managed transactions (CMTs) are provided as well. This container is used to execute EJB components within a Java SE environment. Example 12-18 shows how to create an instance of an embeddable container, as a first step.

Example 12-18 Defining embedded container

```
EJBContainer ec = EJBContainer.createEJBContainer();
Context ctx = ec.getContext();
EJBBank bank = (EJBBank) ctx.lookup("java:global/EJBExample/EJBBank");
```

In the second step in Example 12-18, we get a JNDI context. In the third step, we use the lookup method to retrieve an EJB, in this case, the bean EJBBank.

For detailed information about the Embedded Container API, see Chapter 22.2.1, “EJBContainer”, in *JSR 318: Enterprise JavaBeans 3.1*.

Important: In the `\7835code\ejb\antScriptEJB.zip` directory, we provide an Ant script to create the jar file of your EJB project. Update the `build.properties` file with your settings before using the `build.xml` file. This script includes the configuration for the RAD8EJB project as an example.

Using deployment descriptors

In the previous sections, we have seen how to define an EJB, how to inject resources into it, and how to specify annotations. We can get the same result by

specifying a deployment descriptor (`ejb-jar.xml`) with the necessary information in the EJB module.

12.1.4 EJB 3.1 application packaging

Session and message-driven beans are packaged in Java standard JAR files. We map from the enterprise archive (EAR) project to the EJB project containing the beans. To do this, we use the Deployment Assembly properties sheet, which replaces the J2EE Module Dependencies properties sheet used in previous versions of Rational Application Developer. The integrated development environment (IDE) will automatically update the `application.xml` file, if one exists.

However, in EJB 3.x, you are not required to define the EJB and related resources in an `ejb-jar.xml` file, because they are usually defined through the use of annotations. The major use of the deployment descriptor files is to override or complete behavior that is specified by annotations.

EJB 3.1 offers the capability to package and deploy EJB components directly in a WAR file as a new feature for the packaging approach.

12.1.5 EJB 3.1 Lite

Because the full EJB 3.x API consists of a large set of features with the support for implementing business logic in a wide variety of enterprise applications, EJB 3.1 Lite was defined to provide a minimal subset of the full EJB 3.1 API. This new defined runtime environment offers a selection of EJB features, as shown in Table 12-1.

Table 12-1 Overview comparison of EJB 3.1 Lite and full EJB 3.1

Feature	EJB 3.1 Lite	Full EJB 3.1
Session beans (stateless, stateful, and Singleton)	Yes	Yes
MDB	No	Yes
Entity beans 1.x/2.x	No	Yes
No-interface view	Yes	Yes
Local interface	Yes	Yes
Remote interface	No	Yes
2.x interfaces	No	Yes*

Feature	EJB 3.1 Lite	Full EJB 3.1
Web services (JAX-WS, JAX-RS, and JAX-RPC)	No	Yes ^a
Timer service	No	Yes
Asynchronous calls	No	Yes
Interceptors	Yes	Yes
RMI/IIOP interoperability	No	Yes
Transaction support	Yes	Yes
Security	Yes	Yes
Embeddable API	Yes	Yes

a = Pruning candidates for future versions of the EJB specification

For detailed information, see Section 21.1, “EJB 3.1 Lite”, in *JSR 318: Enterprise JavaBeans 3.1*.

12.1.6 EJB 3.1 features in Rational Application Developer

The following features are supported in Rational Application Developer:

- ▶ Singleton bean
- ▶ No interface-view for session beans
- ▶ Asynchronous invocations
- ▶ EJB timer service
- ▶ Portable JNDI name
- ▶ Embedded Container API
- ▶ War deployment, as mentioned in EJB 3.1 application packaging
- ▶ EJB 3.1 Lite

12.2 Developing an EJB module

The EJB module consists of a web module with a simple servlet, and an EJB module with an EJB 3.1 session bean that uses the JPA entities of the RAD8JPA project to access the database. This section describes the steps for developing the sample EJB module.

To develop EJB applications, you have to enable the EJB development capability in Rational Application Developer (the capability might already be enabled):

1. Select **Window** → **Preferences**.
2. Select **General** → **Capabilities** → **Enterprise Java Developer** and click **OK**.

An EJB module, with underlying JPA entities, typically contains components that work together to perform business logic. This logic can be self-contained or access external data and functions as needed. It needs to consist of a facade (session bean) and the business entities (JPA entities). The facade is usually implemented using one or more session beans and MDBs.

In this chapter, we develop a session EJB as a facade for the JPA entities (Customer, Account, and Transaction), as shown in Figure 12-6 on page 603. The RAD8JPA project has to be available in your workspace. You can import the project from the `\7835\codesolution\jpa` directory.

Furthermore, we assume that an instance of the WebSphere Application Server V8.0 Beta is configured and available in your workspace.

Sample code: The sample code described in this chapter can be completed by following the documented procedures. Alternatively, you can import the sample EJB project and corresponding JPA project provided in the `\7835\codesolution\ejb\RAD8EJB.zip` directory. See Appendix B, “Additional material” on page 1423, for instructions to download the sample code. For more information about installing the software, see Appendix A, “Installing the products” on page 1783.

12.2.1 Sample application overview

In this chapter, we reuse the design of the application, which is described in Chapter 7, “Developing Java applications” on page 229, with minor changes. However, the content of this chapter does not depend on the Java chapter. You can complete the sample in this chapter without knowledge of the sample developed in the Java chapter.

Figure 12-6 on page 603 shows the sample application model layer design.

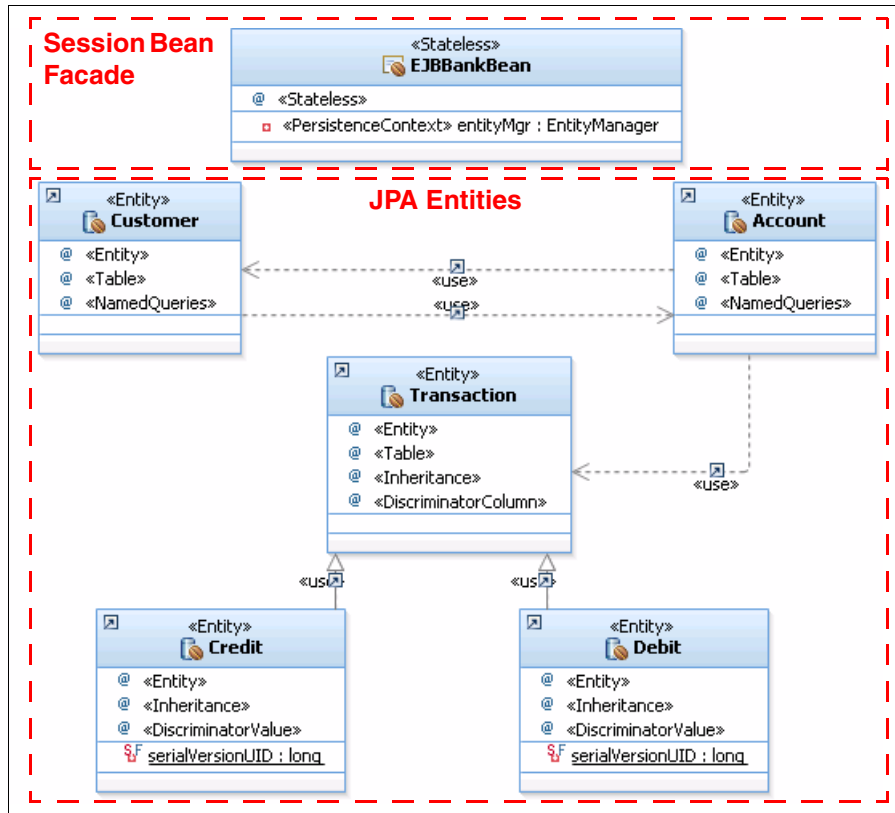


Figure 12-6 EJB module class diagram for the sample application

The EJBBankBean session bean acts as a facade for the EJB model. The business entities (Customer, Account, Transaction, Credit, and Debit) are implemented as JPA entity beans, as opposed to regular JavaBeans. By doing so, we automatically gain persistence, security, distribution, and transaction management services. This also implies that the control and view layers are not able to reference these entities directly, because they can be placed in a separate Java virtual machine (JVM). Only the session bean EJBBankBean can access the business entities.

Figure 12-7 on page 604 shows the application component model and the flow of events.

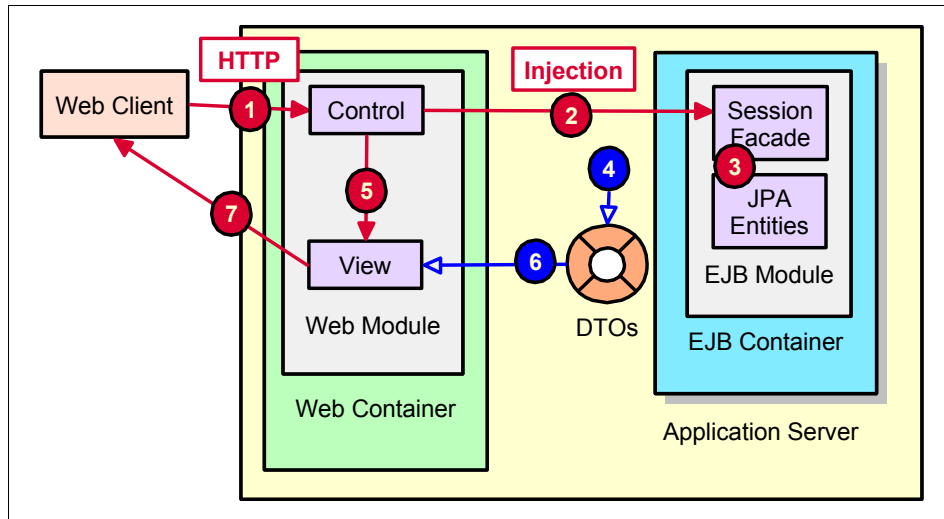


Figure 12-7 Application component model and workflow

Figure 12-7 shows the following flow of events:

1. The HTTP request is issued by the web client to the server. This request is answered by a servlet in the control layer, also known as the *front controller*, which extracts the parameters from the request. The servlet sends the request to the appropriate control JavaBean. This bean verifies whether the request is valid in the current user and application states.
2. If the request is valid, the control layer sends the request through the @EJB injected interface to the session EJB facade. This involves using JNDI to locate the session bean's interface and creating a new instance of the bean.
3. The session EJB executes the appropriate business logic related to the request. This includes accessing JPA entities in the model layer.
4. The facade returns data transfer objects (DTOs) to the calling controller servlet with the response data. The DTO returned can be a JPA entity, a collection of JPA entities, or any Java object. In general, it is not necessary to create extra DTOs for entity data.
5. The front controller servlet sets the response DTO as a request attribute and forwards the request to the appropriate JSP in the view layer, which is responsible for rendering the response back to the client.
6. The view JSP accesses the response DTO to build the user response.
7. The result view, possibly in HTML, is returned to the client.

12.2.2 Creating an EJB project

To develop the session EJB, we create an EJB project. It is also typical to create an EAR project that is the container for deploying the EJB project.

To create the EJB project, perform the following steps:

1. In the Java EE perspective, within the Enterprise Explorer view, right-click and select **New** → **Project**.
2. In the New Project wizard, select **EJB** → **EJB Project** and click **Next**.
3. In the New EJB Project window, shown in Figure 12-8 on page 606, define the project details:
 - a. In the Name field, type RAD8EJB.
 - b. For Target Runtime, select **WebSphere Application Server v8.0 Beta**.
 - c. For EJB module version, select **3.1**.
 - d. For Configuration, select **Minimal Configuration**. Optional: Click **Modify** to see the project facets (EJB Module 3.1, Java 6.0, and WebSphere EJB (Extended) 8.0).
 - e. Select **Add project to an EAR** (default), and in the EAR Project Name field, type RAD8EJBEAR. By default, the wizard creates a new EAR project, but you can also select an existing project from the list of options for the EAR Project Name field. If you want to create a new project and configure its location, click **New**. For our example, we use the given default value.
 - f. Click **Next**.

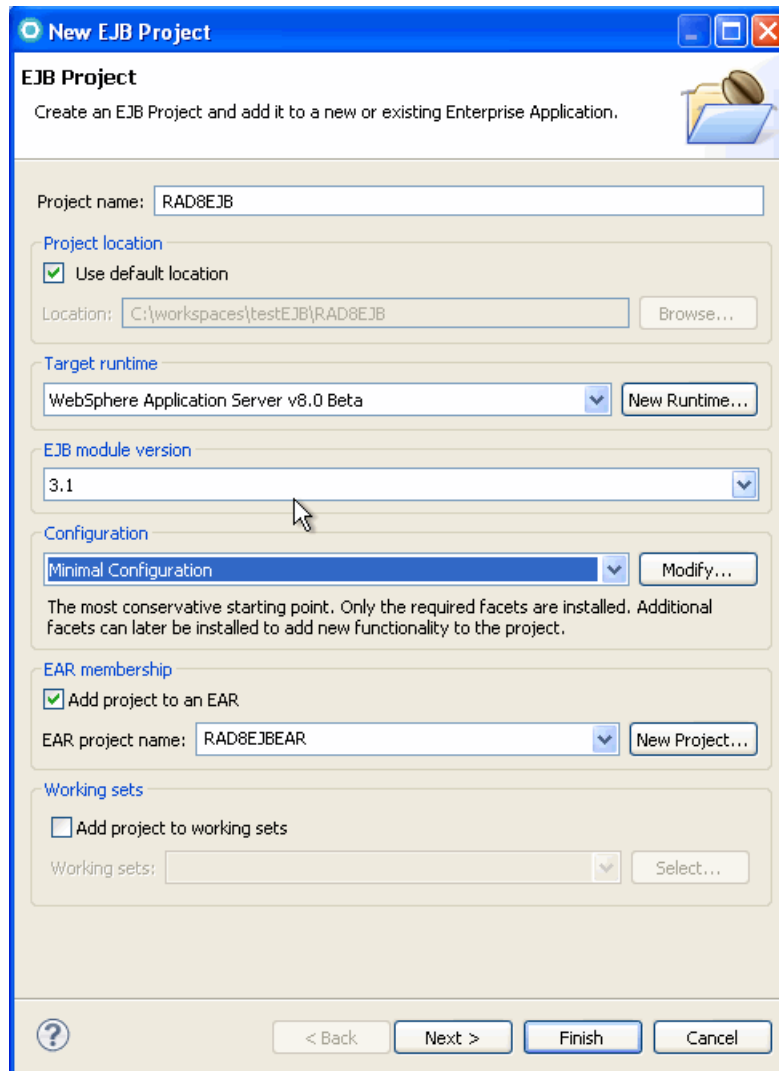


Figure 12-8 Creating an EJB project: EJB Project window

4. In the New EJB Project wizard, in the Java window, accept the default value `ejbModule` for the Source folder.
5. In the New EJB Project wizard, in the EJB Module window, perform the following steps, as shown in Figure 12-9 on page 607:
 - a. Clear **Create an EJB Client JAR module to hold client interfaces and classes** (default).
 - b. Select **Generate ejb-jar.xml deployment descriptor** and click **Finish**.

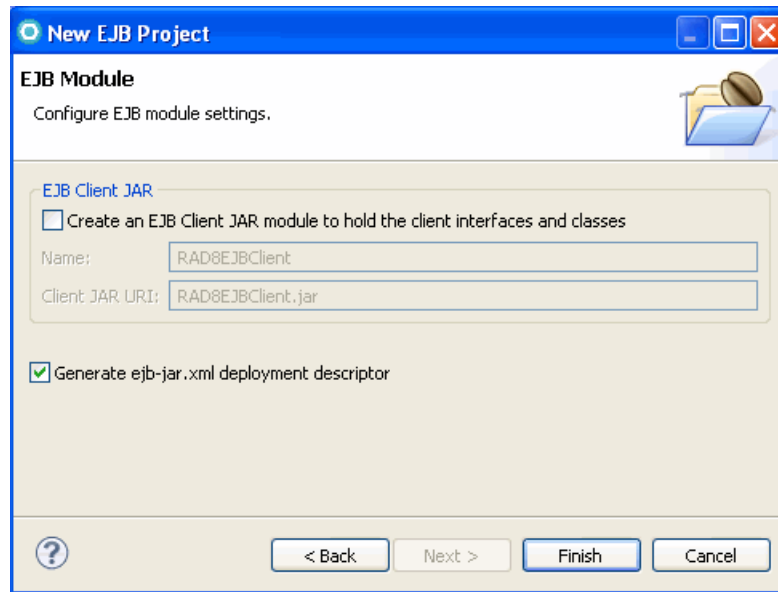


Figure 12-9 Creating an EJB project - EJB Module window

EJB client JAR file: The EJB client JAR file holds the interfaces of the enterprise beans and other classes on which these interfaces depend. For example, it holds their superclasses and implemented interfaces, the classes and interfaces used as method parameters, results, and exceptions. The EJB client JAR can be deployed together with a client application that accesses the EJB. This results in a smaller client application compared to deploying the EJB project with the client application.

6. If the current perspective is not the Java EE perspective when you create the project, when Rational Application Developer prompts you to switch to the Java EE perspective, click **Yes**.
7. The Technology Quickstarts view opens. Close the view.

The Enterprise Explorer view contains the RAD8EJB project and the RAD8EJBEAR enterprise application. Rational Application Developer indicates that at least one EJB bean has to be defined within the RAD8EJB project. We create this session bean in 12.2.5, “Implementing the session facade” on page 612, when we have enabled the JPA project.

12.2.3 Making the JPA entities available to the EJB project

We assume that you imported the JPA project RAD8JPA into your workspace. To make the JPA entities available to the EJBs, add the RAD8JPA project to the RAD8EJBEBEAR enterprise application and create a dependency, while performing the following steps:

1. Right-click the **RAD8EJBEBEAR** project and select **Properties**.
2. In the Properties window, select **Deployment Assembly**, and for EAR Module Assembly, click **Add**.
3. In the New Assembly directive wizard, in the Select Directive Type window, select **Project**. Click **Next**.
4. In the New Assembly directive wizard, in the Project window, select **RAD8JPA**. Click **Finish**.

The Properties window now looks like Figure 12-10 on page 609.

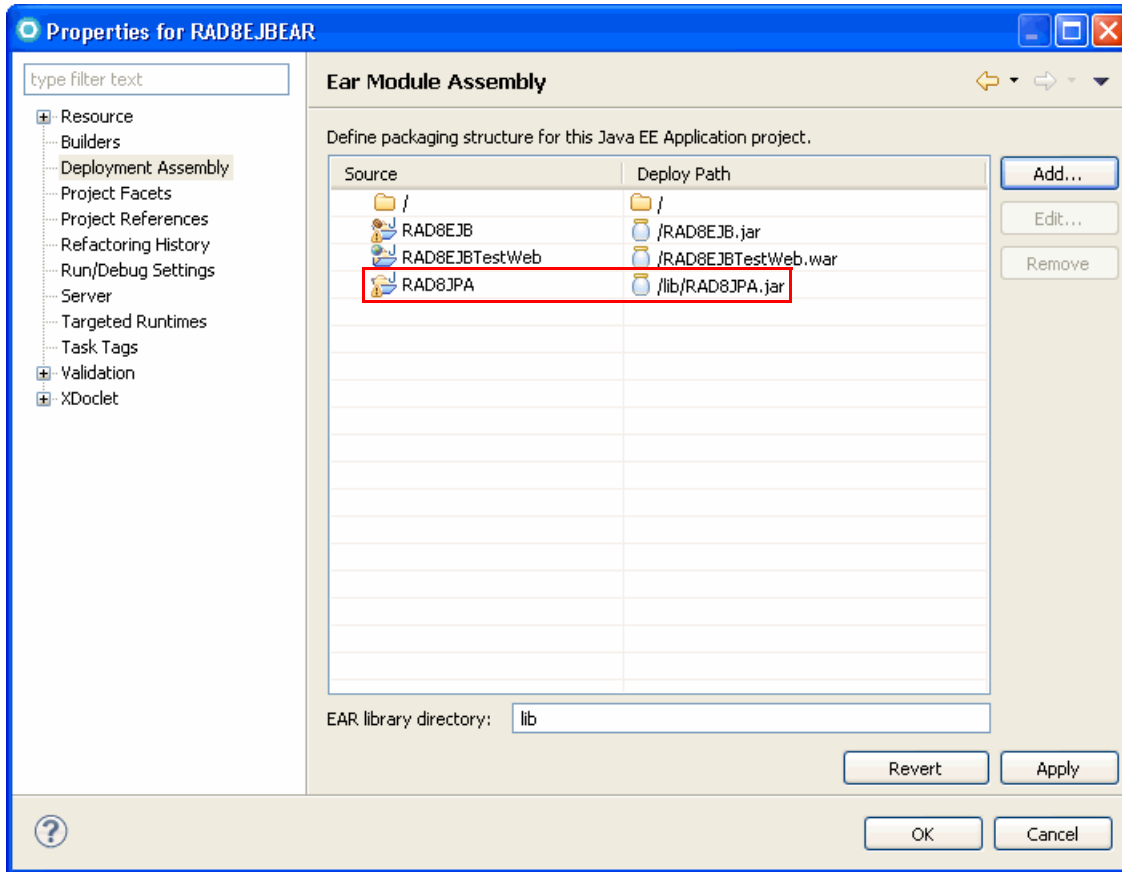


Figure 12-10 Selecting the RAD8JPA project

12.2.4 Setting up the ITSOBANK database

The JPA entities are based on the ITSOBANK database. Therefore, we must define a database connection within Rational Application Developer that the mapping tools use to extract schema information from the database.

See “Setting up the ITSOBANK database” on page 1880 for instructions to create the ITSOBANK database. We can either use the DB2 or Derby database. For simplicity, we use the built-in Derby database in this chapter.

Configuring the data source for the ITSOBANK

You can choose from multiple methods to configure the data source, including using the WebSphere administrative console or using the WebSphere enhanced

EAR, which stores the configuration in the deployment descriptor and is deployed with the application.

For a definition of the data source in the WebSphere administrative console, see “Configuring the data source in WebSphere Application Server” on page 1882.

In this section, we explain how to configure the data source using the WebSphere enhanced EAR capabilities. The enhanced EAR is configured in the Deployment tab of the EAR Deployment Descriptor editor. If you select to import the complete sample code, you only have to verify that the value of the `databaseName` property in the deployment descriptor matches the location of the database.

Configuring the data source using the enhanced EAR

Before you perform the following steps, you have to start the server. To configure a new data source using the enhanced EAR capability in the deployment descriptor, follow these steps:

1. Right-click the **RAD8EJB** project. Select **Java EE** → **Open WebSphere Application Server Deployment**.
2. In the WebSphere Deployment editor, select **Derby JDBC Provider (XA)** from the JDBC provider list. This JDBC provider is configured by default.
3. Click **Add** next to data source.
4. Under the JDBC provider, select **Derby JDBC Provider (XA)** and **Version 5.0 data source**. Click **Next**.
5. In the Create a Data Source window, which is shown in Figure 12-11 on page 611, define the following details:
 - a. For Name, type `ITSOBANKejb`.
 - b. For JNDI name, type `jdbc/itsobank`.
 - c. For Description, type `Data Source for ITSOBANK EJBs`.
 - d. Clear **Use this data source in container managed persistence (CMP)**.

Create Data Source

Select the type of data source to create.

Name: ITSOBANKejb

JNDI name: jdbc/itsobank

Description: Datasource for ITSOBANK EJBs

Category:

Statement cache size: 10

Data source helper class name: com.ibm.websphere.rsadapter.DerbyDataStoreHelper

Connection timeout: 180

Maximum connections: 10

Minimum connections: 1

Reap time: 180

Unused timeout: 1800

Aged timeout: 0

Purge policy: EntirePool

Component-managed authentication alias:

Container-managed authentication alias:

Use this data source in container managed persistence (CMP)

* Required field.

< Back Next > Finish Cancel

Figure 12-11 Data source definition: Name

- e. Click **Next**.
6. In the Create Resource Properties window, select **databaseName** and enter the value `C:\7835code\database\derby\ITSOBANK`, which is the path where your installed database is located. Clear the description, as shown in Figure 12-12 on page 612.

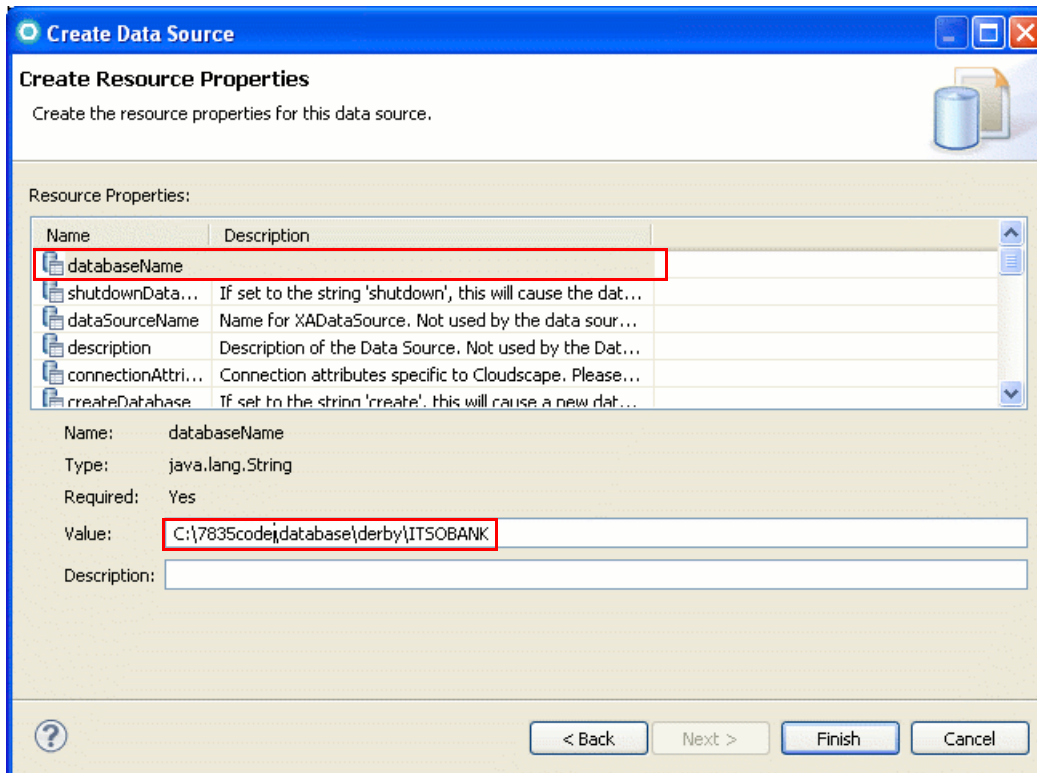


Figure 12-12 Data source definition: Database definition

7. Click **Finish**.
8. Save and close the deployment descriptor.

12.2.5 Implementing the session facade

The front-end application communicates with the JPA entity model through a session facade. This design pattern makes the entities invisible to the EJB client.

In this section, we build the session facade with the session bean `EJBBankBean`. Therefore, we describe all necessary steps to implement the session facade and add the facade methods that are used by clients to perform banking operations, including:

- ▶ Preparing an exception
- ▶ Creating the `EJBBankBean` session bean
- ▶ Defining the business interface
- ▶ Creating an Entity Manager

- ▶ Generating skeleton methods
- ▶ Completing the methods in EJBBankBean
- ▶ Deploying the application to the server

Preparing an exception

The business logic of the session bean throws an exception when errors occur. Create an application exception named `ITSOBankException`, when performing the following steps:

1. Right-click the **RAD8EJB** project and select **New** → **Class**.
2. In the New Java Class window, define the following details:
 - a. For Package, type `itso.bank.exception`.
 - b. For Name, type `ITSOBankException`.
 - c. Set Superclass to **java.lang.Exception**.
3. Click **Finish**.
4. Complete the code in the editor, as shown in Example 12-19.

Example 12-19 Class definition ITSOBankException

```
public class ITSOBankException extends Exception {
    private static final long serialVersionUID = 1L;

    public ITSOBankException(String message) {
        super(message);
    }
}
```

5. Save and close the class.

Creating the EJBBankBean session bean

To create the session bean `EJBBankBean`, follow these steps:

1. Right-click the **RAD8EJB** project and select **New** → **Session Bean**.
2. In the Create EJB 3.x Session Bean window, as shown in Figure 12-13 on page 614, define the following details:
 - a. For Java package, type `itso.bank.session`.
 - b. For Class name, type `EJBBankBean`.
 - c. For State type, select **Stateless**.
 - d. For Create business interface, select **Local** and set the name to **itso.bank.service.EJBBankService**.
 - e. Click **Next**.

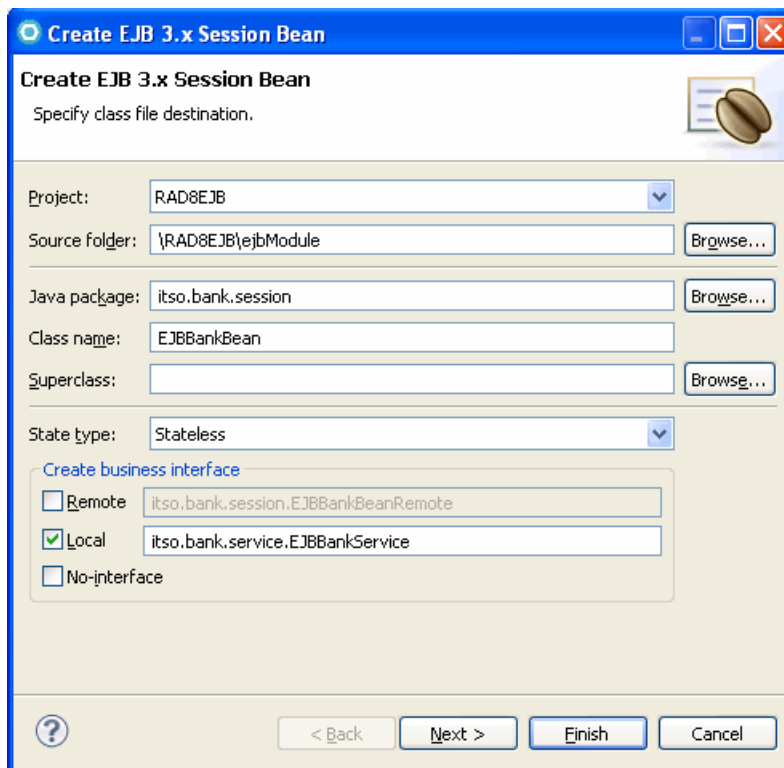


Figure 12-13 Creating a session bean (part 1 of 2)

3. In the next window, which is shown in Figure 12-14 on page 615, accept the default value **Container** for Transaction type and click **Next**.

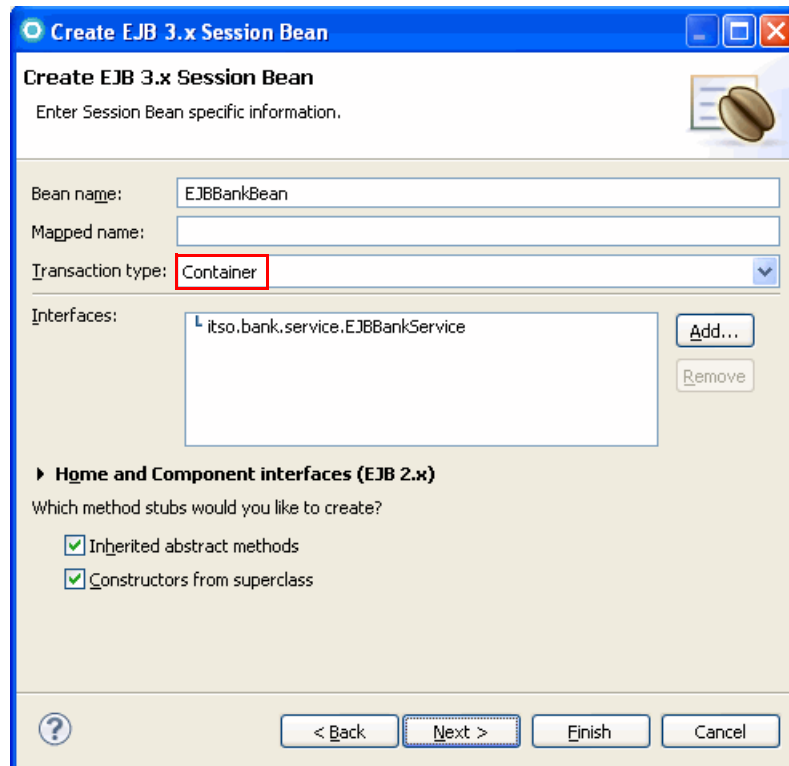


Figure 12-14 Creating a session bean (part 2 of 2)

4. In the Select Class Diagram for Visualization window, select **Add bean to Class Diagram** and accept the default name of **classdiagram.dnx**.
5. Click **Finish**.
6. When prompted for the enablement of EJB 3.1 Modeling, click **OK**.
7. Save and close the class diagram.

The EJBBankBean is open in the editor. Notice the `@Stateless` annotation.

Before you can write the session bean code, complete the business interface, `EJBBankService`.

Defining the business interface

EJB 3.1 also provides a business interface mechanism, which is the interface that clients use to access the session bean. The session bean can implement multiple interfaces, for example, a local interface and a remote interface. For now, we keep it simple with one local interface, `EJBBankService`.

The session bean wizard has created the `EJBBankService` interface. To complete the code, follow these steps:

1. Open the **EJBBankService** interface. Notice the `@Local` annotation.
2. In the Java editor, add the methods to the interface, as shown in Example 12-20. The code is available in the `\7835code\ejb\source\EJBBankService.txt` file.

Example 12-20 Business interface of the session bean

@Local

```
public interface EJBBankService {

    public Customer getCustomer(String ssn) throws ITSOBankException;
    public Customer[] getCustomersAll();
    public Customer[] getCustomers(String partialName) throws ITSOBankException;
    public void updateCustomer(String ssn, String title, String firstName, String
lastName) throws ITSOBankException;
    public Account[] getAccounts(String ssn) throws ITSOBankException;
    public Account getAccount(String id) throws ITSOBankException;
    public Transaction[] getTransactions(String accountID) throws ITSOBankException;
    public void deposit(String id, BigDecimal amount) throws ITSOBankException;
    public void withdraw(String id, BigDecimal amount) throws ITSOBankException;
    public void transfer(String idDebit, String idCredit, BigDecimal amount) throws
ITSOBankException;
    public void closeAccount(String ssn, String id) throws ITSOBankException;
    public String openAccount(String ssn) throws ITSOBankException;
    public void addCustomer(Customer customer) throws ITSOBankException;
    public void deleteCustomer(String ssn) throws ITSOBankException;
}
```

3. To organize the imports, press `Ctrl+Shift+O`. When prompted, select **`java.math.BigDecimal`** and **`itso.bank.entities.Transaction`**. Save and close the interface.

Creating an Entity Manager

The session bean works with the JPA entities to access the `ITSOBANK` database. We require an Entity Manager that is bound to the persistence context. To create an Entity Manager, follow these steps:

1. Add these definitions to the `EJBBankBean` class:

```
@PersistenceContext (unitName="RAD8JPA",
                    type=PersistenceContextType.TRANSACTION)
private EntityManager entityMgr;
```

The `@PersistenceContext` annotation defines the persistence context unit with transactional behavior. The unit name matches the name in the `persistence.xml` file in the RAD8JPA project:

```
<persistence-unit name="RAD8JPA">
```

The `EntityManager` instance is used to execute JPA methods to retrieve, insert, update, delete, and query instances.

2. Organize the imports by selecting the **javax.persistence** package.

Generating skeleton methods

We can generate method skeletons for the methods of the business interface that must be implemented:

1. Open the **EJBBankBean** (if you closed it).
2. Select **Source** → **Override/Implement Methods**.
3. In the Override/Implement Methods window, select all the methods of the `EJBBankService` interface. For Insertion point, select **After 'EJBBankBean()'**. Click **OK**. The method skeletons are generated.
4. Delete the default constructor.

Completing the methods in EJBBankBean

Tip: You can copy the Java code for this section from the `\7835code\ejb\source\EJBBankBean.txt` file.

We complete the methods of the session bean in a logical sequence, not in the alphabetical sequence of the generated skeletons.

getCustomer method

The `getCustomer` method retrieves one customer by Social Security number (SSN). We use `entityMgr.find` to retrieve one instance. Alternatively, we might use the `getCustomerBySSN` query (code in comments). If no instance is found, null is returned, as shown in Example 12-21.

Example 12-21 Session bean getCustomer method

```
public Customer getCustomer(String ssn) throws ITS0BankException {
    System.out.println("getCustomer: " + ssn);
    //Query query = null;
    try {
        //query = entityMgr.createNamedQuery("getCustomerBySSN");
        //query.setParameter("ssn", ssn);
        //return (Customer)query.getSingleResult();
    }
}
```

```

        return entityMgr.find(Customer.class, ssn);
    } catch (Exception e) {
        System.out.println("Exception: " + e.getMessage());
        throw new ITSOBankException(ssn);
    }
}

```

getCustomers method

The `getCustomers` method uses a query to retrieve a collection of customers, as shown in Example 12-22. The query is created and executed. The result list is converted into an array and returned. Remember the defined query from the Customer entity:

```

@NamedQuery(name="getCustomersByPartialName",
            query="select c from Customer c where c.lastName like :name")

```

This query looks similar to SQL but works on entity objects. In our case, the entity name and the table name are the same, but they do not have to be identical.

Example 12-22 Session bean getCustomers method

```

public Customer[] getCustomers(String partialName) throws
ITSOBankException {
    System.out.println("getCustomer: " + partialName);
    Query query = null;
    try {
        query = entityMgr.createNamedQuery("getCustomersByPartialName");
        query.setParameter("name", partialName);
        List<Customer> beanlist = query.getResultList();
        Customer[] array = new Customer[beanlist.size()];
        return beanlist.toArray(array);
    } catch (Exception e) {
        throw new ITSOBankException(partialName);
    }
}

```

The updateCustomer method

The `updateCustomer` method is simple, as shown in Example 12-23. No call to the Entity Manager is necessary. The table is updated automatically when the method (transaction) ends.

Example 12-23 Session bean updateCustomer method

```

public void updateCustomer(String ssn, String title, String firstName,
                           String lastName) throws ITSOBankException {

```

```

        System.out.println("updateCustomer: " + ssn);
        Customer customer = getCustomer(ssn);
        customer.setTitle(title);
        customer.setLastName(lastName);
        customer.setFirstName(firstName);
        System.out.println("updateCustomer: " + customer.getTitle() + " "
            + customer.getFirstName() + " " +
customer.getLastName());
    }

```

The getAccount method

The `getAccount` method retrieves one account by key. It is similar to the `getCustomer` method.

The getAccounts method

The `getAccounts` method uses a query to retrieve all the accounts of a customer, as shown in Example 12-24. The `Account` entity has the following query:

```

select a from Account a, in(a.customers) c where c.ssn =:ssn
        order by a.id

```

This query looks for accounts that belong to a customer with a given SSN. You can also use this alternate query in the `Customer` class:

```

select a from Customer c, in(c.accounts) a where c.ssn =:ssn
        order by a.id

```

Example 12-24 Session bean getAccounts method

```

public Account[] getAccounts(String ssn) throws ITSBankException {
    System.out.println("getAccounts: " + ssn);
    Query query = null;
    try {
        query = entityMgr.createNamedQuery("getAccountsBySSN");
        query.setParameter("ssn", ssn);
        List<Account>accountList = query.getResultList();
        Account[] array = new Account[accountList.size()];
        return accountList.toArray(array);
    } catch (Exception e) {
        System.out.println("Exception: " + e.getMessage());
        throw new ITSBankException(ssn);
    }
}

```

The getTransactions method

The `getTransactions` method retrieves the transactions of an account, as shown in Example 12-25. It is similar to the `getAccounts` method.

Example 12-25 Session bean getTransactions method

```
public Transaction[] getTransactions(String accountID) throws
ITSOBankException {
    System.out.println("getTransactions: " + accountID);
    Query query = null;
    try {
        query = entityMgr.createNamedQuery("getTransactionsByID");
        query.setParameter("aid", accountID);
        List<Transaction> transactionsList = query.getResultList();
        Transaction[] array = new Transaction[transactionsList.size()];
        return transactionsList.toArray(array);
    } catch (Exception e) {
        System.out.println("Exception: " + e.getMessage());
        throw new ITSOBankException(accountID);
    }
}
```

The deposit and withdraw methods

The `deposit` method adds money to an account by retrieving the account and calling its `processTransaction` method with the `Transaction.CREDIT` code. The new transaction instance is persisted, as shown in Example 12-26. The `withdraw` method is similar and uses the `Transaction.DEBIT` code.

Example 12-26 Session bean deposit method

```
public void deposit(String id, BigDecimal amount) throws
ITSOBankException {
    System.out.println("deposit: " + id + " amount " + amount);
    Account account = getAccount(id);
    try {
        Transaction tx = account.processTransaction(amount,
Transaction.CREDIT);
        entityMgr.persist(tx);
    } catch (Exception e) {
        throw new ITSOBankException(e.getMessage());
    }
};
}
```

The transfer method

The transfer method calls `withdraw` and `deposit` on two accounts to move funds from one account to the other account, as shown in Example 12-27.

Example 12-27 Session bean transfer method

```
public void transfer(String idDebit, String idCredit, BigDecimal
amount)
    throws ITS0BankException {
    System.out.println("transfer: " + idCredit + " " + idDebit + "
amount "
    + amount);
    withdraw(idDebit, amount);
    deposit(idCredit, amount);
}
```

The openAccount method

The `openAccount` method creates a new account instance with a randomly constructed account number. The instance is persisted, and the customer is added to the customers, as shown in Example 12-28.

Example 12-28 Session bean openAccount method

```
public String openAccount(String ssn) throws ITS0BankException {
    System.out.println("openAccount: " + ssn);
    Customer customer = getCustomer(ssn);
    int acctNumber = (new java.util.Random()).nextInt(899999) + 100000;
    String id = "00" + ssn.substring(0, 1) + "-" + acctNumber;
    Account account = new Account();
    account.setId(id);
    entityMgr.persist(account);
    List<Customer> custSet = Arrays.asList(customer);
    account.setCustomers(custSet);
    System.out.println("openAccount: " + id);
    return id;
}
```

Adding the “m:m” relationship: The m:m relationship must be added from the *owning* side of the relationship, in our case, from the `Account`. The code to add the relationship from the `Customer` side runs without error, but the relationship is not added.

The closeAccount method

The `closeAccount` method retrieves an account and all its transactions, then deletes all instances using the Entity Manager `remove` method, as shown in Example 12-29.

Example 12-29 Session bean closeAccount method

```
public void closeAccount(String ssn, String id) throws
ITSOBankException {
    System.out.println("closeAccount: " + id + " of customer " + ssn);
    Customer customer = getCustomer(ssn);
    Account account = getAccount(id);
    Transaction[] trans = getTransactions(id);
    for (Transaction tx : trans) {
        entityMgr.remove(tx);
    }
    entityMgr.remove(account);
    System.out.println("closed account with " + trans.length
        + " transactions");
}
```

The addCustomer method

The `addCustomer` method accepts a fully constructed `Customer` instance and makes it persistent, as shown in Example 12-30.

Example 12-30 Session bean addCustomer method

```
public void addCustomer(Customer customer) throws ITSOBankException {
    System.out.println("addCustomer: " + customer.getSsn());
    entityMgr.persist(customer);
}
```

The deleteCustomer method

The `deleteCustomer` method retrieves a customer and all its accounts and then closes the accounts and deletes the customer, as shown in Example 12-31.

Example 12-31 Session bean deleteCustomer method

```
public void deleteCustomer(String ssn) throws ITSOBankException {
    System.out.println("deleteCustomer: " + ssn);
    Customer customer = getCustomer(ssn);
    Account[] accounts = getAccounts(ssn);
    for (Account acct : accounts) {
        closeAccount(ssn, acct.getId());
    }
}
```

```
entityMgr.remove(customer);  
}
```

Organize the imports (select `javax.persistence.Query`, and `java.util.List`).

The `EJBBankBean` session bean is now complete. In the following sections, we test the EJB using a servlet and then proceed to integrate the EJB with a web application.

12.3 Testing the session EJB and the JPA entities

To test the session EJB, we can use the Universal Test Client, as described in 12.3.1, “Testing with the Universal Test Client” on page 624. As a second approach, we develop a simple servlet that executes all the functions, as described in 12.3.2, “Creating a web application to test the session bean” on page 626.

Deploying the application to the server

To deploy the test application, perform these steps:

1. Start WebSphere Application Server V8.0 Beta in the Servers view.

JNDI name for data source: Make sure that the data source for the ITSOBANK database is configured with a JNDI name of `jdbc/itsobank` either in the WebSphere Deployment editor (“Configuring the data source for the ITSOBANK” on page 609) or in the administrative console of the server (“Configuring the data source in WebSphere Application Server” on page 1882).

2. Select the server and click **Add and Remove Projects**. Add the **RAD8EJB** enterprise application.
3. Click **Finish** and wait for the publishing to finish.

Notice the EJB binding messages in the console:

```
[...] 00000010 ResourceMgrIm I   WSVR0049I: Binding ITSOBANKejb as  
jdbc/itsobank  
[...] 00000015 EJBContainerI I   CNTR0167I: The server is binding  
the EJBBankService interface of the EJBBankBean enterprise bean in  
the RAD8EJB.jar module of the RAD8EJB application. The binding  
location is: ejblocal:RAD8EJB/RAD8EJB.jar/EJBBankBean#itso.bank  
.service.EJBBankService
```

```
[...] 00000015 EJBContainerI I CNTR0167I: The server is binding
the EJBBankService interface of the EJBBankBean enterprise bean in
the RAD8EJB.jar module of the RAD8EJBEAR application. The binding
location is: ejblocal:itso.bank.service.EJBBankService
```

12.3.1 Testing with the Universal Test Client

Before we integrate the EJB application with the web application, we test the session bean with the access to the JPA entities. We use the enterprise application Universal Test Client (UTC), which is included in Rational Application Developer.

In this section, we describe several operations that you can perform with the Universal Test Client. We use the test client to retrieve a customer and its accounts.

To test the session bean, follow these steps:

1. In the Servers view, right-click the server and select **Universal Test Client** → **Run**.
2. Accept the certificate and log in as admin/admin (the user ID that you set up when installing Rational Application Developer).
3. The Universal Test Client opens, as shown in Figure 12-15.

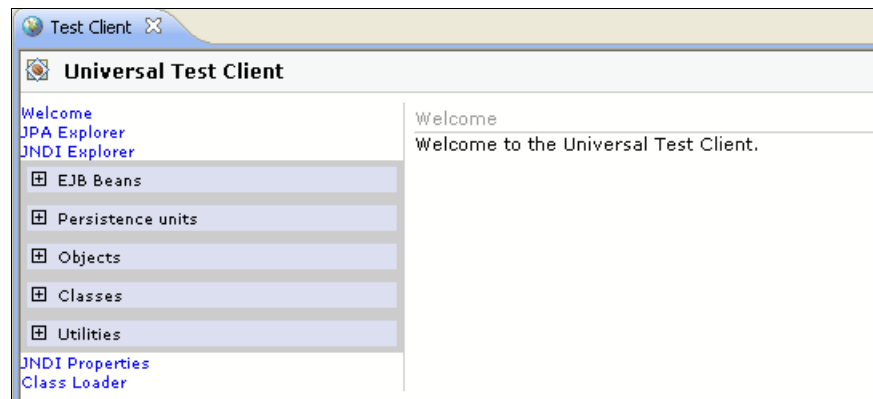


Figure 12-15 Universal Test Client welcome

4. In the Universal Test Client window, which is shown in Figure 12-16 on page 625, select **JNDI Explorer** on the left side. On the right side, expand **[Local EJB Beans]**.
5. Select **itso.bank.service.EJBBankService**. The EJBBankService is displayed under EJB Beans, as shown in Figure 12-16 on page 625.

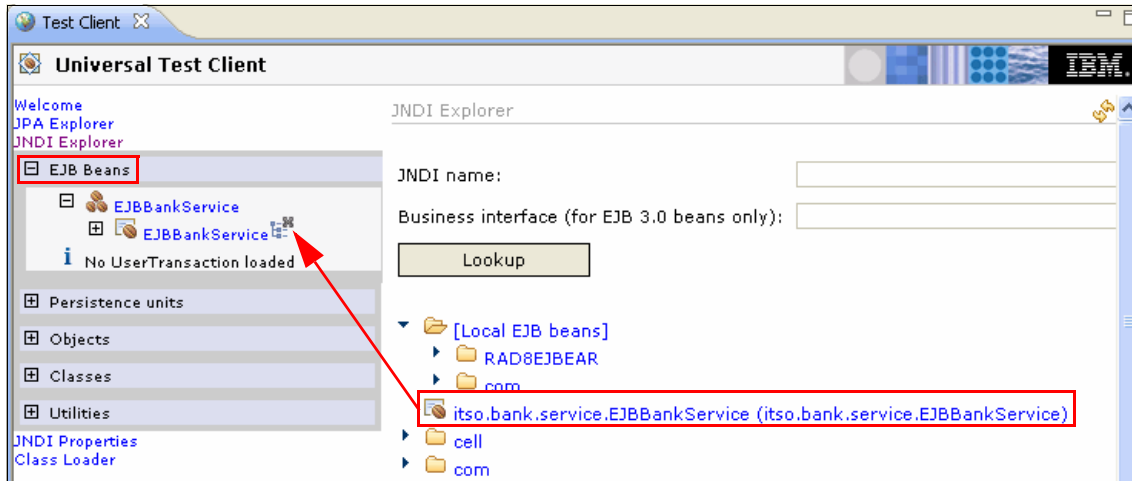


Figure 12-16 UTC: JNDI Explorer

6. Expand **EJBBankService** (on the left) and select the **getCustomer** method. The method with its parameter opens on the right, as shown in Figure 12-17 on page 626.
7. Type 333-33-3333 for the value on the right and click **Invoke**.

A Customer instance is displayed as result, as shown in Figure 12-17 as well.

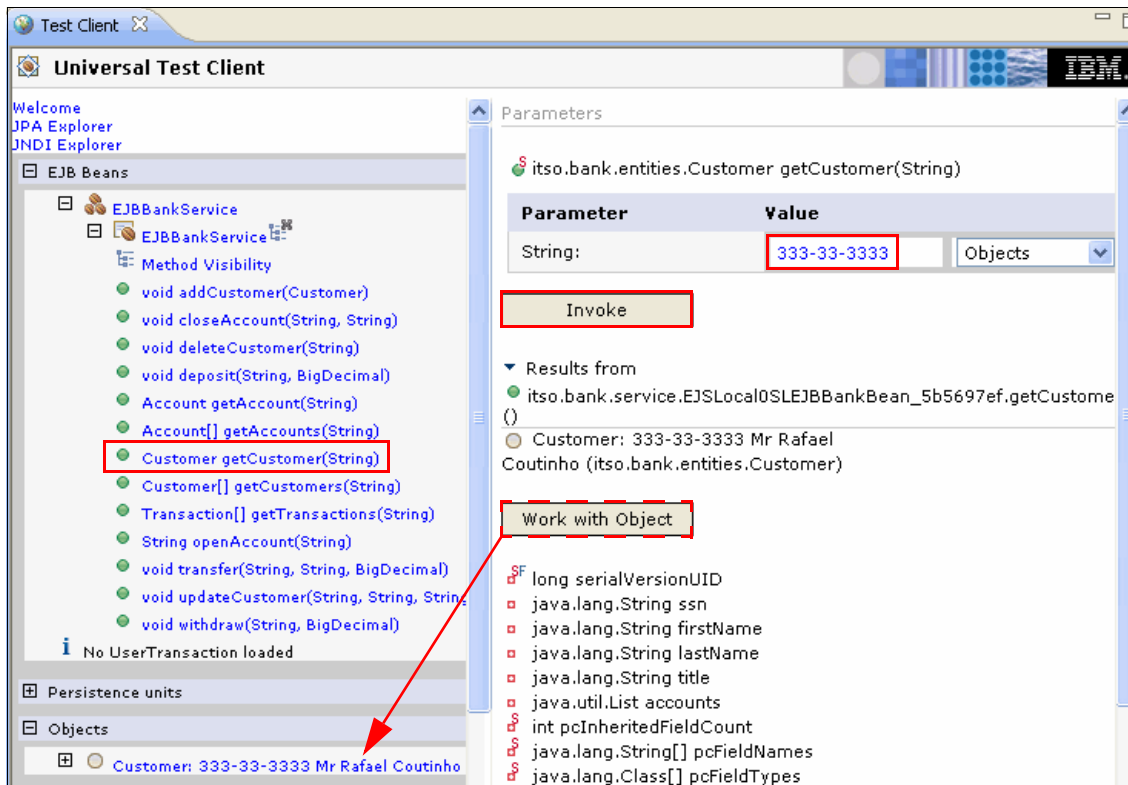


Figure 12-17 UTC: Retrieve a customer

8. Click **Work with Object**. The customer instance is displayed under Objects. You can expand the object and invoke its methods (for example, `getLastName`) to see the customer name.

Use the Universal Test Client to make sure that all of the EJB methods work. When you are done, close the Universal Test Client pane.

12.3.2 Creating a web application to test the session bean

To test the EJB 3.1 session bean and entity model, create a small web application with one servlet. Therefore, perform the following steps:

1. Within the Enterprise Explorer view, right-click and select **New** → **Project**.
2. In the New Project wizard, select **Web** → **Dynamic Web Project** and click **Next**.

3. In the New Dynamic Web Project wizard, define the project details, as shown in Figure 12-18 on page 628:
 - a. For Name, type RAD8EJBTestWeb.
 - b. For Dynamic Web Module version, select **3.0**.
 - c. For Configuration, select **Default Configuration for WebSphere Application Server v8.0 Beta**.
 - d. Select **Add the project to an EAR**. The value **RAD8EJBEAR** is set as the default (the name of the previously defined EAR project for the RAD8EJB project).
 - e. Click **Finish** and close the help window that opens.

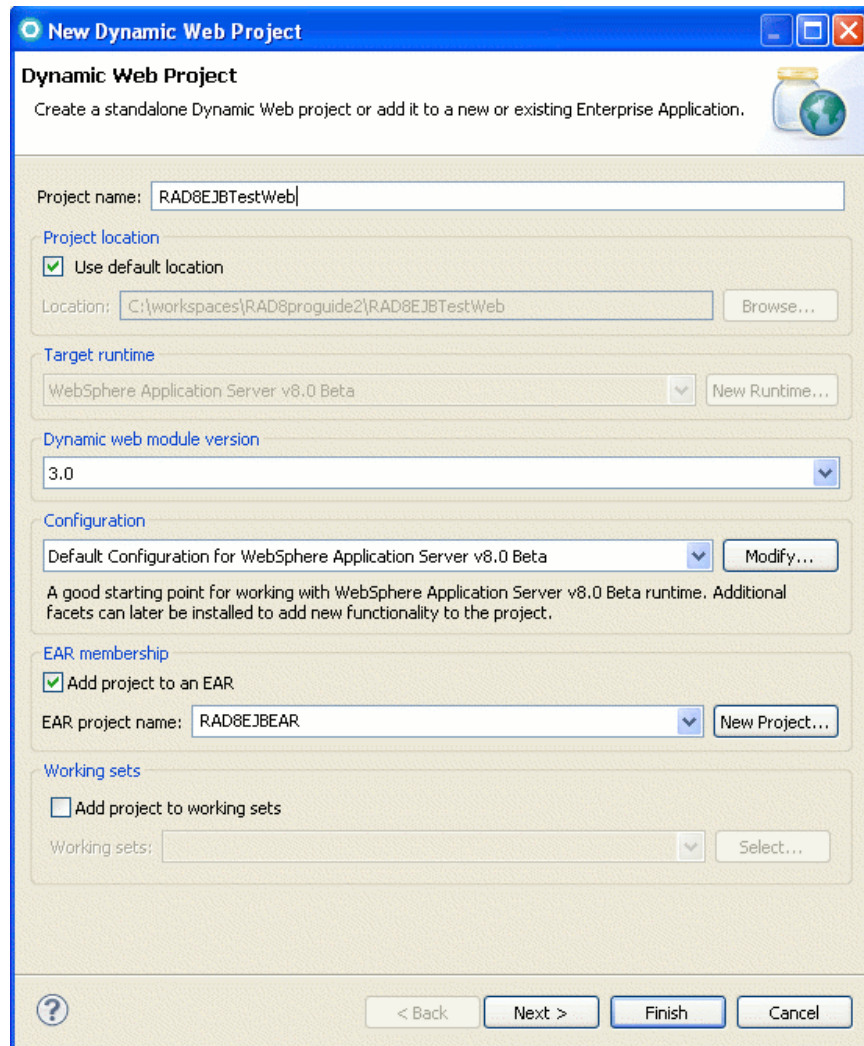


Figure 12-18 Create RAD8EJBTestWeb project

The Enterprise Explorer view contains the RAD8EJBTestWeb project, which is added to the RAD8EJBEAR enterprise application. Define the dependency to the EJB project RAD8EJB with the following steps:

1. Right-click the **RAD8EJBTestWeb** project and select **Properties**.
2. In the Properties window, select **Project References**, and for Project References, select the **RAD8JPA** module.
3. Click **OK**.

To create a new servlet within this RAD8EJBTestWeb project, perform the following steps:

1. Right-click the **RAD8EJBTestWeb** project and select **New** → **Servlet**.
2. For Package name, type `itso.test.servlet`, and for Class name, type `BankTest`, as shown in Figure 12-19.

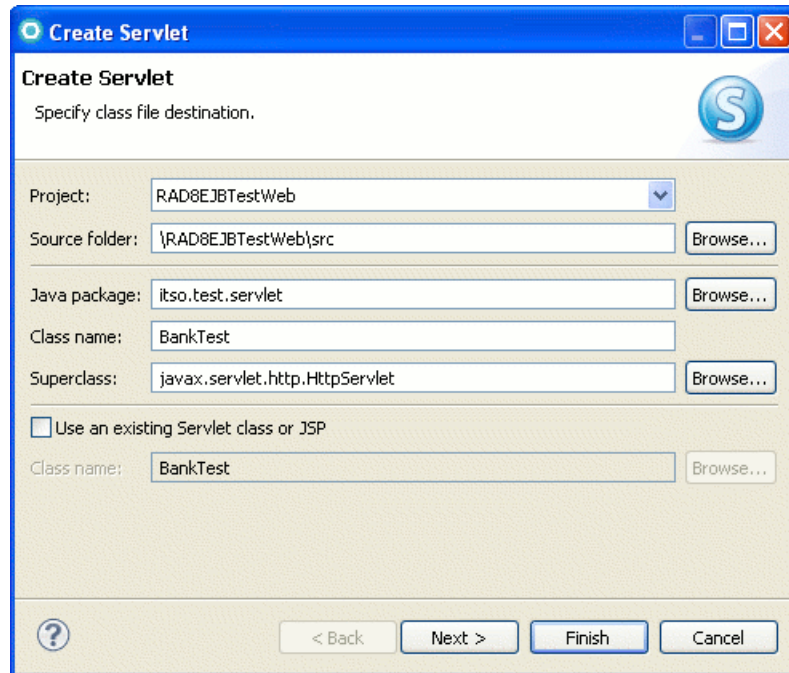


Figure 12-19 Creating servlet `BankTest`: Specifying the class file destination

3. Click **Next** twice.
4. Select to generate the **doPost** and **doGet** methods, as shown in Figure 12-20 on page 630.

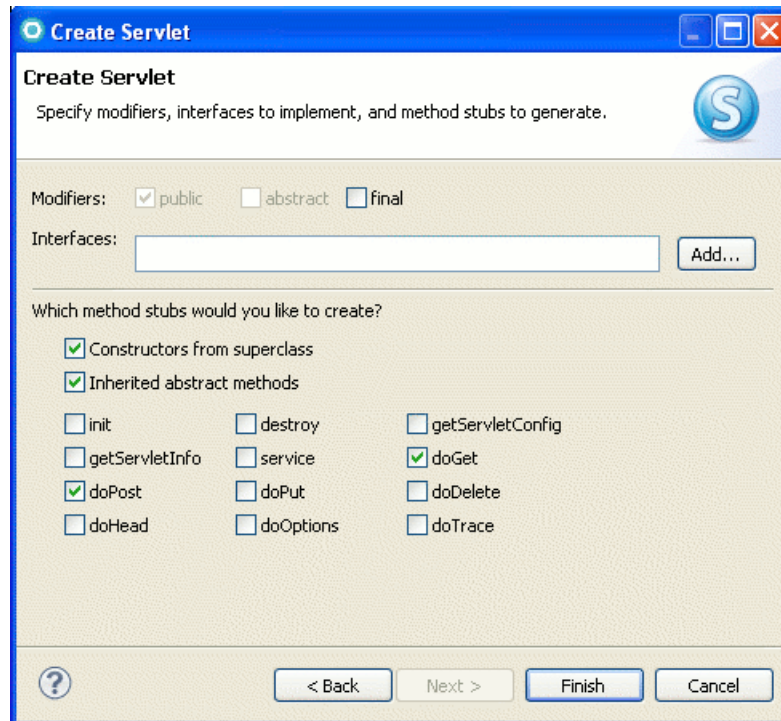


Figure 12-20 Creating servlet BankTest: Specifying interfaces and method stubs

5. Click **Finish**.
6. After the class definition BankTest, add an injector for the business interface:


```
@javax.ejb.EJB EJBBankService bank;
```

 The injection of the business interface into the servlet resolves to the automatic binding of the session EJB.
7. In the doGet method, enter the code:


```
doPost(request, response);
```
8. Complete the doPost method with the code that is shown in Example 12-32, which is available in the \7835code\ejb\source\BankTest.txt file. This servlet executes the methods of the session bean, after getting a reference to the business interface.

Example 12-32 Servlet to test the EJB 3.1 module (abbreviated)

```
protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
```

```

try {
    PrintWriter out = response.getWriter();
    String partialName = request.getParameter("partialName");
    out.println("<html><body><h2>Customer Listing</h2>");
    if (partialName == null) partialName = "%";
    else partialName = "%" + partialName + "%";

    out.println("<p>Customers by partial Name: " + partialName + "<br>");
    Customer[] customers = bank.getCustomers(partialName);

    for (Customer cust : customers) {
        out.println("<br>" + cust);
    }

    Customer cust1 = bank.getCustomer("222-22-2222");
    out.println("<p>" + cust1);

    Account[] accts = bank.getAccounts(cust1.getSsn());
    out.println("<br>Customer: " + cust1.getSsn() + " has " + accts.length + "
accounts");
    Account acct = bank.getAccount("002-222002");
    out.println("<p>" + acct);

    out.println("<p>Transactions of account: " + acct.getId());
    Transaction[] trans = bank.getTransactions("002-222002");
    out.println("<p><table border=1><tr><th>Type</th><th>Time</th>...");
    for (Transaction t : trans) {
        out.println("<tr><td>" + t.getTransType() + "</td><td>" + ...);
    }
    out.println("</table>");

    String newssn = "xxx-xx-xxxx";
    bank.deleteCustomer(newssn); // for rerun
    out.println("<p>Add a customer: " + newssn);
    Customer custnew = new Customer();
    custnew.setSsn(newssn);
    custnew.setTitle("Mrs");
    custnew.setFirstName("Lara");
    custnew.setLastName("Keen");
    bank.addCustomer(custnew);
    Customer cust2 = bank.getCustomer(newssn);
    out.println("<br>" + cust2);

    out.println("<p>Open two accounts for customer: " + newssn);
    String id1 = bank.openAccount(newssn);

```

```

String id2 = bank.openAccount(newssn);
out.println("<br>New accounts: " + id1 + " " + id2);
Account[] acctnew = bank.getAccounts(newssn);
out.println("<br>Customer: " + newssn + " has " + acctnew.length ...);
Account acct1 = bank.getAccount(id1);
out.println("<br>" + acct1);

out.println("<p>Deposit and withdraw from account: " + id1);
bank.deposit(id1, new java.math.BigDecimal("777.77"));
bank.withdraw(id1, new java.math.BigDecimal("111.11"));
acct1 = bank.getAccount(id1);
out.println("<br>Account: " + id1 + " balance " + acct1.getBalance());

trans = bank.getTransactions(id1);
out.println("<p><table border=1><tr><th>Type</th><th>Time</th>...");
for (Transaction t : trans) {
    out.println("<tr><td>" + t.getTransType() + ...");
}
out.println("</table>");

out.println("<p>Close the account: " + id1);
bank.closeAccount(newssn, id1);

out.println("<p>Update the customer: " + newssn);
bank.updateCustomer(newssn, "Mrs", "Sylvi", "Sollami");
cust2 = bank.getCustomer(newssn);
out.println("<br>" + cust2);
out.println("<p>Delete the customer: " + newssn);
bank.deleteCustomer(newssn);

out.println("<p>Retrieve non existing customer: ");
Customer cust3 = bank.getCustomer("zzz-zz-zzzz");
out.println("<br>customer: " + cust3);

    out.println("<p>End</body></html>");
} catch (Exception e) {
    System.out.println("Exception: " + e.getMessage());
    e.printStackTrace();
}
}

```

12.3.3 Testing the sample web application

To test the web application, run the servlet:

1. Expand the test web project **Deployment Descriptor** → **Servlets**. Select the **BankTest** servlet, right-click, and select **Run As** → **Run on Server**.
2. In the Run On Server window, select the **WebSphere Application Server v8.0 Beta** server, select **Always use this server when running this project**, and click **Finish**.
3. Accept the security certificate (if security is enabled).

Example 12-33 shows a sample output of the servlet.

Example 12-33 Servlet output (abbreviated)

```
Customer Listing
Customers by partial Name: %

Customer: 111-11-1111 Mr Henry Cui
Customer: 222-22-2222 Mr Craig Fleming
Customer: 333-33-3333 Mr Rafael Coutinho
Customer: 444-44-4444 Mr Salvatore Sollami
Customer: 555-55-5555 Mr Brian Hainey
Customer: 666-66-6666 Mr Steve Baber
Customer: 777-77-7777 Mr Sundaragopal Venkatraman
Customer: 888-88-8888 Mrs Lara Ziosi
Customer: 999-99-9999 Mrs Sylvi Lippmann
Customer: 000-00-0000 Mrs Venkata Kumari
Customer: 000-00-1111 Mr Martin Keen

Customer: 222-22-2222 Mr Craig Fleming
Customer: 222-22-2222 has 3 accounts

Account: 002-222002 balance 87.96

Transactions of account: 002-222002

Type Time Amount
Debit 2002-06-06 12:12:12.0 3.33
Credit 2003-07-07 14:14:14.0 6666.66
Credit 2004-01-08 23:03:20.0 700.77

Add a customer: xxx-xx-xxxx
Customer: xxx-xx-xxxx Mrs Lara Keen
```

```
Open two accounts for customer: xxx-xx-xxxx
New accounts: 00x-496969 00x-915357
Customer: xxx-xx-xxxx has 2 accounts
Account: 00x-496969 balance 0.00

Deposit and withdraw from account: 00x-496969
Account: 00x-496969 balance 666.66

Type Time Amount
Credit 2010-10-18 19:37:22.906 777.77
Debit 2010-10-18 19:37:23.0 111.11

Close the account: 00x-496969

Update the customer: xxx-xx-xxxx
Customer: xxx-xx-xxxx Mrs Sylvi Sollami

Delete the customer: xxx-xx-xxxx

Retrieve non existing customer:
customer: null

End
```

12.3.4 Visualizing the test application

You can improve the generated class diagram by adding the business interface, the entities, and the servlet to the diagram, as shown in Figure 12-21 on page 635.

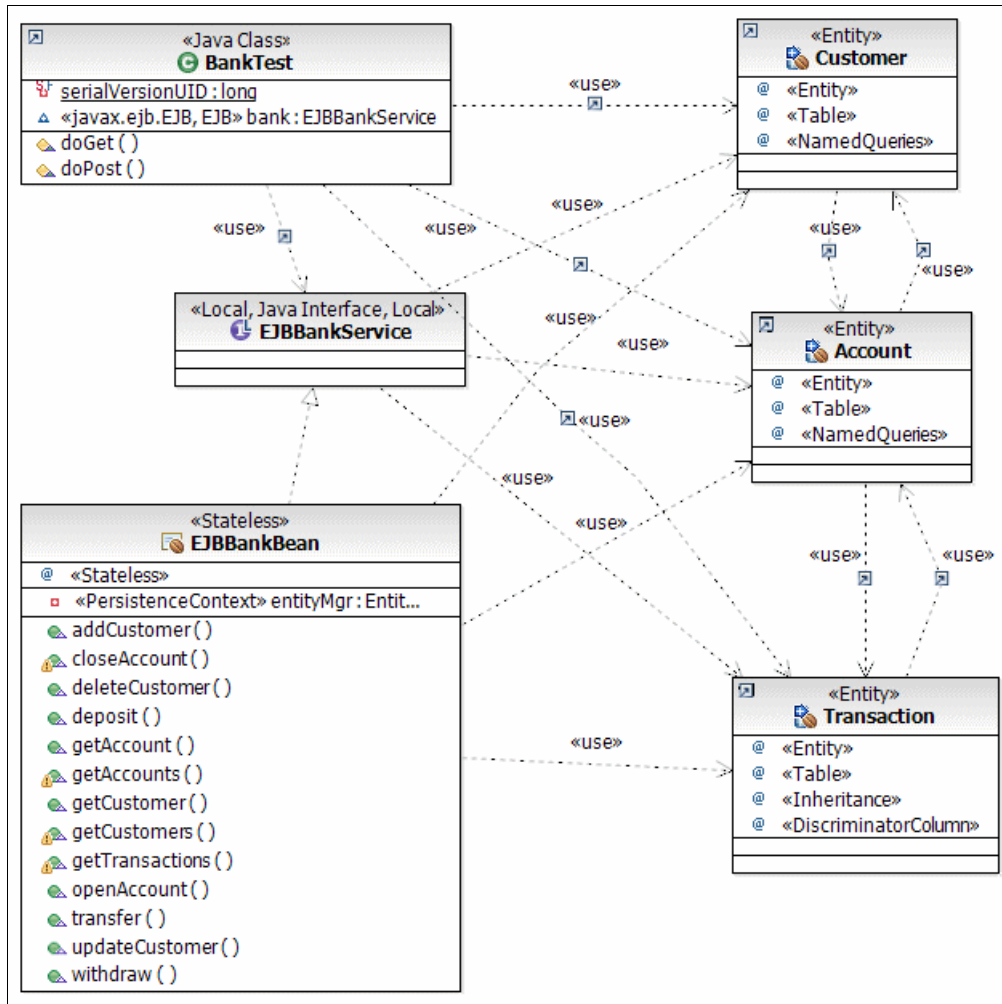


Figure 12-21 Class diagram of the test web application

12.4 Invoking EJBs from web applications

In this section, we describe how to create a web application. The RAD8EJBWeb application uses the JPA entities that are provided by the RAD8JPA project and accesses these entities through the EJBBankBean session bean of the RAD8EJB project.

12.4.1 Implementing the RAD8EJBWeb application

The RAD8EJBWeb application use EJB 3.1 APIs to communicate with the EJBBankBean session bean.

You can import the finished application from the \7835codesolution\ejb\RAD8EJBWeb.zip file.

Importing projects: If you already have RAD8EJB and RAD8JPA projects in the workspace, only import RAD8EJBWeb and RAD8EJBWebEAR.

Web application navigation

Figure 12-22 shows the navigation between the web pages.

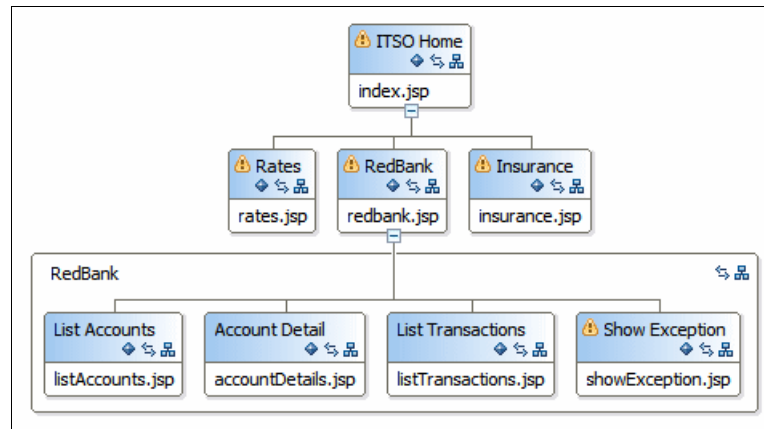


Figure 12-22 Website navigation

Note the following points:

- ▶ From the home page (index.jsp), there are three static pages (rates.jsp, insurance.jsp, and redbank.jsp).
- ▶ The redbank.jsp is the login panel for customers.
- ▶ After the login, the customer's details and the list of accounts are displayed (listAccounts.jsp).
- ▶ An account is selected in the list of accounts, and the details of the account and a form for transaction list, deposit, withdraw, and transfer operations are displayed (accountDetails.jsp).

- ▶ From the account details form, banking transactions are executed:
 - List transaction shows the list of previous debit and credit transactions (`listTransactions.jsp`).
 - Deposit, withdraw, and transfer operations are executed, and the updated account information is displayed in the same page.
- ▶ Additional functions are to delete an account, update customer information, add an account to a customer, and to delete the customer.
- ▶ In case of errors, an error page is displayed (`showException.jsp`).

The JSP are based on the template that provides navigation bars through headers and footers:

`/theme/itso_jsp_template.jtpl`, `nav_head.jsp`, `footer.jsp`

Servlets and commands

Several servlets provide the processing and switching between the web pages:

ListAccounts	Performs the customer login, retrieves the customer and the accounts, and forwards them to the <code>accountDetails.jsp</code> .
AccountDetails	Retrieves one account and forwards it to the <code>accountDetails.jsp</code> .
PerformTransaction	Validates the form values and calls one of the commands (<code>ListTransactionsCommand</code> , <code>DepositCommand</code> , <code>WithdrawCommand</code> , or <code>TransferCommand</code>). The commands perform the requested banking transaction and forwards it to the <code>listTransactions.jsp</code> or the <code>accountDetails.jsp</code> .
UpdateCustomer	Processes updates of customer information and the deletion of a customer.
DeleteAccount	Deletes an account and forwards it to the <code>listAccounts.jsp</code> .
NewAccount	Creates an account and forwards it to the <code>listAccounts.jsp</code> .
Logout	Logs out and displays the home page.

Java EE dependencies

The enterprise application (`RAD8EJBWebEAR`) includes the web module (`RAD8EJBWeb`), the EJB module (`RAD8EJB`), and the JPA Utility project (`RAD8JPA`).

The web module (`RAD8EJBWeb`) has a dependency on the EJB module (`RAD8EJB`), which has a dependency on the JPA project (`RAD8JPA`).

Accessing the session EJB

All database processing is done through the EJBBankBean session bean, using the business interface EJBBankService.

The servlets use EJB 3.1 injection to access the session bean:

```
@EJB EJBBankService bank;
```

After this injection, all the methods of the session bean can be invoked, such as the following methods that are shown in Example 12-34.

Example 12-34 EJBBankService methods

```
Customer customer = bank.getCustomer(customerNumber);  
Account[] accounts = bank.getAccounts(customerNumber);  
bank.deposit(accountId, amount);
```

Additional functionality

We improved the application and added the following functions:

- ▶ On the customer details panel (`listAccounts.jsp`), we added three buttons:
 - New Customer: Enter data into the title, first name, and last name fields, then click **New Customer**. A customer is created with a random Social Security number.
 - Add Account: This action adds an account to the customer, with a random account number and zero balance.
 - Delete Customer: Deletes the customer and all related accounts.

The logic for adding and deleting a customer is in the `UpdateCustomer` servlet. The logic for a new account is in `NewAccount` servlet.

- ▶ On the account details page (`accountDetails.jsp`), we added the **Delete Account** button. You click this button to delete the account with all its transactions. The customer with its remaining accounts is displayed next. The logic for deleting an account is in `DeleteAccount` servlet.
- ▶ For the Login panel, we added logic in the `ListAccounts` servlet so that the user can enter a last name instead of the SSN.

If the search by SSN fails, we retrieve all customers with that partial name. If only one result is found, we accept it and display the customer. This allows entry of partial names, such as `So%`, to find the `Solami` customer.

12.4.2 Running the web application

Before running the web application, we must have the data source for the ITSOBANK database configured. See 12.2.4, “Setting up the ITSOBANK database” on page 609, for instructions. You can either configure the enhanced EAR in the RAD8EJBWebEAR application or define the data source in the server. We suggest that you define the data source in the server, as described in “Configuring the data source in WebSphere Application Server” on page 1882.

To run the web application, perform these steps:

1. In the Servers view, right-click the server and select **Add and Remove Projects**. Remove the **RAD8EJB** application and add the **RAD8EJBWebEAR** application. Then click **Finish**.
2. Right-click the **RAD8EJBWeb** project and select **Run As** → **Run on Server**.
3. When prompted, select **WebSphere Application Server v8.0 Beta**.
4. Your start page is the `redbank.jsp` login page, as shown in Figure 12-23. Because we want to focus on the Redbank application, we set the `redbank.jsp` as a welcome-list entry in the `web.xml` configuration file, as well.

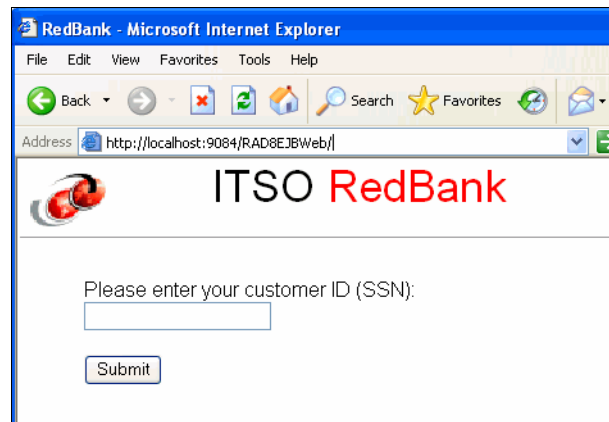


Figure 12-23 RedBank: Login

5. Enter a customer number, such as 333-33-3333, and click **Submit**. The customer details and the list of accounts are displayed in Figure 12-24.

The screenshot shows a web browser window titled "List Accounts" with the URL `http://localhost:9081/RAD8EJBWeb/ListAccounts`. The page header features the ITSO RedBank logo and navigation tabs for "Rates", "RedBank", and "Insurance". The main content area displays customer information for SSN 333-33-3333, with fields for Title (Mr), First name (Rafeel), and Last name (Coutinho). Below this information are buttons for "Update", "New Customer", and "Delete Customer". A table lists three accounts with their respective balances. The first account, 003-333001, is highlighted with a red box. At the bottom of the page are buttons for "Add Account" and "Logout". The footer contains links for "ITSO Home" and "RedBank".

Account Number	Balance
003-333001	9,041.85
003-333002	1,568.79
003-333003	21.56

Figure 12-24 RedBank: Customer with accounts

6. Click an account, such as **003-333001**, and the details and possible actions are displayed, as shown in Figure 12-25 on page 641.

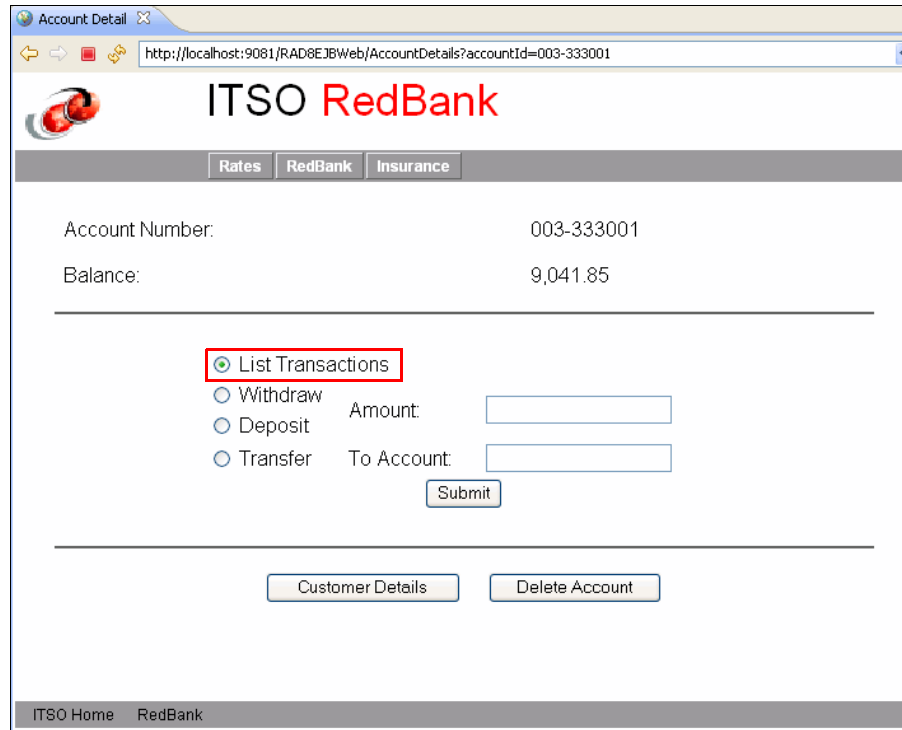
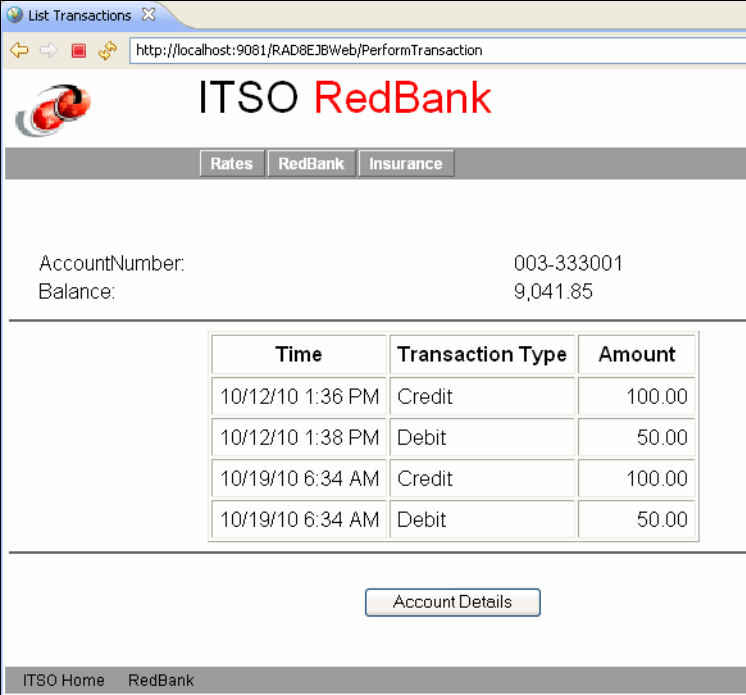


Figure 12-25 RedBank: Account details

7. Select **List Transactions** and click **Submit**. The transactions are listed, as shown in Figure 12-26.



AccountNumber: 003-333001
Balance: 9,041.85

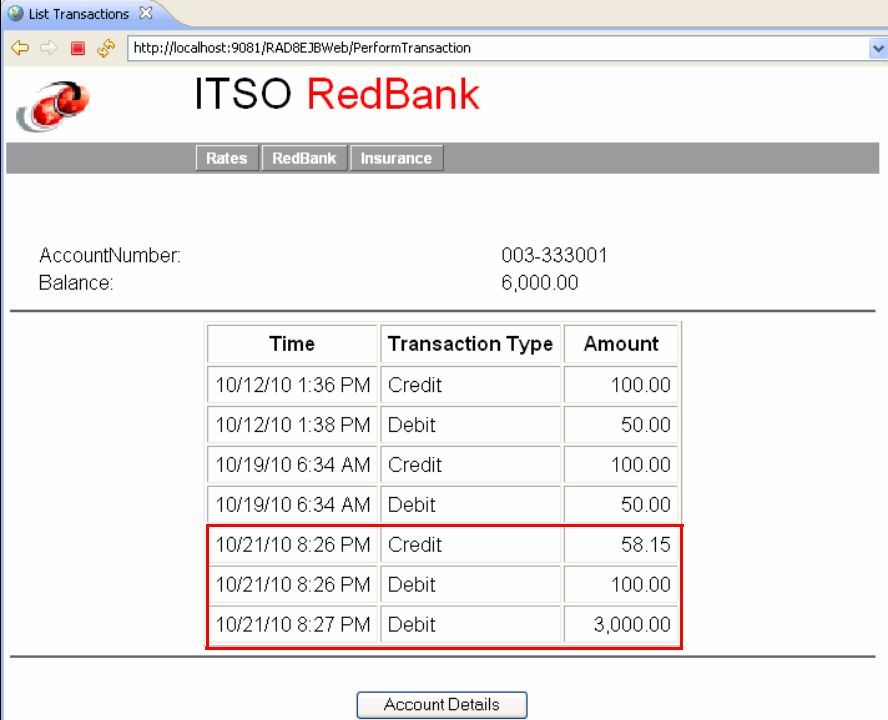
Time	Transaction Type	Amount
10/12/10 1:36 PM	Credit	100.00
10/12/10 1:38 PM	Debit	50.00
10/19/10 6:34 AM	Credit	100.00
10/19/10 6:34 AM	Debit	50.00

[Account Details](#)

Figure 12-26 RedBank: Transactions

8. Click **Account Details** to return to the account.
9. Select **Deposit**, enter an amount (58.15), and click **Submit**. The balance is updated to 9,100.00.
10. Select **Withdraw**, enter an amount (100), and click **Submit**. The balance is updated to 9,000.00.
11. Select **Transfer**. Enter an amount (3000) and a target account (003-333002) and click **Submit**. The balance is updated to 6,000.00.

12. Select **List Transactions** and click **Submit**. The transactions are listed and there are three more entries, as shown in Figure 12-27.



AccountNumber: 003-333001
Balance: 6,000.00

Time	Transaction Type	Amount
10/12/10 1:36 PM	Credit	100.00
10/12/10 1:38 PM	Debit	50.00
10/19/10 6:34 AM	Credit	100.00
10/19/10 6:34 AM	Debit	50.00
10/21/10 8:26 PM	Credit	58.15
10/21/10 8:26 PM	Debit	100.00
10/21/10 8:27 PM	Debit	3,000.00

Account Details

Figure 12-27 RedBank: Transactions added

13. Click **AccountDetails** to return to the account. Click **Customer Details** to return to the customer.
14. Click the second account and then click **Submit**. You can see that the second account has a transaction from the transfer operation, as shown in Figure 12-28 on page 644.

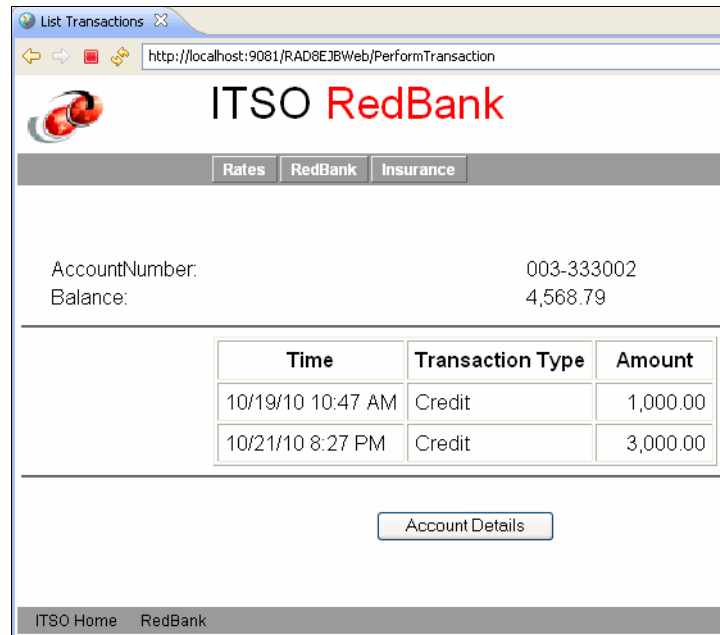


Figure 12-28 Transfer result to account 003-333002

15. Back in the customer details, change the last name and click **Update**. The customer information is updated.
16. Overtyping the first and last names with Jonny Lippmann and clicking **New Customer**. Then a new customer with SSN 395-60-9710 is created, as shown in Figure 12-29 on page 645.

17. Click **Add Account**, and an account 003-365234 with balance 0.00 is added to the customer, as shown in Figure 12-29.

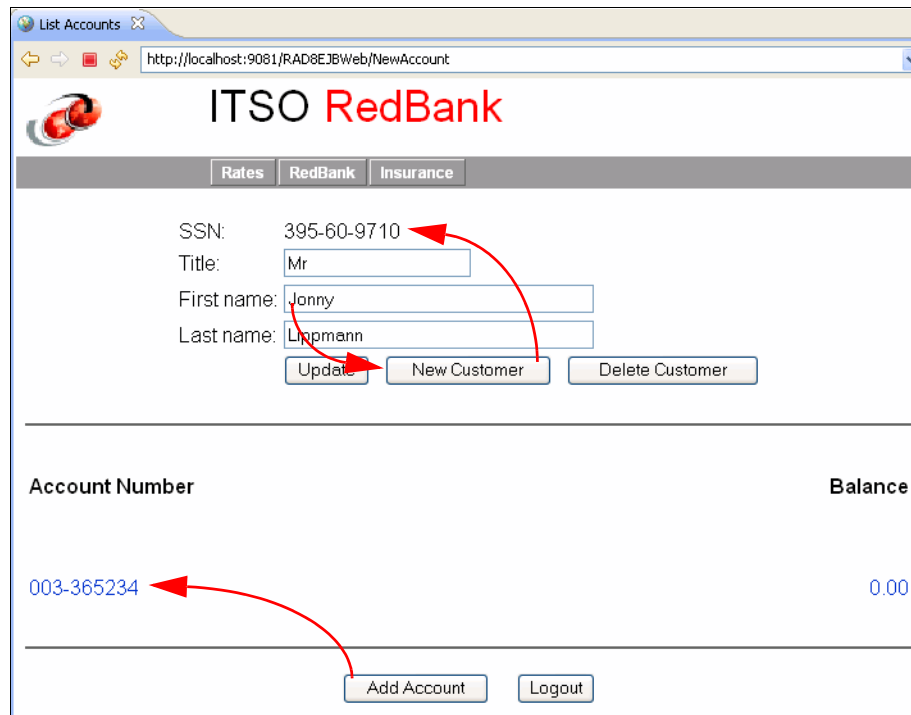


Figure 12-29 RedBank: New customer and new account

18. Perform transactions on the new account.
19. Go back to customer details and click **Delete Customer**.
20. In the Login panel, enter an incorrect value and click **Submit**. The customer details panel is displayed with a "NOT FOUND" last name.
21. Click **Logout**.

12.4.3 Cleaning up

Remove the RAD8EJBWebEAR application from the server.

12.4.4 Adding a remote interface

For testing by using JUnit and for certain web applications, as described in “Configuring the data source in WebSphere Application Server” on page 1882, we define a remote interface for the EJBBankBean session bean. Perform the following steps:

1. In the RAD8EJB project, `itso.bank.service` package, create an interface named `EJBBankRemote`, which extends the business interface, `EJBBankService`.
2. Add one method to the interface, `getCustomersAll`, to retrieve all the customers.
3. Add an `@Remote` annotation before your interface class definition. Example 12-35 shows the defined remote interface.

Example 12-35 Remote interface of the session bean

```
package itso.bank.service;
import itso.bank.entities.Customer;
import javax.ejb.Remote;
@Remote
public interface EJBBankRemote extends EJBBankService {
    public Customer[] getCustomersAll();
}
```

4. Open the `EJBBankBean` session bean:
 - a. Add the `EJBBankRemote` interface to the implements list.
 - b. Implement the `getCustomersAll` method, as shown in Example 12-36. This method is similar to the `getCustomers` method, using the `getCustomers` named query (without a parameter).

Example 12-36 Extend EJBBankBean with remote interface

```
public class EJBBankBean implements EJBBankService, EJBBankRemote
{
    .....
    public Customer[] getCustomersAll() {
        System.out.println("getCustomers: all");
        Query query = null;
        try {
            query = entityMgr.createNamedQuery("getCustomers");
            List<Customer> beanlist = query.getResultList();
            Customer[] array = new Customer[beanlist.size()];
            return beanlist.toArray(array);
        } catch (Exception e) {
```

```
        System.out.println("Exception: " + e.getMessage());
        return null;
    }
}
```

12.5 More information

For more information about EJB 3.1 and EJB 3.0, see the following resources:

- ▶ *WebSphere Application Server Version 6.1 Feature Pack for EJB 3.0*, SG24-7611
- ▶ *JSR 318: Enterprise JavaBeans 3.1*:
<http://jcp.org/en/jsr/summary?id=318>
- ▶ WebSphere Application Server Information Center:
http://publib.boulder.ibm.com/infocenter/wasinfo/beta/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tejb_timerserviceejb_enh.html



Developing Java Platform, Enterprise Edition (Java EE) application clients

In this chapter, we introduce Java Platform, Enterprise Edition (Java EE) application clients and the facilities supplied by the Java EE application client container. In addition, we highlight the features provided by Rational Application Developer for developing and testing Java EE application clients.

The chapter is organized into the following sections:

- ▶ Introduction to Java EE application clients
- ▶ Overview of the sample application
- ▶ Preparing the sample application
- ▶ Developing the Java EE application client
- ▶ Testing the Java EE application client
- ▶ Packaging the Java EE application client

The sample code for this chapter is in the `7835code\j2eec1ient` folder.

13.1 Introduction to Java EE application clients

A J2EE application server, similar to WebSphere Application Server, exposes several types of resources to remote clients:

- ▶ Enterprise JavaBeans (EJB)
- ▶ Java Database Connectivity (JDBC) data sources
- ▶ Java Message Service (JMS) resources (queues and topics)
- ▶ Java Naming and Directory Interface (JNDI) services

These resources are most often accessed from a component that is running within the Java EE application server itself, such as an EJB, servlet, or JSP. However, these resources can also be used from a stand-alone Java application (known as a *Java EE Application Client*) running in its own Java virtual machine (JVM), possibly on a computer that is separate from the server. Figure 13-1 shows the resource access scenarios described.

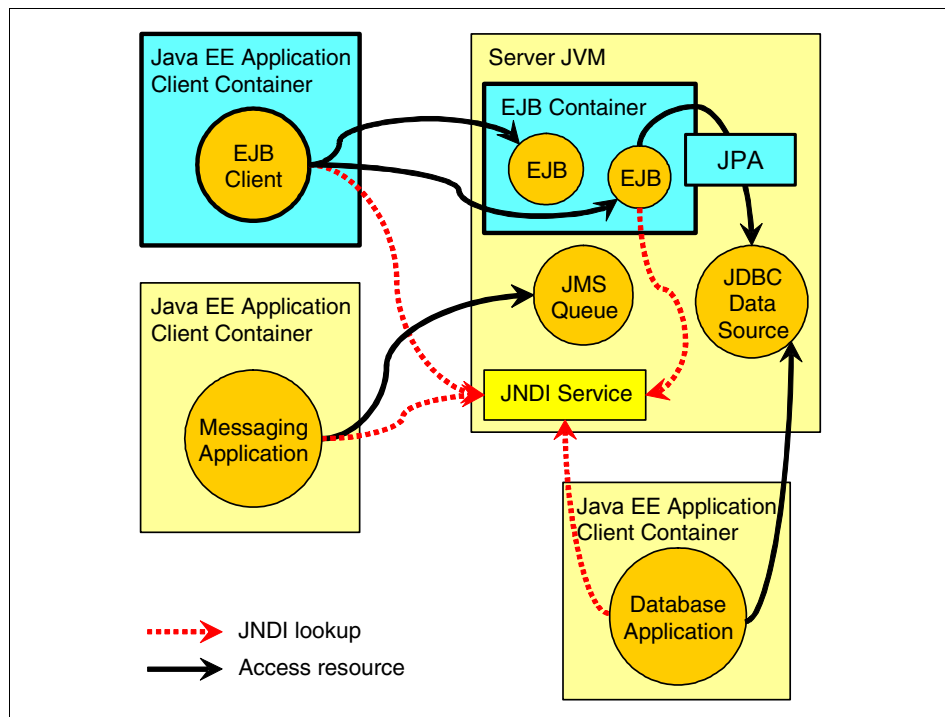


Figure 13-1 Java applications using Java EE server resources

Clients: The clients shown in Figure 13-1 might conceptually be running on the same physical node or in the same JVM as the application server. However, in this chapter, the focus is on clients running in distributed environments. Throughout this chapter, we develop an EJB client, invoking the EJB from Chapter 12, “Developing Enterprise JavaBeans (EJB) applications” on page 577, to provide a simple ITSO Bank client application.

Because a regular JVM does not support accessing such application server resources, additional setup for the runtime environment is required for a Java EE application. There are two methods to achieve this setup:

- ▶ Add the required packages to the Java Runtime Environment manually.
- ▶ Package the application according to the Java EE application client specification, and execute the application in a Java EE application client container.

In this chapter, we focus on the second option. In addition to providing the correct runtime resources for Java applications accessing Java EE server resources, the Java EE application client container provides additional features, such as mapping references to JNDI names and integration with server security features.

IBM WebSphere Application Server v8.0 Beta includes a Java EE application client container and a facility for starting Java EE application clients. The Java EE application client container, known as *Application Client for WebSphere Application Server*, can be installed separately from the WebSphere Application Server installation CDs, or downloaded from developerWorks, and runs a completely separate JVM on the client machine.

When the JVM starts, it loads the necessary runtime support classes to make it possible to communicate with WebSphere Application Server and to support Java EE application clients that will use server-side resources.

Application Client for WebSphere Application Server: Although the Java EE specification describes the JAR format as the packaging format for Java EE application clients, the Application Client for WebSphere Application Server expects the application to be packaged as a JAR inside an Enterprise Application Archive (EAR). The Application Client for WebSphere Application Server does not support the execution of a stand-alone Java client JAR.

Rational Application Developer includes tooling to assist with developing and configuring Java EE application clients, and a test facility that allows Java EE application clients to be executed in an appropriate container. The focus of this

chapter is on the Rational Application Developer tooling for Java EE application clients. We look only at this facility.

The Java EE programming model simplifies the process of creating Java applications. Java Enterprise applications (Java EE applications) are applications that conform to the Java Platform, Enterprise Edition (Java EE) specification. Prior to Java EE, the specification name was Java 2 Platform, Enterprise Edition (J2EE). The term Java EE includes Java EE and J2EE specifications.

In the Java EE specifications, programming requirements have been streamlined, and XML deployment descriptors are optional. Instead, you can specify many details of assembly and deployment with Java annotations. Java EE provides default values in many situations so explicit specification of these values is not required.

Code validation, content assistance, Quick Fixes, and refactoring simplify working with your code. Code validators check your projects for errors. When an error is found, you can double-click it in the Problems view in the product workbench to go to the error location. For certain error types, you can use a Quick Fix to correct the error automatically. For both Java source and Java annotations, you can rely on content assistance to simplify your programming task. When you refactor source code, the tools automatically update the associated metadata.

For additional information about Java EE, see the official specification:

- ▶ *Java EE 5: Java Specification Request (JSR) 244: Java Platform, Enterprise Edition 5 (Java EE 5) Specification*
- ▶ *Java EE 6: JSR 316: Java Platform, Enterprise Edition 6 (Java EE 6) Specification*

13.2 Overview of the sample application

In this chapter, we develop a simple Java EE application client. It invokes the services of the EJB application that was developed in Chapter 12, “Developing Enterprise JavaBeans (EJB) applications” on page 577, to look up the customer information and account overview from a specified customer Social Security number (SSN).

The application uses a graphical user interface (GUI), implemented with Swing components, which shows the details for the customer with SSN 444-44-4444 (Figure 13-2 on page 653).

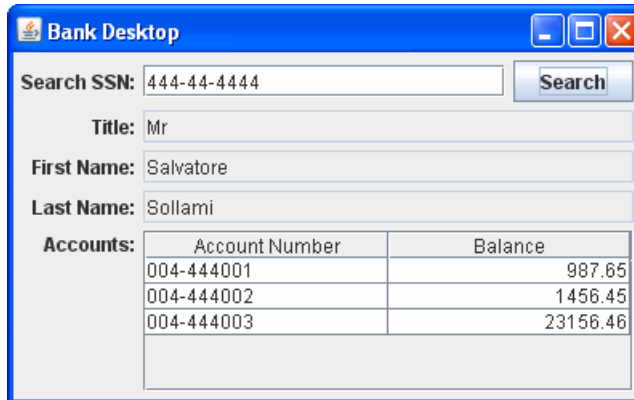


Figure 13-2 Interface for the sample application client

Figure 13-3 on page 654 shows a class diagram of the finished sample application:

- ▶ The classes on the right side of the class diagram are from the EJB enterprise application. The three classes on the left are part of the application client.
- ▶ As the class diagram outlines, the application client controller class, `BankDesktopController`, uses the `EJBBankBean` session EJB to retrieve `Customer` and `Account` object instances, representing the customer and associated accounts that are retrieved from the `ITSOBANK` database.

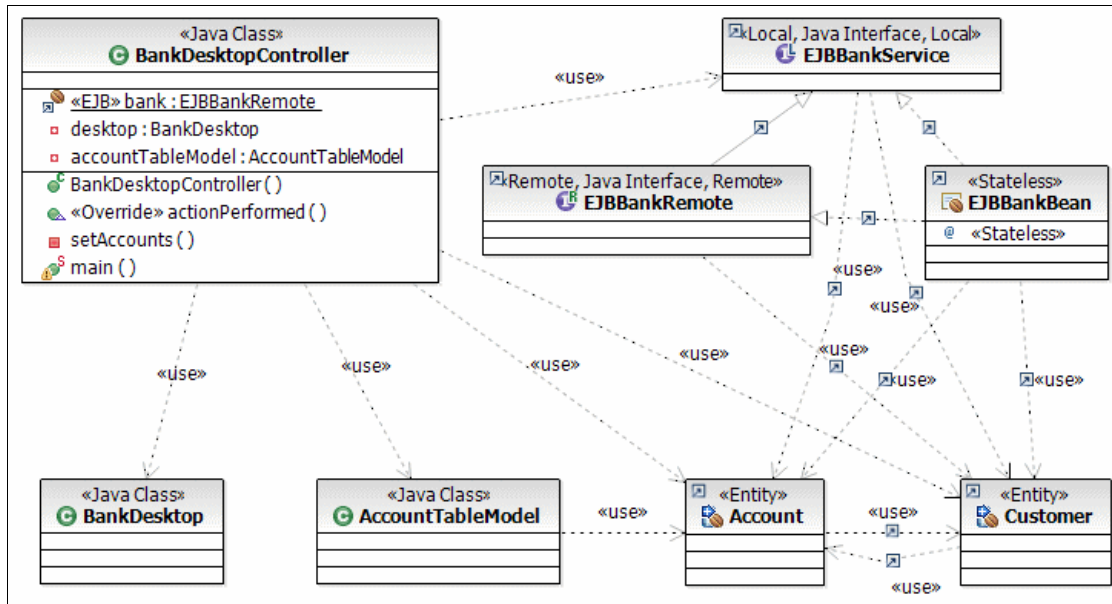


Figure 13-3 Class diagram for the Bank Java EE application client

13.3 Preparing the sample application

Prior to working on the sample for this chapter, we have to set up the database for the sample application, import the sample application, and ensure that everything works.

13.3.1 Importing the enterprise application sample

To import the enterprise application sample that we use as a starting point for this chapter, follow these steps:

1. From the workbench, select **File** → **Import**.
2. In the Import window, select **General** → **Existing Projects into Workspace** and click **Next**.
3. In the Import Projects window, select **Select archive file** and locate the `c:\7835code\j2eeclient\RAD8AppClient.zip` file.
4. Select all projects (**RAD8EJB**, **RAD8EJB**, **RAD8EJBTestWeb**, and **RAD8JPA**). Click **Finish**.

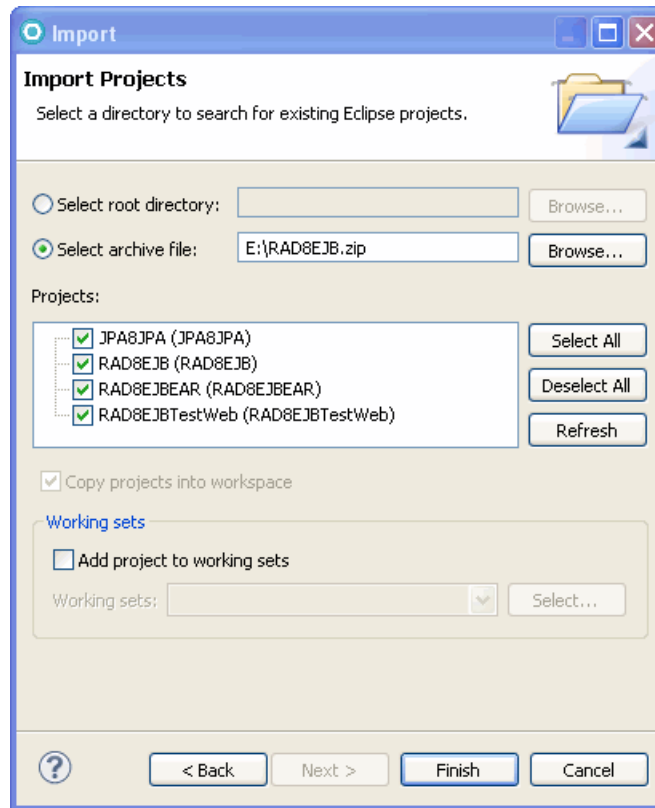


Figure 13-4 Import wizard to import the project into the workspace

After the Import wizard (Figure 13-4) has completed the import, four projects are added to the workspace:

- ▶ RAD8EJB: This project contains the EJB that makes up the business logic of the ITS0 Bank. The EJBBean session bean acts as a facade for the EJB application. This project is packaged inside RAD8EJBEAR when exported and deployed on an application server.
- ▶ RAD8EJBEAR: This project is the deployable enterprise application, which functions as a container for the remaining projects. This enterprise application must be executed on an application server.
- ▶ RAD8EJBTestWeb: This project is the sample web application that is developed to test the EJB 3.0 session bean and entity model. This project is packaged inside RAD8EJBEAR when exported and deployed on an application server.
- ▶ RAD8JPA: This Java project holds JPA entities that are passed between the session facade and the client applications.

13.3.2 Setting up the sample database

The JPA entities used in this sample are based on the ITSOBANK database. Therefore, we must define a database connection within Rational Application Developer that the mapping tools use to extract schema information from the database.

Setting up the ITSOBANK database

See “Setting up the ITSOBANK database” on page 1880 for instructions to create the ITSOBANK database. We can either use the DB2 or Derby database. For simplicity, we use the built-in Derby database in this chapter.

Configuring the data source

Use the JPA tools to configure the data source for the ITSOBANK database:

1. In the Enterprise Explorer view of the Java EE perspective, right-click **RAD8JPA** and select **JPA Tools** → **Configure Project for JDBC Deployment**.
2. In the Set up connections for deployment dialog window, click **Add connections**.
3. In the New Connection dialog window, perform the following tasks (Figure 13-5 on page 657):
 - a. Select a database manager of **Derby**.
 - b. Set the JDBC driver to **Derby 10.2 - Embedded JDBC Driver Default**.
 - c. Set the database location to point to the location of the ITSOBANK database (for example, C:\7835code\database\derby\ITSOBANK).

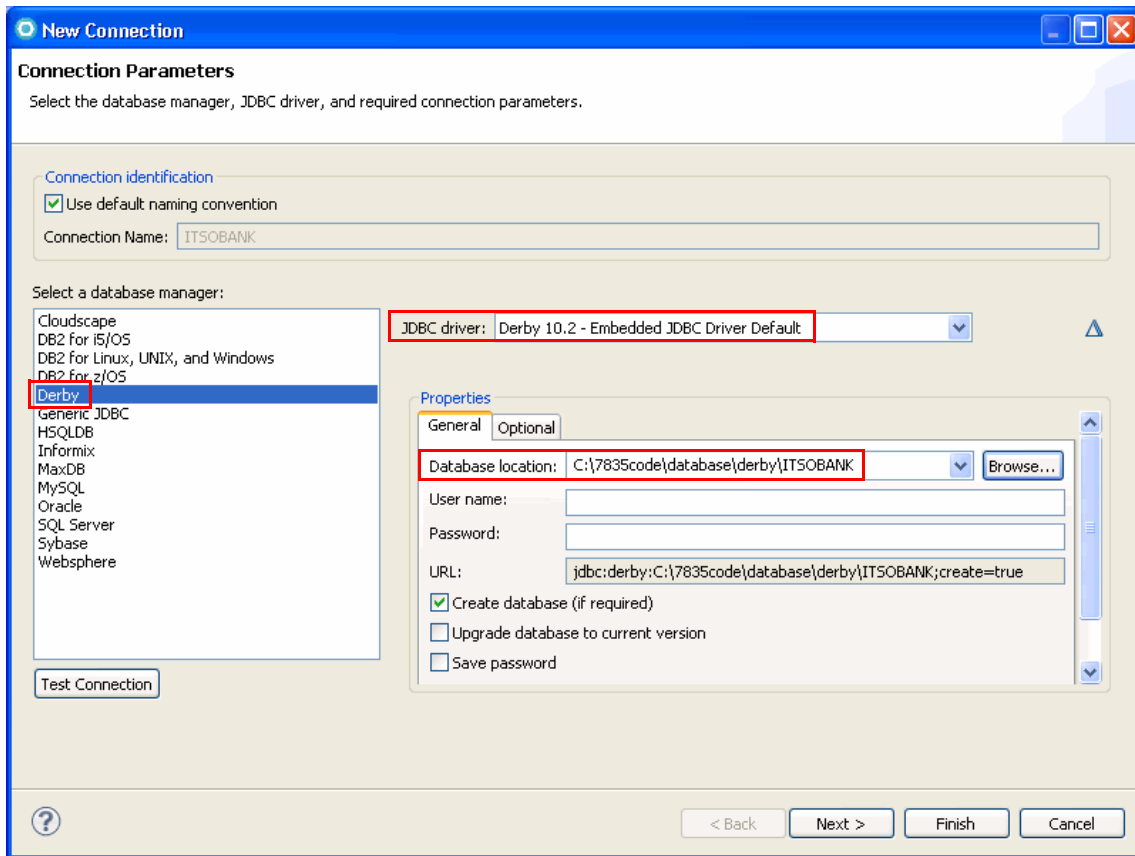


Figure 13-5 New JDBC connection

4. Click **Finish**.
5. Back in the Set up connections for deployment dialog window (Figure 13-6 on page 658), click **OK**.

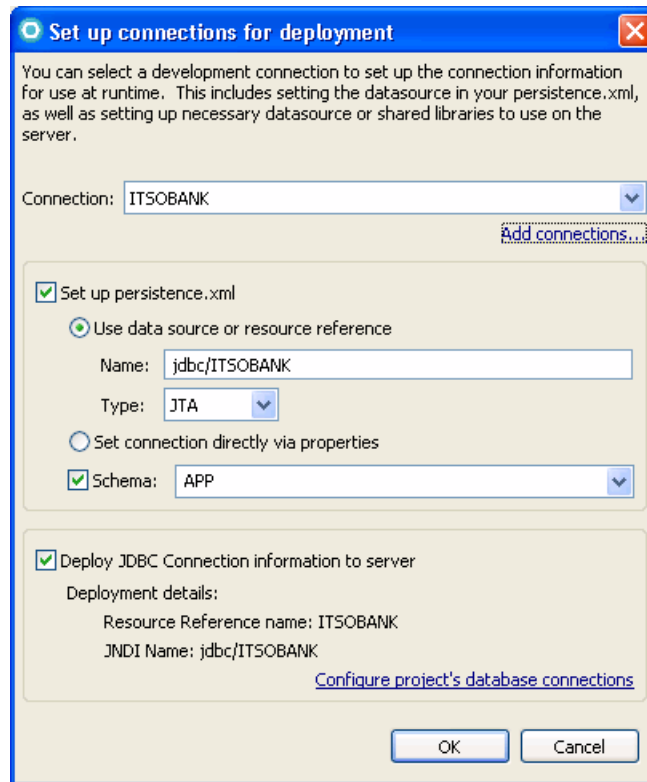


Figure 13-6 Setting up JDBC connections

Testing the sample application

To test the sample application, perform the following steps:

1. Start WebSphere Application Server in the Servers view.
2. In the Enterprise Explorer, expand **RAD8EJBTestWeb** → **Java Resources** → **src** → **itso.test.servlet**.
3. Right-click **BankTest.java** and select **Run As** → **Run on Server**.
4. In the **Run On Server** dialog, click **Finish**.
5. If the sample application is able to connect successfully to the ITSOBANK data source, you will see the results shown in Figure 13-7 on page 659.

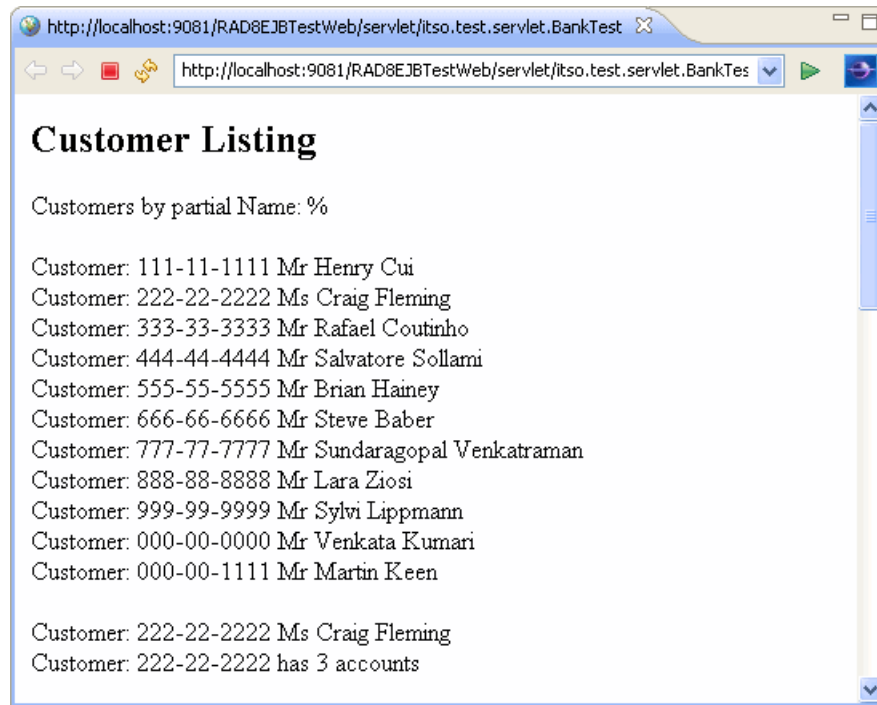


Figure 13-7 Test showing customer information from the ITSOBANK database

13.4 Developing the Java EE application client

In this section, we show how to use Rational Application Developer to create a project that contains a Java EE application client. This application client will be associated with its own enterprise application.

New client application: Do not use the new client application with the existing RAD8EJBEAR enterprise application, although this approach is possible. The EJB project contains other server resources that must not be distributed to the clients, such as passwords or proprietary business logic.

To develop the Java EE application client sample, we complete the following tasks:

- ▶ Creating the Java EE application client projects
- ▶ Configuring the Java EE application client projects
- ▶ Importing the graphical user interface and control classes
- ▶ Creating the BankDesktopController class

- ▶ Completing the BankDesktopController class
- ▶ Creating an EJB reference and binding
- ▶ Registering the BankDesktopController class as the main class

13.4.1 Creating the Java EE application client projects

To create a Java EE application client project, follow these steps:

1. In the Java EE perspective, select **File** → **New** → **Project**. Expand the **Java EE** folder and select **Application Client Project**. Click **Next**.

Alternatively, right-click in the Enterprise Explorer view and select **New** → **Application Client Project**.

2. In the New Application Client Project window (Figure 13-8 on page 661), perform these tasks:
 - a. For Project name, type RAD8AppClient.
 - b. For EAR Project Name, type RAD8AppClientEAR.
 - c. For Target Runtime, accept the default of **WebSphere Application Server v8.0 Beta**. For Application Client module version, accept the default of **6.0**.
 - d. Click **Next**.

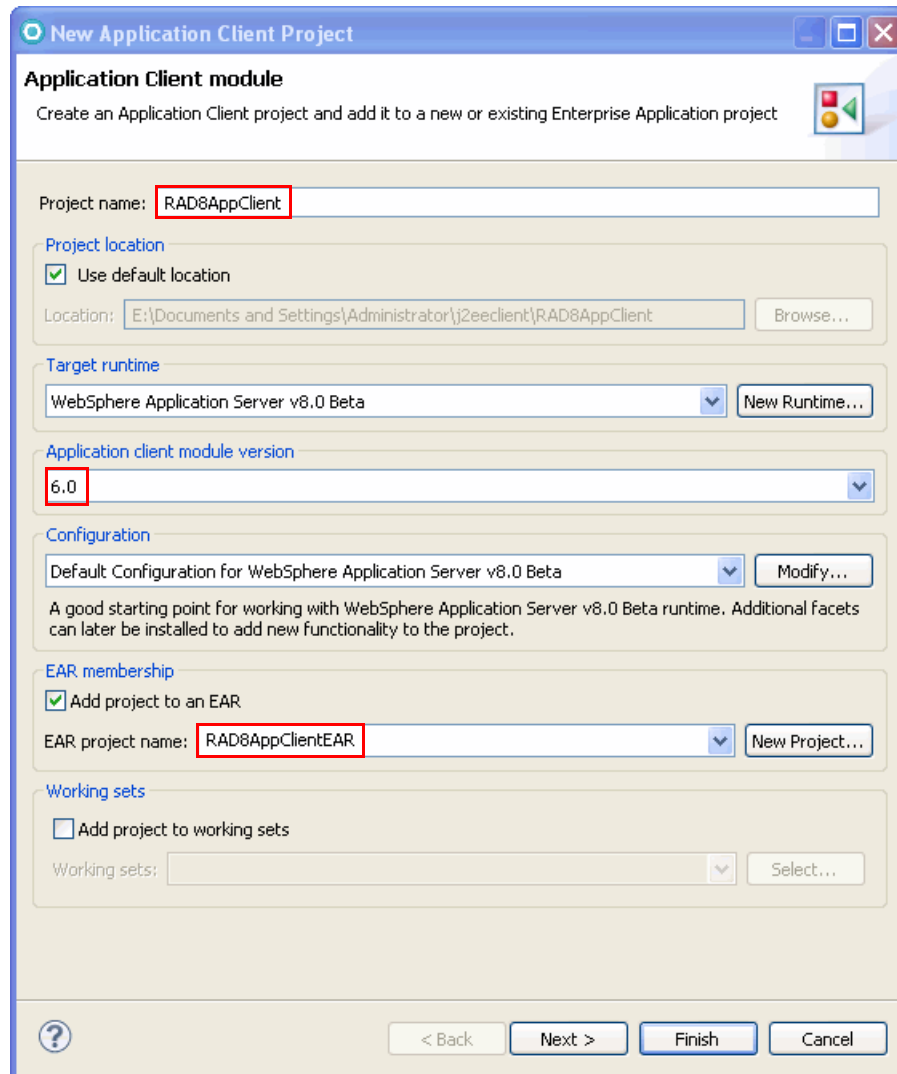


Figure 13-8 New Application Client Project

3. In the Application Client module window (Figure 13-9 on page 662), verify the presence of the source folder on the build path named `appClientModule` and click **Next**.

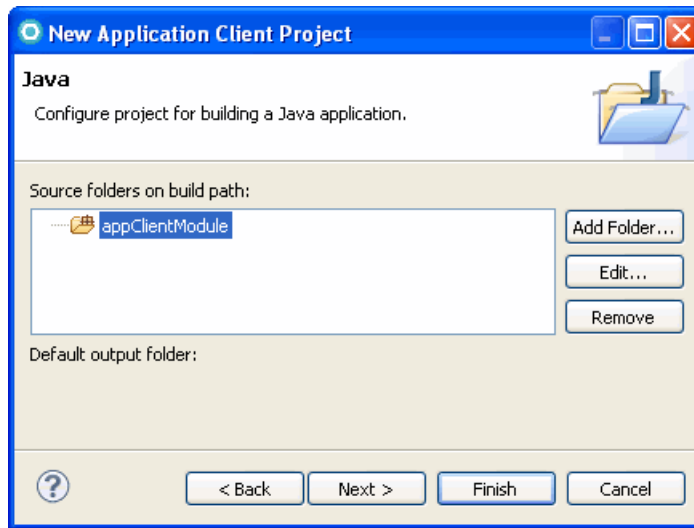


Figure 13-9 Application Client module

4. In the next window of the Application Client Module (Figure 13-10), clear **Create a default Main class**, select **Generate application-client.xml deployment descriptor**, and click **Finish**.

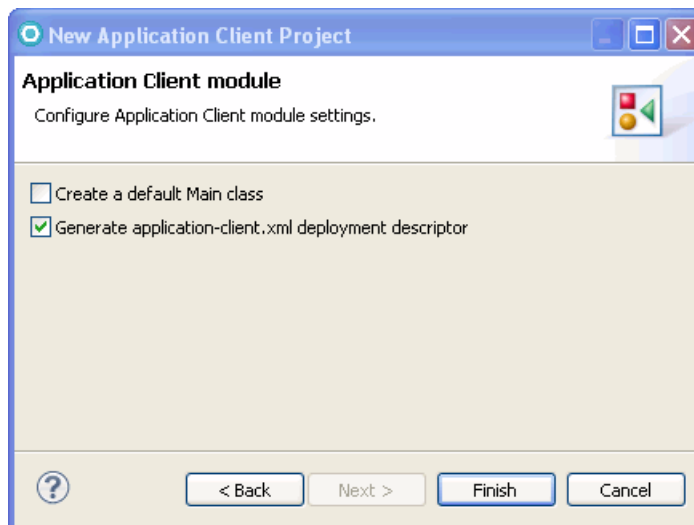


Figure 13-10 Application Client Module (continued)

When the wizard is complete, the following two projects are created in your workspace:

- ▶ **RAD8AppClientEAR**: This enterprise application project acts as a container for the code to be deployed on the application client node.
- ▶ **RAD8AppClient**: This project contains the code for the application client. For now, it is empty, except for the `META-INF/application-client.xml` file, which is the application client deployment descriptor.

13.4.2 Configuring the Java EE application client projects

The application client project has to reference the **RAD8EJB** and **RAD8JPA** projects. In this section, we configure this dependency by adding the **RAD8EJBClient** as a dependency to both projects:

1. In the Enterprise Explorer, right-click **RAD8AppClientEAR** and select **Properties**.
2. In the Properties window, select **Deployment Assembly**. In the right pane, click **Add**. In the New Assembly Directive window (Figure 13-11), select **Project** and click **Next**.

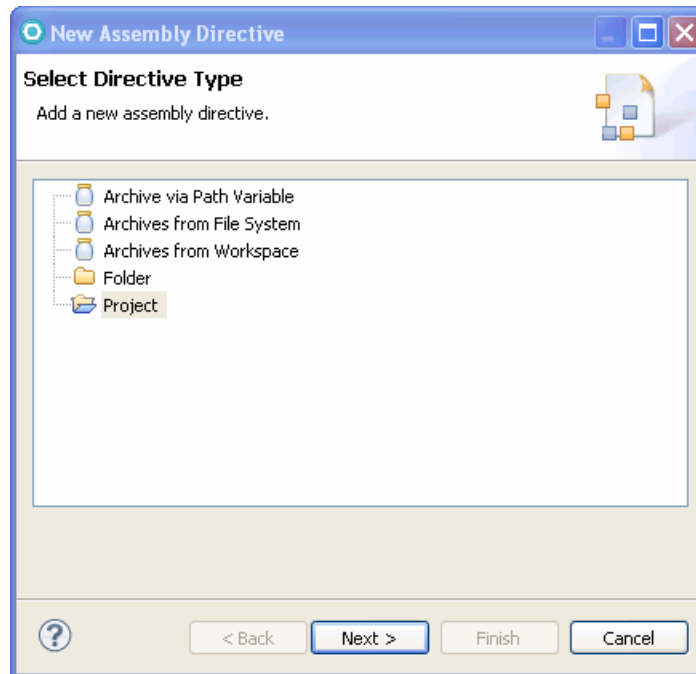


Figure 13-11 New Assembly Directive window

3. In the Projects window (Figure 13-12), select the **RAD8EJB** project and click **Finish**.

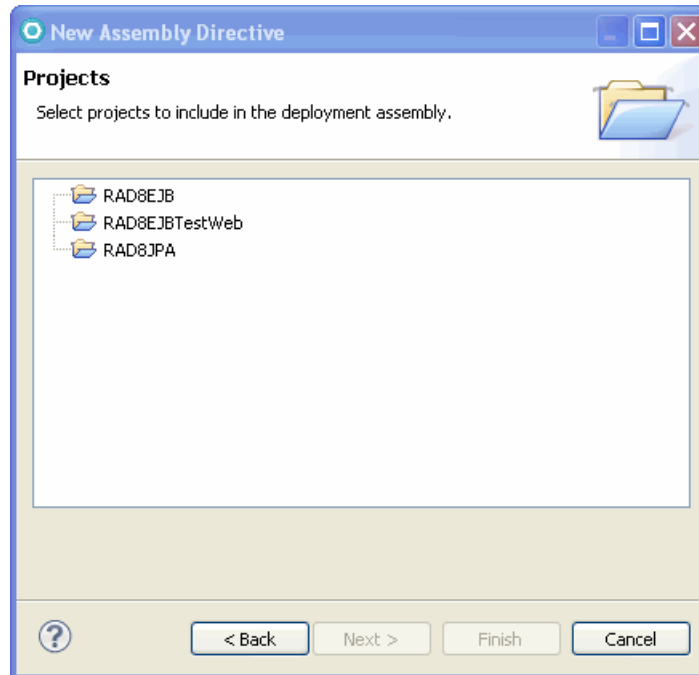


Figure 13-12 New Assembly Directive: Projects window (continued)

4. Repeat the same operation to add the RAD8JPA project to the Ear Module Assembly. The Ear Module Assembly with the RAD8EJB and RAD8JPA projects is shown in Figure 13-13 on page 665.

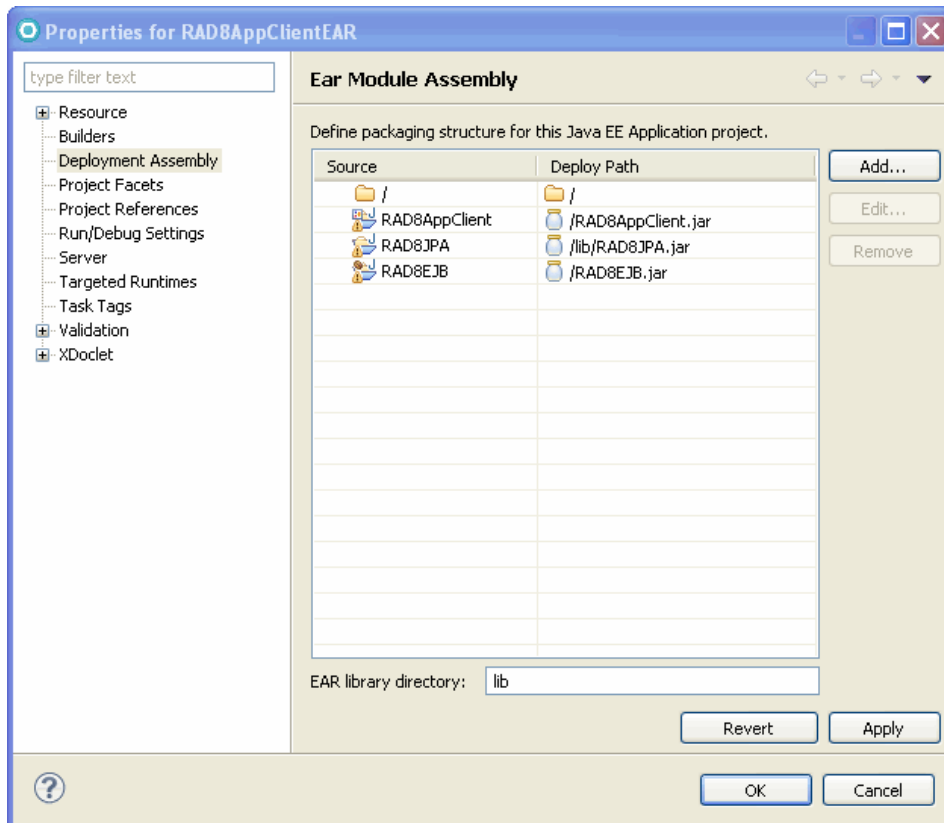


Figure 13-13 EAR Module Assembly

5. Click **OK**.
6. Right-click **RAD8AppClient** and select **Properties**.
7. In the Properties window, select **Deployment Assembly**. In the right pane, select the **Manifest Entries** tab. Click **Add**. In the Add Manifest Entries dialog, select the **RAD8EJB** project (Figure 13-14 on page 666). Click **Finish**.

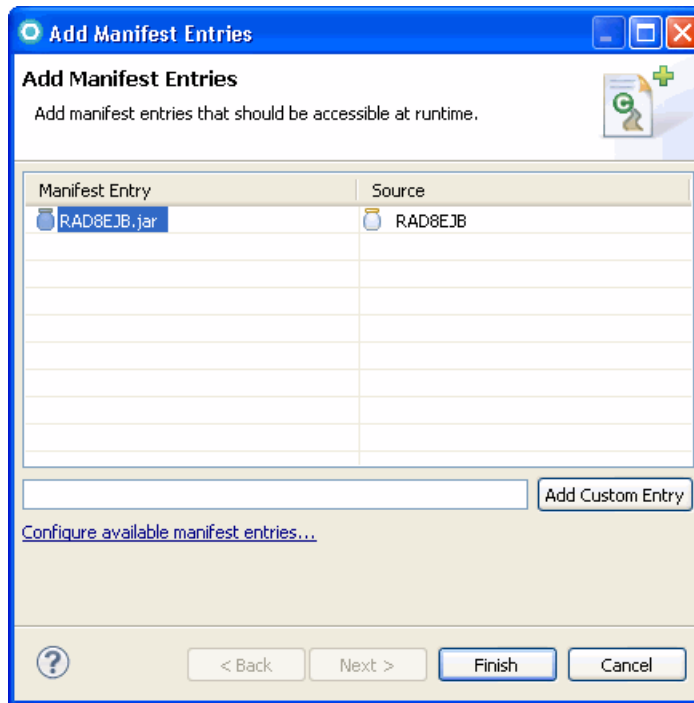


Figure 13-14 Add Manifest Entries dialog

8. Click **OK**.

13.4.3 Importing the graphical user interface and control classes

In this section, we complete the GUI for the Java EE application client. This sample uses Swing components for the user interface.

Because this chapter focuses on the aspects relating to development of Java EE application clients, we import the finished user interface and focus on implementing the code for accessing the EJB.

To import the framework classes for the Java EE application client, follow these steps:

1. In the Enterprise Explorer, right-click **RAD8AppClient** and select **New** → **Package**. Type `itso.rad8.client.ui` as Name and click **Finish**.
2. Right-click **itso.rad8.client.ui** and select **Import**.
3. In the Import window, expand **General** → **File System** and click **Next**.

4. In the Import file window, click **Browse**. Then find and select **c:\7835code\j2eeclient\appclient**.
5. In the Import: File system window (Figure 13-15), select **AccountTableModel.java** and **BankDesktop.java** and click **Finish**.

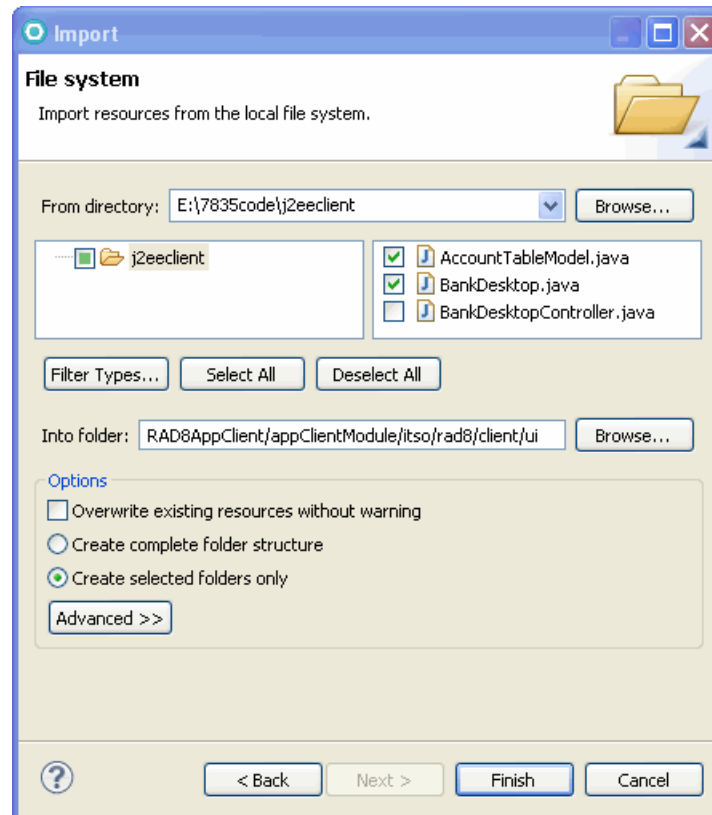


Figure 13-15 Import existing GUI classes

Two classes are imported to the RAD8AppClient project:

- ▶ **itso.rad8.client.ui.BankDesktop**: This visual class, which extends the Swing JFrame, contains the view for the Java EE application client.
- ▶ **itso.rad8.client.ui.AccountTableModel**: This implementation of the interface `javax.swing.table.AbstractTableModel` provides the relevant `TableModel` interface for a `JTable`, given an array of `Account` instances.

13.4.4 Creating the BankDesktopController class

In this section, we create the controller class for the Java EE application client. This class is also the main class for the application and contains the EJB lookup code.

To create the BankDesktopController class, follow these steps:

1. In the Enterprise Explorer, expand **RAD8AppClient**, right-click **appClientModule**, and select **New** → **Class**.
2. In the New Java Class window (Figure 13-16 on page 669), complete the following steps:
 - a. In the Package field, type `itso.rad8.client.control`.
 - b. In the Name field, type `BankDesktopController`.
 - c. For “Which method stubs would you like to create?”, select **public static void main(String[] args)** and **Constructors from superclass**.
 - d. Click **Add** next to the Interface section.
 - e. In the Implemented Interfaces Selection window, in the Choose interfaces field, type `ActionListener`, and in the Matching types list, select **ActionListener - java.awt.event**. Click **OK**.
 - f. Click **Finish**.

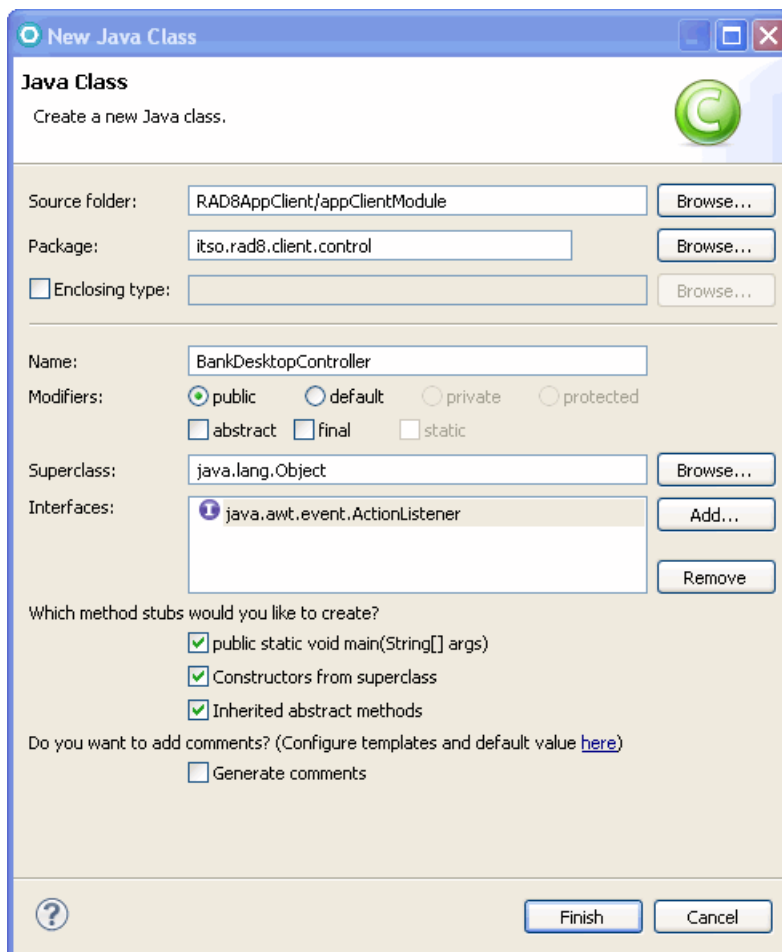


Figure 13-16 Creating the `BankDesktopController` class

13.4.5 Completing the `BankDesktopController` class

In this section, we add control logic to the `BankDesktopController` class and the code to look up customer and account information from the EJB application.

Code: You can copy the code in this section from the complete `BankDesktopController` class that is supplied in the sample code in the `c:\7835code\j2eeclient\appclient\BankDesktopController.java` file.

To complete the `BankDesktopController` class, follow these steps. To begin, `BankDesktopController.java` is open in the Java editor. The client invokes methods of the `EJBBankBean` session bean through its remote interface.

1. Inject the remote interface of the session bean:

```
public class BankDesktopController implements ActionListener {  
  
    @EJB(name="ejb/bank",beanInterface=EJBBankRemote.class)  
    static EJBBankRemote bank;
```

After every step, select **Source** → **Organize Imports** (or press `Ctrl+Shift+O`) to generate the required import statement.

2. Add two fields to the beginning of the class definition:

```
private BankDesktop desktop = null;  
private AccountTableModel accountTableModel = null;
```

3. Locate the constructor and add the new code (the new code is in bold):

```
public BankDesktopController() {  
    desktop = new BankDesktop();  
    accountTableModel = new AccountTableModel();  
    desktop.getTableAccounts().setModel(accountTableModel);  
    desktop.getBtnSearch().addActionListener(this);  
    desktop.setVisible(true);  
}
```

4. Locate the main method, add the throws clause, and add three lines:

```
public static void main(String[] args) throws Exception {  
    BankDesktopController controller = new BankDesktopController();  
    // without the next line, app client fails in IDE, works outside  
    bank.getAccounts("xxx");  
}
```

5. Locate the `actionPerformed` method stub and complete the method, as shown in Example 13-1.

Example 13-1 Complete actionPerformed method

```
public void actionPerformed(ActionEvent e) {  
    // we know that we are only listening to action events from  
    // the search button, so...  
    String ssn = desktop.getTfSSN().getText();  
    try {  
        // look up the customer  
        Customer customer = bank.getCustomer(ssn);  
        if (customer == null) throw new ITSBankException
```

```

        ("Customer not found: " +
        ssn);
        // look up the accounts
        Account[] accounts = bank.getAccounts(ssn);
        // update the user interface
        desktop.getTfTitle().setText(customer.getTitle());
        desktop.getTfFirstName().setText(customer.getFirstName());
        desktop.getTfLastName().setText(customer.getLastName());
        // store the accounts in the table model and set the model in
the GUI
        accountTableModel.setAccounts(accounts);
    } catch (ITSOBankException x) {
        // unknown customer. Report this using the output fields...
        desktop.getTfTitle().setText("(not found)");
        desktop.getTfFirstName().setText("(not found)");
        desktop.getTfLastName().setText("(not found)");
        accountTableModel.setAccounts(new Account[0]);
    }
}

```

6. Save and close BankDesktopController.

13.4.6 Creating an EJB reference and binding

An EJB 3.0 session bean has a short binding and a long binding that can be used to inject the EJB reference. The EJBBankBean session bean has the following short and long bindings for the remote interface:

```

itso.bank.service.EJBBankRemote
ejb/RAD8EJBEAR/RAD8EJB.jar/EJBBankBean#itso.bank.service.
EJBBankRemote

```

For this example, we use the long binding to invoke the EJBBankRemote interface. You must define an EJB reference in the deployment descriptor. Follow these steps to add this EJB reference with the long binding:

1. Expand **RAD8AppClient** and open (double-click) the **RAD8AppClient** descriptor editor.

Alternatively, right-click **application-client.xml** in **appClientModule/META-INF** and select **Open With** → **Application Client 6 Descriptor Editor**.

2. On the Design tab (Figure 13-17), follow these steps:

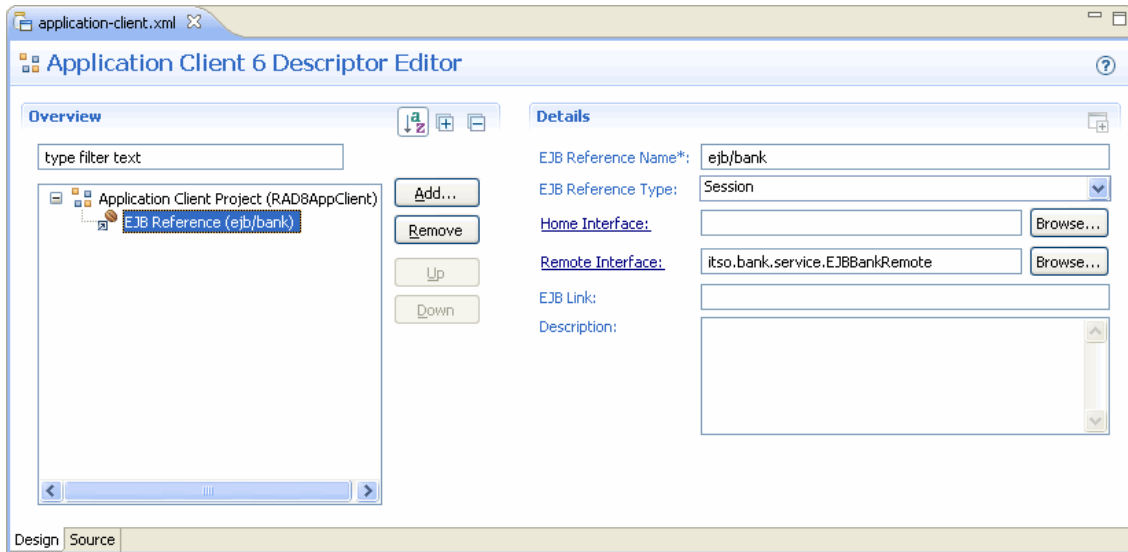


Figure 13-17 Create an EJB reference

- a. Click **Add** to add a new EJB reference.
- b. Select **EJB Reference** and click **OK**.
- c. For EJB Reference Name, type `ejb/bank`.
- d. For EJB Reference Type, select **Session**.
- e. For Remote Interface, type `itso.bank.service.EJBBankRemote`.

The Source tab shows the XML source of the EJB reference:

```
<ejb-ref>
  <ejb-ref-name>ejb/bank</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <remote>itso.bank.service.EJBBankRemote</remote>
</ejb-ref>
```

3. Save and close the `application-client.xml` file.
4. Right-click **RAD8AppClient** and select **Java EE** → **Generate WebSphere Bindings Deployment Descriptor** to create a stub of the `ibm-application-client-bnd.xml` file.

5. On the Design tab (Figure 13-18), perform these steps:
 - a. Click **Add** to add a new EJB reference.
 - b. Select **EJB Reference** and click **OK**.
 - c. For Name, type `ejb/bank`.
 - d. For Binding Name, type:


```
ejb/RAD8EJBEAR/RAD8EJB.jar/EJBBankBean#itso.bank.service.EJBBankRemote
```

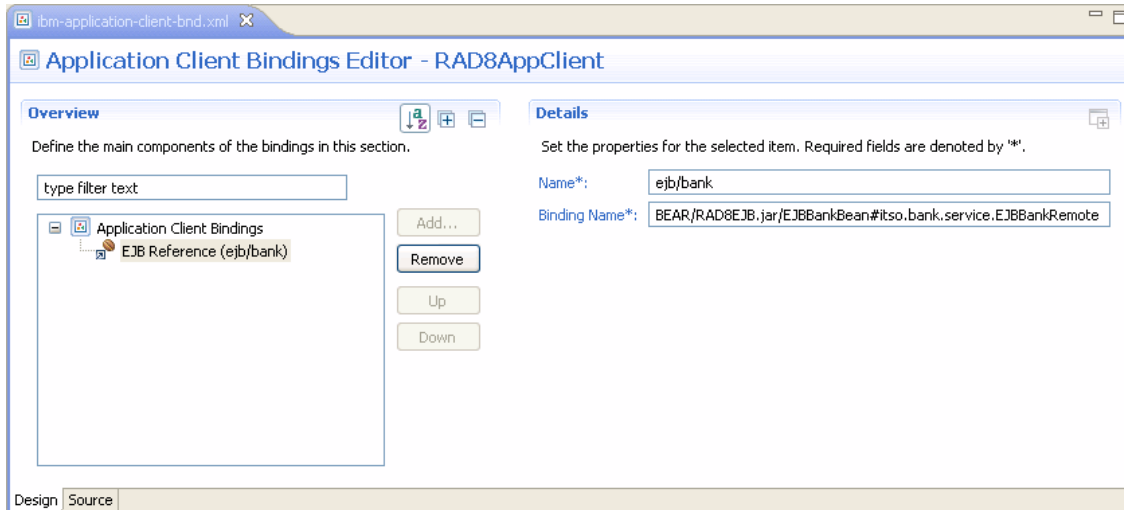


Figure 13-18 Creating an EJB binding

The Source tab shows the XML source of the EJB binding:

```
<ejb-ref name="ejb/bank"
binding-name="ejb/RAD8EJBEAR/RAD8EJB.jar/EJBBankBean#itso.bank.service.EJBBankRemote" />
```

6. Save and close the `ibm-application-client-bnd.xml` file.

The code and configuration for the ITS0 Bank Java EE application client is now complete. Now you need to register the `BankDesktopController` class as the main class for the application client.

13.4.7 Registering the `BankDesktopController` class as the main class

The `BankDesktopController` class contains the logic for the Java EE application client.

You have to register this class as the main class for the application client, so that the Java EE application client containers start the application properly:

1. Double-click **MANIFEST.MF** in **RAD8AppClient** → **appClientModule** → **META-INF**.
2. In the Manifest Editor, perform the following steps (Figure 13-19):
 - a. Click **Browse** next to the Main-Class entry.
 - b. In the Type Selection window, for the Select a class using field, start typing BankDesk. In the Matching types list, select the **BankDesktopController** and click **OK**.
 - c. In the Dependencies section of the Manifest Editor, click **Allow both** and select **RAD8EJB.jar** (Figure 13-19).

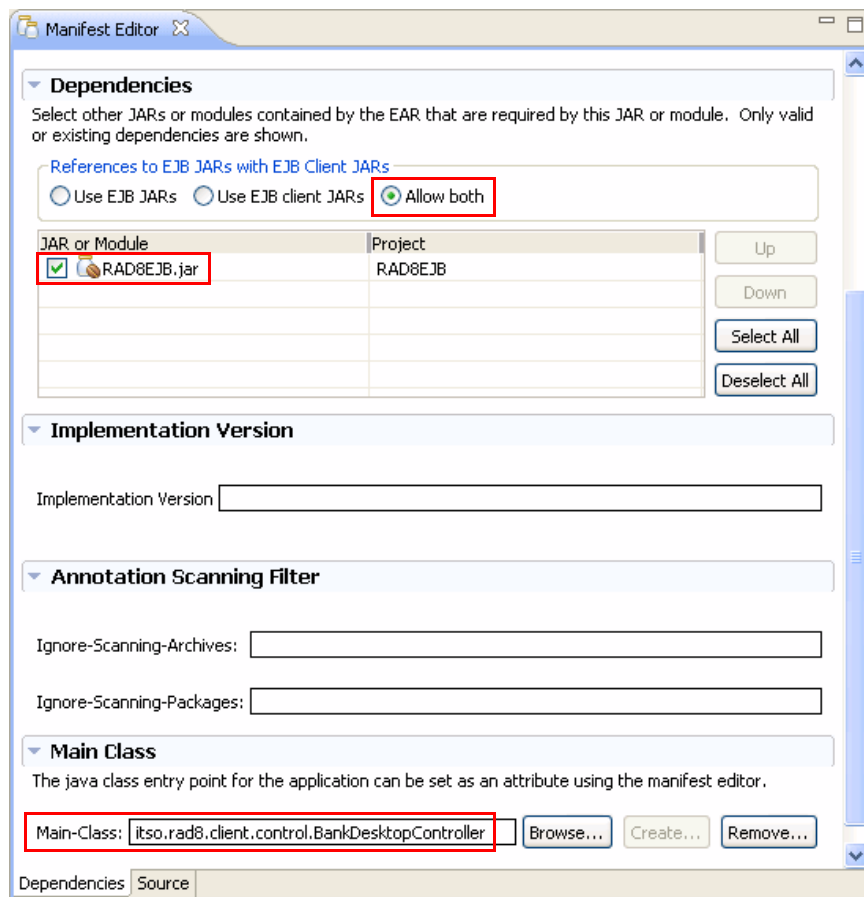


Figure 13-19 Manifest Editor

- d. Save and close the Manifest Editor.

13.5 Testing the Java EE application client

Now that the code has been updated, test the Java EE application client:

1. Make sure that the WebSphere Application Server v8.0 Beta is started.
2. Ensure that the RAD8EJBEAR enterprise application is deployed on the server. In the Servers view, right-click **WebSphere Application Server v8.0 Beta** and select **Add and Remove**. Add the RAD8EJBEAR project if it is not deployed to the server already.

Do not add RAD8AppClientEAR to the server. We run the application client outside of the server.

3. In the Enterprise Explorer, right-click **RAD8AppClient** and select **Run As** → **Run Configurations**.
4. In the Run Configurations window (Figure 13-20 on page 676), in the left pane, double-click **WebSphere Application Server v8.0 Beta Application Client**. A New_configuration entry is added and displayed in the right pane.

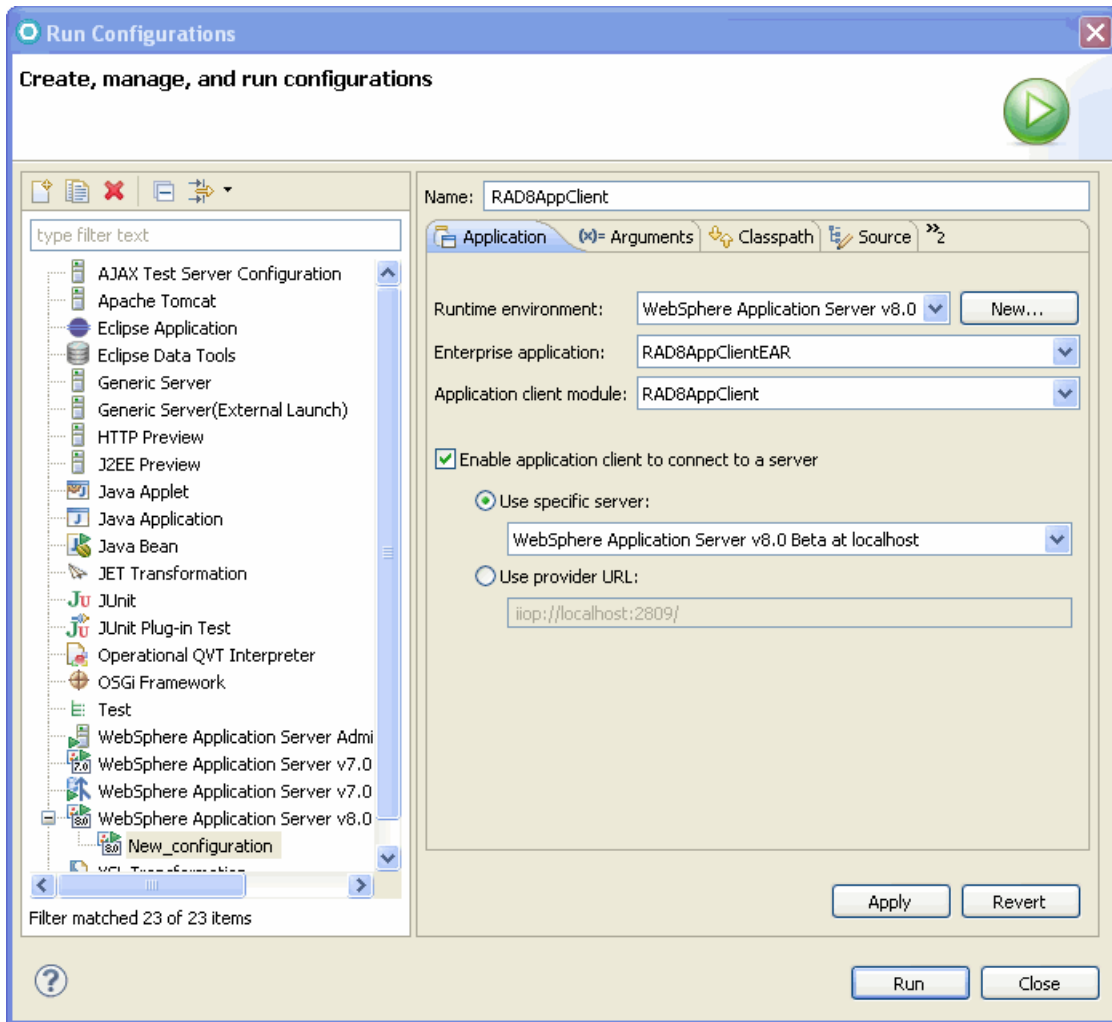


Figure 13-20 Creating a new run configuration for the application client

Follow these steps:

- a. In the Name field, type `RAD8AppClient`.
- b. For Enterprise application, select **RAD8AppClientEAR**, and for Application Client module, select **RAD8AppClient**.
- c. Select **Enable application client to connect to a server** and accept **WebSphere Application Server v8.0 Beta** as the specific server.
- d. Click **Apply** and then click **Run**.
5. If prompted with the SSL Signer Exchange Prompt, click **Yes**.

- In the Login at the Target Server window (Figure 13-21), which opens if security is enabled in the WebSphere Application Server, for both the user ID and password, type admin (the user ID configured for the server). Then click **OK**.

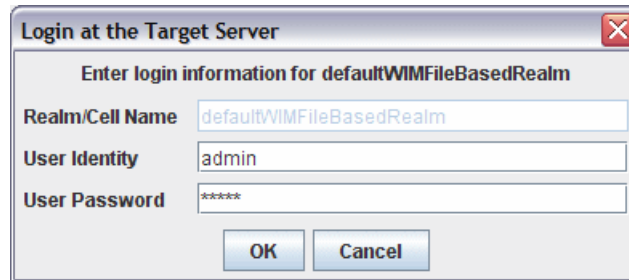


Figure 13-21 Logging in to the secure server

Several messages are displayed in the console:

```
IBM WebSphere Application Server, Release 8.0
Java EE Application Client Tool
Copyright IBM Corp., 1997-2008
.....
WSCL0013I: Initializing the Java EE Application Client Environment.
.....
WSCL0035I: Initialization of the Java EE Application Client
Environment
                has completed.
WSCL0014I: Invoking the Application Client class
                itso.rad8.client.control.BankDesktopController
```

- In the Bank Desktop window (Figure 13-22), type a customer SSN, such as 444-44-4444, and click **Search**.

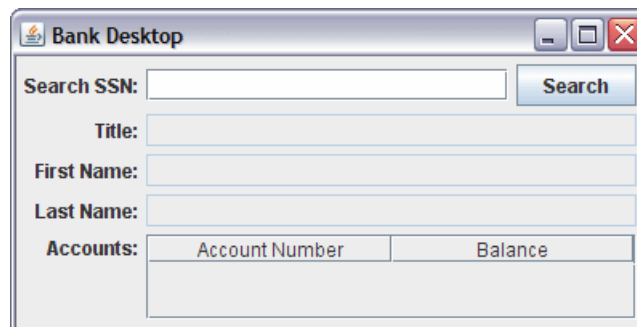
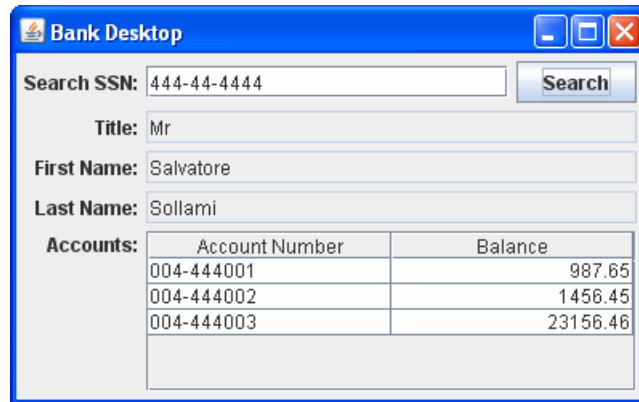


Figure 13-22 Bank Desktop window

8. The customer and the accounts are displayed (Figure 13-23).



Account Number	Balance
004-444001	987.65
004-444002	1456.45
004-444003	23156.46

Figure 13-23 Results shown in the Bank Desktop window

9. Close the client window.

Initial communication with server: In the main method of the BankDesktopController, we added the following line:

```
bank.getAccounts("xxx");
```

Without this initial communication with the server to run the getAccounts query, the application client fails when run inside Rational Application Developer:

```
Exception in thread "AWT-EventQueue-0" java.rmi.MarshalException:
CORBA MARSHAL 0x4942f896 No; nested exception is:
    org.omg.CORBA.MARSHAL: Unable to read value from underlying
bridge :
    null vmcid: IBM minor code: 896 completed: No
at com.ibm.CORBA.iiop.UtilDelegateImpl.mapSystemException
(UtilDelegateImpl.java:271)
at javax.rmi.CORBA.Util.mapSystemException(Util.java:84)
at itso.bank.service._EJBBankRemote_Stub.getAccounts
(_EJBBankRemote_Stub.java)
at itso.rad8.client.control.BankDesktopController.actionPerformed
(BankDesktopController.java:40)
```

13.6 Packaging the Java EE application client

To run the application client outside Rational Application Developer, you must package the application.

JAR file versus EAR file: Although the Java EE specification names the JAR format as the principle means for distributing Java EE application clients, the WebSphere Application Server application client container expects an enterprise application archive (EAR) file.

13.6.1 Packaging the application

To package the application client for deployment, follow these steps:

1. In the Enterprise Explorer, right-click **RAD8AppClientEAR** and select **Export** → **EAR file**.
2. In the EAR Export window (Figure 13-24), click **Browse** to select a destination, for example, `c:\7835code\deployment\RAD8AppClientEAR.ear`. Click **Finish** to generate the EAR.

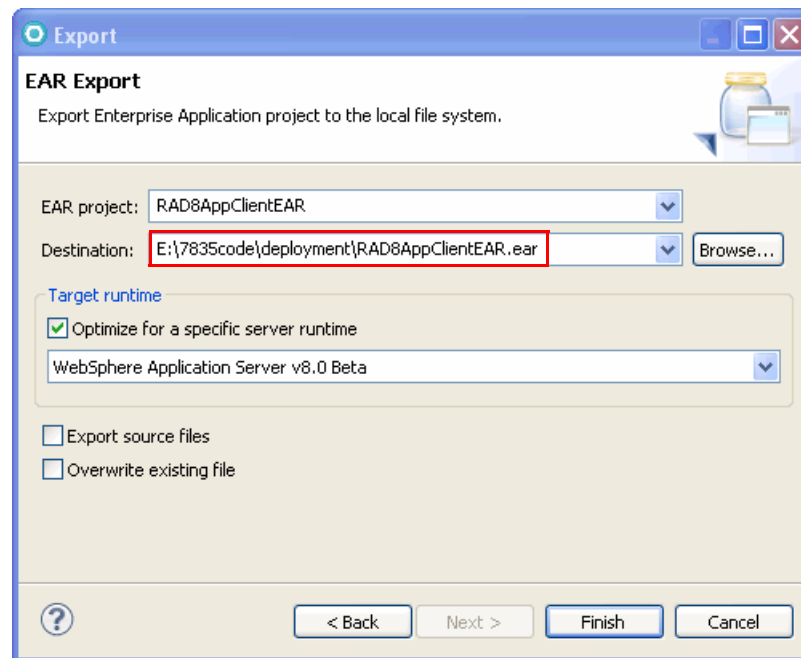


Figure 13-24 Application client export

You can now export the EAR file to a client node and run it by using the Application Client for WebSphere Application Server.

13.6.2 Running the deployed application client

You can use the **launchClient** command to run the application client outside of Rational Application Developer. For information about the command, see the following web address:

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/rccli_javacmd.html

To run the application client, follow these steps:

1. Open a command window at the \bin of WebSphere Application Server.
2. Enter the **launchClient** command:

```
launchclient c:\7835code\deployment\RAD8AppClientEAR.ear  
             -CCBootstrapHost=localhost -CCBootstrapPort=28xx
```

-CCBootstrap: The `-CCBootstrapHost` parameter specifies the host machine where the WebSphere Application Server v8.0 Beta is running (default is `localhost`). The `-CCBootstrapPort` parameter specifies the RMI port (default 2809).

You can find the RMI port by opening the server configuration (double-click the server in the Servers view) and look at server connection types.

3. After the Bank Desktop window opens, you can run the application client (see Figure 13-22 on page 677 and Figure 13-23 on page 678).



Developing web services applications

In this chapter, we introduce the concept of a service-oriented architecture (SOA). We explain how to realize this type of an architecture by using the Java Enterprise Edition (Java EE 6) web services specifications: *Java Specification Request (JSR) 224: Java API for XML-Based Web Services (JAX-WS) 2.2* and *JSR 311: Java API for RESTful Web Services 1.1 (JAX-RS)*.

We explore the features that are provided by Rational Application Developer for web services development and security. We also demonstrate how Rational Application Developer can help with testing web services and developing web services client applications.

The chapter is organized into the following sections:

- ▶ Introduction to web services
- ▶ New function in Java EE 6 for web services
- ▶ JAX-WS programming model
- ▶ Web services development approaches
- ▶ Web services tools in Rational Application Developer
- ▶ Preparing for the JAX-WS samples
- ▶ Creating bottom-up web services from a JavaBean
- ▶ Creating a synchronous web service JSP client
- ▶ Creating a web service JavaServer Faces client

- ▶ Creating a web service thin client
- ▶ Creating asynchronous web service clients
- ▶ Creating web services from an EJB
- ▶ Creating a top-down web service from a WSDL
- ▶ Creating web services with Ant tasks
- ▶ Sending binary data using MTOM
- ▶ JAX-RS programming model
- ▶ Web services security
- ▶ WS-Policy
- ▶ WS-MetadataExchange (WS-MEX)
- ▶ Security Assertion Markup Language (SAML) support
- ▶ More information

The sample code for this chapter is in the 7835code\webservice folder.

14.1 Introduction to web services

This section introduces architecture and concepts of the SOA and web services.

14.1.1 SOA

In an SOA, applications are made up of loosely coupled software services, which interact to provide all the functionality needed by the application. Each service is generally designed to be self-contained and stateless to simplify the communication that takes place between them.

There are three major roles involved in an SOA:

- ▶ Service provider
- ▶ Service broker
- ▶ Service requester

Figure 14-1 shows the interactions between these roles.

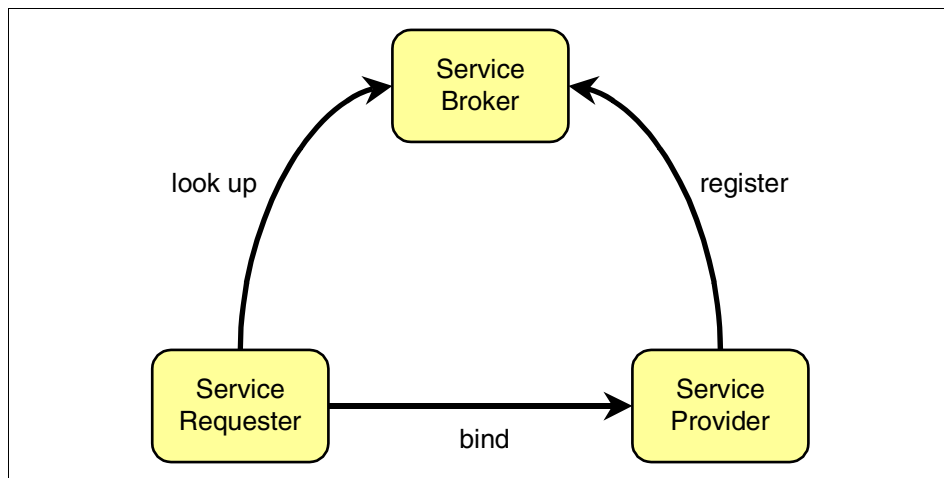


Figure 14-1 Service-oriented architecture

Service provider

The *service provider* creates a service and can publish its interface and access information to a *service broker*.

A service provider must decide which services to expose and how to expose them. Often, a trade-off exists between security and interoperability; the service provider must make technology decisions based on this trade-off. If the service provider uses a service broker, decisions must be made about how to categorize

the service, and the service must be registered with the service broker using agreed-upon protocols.

Service broker

The *service broker*, also known as the *service registry*, is responsible for making the service interface and implementation access information that is available to any potential service requester.

The service broker provides mechanisms for registering and finding services. A particular broker might be public (for example, available on the Internet) or private, only available to a limited audience (for example, on an intranet). The type and format of the information stored by a broker and the access mechanisms used is implementation-dependent.

Service requester

The *service requester*, also known as a *service client*, discovers services and then uses them as part of its operation.

A service requester uses services provided by service providers. Using an agreed-upon protocol, the requester can find the required information about services using a broker (or this information can be obtained in another way). After the service requester has the necessary details of the service, it can bind or connect to the service and invoke operations on it. The binding is usually static, but the possibility of dynamically discovering the service details from a service broker and configuring the client accordingly makes dynamic binding possible.

14.1.2 Web services as an SOA implementation

Web services provides a technology foundation for implementing an SOA. A major focus of this technology is interoperability. The functional building blocks must be accessible over standard Internet protocols. Internet protocols are independent of platforms and programming languages, which ensures that high levels of interoperability are possible.

Web services are self-contained software services that can be accessed using simple protocols over a network. They can also be described using standard mechanisms, and these descriptions can be published and located using standard registries. Web services can perform a wide variety of tasks, ranging from simple request-reply tasks to full business process interactions.

By using tools, such as Rational Application Developer, existing resources can be exposed as web services easily.

The following core technologies are used for web services:

- ▶ Extensible Markup Language (XML)
- ▶ SOAP
- ▶ Web Services Description Language (WSDL)

XML

XML is the markup language that underlies web services. XML is a generic language that can be used to describe any content in a structured way, separated from its presentation to a specific device. All elements of web services use XML extensively, including XML namespaces and XML schemas.

The specification for XML is available at the following address:

<http://www.w3.org/XML/>

SOAP

SOAP is a network, transport, and programming language-neutral protocol that allows a client to call a remote service. The message format is XML. SOAP is used for all communication between the service requester and the service provider. The format of the individual SOAP messages depends on the specific details of the service being used.

The specification for SOAP is available at the following address:

<http://www.w3.org/TR/soap/>

WSDL

WSDL is an XML-based interface and implementation description language. The service provider uses a WSDL document to specify the following items:

- ▶ The operations that a web service provides
- ▶ The parameters and data types of these operations
- ▶ The service access information

WSDL is one way to make service interface and implementation information available in a service registry. A server can use a WSDL document to deploy a web service. A service requester can use a WSDL document to work out how to access a web service (or a tool can be used for this purpose).

The specification for WSDL is available at the following address:

<http://www.w3.org/TR/wsdl/>

14.2 New function in Java EE 6 for web services

Java EE 6 includes several API specifications that provide web services support. Several of these specifications were already included in Java EE 5 and have been upgraded in Java EE 6. Several of these specifications are entirely new in Java EE 6. The most notable example is *JSR 311: JAX-RS: Java API for RESTful Web Services 1.1*.

The specifications for web services support in Java EE are available at the following web address:

<http://www.oracle.com/technetwork/java/javae6/tech/webservices-139501.html>

For information about standards related to web services supported by Rational Application Developer, see the following address:

<http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=com.ibm.webservice.doc/topics/core/cwsfpstandards.html>

This information center describes which versions of the standards are supported by WebSphere Application Server v8.0 Beta, V7.0, and V6.1 with or without the Feature Pack for Web Services.

14.2.1 JSR 224: Java API for XML-Based Web Services (JAX-WS) 2.2

The Java API for XML-Based Web Services (JAX-WS) is a programming model that simplifies application development through the support of a standard, annotation-based model to develop web services applications and clients.

The JAX-WS programming standard aligns itself with the document-centric messaging model and replaces the remote procedure call programming model defined by the Java API for XML-based RPC (JAX-RPC) specification. Although Rational Application Developer still supports the JAX-RPC programming model and applications, JAX-RPC has limitations and does not support many current document-centric services. JAX-RPC will not be described further in this book.

Table 14-1 on page 687 shows which WebSphere Application Server versions support JAX-WS 2.0, 2.1, and 2.2.

Table 14-1 WebSphere Application Server support for JAX-WS versions

Java EE version	JAX-WS version	WebSphere Application Server version
Java EE 5	JAX-WS 2.0	6.1 with Feature Pack for Web Services 7.0 8.0
Java EE 5	JAX-WS 2.1	7.0 8.0
Java EE 6	JAX-WS 2.2	8.0

JAX-WS 2.1 introduces support for the WS-Addressing in a standardized API. Using this function, you can create, transmit, and use endpoint references to target a web service endpoint. You can use this API to specify the action uniform resource identifiers (URIs) that are associated with the WSDL operations of your Web service.

JAX-WS 2.1 introduces the concept of features as a way to programmatically control specific functions and behaviors. Three standard features are available: the AddressingFeature for WS-Addressing, the MTOMFeature when optimizing the transmission of binary attachments, and the RespectBindingFeature for wsdl:binding extensions. JAX-WS 2.1 requires Java Architecture for XML Binding (JAXB) Version 2.1 for data binding.

For more information about the new features of JAX-WS 2.1, refer to the WebSphere Application Server 7.0 Information Center:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/cwbs_jaxws.html

WebSphere Application Server Version 8.0 supports the *JSR 109: JAX-WS Version 2.2 and Web Services for Java EE Version 1.3 specifications*.

The JAX-WS 2.2 specification supersedes and includes functions within the JAX-WS 2.1 specification. JAX-WS 2.2 adds client-side support for using WebServiceFeature-related annotations, such as @MTOM, @Addressing, and the @RespectBinding annotations. JAX-WS 2.1 had previously added support for these annotations on the server.

For more information about the new features of JAX-WS 2.2, refer to this website:

<http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/com.ibm.webservice.doc/topics/core/cjaxws.html>

In Rational Application Developer, you can choose which version of JAX-WS code to produce when generating web services *top-down* (from an existing WSDL file) and when generating web services clients. You can find the corresponding options by selecting **Windows** → **Preferences** → **Web Services** → **WebSphere** → **JAX-WS Code Generation**:

- ▶ **Top Down** → **Version of JAX-WS code to be generated**
- ▶ **Client** → **Version of JAX-WS code to be generated**

These default options can be further overridden in the Web Services code generation wizard.

14.2.2 JSR 222: Java Architecture for XML Binding (JAXB) 2.2

Java Architecture for XML Binding (JAXB) is a Java technology that provides an easy and convenient way to map Java classes and XML schema for the simplified development of web services. JAXB uses the flexibility of platform-neutral XML data in Java applications to bind XML schema to Java applications without requiring extensive knowledge of XML programming.

JAXB is the default data binding technology that the JAX-WS tooling uses and is the default implementation within this product. You can develop JAXB objects for use within JAX-WS applications.

JAX-WS tooling relies on JAXB tooling for default data binding for two-way mappings between Java objects and XML documents. JAXB data binding replaces the data binding described by the JAX-RPC specification.

WebSphere Application Server V7.0 supports the JAXB 2.1 specification. JAX-WS 2.1 requires JAXB 2.1 for data binding. JAXB 2.1 provides enhancements, such as improved compilation support and support for the @XML annotation, and full schema 1.0 support.

WebSphere Application Server v8.0 Beta supports the JAXB 2.2 specification. JAXB 2.2 provides minor enhancements to its annotations for improved schema generation and better integration with JAX-WS. JAX-WS 2.2 requires Java Architecture for XML Binding (JAXB) Version 2.2 for data binding.

14.2.3 JSR 109: Implementing Enterprise Web Services

Implementing Enterprise Web Services: JSR 109 defines the programming model and runtime architecture to deploy and look up web services in the Java EE environment, more specifically, in the web, Enterprise JavaBeans (EJB), and client application containers. One of the major goals of JSR 109 is to ensure that vendors' implementations interoperate.

WebSphere Application Server V8 Beta introduces support for *Web Services for Java EE (JSR 109) Version 1.3 specification*. The Web Services for Java EE 1.3 specification introduces support for WebServiceFeature-related annotations, as well as support for using deployment descriptor elements to configure these features on both the client and server.

14.2.4 Related web services standards

Next we describe the related web services specifications.

JSR 67: SOAP with Attachments API for Java (SAAJ)

The SOAP with Attachments API for Java (SAAJ) interface is used for SOAP messaging that provides a standard way to send XML documents over the Internet from a Java programming model. SAAJ is used to manipulate the SOAP message to the appropriate context as it traverses through the runtime environment.

JSR 173: Streaming API for XML (StAX)

Streaming API for XML (StAX) is a streaming Java-based, event-driven, pull-parsing API for reading and writing XML documents. With StAX, you can create bidirectional XML parsers that are fast, relatively easy to program, and have a light memory footprint.

JSR 181: Web Services Metadata for the Java Platform

Web Services Metadata for the Java Platform defines an annotated Java format that uses *JSR 175: Metadata Facility for the Java Programming Language* to enable the easy definition of Java web services in a Java EE container.

Web Services Interoperability Organization

In an effort to improve the interoperability of web services, the Web Services Interoperability Organization (known as WS-I) was formed. WS-I produces a specification known as the *WS-I Basic Profile*. This specification describes the technology choices that maximize interoperability between web services and clients running on separate platforms, using separate runtime systems, and written in multiple languages.

The WS-I Basic Profile is available at the following address:

<http://ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile>

Web Services Security

The WS-Security specification describes extensions to SOAP that allow for the quality of protection of SOAP messages, including message authentication,

message integrity, and message confidentiality. The specified mechanisms can be used to accommodate a wide variety of security models and encryption technologies. It also provides a general-purpose mechanism for associating security tokens with message content. For additional information, see the following web address:

http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

14.3 JAX-WS programming model

JAX-WS is the strategic programming model for developing web services and is a required part of the Java EE 5 and Java EE 6 platforms. JAX-WS simplifies application development through the support of a standard, annotation-based model to develop web service applications and clients. The JAX-WS programming standard strategically aligns itself with the current industry trend toward a more document-centric messaging model.

Implementing the JAX-WS programming standard provides the enhancements described in the following sections for developing web services and clients.

14.3.1 Better platform independence for Java applications

Using JAX-WS APIs and developing web services and clients are simplified with better platform independence for Java applications. JAX-WS takes advantage of dynamic proxies whereas JAX-RPC uses generated stubs. The dynamic proxy client invokes a web service that is based on a service endpoint interface (SEI) that is generated or provided. The dynamic proxy client is similar to the stub client in the JAX-RPC programming model. Although the JAX-WS dynamic proxy client and the JAX-RPC stub client are both based on the SEI that is generated from a WSDL file, note the following major differences:

- ▶ The dynamic proxy client is dynamically generated at run time using the Java 5 dynamic proxy functionality. The JAX-RPC-based stub client is a non-portable Java file that is generated by tooling.
- ▶ Unlike the JAX-RPC stub clients, the dynamic proxy client does not require you to regenerate a stub prior to running the client on an application server for a separate vendor, because the generated interface does not require the specific vendor information.

14.3.2 Annotations

JAX-WS introduces support for annotating Java classes with metadata to indicate that the Java class is a web service. JAX-WS supports the use of annotations based on the *JSR 175: Metadata Facility for the Java Programming Language specification*, the *JSR 181: Web Services Metadata for the Java Platform specification*, and annotations that are defined by the *JAX-WS 2.0/2.1/2.2 specification*. Using annotations in the Java source and Java class simplifies the development of web services by defining part of the additional information that is typically obtained from deployment descriptor files, WSDL files, or mapping metadata from XML and WSDL files into the source artifacts.

For example, you can embed a simple `@WebService` annotation in the Java source to expose the bean as a web service (Example 14-1).

Example 14-1 JAX-WS annotation

```
@WebService
public class BankBean {
    public String getCustomerFullName(String ssn) { ... }
}
```

The `@WebService` annotation tells the server runtime environment to expose all public methods on that bean as a web service. Additional levels of granularity can be controlled by adding additional annotations on individual methods or parameters. The use of annotations makes it much easier to expose Java artifacts as web services. In addition, as artifacts are created from using part of the top-down mapping tools starting from a WSDL file, annotations are included within the source and Java classes as a way of capturing the metadata along with the source files.

14.3.3 Invoking web services asynchronously

With JAX-WS, web services can be called both synchronously and asynchronously. JAX-WS adds support for both a polling mechanism and callback mechanism when calling web services asynchronously. By using a polling model, a client can issue a request and get a response object back, which is polled to determine whether the server has responded. When the server responds, the actual response is retrieved. With the polling model, the client can continue to process other work without waiting for a response to return.

With the callback model, the client provides a callback handler to accept and process the inbound response object. Both the polling and callback models enable the client to focus on continuing to process work while providing for a more dynamic and efficient model to invoke web services.

For example, a web service interface has methods for both synchronous and asynchronous requests (Example 14-2). Asynchronous requests are identified in bold.

Example 14-2 Asynchronous methods in the web service interface

```
@WebService
public interface CreditRatingService {
    // sync operation
    Score getCreditScore(Customer customer);
    // async operation with polling
    Response<Score> getCreditScoreAsync(Customer customer);
    // async operation with callback
    Future<?> getCreditScoreAsync(Customer customer,
                                   AsyncHandler<Score> handler);
}
```

The asynchronous invocation that uses the callback mechanism requires an additional input by the client programmer. The *callback handler* is an object that contains the application code that is executed when an asynchronous response is received. Example 14-3 shows an asynchronous callback handler.

Example 14-3 Asynchronous callback handler

```
CreditRatingService svc = ...;

Future<?> invocation = svc.getCreditScoreAsync(customerFred,
    new AsyncHandler<Score>() {
        public void handleResponse(Response<Score> response) {
            Score score = response.get();
            // do work here...
        }
    }
);
```

Example 14-4 shows an asynchronous polling client.

Example 14-4 Asynchronous polling

```
CreditRatingService svc = ...;
Response<Score> response = svc.getCreditScoreAsync(customerFred);

while (!response.isDone()) {
    // do something while we wait
}
```



```
// no cast needed, thanks to generics
Score score = response.get();
```

14.3.4 Dynamic and static clients

The dynamic client programming API for JAX-WS is called the *dispatch client* (`javax.xml.ws.Dispatch`). The dispatch client is an XML messaging-oriented client. The data is sent in either PAYLOAD or MESSAGE mode:

- ▶ **PAYLOAD:** When using the PAYLOAD mode, the dispatch client is only responsible for providing the contents of the `<soap:Body>` element and JAX-WS adds the `<soap:Envelope>` and `<soap:Header>` elements.
- ▶ **MESSAGE:** When using the MESSAGE mode, the dispatch client is responsible for providing the entire SOAP envelope including the `<soap:Envelope>`, `<soap:Header>`, and `<soap:Body>` elements and JAX-WS does not add anything additional to the message. The dispatch client supports asynchronous invocations using a callback or polling mechanism.

The static client programming model for JAX-WS is called the *proxy client*. The proxy client invokes a web service based on an SEI that is generated or provided.

14.3.5 Message Transmission Optimization Mechanism support

With JAX-WS, you can send binary attachments, such as images or files, along with web services requests. JAX-WS adds support for optimized transmission of binary data as specified by Message Transmission Optimization Mechanism (MTOM).

14.3.6 Multiple payload structures

JAX-WS exposes the XML Source, SAAJ 1.3, and JAXB 2.2 binding technologies to the user.

With XML Source, a user can pass a `javax.xml.transform.Source` to the run time, which represents the data in a source object to be passed to the run time. SAAJ 1.3 now has the ability to pass an entire SOAP document across the interface, rather than only the payload. This action is done by the client passing the SAAJ `SOAPMessage` object across the interface. JAX-WS uses the JAXB 2.2 support as the data binding technology of choice between Java and XML.

14.3.7 SOAP 1.2 support

Support for SOAP 1.2 was added to JAX-WS 2.0. JAX-WS supports both SOAP 1.1 and SOAP 1.2. SOAP 1.2 provides a more specific definition of the SOAP processing model, which removes many of the ambiguities that sometimes led to interoperability problems in the absence of the WS-I profiles. SOAP 1.2 reduces the chances of interoperability issues with SOAP 1.1 implementations between separate vendors. It is not interoperable with earlier versions.

14.4 Web services development approaches

You can follow two general approaches to web service development:

- ▶ In the *top-down approach*, a web service is based on the web service interface and XML types, defined in WSDL and XML Schema Definition (XSD) files. You first design the implementation of the web service by creating a WSDL file using the WSDL editor. You can then use the Web Service wizard to create the web service and skeleton Java classes to which you can add the required code. You then modify the skeleton implementation to interface with the business logic.

The top-down approach provides more control over the web service interface and the XML types used. Use this approach for developing new web services.

- ▶ In the *bottom-up approach*, a web service is created based on the existing business logic in JavaBeans or EJB. A WSDL file is generated to describe the resulting web service interface.

The bottom-up pattern is often used for exposing existing function as a web service. It might be faster, and no XSD or WSDL design skills are needed. However, if complex objects (for example, Java collection types) are used, the resulting WSDL might be difficult to understand and less interoperable.

14.5 Web services tools in Rational Application Developer

Rational Application Developer provides tools to create web services from existing Java and other resources or from WSDL files. Rational Application Developer also provides tools for web services client development and for testing web services.

14.5.1 Creating a web service from existing resources

Rational Application Developer provides wizards for exposing a variety of resources as web services. You can use the following resources to build a web service:

- ▶ **JavaBean:** The Web Service wizard assists you in creating a new web service from a simple Java class, configures it for deployment, and deploys the web service to a server. The server can be the WebSphere Application Server V6.1, V7.0, or v8.0 Beta that is included with Rational Application Developer or another application server.
- ▶ **EJB:** The Web Service wizard assists you in creating a new web service from a stateless session EJB, configuring it for deployment, and deploying the web service to a server.

14.5.2 Creating a skeleton web service

Rational Application Developer provides the functionality to create web services from a description in a WSDL or Web Services Inspection Language (WSIL) file:

- ▶ **JavaBean from WSDL:** The web services tools assist you in creating a skeleton JavaBean from an existing WSDL document. The skeleton bean contains a set of methods that correspond to the operations described in the WSDL document. When the bean is created, each method has a trivial implementation that you replace by editing the bean.
- ▶ **EJB from WSDL:** The web services tools support the generation of a skeleton EJB from an existing WSDL file. Apart from the type of component produced, the process is similar to that for JavaBeans.

14.5.3 Client development

To assist in the development of web service clients, Rational Application Developer provides the following features:

- ▶ **Java client proxy from WSDL:** The Web Service client wizard assists you in generating a proxy JavaBean. This proxy can be used within a client application to greatly simplify the client programming required to access a web service.
- ▶ **Sample web application from WSDL:** Rational Application Developer can generate a sample web application, which includes the proxy classes described before, and sample JavaServer Pages (JSP) that use the proxy classes.

- ▶ **Web Service Discovery Dialog:** On this window, you can discover a web service that exists online or in your workspace, create a proxy for the web service, and then place the methods of the proxy into a Faces JSP file.

14.5.4 Testing tools for web services

To allow developers to test web services, Rational Application Developer provides a range of features:

- ▶ **WebSphere Application Server v8.0 Beta, V7.0, and V6.1 test environment:** These servers are included with Rational Application Developer as test servers and can be used to host web services. This feature provides a range of web services run times, including an implementation of the J2EE specification standards.
- ▶ **Generic service client:** The generic service client can invoke calls to any service that uses an HTTP, a Java Message Service (JMS), or WebSphere MQ transport and can view the message returned by the service.
- ▶ **Sample JSP application:** The web application mentioned before can be used to test web services and the generated proxy it uses.
- ▶ **Web Services Explorer:** This simple test environment can be used to test any web service, based only on the WSDL file for the service. The service can be running on a local test server or anywhere else in the network. The Web Services Explorer is a JSP web application that is hosted on the Apache Tomcat servlet engine in Eclipse. The Web Services Explorer uses the WSDL to render a SOAP request. It does not involve data marshalling and unmarshalling. The return parameter is stripped out, and the values are displayed in a predefined format.
- ▶ **Universal Test Client:** The Universal Test Client (UTC) is a powerful and flexible test application that is normally used for testing EJB. Its flexibility makes it possible to test ordinary Java classes, so it can be used to test the generated proxy classes created to simplify client development.
- ▶ **TCP/IP Monitor:** The TCP/IP Monitor works similarly to a proxy server, passing TCP/IP requests to another server and directing the returned responses back to the originating client. The TCP/IP messages that are exchanged are displayed in a special view within Rational Application Developer.

14.6 Preparing for the JAX-WS samples

To prepare for this sample, we import sample code, which is a simple web application that includes Java classes and an EJB.

14.6.1 Importing the sample

In this section, prepare the environment for the JAX-WS web services application samples:

1. In the Java EE perspective, select **File** → **Import**.
2. Select **General** → **Existing Projects into Workspace**.
3. In the Import Projects window, select **Select archive file**.
4. Click **Browse**. Navigate to the `c:\7835code\webservices` folder and select the **RAD8WebServiceStart.zip** file. Click **Open**.
5. Click **Select All** and click **Finish**.

After the build, no warning or error messages are displayed in the workspace.

Sample projects

The sample application that we use for creating the web service consists of the following projects:

- ▶ **RAD8WebServiceUtility** project: This project is a simple banking model with `BankMemory`, `Customer`, and `Account` beans. It is a simplified version of the `RAD8Java` project that is used in Chapter 7, “Developing Java applications” on page 229.
- ▶ **RAD8WebServiceWeb** project: This project contains the `SimpleBankBean`, a `JavaBean` with a few methods that retrieve data from the `MemoryBank`, a search HTML page, and a resulting JSP. We use annotations to generate web services for this project.
- ▶ **RAD8WebServiceWeb2** project: This project contains the same code as the `RAD75WebServiceWeb` project. We use the Web Service wizard to generate web services for this project.
- ▶ **RAD8WebServiceEJB** project: This project contains the `SimpleBankFacade` session EJB with a few methods that retrieve data from the `MemoryBank`.
- ▶ **RAD8WebServiceEAR** project: This project is the enterprise application that contains the other four projects.

14.6.2 Testing the application

To start and test the application, follow these steps:

1. In the Servers view, start **WebSphere Application Server v8.0 Beta**.
2. Right-click the server and select **Add and remove projects**.

3. In the Add and Remove Projects window, select **RAD8WebServiceEAR**, click **Add**, and then click **Finish**.
4. Expand **RAD8WebServiceWeb** → **WebContent**, right-click **search.html**, and select **Run As** → **Run on Server**.
5. Select **Choose an existing server** and select the **v8.0** server to run the application. Then click **Finish**.
6. When the search page opens in a web browser, in the Social Security number field, enter an appropriate value, for example, 111-11-1111, and click **Search**. If everything works correctly, you can see the customer's full name, first account, and its balance, which have been read from the memory data.
7. Test the stateless session EJB, `SimpleBankFacade`, by using the Universal Test Client (UTC). See 12.3.1, "Testing with the Universal Test Client" on page 624, for more information about using the UTC. The following methods are valid:
 - `getCustomerFullName(ssn)`: Retrieves the full name (use 111-11-1111)
 - `getNumAccounts(ssn)`: Retrieves the number of accounts
 - `getAccountId(ssn, int)`: Retrieves the account ID by index (0,1,2,...)
 - `getAccountBalance(accountId)`: Retrieves the balance

We now have resources in preparation for the web services sample, including a JavaBean in the `RAD8WebServiceWeb` project and a session EJB in the `RAD8WebServiceEJB` project. We use these resources as a base for developing and testing the web services examples.

14.7 Creating bottom-up web services from a JavaBean

In this section, we create a web service from an existing Java class using the bottom-up approach. The imported application contains a Java class called `SimpleBankBean`, which has various methods to get customer and account information from the bank. We can either use the Web Service wizard to generate the web service or use the annotations directly. The Web Service wizard does not inject annotations to the delegate class derived from the JavaBean. Therefore, these two approaches are essentially the same.

14.7.1 Creating a web service using annotations

The JAX-WS programming standard relies on the use of annotations to specify metadata that is associated with web service implementations. The standard also relies on annotations to simplify the development of web services. The

JAX-WS standard supports the use of annotations that are based on several JSRs:

- ▶ *A Metadata Facility for the Java Programming Language (JSR 175)*
- ▶ *Web Services Metadata for the Java Platform (JSR 181)*
- ▶ *Java API for XML-Based Web Services (JAX-WS) 2.2 (JSR 224)*
- ▶ *Common Annotations for the Java Platform (JSR 250)*
- ▶ *Java Architecture for XML Binding (JAXB) (JSR 222)*

Using annotations from the JSR 181 standard, we can annotate a service implementation class or a service interface. Then we can generate a web service with a wizard or by publishing the application to a server. Using annotations within both Java source code and Java classes simplifies web service development. Using annotations in this way defines additional information that is typically obtained from deployment descriptor files, WSDL files, or mapping metadata from XML and WSDL into source artifacts.

In this section, we create a bottom-up web service from a JavaBean by using annotations. The web services are generated by publishing the application to a server. No wizard is required in this example.

Annotating a JavaBean

We can annotate types, methods, fields, and parameters in the JavaBean to specify a web service. To annotate the JavaBean, follow these steps:

1. In the RAD75WebServiceWeb project, open the **SimpleBankBean** (in `itso.rad8.bank.model.simple`).
2. Before the class declaration, type `@W` and press `Ctrl+Spacebar` for content assist. Scroll down to the bottom and select **WebService(Web Service Template) - javax.jws** (Figure 14-2).

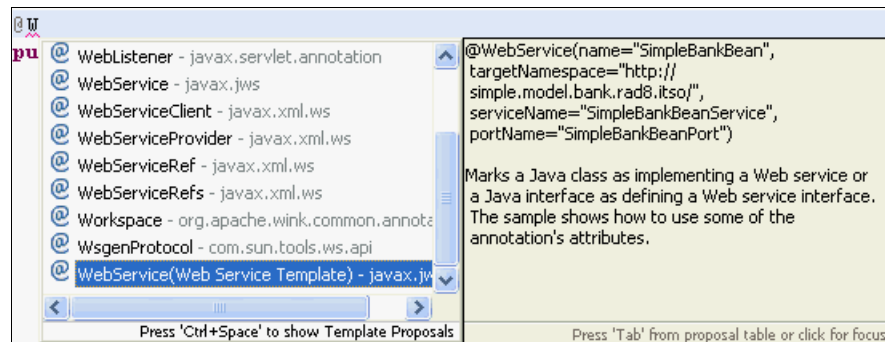


Figure 14-2 Content assist for `WebService` annotation

The annotation template is added to the Java class (Example 14-5 on page 700).

Example 14-5 Web service annotation template

```
@WebService(name="SimpleBankBean",
    targetNamespace="http://simple.model.bank.rad8.itso/",
    serviceName="SimpleBankBeanService",
    portName="SimpleBankBeanPort")
```

The `@WebService` annotation marks a Java class as implementing a web service:

- The *name* attribute is used as the name of the `wsdl:portType` when mapped to WSDL 1.1.
 - The *targetNamespace* attribute is the XML namespace used for the WSDL and XML elements generated from this web service.
 - The *serviceName* attribute specifies the service name of the web service: `wsdl:service`.
 - The *portName* attribute is the name of the endpoint port.
3. Change the web service name, service name, and port name, as listed in Example 14-6.

Example 14-6 Annotating a JavaBean web service

```
@WebService(name="Bank",
    targetNamespace="http://simple.model.bank.rad8.itso/",
    serviceName="BankService", portName="BankPort")
```

4. Before the `getCustomerFullName` method, type `@W` and press `Ctrl+Spacebar` for content assist. Scroll down to the bottom and select **WebMethod(Web Service Template) - javax.jws** (Figure 14-3).

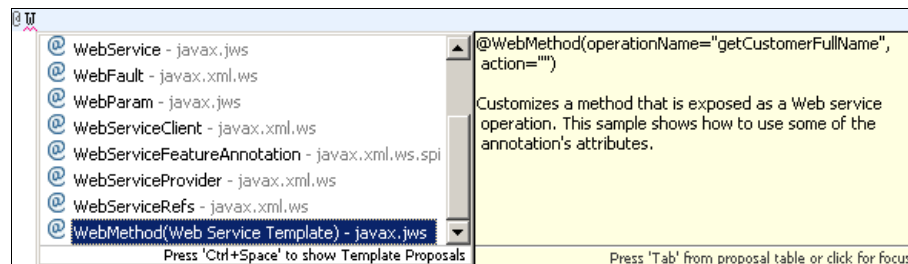


Figure 14-3 Annotate method

The `@WebMethod` annotation is added to the method (Example 14-7).

Example 14-7 WebMethod template

```
@WebMethod(operationName="getCustomerFullName", action="")
```

The `@WebMethod` annotation identifies the individual methods of the Java class that are exposed externally as web service operations. In this example, we expose the `getCustomerFullName` method as a web service operation. The `operationName` is the name of the `wsdl:operation` matching this method. The `action` determines the value of the soap action for this operation.

5. Change the `operationName` and `action` (Example 14-8).

Example 14-8 @WebMethod annotation

```
@WebMethod(operationName="RetrieveCustomerName",  
            action="urn:getCustomerFullName")
```

6. Annotate the method input and output (Example 14-9).

Example 14-9 Annotate the method input and output

```
@WebMethod(operationName="RetrieveCustomerName",  
            action="urn:getCustomerFullName")  
@WebResult(name="CustomerFullName")  
public String getCustomerFullName(@WebParam(name="ssn")String ssn)  
            throws CustomerDoesNotExistException
```

The `@WebParam` and `@WebResult` annotations customize the mapping of the method parameters and results to message parts and XML elements.

7. Select **Source** → **Organize Imports** (or press `Ctrl+Shift+O`) to resolve the imports.

Validating web service annotations

When developing web services, you can benefit from two levels of validation. The first level involves validating syntax and Java-based default values. This level of validation is performed by the Eclipse Java development tools (JDT). The second level of validation involves the implicit default checking and verification of WSDL contracts. This second level is performed by a JAX-WS annotation processor.

When you enable the annotation processor, warning and error messages for annotations are displayed similarly to Java errors. You can work with these messages in various workbench locations, such as the Problems view.

For instance, after annotating one method as `@WebMethod`, you see a QuickFix icon with the warning that is reported in Example 14-10 on page 702.

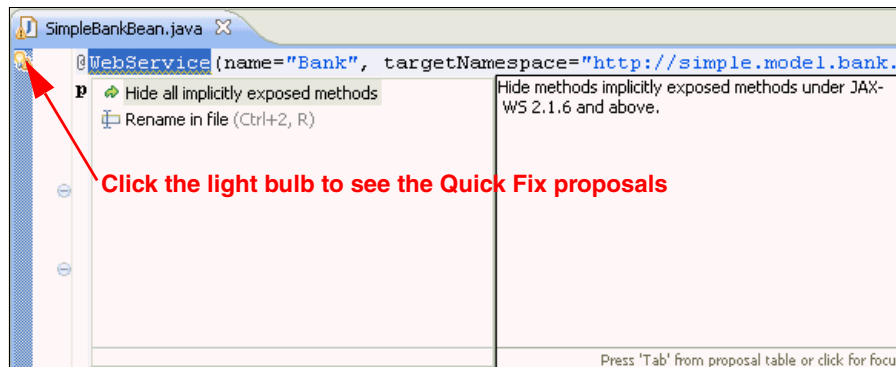
Example 14-10 Warning after adding @WebMethod in front of one method

JAX-WS 2.1.6, 3.3: The following methods will be implicitly exposed as web methods: [BigDecimal getAccountBalance(String accountId), String getAccountId(String customerId, int account), int getNumAccounts(String customerId)]

If you click the light bulb icon corresponding to this QuickFix, you see two proposed solutions, as shown in Figure 14-4:

- ▶ Hide all implicitly exposed methods
- ▶ Rename in file

Select the first proposal: All mentioned methods are annotated with @WebMethod(exclude=true).



```

@WebService(name="!Bank",
targetNamespace="simple.model.bank.rad8.itso/",
serviceName="BankService", portName="BankPort")
public class SimpleBankBean implements Serializable {
    private static final long serialVersionUID = -637536840546155853L;
    public SimpleBankBean() {
    }
}
@WebMethod(operationName="!RetrieveCustomerName",
action="urn:getCustomerFullName")
@WebResult(name="CustomerFullName")
@Oneway
    public String getCustomerFullName(@WebParam(name="ssn")String ssn)
        throws CustomerDoesNotExistException {

```

Example 14-12 JAX-WS annotation processor validation results

JSR-181, 4.3.1: Oneway methods cannot return a value
 JSR-181, 4.3.1: Oneway methods cannot throw checked exceptions
 name must be a valid nmToken
 operationName must be a valid nmToken
 targetNamespace must be a valid URI

Creating a web service from an annotated JavaBean by publishing to the server

After annotating a JavaBean, you can generate a web service application by publishing the application project of the bean directly to a server. When the web service is generated, no WSDL file is created in your project.

Perform these steps to create a web service from an annotated JavaBean:

1. In the Servers view, start WebSphere Application Server v8.0 Beta (if it is not running).
2. Publish the web service project on the server. Depending on the server configuration, this step happens either automatically or manually (by right-clicking the server and selecting **Publish**).

Testing the JAX-WS web service: The Generic Service Client

To test the web service by using the Generic Service Client, follow these steps:

1. Make sure that the project is already published to the server.
2. Switch to the **Services** view that is under the Enterprise Explorer.

- Expand the **JAX-WS** folder, right-click **RAD8WebServiceWeb: {http://simple.model.bank.rad8bank.itso/}BankService**, and select **Test with Generic Service Client** (Figure 14-5).

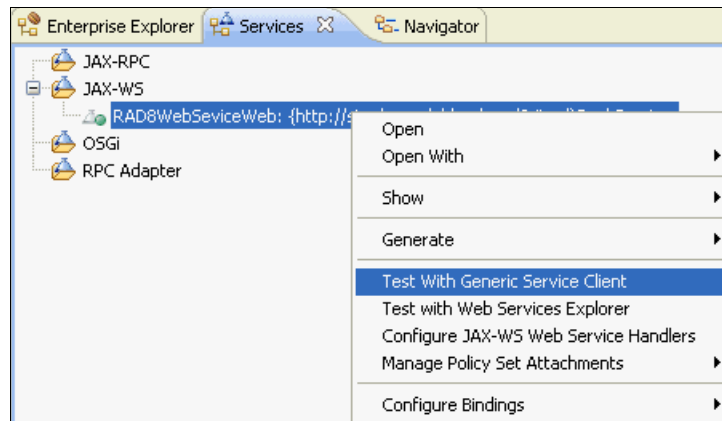


Figure 14-5 Test with Generic Service Client

The Generic Service Client opens, as shown in Figure 14-6 on page 705.

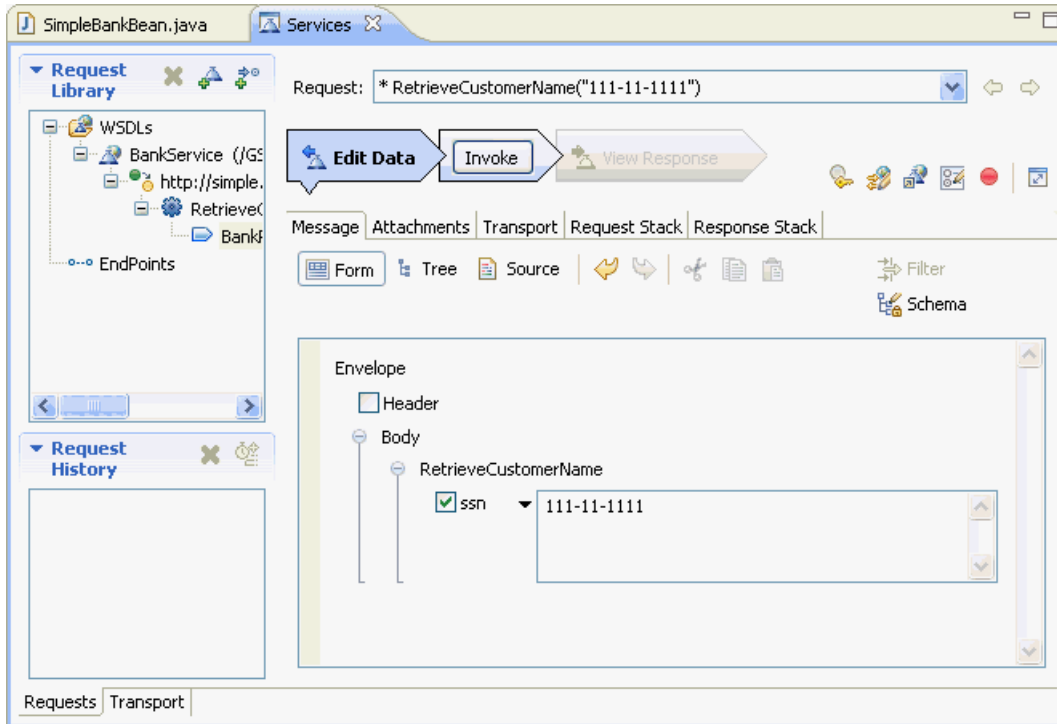


Figure 14-6 Generic Service Client

4. The **RetrieveCustomerName** operation is already selected.
5. Click the field **ssn**. In the **ssn** field, type 111-11-1111 and then click **Invoke**. The result (Mr. Henry Cui) is displayed in the Form pane. See Figure 14-7 on page 706.

Tip: The Generic Service Client creates a WSDL dynamically and places it inside a hidden project called GSC Store inside the Rational Application Developer workspace. For this WSDL to have the correct URL (host name and port) to invoke the service on your WebSphere Application Server Test Environment, you must publish the project to WebSphere Application Server before invoking the GSC.

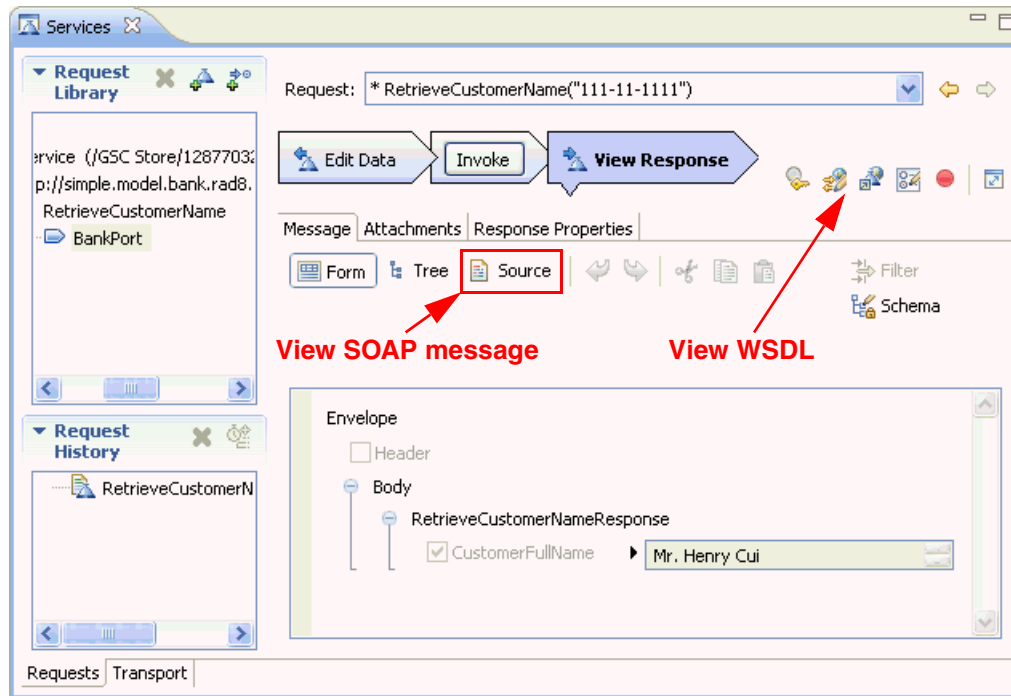


Figure 14-7 Results of invocation of the web service with GSC

- Click the **Source** pane to view the SOAP messages as raw XML, as shown in Example 14-13.

Example 14-13 SOAP message

```

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"><soapenv:Body><ns2:RetrieveCustomerNameResponse
xmlns:ns2="http://simple.model.bank.rad8.itso/"><CustomerFullName>Mr
. Henry
Cui</CustomerFullName></ns2:RetrieveCustomerNameResponse></soapenv:Body></soapenv:Envelope>

```

Viewing the dynamically generated WSDL

In JAX-WS web services, the deployment descriptors are optional, because they use annotations. The WSDL file can be dynamically generated by the run time based on information that it gathers from the annotations added to the Java classes.

The URL for the dynamically generated WSDL is in the following format:

```
http://<hostname>:<port>/<Web project context root>/<service name>?wsdl
```

To view the dynamically generated WSDL, enter the following URL in the browser (908x is the port number, most probably 9080 or 9081):

```
http://localhost:908x/RAD8WebServiceWeb/BankService?wsdl
```

Tip: You can also see the WSDL from the Generic Service Client, as shown in Figure 14-7 on page 706.

The dynamically generated WSDL file is displayed. We also notice that the URL for the WSDL is changed:

```
http://localhost:908x/RAD75WebServiceWeb/BankService/BankService.wsdl
```

Examine the generated WSDL. We can see that the generated WSDL matches the web services annotations that we added. Example 14-14 shows an extract of the generated WSDL snippet.

Example 14-14 Dynamically generated WSDL snippet

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="BankService"
  targetNamespace="http://simple.model.bank.rad8.itso/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:tns=
    "http://simple.model.bank.rad8.itso/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types> .....
  <message> .....
  .....
  <portType name="Bank">
    <operation name="RetrieveCustomerName">
      <input message="tns:RetrieveCustomerName" />
      <output message="tns:RetrieveCustomerNameResponse" />
      <fault name="CustomerDoesNotExistException"
        message="tns:CustomerDoesNotExistException" />
    </operation>
  </portType>
  <binding name="BankPortBinding" type="tns:bank">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="RetrieveCustomerName">
      <soap:operation soapAction="urn:getCustomerFullName" />
      <input>
```

```

        <soap:body use="literal" />
    </input>
    <output>
        <soap:body use="literal" />
    </output>
    <fault name="CustomerDoesNotExistException">
        <soap:fault name="CustomerDoesNotExistException"
use="literal"/>
    </fault>
</operation>
</binding>
<service name="BankService">
    <port name="BankPort" binding="tns:BankPortBinding">
        <soap:address
location="http://localhost:9080/RAD8WebServiceWeb/BankService" />
    </port>
</service>
</definitions>

```

To see the dynamically generated XML schema, enter the following URL:

```
http://localhost:908x/RAD8WebServiceWeb/BankService/BankService_schema1.xsd
```

For a simple test to verify that the web service is running in the server, enter the following URL:

```
http://localhost:908x/RAD8WebServiceWeb/BankService
```

The following result is displayed in the browser:

```
{http://simple.model.bank.rad8.itso/}BankService
Hello! This is an Axis2 Web Service!
```

14.7.2 Creating web services using the Web Service wizard

The Web Service wizard assists you in creating a new web service, configuring it for deployment, and deploying the web service to a server. To create a web service from a JavaBean, follow these steps:

1. In the Java EE perspective, expand **RAD8WebServiceWeb2** → **Java Resources: src** → **itso.rad8.bank.model.simple**. Right-click **SimpleBankBean.java** and select **Web Services** → **Create Web service**. The Web Service wizard starts.

2. In the Web Services window, select the following options:
 - a. For Web service type, ensure that **Bottom up Java bean Web Service** is selected (default).
 - b. Under Service implementation, move the slider to the **Test** position (top) to access testing options for the service on subsequent windows.

The slider: The slider offers a more granular division of web services development. By using the slider, you can select from the following stages of web services development:

Develop	Develops the WSDL definition and implementation of the web service. It includes tasks, such as creating the modules that will contain the generated code, WSDL files, deployment descriptors, and Java files when appropriate.
Assemble	Ensures that the project that hosts the web service or client is associated with an EAR when required by the target application server.
Deploy	Creates the deployment code for the service.
Install	Installs and configures the web module and EAR files on the target server. If any changes to the endpoints of the WSDL file are required, they are made in this stage.
Start	Starts the web service after the service is installed on the server.
Test	Provides various options for testing the service, such as using the Web Services Explorer or sample JSP.

- c. Ensure that the following server-side configurations are selected, as shown in Figure 14-8 on page 710:
 - Server runtime: **WebSphere Application Server v8.0 Beta**
 - Web service run time: **IBM WebSphere JAX-WS**
 - Service project: **RAD8WebServiceWeb2**
 - Service EAR project: **RAD8WebServiceEAR**

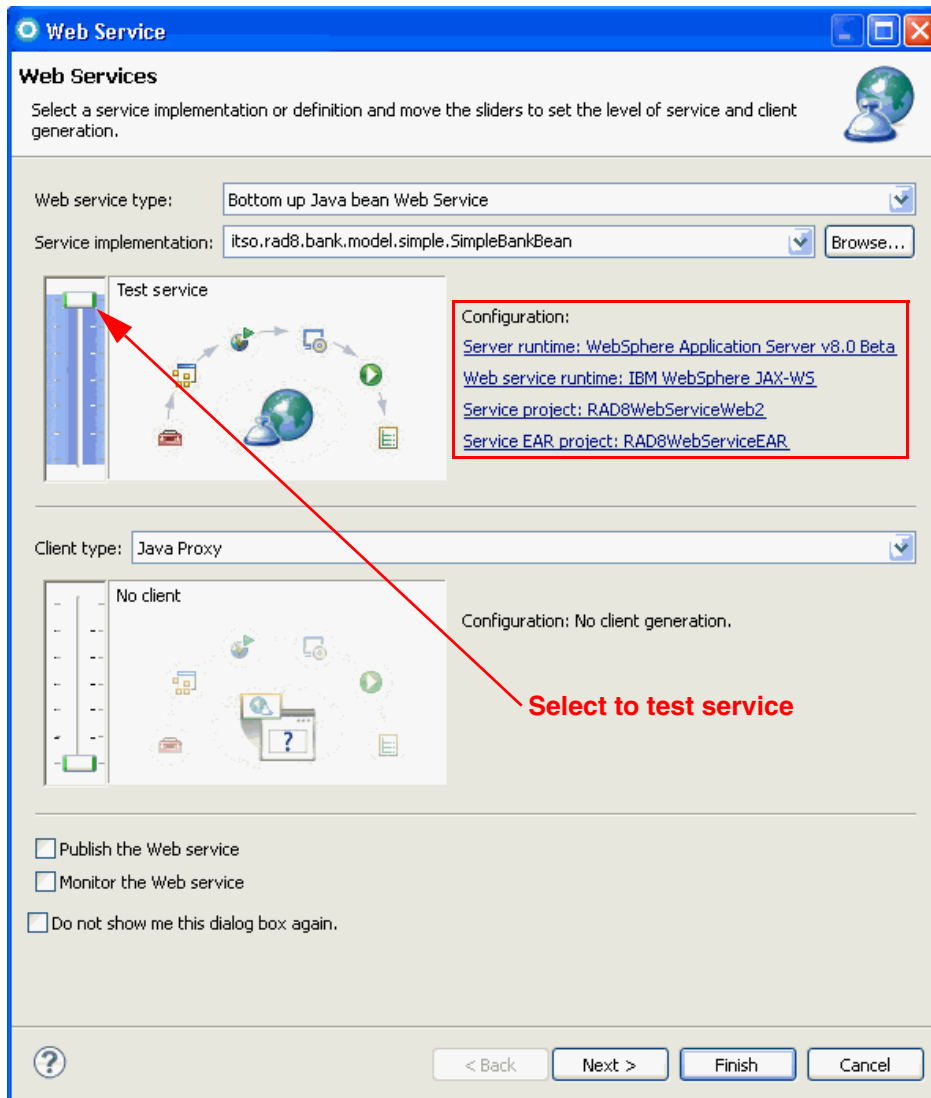


Figure 14-8 Web Services dialog window

- Under Configuration, if you click the **Server: WebSphere Application Server v8.0 Beta** link, the Service Deployment Configuration window (Figure 14-9 on page 711) opens. In this window, you can select the server and run time. We use the default settings of this window. Click **Cancel** to close the window and return to the Web Services window.

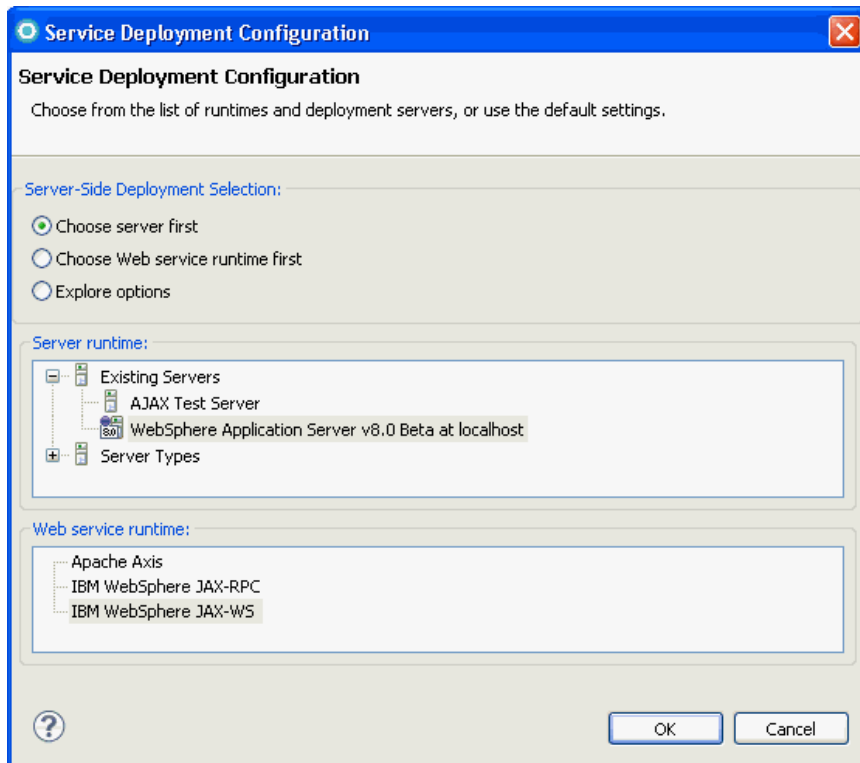


Figure 14-9 Web Services wizard: Service Deployment Configuration

- d. Clear the **Publish the Web service** check box (because we do not publish to a Universal Description, Discovery, and Integration (UDDI) registry).
 - e. Clear the **Monitor the Web service** check box (because we select to monitor the web service later).
 - f. Click **Next**.
3. In the WebSphere JAX-WS Bottom Up Web Service Configuration window (Figure 14-10 on page 713):
 - a. For Delegate class name, accept the default (**SimpleBankBeanDelegate**).
 The delegate class is a wrapper that contains all the methods from the JavaBean and the JAX-WS annotation that the run time recognizes as a web service.
 - b. For Java to WSDL mapping style, accept the default.

The style defines the encoding style for messages that are sent to and from the web service. The recommended WSDL style is **Document Wrapped**.

c. Select **Generate WSDL file into the project**.

Because the annotations in the delegate class are used to indicate to the run time that the bean is a web service, a static WSDL file is no longer generated to your project automatically. The run time can dynamically generate a WSDL file from the information in the bean. Select this option to generate a static WSDL file for the web service. There are several reasons to select this option:

- Performance improvements. For a large bean with lots of methods and complex data types, this option prevents the penalty of the initial generation by the run time when the service is accessed.
- This option is required for SOAP 1.2.
- To enforce a contract with the bean via `@WebService.wsdlLocation`. The JAX-WS annotations processor will validate the WSDL against the bean.

Change the name of the generated WSDL to **BankService.wsdl**.

d. Select **Generate Web service deployment descriptor**.

For JAX-WS web services, deployment information is generated dynamically by the run time; static deployment descriptors are optional. Selecting this check box generates the deployment descriptors.

e. Click **Next**.

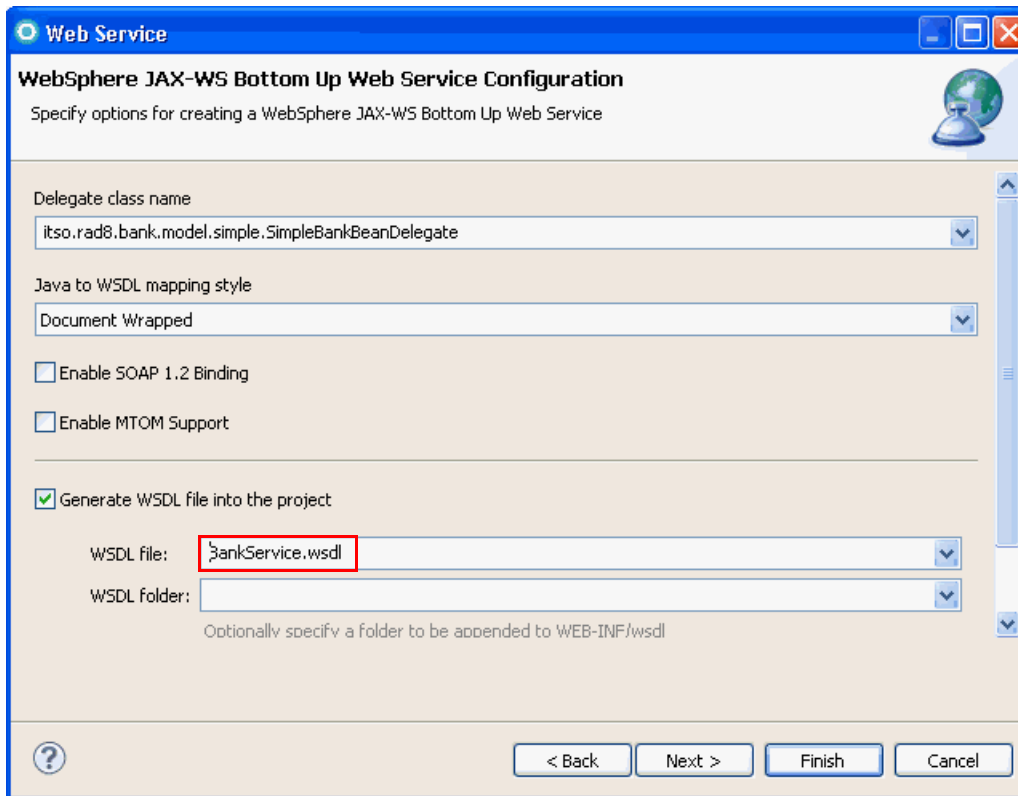


Figure 14-10 WebSphere JAX-RS Bottom Up Web Service Configuration

4. In the WebSphere JAX-WS WSDL Configuration window (Figure 14-11 on page 714), perform these tasks:
 - a. Select **WSDL Target Namespace**, and for the WSDL Target Namespace, enter `http://bank.rad8.itso/`.
 - b. Select **Configure WSDL Service Name**, and for the WSDL Service Name, enter `BankService`.
 - c. Select **Configure WSDL Port Name**, and for the WSDL Port Name, enter `BankPort`.
 - d. Click **Next**.

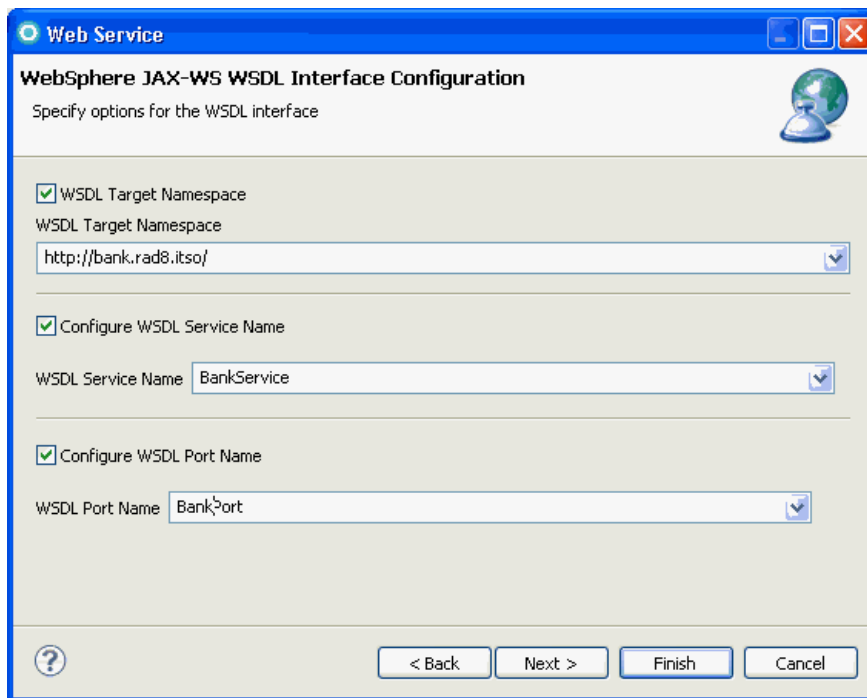


Figure 14-11 WebSphere JAX-WS WSDL Interface Configuration

The web service is generated and deployed to the server.

5. In the Test Web Service window (Figure 14-12 on page 715), which opens because we moved the slider for the service to the Test position, select the **Generic Service Client** and click **Launch**.

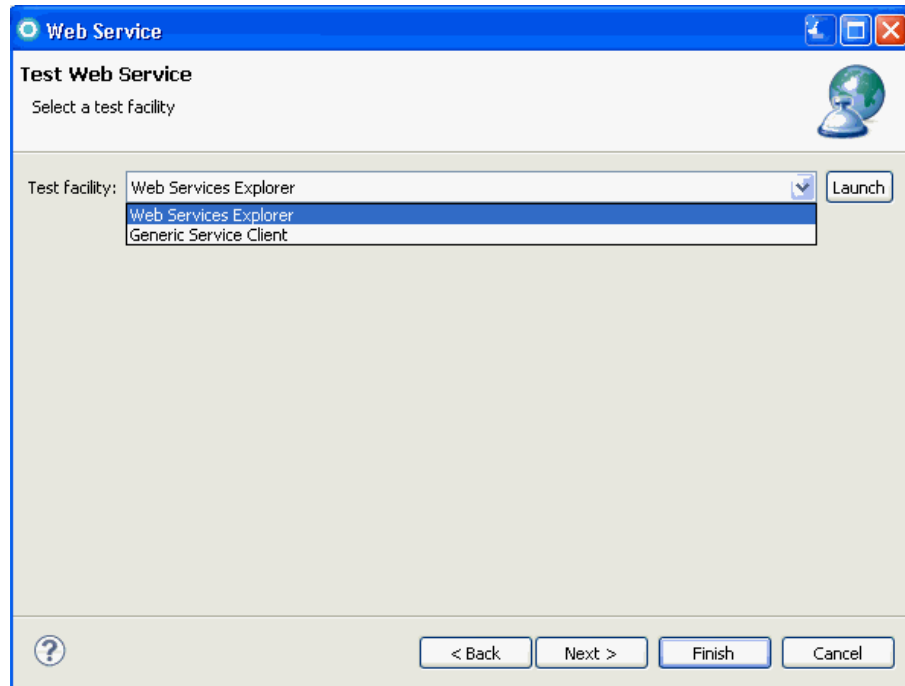


Figure 14-12 Select the Generic Service Client as a test facility

6. The Generic Service Client opens in an external web browser (see Figure 14-13 on page 716). Complete these tasks:
 - a. Select the **getNumAccounts** operation and click **Add**.
 - b. Enter a value for the customer ID, such as 111-11-1111, and click **Go**.
The result 2 is displayed in the status pane. Optional: Try other operations.
7. Close the Web Services Explorer. Click **Finish** to exit the Web Service wizard.

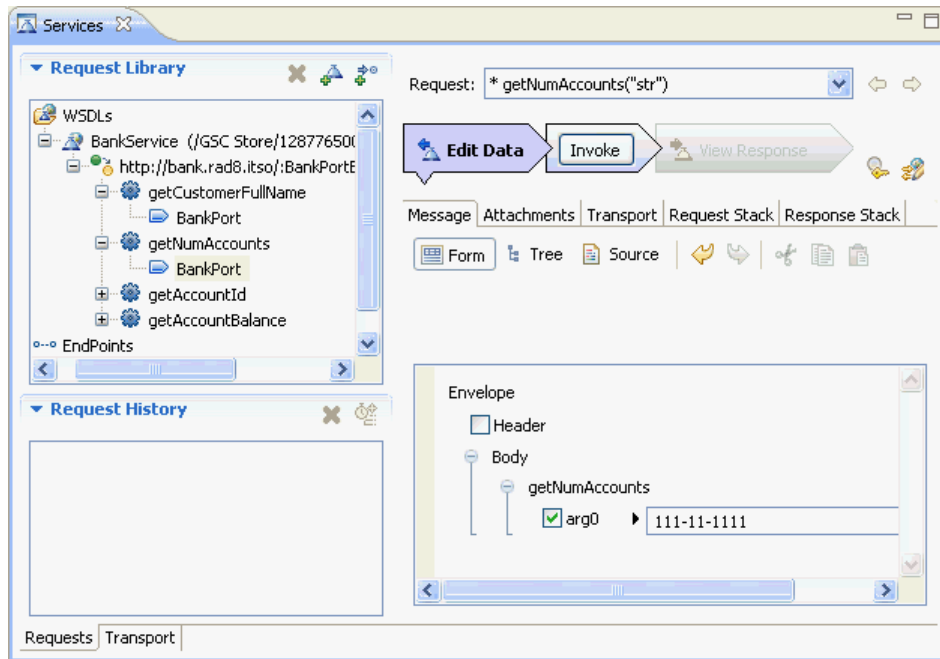


Figure 14-13 Test `getNumAccounts` with Generic Service Client

The web services are available at two endpoints: the HTTP endpoint and the HTTPS endpoint. If your server is not secured, the endpoint is:

`http://localhost:908x/RAD8WebServiceWeb2/BankService`

If your server is secured, the web service listed in the Generic Server Client has the following endpoint:

`https://localhost:944x/RAD8WebServiceWeb2/BeanService`

You can see the current endpoint by selecting the **Add EndPoint Request** icon, as shown in Figure 14-14 on page 717.

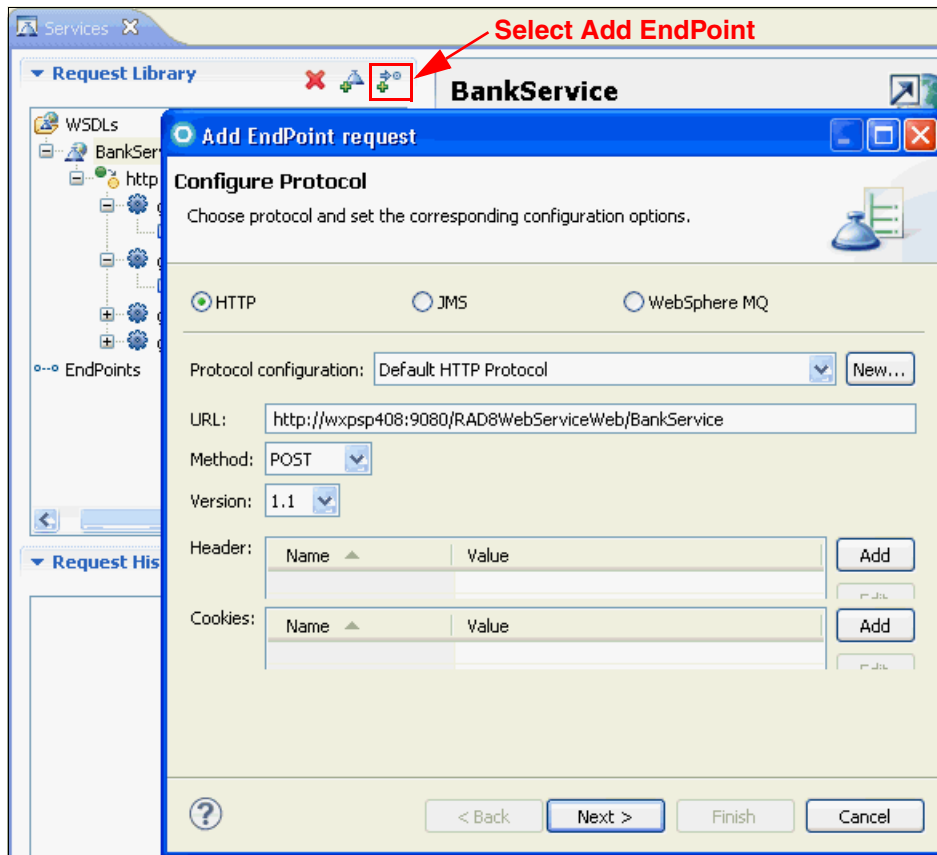


Figure 14-14 Add EndPoint request

To test the HTTPS protected web service with the Generic Service Client, you can configure a new protocol configuration. The signer certificate from the WebSphere Application Server must be imported into the Eclipse truststore. For more information about creating a new Secure Sockets Layer (SSL) configuration, see this website:

<http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=com.ibm.rational.ttt.common.doc/topics/tgsccreatessl.html>

You have successfully created web services from a JavaBean.

14.7.3 Resources generated by the Web Service wizard

After code generation, examine the generated code.

You can see that the wizard generates the following artifacts:

- ▶ A delegate class named `SimpleBankBeanDelegate`. The delegate class is a wrapper that contains all the methods from the `JavaBean` and the `JAX-WS` annotation that the run time recognizes as a web service. The annotation `@javax.jws.WebService` in the delegate class tells the server runtime environment to expose all public methods on that bean as a web service. The `targetNamespace`, the `serviceName`, and the `portName` are what we specified in the `Web Service` wizard.

```
@javax.jws.WebService (targetNamespace="http://bank.rad8.itso/",  
    serviceName="BankService", portName="BankPort",  
    wsdlLocation="WEB-INF/wsdl/BankService.wsdl")
```

- ▶ A `webservices.xml` file in the `WebContent/WEB-INF` folder. This file is the optional web services deployment descriptor. A deployment descriptor can be used to override or enhance the information provided in the service. For example, if the `<wsdl-service>` element is provided in the deployment descriptor, the namespace used in this element overrides the `targetNamespace` member attribute in the annotation.
- ▶ A WSDL file (`BankService.wsdl`) and an XSD file (`BankService_schema1.xsd`) in the `WEB-INF/wsdl` folder. If you plan to create the client at a later time or publish the WSDL for other users, you can use this WSDL file.

You can locate the projects developed up to this point in the `c:\7835codesolution\webservices` folder in the `RAD8WebServiceImplemented.zip` file.

14.8 Creating a synchronous web service JSP client

The `Web Service Client` wizard assists you in generating a `JavaBean` proxy and a sample application. The sample web application demonstrates how to invoke the web services proxy. You can invoke the web services using the `JAX-WS` synchronous model, or the asynchronous model. In the section, we generate a synchronous web service client.

14.8.1 Generating and testing the web service client

To generate a client and test the client proxy, follow these steps:

1. Switch to the **Services** view, right-click **RAD8WebServiceWeb**, and select **Generate** → **Client**, as shown in Figure 14-15 on page 719.

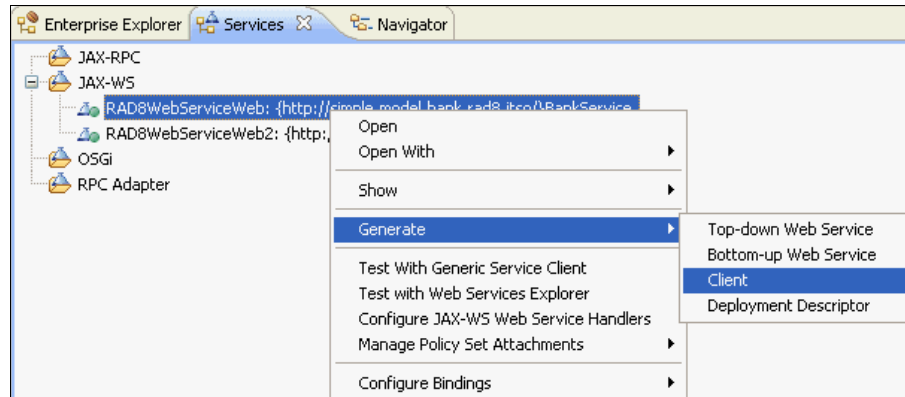


Figure 14-15 Invoking Generate Client

2. In the Web Services Client window (Figure 14-16 on page 720), perform these steps:
 - a. Move the slider up to the **Test** position. This position provides options for testing the service using a JSP-based sample application.
 - b. Select **Monitor the Web service**.
 - c. Place the web service and web service client in separate web and EAR projects. Click **Client project**.
3. In the Specify Client Project Settings window, complete the following actions and then click **Next**:
 - a. Change the client project name to `RAD8WebServiceClient`.
 - b. For Project type, accept **Dynamic Web project**.
 - c. For Client EAR project name, accept **RAD8WebServiceClientEAR**.
 - d. Click **OK**. The wizard creates the Web and EAR projects.

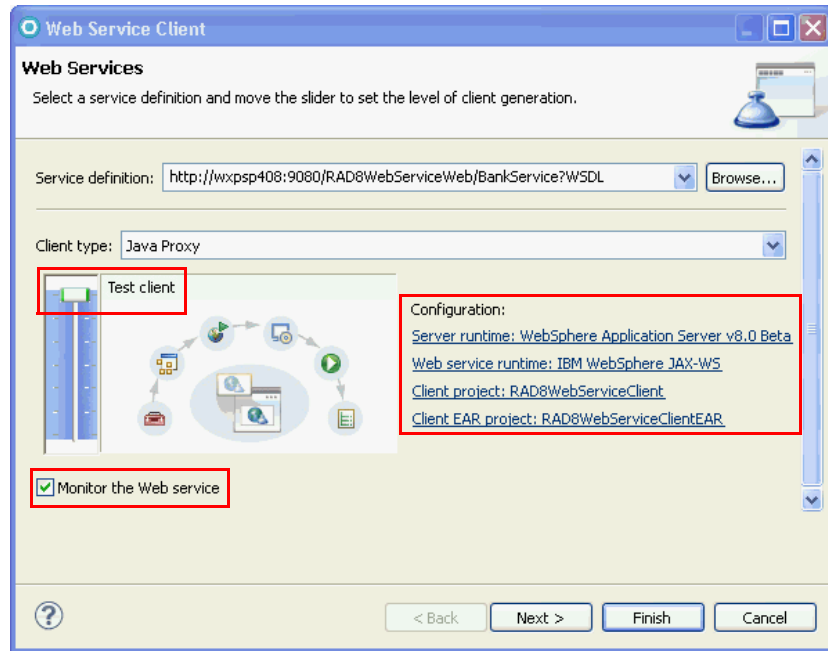


Figure 14-16 Generating the web service client

4. Perform these steps in the WebSphere JAX-WS Web Service Client Configuration window (Figure 14-17 on page 721):
 - a. Accept the default name and location of the Deployment Descriptor.
 - b. Accept **Generate Portable Client**.
 - c. Clear **Enable MTOM**.
 - d. The version of JAX-WS to be generated is **2.2**, by default.
 - e. Click **Next**.

The client code is generated into the new client project.

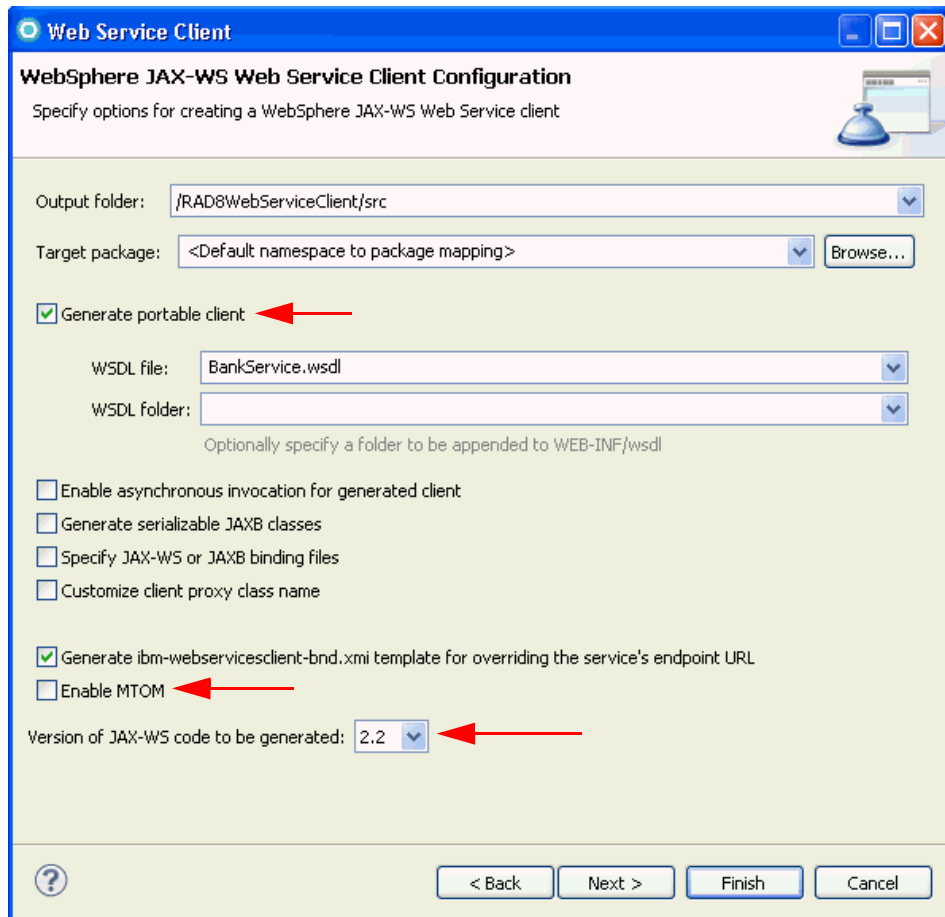


Figure 14-17 JAX-WS Web Service Client Configuration

5. In the Web Service Client Test window (Figure 14-18 on page 722), use these settings:
 - a. Select **Test the generated proxy**.
 - b. For Test facility, select **JAX-WS JSPs** (default).
 - c. For Folder, select **sampleBankPortProxy** (default). You can specify a separate folder for the generated application if you want.
 - d. Under Methods, leave all methods selected.
 - e. Select the **Run test on server** check box.
 - f. Click **Finish**.

The sample application is published to the server, and the sample JSP is displayed in a Web browser.

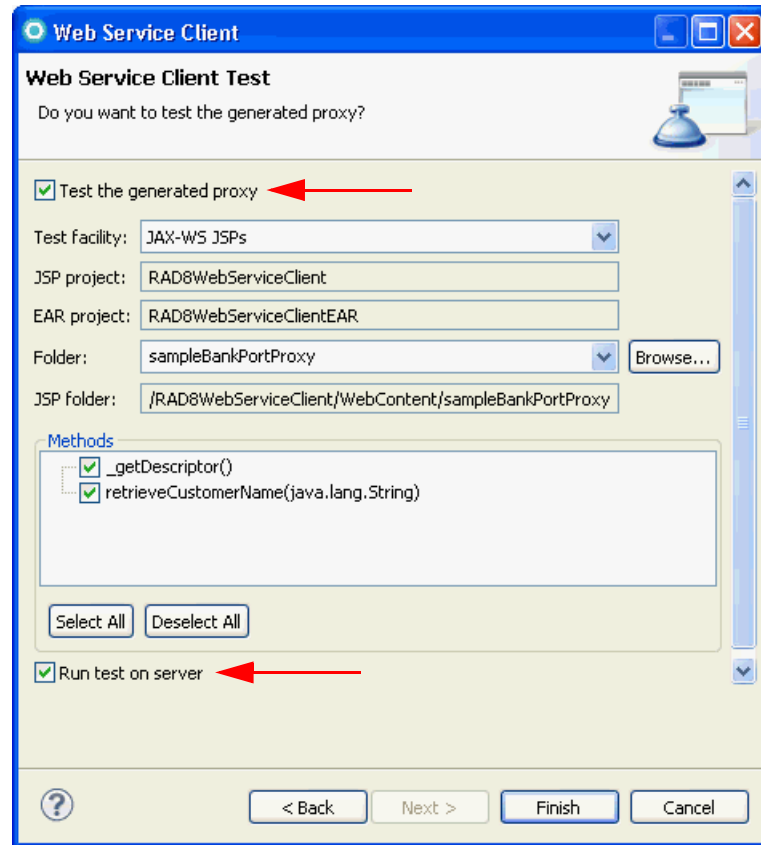


Figure 14-18 Web Service Client Test

6. In the Web Services Test Client window (Figure 14-19 on page 723), perform these steps:
 - a. Select the **retrieveCustomerName** method.
 - b. Enter a valid value in the customer ID field, such as 111-11-1111.
 - c. Click **Invoke**.

The results are displayed in the Result pane.

Notice the endpoint in the Quality of Service pane:

`http://hostname:12036/RAD8WebServiceWeb/BankService`

You might see another port number. It depends on the port number that the wizard generated for the TCP/IP Monitor.

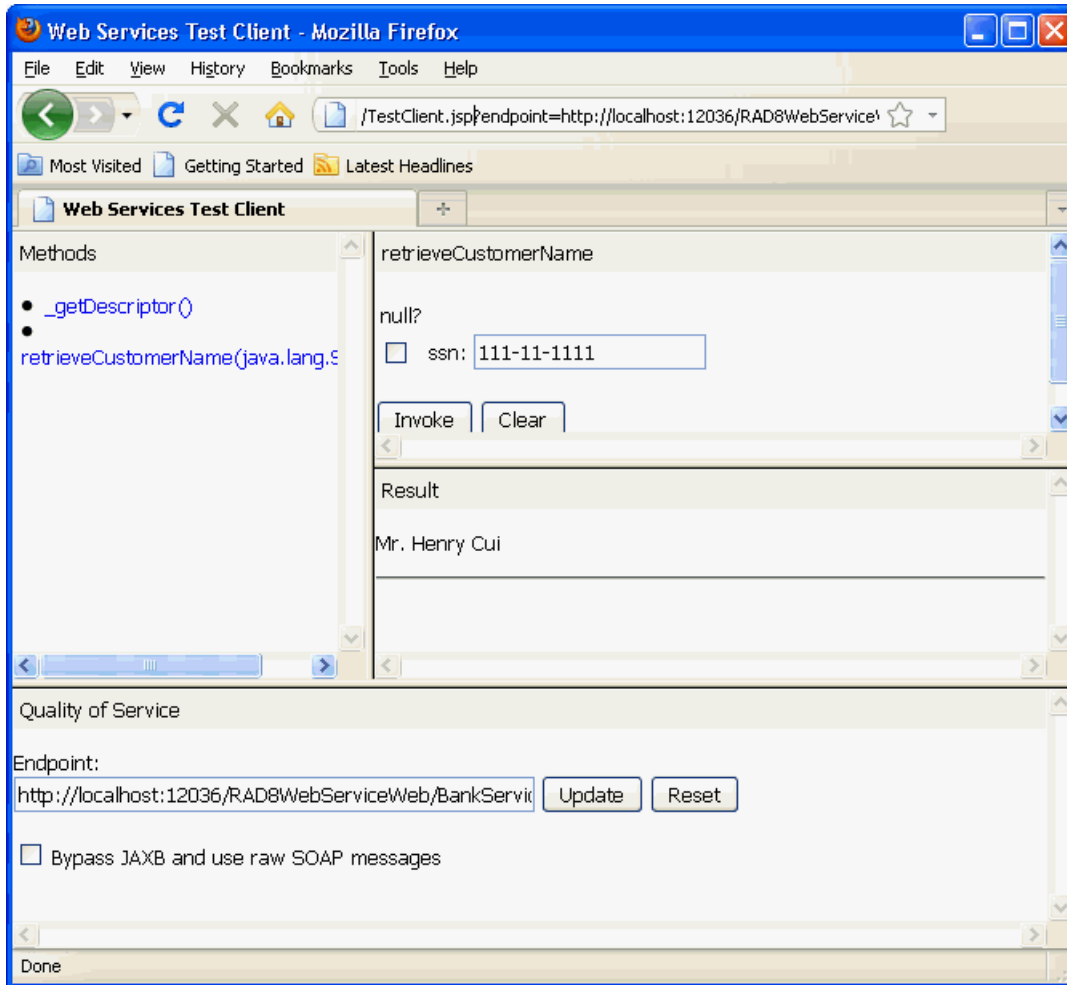


Figure 14-19 Testing the generated JSP

The TCP/IP Monitor is also started. With the TCP/IP Monitor, you can intercept and examine the SOAP traffic that comes in and out of a web service call.

7. If you select **Window** → **Preferences** and then select **Run/Debug** → **TCP/IP Monitor**, you can see that a new monitor has been added. It is configured to listen to the same local port number (12036).

The TCP/IP Monitor is started and ready to listen to the SOAP request and direct it to the web service provider (possibly on a separate host and at port 908x).

Monitor the Web service: When you select the “Monitor the Web service” option in the Web Service window, the Web Service Client wizard dynamically creates the TCP/IP Monitor. It uses an algorithm to locate an available listening port for the monitor. The sample JSP client window uses the URL to dynamically set the web service endpoint to match the monitor port. Using the wizard to create the TCP/IP Monitor is convenient, because you do not have to spend time determining how to redirect the SOAP request to the TCP/IP Monitor, especially when monitoring remote web services.

All requests and responses are routed through the TCP/IP Monitor and are displayed in the TCP/IP Monitor view. The TCP/IP Monitor view might be displayed in the same pane as the Servers view.

The TCP/IP Monitor view shows all the intercepted requests in the top pane, and when a request is selected, the messages passed in each direction are shown in the bottom panes (the request in the left pane, and the response in the right pane). The TCP/IP Monitor can be a useful tool in debugging web services and clients.

8. Select the **XML** view to see the SOAP request and response in XML format, as shown in Figure 14-20.

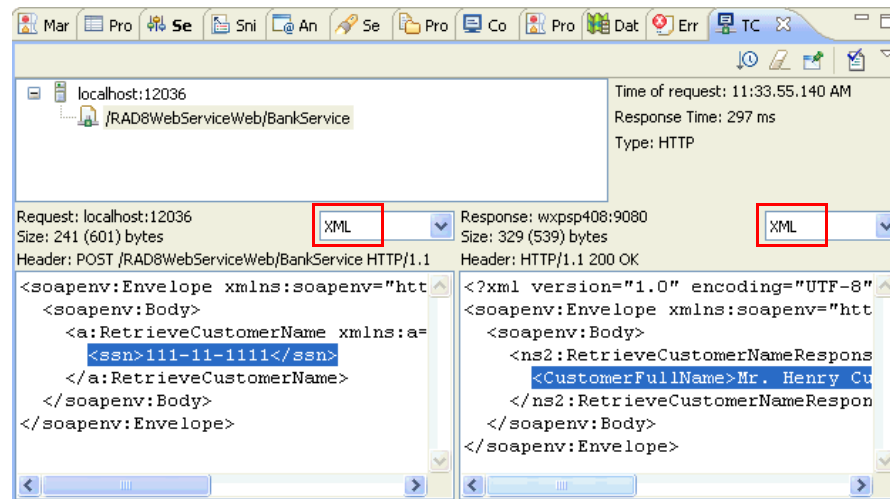



Figure 14-20 TCP/IP Monitor

9. Optional: To ensure that the web service SOAP traffic is WS-I compliant, you can generate a log file by clicking the  icon in the upper-right corner. In the window that opens, select a name for the log file and specify where you want to store it (for example, in the client project).

The log file is validated for WS-I compliance. You see a confirmation message, “The WS-I Message Log file is valid.” You can open the log file in an XML editor to examine its contents.

To stop the TCP/IP Monitor, perform these steps:

1. Select **Window** → **Preferences**.
2. Select **Run/Debug** → **TCP/IP Monitor**.
3. Select the TCP/IP Monitor from the list and select **Stop**.

Manually starting the TCP/IP Monitor: To start the TCP/IP Monitor manually, remember that the Local Monitoring port is a randomly chosen free port on localhost, while the host and port refer to the actual parameters of the server where your service is running. To test the service through the monitor, you have to manually change the host and port in the endpoint of the service you are testing, so that your request is sent to the monitor instead of the actual server.

Resources generated by the Web Service Client wizard

Figure 14-21 on page 726 shows the generated web service client artifacts.

We provide a description of each of the web service client artifacts here:

- ▶ `Bank.java` is the annotated service interface based on the WSDL to Java mapping.
- ▶ `BankService.java` is generated from the WSDL service. It is a factory class that returns an instance that implements the service’s interface, which is also known as a *JAX-WS Service class*. In JAX-RPC, this implementation class is called a *stub*. In JAX-WS, no stub class exists; the stub is a class that is dynamically generated from WSDL.
- ▶ `BankPortProxy.java` is an IBM-proprietary proxy class. JAX-WS does not define this class. It is a convenience class that implements the web service’s interface and hides programming details, such as the service factory and binding provider calls.
- ▶ The rest of the Java classes are the JAXB artifacts that are based on the schema types used by the WSDL.
- ▶ The `sampleBankPortProxy` folder contains the generated sample JSP, which demonstrates how to invoke the web services proxy.
- ▶ The `ibm-webservicesclient-bnd.xml` file is the IBM proprietary Web Services Client binding file.
- ▶ The `web.xml` is no longer required and was not generated.

You can get the results in this file:

c:\7835codesolution\webservices\RAD8WebServiceJSPClient.zip

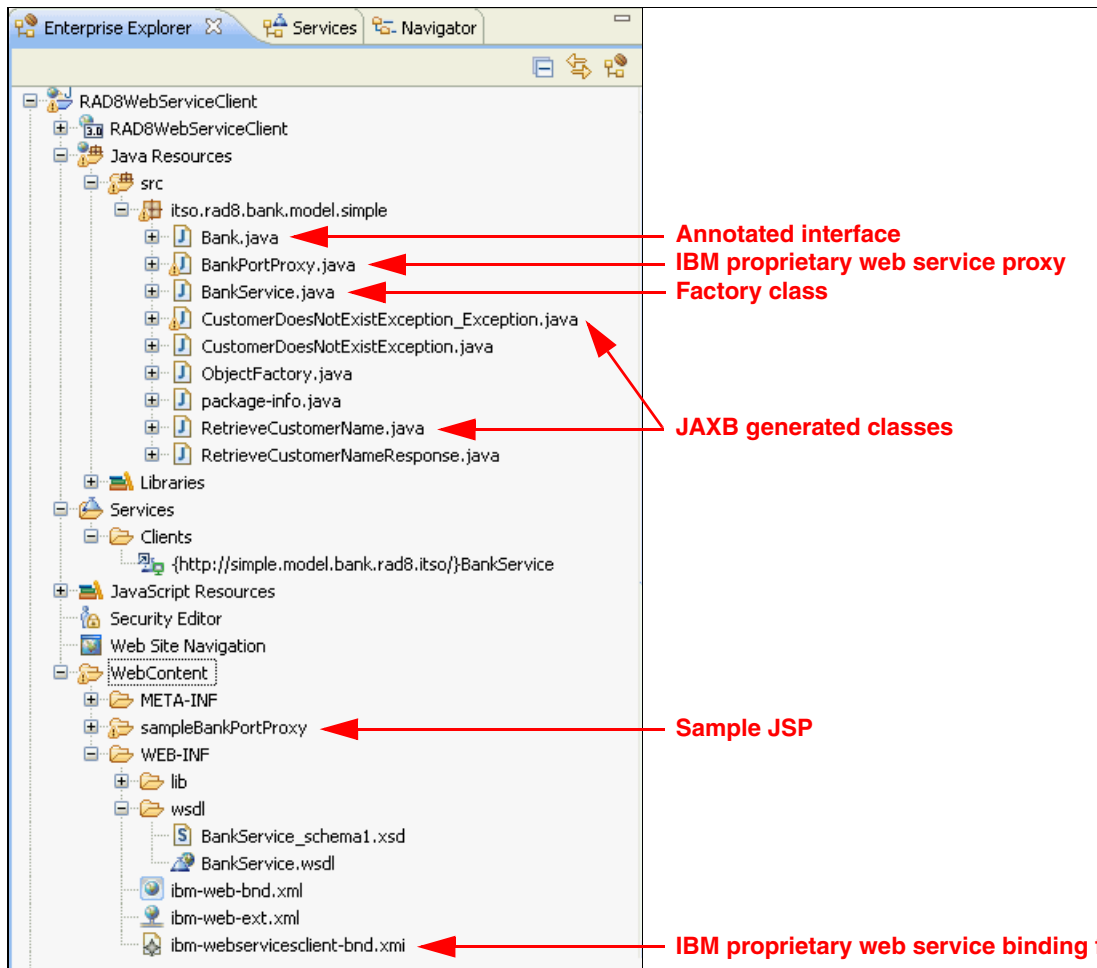


Figure 14-21 Generated web service client artifacts

14.9 Creating a web service JavaServer Faces client

With the Web Service Discovery window, you can discover a web service that exists online or in the workspace, create a proxy to the web service, and then place the methods of the proxy on a Faces JSP file:

1. Click **Add and Remove Projects** to remove the `RAD8WebServiceClientEAR` from the server. (We add a project to the EAR and automatic publishing

interferes.) Alternatively, expand the server, right-click the project, and select **Remove**.

2. Create a dynamic web project by selecting **File** → **New** → **Dynamic Web Project**.
3. In the Dynamic Web Project window (Figure 14-22), complete the following steps:
 - a. For Project name, enter **RAD8WebServiceJSFClient**.
 - b. In the Configuration section, select **JavaServer Faces v2.0 Project** to add the required JSF facets to the project facets list.
 - c. For EAR Project Name, select **RAD8WebServiceClientEAR**.
 - d. Click **Finish**.

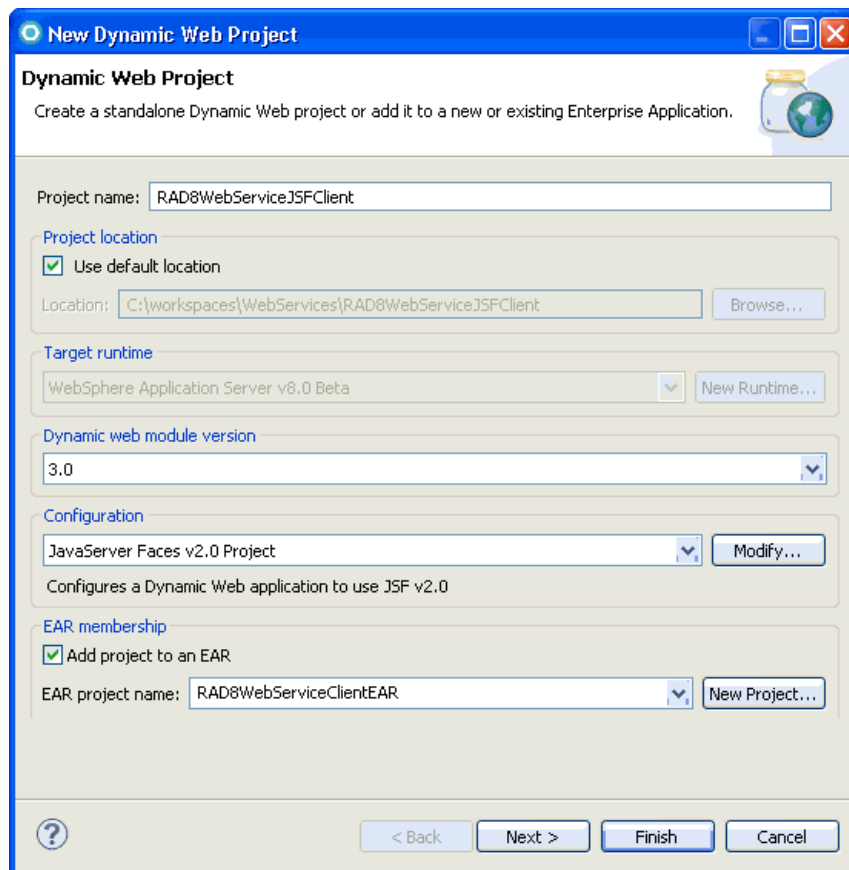


Figure 14-22 Create a new JSF 2.0 project

4. If you are prompted to open the Web perspective, click **Yes**.

5. In the RAD8WebServiceJSFClient project, right-click **WebContent** and select **New** → **Web Page**.
6. For File name, enter WSJSFClient. For Basic template, select **Facelet** and click **Finish**. The WSJSFClient.jsp opens in an editor.
7. Select the **Design** or **Split** tab.
8. In the Palette, select the **Data and Services** category. Select **Web Service** and click the JSF page, as shown in Figure 14-23.

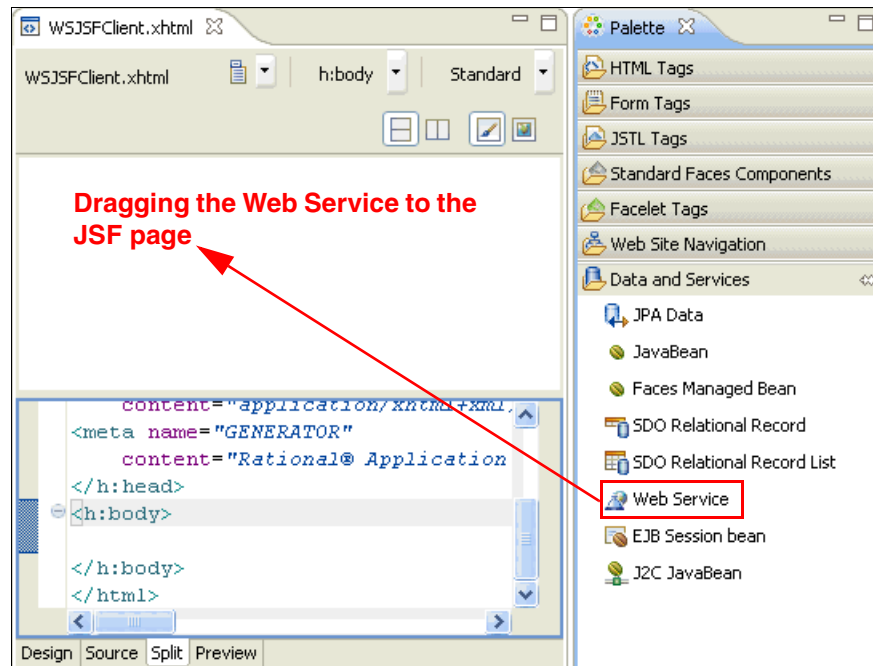


Figure 14-23 Dragging the Web Service to the JSF page

9. In the Add Web Service window, as shown in Figure 14-24, click **Add**. In the Web Services Discovery window, select **Web services from your workspace**.

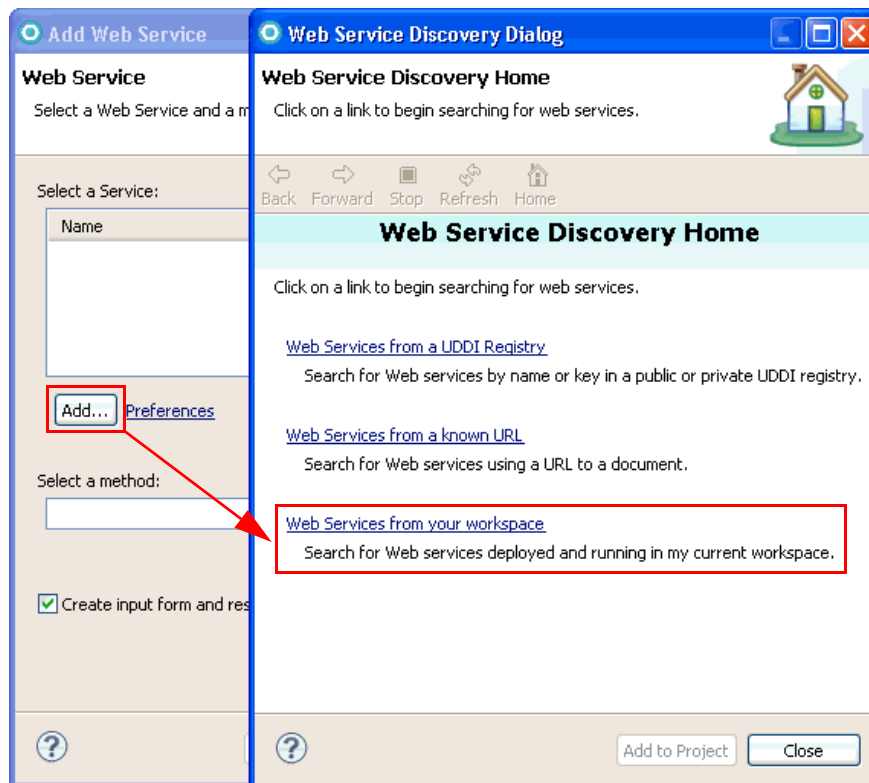


Figure 14-24 Adding Web Services from your workspace

10. In the Web Services from your workspace window, which is shown in Figure 14-25, click **BankService** with the URL of the **RAD8WebServiceWeb** project (not the RAD8WebServiceWeb2 project).

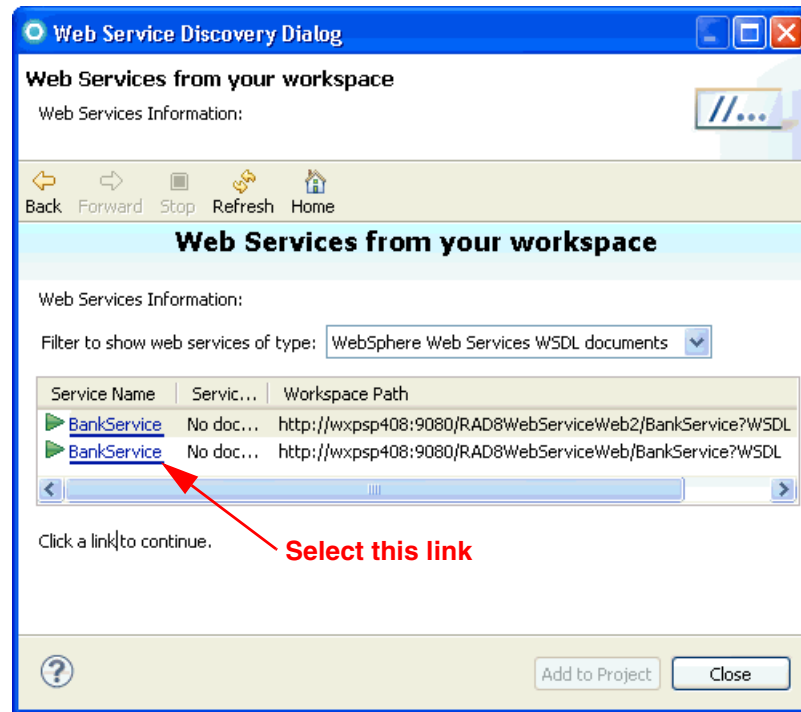


Figure 14-25 Web Service Discovery Dialog: Web Services from your workspace

11. Select **Port: BankPort** and click **Add to Project** (Figure 14-26).

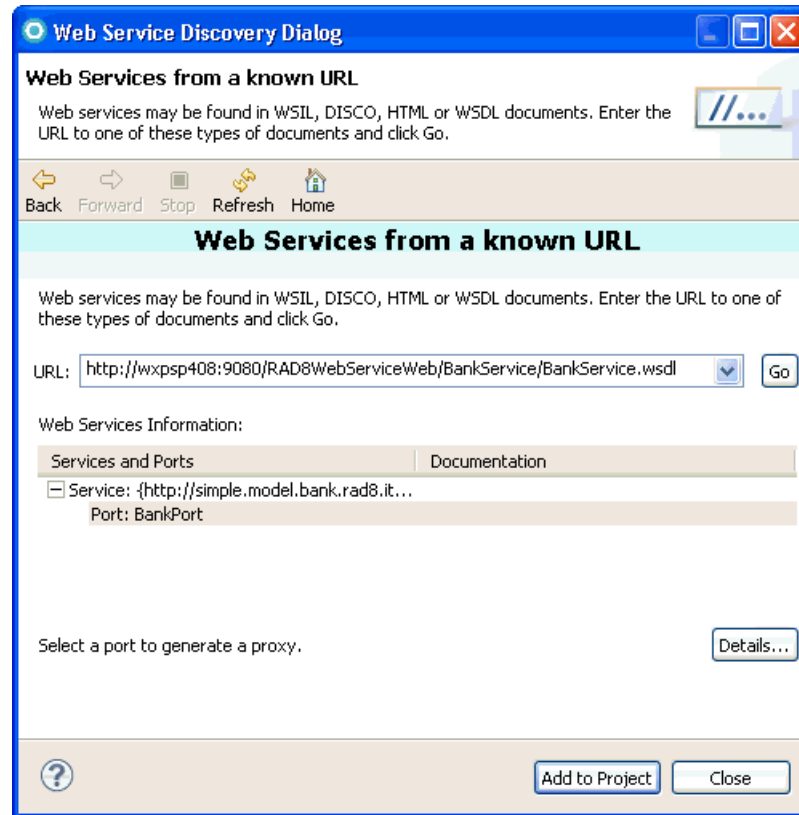


Figure 14-26 Web Services Discovery Dialog: Clicking the Add to Project button

The web service that you selected is now listed in the list of web services.

12. In the Web Service window (Figure 14-27), perform these steps:
- For Service Name, select **Bank**.
 - For the method, select **retrieveCustomerName(String)**.
 - Select **Create input form and results display**.
 - Click **Next**.

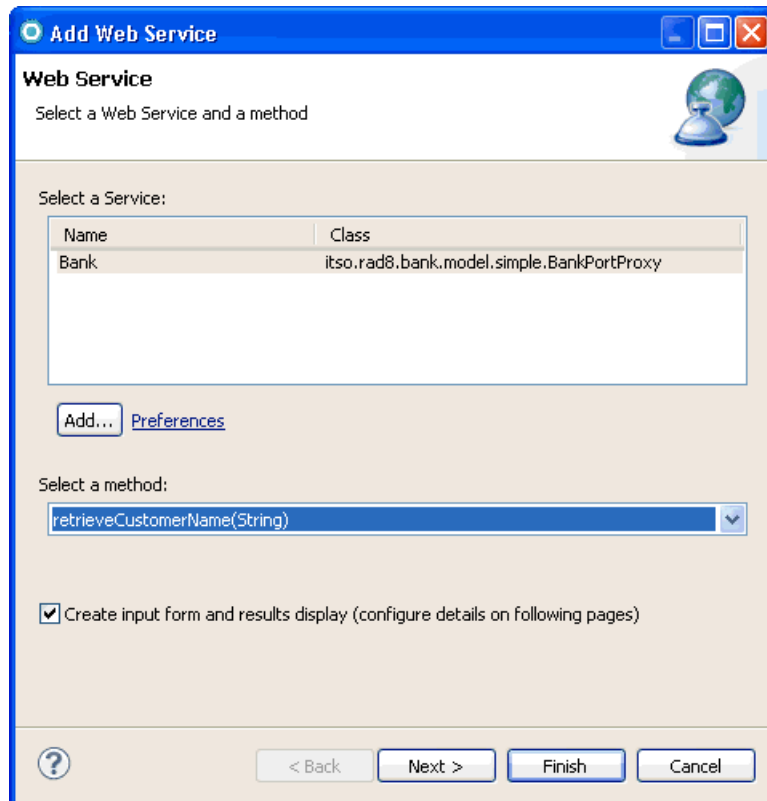


Figure 14-27 Selecting a web service and method

13. In the Input Form window (Figure 14-28), perform these steps:
 - a. Change the label to Enter Social Security Number:.
 - b. Click **Options** and change the label from Submit to Get Full Name.
 - c. Click **OK** and click **Next**.

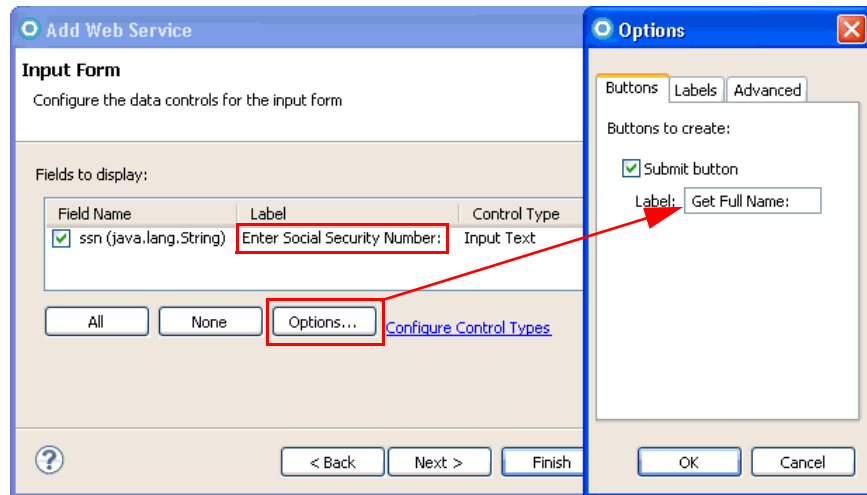


Figure 14-28 Web service input form

14. In the Results form window (Figure 14-29), change the Label to Customer's full name is:.
15. Click **Finish** to generate the input and output parts into the JSF page (Figure 14-29). Save the file.

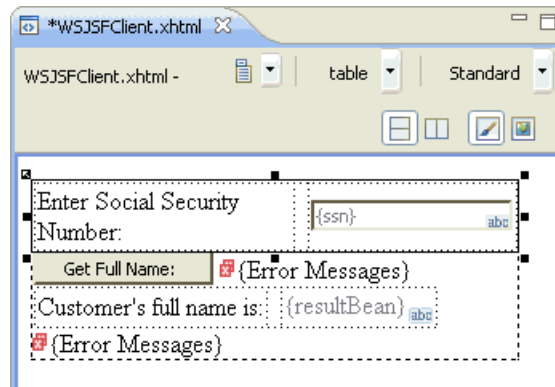


Figure 14-29 JSF page with the web service invocation

16. Right-click **WSJSFClient.jsp** and select **Run As** → **Run on Server**. The client application is deployed to the server for testing. Perform this test:
 - a. In the Enter Social Security Number field, type 111-11-1111.
 - b. Click **Get Full Name**.

The result is displayed (Figure 14-30).

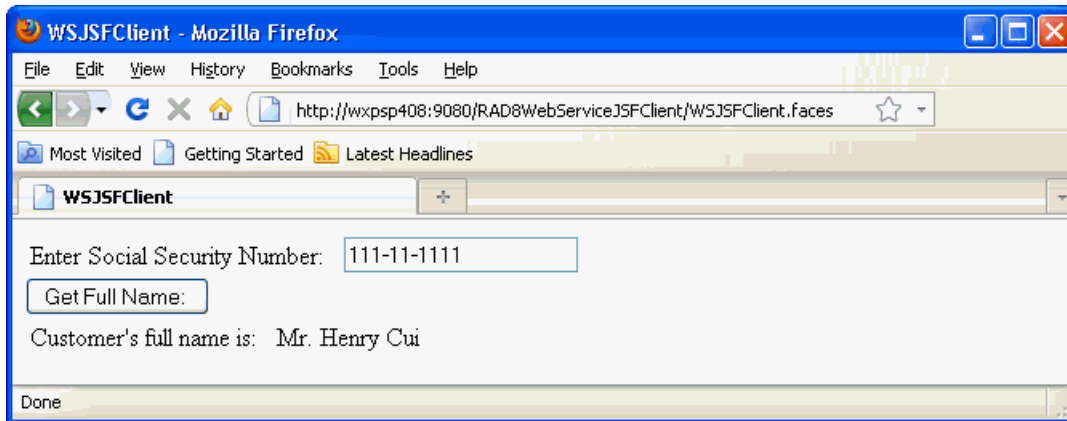


Figure 14-30 JSF client run

The projects that have been developed up to this point are available in this folder:

C:\7835codesolution\webservices\RAD8WebServiceJSFClient.zip

14.10 Creating a web service thin client

WebSphere Application Server provides an unmanaged client implementation that is based on the JAX-WS 2.2 specification. The *thin client* for JAX-WS with WebSphere Application Server is an unmanaged and stand-alone Java client environment. The thin client enables running JAX-WS client applications to invoke web services that are hosted by WebSphere Application Server. A web service thin client relies only on a Java developer kit that is compatible with IBM WebSphere Application Server V8 Beta and a thin client JAR file that is available in the `<WAS_HOME>\runtimes\com.ibm.jaxws.thinclient_8.0.0.jar` file where typically `<WAS_HOME>=C:\Program Files\IBM\WebSphere\AppServer`.

Creating the thin client project and generating the client code

To create the web service thin client, follow these steps:

1. Create a Java project by selecting **File** → **New** → **Project** → **Java Project**.
2. For the Project name, enter `RAD8WebServiceThinClient` and click **Finish**.
3. In the Java EE perspective: Services view, expand **JAX-WS**, right-click **RAD8WebServiceWeb: {http://...}BankService**, and select **Generate** → **Client**.
4. Complete the following actions:
 - a. Keep the slider at the **Deploy client** level. Click the hyperlink **Client project**.
 - b. In the Specify Client Project Settings window (Figure 14-31), for the Client project, select **RAD8WebServiceThinClient** and click **OK**.

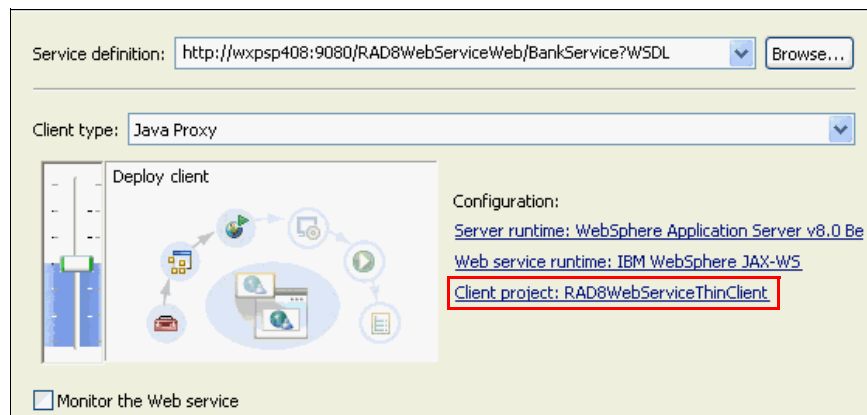


Figure 14-31 Generating a thin client

- c. Click **Finish** to generate the helper classes and WSDL file into the client project.
5. After the code generation, switch to the **Enterprise Explorer** view. Right-click **RAD8WebServiceThinClient** and select **Properties**. Select **Java Build Path**. Click the **Libraries** tab (Figure 14-32 on page 736).

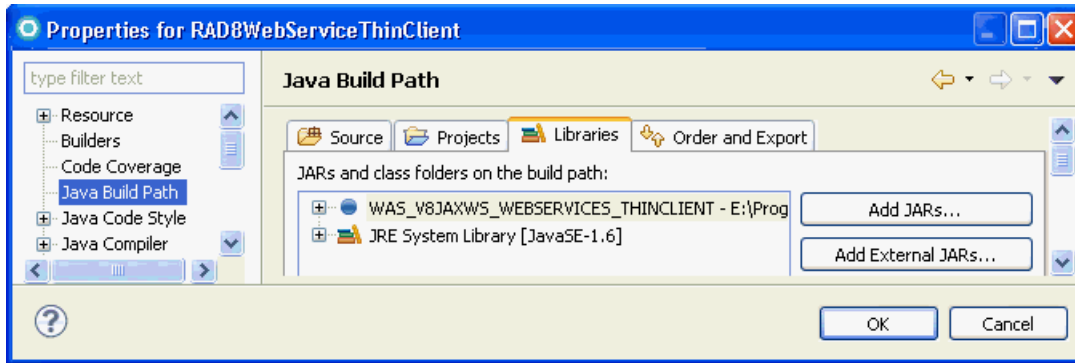


Figure 14-32 Web service thin client build path

Notice that the thin client only requires the Java Runtime Environment (JRE) and a thin client JAR file. The wizard adds a class path variable `WAS_V8JAXWS_WEBSERVICES_THINCLIENT`, which points to the `com.ibm.jaxws.thinclient_8.0.0.jar` file.

Creating the client class to invoke the web service

To invoke the web service, create a Java class:

1. Right-click **RAD8WebServiceThinClient** and select **New** → **Class**.
2. For the Package name, type `itso.rad8.bank.test`, and for the Class name, type `WSThinClientTest`. Select **public static void main(String[] args)** and click **Finish**.
3. Copy and paste the code from `WSThinClientTest.java` in `C:\7835code\webservices\thinclient` (Example 14-15).

Example 14-15 `WSThinClientTest`

```
package itso.rad8.bank.test;

import itso.rad8.bank.model.simple.BankPortProxy;
import itso.rad8.bank.model.simple
    .CustomerDoesNotExistException_Exception;
import java.util.Scanner;

public class WSThinClientTest {

    public static void main(String[] args) {
        try {
            Scanner scanner = new Scanner(System.in);
            BankPortProxy proxy = new BankPortProxy();
```

```

        System.out.println
            ("Please enter customer's social security number: ");
        String ssn = scanner.next();
        System.out.println("Customer's name is " +
            proxy.retrieveCustomerName(ssn));
    } catch (CustomerDoesNotExistException_Exception e) {
        System.out.println("The customer does not exist!");
    }
}
}
}

```

Notice how easy it is to invoke the web service. You instantiate the proxy class (BankPortProxy) and call the method (retrieveCustomerName) in the proxy.

4. Right-click **WSThinClientTest.java** and select **Run As** → **Java Application**.
5. When prompted in the console, for the customer's Social Security number, type 111-11-1111, and the customer's name is displayed:

```

Retrieving document at
'file:/C:/workspaces/WebServices/RAD8WebServiceThinClient/bin/META-INF/wsdl/'.
Retrieving schema at 'BankService_schema1.xsd', relative to
'file:/C:/workspaces/WebServices/RAD8WebServiceThinClient/bin/META-INF/wsdl/'.
Please enter customer's social security number:
111-11-1111
Customer's name is Mr. Henry Cui

```

14.11 Creating asynchronous web service clients

An asynchronous invocation of a web service sends a request to the service endpoint and then immediately returns control to the client program without waiting for the response to return from the service. JAX-WS asynchronous web service clients consume web services using either the polling approach or the callback approach:

- ▶ Using a *polling model*, a client can issue a request and receive a response object that is polled to determine if the server has responded. When the server responds, the actual response is retrieved.
- ▶ Using the *callback model*, the client provides a callback handler to accept and process the inbound response object. The handleResponse method of the handler is called when the result is available.

Both the polling and callback models enable the client to focus on continuing to process work without waiting for a response to return, while providing for a more dynamic and efficient model to invoke web services.

14.11.1 Polling client

Using the polling model, a client can issue a request and receive a response object that can subsequently be polled to determine if the server has responded. When the server responds, the actual response can then be retrieved. The response object returns the response content when the `get` method is called. The client receives an object of type `javax.xml.ws.Response` from the `invokeAsync` method. That `Response` object is used to monitor the status of the request to the server, determine when the operation has completed, and to retrieve the response results.

To create an asynchronous web service client using the polling model, follow these steps:

1. In the Java EE perspective: Services view, expand **JAX-WS**, right-click **RAD8WebServiceWeb: {http://...}BankService**, and select **Generate** → **Client**.
2. Keep the slider at the **Deploy client** level. Click the hyperlink **Client project**. In the Specify Client Project Settings window, select **RAD8WebServiceThinClient** and click **OK**. Click **Next**.
3. In the WebSphere JAX-WS Web Service Client Configuration window (Figure 14-33 on page 739), select **Enable asynchronous invocation for generated client** and click **Finish**.

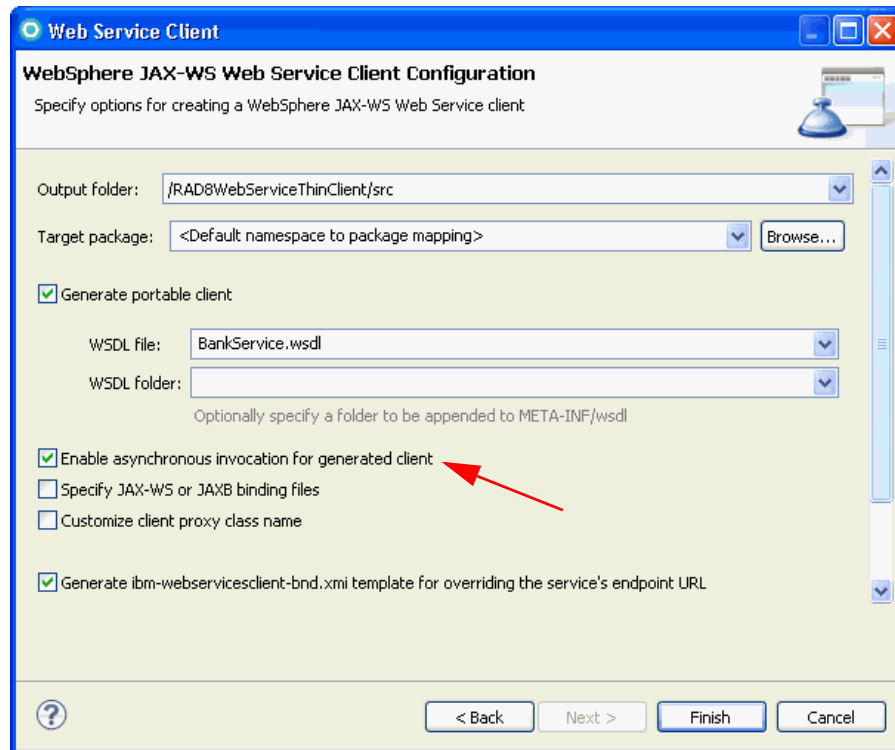


Figure 14-33 Selecting Enable asynchronous invocation for generated client

4. After the code generation, open **BankPortProxy.java** (Example 14-16). For each method in the web service, two additional methods are created, the *polling* and *callback* methods, which allow the client to function asynchronously. The `retrieveCustomerNameAsync` method that returns a `Response` is used for polling. The method that returns `Future` is used for callback.

Example 14-16 BankPortProxy asynchronous methods

```
public Response<RetrieveCustomerNameResponse>
    retrieveCustomerNameAsync(String ssn) {
    return
_getDescriptor().getProxy().retrieveCustomerNameAsync(ssn);
}

public Future<?> retrieveCustomerNameAsync(String ssn,
    AsyncHandler<RetrieveCustomerNameResponse>
    asyncHandler) {
    return _getDescriptor().getProxy().retrieveCustomerNameAsync
```

```
        (ssn, asyncHandler);
    }
}
```

5. Create a new class called `BankPollingClient` in the `itso.rad8.bank.test` package. Copy and paste the code from `C:\7835code\webservices\thinclient` (Example 14-17).

Example 14-17 BankPollingClient

```
package itso.rad8.bank.test;

import itso.rad8.bank.model.simple.BankPortProxy;
import itso.rad8.bank.model.simple.RetrieveCustomerNameResponse;
import java.util.concurrent.ExecutionException;
import javax.xml.ws.Response;

public class BankPollingClient {

    public static void main(String[] args) {
        try {
            BankPortProxy proxy = new BankPortProxy();
            Response<RetrieveCustomerNameResponse> resp =
                proxy.retrieveCustomerNameAsync("111-11-1111");
            // Poll for the response.
            while (!resp.isDone()) {
                // You can do some work that does not depend on the
customer
                name being available
                // For this example, we just check if the result is
available
                every 0.2 seconds.
                System.out.println
                    ("retrieveCustomerName async still not complete.");
                Thread.sleep(200);
            }
            RetrieveCustomerNameResponse rcnr = resp.get();
            System.out.println
                ("retrieveCustomerName async invocation
complete.");
            System.out.println("Customer's name is " +
                rcnr.getCustomerFullName());
        } catch (InterruptedException e) {
            System.out.println(e.getCause());
        } catch (ExecutionException e) {
            System.out.println(e.getCause());
        }
    }
}
```



```
    }  
  }  
}
```

6. Right-click **BankPollingClient.java** and select **Run As** → **Java Application**. The output is written to the console:

```
Retrieving document at  
'file:/C:/workspaces/WebServices/RAD8WebServiceThinClient/bin/META-INF/wsdl/'.  
Retrieving schema at 'BankService_schema1.xsd', relative to  
'file:/C:/workspaces/WebServices/RAD8WebServiceThinClient/bin/META-INF/wsdl/'.  
retrieveCustomerName async still not complete.  
retrieveCustomerName async still not complete.  
retrieveCustomerName async still not complete.  
retrieveCustomerName async still not complete.  
retrieveCustomerName async still not complete.  
retrieveCustomerName async invocation complete.  
Customer's name is Mr. Henry Cui
```

From the results, you can see that the asynchronous call allows you to perform other work while waiting for the response from the server. Eventually, you can obtain the results of the invocation.

14.11.2 Callback client

To implement an asynchronous invocation that uses the callback model, the client provides an `AsyncHandler` callback handler to accept and process the inbound response object. The client callback handler implements the `javax.xml.ws.AsyncHandler` interface, which contains the application code that is run when an asynchronous response is received from the server.

The `AsyncHandler` interface contains the `handleResponse(Response)` method that is called after the run time has received and processed the asynchronous response from the server. The response is delivered to the callback handler in the form of a `javax.xml.ws.Response` object. The response object returns the response content when the `get` method is called.

Additionally, if an error was received, an exception is returned to the client during that call. The response method is then invoked according to the threading model used by the executor method, `java.util.concurrent.Executor`, on the client's `javax.xml.ws.Service` instance that was used to create the dynamic proxy or dispatch client instance. The executor is used to invoke any asynchronous

callbacks registered by the application. Use the `setExecutor` and `getExecutor` methods to modify and retrieve the executor configured for the service.

To create an asynchronous web service client using the callback model, follow these steps:

1. Create the callback handler class `RetrieveCustomerCallbackHandler` in the `itso.rad8.bank.test` package. Copy and paste the code from `C:\7835code\webservices\thinclient` (Example 14-18).

Example 14-18 RetrieveCustomerCallbackHandler

```
package itso.rad8.bank.test;

import itso.rad8.bank.model.simple.RetrieveCustomerNameResponse;
import java.util.concurrent.ExecutionException;
import javax.xml.ws.AsyncHandler;
import javax.xml.ws.Response;

public class RetrieveCustomerCallbackHandler implements
           AsyncHandler<RetrieveCustomerNameResponse> {
    private String customerFullName;

    public void handleResponse(Response<RetrieveCustomerNameResponse>
resp){
        try {
            RetrieveCustomerNameResponse rcnr = resp.get();
            customerFullName = rcnr.getCustomerFullName();
        } catch (ExecutionException e) {
            System.out.println(e.getCause());
        } catch (InterruptedException e) {
            System.out.println(e.getCause());
        }
    }

    public String getResponse() {
        return customerFullName;
    }
}
```

2. Create the `BankCallbackClient` callback client class in the `itso.rad8.bank.test` package. Copy and paste the code from `C:\7835code\webservices\thinclient` (Example 14-19).

Example 14-19 BankCallbackClient

```
package itso.rad8.bank.test;
```

```

import itso.rad8.bank.model.simple.BankPortProxy;
import java.util.concurrent.Future;

public class BankCallbackClient {

    public static void main(String[] args) throws Exception {
        BankPortProxy proxy = new BankPortProxy();
        // Set up the callback handler.
        RetrieveCustomerCallbackHandler callbackHandler =
            new RetrieveCustomerCallbackHandler();
        // Make the Web service call.
        Future<?> response = proxy.retrieveCustomerNameAsync
            ("111-11-1111", callbackHandler);
        System.out.println("Wait 5 seconds.");
        // Give the callback handler a chance to be called.
        Thread.sleep(5000);
        System.out.println("Customer's full name is "
            + callbackHandler.getResponse() + ".");
        System.out.println("RetrieveCustomerName async end.");
    }
}

```

3. Right-click **BankCallbackClient.java** and select **Run As** → **Java Application**. The output is written to the console:

```

Retrieving document at
'file:/C:/workspaces/WebServices/RAD8WebServiceThinClient/bin/META-INF/wsdl/'.
Retrieving schema at 'BankService_schema1.xsd', relative to
'file:/C:/workspaces/WebServices/RAD8WebServiceThinClient/bin/META-INF/wsdl/'.
Wait 5 seconds.
Customer's full name is Mr. Henry Cui.
RetrieveCustomerName async end.

```

14.11.3 Asynchronous message exchange client

By default, asynchronous client invocations do not have asynchronous behavior of the message exchange pattern on the wire. The programming model is asynchronous; however, the exchange of request or response messages with the server is not asynchronous. IBM has provided a feature that goes beyond the JAX-WS specification to provide the asynchronous message exchange support.

In the asynchronous message exchange case, the client listens on a separate HTTP channel to receive the response messages from a service-initiated HTTP

channel. The client uses WS-Addressing to provide the ReplyTo endpoint reference (EPR) value to the service. The service initiates a connection to the ReplyTo EPR to send a response. To use an asynchronous message exchange, the `com.ibm.websphere.webservices.use.async.mep` property must be set on the client request context with a boolean value of `true`. When this property is enabled, the messages exchanged between the client and server differ from messages exchanged synchronously.

To create an asynchronous message exchange client, follow these steps:

1. Create the `BankCallbackMEPClient` class in the `itso.rad8.bank.test` package. Copy and paste the code from `C:\7835code\webservices\thinclient` (Example 14-20).

Example 14-20 BankCallbackMEPClient

```
package itso.rad8.bank.test;

import itso.rad8.bank.model.simple.BankPortProxy;
import java.util.concurrent.Future;
import javax.xml.ws.BindingProvider;

public class BankCallbackMEPClient {

    public static void main(String[] args) throws Exception {
        BankPortProxy proxy = new BankPortProxy();
        //proxy._getDescriptor().setEndpoint

        ("http://localhost:11487/RAD75WebServiceWeb/BankService");
        // setup the property for asynchronous message exchange
        BindingProvider bp = (BindingProvider)
            proxy._getDescriptor().getProxy();
        bp.getRequestContext().put
            ("com.ibm.websphere.webservices.use.async.mep",
Boolean.TRUE);
        // Set up the callback handler.
        RetrieveCustomerCallbackHandler callbackHandler =
            new RetrieveCustomerCallbackHandler();
        // Make the Web service call.
        Future<?> response = proxy.retrieveCustomerNameAsync
            ("111-11-1111", callbackHandler);
        System.out.println("Wait 5 seconds.");
        // Give the callback handler a chance to be called.
        Thread.sleep(5000);
        System.out.println("Customer's full name is "
            + callbackHandler.getResponse() + ".");
    }
}
```

```

        System.out.println("RetrieveCustomerName async end.");
    }
}

```

2. Right-click **BankCallbackMEPClient.java** and select **Run As → Java Application**. The output is written to the console:

```

Retrieving document at
'file:/C:/workspaces/WebServices/RAD8WebServiceThinClient/bin/META-INF/wsd1/'.
Retrieving schema at 'BankService_schema1.xsd', relative to
'file:/C:/workspaces/WebServices/RAD8WebServiceThinClient/bin/META-INF/wsd1/'.
[WASHttpAsyncResponseListener] listening on port 4553
Wait 5 seconds.
Customer's full name is Mr. Henry Cui.
RetrieveCustomerName async end.

```

Notice the new line in the WebSphere Application Server Console:

```

[10/22/10 14:43:56:359 PDT] 00000024 WSChannelFram A   CHF0019I:
The Transport Channel Service has started chain
HttpOutboundChain:wxpsp408.rcsn1.ams.nl.ibm.com:4553.

```

3. Optional: If you want to see the SOAP request message, activate the comment line:

```

proxy._getDescriptor().setEndpoint

```

```

("http://wxpsp408:12036/RAD75WebServiceWeb/BankService");

```

You must supply your host name instead of **wxpsp408** and the port **12036** must match the port of the TCP/IP Monitor.

4. Run the application again. Example 14-21 shows the SOAP request.

Example 14-21 SOAP request for asynchronous message exchange

```

--MIMEBoundary_3028c58b531c6cb4c683e77e89daa042d5c97cdfa680727e
Content-Type: application/xop+xml; charset=UTF-8; type="text/xml"
Content-Transfer-Encoding: binary
Content-ID:
<0.2028c58b531c6cb4c683e77e89daa042d5c97cdfa680727e@apache.org>

```

```

<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
<wsa:To>http://wxpsp408:12036/RAD8WebServiceWeb/BankService</wsa:To>
<wsa:ReplyTo>

```

```
<wsa:Address>http://wxpsp408.rcsn1.ams.nl.ibm.com:4991/axis2/service
s/BankService.BankPort</wsa:Address>
</wsa:ReplyTo>
<wsa:MessageID>urn:uuid:d0da36f9-9623-4d6e-8249-d57a118c43e2</wsa:Me
ssageID>
<wsa:Action>urn:getCustomerFullName</wsa:Action>
</soapenv:Header>
<soapenv:Body>
<a:RetrieveCustomerName
xmlns:a="http://simple.model.bank.rad8.itso/">
<ssn>111-11-1111</ssn>
</a:RetrieveCustomerName>
</soapenv:Body>
</soapenv:Envelope>
--MIMEBoundary_3028c58b531c6cb4c683e77e89daa042d5c97cdfa680727e--
```

Because the client listens on a separate HTTP channel to receive the response messages from a service-initiated HTTP channel, the TCP/IP Monitor is unable to capture the SOAP response.

The completed Thin Client project is available in this file:

C:\7835codesolution\webservices\RAD8WebServiceThinClient.zip

14.12 Creating web services from an EJB

You can generate EJB web services by using either the Web Service wizard or annotations.

In this section, you create a JAX-WS web service from an EJB session bean using annotations:

1. Expand the EJB project **RAD8WebServiceEJB** and open the **SimpleBankFacadeBean** (in `ejbModule/itso.rad8.bank.ejb.facade`).
2. Add the `@WebService` annotation on the line over the `@Stateless` annotation (Example 14-22). Press `Ctrl+Shift+O` to resolve the import.

Example 14-22 Annotating a stateless session EJB

```
@WebService
@Stateless
public class SimpleBankFacadeBean implements
SimpleBankFacadeBeanLocal {
```

.....

3. Wait for the RAD8WebServiceEAR application to publish on the server (or force a manual publish). Notice that a new web service named RAD8WebServiceEJB is added in the Services view under JAX-WS.

An HTTP router module is required to allow the transport of SOAP messages over the HTTP protocol.

4. In the Services view (Figure 14-34), right-click the new **RAD8WebServiceEJB** and select **Create Router Modules (EndpointEnabler)**.

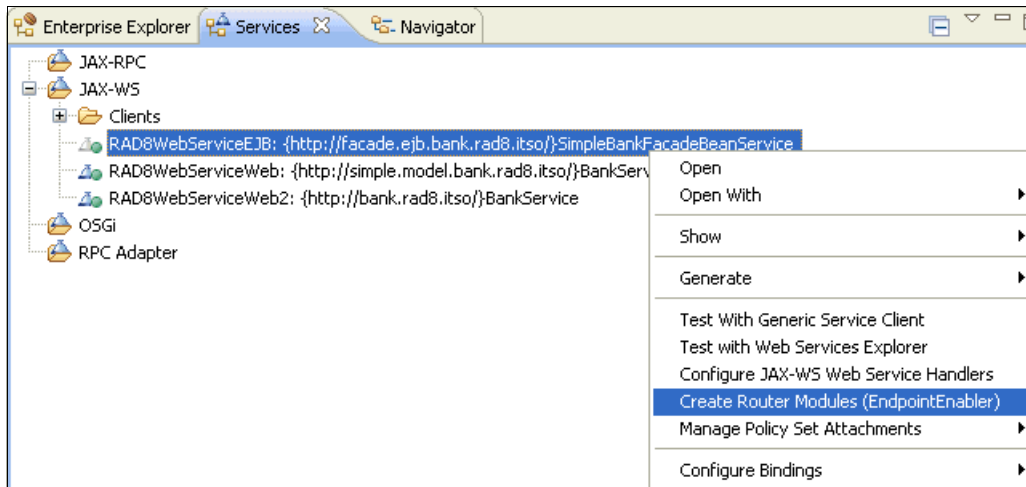


Figure 14-34 Selecting Create Router Modules

- In the Create Router Project window, accept **HTTP** as the default EJB web service binding (Figure 14-35). Although two EJB bindings are listed, HTTP and JMS, for this example, we use SOAP over HTTP. Click **Finish**.

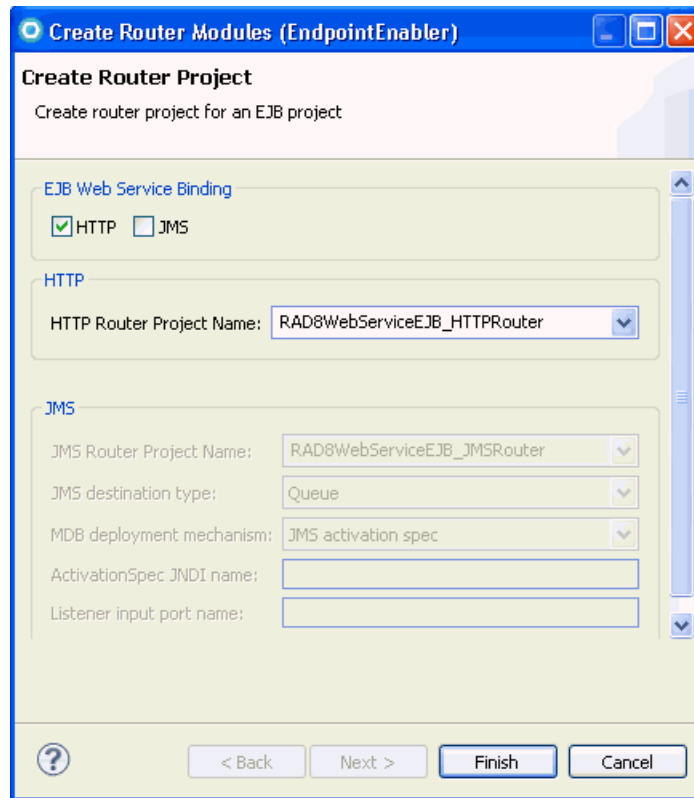


Figure 14-35 Create Router Project window

- Open the deployment descriptor of the `RAD8WebServiceEJB_HTTPRouter` project to see the generated servlet.
- In the Services view, right-click **RAD8WebServiceEJB** and select **Test with Generic Service Client**.
- Select the **getAccountBalance** operation and select the **SimpleBankFacadeBeanPort** under it.
- In the **Edit Data** → **Message** → **Form** tab, expand **getAccountNumber**.
- Select **arg0** (which represents the Account Number input parameter).
- For the Account number, type 001-999000777.

12. Click **Invoke**. You can see the result of the web service call:

```
getAccountBalanceResponse  
12345.67
```

All of the projects that we have completed so far are available in this file:

C:\7835codesolution\webservices\RAD8WebServiceEJBService.zip

14.13 Creating a top-down web service from a WSDL

When creating a web service using a top-down approach, first you design the implementation of the web service by creating a WSDL file. You can do this by using the WSDL editor. You can then use the Web Service wizard to create the web service and skeleton Java classes to which you can add the required code. The top-down approach is the recommended way of creating a web service.

14.13.1 Designing the WSDL by using the WSDL editor

In this section, you create a WSDL with two operations: `getAccount` (using an account ID to retrieve an account) and `getCustomer` (using a customer ID to retrieve a customer).

By using the WSDL editor, you can easily and graphically create, modify, view, and validate WSDL files. To create a WSDL, follow these steps:

1. Create a dynamic web project to host the new web service:
 - Web project: `RAD8TopDownBankWS`
 - EAR project: `RAD8TopDownBankEAR`
2. Create a WSDL file:
 - a. Right-click **WebContent** (in `RAD8TopDownBankWS`).
 - b. Select **New** → **Other**.
 - c. In the New Wizard, select **Web services** → **WSDL**.
 - d. Click **Next**.
 - e. Change the File name to `BankWS.wsdl` and click **Next**.
 - f. In the Options window (Figure 14-36 on page 750), ensure that the default options **Create WSDL Skeleton** and **document literal** are selected. Then click **Finish**.

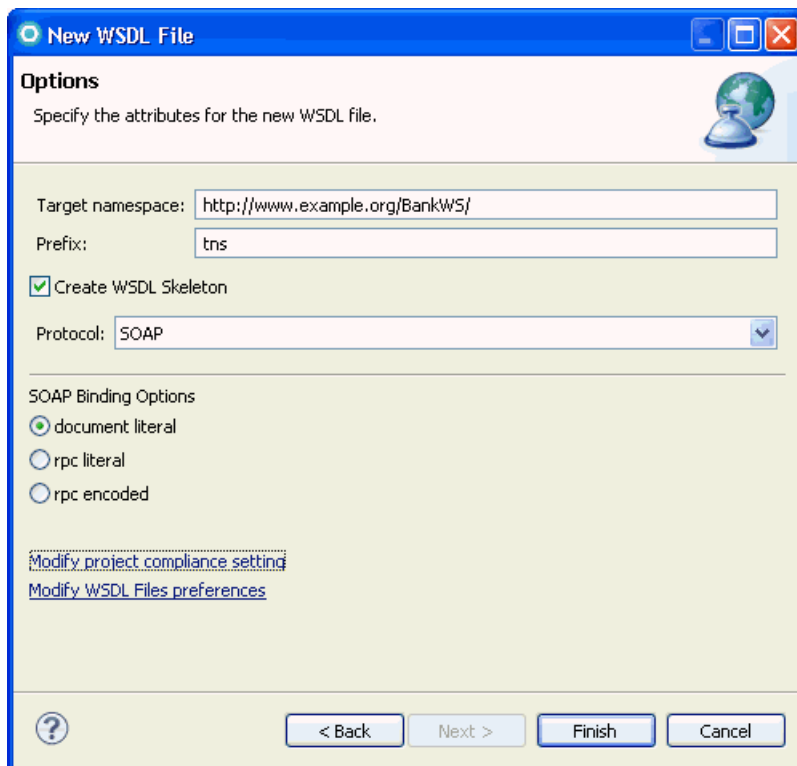


Figure 14-36 New WSDL File wizard: Options window

3. When the WSDL editor opens with the new WSDL file, select the **Design** tab:
 - a. In the WSDL editor, for View (in the upper-right corner), select **Advanced**.
 - b. Select the **Properties** view. Now you are ready to edit the WSDL file (Figure 14-37 on page 751).

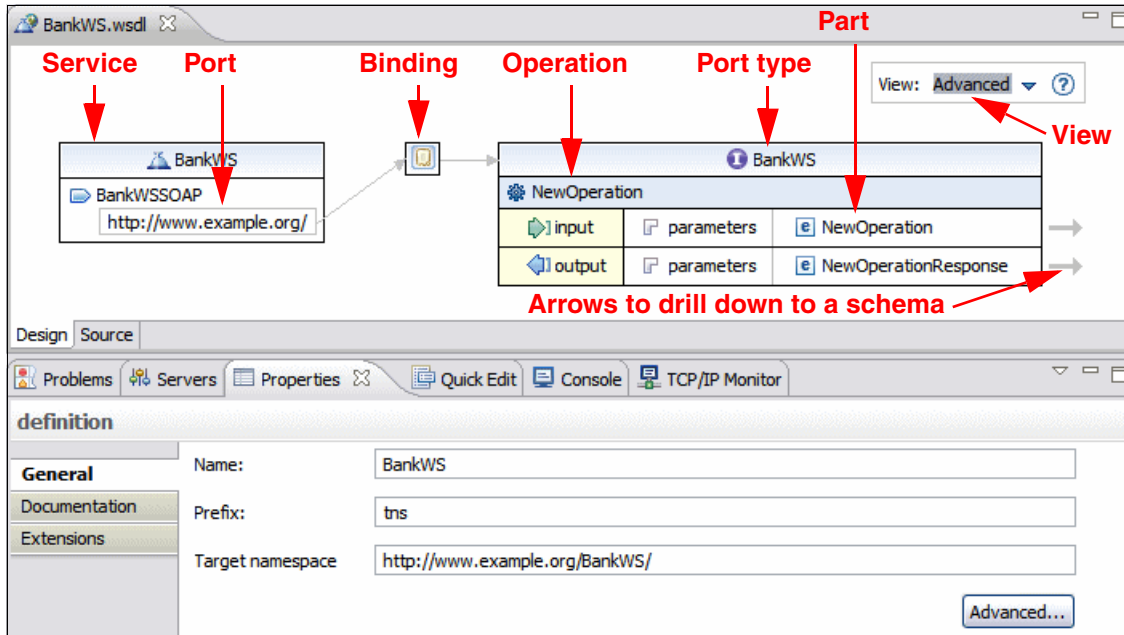


Figure 14-37 WSDL editor

Editing the WSDL file

Define the operations and parameters of the WSDL:

1. Change the operation name by double-clicking **NewOperation** and overtyping the name with `getAccount`. (You can also select the operation and change the name in the Properties view.)
As a result, the input parameter changes to `getAccount` and the output parameter changes to `getAccountResponse`.
2. Add a new operation. In the Design view, right-click the port type **BankWS**, select **Add Operation**, and name the operation `getCustomer`.
As a result, the input parameter is set to `getCustomer` and the output parameter changes to `getCustomerResponse`.
3. To change the input type of the WSDL operation `getAccount`, click the **arrow** to the right of the input parameter to drill down to the schema.

4. When the Inline Schema editor opens, select the **Design** tab and switch to the **Detailed** view (Figure 14-38).

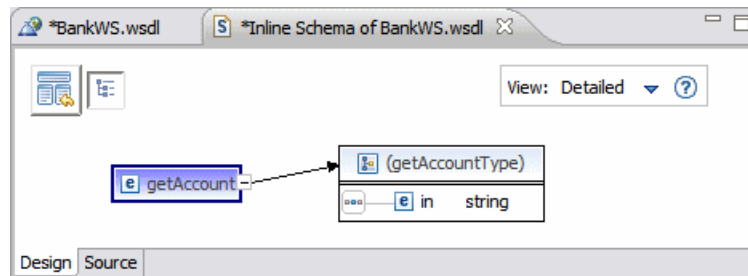



Figure 14-38 Schema editor: Start

- a. Select the **in** element, and in the Properties view, change the name to `accountId` (leave the type as `xsd:string`).
- b. Click the **Show schema index view** icon () in the upper-left corner to see all the directives, elements, types, attributes, and groups in the WSDL (Figure 14-39).

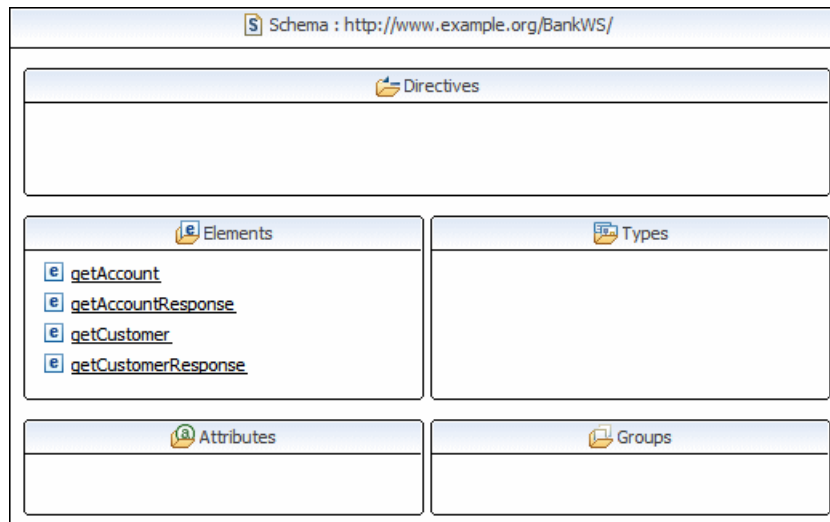



Figure 14-39 Schema editor: Index view

- c. In the Types category, right-click and select **Add Complex Type**. Change the name to `Account`. Follow these steps:
 - i. Right-click **Account** and select **Add Sequence**.
 - ii. You are taken out of the Index view and back into the Online Schema.

- iii. Right-click **Account** and select **Add Element**.
- iv. Change the Element name to id.
- v. Right-click the content model object () and select **Add Element**.
- vi. Change the name to balance.
- vii. Click **Browse** and select the type as **decimal** (Figure 14-40).

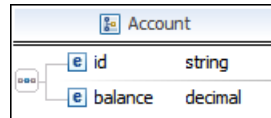



Figure 14-40 Schema editor: Account

- d. Click the **Show schema index view** icon () in the upper-left corner. Perform these steps:
 - i. In the Types section, right-click and select **Add Complex Type**.
 - ii. Change the name to Customer.
 - iii. Right-click **Customer** and select **Add Sequence**. Add four elements: **ssn**, **firstName**, **lastName**, and **title**, all of type string (Figure 14-41).

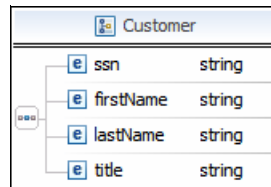



Figure 14-41 Schema editor: Customer

- e. Click the **Show schema index view** icon () in the upper-left corner. Add the global elements for the complex types:
 - i. Right-click the **Elements** category and click **Add Element**.
 - ii. Change the element name to Account.
 - iii. Right-click **Account** and select **Set Type**.
 - iv. Click **Browse** and select **Account**.
 - v. Add the global element **Customer**.
 - vi. Set type to Customer (Figure 14-42 on page 754).

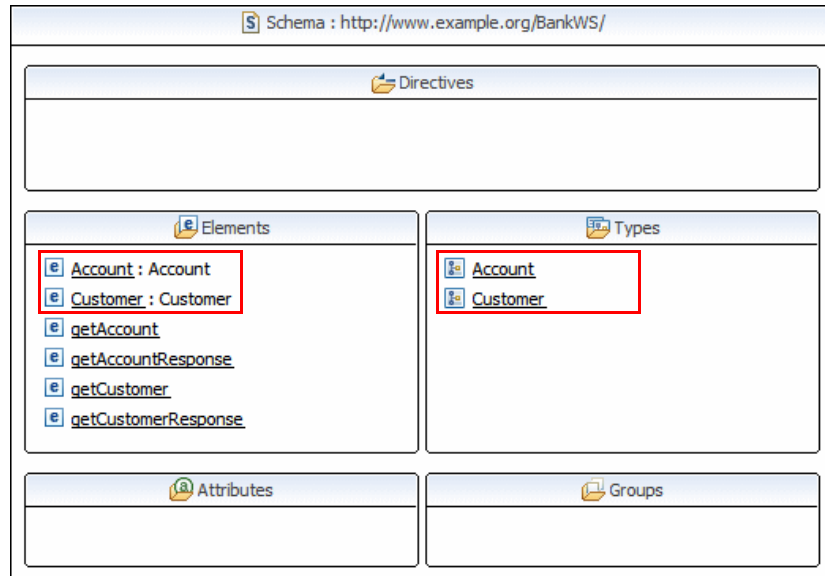


Figure 14-42 Schema editor: Global elements

5. In the WSDL editor, click the **arrow** to the right of the **getAccountResponse** element to drill down to the schema:
 - a. Right-click the **out** element and select **Set Type**.
 - b. Click **Browse** and select **Account** (Figure 14-43).

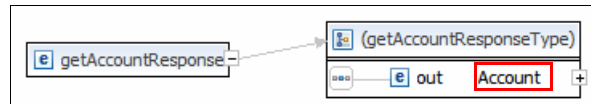


Figure 14-43 Schema editor: Output message

6. In the WSDL editor, click the **arrow** to the right of the **getCustomer** element to drill down to the schema. Change the element name from `in` to `customerId`.
7. In the WSDL editor, click the **arrow** to the right of the **getCustomerResponse** element to drill down to the schema:
 - a. Right-click the **out** element and select **Set Type**.
 - b. Click **Browse** and select **Customer**.
8. In the WSDL editor, right-click the **binding** icon (see Figure 14-37 on page 751). Select **Generate Binding Content**.

- In the Binding wizard (Figure 14-44), select **Overwrite existing binding information** and click **Finish**.

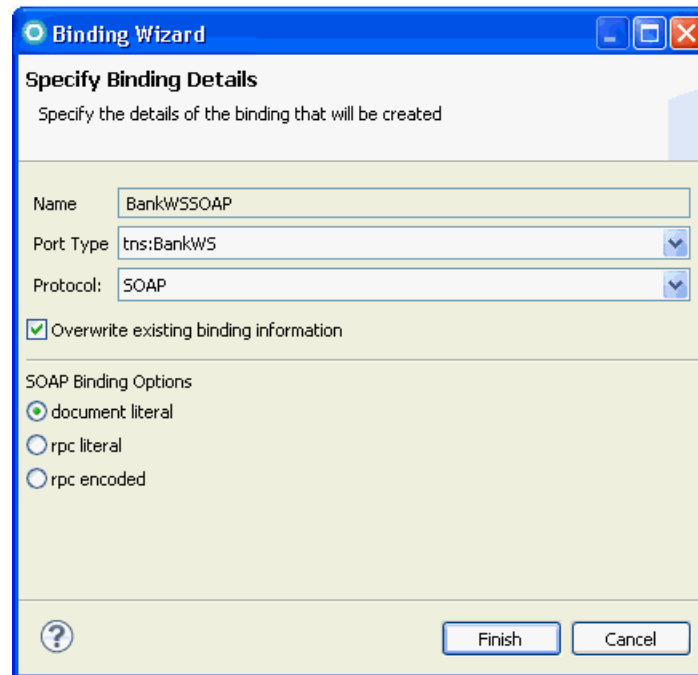


Figure 14-44 Specify Binding Details wizard

- Save the schema and WSDL file (Figure 14-45).

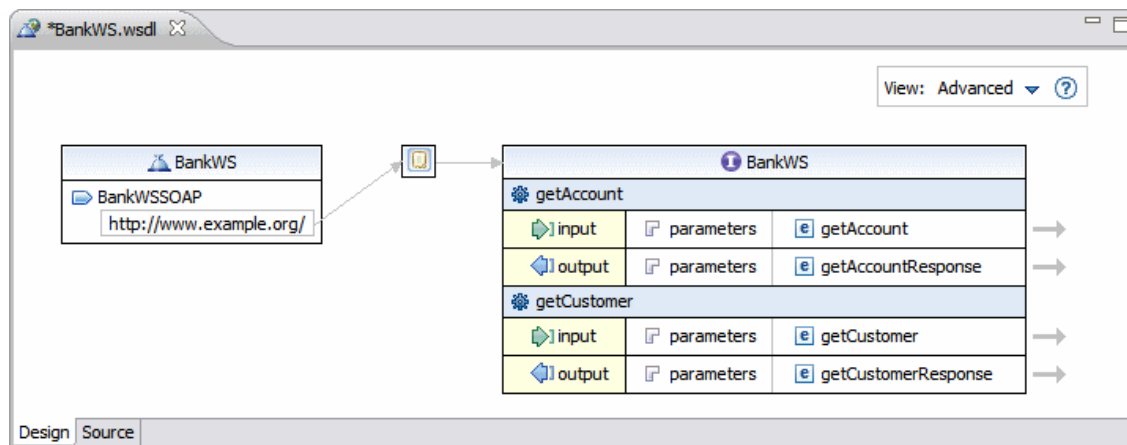


Figure 14-45 BankWS.wsdl

11. In the Enterprise Explorer, right-click **BankWS.wsdl** and select **Validate**. In the window that opens showing a message that confirms that there are no errors or warnings, click **OK**.

If you have a problem when creating the WSDL file, you can import the BankWS.wsdl file from the C:\7835code\webservices\topdown directory.

14.13.2 Generating the skeleton JavaBean web service

To generate a skeleton JavaBean web service from a WSDL, follow these steps:

1. Right-click **BankWS.wsdl** and select **Web Services** → **Generate Java bean skeleton**.
2. Keep the slider at the **Start service** level. Notice that the code is generated into the RAD8TopDownBankWS project. Click **Finish**.

After the code generation, the skeleton class BankWSSOAPImpl.java opens in the Java editor. Notice the annotation of the class:

```
@javax.jws.WebService (endpointInterface="org.example.bankws.BankWS",
targetNamespace="http://www.example.org/BankWS/", serviceName="BankWS",
portName="BankWSSOAP")
```

The RAD8TopDownBankEAR is deployed to the server, and the web service is displayed in the Services view.

Implementing the generated JavaBean skeleton

You must provide the business logic for the generated JavaBean skeleton. Use the simple implementation that is shown in Example 14-23.

Example 14-23 Implementation of the generated JavaBean skeleton

```
package org.example.bankws;

import java.math.BigDecimal;

@javax.jws.WebService (endpointInterface="org.example.bankws.BankWS",
    targetNamespace="http://www.example.org/BankWS/",
    serviceName="BankWS",
    portName="BankWSSOAP")
public class BankWSSOAPImpl{

    public Account getAccount(String accountId) {
        Account account = new Account();
        account.setId(accountId);
        account.setBalance(new BigDecimal(1000.00));
    }
}
```



```
        return account;
    }

    public Customer getCustomer(String customerId) {
        Customer customer = new Customer();
        customer.setSsn(customerId);
        customer.setFirstName("Henry");
        customer.setLastName("Cui");
        customer.setTitle("Mr.");
        return customer;
    }
}
```

After you save the .java file, wait for the Server to republish or publish manually.

14.13.3 Testing the generated web service

To test the web service, use the Generic Service Client.

BankWS.wsdl file: You cannot use the BankWS.wsdl file in the WebContent folder to test the web service. The web service endpoint was set to `http://www.example.org/` when we created this WSDL. The dynamic WSDL loading from the Services view sets the endpoint correctly.

Follow these steps:

1. To test the web service, in the Services view, expand **JAX-WS**, right-click **RAD8TopDownBankWS**, and select **Test with Generic Service Client**.
2. Test the **getCustomer** and **getAccount** operations. You see the correct result in the Generic Service Client.

The Top Down Project is available in this file:

`C:\7835codesolution\webservices\RAD8TopDownWebService.zip`

Explore: After you create a web service, you might want to make the following changes to it:

- ▶ For example, you might want to add a new WSDL operation `getBalanceByAccountId` to the WSDL. After the WSDL is changed, you have to regenerate the web service code. The existing business logic might be wiped out.
- ▶ To retain your changes while updating the web service, you can use the *skeleton merge* feature. With this feature, you can regenerate the web service while keeping your changes intact. Select **Window** → **Preferences** → **Web Services** → **Resource Management** → **Merge generated skeleton file**.

14.14 Creating web services with Ant tasks

If you prefer not to use the Web Service wizard, you can use Apache Ant tasks to create a web service using the IBM WebSphere JAX-WS runtime environment. The Ant tasks support creating web services using both the top-down and bottom-up approaches. After you have created a web service, you can then deploy it to a server, test it, and publish it as a business entity or business service. Additionally, you can create web service clients using the Ant tasks.

14.14.1 Creation procedure

In this section, we use Ant tasks to automate the top-down code generation process that we did in the last section:

1. Remove `RAD8WebServiceEAR` from the server while we make modifications.
2. Create a dynamic web project to host the web service generated by Ant tasks: `RAD8WebServiceAnt` in `RAD8WebServiceEAR`.
3. Copy the `BankWS.wsdl` file from the `RAD8TopDownBankWS/WebContent` folder to the `RAD8WebServiceAnt` folder (not under `WebContent`).
4. Right-click **RAD8WebServiceAnt** and select **New** → **Other**. Perform these steps:
 - a. In the New wizard, select **Web Services** → **Ant Files**.
 - b. Click **Next**.
5. In the Create Ant Files window (Figure 14-46 on page 759), perform these steps:
 - a. For Web service run time, select **IBM WebSphere JAX-WS**.

- b. For Web service type, select **Top down Java bean Web Service**.
- c. Click **Finish**.

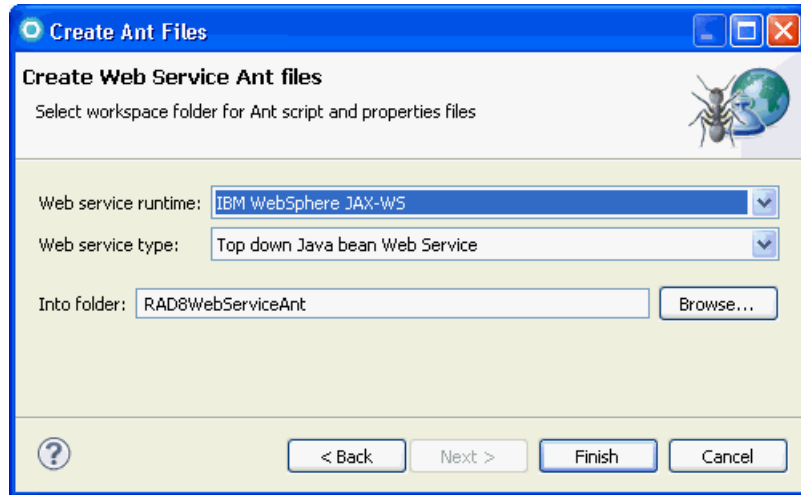


Figure 14-46 Create Web Service Ant files window

A `wsgenTemplates` folder is created with the `was_jaxws_tdjava.xml` and `was_jaxws_tdjava.properties` files.

6. Open the **was_jaxws_tdjava.properties** file and perform these steps:
 - a. Change `InitialSelection=` to `InitialSelection=/RAD8WebServiceAnt/BankWS.wsdl`.
 - b. Make sure that the `Service.ServerId` line is equal to `Service.ServerId=com.ibm.ws.ast.st.v8.server.base`.
 - c. Save and close the file.

14.14.2 Running the web service Ant task

Perform these steps to run the Ant task:

1. Right-click **was_jaxws_tdjava.xml** and select **Run As** → **Ant Build**.
2. The Configuration Editor opens.
3. In the Edit Configuration window, perform these steps:
 - a. Select the **JRE** tab.
 - b. Select **Run in the same JRE as the workspace**. This option is required; otherwise, the ANT tasks present in the `was_jaxws_tdjava.xml` file cannot be found at run time.

4. Click **Apply** and then click **Run**.

The web service is generated:

- ▶ The web service artifacts are generated in the Java Resources folder.
- ▶ You can implement the generated skeleton (`BankWSSOAPImpl`) and test the web service, as we did in the last section.

All projects developed so far are available in this file:

`C:\7835codesolution\webservices\RAD8ANTWebService.zip`

14.15 Sending binary data using MTOM

SOAP Message Transmission Optimization Mechanism (MTOM) is a standard that is developed by the World Wide Web Consortium (W3C). MTOM describes a mechanism for optimizing the transmission or wire format of a SOAP message by selectively re-encoding portions of the message while presenting an XML Information Set (Infoset) to the SOAP application.

MTOM uses the XML-binary Optimized Packaging (XOP) in the context of SOAP and Multipurpose Internet Mail Extensions (MIME) over HTTP. XOP defines a serialization mechanism for the XML Infoset with binary content that is not only applicable to SOAP and MIME packaging, but to any XML Infoset and any packaging mechanism. It is an alternate serialization of XML that happens to look like a MIME multipart or related package, with XML documents as the root part.

That root part is similar to the XML serialization of the document, except that base64-encoded data is replaced by a reference to one of the MIME parts, which is not base64 encoded. This reference enables you to avoid the bulk and overhead in processing that are associated with encoding. Encoding is the only way that binary data can work directly with XML.

In this section, we use the top-down approach to create a JAX-WS web service to send binary attachments along with a SOAP request, and to receive binary attachments along with a SOAP response using MTOM.

The web service client sends three types of documents: Microsoft Word, image, and PDF file. We describe several ways to send the documents:

- ▶ The client uses `byte[]` to send the Word document.
- ▶ The client uses `java.awt.Image` to send the image file.
- ▶ The client uses `javax.activation.DataHandler` to send the PDF file.

After the web service receives the binary data from the client, it stores the received document on the local hard disk and then passes the same document back to the client. In an actual scenario, the provider or the consumer can send an acknowledgement message, after it receives the binary data from the other side. For our example, we want to show how to enable the MTOM on both the client and the server side in a compact example.

14.15.1 Creating a web service project and importing the WSDL

To create a web service project, follow these steps:

1. Select **File** → **New** → **Dynamic Web Project**.
2. In the window that opens, complete the following actions:
 - a. For Project Name, type RAD8WSMTOM.
 - b. For EAR Project Name, type RAD8WSMTOMEAR.
 - c. Click **Finish**.
3. Import the `c:\7835code\webservice\mtom\ProcessDocumentService.wsdl` file into the RAD8WSMTOM/WebContent folder.
4. Open the **ProcessDocumentService.wsdl** file. Look at the source, and you see the attributes that are highlighted in Example 14-24.

Example 14-24 Extract of the ProcessDocumentService.wsdl file

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xmime="http://www.w3.org/2005/05/xmlmime"
           xmlns:tns="http://mtom.rad8.ibm.com/"
           targetNamespace="http://mtom.rad8.ibm.com/"
           version="1.0">
  <xs:complexType name="sendPDFFile">
    <xs:sequence>
      <xs:element minOccurs="0" name="arg0" type="xs:base64Binary"
        xmime:expectedContentTypes="*/*/"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="sendWordFile">
    <xs:sequence>
      <xs:element minOccurs="0" name="arg0" type="xs:base64Binary"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="sendImage">
    <xs:sequence>
      <xs:element minOccurs="0" name="arg0" type="xs:base64Binary"/>
```

```
xmime:expectedContentTypes="image/jpeg"/>
  </xs:sequence>
</xs:complexType>
```

Default mapping

The default mapping for `xs:base64Binary` is `byte[]` in Java. If you want to use another mapping, you can add the `xmime:expectedContentTypes` attribute to the element containing the binary data. This attribute is defined in the <http://www.w3.org/2005/05/xmlmime> namespace and specifies the MIME types that the element is expected to contain. The setting of this attribute changes how the code generators create the JAXB class for the data. Depending on the `expectedContentTypes` value in the WSDL file, the JAXB artifacts generated are in the Java type, as described in Table 14-2.

Table 14-2 Mapping between MIME type and Java type

MIME type	Java type
image/gif	java.awt.Image
image/jpeg	java.awt.Image
text/plain	java.lang.String
text/xml	javax.xml.transform.Source
application/xml	javax.xml.transform.Source
/	javax.activation.DataHandler

Based on this table, we can make the following predictions:

- ▶ `sendWordFile` will be mapped to `byte[]` in Java.
- ▶ `sendPDFFile` will be mapped to `javax.activation.DataHandler`.
- ▶ `sendImage` will be mapped to `java.awt.Image`.

14.15.2 Generating the web service and client

To create the web service and client using the Web Service wizard, follow these steps:

1. Right-click **ProcessDocumentService.wsdl** and select **Web Services** → **Generate Java bean skeleton**.

2. When the Web Service wizard starts with the Web Services page, complete the following steps:
 - a. Select the following options for the web service:
 - i. For Server, select **WebSphere Application Server v8.0 Beta**.
 - ii. For Web service run time, select **IBM WebSphere JAX-WS**.
 - iii. For Service project, select **RAD8WSMTOM**.
 - iv. For Service EAR project, select **RAD8WSMTOMEAR**.
 - b. Select the following options for the web service client:
 - i. Move the slider to **Test client**.
 - ii. For Server, select **WebSphere Application Server v8.0 Beta**.
 - iii. For Web service run time, select **IBM WebSphere JAX-WS**.
 - iv. For Client project, select **RAD8WSMTOMClient**.
 - v. For Client EAR project, select **RAD8WSMTOMClientEAR**.

Because the web service client project is not yet in the workspace when you run the Web Service wizard, the wizard creates the project for you.
 - c. Select **Monitor the Web service** and then click **Next**.
3. In the WebSphere JAX-WS Top Down Web Service Configuration window (Figure 14-47 on page 764), perform these steps:
 - a. Select **Enable MTOM Support**.
 - b. Click **Next**.

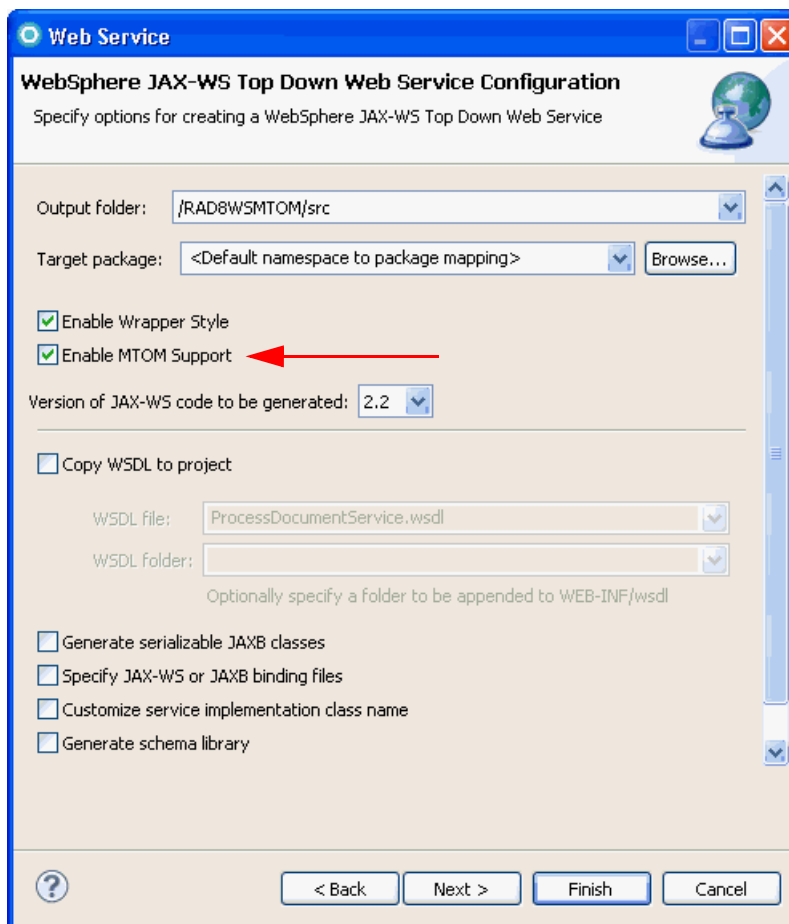


Figure 14-47 Selecting the Enable MTOM Support option

4. In the Warning message window (Figure 14-48 on page 765) that opens, click **Details** to view the complete message. Click **Ignore** to continue the code generation.

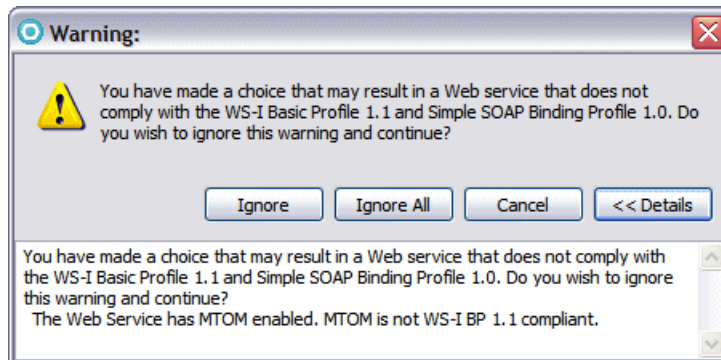


Figure 14-48 WS-I warning against MTOM

5. In the Test Web Service window, select **Next** (we will not test this way, because it is difficult to supply the required input types, such as an array of bytes).
6. In the WebSphere JAX-WS Web Service Client Configuration window, accept the defaults (**Enable MTOM** is selected) and click **Next**.
7. In the Web Service Client Test window, for the Test Facility, select **JAX-WS JSPs** and click **Finish**. The generated JavaBean skeleton and the sample JSP client open.

14.15.3 Implementing the JavaBean skeleton

Before you test the sample JSP client, you must implement the generated JavaBean skeleton. The web service stores the received document on the local hard drive and then passes the same document back to the client. Follow these steps:

1. Examine the generated skeleton class `ProcessDocumentPortBindingImpl`. We can see that `sendWordFile` is mapped to `byte[]`, `sendPDFFile` is mapped to `javax.activation.DataHandler`, and `sendImage` is mapped to `java.awt.Image`, as we expected.
2. The Wizard has inserted the annotation:

```
@javax.xml.ws.BindingType
    (value=javax.xml.ws.soap.SOAPBinding.SOAP11HTTP_MTOM_BINDING)
```

You can use the `@BindingType (javax.xml.ws.BindingType)` annotation on an endpoint implementation class to specify that the endpoint supports one of the MTOM binding types so that the response messages are MTOM-enabled. The `javax.xml.ws.SOAPBinding` class defines two constants,

SOAP11HTTP_MTOM_BINDING and SOAP12HTTP_MTOM_BINDING, that you can use for the value of the @BindingType annotation.

To enable MTOM on an endpoint, you can also place the @MTOM (javax.xml.ws.soap.MTOM) annotation on the endpoint. The @MTOM annotation has two parameters, enabled and threshold. The enabled parameter has a boolean value and indicates if MTOM is enabled for the JAX-WS endpoint. The threshold parameter has an integer value that must be greater than or equal to zero. When MTOM is enabled, any binary data whose size, in bytes, exceeds the threshold value is XML-binary Optimized Packaging (XOP)-encoded or sent as an attachment. When the message size is less than the threshold value, the message is inlined in the XML document as either base64 or hexBinary data.

The presence and value of an @MTOM annotation overrides the value of the @BindingType annotation. For example, if the @BindingType indicates that MTOM is enabled, but an @MTOM annotation is present with an enabled value of false, MTOM is not enabled.

3. Copy the code from C:\7835code\webservices\mtom and paste it to ProcessDocumentPortBindingImpl.java (Example 14-25).

Example 14-25 ProcessDocumentPortBindingImpl.java

```
package com.ibm.rad8.mtom;

import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import javax.activation.DataHandler;
import javax.imageio.ImageIO;

@javax.jws.WebService
(endpointInterface="com.ibm.rad8.mtom.ProcessDocumentDelegate",
targetNamespace="http://mtom.rad8.ibm.com/",
serviceName="ProcessDocumentService",
portName="ProcessDocumentPort")
@javax.xml.ws.BindingType
(value=javax.xml.ws.soap.SOAPBinding.SOAP11HTTP_MTOM_BINDING)
public class ProcessDocumentPortBindingImpl{

    public byte[] sendWordFile(byte[] arg0) {
        try {
```

```

        FileOutputStream fileOut = new FileOutputStream
            (new
File("C:/7835code/webservices/mtomresult/RAD-intro.doc"));
        fileOut.write(arg0);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return arg0;
}
public Image sendImage(Image arg0) {
    try {
        File file = new File
            ("C:/7835code/webservices/mntomresult/BlueHills.jpg");
        BufferedImage bi = new BufferedImage(arg0.getWidth(null),
            arg0.getHeight(null),
BufferedImage.TYPE_INT_RGB);
        Graphics2D g2d = bi.createGraphics();
        g2d.drawImage(arg0, 0, 0, null);
        ImageIO.write(bi, "jpeg", file);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return arg0;
}
public DataHandler sendPDFFile(DataHandler arg0) {
    try {
        FileOutputStream fileOut = new FileOutputStream(new File(
            "C:/7835code/webservices/mtoresult/JAX-WS.pdf"));
        BufferedInputStream fileIn = new BufferedInputStream
            (arg0.getInputStream());
        while (fileIn.available() != 0) {
            fileOut.write(fileIn.read());
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return arg0;
}
}
}

```

Examine the code in Example 14-25 on page 766, and notice the following points:

- ▶ The `sendWordFile` method takes `byte[]` as input and stores the binary data as `C:/7835code/webservices/mtomresult/RAD-intro.doc`.

- ▶ The `sendImage` method takes an image as input and stores the binary data as `C:/7835code/WebServices/mtomresult/BlueHills.jpg`.
- ▶ The `sendPDFFile` method takes a `DataHandler` as input and stores the data in `C:/7835code/WebServices/mtomresult/JAX-WS.pdf`.
- ▶ All the three methods return the received data to the client after storing it on the local drive.

14.15.4 Testing and monitoring the MTOM-enabled web service

Now it is time to see if MTOM really optimizes the transmission of the data. We use the `C:\7835code\webservices\mtomresult` output folder to store the document received by the web service `JavaBean`.

1. In the Web Services Test Client view (Figure 14-49 on page 769), complete the following actions:
 - a. In the sample JSP client, select the **sendImage** method.
 - b. Click **Browse** and navigate to `C:\7835code\webservices\mtom`. Select **BlueHills.jpg** and click **Open**.
 - c. Click **Invoke** to invoke the `sendImage` method.
 - d. In the Result pane, click **View image**. The image is displayed in the Results pane.

Tip: The security settings of your external browser might prevent this sample from being able to access the local files, producing exceptions, such as `FileNotFoundException`. If you encounter this issue, try to use the internal browser (**Window** → **Preferences: General** → **Web Browser** → **Use internal Web Browser**).

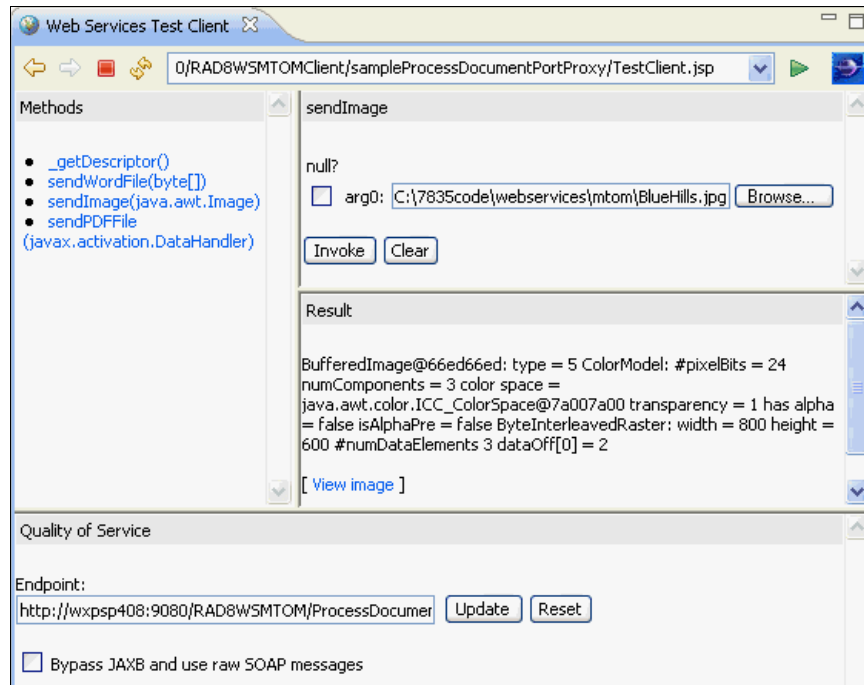



Figure 14-49 Invoking the MTOM web service sendImage method

2. Examine the C:\7835code\webservices\mtomresult folder. You can see that the BlueHills.jpg file is stored in this folder. The size differs from the size of the same file in the mtom folder. Probably, a separate JPEG compression is used.

3. Select the **TCP/IP Monitor** tab to view the SOAP traffic. Click the  icon and then select **Show Header**. Figure 14-50 shows the HTTP header and the SOAP traffic.

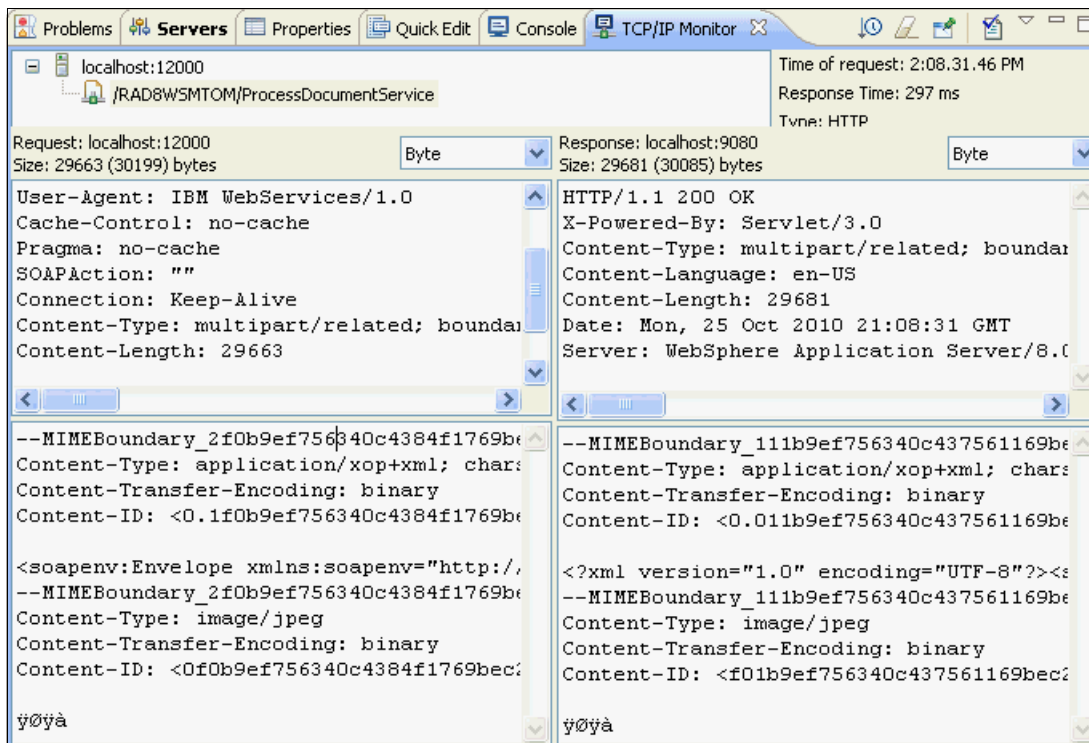


Figure 14-50 SOAP traffic when MTOM is enabled for the web service and client

Look at the SOAP request and response:

- ▶ The web service (provider) has MTOM enabled after the code generation. Therefore, the SOAP response has a smaller payload. The web service sends the binary data as a MIME attachment outside of the XML document to realize the optimization.
- ▶ The SOAP request also has MTOM enabled and therefore a smaller payload.

Example 14-26 on page 771 shows the SOAP response and its HTTP header (manually formatted).

Example 14-26 HTTP header SOAP response message with MTOM enabled

```
HTTP/1.1 200 OK
X-Powered-By: Servlet/3.0
Content-Type: multipart/related;
boundary="MIMEBoundary_111b9ef756340c437561169bec2d4285933d44b19cd7a882
"; type="application/xop+xml";
start="<0.011b9ef756340c437561169bec2d4285933d44b19cd7a882@apache.org>"
; start-info="text/xml"
Content-Language: en-US
Content-Length: 29681
Date: Mon, 25 Oct 2010 21:08:31 GMT
Server: WebSphere Application Server/8.0
=====
===
--MIMEBoundary_111b9ef756340c437561169bec2d4285933d44b19cd7a882
Content-Type: application/xop+xml; charset=UTF-8; type="text/xml"
Content-Transfer-Encoding: binary
Content-ID:
<0.011b9ef756340c437561169bec2d4285933d44b19cd7a882@apache.org>

<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"><soapenv:Body
><a:sendImageResponse
xmlns:a="http://mtom.rad8.ibm.com/"><return><xop:Include
xmlns:xop="http://www.w3.org/2004/08/xop/include"
href="cid:f01b9ef756340c437561169bec2d4285933d44b19cd7a882@apache.org"/
></return></a:sendImageResponse></soapenv:Body></soapenv:Envelope>
--MIMEBoundary_111b9ef756340c437561169bec2d4285933d44b19cd7a882
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID:
<f01b9ef756340c437561169bec2d4285933d44b19cd7a882@apache.org>
```

The type and content-type attributes have the value `application/xop+xml`, which indicates that the message was successfully optimized using XOP when MTOM was enabled.

To see the difference when MTOM is not enabled, let us test the new `@MTOM` annotation on the server.

Open the file **ProcessDocumentPortBindingImpl.java** and replace the following line:

```
@javax.xml.ws.BindingType
(value=javax.xml.ws.soap.SOAPBinding.SOAP11HTTP_MTOM_BINDING)
```

Replace it with this line:

```
@MTOM(enabled=true,threshold=200000)
```

Then use CTRL+SHIFT+O to arrange the import statements. This new line means that if the attachment is smaller in size than the threshold (in bytes), the service needs to send binary data as base64 encoded data within the XML document, instead of optimizing it using XOP.

After you save the file and republish the server, perform the same test as before and look at the TCP/IP monitor. Figure 14-51 shows that, in this case, the image is embedded as base64 encoded data in the response.

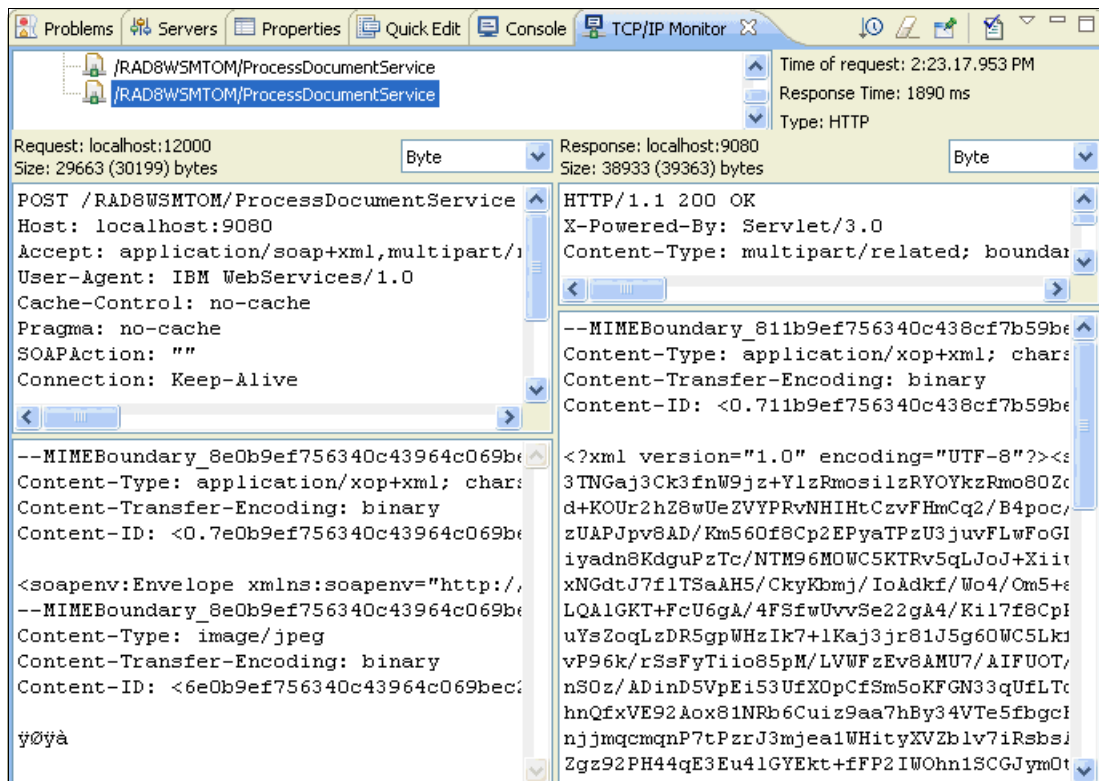


Figure 14-51 Using @MTOM with a threshold higher than the attachment size

Example 14-27 shows the HTTP Header and response body in this case, with the binary data omitted.

Example 14-27 HTTP Header and response when using @MTOM with a high threshold

```
HTTP/1.1 200 OK
X-Powered-By: Servlet/3.0
Content-Type: multipart/related;
boundary="MIMEBoundary_811b9ef756340c438cf7b59bec2d4285833d44b19cd7a882";
type="application/xop+xml";
start="<0.711b9ef756340c438cf7b59bec2d4285833d44b19cd7a882@apache.org>"
; start-info="text/xml"
Content-Language: en-US
Transfer-Encoding: chunked
Date: Mon, 25 Oct 2010 21:23:19 GMT
Server: WebSphere Application Server/8.0
=====
--MIMEBoundary_811b9ef756340c438cf7b59bec2d4285833d44b19cd7a882
Content-Type: application/xop+xml; charset=UTF-8; type="text/xml"
Content-Transfer-Encoding: binary
Content-ID:
<0.711b9ef756340c438cf7b59bec2d4285833d44b19cd7a882@apache.org>

<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"><soapenv:Body
><a:sendImageResponse xmlns:a="http://mtom.rad8.ibm.com/"><return>
...omitted encoded binary data....
</return></a:sendImageResponse></soapenv:Body></soapenv:Envelope>
--MIMEBoundary_811b9ef756340c438cf7b59bec2d4285833d44b19cd7a882--
```

14.15.5 How MTOM was enabled on the client

Let us review how the Web Services wizard has enabled MTOM on the client:

1. In the Enterprise Explorer, expand **RAD8MTOMClient** → **Java Resources** → **src** → **com.ibm.rad8.mtom** and open **ProcessDocumentPortProxy.java**.
2. Review the method `setMTOMEnabled` that is shown in Example 14-28.

Example 14-28 How MTOM was enabled on the client

```
public class ProcessDocumentPortProxy{

    protected Descriptor _descriptor;

    public class Descriptor {
```

```

.....
    public void setMTOMEnabled(boolean enable) {
        SOAPBinding binding = (SOAPBinding) ((BindingProvider)
        _proxy).getBinding();
        binding.setMTOMEnabled(enable);
    }
} //end of class Descriptor

public ProcessDocumentPortProxy() {
    _descriptor = new Descriptor();
    _descriptor.setMTOMEnabled(true);
}

public ProcessDocumentPortProxy(URL wsdlLocation, QName
serviceName) {
    _descriptor = new Descriptor(wsdlLocation, serviceName);
    _descriptor.setMTOMEnabled(true);
}
...
}

```

There are now other ways of enabling MTOM on the client. For example, as of JAX-WS 2.2, you can use @MTOM. As of JAX-WS 2.1, you can use MTOMFeature. For examples, refer to the WebSphere Application Server Information Center:

http://publib.boulder.ibm.com/infocenter/wasinfo/beta/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tejb_timerserviceejb_enh.html

The completed MTOM example is available in the file:

C:\7835codesolution\webservices\RAD8WSMTOM.zip

14.16 JAX-RS programming model

The JAX-RS programming model is based on the principles of Representational State Transfer (REST) architectures, which were introduced by Roy Fielding in his dissertation *Architectural Styles and the Design of Network-based Software Architectures*:

<http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

Resources

Resources are the key concept in REST. Resources must be addressable, for example, using Uniform Resource Identifiers (URIs).

Resource representations

Resources can have multiple representations that are suitable for being served to various types of clients. Examples of representations are HTML (to be consumed by web browsers), XML (to be consumed by Java clients), and JavaScript Object Notation (JSON) (to be consumed by JavaScript clients). These representations offered to clients are independent of the way that the actual data referenced by the resources is stored on the server, which can be in a relational database.

Uniform interface

Contrary to SOAP-based web services, where each service defines its own interface, introducing the need for WSDLs to expose the specific interface to clients, in RESTful architectures the set of methods that can be invoked on the resources is limited and well-known. In particular, JAX-RS is based on the HTTP protocol and restricts the possible methods to the HTTP methods, as described in the Hypertext Transfer Protocol -- HTTP/1.1 Request for Comments (RFC):

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>

GET

GET is used to read the resource. It does not change the resource (it is *safe*). Therefore, it can be called multiple times, and it continues to produce the same side effects (it is *idempotent*).

PUT

PUT is used to either insert or update a resource. The client provides the identity (URI) of the resource to update or create. Because the identity of the resource is already known, this method is idempotent, although obviously, it is not safe.

DELETE

Delete is used to delete the resource. It is idempotent.

POST

POST is used to create a new resource on the server. The important difference compared to PUT is that the client of a POST request sends the URI of the resource that *includes* the new resource to be created, while the client of PUT provides the URI of the resources to be created. POST is not idempotent, and it is not safe.

HEAD

HEAD returns the same information as GET, apart from the actual response body. The client receives a status code and headers, if any.

OPTIONS

OPTIONS is used to retrieve the communication options associated with a resource, or the capabilities of a server, without actually retrieving the resource.

Statelessness

The requirement for statelessness communication is present to allow applications to scale, and it is based on the success of the HTTP protocol.

Hypermedia as the Engine of Application State (HATEOAS)

The server response to a request for a given resource must consist of *hypermedia* containing *links* to other resources that the client can access.

14.16.1 Implementation of JAX-RS in WebSphere Application Server

The IBM JAX-RS implementation is based on Apache Wink. Wink is a project developed within the Apache Software Foundation that provides a lightweight framework for developing RESTful applications. Wink supports REST services implemented using JAX-RS to describe the resources on the server. However, a client API is also provided by Wink. This client API is specific to the Wink runtime environment, because there is no JAX-RS-defined client API.

The IBM implementation of JAX-RS 1.1 is an extension of the base Wink 1.1.1 runtime environment. IBM JAX-RS includes the following features:

- ▶ JAX-RS server run time
- ▶ Stand-alone client API with the option to use Apache HttpClient 4.0 as the underlying client
- ▶ Built-in entity provider support for JSON4J
- ▶ An Atom JAXB model in addition to Apache Abdera support
- ▶ Multipart content support
- ▶ A handler system to integrate user handlers into the processing of requests and responses

JAX-RS is supported by the following run times (Table 14-3).

Table 14-3 Versions of WebSphere Application Server that support JAX-RS

JAX-RS	Notes	WebSphere Application Server
1.0	Requires Dynamic Web Module 2.3 or higher and Java 1.5 or higher	7.0 with Feature Pack for Web 2.0 8.0

JAX-RS	Notes	WebSphere Application Server
1.1	Requires Dynamic Web Module 2.4 or higher and Java 1.6 or higher. Based on Apache Wink 1.1.1	7.0 with Feature Pack for Web 2.0 8.0

14.16.2 JAX-RS project setup

You can set up a JAX-RS enabled project in one of the following two ways, depending on whether you use a new project or an existing one.

Follow these steps to configure a new Dynamic Web project:

1. Select **File** → **New** → **Dynamic Web project**.
2. Perform these steps:
 - a. In the Project Name field, enter RAD8JAX-RSWeb.
 - b. In the Dynamic Web Module version field, select **3.0**.
 - c. In the Configuration field, select **IBM JAX-RS Configuration**.
 - d. In the EAR field, enter the name RAD8JAX-RSEAR.
 - e. Select **Next** three times until you reach the window that is shown in Figure 14-52 on page 778.

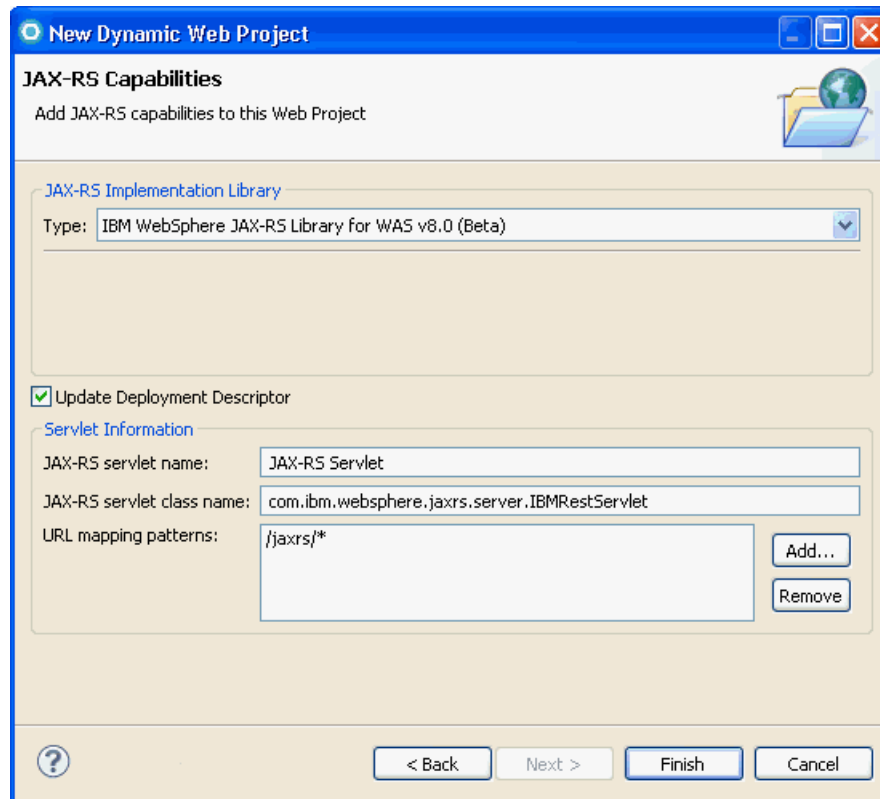


Figure 14-52 Adding JAX-RS Capabilities to the web project

3. Following these steps, add the JAX-RS facet. If the target server is V7.0, these steps add the Ajax Proxy and server-side technology facets. The JAX-RS facet adds the library, servlet information, and support for JAX-RS annotations processing and JAX-RS quick fixes.
4. Select **Finish**.

Follow these steps to configure an existing project:

1. Right-click the project and select **Properties**.
2. Select **Project Facets**.
3. In the Configuration field, select **IBM JAX-RS Configuration**.
4. Select **Further configuration required**.
5. In the JAX-RS Implementation Library field, select **IBM WebSphere JAX-RS Library for WAS v8.0 (Beta)**.
6. Select **Update Deployment Descriptor** and select **OK**.

Example 14-29 shows the servlet and servlet mapping that are added to the Deployment Descriptor (`web.xml`).

Example 14-29 JAX-RS servlet and servlet mapping in web.xml

```
<servlet>
  <description>
    JAX-RS Tools Generated - Do not modify</description>
  <servlet-name>JAX-RS Servlet</servlet-name>
  <servlet-class>com.ibm.websphere.jaxrs.server.IBMRestServlet
</servlet-class>
  <load-on-startup>1</load-on-startup>
  <enabled>>true</enabled>
  <async-supported>>false</async-supported>
</servlet>
<servlet-mapping>
  <servlet-name>JAX-RS Servlet</servlet-name>
  <url-pattern>
    /jaxrs/*</url-pattern>
</servlet-mapping>
```

For more information about the configuration of the IBM JAX-RS servlet `com.ibm.websphere.jaxrs.server.IBMRestServlet`, see these websites:

http://www14.software.ibm.com/webapp/wsbroker/redirect?version=matt&product=was-base-dist&topic=twbs_jaxrs_configwebxml

http://www14.software.ibm.com/webapp/wsbroker/redirect?version=v700web&product=was-base-dist&topic=twbs_jaxrs_configwebxml

For both a new and an existing project, we must add a class that extends `javax.ws.rs.core.Application`. This class indicates which classes with the `@Path` and `@Provider` annotations need to be deployed by the JAX-RS run time.

Rational Application Developer provides a quick fix to help you create this class:

1. Open the Deployment Descriptor (**web.xml**).
2. In the Design tab, select **Servlet (JAX-RS Servlet)**.
3. Click **Add**.
4. Select **Initialization Parameter**.
5. Save the `web.xml` file.
6. In the Problems view, the following warning appears:
“JSR-311, 2.3.2: The param-name should be `javax.ws.rs.Application`.”

7. Right-click the warning and select **Quick Fix**.
8. In the Quick Fix window, select **Create a new JAX-RS Application sub-class and set the param-value in the web.xml**, as shown in Figure 14-53.
9. In the Class creation wizard, perform these tasks:
 - a. For Package, type `itso.bank.jaxrs`.
 - b. For Name, type `BankJAXRSApplication`.
 - c. Accept the **javax.ws.rs.core.Application** superclass.
 - d. Select **Finish**.
10. In the Java editor, right-click and select **Source** → **Override/Implement Methods**.
11. Select the **getClasses()** method from the Application class.
12. Modify the method body of `getClasses()`, as shown in Example 14-30 on page 781.

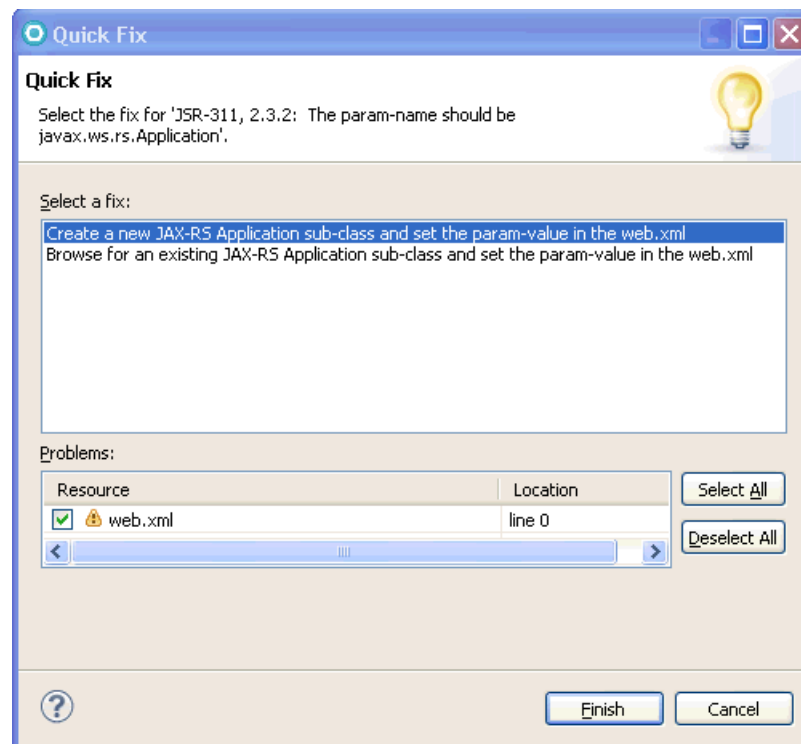


Figure 14-53 Quick fix to create a new JAX-RS Application sub-class


```
package itso.bank.jaxrs;

import java.util.HashSet;
import java.util.Set;

import javax.ws.rs.core.Application;

public class BankJAXRSApplication extends Application {

    @Override
    public Set<Class<?>> getClasses() {
        Set<Class<?>> classes = new HashSet<Class<?>>();
        return classes;
    }
}
```

We will add classes that represent resources (annotated with `@Path`) to the collection returned by `getClasses()`. Instances of these classes will be created by the run time with a per-request life cycle.

You can get the project, which we have developed so far, in this file:

C:\7835code\webservices\RAD8JAX-RSstart.zip

14.16.3 Exposing a JPA application as a RESTful service

This section shows how the JPA application, which is described in Chapter 10, “Persistence using the Java Persistence API” on page 443, can be exposed as a JAX-RS service. First, we import the JPA project RAD8JPA:

1. Select **File** → **Import**.
2. Select **General** → **Existing project into Workspace**.
3. Choose **Select archive file** and click **Browse** for
C:\7835codesolution\jpa\RAD8JPA.zip.
4. Import the project **RAD8JPA**.

Before we can use this project, we must add it to our EAR:

1. Right-click the EAR project **RAD8JAX-RSWeb**.
2. Select **Properties**.
3. Select **Deployment Assembly**.
4. Select **Add**.

5. Select **Project**.
6. Select **Next**.
7. Select **RAD8JPA**.
8. Select **Finish** and then click **OK**.

Now we configure the project for deploying the JPA entities on the server:

1. Make sure that you have the connection called `ITSOBANKDerby`, which was created in “Creating a connection to the ITSOBANK database” on page 395.
2. Configure a data source called `jdbc/itsobank`, either on WebSphere Application Server or by using the enhanced EAR file (23.8, “Configuring application and server resources” on page 1258). You can also use JPA Tools to perform this task:
 - a. Right-click **RAD8JPA**.
 - b. Select **JPA Tools** → **Configure project for JDBC Deployment**.
 - c. In Connection, select **ITSOBANKDerby**.
 - d. Clear **Set Up persistence.xml**, because it is already configured.
 - e. Select **Deploy connection information to server**.
 - f. The data source will be called in the same way that the connection is called, so you need to rename the data source:
 - i. Right-click **RAD8JAX-RSEAR**.
 - ii. Select **Java EE** → **Open WebSphere Application Server Deployment**.
 - iii. Rename `jdbc/ITSOBANKDerby` to `jdbc/itsobank`.

We are now ready to start exposing the JPA entities as JAX-RS resources:

1. In project `RAD8JAX-RSWeb`, create a Java package called `itso.bank.resources`.
2. Add to this package three Java classes:
 - `AccountResource`
 - `CustomerResource`
 - `TransactionResource`
3. In the class `BankJAXRSApplication`, add these three classes to the `HashSet` that is returned by the method `getClasses`, as shown in Example 14-31.

Example 14-31 Completed BankJAXRSApplication class

```
package itso.bank.jaxrs;  
  
import itso.bank.resources.AccountResource;  
import itso.bank.resources.CustomerResource;  
import itso.bank.resources.TransactionResource;
```

```

import java.util.HashSet;
import java.util.Set;

import javax.ws.rs.core.Application;

public class BankJAXRSApplication extends Application {

    @Override
    public Set<Class<?>> getClasses() {
        Set<Class<?>> classes = new HashSet<Class<?>>();
        classes.add(CustomerResource.class);
        classes.add(AccountResource.class);
        classes.add(TransactionResource.class);
        return classes;
    }
}

```

In order to define the URIs for the Resources, we use the `@Path` annotation.

`@Path(@javax.ws.rs.Path)`

`@Path` can be applied to Java classes and methods. It is used to define the URI (relative to the web project context-root) for an incoming HTTP request that must be answered by that class/method.

Table 14-4 shows how the values of the `@Path` annotation map to URIs, based on the previously chosen project name (RAD8JAX-RSWeb) and on the JAX-RS servlet mapping already declared in `web.xml`.

Table 14-4 Mapping of @Path annotations to URIs

Annotation	URL
<code>@Path("/customers")</code>	<code>http://hostname:portname/RAD8JAX-RSWeb/jaxrs/customers</code>
<code>@Path("/accounts")</code>	<code>http://hostname:portname/RAD8JAX-RSWeb/jaxrs/accounts</code>
<code>@Path("/transaction")</code>	<code>http://hostname:portname/RAD8JAX-RSWeb/jaxrs/transaction</code>

In our first implementation of `CustomerResource` (Example 14-32 on page 784), we only want to return a JSON representation of an array of `Customer` entities. To access the entities, we reuse the JPA Manager Beans (`CustomerManager` in this case) and we make use of `JSON4J` to convert the Java Representation to JSON.

```
package itso.bank.resources;

import itso.bank.entities.Customer;
import itso.bank.entities.controller.CustomerManager;

import java.io.IOException;
import java.util.List;

import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;

import com.ibm.json.java.JSONArray;
import com.ibm.json.java.JSONObject;

@Path("/customers")
public class CustomerResource {

    private CustomerManager manager;
    private EntityManagerFactory emf;

    public CustomerResource() {
        super();
        emf = Persistence.createEntityManagerFactory("RAD8JPA");
        manager = new CustomerManager(emf);
    }

    @GET
    @Produces("application/json")
    public JSONArray getAllCustomers() throws IOException {
        final List<Customer> allCustomers = manager.getCustomers();
        JSONArray jsonArray = jsonCustomerArray(allCustomers);
        return jsonArray;
    }

    private static JSONArray jsonCustomerArray(final List<Customer>
allCustomers) {
        JSONArray jsonArray = new JSONArray(allCustomers.size());
        for (Customer customer : allCustomers) {
            jsonArray.add(jsonCustomer(customer));
        }
    }
}
```

```

    }
    return jsonArray;
}
private static JSONObject jsonCustomer(Customer customer) {
    JSONObject obj = new JSONObject();
    obj.put("title", customer.getTitle());
    obj.put("firstName", customer.getFirstName());
    obj.put("lastName", customer.getLastName());
    obj.put("ssn", customer.getSsn());
    return obj;
}
}

```

We have introduced two additional annotations.

@Get(@javax.ws.rs.Get)

This annotation is used to denote a method that performs the safe, idempotent HTTP GET operation.

@Produces(@javax.ws.rs.Produces)

This annotation indicates which media type is returned by a method annotated with @Get.

Testing the first implementation of CustomerResource

To test this code, publish the EAR to the server and then simply open a web browser and enter the following URL:

<http://localhost:9080/RAD8JAX-RSWeb/jaxrs/customers>

Typically, web browsers do not know how to handle the type “application/json”, which is meant to be consumed by JavaScript, and you are likely prompted to save the file or open it with an application of your choosing. After you open the file in any text editor, you see the contents, as shown in Example 14-33.

Example 14-33 JSONArray produced by the method getAllCustomers

```

[{"lastName":"Cui","title":"Mr","firstName":"Henry","ssn":"111-11-1111"},
{"lastName":"Fleming","title":"Ms","firstName":"Craig","ssn":"222-22-2222"},
{"lastName":"Coutinho","title":"Mr","firstName":"Rafael","ssn":"333-33-3333"},
{"lastName":"Sollami","title":"Mr","firstName":"Salvatore","ssn":"444-44-4444"},
{"lastName":"Hainey","title":"Mr","firstName":"Brian","ssn":"555-55-5555"},
{"lastName":"Baber","title":"Mr","firstName":"Steve","ssn":"666-66-6666"},
{"lastName":"Venkatraman","title":"Mr","firstName":"Sundaragopal","ssn":"777-77-7777"},
{"lastName":"Ziosi","titl

```

```
e": "Mr", "firstName": "Lara", "ssn": "888-88-8888"}, {"lastName": "Lippmann",
"title": "Mr", "firstName": "Sylvi", "ssn": "999-99-9999"}, {"lastName": "Kuma
ri", "title": "Mr", "firstName": "Venkata", "ssn": "000-00-0000"}, {"lastName":
:"Keen", "title": "Mr", "firstName": "Martin", "ssn": "000-00-1111"}]
```

We can also test with a stand-alone Java client:

1. Create a new Java project called RAD8JAX-RSClient.
2. Right-click the project and select **Properties** → **Java Build Path**.
3. In Libraries, select **Add External Jar** and browse for `<WAS_HOME>\runtimes\com.ibm.jaxrs.thinclient_8.0.0.jar`, which is the redistributable WebSphere Application Server JAX-RS Thin Client that you can use in Java stand-alone applications.
4. Add a new class in the `itso.bank.jaxrs.client` package with the code that is shown in Example 14-34.

Example 14-34 GetAllCustomersClient stand-alone JAX-RS Client

```
package itso.bank.jaxrs.client;

import java.io.IOException;
import org.apache.wink.client.ClientResponse;
import com.ibm.json.java.JSONArray;

public class GetAllCustomersClient {

    private org.apache.wink.client.ClientConfig clientConfig = new
    org.apache.wink.client.ClientConfig();
    private org.apache.wink.client.RestClient client = new
    org.apache.wink.client.RestClient(clientConfig);
    private final String base =
    "http://localhost:9080/RAD8JAX-RSWeb/jaxrs";

    public static void main(String args[]) throws IOException {
        GetAllCustomersClient getAllCustomersClient = new
        GetAllCustomersClient();
        getAllCustomersClient.getResource(getAllCustomersClient.base +
        "/customers");
    }

    public JSONArray getResource(String URI) {
        org.apache.wink.client.Resource resource = client.resource(URI);
        ClientResponse response = resource.get();
    }
}
```

```

        System.out.println("Getting: " + URI);
        System.out.println("Received Message:\n" + response.getMessage());
        System.out
            .println("Received Entity:\n" +
response.getEntity(JSONArray.class));
        return response.getEntity(JSONArray.class);
    }
}

```

5. Right-click this class and select **Run As** → **Java Application**.
6. You obtain similar output to the output that is shown in the browser.

Example 14-35 Java application output

```

Getting: http://localhost:9080/RAD8JAX-RSWeb/jaxrs/customers
Received Message:
OK
Received Entity:
[{"lastName":"Cui","title":"Mr","firstName":"Henry","ssn":"111-11-11
11"},.....

```

Finishing the implementation of CustomerResource

We can now implement additional methods for CustomerResource, such as `getCustomerByPartialName`, which use a `NamedQuery` defined in the class `CustomerManager`, as shown in Example 14-36.

Example 14-36 Method that invokes a JPA NamedQuery

```

@Path("pname/{pname}")
@GET
@Produces("application/json")
public JSONArray getCustomersByPartialName(@PathParam(value = "pname")
String pname) {
    final List<Customer> allCustomers =
manager.getCustomersByPartialName(pname);
JSONArray jsonArray = jsonCustomerArray(allCustomers);
    return jsonArray;
}

```

We have introduced a new annotation.

@PathParam(@javax.ws.rs.PathParam)

`@PathParam` is placed in front of an operation parameter. It takes the `value` attribute, which is also referenced in the `@Path` annotation and is enclosed in

braces ({}), which allows the JAX-RS run time to inject the (converted) segment of the URL into the Java method parameter.

To test the new method, point the browser at this website:

`http://localhost:9080/RAD8JAX-RSWeb/jaxrs/customers/pname/Ziosi`

The browser will return a file that contains this information:

```
[{"lastName":"Ziosi","title":"Mr","firstName":"Lara","ssn":"888-88-8888"}]
```

Mapping Account and Transaction to JSONObject

The next step is to create methods for other NamedQueries that are exposed by CustomerManager. These queries can return other types of entities, so we need to first define how all other entities map to JSONObject. We define static methods in AccountResource (Example 14-37) and TransactionResource (Example 14-38) to convert Account and Transaction to JSONObject.

Example 14-37 Code to add to AccountResource

```
public static JSONArray jsonAccountArray(final List<Account>
allAccounts) {
    JSONArray jsonArray = new JSONArray(allAccounts.size());
    for (Account account : allAccounts) {
        jsonArray.add(jsonAccount(account));
    }
    return jsonArray;
}

public static JSONObject jsonAccount(Account account) {
    JSONObject obj = new JSONObject();
    obj.put("id", account.getId());
    obj.put("balance", account.getBalance());
    return obj;
}
```

Example 14-38 Code to add to TransactionResource

```
public static JSONArray jsonTransactionArray(
    final List<Transaction> allTransactions) {
    JSONArray jsonArray = new JSONArray(allTransactions.size());
    for (Transaction transaction : allTransactions) {
        jsonArray.add(jsonTransaction(transaction));
    }
    return jsonArray;
}
```



```

public static JSONObject jsonTransaction(Transaction transaction) {
    JSONObject obj = new JSONObject();
    obj.put("id", transaction.getId());
    obj.put("amount", transaction.getAmount().toString());
    obj.put("transTime", transaction.getTransTime().toString());
    obj.put("transType", transaction.getTransType());
    return obj;
}

```

We have chosen to represent Debit and Credit entities with only one type of resource called `TransactionResource`, which is mapped to a `JSONObject` that has a transaction type (`transType`).

We can now define the additional method of class `CustomerResource`.

Example 14-39 `getAccountsForSSN` in `CustomerResource`

```

@Path("accounts/{ssn}")
@GET
@Produces("application/json")
public JSONArray getAccountsForSsn(@PathParam(value = "ssn") String
ssn) {
    final List<Account> allAccounts = manager.getAccountsForSSN(ssn);
    JSONArray jsonArray = AccountResource.jsonAccountArray(allAccounts);
    return jsonArray;
}

```

To test the new method, point the browser at this website:

<http://localhost:9080/RAD8JAX-RSWeb/jaxrs/customers/accounts/111-11-1111>

This step returns a `JSONArray` with the accounts corresponding to the `ssn`:

```

[{"id":"001-111001","balance":12645.67}, {"id":"001-111002","balance":68
43.21}, {"id":"001-111003","balance":398.76}]

```

The complete implementation of `AccountResource` conceptually does not differ from `CustomerResource`. You can inspect it in the completed project.

Posting data

`TransactionResource` is more interesting, because you see how to create new transactions (Example 14-40 on page 790).

```
@POST
@Consumes("application/json")
@Produces("application/json")
public JSONObject createTransaction(JSONObject inputObj)
    throws WebApplicationException {
    String transType = (String) inputObj.get("transType");
    Transaction transaction = null;
    EntityManagerFactory emf = Persistence
        .createEntityManagerFactory("RAD8JPA");
    AccountManager accountManager = new AccountManager(emf);
    Account account = accountManager.findAccountById((String) inputObj
        .get("accountId"));
    if(account ==null)
        throw new
WebApplicationException(jsonObjectResponse(Status.BAD_REQUEST, "No
Account Found"));
    Transaction persistedTransaction = null;
    if (transType.equals("Credit")) {
        transaction = new Credit(BigDecimal.valueOf(Double
            .parseDouble(inputObj.get("amount").toString())));
        try {
            transaction.setAccount(account);
            creditManager.createCredit((Credit) transaction);
            persistedTransaction = creditManager
                .findCreditById(transaction.getId());
        } catch (Exception e) {
            throw new
WebApplicationException(jsonObjectResponse(Status.INTERNAL_SERVER_ERROR
,e.getMessage()));
        }
    } else if (transType.equals("Debit")) {
        transaction = new Debit(BigDecimal.valueOf(Double
            .parseDouble(inputObj.get("amount").toString())));
        try {
            transaction.setAccount(account);
            debitManager.createDebit((Debit) transaction);
            persistedTransaction = debitManager
                .findDebitById(transaction.getId());
        } catch (Exception e) {
            throw new
WebApplicationException(jsonObjectResponse(Status.INTERNAL_SERVER_ERROR
,e.getMessage()));
        }
    }
}
```

```

    } else {
        throw new
WebApplicationException(jsonObjectResponse(Status.BAD_REQUEST,transType
+ " should be Debit or Credit"));
    }
    return jsonTransaction(persistedTransaction);
}

```

This method takes as input a JSONObject representing a Transaction. It tries first to find the Account corresponding to the accountId field of the input JSONObject. Then it tries to determine whether the input corresponds to a Credit or Debit transaction, based on the value of the transType field of the input JSONObject. If both pieces of information can be retrieved successfully, it tries to create a new Credit or Debit object, and finally it tries to persist it.

We have introduced two new annotations.

@Post(@javax.ws.rs.Post)

The JAX-RS run time directs the HTTP POST requests that match the URL of the enclosing @PATH annotation to the method annotated with @Post. The method annotated with @Post is typically used to create new elements that will have the URL of the enclosing @PATH annotation.

@Consumes(@javax.ws.rs.Consumes)

The @Consumes annotation defines what media type the method expects to receive as input from the HTTP request.

Managing exceptions in JAX-RS

Many possible error conditions can cause the transaction creation to fail, such as the supplied accountId might not match any existing account or the transType might be spelled incorrectly, and so on.

The class javax.ws.rs.WebApplicationException can be used to represent exceptions in JAX-RS. The constructor that we have used takes as input a javax.ws.rs.core.Response object. Because we want to always return a JSONObject or a JSONArray, even when an error condition occurs, we have constructed a special Response object with the help of the following utility class (Example 14-41).

Example 14-41 Utility class to return Response based on JSONObject or JSONArray

```

package itso.bank.resources;

import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;

```

```

import com.ibm.json.java.JSONArray;
import com.ibm.json.java.JSONObject;

public class ErrorUtil {
    public static JSONObject errorObject(String msg){
        JSONObject object = new JSONObject();
        object.put("error", msg);
        return object;
    }
    public static JSONArray errorArray(String msg){
        JSONObject object = errorObject(msg);
        JSONArray array = new JSONArray();
        array.add(object);
        return array;
    }
    public static Response jsonObjectResponse(Status status, String
msg){
        return
Response.status(Status.BAD_REQUEST).type("application/json").entity(err
orObject(msg)).build();
    }
    public static Response jsonArrayResponse(Status status, String msg){
        return
Response.status(Status.BAD_REQUEST).type("application/json").entity(err
orArray(msg)).build();
    }
}

```

Testing the completed JAX-RS application

After introducing suitable exceptions in all other methods and implementing a method to get all transactions in the `TransactionResource` class, as documented in the provided completed application sources, we are ready to test the complete application with the following code, which must be added to the Java Project `RAD8JAX-RSClient`.

Example 14-42 Complete application

```

package itso.bank.jaxrs.client;

import java.io.IOException;
import java.math.BigDecimal;
import org.apache.wink.client.ClientResponse;
import org.apache.wink.client.Resource;

```

```

import com.ibm.json.java.JSONArray;
import com.ibm.json.java.JSONObject;

public class MakeTransactionsClient {

    private org.apache.wink.client.ClientConfig clientConfig = new
org.apache.wink.client.ClientConfig();
    private org.apache.wink.client.RestClient client = new
org.apache.wink.client.RestClient(clientConfig);
    private final String base =
"http://localhost:9080/RAD8JAX-RSWeb/jaxrs";

    public static void main(String args[]) throws IOException {
        MakeTransactionsClient makeTransactionsClient = new
MakeTransactionsClient();
        String[] ssn = makeTransactionsClient.getCustomerSSN("Ziosi");
        for (int i = 0; i < ssn.length; i++) {
            String[] accountId =
makeTransactionsClient.getAccountId(ssn[i]);
            for (int j = 0; j < accountId.length; j++) {
                makeTransactionsClient.makeTransaction(new
BigDecimal(100.00),"Debit", accountId[j]);
                makeTransactionsClient.makeTransaction(new
BigDecimal(450.00),"Credit", accountId[j]);
                //Bad Request, mis-spelled Credit
                makeTransactionsClient.makeTransaction(new
BigDecimal(450.00),"credit", accountId[j]);
                //Non-existing account
                makeTransactionsClient.makeTransaction(new
BigDecimal(450.00),"credit", "NonExistingAccount");

makeTransactionsClient.listTransactionsForAccount(accountId[j]);
            }
        }

        private JSONArray getResource(String URI){
            org.apache.wink.client.Resource resource =
client.resource(URI);
            resource.contentType("application/json");
            ClientResponse response =resource.get();
            System.out.printf("\n%20s %s\n", "Getting:",URI);
            System.out.printf("%20s %s\n", "Received
Message:", response.getMessage());
            System.out.printf("%20s %s\n", "Received
Entity:",prettyPrint(response.getEntity(JSONArray.class)));
        }
    }
}

```

```

        return response.getEntity(JSONArray.class);
    }
    private String[] getCustomerSSN(String pname) throws IOException {
        String customerURI = this.base + "/customers/pname/" + pname;
        JSONArray jsonArray = getResource(customerURI);
        String[] ssn = new String[jsonArray.size()];
        for (int i = 0; i < jsonArray.size(); i++) {
            JSONObject jsonObject = (JSONObject) jsonArray.get(i);
            ssn[i] = (String) jsonObject.get("ssn");
            System.out.printf("\n%20s %s\n", "Found ssn:", ssn[i]);
        }
        return ssn;
    }
    private String[] getAccountId(String ssn) throws IOException {
        String accountsURI = base + "/accounts/ssn/" + ssn;
        JSONArray jsonArray = getResource(accountsURI);
        String accountId[] = new String[jsonArray.size()];
        for (int i = 0; i < jsonArray.size(); i++) {
            JSONObject jsonObject = (JSONObject) jsonArray.get(i);
            accountId[i] = (String) jsonObject.get("id");
            System.out.printf("\n%20s %s\n", "Found
accountID:", accountId[i]);
        }
        return accountId;
    }
    private void makeTransaction(BigDecimal amount, String transType,
        String accountId) throws IOException {
        String transactionURI = base + "/transaction";
        Resource resource = client.resource(transactionURI);
        resource.contentType("application/json");
        JSONObject jsonObj = new JSONObject();
        jsonObj.put("amount", amount);
        jsonObj.put("transType", transType);
        jsonObj.put("accountId", accountId);
        String jsonStr = jsonObj.serialize();
        System.out.printf("\n%20s %s\n", "Posting To:", transactionURI);
        System.out.printf("%20s %s\n", "PostedData:", jsonStr);
        ClientResponse response = resource.post(jsonObj);
        System.out.printf("%20s %s\n", "Received
Message:", response.getMessage());
        System.out.printf("%20s %s\n", "Received Entity:",
response.getEntity(JSONObject.class));
    }
    private void listTransactionsForAccount(String accountId) {
        String transactionsURI = base + "/accounts/" + accountId

```

```

        + "/transactions";
    getResource(transactionsURI);
    }
private String prettyPrint(JSONArray entity) {
    Object[]objects =entity.toArray();
    String pretty="";
    String spaces="          ";
    for(int i=0;i<objects.length;i++){
        pretty=pretty+objects[i)+"\n"+spaces;
    }
    return pretty;
}
}
}

```

Figure 14-54 shows the results of running this code.

```

<terminated> MakeTransactionsClient [Java Application] E:\Program Files\IBM\SDP\jdk\bin\javaw.exe (Oct 27, 2010 12:06:31 AM)
    Getting: http://localhost:9080/RAD8JAX-RSWeb/jaxrs/customers/pname/Ziosi
Received Message: OK
    Received Entity: {"lastName":"Ziosi","title":"Mr","firstName":"Lara","ssn":"888-88-8888"}

    Found ssn: 888-88-8888

    Getting: http://localhost:9080/RAD8JAX-RSWeb/jaxrs/accounts/ssn/888-88-8888
Received Message: OK
    Received Entity: {"id":"008-888001","balance":500.0}

    Found accountID: 008-888001

    Posting To: http://localhost:9080/RAD8JAX-RSWeb/jaxrs/transaction
    PostedData: {"transType":"Debit","accountId":"008-888001","amount":100}
Received Message: OK
    Received Entity: {"transType":"Debit","transTime":"2010-10-27 00:06:33.921","amount":"100.00","id":"47712504"}

    Posting To: http://localhost:9080/RAD8JAX-RSWeb/jaxrs/transaction
    PostedData: {"transType":"Credit","accountId":"008-888001","amount":450}
Received Message: OK
    Received Entity: {"transType":"Credit","transTime":"2010-10-27 00:06:34.39","amount":"450.00","id":"7dd1cec8"}

    Posting To: http://localhost:9080/RAD8JAX-RSWeb/jaxrs/transaction
    PostedData: {"transType":"credit","accountId":"008-888001","amount":450}
Received Message: Bad Request
    Received Entity: {"error":"credit should be Debit or Credit"}

    Posting To: http://localhost:9080/RAD8JAX-RSWeb/jaxrs/transaction
    PostedData: {"transType":"credit","accountId":"NonExistingAccount","amount":450}
Received Message: Bad Request
    Received Entity: {"error":"No Account Found"}

    Getting: http://localhost:9080/RAD8JAX-RSWeb/jaxrs/accounts/008-888001/transactions
Received Message: OK
    Received Entity: {"transType":"Debit","transTime":"2010-10-26 22:07:17.968","amount":"100.00","id":"48d2768f"}
    {"transType":"Credit","transTime":"2010-10-26 22:07:18.687","amount":"450.00","id":"7e7d7542"}

```

Figure 14-54 Results of invoking *MakeTransactionsClient.java*

You can locate the completed application in this file:

C:\7835codesolution\webservices\RAD8JAX-RS.zip.

14.17 Web services security

Web services security for WebSphere Application Server v8.0 Beta is based on standards included in the Organization for the Advancement of Structured Information Standards (OASIS) Web Services Security (WS-Security) Version 1.0/1.1 specification, the Username Token Profile 1.0/1.1, and the X.509 Certificate Token Profile 1.0/1.1.

WS-Security addresses three major issues involved in securing SOAP message exchanges: authentication, message integrity, and message confidentiality.

14.17.1 Authentication

Authentication is used to ensure that parties within a business transaction are really who they claim to be; thus, proof of identity is required. This proof can be claimed in the following ways:

- ▶ Presenting a user identifier and password (referred to as a *username token* in the WS-Security domain)
- ▶ Using an X.509 certificate issued by a trusted certificate authority

The *certificate* contains identity credentials and has a pair of private and public keys associated with it. The proof of identity presented by a party includes the certificate itself and a separate piece of information that is digitally signed using the certificate's private key. By validating the signed information using the public key associated with the party's certificate, the receiver can authenticate the sender as being the owner of the certificate, thereby validating the sender's identity.

Two WS-Security specifications, the Username Token Profile 1.0/1.1 and the X.509 Certificate Token Profile 1.0/1.1, explain how to use these authentication mechanisms with WS-Security.

14.17.2 Message integrity

To validate that a message has not been tampered with or corrupted during its transmission over the Internet, the message can be digitally signed by using security keys. The sender uses the private key of its X.509 certificate to digitally sign the SOAP request. The receiver uses the sender's public key to check the

signature and identity of the signer. The receiver signs the response with its private key. The sender can validate that the response has not been tampered with or corrupted by using the receiver's public key to check the signature and identity of the responder.

14.17.3 Message confidentiality

To keep the message safe from eavesdropping, encryption technology is used to scramble the information in web services requests and responses. The encryption ensures that no one accesses the data in transit, in memory, or after it has been persisted, unless that person has the private key of the recipient. The WS-Security: SOAP Message Security 1.0/1.1 specification describes enhancements to SOAP messaging to provide message confidentiality.

Two options are available to configure WS-Security for JAX-WS web services:

- ▶ Policy sets
- ▶ Programming API for securing SOAP message with the WS-Security API and Service Programming Interfaces (SPI) for a service provider

We use policy sets in our examples.

14.17.4 Policy set

You can use policy sets to simplify configuring the qualities of service for web services and clients. *Policy sets* are assertions about how web services are defined. By using policy sets, you can combine configurations for separate policies. You can use policy sets with JAX-WS applications, but not with JAX-RPC applications.

A policy set is identified by a unique name. An instance of a policy set consists of a collection of policy types. An empty policy set has no policy instance defined.

Policies are defined on the basis of a quality of service (QoS). Policy definitions are typically based on the WS-Policy standards language. For example, the WS-Security policy is based on the current WS-SecurityPolicy language from the OASIS standards.

Policy sets omit application or user-specific information, such as keys for signing, keystore information, or persistent store information. Instead, application and user-specific information is defined in the bindings. Typically, bindings are specific to the application or the user, and bindings are not normally shared. On the server side, if you do not specify a binding for a policy set, a default binding is

used for that policy set. On the client side, you must specify a binding for each policy set.

A *policy set attachment* defines which policy set is attached to service resources, and which bindings are used for the attachment. The bindings define how the policy set is attached to the resources. An attachment is defined outside of the policy set, as metadata associated with the application. To enable a policy set to work with an application, a binding is required.

14.17.5 Applying WS-Security to a web service and client

In this section, we apply the Username WS-Security default policy set to a web service and client. This policy set provides the following features:

- ▶ Message integrity by digital signature (using the Rivest-Shamir-Adleman (RSA) algorithm public-key cryptography) to sign the body, time stamp, and WS-Addressing headers using the WS-Security specifications.
- ▶ Message confidentiality by encryption (using RSA public-key cryptography) to encrypt the body, signature, and signature confirmation elements using the WS-Security specifications.
- ▶ A username token included in the request message to authenticate the client to the service. The username token is encrypted in the request.

Sample bindings for JAX-WS applications

WebSphere Application Server v8.0 Beta includes provider and client sample bindings for testing purposes. In the bindings, the product provides sample values for supporting tokens for various token types, such as the X.509 token and the username token. The bindings also include sample values for message protection information for token types, such as X.509. Both provider and client sample bindings can be applied to the applications attached with a policy set.

In a production environment, you must modify the bindings to meet your security needs before using them in a production environment by making a copy of the bindings and then modifying the copy. For example, you must change the key and keystore settings to ensure security, and you must modify the binding settings to match your environment.

Configuring the username token

When using the Username WS-Security default policy set, you must configure the user name and password for username token authentication separately from the security settings defined in the bindings. The sample binding does not include a user name or password for token authentication, because it is specific

to the target deployed system. You must specify a valid user name and password in your environment using the WebSphere administrative console:

1. In the Servers view, right-click **WebSphere Application Server v8.0 Beta** and select **Administration** → **Run administrative console**.
2. Log in with the user ID and password (admin). We assume that your WebSphere Profile is secured and that the administrator user is called admin.
3. Select **Services** → **Policy sets** → **General client policy set bindings** (Figure 14-55).

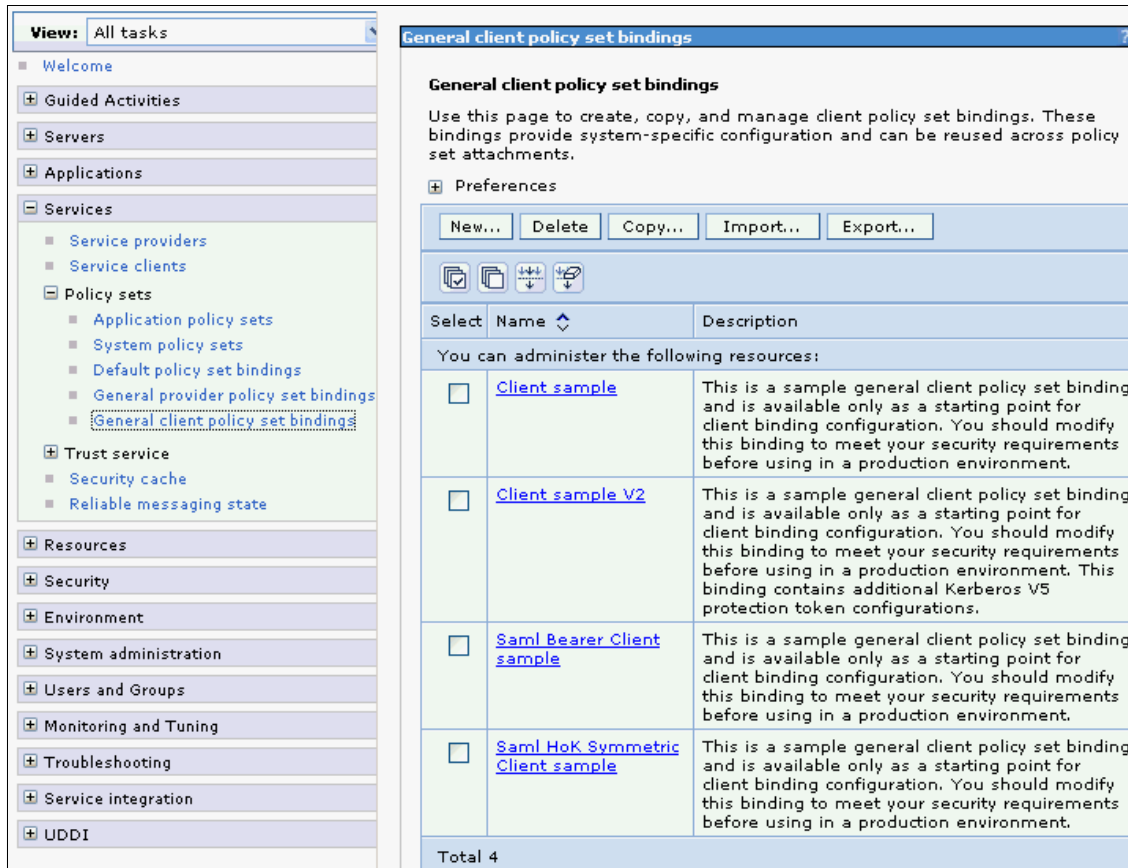


Figure 14-55 General Client policy set bindings

4. Click **Client sample** to edit the binding.
5. Click **WS-Security**.
6. Click **Authentication and protection**.

7. In the Authentication tokens list, select **gen_signunametoken** to edit the username token settings.
8. In the Additional Bindings section (bottom), click **Callback handler**.
9. For User name, enter admin. For password, enter admin and confirm the password. Click **Apply** (Figure 14-56).
10. Click **Save** and then click **Logout**.

General client policy set bindings

[General client policy set bindings](#) > [Client sample](#) > [WS-Security](#) > [Authentication and protection](#) > [gen_signunametoken](#) > **Callback handler**

Specifies the parameters for the callback handler that are used for generating the token. Because you can plug-in a custom callback handler, you must specify the implementation class name. The application server provides options for identity assertion, basic authentication, and the keystore that are passed to the callback handler implementation.

Class Name

Use built-in default

Use custom

Basic Authentication

User name

Password

Confirm password

Figure 14-56 Setting Basic Authentication for gen_signunametoken Callback Handler

Attaching the Username WS-Security policy set to the web service

To attach the Username WS-Security default policy set to the web service, follow these steps:

1. In the Java EE perspective, in the Services view (Figure 14-57 on page 801), expand the **JAX-WS** node. Right-click **RAD8WebServiceWeb:{...}BankService** and select **Manage Policy Set Attachments** → **Server Side**.

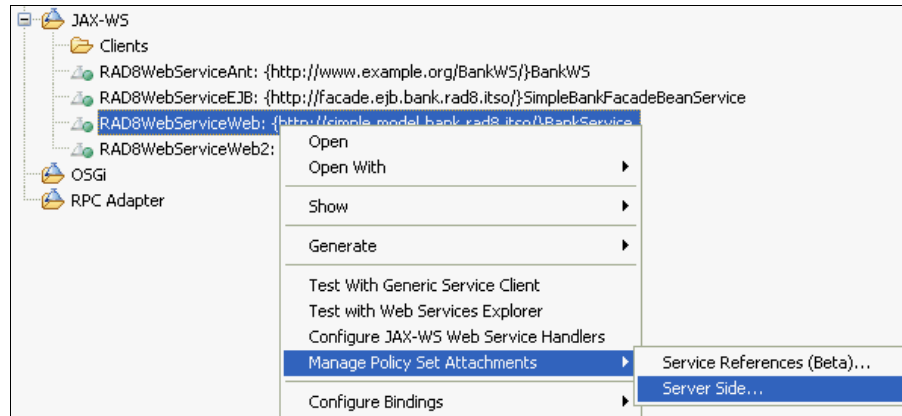


Figure 14-57 Manage Policy Set Attachments

2. In the Add Policy Set Attachment to Service window, click **Add**.
3. In the End Point Definition Dialog window (Figure 14-59 on page 802), for Policy Set, select **Username WSSecurity default**, and for Binding, ensure that **Provider Sample** is selected. This binding is a service-side general binding packaged with WebSphere Application Server. Click **OK**.

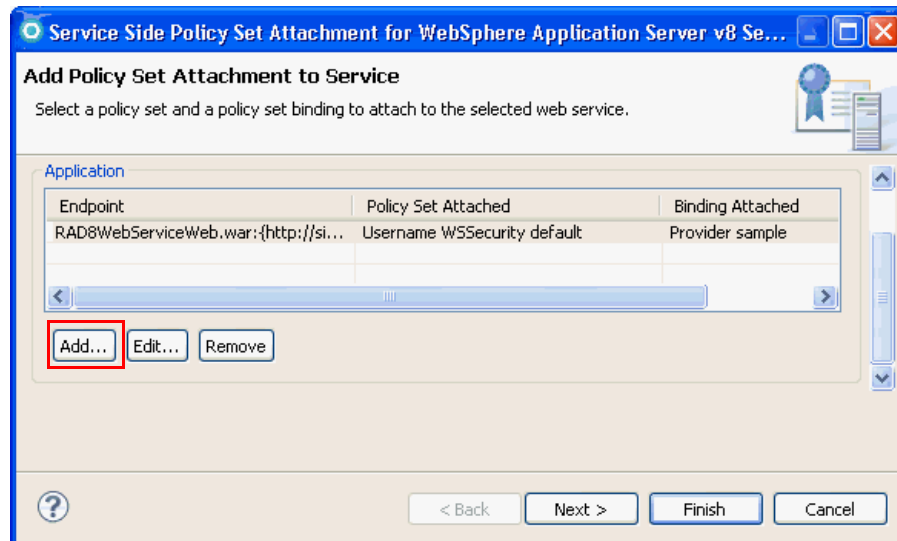


Figure 14-58 Add Policy Set attachment to service

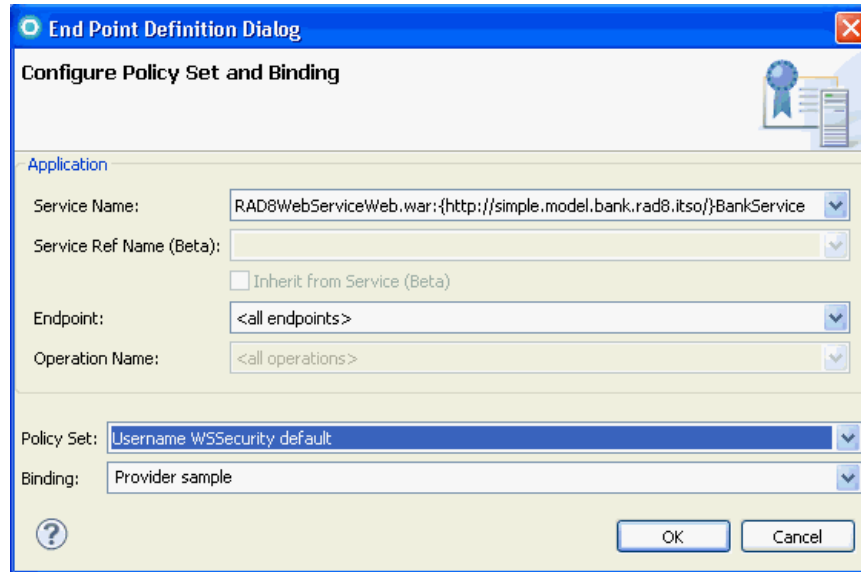


Figure 14-59 Configure Policy Set and Binding

You can apply a policy set at the service, port, or operation level. Separate policy sets can be applied to various endpoints and operations within a single web service. However, the service and client must have the same policy set settings. For this example, we apply the policy set to the entire service, so the Endpoint and Operation Name fields are left blank.

4. When the warning message is displayed, click **Ignore**. WS-Security was included in the WS-I Basic Security Profile. The WS-I Basic Security Profile Version 1.0 was in Final Material status.
5. Back in the Add Policy Set Attachment to Service window, click **Finish**. Notice that the service application is republished to the server.

Attaching the policy set to the web service client

To attach the Username WS-Security default policy set to the web service client, follow these steps:

1. In the Services view, expand the **JAX-WS** → **Clients**. Right-click **RAD8WebServiceClient: service/BankService** and select **Manage Policy Set Attachment**.
2. In the Configure Policy acquisition for Web Service Client window, click **Next** (Figure 14-60 on page 803). In this scenario, we do not want to acquire the policy from the Provider, so do not select Use Provider Policy.

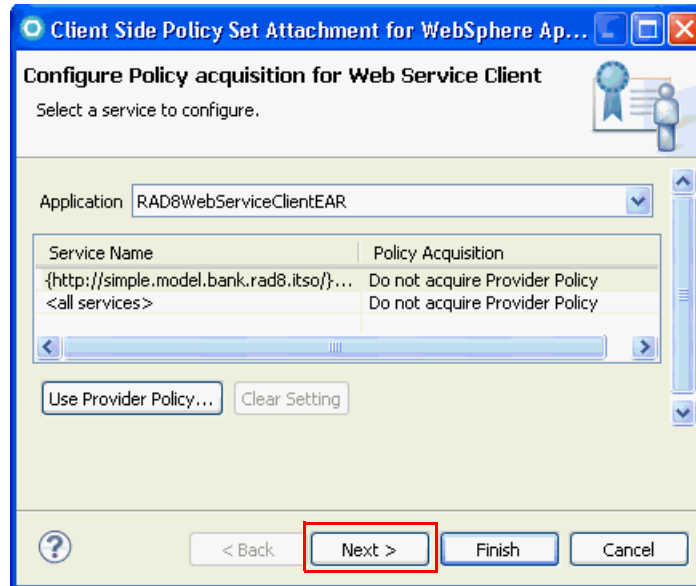


Figure 14-60 Configure Policy Acquisition for Web Service Client window

3. In the Add Policy Set Attachment to Client window, click **Add** (Figure 14-61 on page 804) to attach a policy set to the endpoint and to specify the bindings. The dialog window initially has no entries, and Figure 14-61 on page 804 shows the result after the addition is complete.

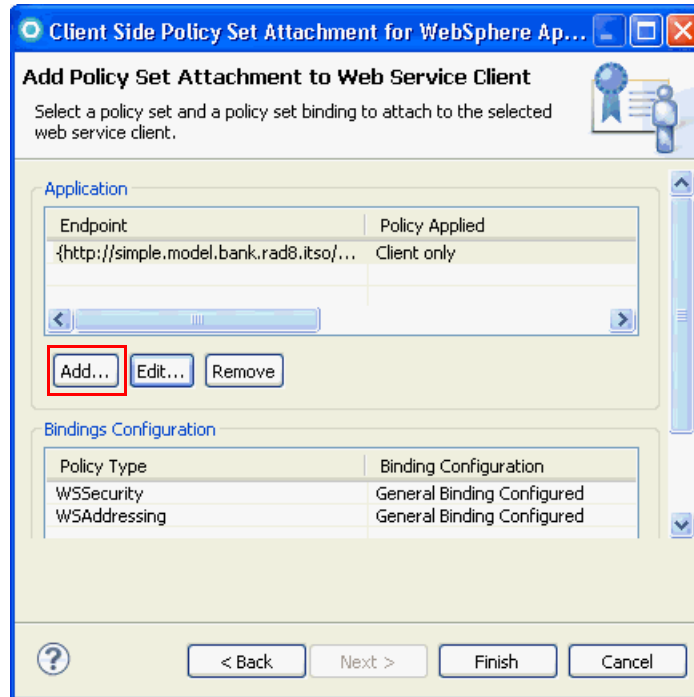


Figure 14-61 Add Policy Set Attachment to Web Service Client window

4. In the End Point Definition Dialog: Configure Policy Set and Binding window (Figure 14-62 on page 805), accept the settings for Service Name (**BankService**), Endpoint (**all**), Policy Set (**Username WSSecurity default**), and Binding (**Client sample**). This binding is a client-side general binding packaged with WebSphere Application Server.
5. Click **OK**.

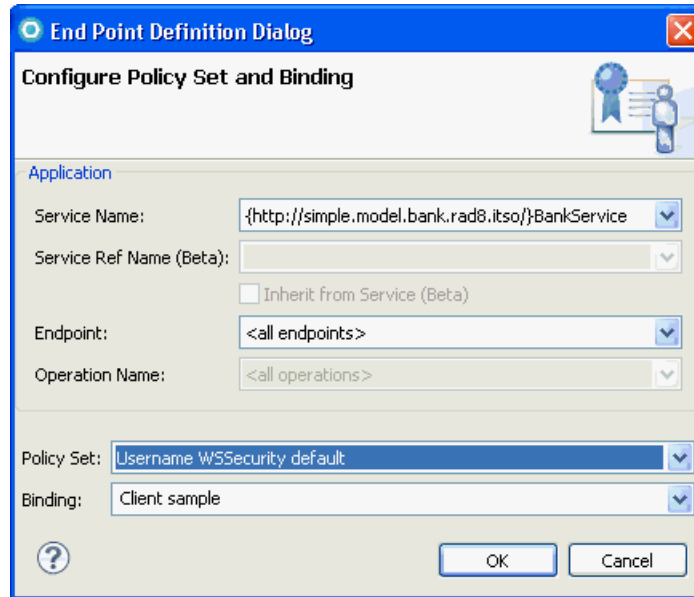


Figure 14-62 Configure Policy Set and Binding window

6. In the message window that opens, click **Ignore**.

The policy types contained by the policy set that you selected are listed in the Bindings Configuration table. The configurations for these policy types are already complete.

7. Click **Finish** to complete the wizard.

Testing the secured web service

To test the secured web service, follow these steps:

1. Select **Window** → **Preferences: Run/Debug** → **TCP/IP Monitor**. Make sure the TCP/IP Monitor is started. Make a note of the monitor port, because you need to reuse it in step 4. Let us call the monitor port `xxxx` for future reference.
2. In the Enterprise Explorer, expand the **RAD8WebServiceClient** project, right-click **TestClient.jsp**, and select **Run As** → **Run on Server**.
3. Select the **v8.0 Beta** server and click **Finish**.
4. In the sample JSP client, Quality of Service pane, change the endpoint to the monitor port and click **Update**:
`http://localhost:xxxx/RAD8WebServiceWeb/BankService`
5. Invoke the **retrieveCustomerName** with a customer number of 111-11-1111.

- In the TCP/IP Monitor view (Figure 14-63), verify that the message is signed and encrypted and that the username token in the SOAP header is encrypted.

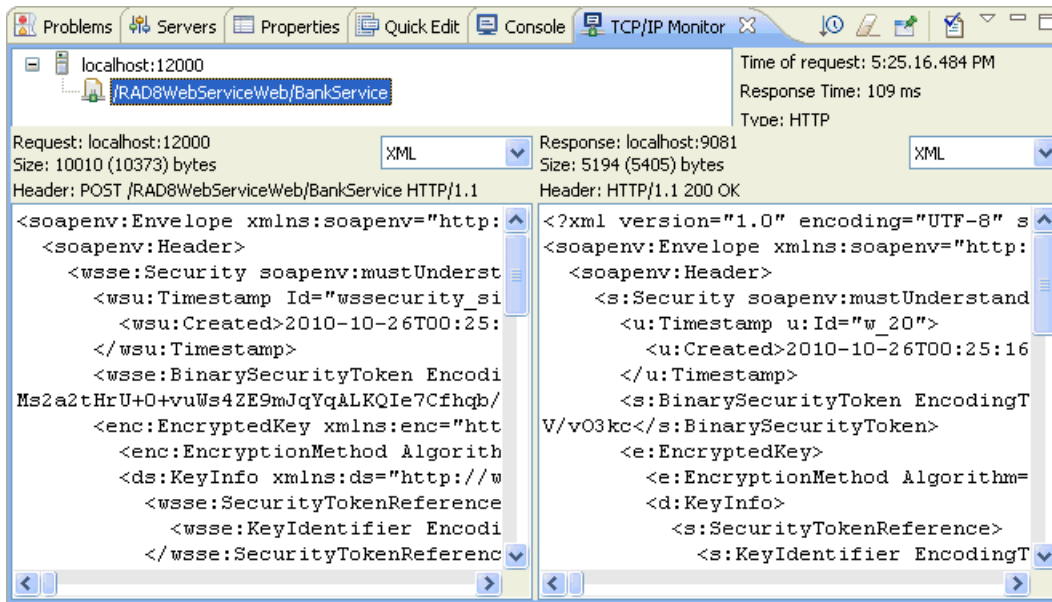


Figure 14-63 TCP/IP Monitor showing signed and encrypted message

Figure 14-43 shows a snippet of the Request.

Example 14-43 Request snippet

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wss
ecurity-secext-1.0.xsd">
      <wsu:Timestamp Id="wssecurity_signature_id_21"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wsse
curity-utility-1.0.xsd">
        <wsu:Created>2010-10-26T00:25:16.453Z</wsu:Created>
      </wsu:Timestamp>
      <wsse:BinarySecurityToken
EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-s
oap-message-security-1.0#Base64Binary"
ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509
-token-profile-1.0#X509v3" Id="x509bst_23">
```

```
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">MIICQzCCAygAwIBA.....  
</wsse:BinarySecurityToken>  
.....
```

You can obtain the complete request and response at these locations:

- ▶ C:\7835codesolution\webservices\UserNameTokenRequest.txt
- ▶ C:\7835codesolution\webservices\UserNameTokenResponse.txt

You can obtain the secured projects archive (where the irrelevant projects have been removed from the two EARs) in this folder:

C:\7835codesolution\webservices\RAD8WSUsernameToken.zip

14.17.6 WS-I Reliable Secure Profile

Continuing from the Reliable Asynchronous Messaging Profile (RAMP) Version 1.0 specification, the Reliable Secure Profile (RSP) working group of the WS-I organization has developed Version 1.0 of an interoperability profile that works with secure and reliable messaging capabilities for web services.

WS-I Reliable Secure Profile 1.0 provides secure reliable session-oriented web services interactions. WS-I Reliable Secure Profile 1.0 builds on WS-I Basic Profile 1.2, WS-I Basic Profile 2.0, WS-I Basic Security Profile 1.0, and WS-I Basic Security Profile 1.1. It adds support for WS-Reliable Messaging 1.1, WS-Make Connection 1.0, and WS-Secure Conversation 1.3:

- ▶ WS-Reliable Messaging 1.1 is a session-based protocol that provides message-level reliability for web services interactions.
- ▶ WS-Make Connection 1.0 was developed by the WS-Reliable Messaging workgroup to address scenarios in which a web services endpoint is behind a firewall or the endpoint has no visible endpoint reference. If a web services endpoint loses connectivity during a reliable session, WS-Make Connection provides an efficient method to re-establish the reliable session.
- ▶ WS-Secure Conversation 1.3 is a session-based security protocol that uses an efficient symmetric key-based encryption algorithm for message-level security.

The configuration steps to apply the WS-I RSP policy set are similar to the steps for the Username WS-Security policy set. Select WS-I RSP for Policy Set when adding a policy set attachment to the service. We leave it as an exercise for you to explore this functionality.

14.18 WS-Policy

The Web Services Policy Framework (WS-Policy) specification is an interoperability standard that is used to describe and communicate the policies of a web service so that service providers can export policy requirements in a standard format. Clients can combine the service provider requirements with their own capabilities to establish the policies that are required for a specific interaction.

WebSphere Application Server conforms to the Web Services Policy Framework (WS-Policy) specification. You can use the WS-Policy protocol to exchange policies in a standard format. A policy represents the capabilities and requirements of a web service, for example, whether a message is secure and how to secure it, and whether a message is delivered reliably and how to deliver a message reliably. You can communicate the policy configuration to any other client, service registry, or service that supports the WS-Policy specification, including non-WebSphere Application Server products in a heterogeneous environment.

For a service provider, the policy configuration can be shared in a published WSDL that is obtained by a client using an HTTP get request or by using the Web Services Metadata Exchange (WS-MetadataExchange) protocol. The WSDL is in the standard WS-PolicyAttachments format.

For a client, the client can obtain the policy of the service provider in the standard WS-PolicyAttachments format and use this information to establish a configuration that is acceptable to both the client and the service provider. That is, the client can be configured dynamically, based on the policies supported by its service provider. The provider policy can be attached at the application or service level.

Relationship to policy set: Policy sets are not inherently concerned with the WS-Policy specification, but work with the configuration of web services and need to be considered as a front end to WS-Policy. Policy sets provide a mechanism to specify a policy within a WebSphere environment. They do not provide a mechanism to communicate this policy to non-WebSphere partners in a heterogeneous environment. In addition, policy set functionality does not provide a mechanism for the client to calculate effective policy (that is, a policy that is acceptable to both client and provider) based the intersection of a list of client and provider policies.

14.18.1 Configuring a service provider to share its policy configuration

Configure a service provider to share its policy configuration:

1. In the Services view, right-click **RAD8WebServiceWeb:{...}BankService** and select **Manage Policy Set Attachment**.
2. In the next window, verify that the *username* WS-Security default is listed as the attached policy set from the last section. Click **Next**.
3. In the Configure Policy Sharing window (left window in Figure 14-64), select the service and click **Configure**. In the Configure Policy Sharing for Web Service window (right window in Figure 14-64), select **Share Policy Information via WSDL** and click **OK**.

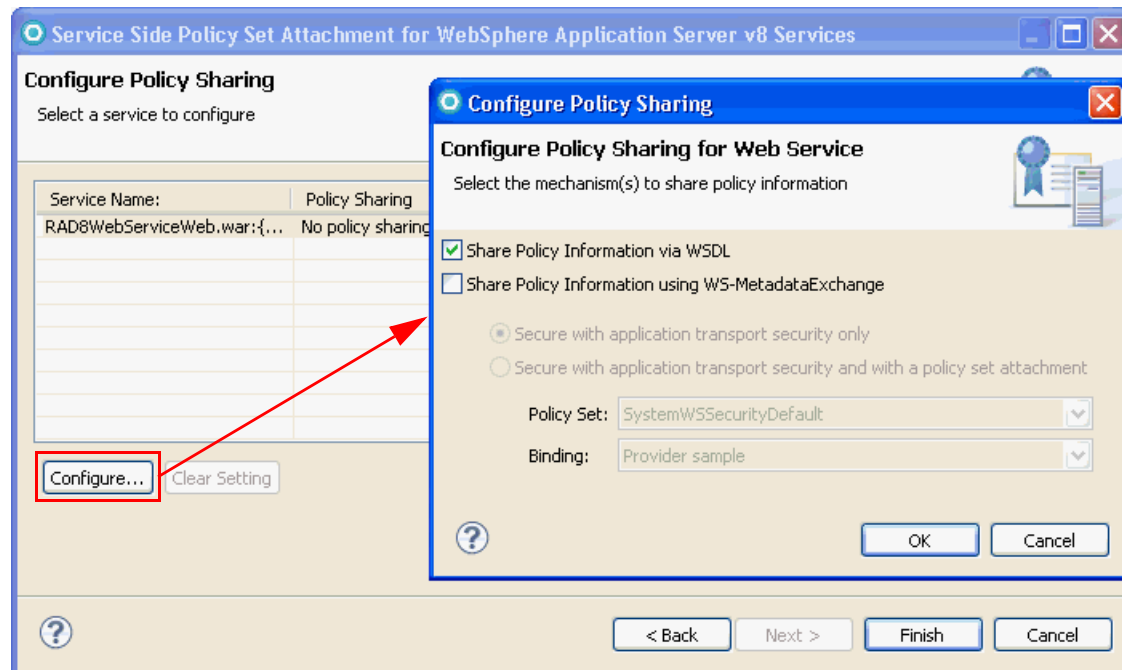


Figure 14-64 Configure Policy Sharing windows

4. Click **Ignore** for the warning and then click **Finish**.
5. After the server is published, open a browser and enter the following URL in the browser (908x is the port number, which is likely 9080):
`http://localhost:908x/RAD8WebServiceWeb/BankService?wsdl`

The WS-Policy information is embedded in the WSDL document (Example 14-44). You can see that the policy configured for the input message includes UsernameToken.

Example 14-44 WS-Policy in WSDL

```
.....
<binding name="BankPortBinding" type="tns:Bank">
  <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsp:PolicyReference URI="#b2670004-122a-4028-ab78-bf5d286647d6"/>
  <operation name="RetrieveCustomerName">
    <soap:operation soapAction="urn:getCustomerFullName"/>
    <input>
      <soap:body use="literal"/>
      <wsp:PolicyReference URI="#402c7f57-35bb-435d-98bb-f9e3575f4d3e"/>
    </input>
    <output>
      <soap:body use="literal"/>
      <wsp:PolicyReference URI="#b7dd0e57-5b38-4ecd-9f67-10d6c32da5b5"/>
    </output>
    <fault name="CustomerDoesNotExistException">
      <soap:fault name="CustomerDoesNotExistException"
use="literal"/>
    </fault>
  </operation>
</binding>
....
<wsp:Policy wsu:Id="402c7f57-35bb-435d-98bb-f9e3575f4d3e">
  <ns2:SignedParts

xmlns:ns2="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
  <ns2:Body />
  <ns2:Header
Namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing" />
  <ns2:Header Namespace="http://www.w3.org/2005/08/addressing" />
</ns2:SignedParts>
  <ns2:EncryptedParts

xmlns:ns2="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
  <ns2:Body />
  </ns2:EncryptedParts>
  <ns2:SignedEncryptedSupportingTokens

xmlns:ns2="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
```

```

    <wsp:Policy>
      <ns2:UsernameToken
        ns2:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/20
        0702/IncludeToken/AlwaysToRecipient">
        <wsp:Policy>
          <ns2:WssUsernameToken10 />
        </wsp:Policy>
      </ns2:UsernameToken>
    </wsp:Policy>
  </ns2:SignedEncryptedSupportingTokens>
</wsp:Policy>
....
</definitions>

```

You can see the complete WSDL here:

C:\7835codesolution\webservices\BankServiceWithUserNameTokenPolicy.wsdl

14.18.2 Configuring the client policy using a service provider policy

To configure the client policy using a service provider policy, follow these steps:

1. Remove the policy that you applied in the previous section, because we use WS-Policy to request the service provider's policy information:
 - a. In the Services view, right-click **RAD8WebServiceClient:service/BankService** and select **Manage Policy Set Attachment**.
 - b. Click **Next**.
 - c. Click **Remove** and then click **Finish**.
2. Right-click **RAD8WebServiceClient: service/BankService** and select **Manage Policy Set Attachment**.
3. In the first Configure Policy acquisition for Web Service Client window (Figure 14-65 on page 812), click **Use Provider Policy**.

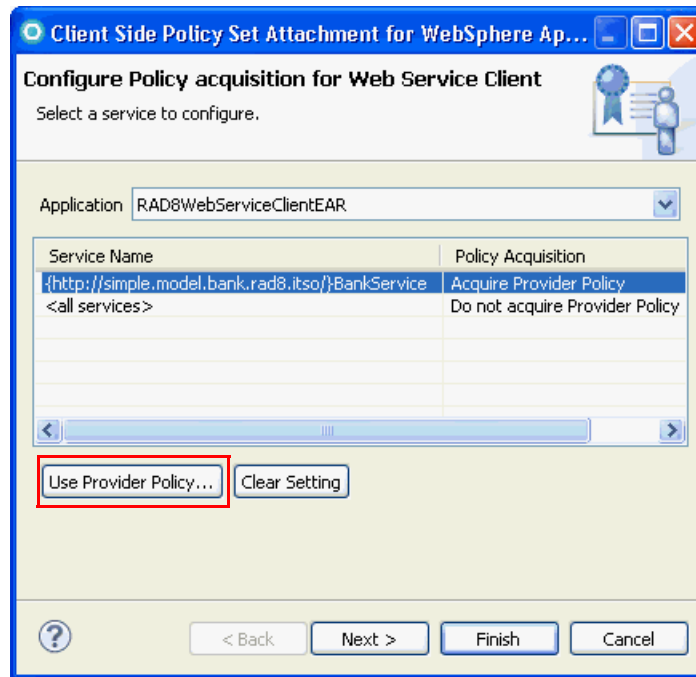


Figure 14-65 Use Provider Policy window

4. In the Configure Policy acquisition for Web Service Client window (Figure 14-66 on page 813), select **HTTP Get request targeted at <default WSDL URL>** and click **OK**. The Policy Acquisition field for the service changes to Acquire Provider Policy in Figure 14-65.

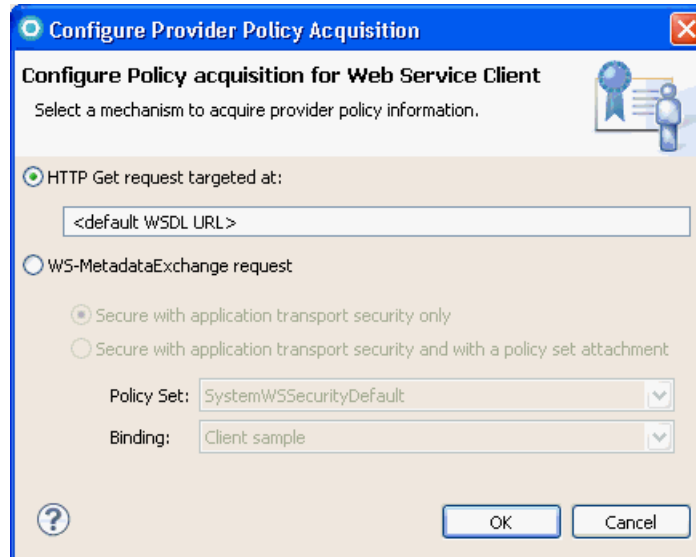


Figure 14-66 Configure Policy Acquisition for Web Service Client window

5. In the warning message window that opens, click **Ignore** and then click **Finish**.
6. Test the web service again. In the TCP/IP Monitor, you can see that the client first acquires the WSDL through the HTTP GET (Figure 14-67 on page 814). The client policy calculations for a service are performed at the first invocation on that service. Calculated policies are cached in the client for performance.

To configure a service provider to share its policy configuration using WS-MEX, follow these steps:

1. In the Services view, right-click **RAD8WebServiceWeb:{...}BankService** and select **Manage Policy Set Attachment**.
2. Verify that the *username* WSSecurity default is listed as the attached policy set from the previous section. Click **Next**.
3. In the Configure Policy Sharing window, select the service and click **Configure**.
4. In the Configure Policy Sharing for Web Service window (Figure 14-68), select **Share Policy Information using WS-MetadataExchange** and click **OK**.

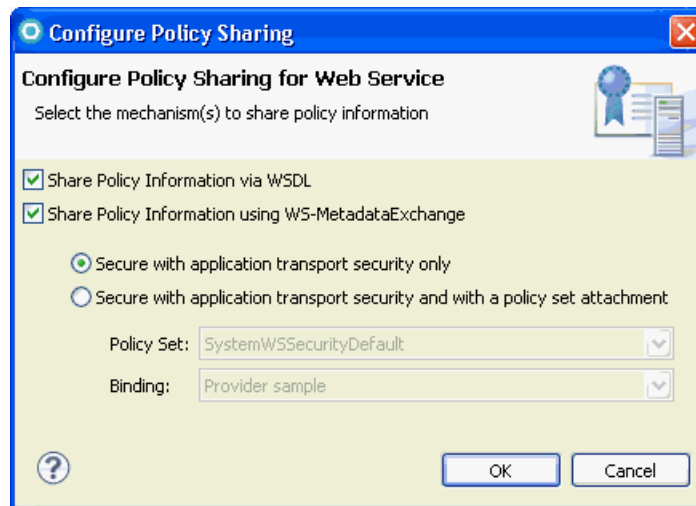


Figure 14-68 Sharing policy set using WS-MetadataExchange

5. In the warning message window that opens, click **Ignore** and click **Finish**.

To configure the client policy configuration using WS-MEX, follow these steps:

1. Right-click **RAD8WebServiceClient: service/BankService**, select **Manage Policy Set Attachment** and click **Use Provider Policy**.
2. In the Configure Policy acquisition for Web Service Client window, select **WS-MetadataExchange** and click **OK**.
3. In the warning message window that opens, click **Ignore** and click **Finish**.
4. Test the web service again. In the TCP/IP Monitor, you can see that the client first issues a WS-MEX GetMetadata request to the actual web service endpoint and that the dialect of the request is WSDL (Example 14-45 on page 816).

Example 14-45 WS-MEX request

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">

<wsa:To>http://wxpsp408:12000/RAD8WebServiceWeb/BankService</wsa:To>

<wsa:MessageID>urn:uuid:227725c2-1602-400d-9449-50a0e966058b</wsa:Me
ssageID>

<wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata/Re
quest</wsa:Action>
  </soapenv:Header>
  <soapenv:Body>
    <b>mex:GetMetadata</b>
    xmlns:mex="http://schemas.xmlsoap.org/ws/2004/09/mex">
      <mex:Dialect>http://schemas.xmlsoap.org/wsdl/</mex:Dialect>
    </mex:GetMetadata>
  </soapenv:Body>
</soapenv:Envelope>
```

The GetMetadata response returns the WSDL with the policy information (Figure 14-69 on page 817). Then you see a second request in the TCP/IP Monitor with the actual request and response.

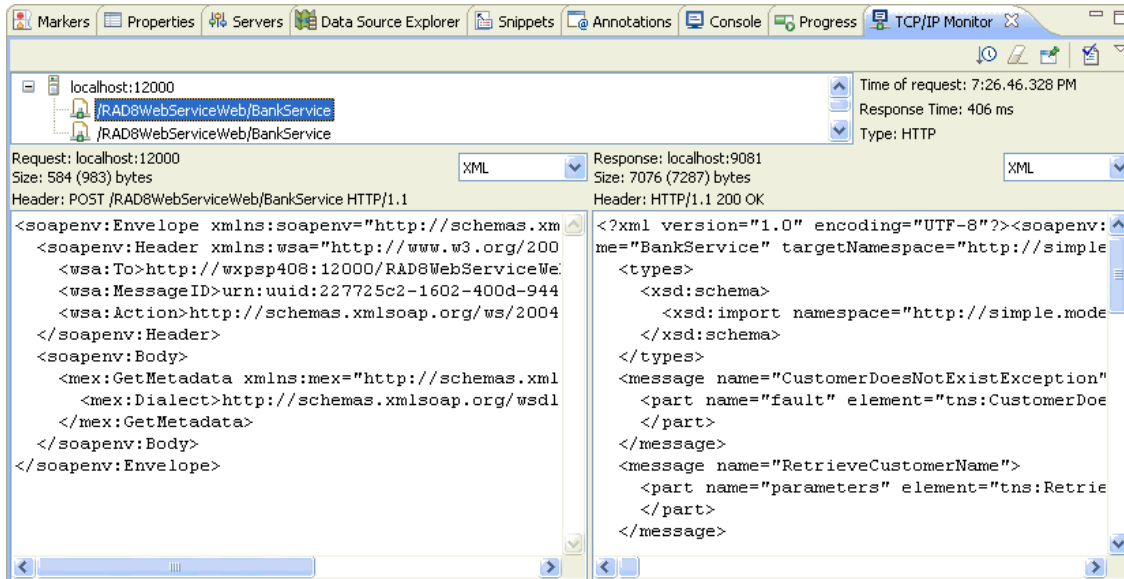


Figure 14-69 TCP/IP Monitor showing WS-MEX Request

14.20 Security Assertion Markup Language (SAML) support

The Security Assertion Markup Language (SAML) is an XML-based OASIS standard for exchanging user identity and security attributes information.

Using the product SAML function, you can apply policy sets to JAX-WS applications to use SAML assertions in web services messages and in web services usage scenarios. You can use SAML assertions to represent user identity and user security attributes, and optionally, to sign and to encrypt SOAP message elements. WebSphere Application Server supports SAML assertions using the bearer subject confirmation method and the holder-of-key subject confirmation method as defined in the OASIS Web Services Security SAML Token Profile Version 1.1 specification. Policy sets and general bindings that support SAML are included with the product SAML function. To use SAML assertions, you must modify the provided sample general binding.

The SAML function also provides a set of application programming interfaces (APIs) that can be used to request SAML tokens from a Security Token Service (STS) using the WS-Trust protocol. APIs are also provided to locally generate and validate SAML tokens.

14.20.1 SAML assertions defined in the SAML Token Profile standard

The Web Services Security SAML Token Profile OASIS standard specifies how to use Security Assertion Markup Language (SAML) assertions with the Web Services Security SOAP Message Security specification.

WebSphere Application Server Version 7.0.0.7 and later supports two versions of the OASIS SAML standard: Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1, and Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0.

The standard describes the use of SAML assertions as security tokens in the <wsse:Security> header, as defined by the WSS: SOAP Message Security specification. An XML signature can be used to bind the subjects and statements in the SAML assertion to the SOAP message.

Subject confirmation methods

Subject confirmation methods define the mechanism by which an entity provides evidence (proof) of the relationship between the subject and the claims of the SAML assertions. The WSS: SAML Token Profile describes the usage of three subject confirmation methods: *bearer*, *holder-of-key*, and *sender-vouches*. WebSphere Application Server Version 7.0.0.9 and later versions support all three confirmation methods.

Bearer

When using the bearer subject confirmation method, proof of the relationship between the subject and claims is implicit. No specific steps are taken to establish the relationship.

Because no key material is associated with a bearer token, protection of the SOAP message, if required, must be performed by using a transport-level mechanism or another security token, such as an X.509 or Kerberos token, for message-level protection.

Holder-of-key

When using the holder-of-key subject confirmation method, proof of the relationship between the subject and claims is established by signing part of the SOAP message with the key specified in the SAML assertion. Because there is key material associated with a holder-of-key token, this token can be used to provide message-level protection (signing and encryption) of the SOAP message.

Sender-vouches

The sender-vouches confirmation method is used when a server needs to propagate the client identity with SOAP messages on behalf of the client. This method is similar to identity assertion, but it has the added flexibility of using SAML assertions to propagate not only the client identity, but also propagate client attributes. The attesting entity must protect the vouched for SAML assertions and SOAP message content so that the receiver can verify that it has not been altered by another party.

14.20.2 SAML APIs

The SAMLTokenFactory API is the major SAML token programming interface. Using this API, you can create SAML tokens, insert SAML attributes, parse and validate SAML assertions as XML representations for the SAML tokens, and create Java Authentication and Authorization Service (JAAS) subjects that represent user identity and attributes as defined in SAML tokens. For more information, refer to the WebSphere Application Server Information Center and look for these classes:

- ▶ `com.ibm.websphere.wssecurity.wssapi.token.SAMLTokenFactory`
- ▶ `com.ibm.websphere.wssecurity.wssapi.token.SAMLToken`

http://www14.software.ibm.com/webapp/wsbroker/redirect?version=compass&product=was-base-dist&topic=cwbs_overviewsamlapis

14.20.3 SAML Bearer sample: Prerequisites

This sample shows how you can bind the SAML11 Bearer WSHTTPS default policy set, which contains the items listed in Table 14-5.

Table 14-5 SAML11 Bearer WSHTTPS default policy set

SAML11 Bearer WSHTTPS default	
Policies	HTTP transport, SSL transport, WS-Addressing, and WS-Security
Transport security	Using SSL for HTTP
Message authentication	Using SAML 1.1 token with bearer confirmation method

The client is configured to generate a SAML Bearer Token, and the service is configured to consume it. The client makes use of the SAML APIs to create the SAML Token programmatically.

The following steps are required if you use WebSphere Application Server V8 Beta:

1. Launch the administrative console.
2. Select **Services** → **Policy Sets** → **Application Policy Sets**.
3. Select **SAML11 Bearer WSHTTPS default**.
4. Select **Import** → **From Default Repository** and select **OK**.
5. Select **Save**.
6. Optional: Explore the other relevant features that are predefined:
 - Verify that in **Services** → **General Client Policy Set Bindings**, you have these bindings:
 - Saml Bearer Client sample
 - Saml HoK Symmetric Client sample
 - Verify that **Security** → **Global security** → **Java Authentication and Authorization Service** → **System logins** contains these logins:
 - wss.consume.saml
 - wss.generate.saml
7. Log out of the administrative console.
8. Restart WebSphere Application Server.

If you use WebSphere Application Server V7, additional steps are required as described here:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/twbs_setupsamlconfig.html

Additionally, you must export the same policy set from WebSphere Application Server and import it into your Rational Application Developer workspace with the following steps:

1. Launch the administrative console.
2. Select **Services** → **Policy Sets** → **Application Policy Sets**.
3. Select **SAML11 Bearer WSHTTPS default**.
4. Select **Export**.
5. A new page opens with a hyperlink to the file SAML11 Bearer WSHTTPS default.zip.
6. Right-click the hyperlink and use your browser menu to save the target locally (typically, **Save target as**). Make note of where you save the file.
7. In Rational Application Developer, select **File** → **Import**.

8. Select **WebServices** → **WebSphere Policy Sets** and select **Next**.
9. Browse to the **SAML11 Bearer WSHTTPS default.zip** file.
10. Select **Finish**.

You must repeat the import operation in each workspace where you want to be able to associate this policy set to a service or client.

14.20.4 SAML Bearer sample: Bindings

This sample is based on the sample binding that is contained the product. Select **Help** → **Help Contents: Samples** → **WebServices** → **WebSphere JAX-WS address book SAML Web service**. From here, we reuse the supplied bindings and the code to generate a SAML token in the client application:

1. Import the **WebSphere JAX-WS address book SAML Web service** sample, which creates the following projects:
 - SAMLBearer_AddressBook
 - SAMLBearer_AddressBookClient
 - SAMLBearer_AddressBookEAR

We leave it for you to test this sample. In the remainder of this section, we show how to reuse the bindings for the RAD8TopDownBankEAR application, which we developed in 14.13, “Creating a top-down web service from a WSDL” on page 749.

2. If you no longer have it in your workspace, import the archive:
C:\7835codesolution\webservices\RAD8TopDownWebService.zip
3. Generate a service client for RAD8TopDownWebService:
 - a. In the Services view, right-click **RAD8TopDownBankWS**.
 - b. Select **Generate** → **Client**.
 - c. Move the slider to the **Test** position, because we want to generate a test of the JAX-WS JSP.
 - d. Change the client project to RAD8TopDownBankWSCClient.
 - e. Change the EAR project to RAD8TopDownBankWSCClientEAR.
 - f. Select **Next**.
 - g. Select **Next**.
 - h. On the Web Service Client Test page, make sure that you accept **JAX-WS JSPs** for Test Facility and accept all defaults.
 - i. Select **Finish**.

4. Expand **SAMLEearer_AddressBookEAR** in the Enterprise Explorer.
5. Expand the **META-INF** folder.
6. You see two folders that contain application-specific bindings for the service provider and for the service client:
 - SAMLEearerProviderBinding
 - SAMLEearerClientBinding
7. Copy the complete folder SAMLEearerProviderBinding into RAD8TopDownBankWSEAR\META-INF. (The folder META-INF needs to be added if it does not exist.)
8. Open the file named **RAD8TopDownBankWSEAR\META-INF\SAMLEearerProviderBinding\PolicyTypes\WSSecurity\bindings.xml**. The WS-Security Policy Binding Editor (Figure 14-70) shows that the Security Inbound Configuration is configured with the SAML V1.1 Bearer Token Consumer.

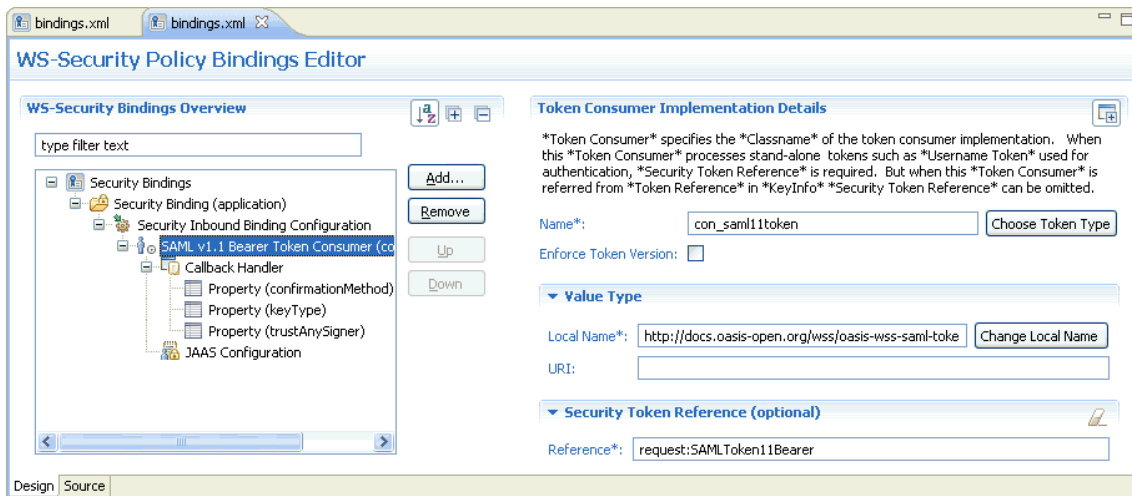


Figure 14-70 WS-Security Policy Binding Editor: Provider WSSecurity Binding

The Callback Handler has the following class name:

```
com.ibm.websphere.wssecurity.callbackhandler.SAMLConsumerCallbackHandler
```

The SAMLConsumerCallbackHandler uses three properties (Table 14-6 on page 823).

Table 14-6 SAMLConsumerCallback Handler properties

Name	Value
Bearer	confirmationMethod
keyType	http://docs.oasis-open.org/ws-sx/ws-trust/200512/Bearer
trustAnySigner	true

The JAAS Configuration uses the `system.wss.consume.saml` JAAS Login Name.

- Example 14-46 shows the complete binding file.

Example 14-46 Provider WSSecurity Binding

```
<?xml version="1.0" encoding="UTF-8"?>
<securityBindings
xmlns="http://www.ibm.com/xmlns/prod/websphere/200710/ws-securitybin
ding">
<securityBinding name="application">
  <securityInboundBindingConfig>
    <tokenConsumer name="con_saml11token"
class="com.ibm.ws.wsssecurity.wssapi.token.impl.CommonTokenConsum
er">
      <valueType
localName="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profi
le-1.1#SAMLV1.1" uri="" />
      <callbackHandler
class="com.ibm.websphere.wsssecurity.callbackhandler.SAMLConsumer
CallbackHandler">
        <properties value="Bearer" name="confirmationMethod"/>
        <properties
value="http://docs.oasis-open.org/ws-sx/ws-trust/200512/Bearer"
name="keyType"/>
        <properties name="trustAnySigner" value="true" />
      </callbackHandler>
      <jAASConfig configName="system.wss.consume.saml"/>
      <securityTokenReference
reference="request:SAMLToken11Bearer"/>
    </tokenConsumer>
  </securityInboundBindingConfig>
</securityBinding>
</securityBindings>
```

10. Copy the complete folder SAMLBearerClientBinding into RAD8TopDownBankWSCClientEAR\META-INF.
11. Open the **RAD8TopDownBankWSCClientEAR\META-INF\SAMLBearerClientBinding\PolicyTypes\WSSecurity\bindings.xml** file. The WS-Security Policy Bindings Editor (Figure 14-71) shows that the Security Outbound Binding Configuration is configured with the SAML V1.1 Bearer Token Generator.

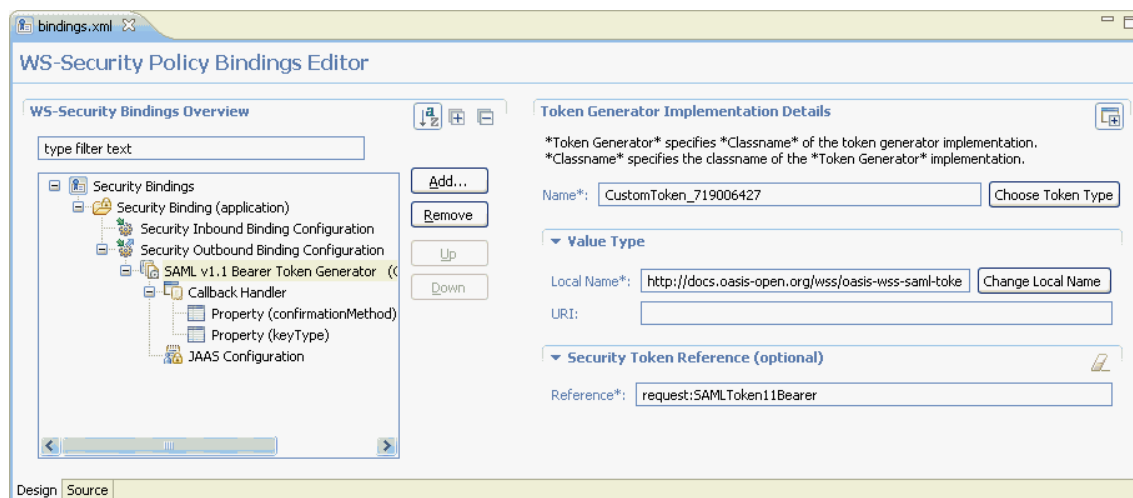


Figure 14-71 WS-Security Policy Bindings Editor for WSSecurity client binding

The JAAS Configuration uses the `system.wss.generate.saml` JAAS Login Name.

The Callback Handler, this time, is `com.ibm.websphere.wsssecurity.callbackhandler.SAMLGenerateCallbackHandler`, and it takes two properties (Table 14-7).

Table 14-7 SAMLGenerateCallbackHandler properties in Client Binding

Name	Value
Bearer	confirmationMethod
keyType	http://docs.oasis-open.org/ws-sx/ws-trust/200512/Bearer

12. Example 14-47 on page 825 shows the complete source of the client binding.

Example 14-47 WSSecurity client binding

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<securityBindings
xmlns="http://www.ibm.com/xmlns/prod/websphere/200710/ws-securitybin
ding">
  <securityBinding name="application">
    <securityOutboundBindingConfig wsuNamespace=""
wsseNamespace="">
      <tokenGenerator
classname="com.ibm.ws.wsssecurity.wssapi.token.impl.CommonTokenGenera
tor" name="CustomToken_719006427">
        <valueType uri=""
localName="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profi
le-1.1#SAMLV1.1"/>
        <securityTokenReference
reference="request:SAMLToken11Bearer"/>
        <jAASConfig configName="system.wss.generate.saml"/>
        <callbackHandler
classname="com.ibm.websphere.wsssecurity.callbackhandler.SAMLGenerate
CallbackHandler">
          <properties value="Bearer"
name="confirmationMethod"/>
          <properties
value="http://docs.oasis-open.org/ws-sx/ws-trust/200512/Bearer"
name="keyType"/>
        </callbackHandler>
      </tokenGenerator>
    </securityOutboundBindingConfig>
    <securityInboundBindingConfig/>
  </securityBinding>
</securityBindings>
```

We have seen how the service provider was configured to consume a SAML V1.1 Bearer Token and how the service client was configured to generate a SAML V1.1 Bearer Token.

We now associate these bindings to the corresponding policy set on the service:

1. In the Service view, right-click **RAD8TopDownBankWS**. Complete these steps:
 - a. Select **Manage Policy Set Attachments** → **Server Side**.
 - b. Select **Add**.

2. You see the dialog window that is shown in Figure 14-72. Complete these steps:
 - a. For Policy set, select **SAML Bearer WSHTTPS default**.
 - b. For Binding, select **SAMLEnabledProviderBinding**.

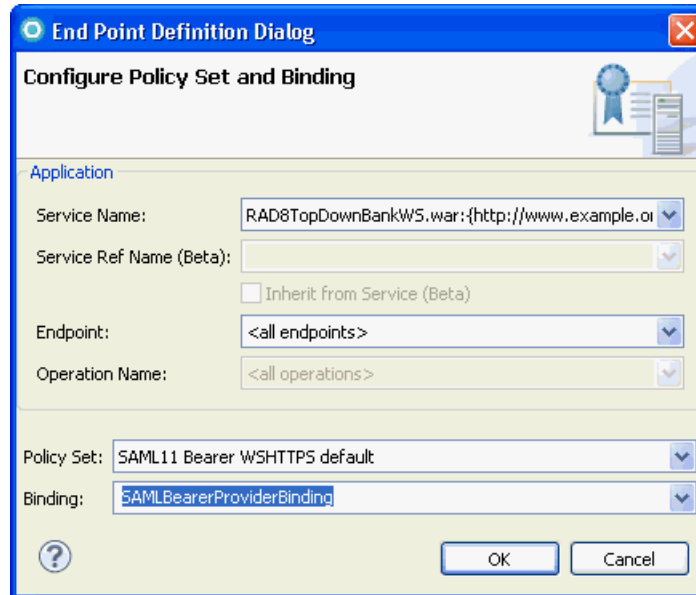


Figure 14-72 Configure Policy Set and Binding (SAMLEnabledProviderBinding)

3. Figure 14-73 on page 827 shows the resulting endpoint policy set and binding.

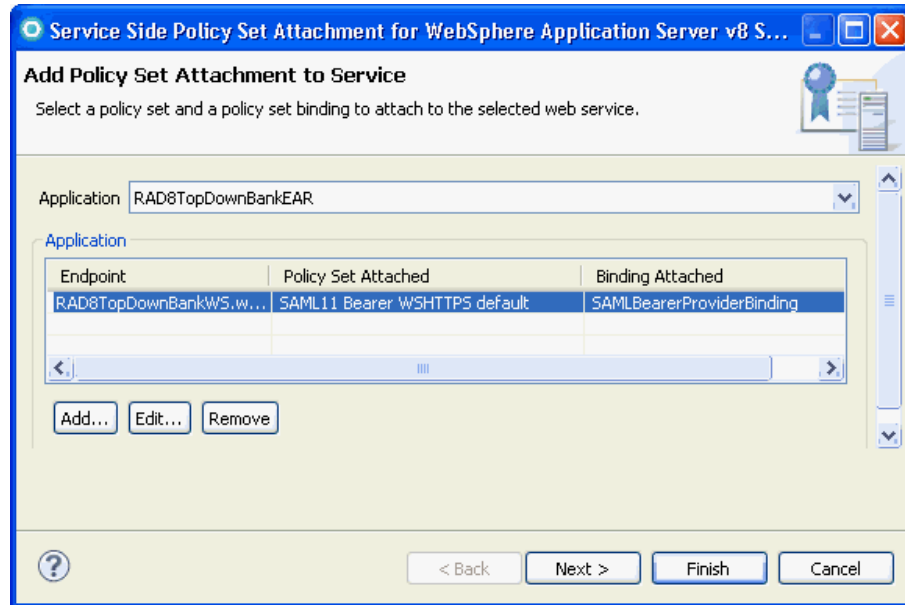


Figure 14-73 Add Policy Set Attachment to Service window

4. Select **Next**.
5. Select **Configure**.
6. Select **Share Policy Information via WSDL**.
7. Select **OK**, click **Ignore**, and click **Finish**.

We now configure the policy set and binding on the client:

1. In the Services view, right-click **RAD8TopDownBankWSClient**. Complete these steps:
 - a. Select **Manage Policy Set Attachments**.
 - b. Select **Add**.

2. You see the dialog window that is shown in Figure 14-74. Complete these steps:
 - a. For Policy set, select **SAML11 Bearer WSHTTPS default**.
 - b. For Binding, select **SAMLEBearerClientBinding**.

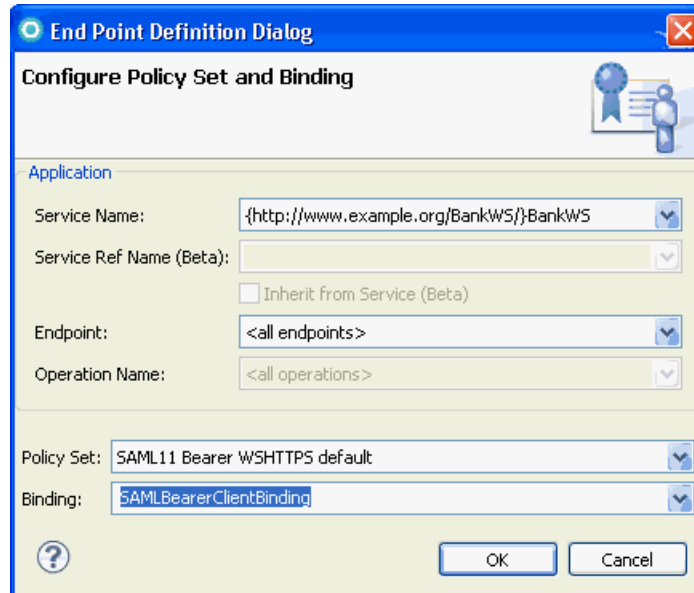


Figure 14-74 Configure Policy Set and Binding (SAMLEBearerClientBinding)

3. Select **OK**.
4. The Add Policy Set Attachment to Web Service Client window opens, as shown in Figure 14-75 on page 829. Select **Finish**.

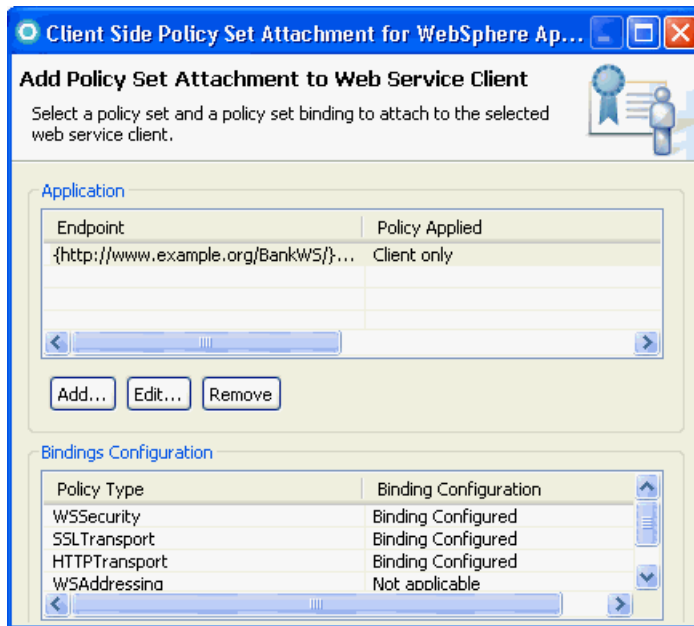


Figure 14-75 Add Policy Set Attachment to Web Service Client

We have completed the configuration of the policy sets and bindings on the client.

14.20.5 SAML Bearer sample: Programmatic token generation

In order for the client to generate the SAML Bearer Token programmatically, add the Java file `SAMLBearerTokenSetup.java`, as shown in Example 14-48, to the `src` folder of the client project `RAD8TopDownBankWSCClient`, in the package `org.example.bankws.saml`.

Example 14-48 SAMLBearerTokenSetup.java

```
package org.example.bankws.saml;

import com.ibm.websphere.wssecurity.wssapi.token.SAMLToken;
import com.ibm.websphere.wssecurity.wssapi.token.SAMLTokenFactory;
import com.ibm.wsspi.wssecurity.saml.config.CredentialConfig;
import com.ibm.wsspi.wssecurity.saml.config.ProviderConfig;
import com.ibm.wsspi.wssecurity.saml.config.RequesterConfig;

public class SAMLBearerTokenSetup {
```

```

    /**
     * This method generates an instance of a version 1.1 SAML Bearer
token
    */
    public static SAMLToken generateSAMLToken() {
        try {

            // Create a SAMLTokenFactory instance for a version 1.1 SAML
token
            SAMLTokenFactory samlFactory =
SAMLTokenFactory.getInstance(SAMLTokenFactory.WssSam1V11Token11);

            // OR: Create a SAMLTokenFactory instance for a version 2.0
SAML token
            //SAMLTokenFactory samlFactory =
SAMLTokenFactory.getInstance(SAMLTokenFactory.WssSam1V11Token20);

            // Create the RequesterConfig instance for the bearer token
RequesterConfig reqData =
samlFactory.newBearerTokenGenerateConfig();
            ProviderConfig samlIssuerCfg =
samlFactory.newDefaultProviderConfig(null);

            CredentialConfig cred = samlFactory.newCredentialConfig();

            // Create the SAML Token using the SAML Token Factory
SAMLToken samlToken = samlFactory.newSAMLToken(cred, reqData,
samlIssuerCfg);
            return samlToken;
        }
        catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}

```

Edit the file `BankWSSoapProxy.java` by adding the method `generateAndAttachSAMLBearerToken()`, as shown in Example 14-49, and call this method from both constructors.

Example 14-49 Generating and adding the SAML Token in `BankWSSoapProxy.java`

```

public void generateAndAttachSAMLBearerToken(){
    // Generate a version 1.1 SAML Bearer Token (self issuance)

```

```

        SAMLToken _samlToken = SAMLBearerTokenSetup.generateSAMLToken();

        // attaching the SAML Bearer Token to the Request Context.

        ((BindingProvider)_getDescriptor().getProxy()).getRequestContext().put(
        SamlConstants.SAMLTOKEN_IN_MESSAGECONTEXT, _samlToken);
        System.out.println("$$ Generated a SAML Token $$");
        System.out.println("SAML Token id is : " +
        _samlToken.getSamlID());
        System.out.println("Attached to the Request Context as property "
        + SamlConstants.SAMLTOKEN_IN_MESSAGECONTEXT);
    }
    public BankWSSOAPProxy() {
        _descriptor = new Descriptor();
        _descriptor.setMTOMEnabled(true);
        generateAndAttachSAMLBearerToken();
    }

    public BankWSSOAPProxy(URL wsdlLocation, QName serviceName) {
        _descriptor = new Descriptor(wsdlLocation, serviceName);
        _descriptor.setMTOMEnabled(true);
        generateAndAttachSAMLBearerToken();
    }
}

```

This point terminates the setup of the sample.

14.20.6 SAML Bearer sample: Testing

Perform these steps to test the sample:

1. Add both **RAD8TopDownBankWSCClientEAR** and **RAD8TopDownBankWSEAR** to the server.
2. In the Enterprise Explorer view, right-click **RAD8TopDownBankWSCClient\WebContent\sampleBankWSSOAPProxy\TestClient.jsp**.
3. Select **Run as** → **Run on Server**.
4. In the bottom pane of the Test Client that is opened in the browser, change the Endpoint so that it uses https. If the initial Endpoint was `http://localhost:9080/RAD8TopDownBankWS/BankWS`, it becomes `https://localhost:9443/RAD8TopDownBankWS/BankWS`. (If you have generated additional profiles with the recommended ports, both port numbers are typically increased by the same number of units.)
5. Select **Update**.

6. Select the method **getAccount**.
7. Enter any number in the accountId field.
8. Select **Invoke**.
9. You see the following results:

```
returnp:
    id: <accountId>
    balance: 1000
```

In the console, you see output indicating that the client generated the SAML Bearer Token (Example 14-50).

Example 14-50 Console showing the creation of the SAML Token

```
00000023 servlet      I com.ibm.ws.webcontainer.servlet.ServletWrapper
init SRVE0242I: [RAD8TopDownBankWSCClientEAR] [/RAD8TopDownBankWSCClient]
[/sampleBankWSSOAPProxy/Input.jsp]: Initialization successful.
00000024 SystemOut    0 Retrieving document at
'file:/C:/workspaces/WebServices/RAD8TopDownBankWSCClient/WebContent/WEB
-INF/wsd1/'.
00000024 SystemOut    0 Retrieving schema at 'BankWS_schema1.xsd',
relative to
'file:/C:/workspaces/WebServices/RAD8TopDownBankWSCClient/WebContent/WEB
-INF/wsd1/'.
00000024 SystemOut    0 $$ Generated a SAML Token  $$
00000024 SystemOut    0 SAML Token id is :
_ECAE899D4F56A343AC1288118909489
00000024 SystemOut    0 Attached to the Request Context as property
com.ibm.wsspi.wssecurity.saml.put.SamlToken
```

If you want to see the actual SOAP message, you cannot use the TCP/IP Monitor, because the message is transmitted using HTTPS. The Generic Service Client supports HTTPS, but it does not use the modified proxy client code to generate the SAML Bearer Token. You can, however, configure tracing in WebSphere Application Server that allows you to see how the server interprets the message:

1. Right-click the server in the Server view.
2. Select **Administration** → **Run Administrative Console**.
3. Select **Troubleshooting** → **Logs and Trace**.
4. Select **server1**.
5. Select **Diagnostic Trace**.
6. Select **Change Log level details**.

7. Right-click **com.ibm.ws.wssecurity.saml**.
8. Select **All Messages and traces**.
9. Select **OK**, which results in the following trace string:

```
*=info: com.ibm.ws.wssecurity.saml.*=all.
```
10. Select **Save**.
11. Log out of the administrative console.
12. Restart WebSphere Application Server for the changes to take effect.
13. Perform the same test as described previously.
14. Open the trace file, which, by default, is located in

```
<WAS_HOME>\profiles\<profile_name>\logs\server1\trace.log.
```
15. You see entries, as shown in Example 14-51. These entries were manually formatted.

Example 14-51 Trace including com.ibm.ws.wssecurity.saml.=all*

```
00000017 EnvelopedSign 3 ResourceShower logs
verify-#_DD551E9C7189EE6A931288120072735:
00000017 EnvelopedSign 3
<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
AssertionID="_DD551E9C7189EE6A931288120072735"
IssueInstant="2010-10-26T19:07:52.734Z"
Issuer="WebSphere" MajorVersion="1" MinorVersion="1">
  <saml:Conditions NotBefore="2010-10-26T19:07:52.750Z"
    NotOnOrAfter="2010-10-26T20:07:52.750Z">
  </saml:Conditions>
  <saml:AttributeStatement>
    <saml:Subject>
      <saml:NameIdentifier>
      </saml:NameIdentifier>
      <saml:SubjectConfirmation>
        <saml:ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:bearer
        </saml:ConfirmationMethod>
      </saml:SubjectConfirmation>
    </saml:Subject>
  </saml:AttributeStatement>
</saml:Assertion>
00000017 EnvelopedSign 3 ResourceShower logs verify-SignedInfo:
00000017 EnvelopedSign 3
<ds:SignedInfo xmlns:ds="http://www.w3.org/2000/09/xmlsig#">
  <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc14n#">
  </ds:CanonicalizationMethod>
```

```
<ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
</ds:SignatureMethod>
<ds:Reference URI="#_DD551E9C7189EE6A931288120072735">
<ds:Transforms>
<ds:Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature">
</ds:Transform>
<ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
</ds:Transform>
</ds:Transforms>
<ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
</ds:DigestMethod>
<ds:DigestValue>crkHuGw2LI4ZXeniAdyh9ggJ5sA=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
```

You can get the complete trace log at this location:

C:\7835codesolution\webservices\SAMLTrace.log

You can get the SAML Secured project in this file:


C:\7835codesolution\webservices\RAD8TopDownWebServiceSAML.zip

14.21 More information

For more information about web services, see the following resources:

- ▶ For information about JAX-WS, Reliable Messaging, Secure Conversation, policy sets, and RSP profiles, see these publications:
 - *Web Services Feature Pack for WebSphere Application Server V6.1*, SG24-7618
 - *IBM WebSphere Application Server V7.0 Web Services Guide*, SG24-7758
- ▶ For JAX-RPC web services tools that ship with Rational Application Developer V7.0, see the *Rational Application Developer V7 Programming Guide*, SG24-7501.
- ▶ IBM developerWorks section about SOA and web services
<http://www.ibm.com/developerworks/webservices>

- ▶ List of current and emerging web services standards on developerWorks (under **SOA and Web services** → **Standards**)
<http://www.ibm.com/developerworks/webservices/standards/>
- ▶ The JAX-WS specification
<http://jcp.org/aboutJava/communityprocess/pfd/jsr224/index.html>
- ▶ The JAXB specification
<http://jcp.org/en/jsr/detail?id=222>
- ▶ The MTOM specification
<http://www.w3.org/TR/soap12-mtom/>
- ▶ JAX-WS annotations:
 - <https://jax-ws.dev.java.net/jax-ws-ea3/docs/annotations.html>
 - http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.wsfepl.multiplatform.doc/info/ae/ae/rwbs_jaxwsannotations.html
- ▶ The WS-Policy specification
<http://www.w3.org/Submission/WS-Policy/>
- ▶ The WS-MetadataExchange specification
<http://www.ibm.com/developerworks/webservices/library/specification/ws-mex/>
- ▶ JAX-RS resources and examples:
 - <http://www.ibm.com/developerworks/web/library/wa-apachewink1/>
 - <http://www.ibm.com/developerworks/webservices/library/ws-restful/>
 - <http://www.ibm.com/developerworks/web/library/wa-datawebapp/>
- ▶ SAML resources and examples:
 - http://www.ibm.com/developerworks/websphere/techjournal/1004_chao/1004_chao.html
 - <https://www.ibm.com/developerworks/wikis/download/attachments/116424904/Introduction+to+SAML+and+support+in+7.0.0.7.pdf>
 - http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/cwbs_samloverview.html



Developing Open Services Gateway initiative (OSGi) applications

This chapter introduces the new feature, Open Services Gateway initiative (OSGi), which is built into Rational Application Developer. OSGi is a module system that is compatible with Java-based systems and implements a dynamic component model. Enterprise systems can use OSGi to improve the maintainability of runtime infrastructures. OSGi applications, in the form of bundles, can be remotely installed, started, stopped, updated, and uninstalled without requiring a restart.

In this chapter, we discuss the following topics:

- ▶ OSGi overview
- ▶ Introduction to OSGi bundles
- ▶ Installation of the Feature Pack for OSGi
- ▶ Tools for OSGi application development
- ▶ Creating OSGi bundle projects
- ▶ Developing OSGi applications

15.1 OSGi overview

In this section, we examine the concepts of OSGi, we learn about its specifications, architecture, features, and benefits.

What is OSGi

OSGi is a dynamic module system service platform for Java and offers an applications framework for developing, assembling, and deploying applications using Java Platform, Enterprise Edition (Java EE) and OSGi technologies. The applications developed using this framework exhibit modularity with loose coupling within modules. They are dynamic and can collaborate with or depend on other components.

Why OSGi

The modularity of OSGi provides a good mechanism to address the issues that are faced by the Java EE applications:

- ▶ OSGi applications are easily portable and adaptable.
- ▶ OSGi applications can access external bundle repositories.
- ▶ OSGi has a built-in bundle repository that can host common and versioned bundles that can be shared across multiple applications instead of each application having its own library.
- ▶ This framework implements service-oriented architecture (SOA) at the module level.
- ▶ This framework also integrates with the Java 2 Platform, Enterprise Edition (J2EE) programming model and provides isolation for enterprise applications that are composed of multiple versioned bundles with dynamic life cycles.

Bundles: OSGi solves the modularity and versioning problems through the concept of a bundle. A *bundle* at the simplest level is a standard, traditional JAR file with additional metadata in the JAR manifest file. So, enabling an existing library for OSGi is a non-invasive change. The extra headers in the JAR manifest are ignored in non-OSGi environments. In an OSGi environment however, the extra headers allow a bundle to be more than a unit of packaging. The bundle now defines a unit of modularity.

Specifications

The OSGi specifications are defined and maintained by the OSGi Alliance, which is an open standards organization. The *OSGi Service Platform Specifications V4.2* bring the benefits of OSGi to the Java EE application developer. OSGi Version 4.2 defines the Blueprint component model. This model defines how you

can exploit OSGi modularity in your applications, in particular, helping with third-party library integration and versioning.

Because OSGi is an open, specification-based technology, many implementations exist. The two most prominent implementations are the Apache Felix project (<http://felix.apache.org>) and the Eclipse Equinox project (<http://eclipse.org/equinox>), which is used in WebSphere Application Server.

For more information: For more information about the OSGi applications framework in WebSphere Application Server, refer to the Feature Pack for OSGi Applications and Java Persistence API (JPA) 2.0 documentation in the WebSphere Application Server library:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.jpafep.multiplatform.doc/info/ae/ae/welcome_fep_jpa.html

For more information about the OSGi specification, refer to the OSGi Alliance Specifications (<http://www.osgi.org>).

Enterprise OSGi

The *OSGi Service Platform Enterprise Specification V4.2* focuses mainly on the Java enterprise applications. It includes the *Blueprint Container Specification*, which defines a component model for OSGi based on the Spring framework in which an OSGi bundle is augmented by an XML configuration file known as a *module blueprint*. The module blueprint wires together separate components in a bundle and configures the dependency injection framework required for inversion of control.

15.1.1 OSGi architecture

The core OSGi service platform has a layered structure. OSGi defines the idea of a bundle as a group of small modules. The service platform architecture is based on the modules that are deployed. The OSGi architecture has four layers, as shown in Figure 15-1 on page 840.

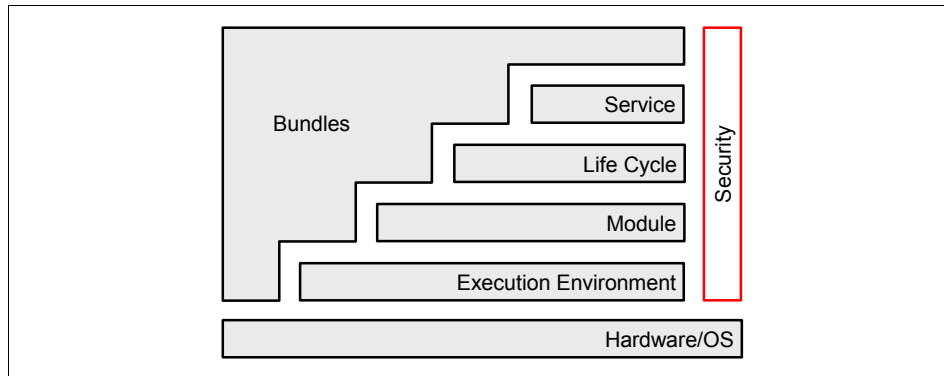


Figure 15-1 Layers of the OSGi framework

We describe the four layers:

► **Execution Environment layer**

This layer specifies the Java environment (Java EE or Java SE) in which the bundle runs. For applications running in WebSphere Application Server, there is no need to specify the environment.

► **Module layer**

This layer processes the metadata present in the bundle's manifest file. The OSGi framework determines the dependencies based on the manifest and calculates independent, required class paths. The class paths address the issues faced by a plain Java class loading and make sure that the following conditions are met:

- Only packages explicitly exported by a particular bundle, through the metadata, are visible to other bundles for import.
- Each package can be resolved to specific versions.
- Multiple versions of a package can be available concurrently to separate clients.

This layer is mainly focused on eliminating problems during class loading and preventing "Class Not Found" exceptions at run time. When a bundle is deployed into the framework, it tries to resolve all the dependencies before it starts the application. In this layer, the bundles are dynamic.

► **Life Cycle layer**

The Life Cycle layer controls installing, uninstalling, starting, stopping, updating, and monitoring the bundles. Each bundle must implement the application programming interface (API) in order to define its behavior at certain stages of deployment.

► **Service Registry layer**

This is the highest layer and supports the SOA. The services are published to the Service Registry by the bundles, and the service registry acts as a medium for collaboration among bundles. An OSGi service is a Plain Old Java Object (POJO) published to the service registry as implementing one or more Java interfaces. The Service Registry layer searches other bundles and provides notification when the registration of a bound bundle changes.

15.2 Introduction to OSGi bundles

Two problems exist that are associated with typical Java applications:

- No concept of modularity exists between the application level and class level.
- No versioning exists and no capability is available to handle multiple versions.

For more information about these problems, see *Getting Started with the Feature Pack for OSGi Applications and JPA 2.0*, SG24-7911.

OSGi bundles offer a solution. A bundle is a Java archive file (.jar) that consists of code, resources, and a manifest file that defines the content and visibility of the bundle. It is a complete, modular unit of deployment.

15.2.1 OSGi classloading

A bundle defines the packages it depends on as well as the packages it provides, tagged with versions in both cases. The OSGi run time uses this information to wire a bundle so that it has access to only the packages declared as dependencies and so that other bundles can only see those packages that are explicitly exported. All other packages are hidden inside the bundle.

Instead of a hierarchical classloading structure that is traditional in Java, particularly JEE, OSGi has a network classloading structure (Figure 15-2 on page 842). In the hierarchical model, classes are searched for from the bottom up, through various layers of application modules, runtime libraries, system classes, and extension libraries. All classes are visible in the same layer and to any layers beneath. So, a new version of a library in the extension libraries affects the entire stack, from the extension libraries down to the application modules.

In contrast, an OSGi bundle resolves dependencies in a network-like manner. Bundles declare the packages and the version that they can provide. Packages that are included in the bundle, but not exported, are not visible to other bundles. They also declare the packages and the version, or range of versions on which

they depend. Packages that are not declared as a dependency are not available to the bundle. The OSGi framework resolves and manages the dependencies between bundles in such a manner that a bundle cannot be started until its dependencies are resolved.

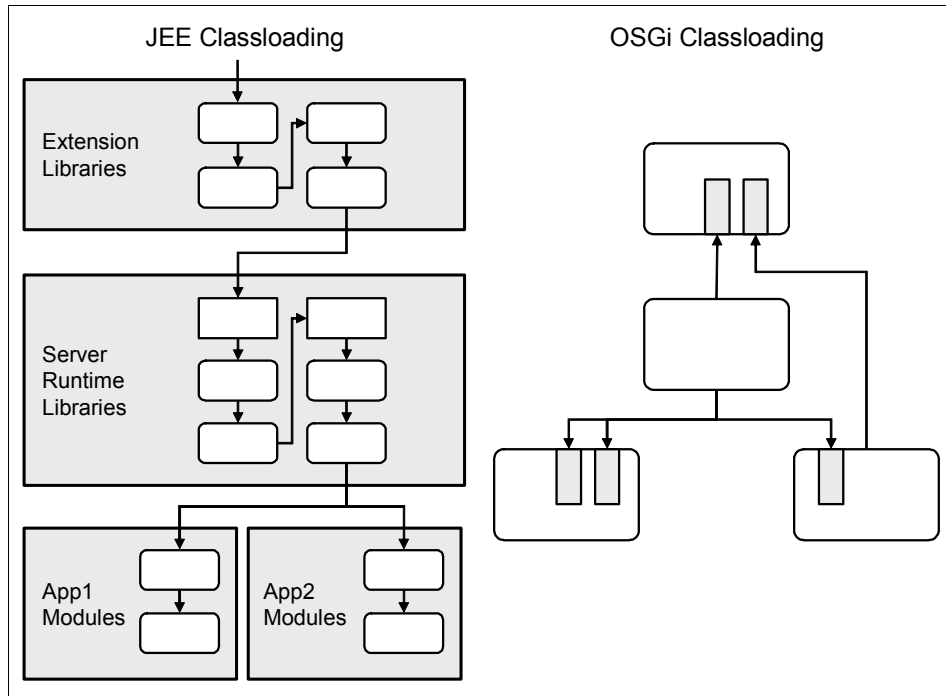


Figure 15-2 JEE hierarchy classloading versus OSGi networked classloading

Therefore, with the addition of dependency metadata, which also allows package dependencies to be marked as optional, it is possible to check if a bundle will work at run time or if there are missing dependencies. So, with correctly written metadata, a bundle will never throw a `NoClassDefFoundError` at run time.

15.2.2 Bundle manifest file

The *bundle manifest file* contains the metadata that allows the OSGi framework to resolve the dependencies of the bundle. The code example that is shown in Example 15-1 is from the sample bundle manifest file.

Example 15-1 Sample bundle manifest file

```
Manifest-Version: 1
Bundle-ManifestVersion: 2
Bundle-Name: MyService bundle
```

```
Bundle-SymbolicName: my.very.useful.library
Bundle-Version: 42.0.0
Bundle-Activator: my.very.useful.library.stringops.Activator
Import-Package: org.osgi.framework;version="[1.5.0,2.0.0)"
Export-Package:
my.very.useful.library.stringops;version=23.2.1,my.very.useful.library.
interop;version=5.0.0
```

Note the following entries in the bundle manifest file:

- ▶ *Bundle-Manifest Version*: The version is set to 2 to indicate that the bundle is written to revision 1.3 or later of the OSGi specification. A version of 1 is used for older versions.
- ▶ *Bundle-Name*: A human readable, display name that is used to identify the bundle.
- ▶ *Bundle-SymbolicName*: A unique identifier for the bundle. Usually separate from the *Bundle-Name*. It is combined with the *Bundle-Version* to uniquely identify the bundle.
- ▶ *Bundle-Version*: Indicates the version of the bundle.
- ▶ *Bundle-Activator*: Indicates the class that processes notifications received from the framework during bundle life-cycle changes.
- ▶ *Import-Package*: Defines packages required by a bundle. A bundle does not need to import `java.*` packages, or packages that are part of the Java Development Kit (JDK). Any other packages that are needed by the bundle, which are not defined within the bundle, need to be explicitly imported using this header. Every package import carries a *version range* that defines the accepted versions of the package. This version range is entirely independent of the bundle version.

Example 15-1 on page 842 only includes one *Import-Package* declaration, indicating that the bundle only needs the `org.osgi.framework` package, which is in addition to its own classes and the `java.*` classes that are available by default. The version tag (`version="[1.5.0,2.0.0)"`) requests a version between 1.5.0 (inclusive, denoted by a square bracket) and 2.0.0 (exclusive, denoted by a round bracket). If no version range is specified, *which is not recommended*, any version of the package will satisfy the dependency. The version tag is a version-range. So, a `version="1.0.0"` is a version range of versions 1.0.0 and later, not that exact version. The correct syntax for an exact version is `version="[1.0.0,1.0.0]"`.

- ▶ *Export-Package*: Defines the packages provided by the bundle that the bundle exposes to other bundles. Only the specified packages can be used by other bundles. Every exported package carries a version, which defaults to "0.0.0" if unspecified.

Importing and exporting the same package

Because of the complexities of the OSGi class loader, special consideration must be given to importing an exported package. When a package is exported, it is possible for the package to be used by a separate bundle before the exporting bundle is even started. This situation can be a problem when Singletons and static fields are used, because an imported class is loaded by a separate class loader than the class loader used by the bundle, which creates separate instances of the service object.

For example, suppose our bundle uses and exports a service object that is responsible for generating sequential order numbers. Also, suppose that three other bundles use that service object for generating order numbers. Because those three other bundles imported the package, they use the “framework class loader” and share the same instance of the service object. If our bundle also imports the package, it also uses the same instance. But, if our bundle does not import the package, the “bundle class loader” will instantiate a new instance of the service object and possibly produce a duplicate list of order numbers.

Leading practices dictate that you typically must import any packages that you export to reduce the number of copies of that package in memory and to ensure that the object instances come from the same class loader.

15.2.3 Life cycle of a bundle

The OSGi framework is responsible for the life-cycle management of a bundle. After you install and start a bundle, it goes through various states. The life cycle of a bundle is much closer to the life cycle of a JEE application than that of a plain JAR file. The jar file is simply loaded at run time and unloaded when the application terminates. The life cycle of the bundle is more complex. Figure 15-3 on page 845 shows the life-cycle flow.

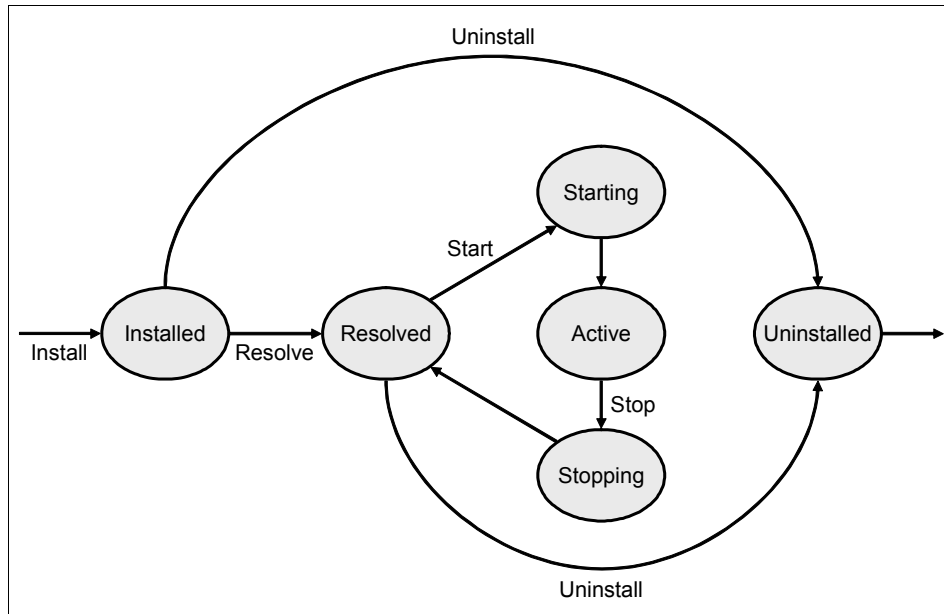


Figure 15-3 Life cycle of a bundle

Note the following life cycle states and events:

► **INSTALLED STATE**

The bundle has been installed, but all of the dependencies of the bundle have not been met. The bundle requires packages that have not been exported by any currently installed bundle.

► **INSTALLED EVENT**

Sent after the bundle has entered the Installed state.

► **RESOLVED STATE**

The bundle is installed and the dependencies of the bundle have been met, but it is not running. If a bundle is started and all of the dependencies of the bundle are met, the bundle leaves this state.

► **RESOLVED EVENT**

Sent after the bundle has entered the Resolved state.

► **STARTING EVENT**

Sent before the bundle is started.

► **STARTING STATE**

A temporary state that the bundle goes through while the bundle is starting.

- ▶ **STARTED EVENT**
Sent after the bundle has been started to indicate that the bundle is now active.
- ▶ **ACTIVE STATE**
This state indicates that the bundle is running.
- ▶ **STOPPING EVENT**
Sent to indicate that the bundle is about to be stopped.
- ▶ **STOPPING STATE**
A temporary state that the bundle goes through while the bundle is stopping.
- ▶ **UNINSTALLED EVENT**
Sent to indicate that the bundle has been removed from the framework.
- ▶ **UNRESOLVED EVENT**
Sent when the framework discovers an unresolved dependency caused by a bundle being uninstalled. The state of the bundle returns to Installed until the dependency can be resolved.
- ▶ **UPDATED EVENT**
Sent when a bundle has been updated.

15.2.4 Blueprint Container Specification

The *OSGi Blueprint Container Specification (Blueprint)* defines a dependency injection framework for OSGi derived from the Spring Dynamic Modules project. The specification defines a component model for OSGi based on the core Spring framework in which an OSGi bundle is augmented by an XML module blueprint. A *module blueprint* is a configuration file that describes how fine-grained components are wired together within the bundle.

The Blueprint XML files describe the components of the applications, their dependencies, and their life cycles. Blueprint XML files are based on the popular Spring framework. The concepts and syntax are familiar to a wide audience of developers.

Blueprint Containers provide *service damping* as the default mode of operation. When using Blueprint Containers, the developer is shielded from several of the most complex aspects of service dynamics, which are handled by the Blueprint Containers. Applications simply wait for a service to become available instead of failing immediately if the requested service is temporarily unavailable.

Bundles can exploit the benefits of the Blueprint Container Specification by including one or more Blueprint descriptors. These descriptors can either be located inside the `OSGI-INF/blueprint` directory or be specified via the `Bundle-Blueprint` manifest header. A bundle that includes Blueprint descriptors is often referred to as a *Blueprint bundle*. Example 15-2 shows a sample Blueprint descriptor that demonstrates the Spring-like syntax. We use this example to highlight the key concepts in the Blueprint Container Specification.

Example 15-2 Blueprint sample XML file

```
1 <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
2 <bean id="myBizObject" class="my.sample.impl.BizObject">
3 <property name="helper" ref="myHelper" />
4 <property name="service" ref="serv" />
5 </bean>
6
7<bean id="myHelper" class="my.sample.util.Helper">
8 <argument ref="blueprintBundleContext" />
9 </bean>
10
11 <service interface="my.sample.BizInterface" ref="myBizObject">
12 <service-properties>
13 <entry key="service.level" value="gold" />
14 </service-properties>
15 </service>
16
17 <reference id="serv" interface="my.services.SampleService"
18 availability="mandatory"
19 filter="(transactional=true)">
20 </reference>
21 </blueprint>
```

At the simplest level, Blueprints can create instances of classes that are already defined inside the bundle (or imported by the bundle), as shown in line 7 of Example 15-2. That capability alone is not useful without the ability to also inject dependencies either per constructor (line 8) or via setter methods (line 3 and 4). Example 15-3 and Example 15-4 on page 848 show the corresponding classes.

Example 15-3 Helper class

```
public class Helper {
public Helper(BundleContext context) { ... }
...}

```

Example 15-4 BizObject.java

```
public class BizObject {  
    public void setHelper(Helper h) { ... }  
    public void setService(SampleService ss) { ... }  
    ...}  
}
```

The injected elements can either explicitly give a primitive value or reference another Blueprint manager (for example, any of the named top-level elements). Blueprint also defines default managers, such as the `blueprintBundleContext` manager (line 8), which essentially is the `BundleContext` object of the bundle containing this blueprint descriptor.

Finally, the sample shows how to integrate with other modules via the service registry. Lines 17-20 in the Blueprint that is shown in Example 15-3 on page 847 declare a dependency on a service with the `SampleService` interface, which, in turn, is then injected normally into the `BizObject` in line 4. Line 19 highlights Blueprint's support for service filters. Line 18 in Example 15-4 is perhaps the most interesting line.

Blueprint service references can be mandatory or optional. If a reference is mandatory, the Blueprint extender does not create any beans in the Blueprint module unless that reference is satisfied by a service. Therefore, the Blueprint extender does not instantiate beans or publish services for that Blueprint module until all mandatory references are satisfied. If the mandatory references do not become satisfied within a given interval (by default, five minutes), Blueprint does give up and destroy the Blueprint module.

When the Blueprint container is already up and running, a service can still go away. This situation does not terminate the Blueprint container. Instead, when a call is made to the service that has gone away, Blueprint waits for a set amount of time (by default, five minutes) for a new service that satisfies the reference to appear before throwing an exception.

This behavior, called *service damping*, ensures that Blueprint beans are unaffected by a temporary absence of a mandatory service. This behavior also means that the service to which a reference is bound can change over time without bringing down the running blueprint. This capability is a powerful facility.

Finally, lines 11-15 in Example 15-3 on page 847 show how easy it is to publish a service via Blueprint and exemplifies Blueprint's support for custom service properties.

The Blueprint specification defines much more than what is covered in the preceding example. In addition to the support for the primitive values shown here,

Blueprint allows the creation of an arbitrary collection of primitives and beans as well as customized conversion between literal values and required class instances. Furthermore, Blueprint bundles can hook into the service dynamics via reference and service registration listeners.

For more information, consult the Blueprint Service specification:

<http://www.osgi.org/Specifications/HomePage>

15.2.5 Types of bundle archives

This section introduces two types of bundle archives.

Enterprise bundle archives (EBA)

An EBA is an archive of more than one bundle that is deployed as a single OSGi application. They are isolated from the other bundles and services of other OSGi applications and run under their own instance of the framework. Bundles that belong to an OSGi application can reference other bundles that are in the shared bundle repository (not included within the application) as long as these external bundles export packages that these bundles import. The archive contains this information:

- ▶ Archive content: A set of jars representing bundles within the EBA.
- ▶ Application manifest
- ▶ Deployment manifest (optional): a file that WebSphere Application Server generates when resolving an application. This file defines the exact packages and their versions to which the application is resolved after deployment. Rational Application Developer allows the user to view this file, and it even persists it within the application to guarantee that the application will get resolved to the exact same bundles in a separate environment.

Composite bundle archives (CBA)

The CBA is a group of bundles that are grouped together and that act as a single bundle as far as the user is concerned. CBAs can be deployed to the WebSphere internal repository, and they can be used by potentially multiple OSGi applications. When the run time resolves a package to a bundle within a CBA, it has the affinity to resolve the rest of the packages within the same CBA if at all possible. The user creates these archives when the run time is required to pick particular versions of packages that work together in a predictable way. This CBA has the `.cba` extension. The CBA contains this information:

- ▶ Composite bundle contents (a set of jars representing the contained bundles)
- ▶ Composite bundle manifest

We discuss EBAs and CBAs when we create the OSGi bundle projects later in this chapter.

15.2.6 Relationships among bundles, application archives, and composite archives

To satisfy the software principles of encapsulation and modularity, break your applications down into small, reusable components. Encapsulate these components in the OSGi bundle. OSGi bundles are complete, modular units of deployment. As you develop a number of bundles under your application suite, you can combine bundles that fall under a common business area into an EBA. The EBA greatly simplifies the deployment and maintenance of your application, because you can deploy the entire archive instead of individual bundles. Next as your application portfolio expands, you will find that you tend to reuse common libraries and functionality. For example, your organization might decide to standardize on a certain set of utility libraries, such as:

- ▶ Simple Logging Facade for Java (SLF4J) V1.6.1
- ▶ Apache Derby V10.6.10 or V10.6.2.1
- ▶ Apache FOP V1.0
- ▶ Apache Commons Codec V1.4

These libraries can be combined into bundles in a CBA, which eliminates the need for each application to maintain separate jar files and ensures consistency across the applications. Additionally, because the archive can support multiple versions of the same package, migration at the application level from one version to another is significantly simplified. You are no longer required to migrate all of the applications at the same time, which can be difficult to coordinate in times of tight budgets and short development cycles.

Another candidate for the CBA is a set of utility classes that span multiple applications. A good example is a set of financial utilities that compute sales tax in various localities around the world, compute amortization, compute loan-to-value on a mortgage, or compute the money factor on a lease. Each of these utilities is a bundle and can be combined into a CBA. Doing so eliminates the need for each application to develop similar utilities and guarantees that each application receives the same result for a given algorithm.

15.3 Installation of the Feature Pack for OSGi

Using Rational Application Developer, you can develop and deploy JEE and OSGi applications to an integrated WebSphere Application Server test environment. In Rational Application Developer, WebSphere Application Server

V8 Beta supports OSGi without the need for feature packs. To enable the OSGi development tools in a WebSphere Application Server V7 test environment, we have to install *IBM WebSphere Application Server Version 7.0 Feature Pack for OSGi Applications and Java Persistence API 2.0*.

For the installation steps, consult *Getting Started with the Feature Pack for OSGi Applications and JPA 2.0*, SG24-7911.

To check if your existing Rational Application Developer workspace has the OSGi development tools feature enabled, use the following steps:

1. In Rational Application Developer, go to **Help** → **About Rational Application Developer for WebSphere Software**.
2. Click **Installation Details** and select the **Installed Software** tab. You see Figure 15-4 on page 852. Scroll down the OSGi group to validate that the OSGi development tools are installed.

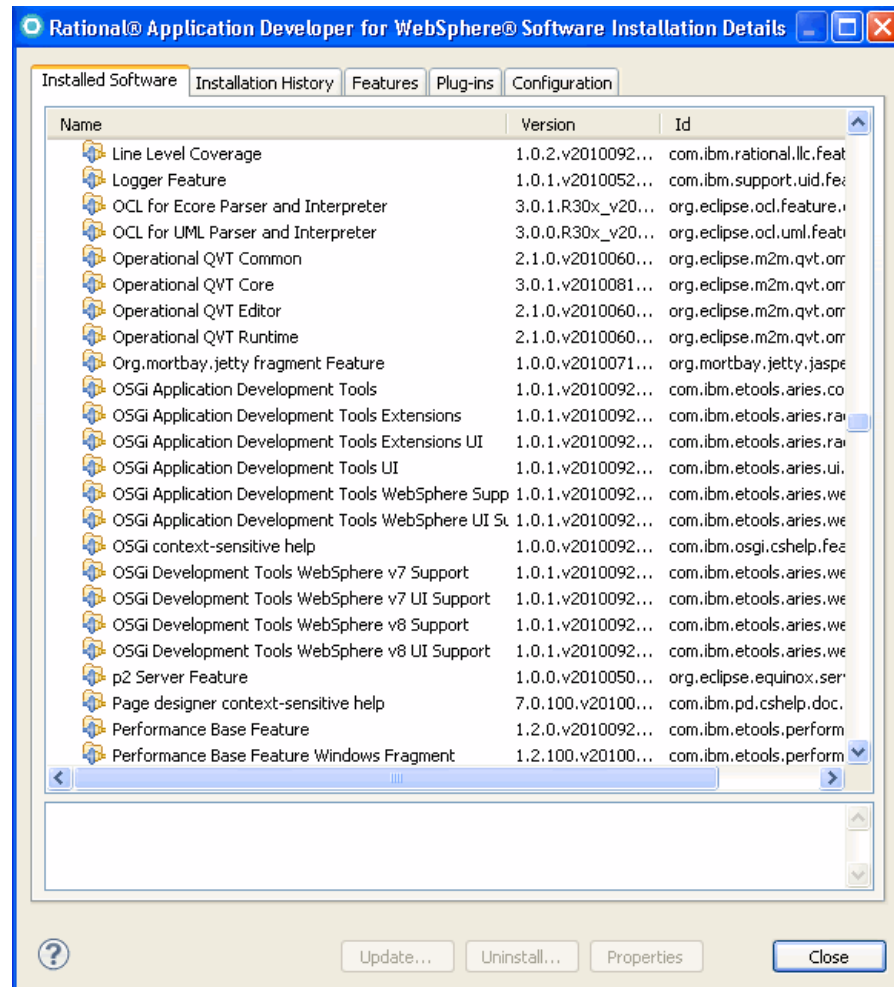


Figure 15-4 Installed OSGi features

15.4 Tools for OSGi application development

The Eclipse Plug-in Development tools provide partial support to develop OSGi bundles. However, IBM Rational Application Developer provides additional development tools, both licensed and at no charge, that help to create Enterprise OSGi applications.

The Rational Application Developer V8 development environment contains a selection of views, wizards, and editors that are customized to be the most useful for an OSGi developer.

The Rational Application Developer V8 development tools offer the following major features:

- ▶ Wizards to create OSGi bundles, applications, and composite bundles projects.
- ▶ A model for assembling bundles into an enterprise OSGi application that can be exported or deployed to WebSphere Application Server.
- ▶ A container for the OSGi blueprint component model enabling the standard dependency injection mechanism.
- ▶ Tools to convert existing Java, web JPA, and plug-in projects into an OSGi bundle.
- ▶ Form-based editors for the OSGi Bundle Manifest, Application Manifest, and Composite Bundle Manifest with content assist.
- ▶ Validation of the correctness of the OSGi bundle and application structure.
- ▶ The capability to publish OSGi enterprise applications to WebSphere Application Server.
- ▶ The import and export mechanism for OSGi bundles, applications, and composite bundles.
- ▶ OSGi Bundle Explorer visualizes your bundles and the dependencies between them.

No-charge tools

In addition to the licensed tooling in Rational Application Developer, a version with marginally reduced features is available at no charge at this website:

<http://www.ibm.com/developerworks/rational/downloads/10/rationaldevtoolsforosgiapplications.html>

The no-charge tools include support for creating OSGi bundles, applications, and Composite Bundles. They provide most of the features that are supported by Rational Application Developer. However, they lack added value features that are only available in Rational Application Developer:

- ▶ Publishing to WebSphere Application Server
- ▶ Form-based editor for the Blueprint files
- ▶ Graphical Bundle Explorer

- ▶ Extra support for refactoring bundles that understands the bundle entries in the application manifest, and class refactoring that understands class entries in the blueprint files that reference them.
- ▶ Extra validations and quick fixes that analyze the OSGi project's structure to ensure correctness.

Installing the no-charge Rational Development Tools for OSGi

Applications: To install the Rational Development Tools for OSGi Applications, you are required to download and install Eclipse software 3.6.1 and make sure to use a Java software development kit (SDK) equal to or greater than Version 5.

After installing the Eclipse software, choose **Help** → **Install New Software** and use the provided Rational Development Tools for OSGi Applications update site for Eclipse to install the tools:

<http://public.dhe.ibm.com/ibmdl/export/pub/software/rational/OSGi/AppTools>

After the tools are installed, you can create new OSGi-based projects. Additionally, the tools provide help about OSGi-related topics.

15.5 Creating OSGi bundle projects

This section describes the following topics:

- ▶ Creating OSGi bundle projects
- ▶ Creating an OSGi application project
- ▶ Creating a composite bundle project
- ▶ Working with the Composite Bundle Manifest
- ▶ Blueprint Container Specification

15.5.1 Creating OSGi bundle projects

To create an OSGi bundle project, follow this procedure:

1. Click **File** → **New** → **Other** → **OSGi** → **OSGi Bundle Project** to open the New OSGi Bundle Project wizard. Complete these tasks:
 - a. In the Project name field, enter **ITS0BankOSGi**.
 - b. In the Target Runtime drop-down list, select **WebSphere Application Server v8 Beta**.

- c. In the Configuration section, perform these steps:
 - i. Optional: Select **Add Web Support** to create a web-enabled OSGi bundle, which adds required support for dynamic web projects, web pages, servlets, and so forth.
 - ii. Optional: Select **Add persistence support** to include JPA support.
 - iii. For other capabilities, you can select **Custom** to add customized facets to the new bundle project.
- d. Select **Add bundle to application**, which is selected by default. It adds the bundle to an OSGi application. If the application does not exist, it will create one. For deploying an OSGi bundle, it is necessary to add it to the application.
- e. Click **Finish**, as shown in Figure 15-5 on page 856.

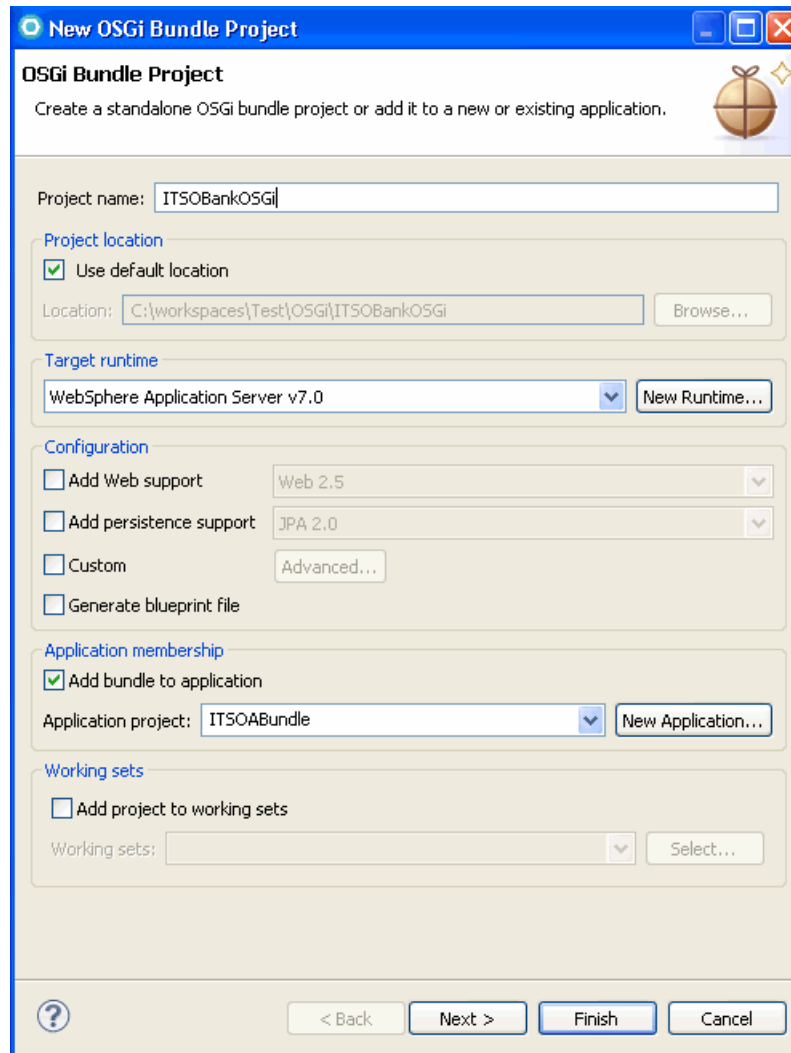


Figure 15-5 OSGi Bundle Project

15.5.2 Creating an OSGi application project

An *application project* is a group of bundles tied together to provide a modular application. The OSGi application is the enterprise deployment unit for OSGi bundles. When publishing an application, all bundles inside it get published together to the server. The OSGi application is a collection of bundles that make up the deployment unit. Applications get deployed to the server. Bundles within

one application can reference each other, but bundles that belong to another application can only reference each other via OSGi services.

Follow these steps to create an OSGi application bundle project:

1. Click **New** → **Other** → **OSGi Application Project** to open the New OSGi Application Project wizard. Complete these steps:
 - a. In the Project name field, enter `ITS0AApplication`.
 - b. In the Target runtime drop-down list, select **WebSphere Application Server v8 Beta** and click **Next**.
2. On the Contained OSGi Bundles page, you can add bundles to the application project. Complete these steps:
 - a. Select the **ITS0BankOSGi 1.0.0** project that you created previously and click **Finish**. The wizard creates the OSGi application that contains an application manifest file at `META-INF/APPLICATION.MF`.
3. Double-click the **APPLICATION.MF** file to open the OSGi Application Manifest editor. In the Contained Bundles section, you see the `ITS0BankOSGi 1.0.0` bundle that you added in the wizard. This section list the OSGi bundles and Plug-in Development Environment (PDE) plug-ins that are contained within the application. You can use the Add and Remove buttons to add or remove bundles from the application (Figure 15-6 on page 858).

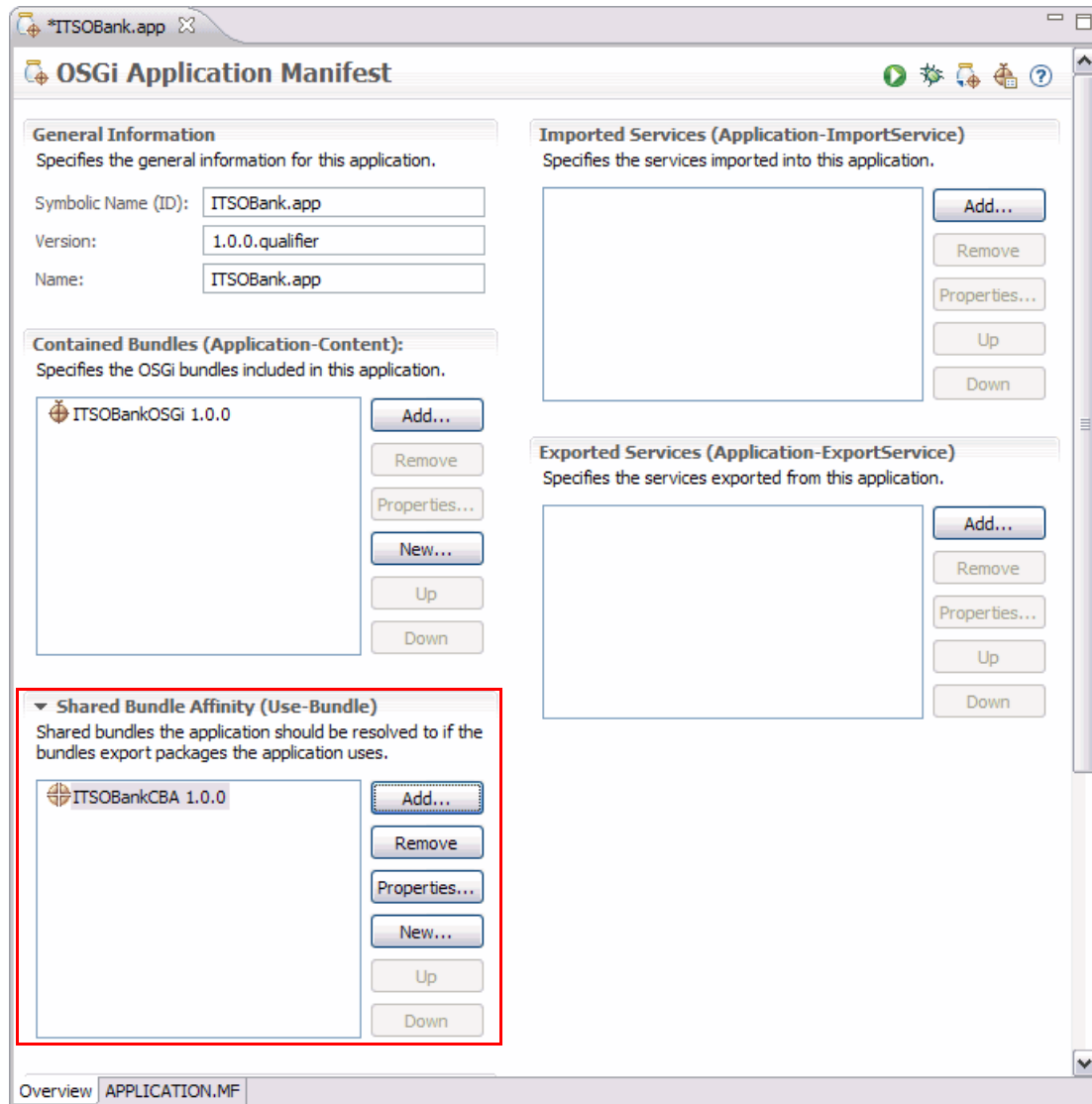


Figure 15-6 OSGi application bundle application manifest file

The Imported Services section lists OSGi Services that are defined outside your OSGi application, that the application will reference. The Exported Services section lists any OSGi Service that is defined by a bundle in this application that you want to make available or expose outside the OSGi application. The Dynamic Web Projects section allows users to use certain Java EE web projects as is without converting them to an OSGi bundle, which is useful for users when

they want to try their projects using OSGi applications before committing to OSGi.

The APPLICATION.MF tab shows the source view of the manifest.

15.5.3 Creating a composite bundle project

The CBA is a group of OSGi bundles that are combined to provide a consistent behavior to a set of applications. Use the OSGi Composite Bundle wizard to create a new composite bundle project:

1. Click **File** → **New** → **Other** → **OSGi** → **OSGi Composite Bundle Project** to open the New OSGi Composite Bundle Project wizard. Complete these steps:
 - a. In the Project name field, enter ITS0BankCBA.
 - b. In the Target runtime drop-down list, select **WebSphere Application Server v8.0 Beta**, as shown in Figure 15-7, and click **Next** to advance to the OSGi Bundles Selection Page.

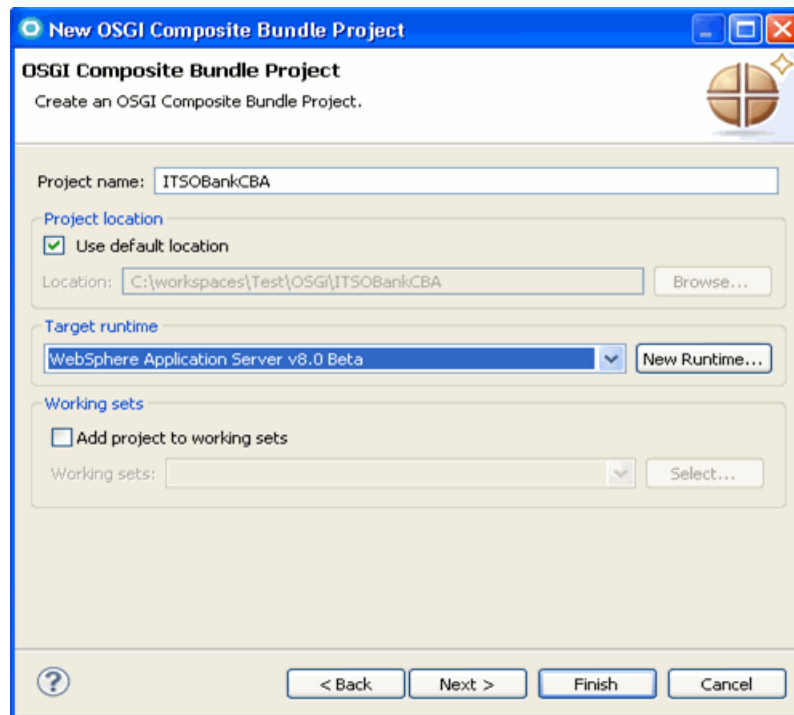


Figure 15-7 New OSGi Composite Bundle Project

2. On the OSGi Bundles Selection Page, select the bundles that you want to add to the Composite Bundle, as shown in Figure 15-8. For this example, select the **ITSOBankOSGi 1.0.0.qualifier** and **com.ibm.ws.jpa 7.0.0**. If these bundles are not visible, select **Show platform bundles**. Click **Finish**.

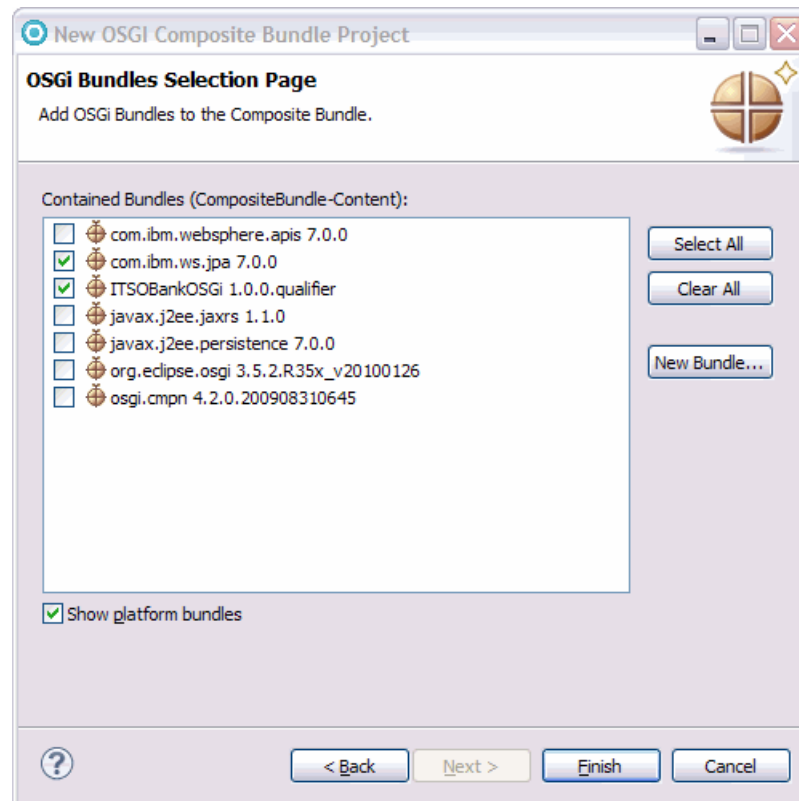


Figure 15-8 Composite Bundle wizard

15.5.4 Working with the Composite Bundle Manifest

After the composite bundle project is created, we can see a Composite Bundle Manifest created at META-INF/COMPOSITEBUNDLE.MF. Double-click the manifest to open the OSGi Composite Bundle Mainifest editor, as seen in Figure 15-9 on page 861.

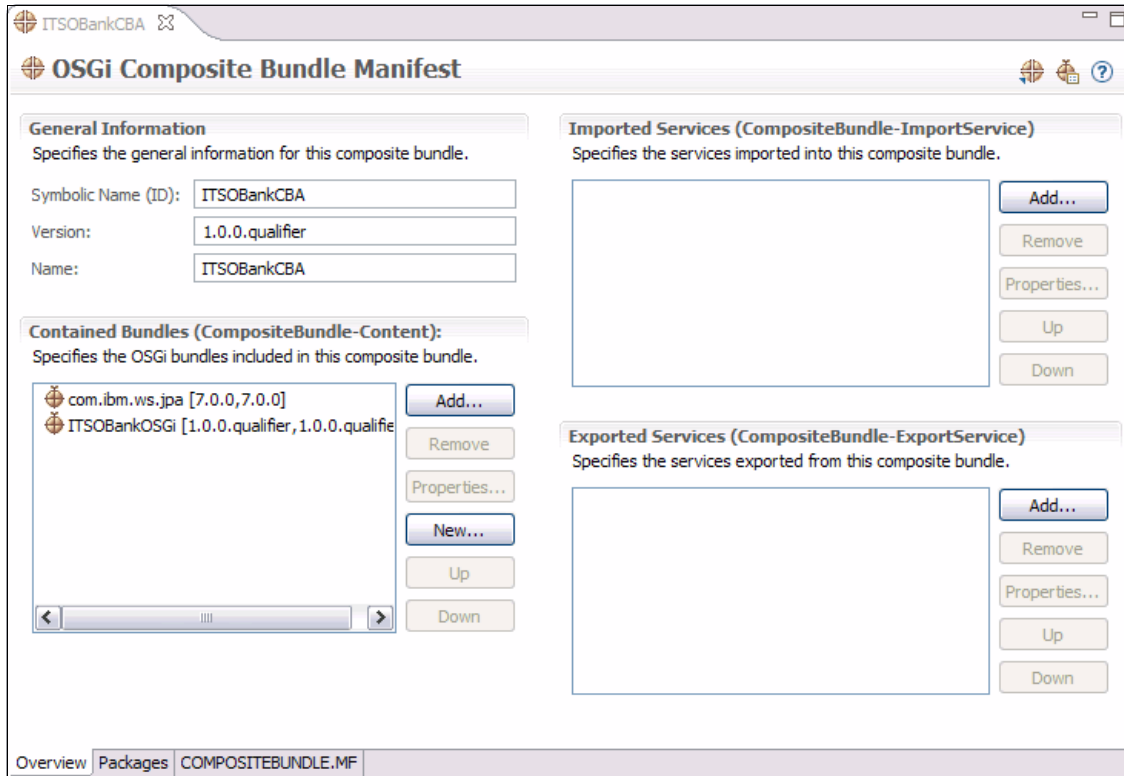


Figure 15-9 OSGi Composite Bundle Manifest file

From here, you can add or remove contained bundles, and import and export services from the CBA. On the Packages tab, you can import and export packages. The COMPOSITEBUNDLE.MF tab provides a source view of the Manifest with content assist.

15.6 Developing OSGi applications

We have given an overview of the procedures for developing an OSGi application in this chapter. For this example, we have used WebSphere Application Server V7 installed with the OSGi and JPA2.0 Feature Pack. We can instead use WebSphere Application Server V8 Beta, which already has OSGi support and does not need any feature packs installed.

The next step is to build a complete OSGi application. We use the same sample ITSO bank application. This application is built using a three-tier architecture with

both the front-end interfaces and back-end business logic in an OSGi application. This OSGi application will be made up of four bundles:

- ▶ A business bundle that contains the business logic and acts as an intermediary between the presentation logic and database
- ▶ A persistence bundle that encapsulates the JPA-based database access pattern
- ▶ An interface bundle for servlets, JavaServer Pages (JSP), and static content, which delegates the actual business functionality to the business bundle
- ▶ An API bundle that contains the interfaces that connect the three other bundles

The sample code is available with the additional material supplied with this book at 7835code\osgi. Store the contents of this directory on your C: drive.

15.6.1 API bundle

The first step is to define the API that delineates the boundaries between presentation (web), business logic, and persistence logic. This API will be placed in a separate OSGi bundle project for the purpose of modularity. Create the API bundle project using the following procedure:

1. Click **New** → **OSGi Bundle Project** to open the New OSGi Bundle Project wizard, as shown in Figure 15-10 on page 863. Complete these steps:
 - a. In the Project name field, enter `itso.bank.app`.
 - b. In the Target runtime drop-down list, select **WebSphere Application Server v7**.
 - c. Clear **Add Bundle to application**.
 - d. Click **Finish**.

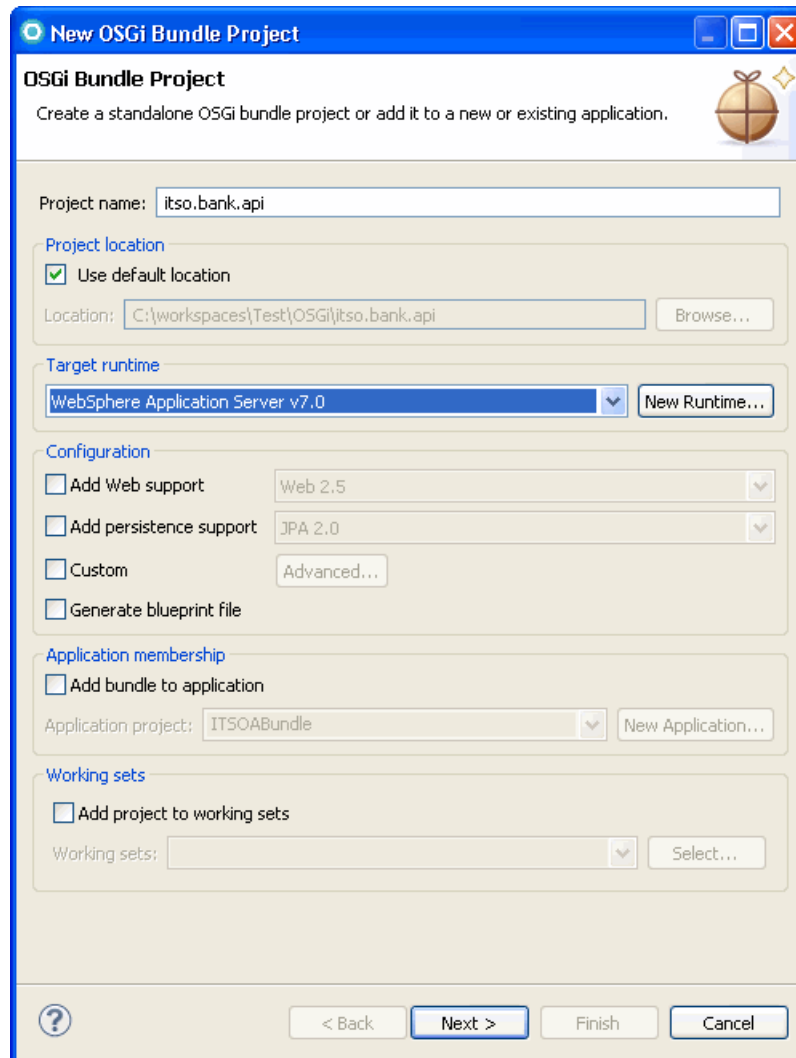


Figure 15-10 New ITSO API bundle project

After you create the project, import the sample classes into the `itso.bank.api` project from the `src` folder in the `C:\osgi\itso.bank.api\src\itso` directory. The project now contains three packages:

- ▶ `itso.bank.api`: A package for the business API
- ▶ `itso.bank.api.persistence`: A package for the persistence API
- ▶ `itso.bank.api.exceptions`: A package for the exception class

In order for the packages to be available to the other bundles, you must export them in the bundle manifest file. Edit the manifest file and export the packages in the **Runtime** tab, as shown in Figure 15-11. Set the version of each package to 1.0.0 by selecting **Properties**.

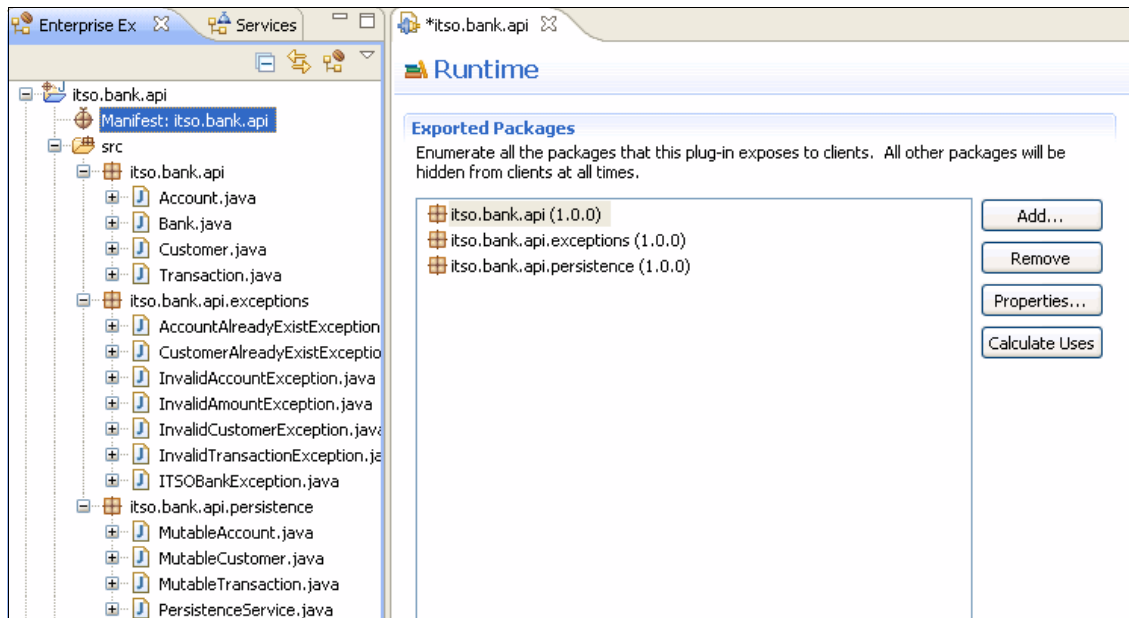


Figure 15-11 ITSO API bundle manifest file

15.6.2 Persistence bundle

The first step, if you have not done so already, is to configure the ITSOBANK database. The database will be used as part of the configuration in the persistence bundle. The next step is to build the persistence layer bundle. The procedure is similar to the steps that were used to create the API bundle:

1. Click **New** → **OSGi Bundle Project** to open the New OSGi Bundle Project wizard, as shown in Figure 15-10 on page 863. Complete these tasks:
 - a. In the Project name field, enter `itso.bank.persistence`.
 - b. In the Target runtime drop-down list, select **WebSphere Application Server v7**.
 - c. Clear the check box **Add Bundle to application**.
 - d. Select the **Add persistence support** check box and accept the default value of **JPA 2.0** from the corresponding drop-down list. Click **Next**.

2. On the JPA Facet page of the wizard, select the connection **ITSOBank** (as shown in Figure 15-12 on page 866). If no connection exists, create a new one by clicking **Add Connection** to define the connection to the ITSOBANK sample Apache Derby database.
3. Click **Finish**.

Important: See “Setting up the ITSOBANK database” on page 1880 for instructions to create the ITSOBANK database. For the JPA entities, we can either use the DB2 or Derby database. For simplicity, we suggest using the built-in Derby database in this chapter.

The files for creating ITSOBANK for this project are available at `\7835code\osgi\itsobank_db_derby`.

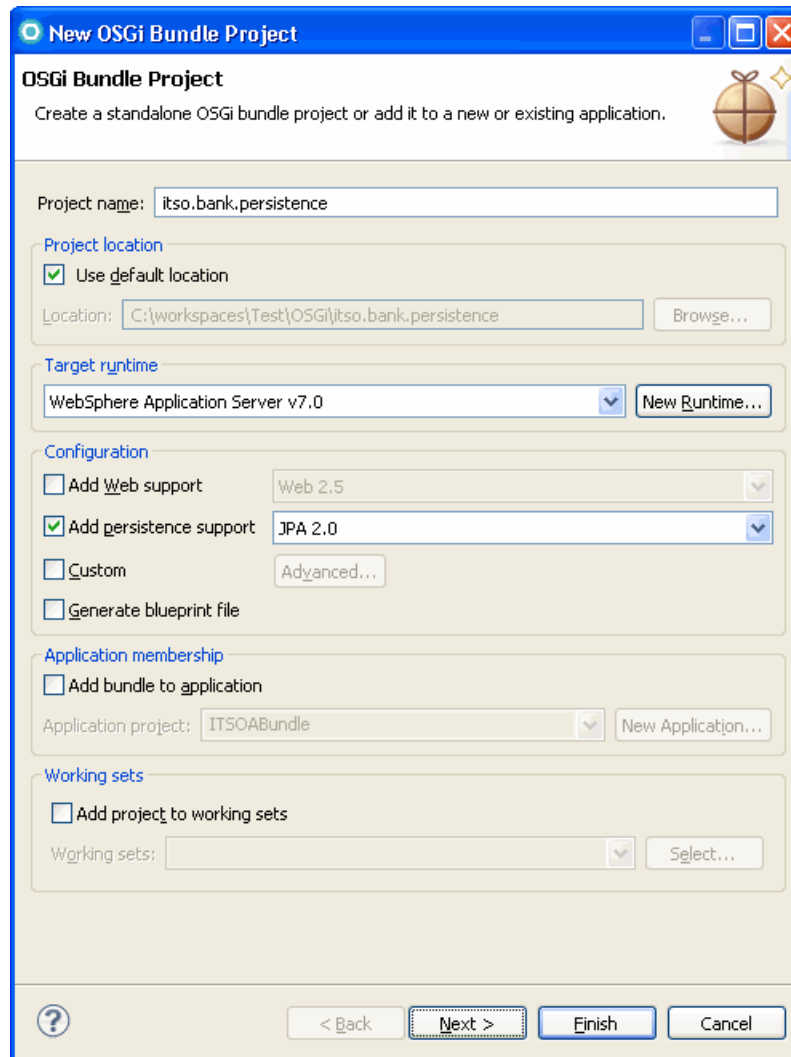


Figure 15-12 ITSO persistence bundle project

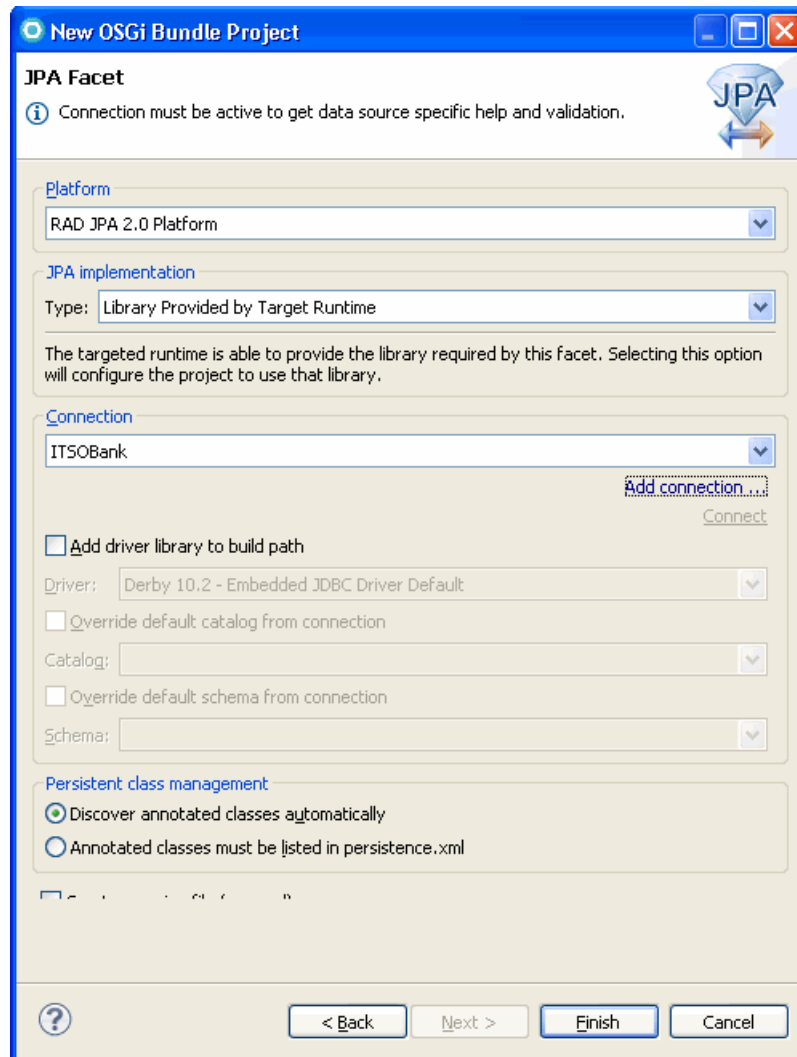


Figure 15-13 ITSO persistence bundle JPA Facet

Import the sample classes into the `itso.bank.persistence` project from the `src` folder in the `C:\osgi\itso.bank.persistence\src\itso` directory. The project now contains two packages:

- ▶ `itso.bank.persistence`: A package for the persistence service implementation
- ▶ `itso.bank.persistence.entity`: A package for the persistence entities

Compilation errors: After importing the classes, you will get compilation errors about imports that are not resolved. In the Markers view, select any of these errors, and check the quick fixes. The first quick fix adds an import to the bundle manifest. Select the first quick fix, and the compilation errors are fixed. You probably need to perform this step as many times as there are packages that you need to import.

Configuring the persistence unit

Rational Application Developer has already created a persistence descriptor containing the `itso.bank.persistence` persistence unit. To be usable, this persistence unit needs to specify how to obtain access to the database. In the sample, we use Java Naming and Directory Interface (JNDI) lookup:

1. In the `itso.bank.persistence` project, expand the **JPA Content** tree item.
2. Double-click the **persistence.xml** file to edit it using the Persistence XML Editor.
3. In the **Design** tab, select the **Persistence Unit** and provide the following information in the details section:
 - a. Set the Java Transaction API (JTA) data source JNDI name to `jdbc/bank`.
 - b. Set the non-JTA data source JNDI name to `jdbc/banknojta`.

These settings use traditional JNDI lookups as the data source access scheme. Two additional alternatives are described in the information center:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.osgifep.multiplatform.doc/topics/ca_jpa.html

4. Finally, right-click the **Persistence Unit** and select **Add** → **Class** and add the three persistence entities: **Account**, **Customer**, and **Transaction**.

Setting up the Blueprint descriptor

The Blueprint XML file describes the components of the application, their dependencies, and their life cycles. To create the blueprint file for the persistence bundle project, follow these steps:

1. Right-click the persistence bundle project and choose **New** → **Blueprint File**.
2. Accept the default values and click **Next**.
3. On the Add Additional Blueprint Namespaces panel, ensure that the check box for **JPA Blueprint Support** is selected.
4. Click **Finish** to create the blueprint file, which opens the new file in the Blueprint XML Editor.

5. Edit the **blueprint.xml**. Add a bean definition to the blueprint using the **Add** button. Select **Bean** from the list and click **OK**.
6. Use **Browse** to find **itso.bank.persistence.JpaPersistenceService**, as shown in Figure 15-14. Click **OK**.

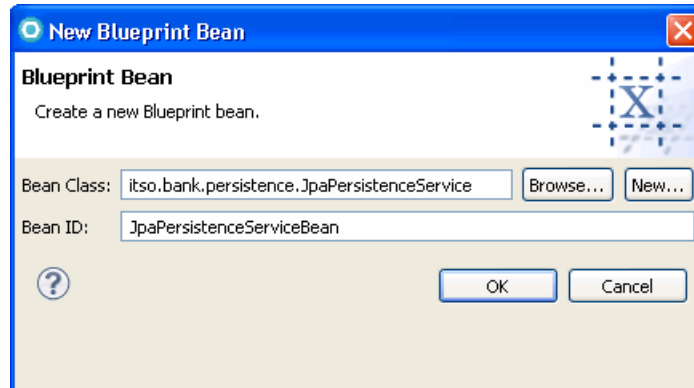


Figure 15-14 Bean definition

7. Next create a service out of the newly created `JpaPersistenceServiceBean`. Right-click the **Blueprint** root node and choose **Add** → **Service**, as shown in Figure 15-15.

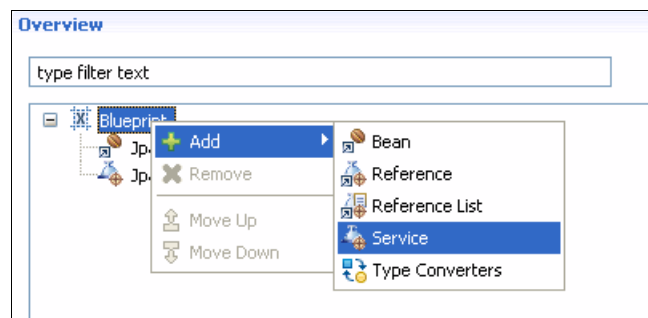


Figure 15-15 Adding the Blueprint service definition

8. Enter the following information in the New Blueprint Service dialog window:
 - a. Browse the Service Interface as **itso.bank.api.persistence.PersistenceService**.
 - b. Enter the Service ID as `JpaPersistenceServiceBeanService`.
 - c. Browse the Bean Reference as **JpaPersistenceServiceBean**.
9. Click **OK** and save the blueprint file.

15.6.3 Business logic bundle

The business logic bundle will contain our domain-specific classes, which contain the business logic of the application. We start by creating a business bundle project. The steps used to create the business logic bundle are similar to the steps used to create the persistence bundle:

1. Click **New** → **OSGi Bundle Project** to open the New OSGi Bundle Project wizard, as shown in Figure 15-16 on page 871. Complete these tasks:
 - a. In the Project name field, enter `itso.bank.biz`.
 - b. In the Target runtime drop-down list, select **WebSphere Application Server v7**.
 - c. Clear the check box **Add bundle to application** and click **Finish**.

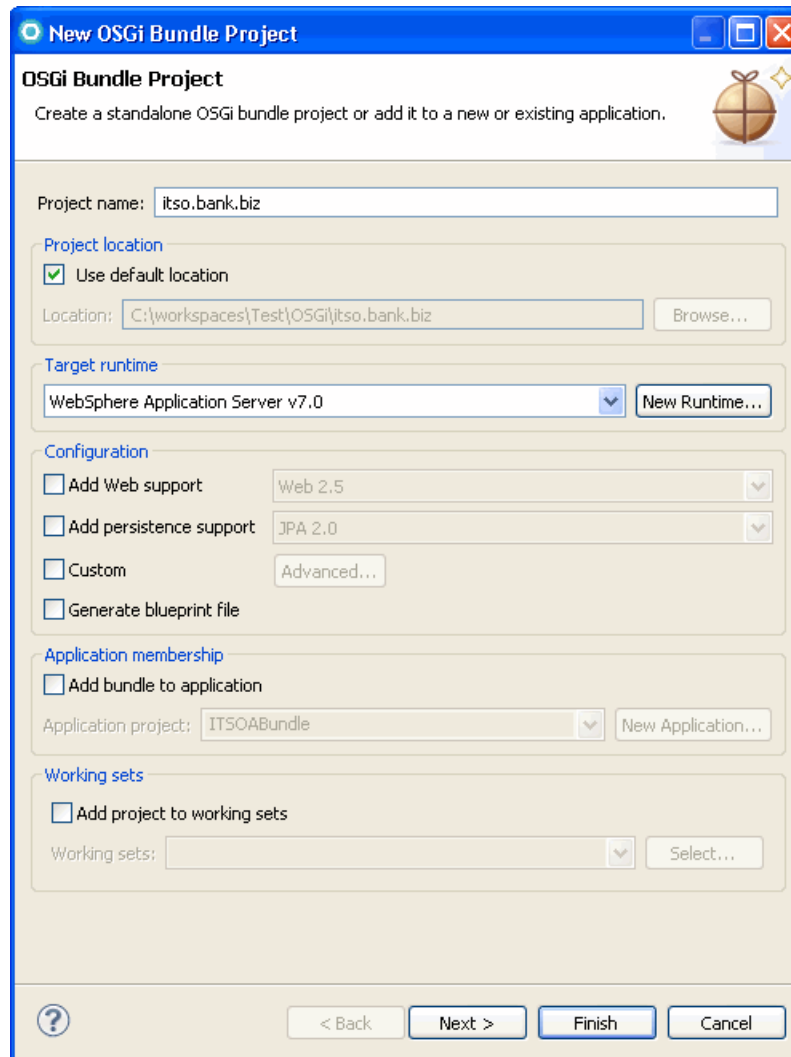


Figure 15-16 ITSO Business bundle project

Import the sample classes into the `itso.bank.biz` project from the `src` folder in the `C:\osgi\itso.bank.biz\src\itso` directory. The project now contains two packages:

- ▶ `itso.bank.biz`: A package for the business logic
- ▶ `itso.bank.biz.proxy`: A package for the proxy implementations

Setting up the Blueprint descriptor

The Blueprint XML file defines the wiring for the application. In this example, the `ITSOBankBean` implementation (`itso.bank.biz.ITSOBank`) contains a field called `PersistenceService` with the appropriate getter and setter. By configuring the property in the Blueprint XML file, we allow the OSGi framework, and in particular, the Blueprint Container specification, to resolve the `itso.bank.api.persistence.PersistenceService` and provide an instance of that class to the `ITSOBankBean`.

The `ITSOBankBean` does not need to create the `PersistenceService` object, and it does not know who created the object. It simply uses the one that is provided to it. This method is the fundamental principle of dependency injection and inversion of control.

To create the blueprint file for the business bundle project, follow these steps:

1. Right-click the business bundle project and choose **New** → **Blueprint File**.
2. Accept the default values and click **Next**.
3. On the Add Additional Blueprint Namespaces panel, ensure that the check box for **Blueprint Transaction Support** is selected.
4. Click **Finish** to create the blueprint file, which opens the new file in the Blueprint XML Editor.
5. Edit the `blueprint.xml`. Add a bean definition to the blueprint using **Add**. Select **Bean** from the list and click **OK**.
6. The bean class implementation points to `itso.bank.biz.ITSOBank`, as shown in Figure 15-17. Click **OK**.

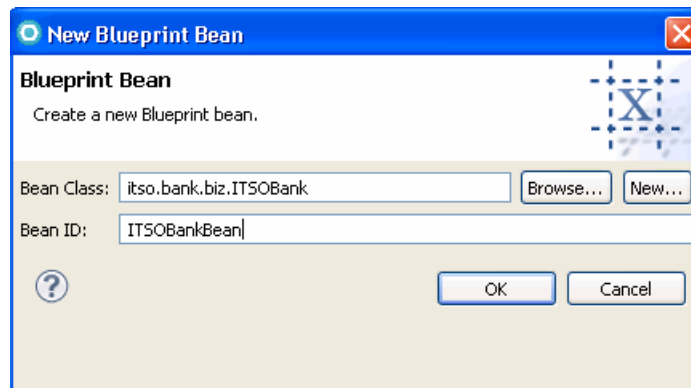


Figure 15-17 ITSO business bean definition

7. The next step is to create an OSGi Service out of the newly created ITSOBankBean. Right-click the **Blueprint** root node and choose **Add** → **Service**.
8. Enter the following information in the New Blueprint Bean: Blueprint Service window (Figure 15-18):
 - a. For Service Interface, enter `itso.bank.api.Bank`.
 - b. For Service ID, enter `ITSOBankBeanService`.
 - c. For Bean Reference, enter `ITSOBankBean`. Click **OK**.

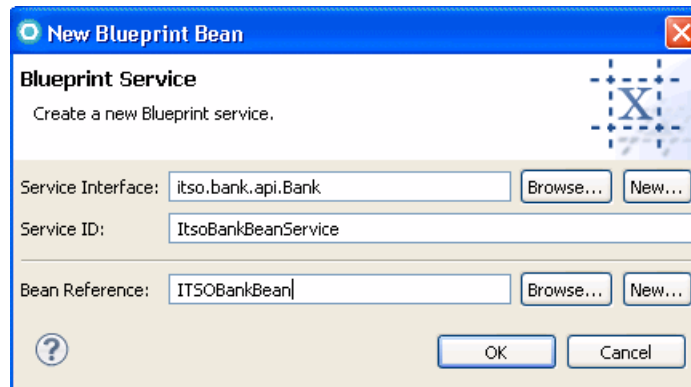


Figure 15-18 ITSO business bean service

9. Add a reference to the PersistenceService created in step 7 on page 869. Right-click the **Blueprint** root node, choose **Add** → **Reference**, and provide the following data:
 - a. Set the Reference Interface to `itso.bank.api.persistence.PersistenceService`.
 - b. Set the Reference ID to `service`.
 - c. Omit the Bean Reference value and click **OK**.
10. Add a property to the ITSOBankBean definition that references the service (Reference) that was created in step 9. Right-click the **ITSOBankBean (Bean)** and choose **Add** → **Property**.
11. Select the new (Property) item and provide the following values in the Details section:
 - a. Enter `persistenceService` as the property name.
 - b. Enter `service` as the reference value.
12. Save the blueprint file.

15.6.4 Web interface bundle

The web interface bundle will contain our web resources used in the presentation layer of the application. We start by creating a web interface bundle project. The steps used to create the web interface bundle are similar to the steps that were used to create the persistence bundle:

1. Click **New** → **OSGi Bundle Project** to open the New OSGi Bundle Project wizard, as shown in Figure 15-19 on page 875. Perform these tasks:
 - a. In the Project name field, enter `itso.bank.web`.
 - b. In the Target runtime drop-down list, select **WebSphere Application Server v7**.
 - c. Clear the check box **Add bundle to application**.
 - d. Click **Add Web support** and accept the default value of **Web 2.5** from the associated drop-down list. Click **Finish**.

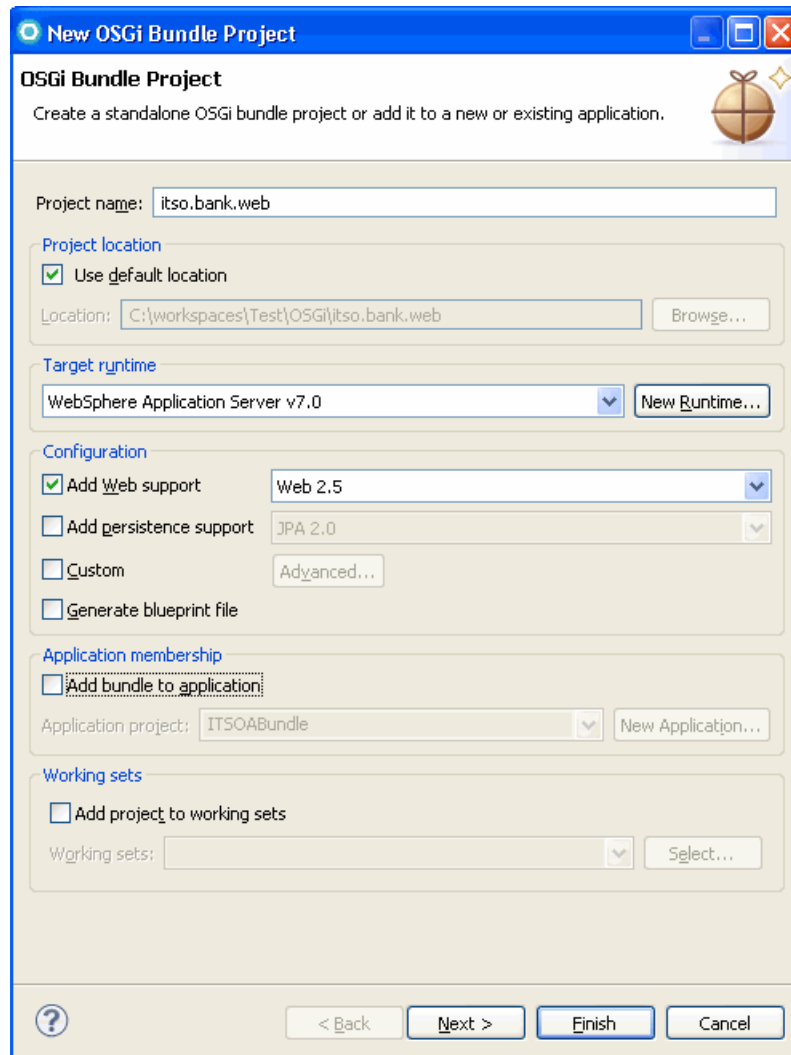


Figure 15-19 ITSO web bundle project

Import the sample classes into the `itso.bank.web` project from the `src` folder in the `C:\osgi\itso.bank.web\src\itso` directory. The project now contains three packages:

- ▶ `itso.bank.web.command`: Hosts command pattern implementations
- ▶ `itso.bank.web.servlet`: Hosts Java servlet implementations
- ▶ `itso.bank.web.util`: Contains a single utility class to look up the bank business service implementation

Also, import the complete webcontent folder into the project structure.

The JSP pages imported from the webcontent folder depend on third-party materials that need to be downloaded and copied to the webcontent folder. The required JARs must be copied to WEB-INF/lib path, as shown in Figure 15-20. The required third-party JARs are listed in the 3rd-party-materials.txt file in the \7835code\osgi sample code directory.

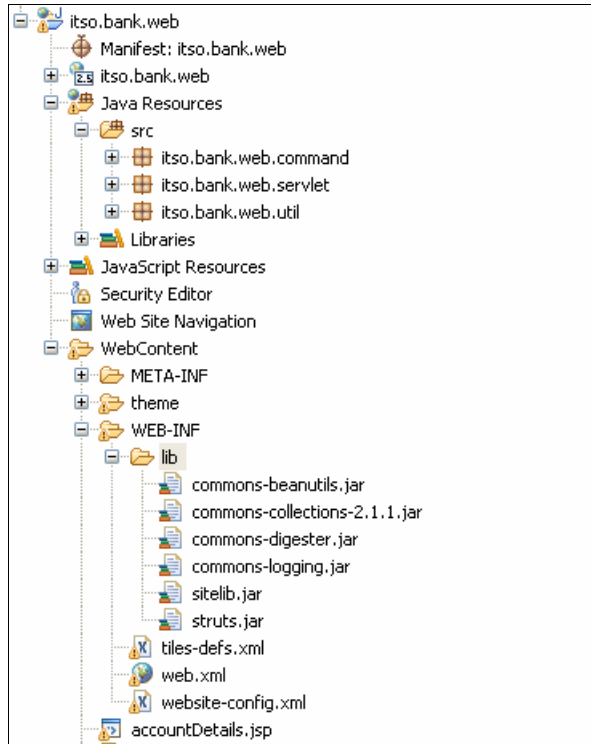


Figure 15-20 Required web content third-party JARs

Connecting to the business services via JNDI

We need to hook up the servlet classes with the Bank implementation, as provided by the `itso.bank.business` bundle. This lookup is done in the `ITS0Bank` utility class via a standard JNDI lookup but using a special OSGi-aware namespace.

The sample class that is shown in Example 15-5 on page 877 is included in `itso.bank.web.util`.

Example 15-5 ITSOBank class

```
public class ITSOBank {
public static Bank getBank() {
Bank bank = null;
try {
Context ctx = new InitialContext();
bank = (Bank) ctx.lookup("osgi:service/itso.bank.api.Bank");
} catch (NamingException e) {
// ... appropriate error handling
}
return bank;
}
}
```

15.6.5 Application OSGi

Finally, to complete the ITSO application, you need to create an OSGi application project. The following steps describe the procedure:

1. Click **New** → **OSGi Application Project** to open the New OSGi Application Project wizard, as shown in Figure 15-21 on page 878. Complete these tasks:
 - a. In the Project name field, enter `itso.bank.app`.
 - b. In the Target runtime drop-down list, select **WebSphere Application Server v7**. Click **Next**.

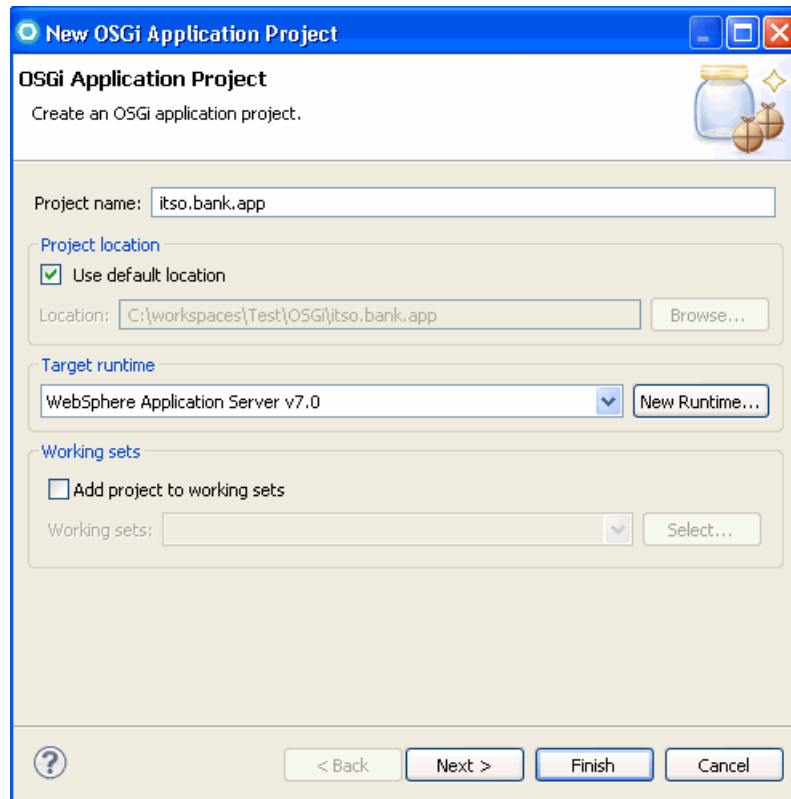


Figure 15-21 ITSO application project

2. On the Contained OSGi Bundles window, select the following bundles, as shown in Figure 15-22 on page 879:
 - itso.bank.api
 - itso.bank.biz
 - itso.bank.persistence
 - itso.bank.web
3. Click **Finish**.

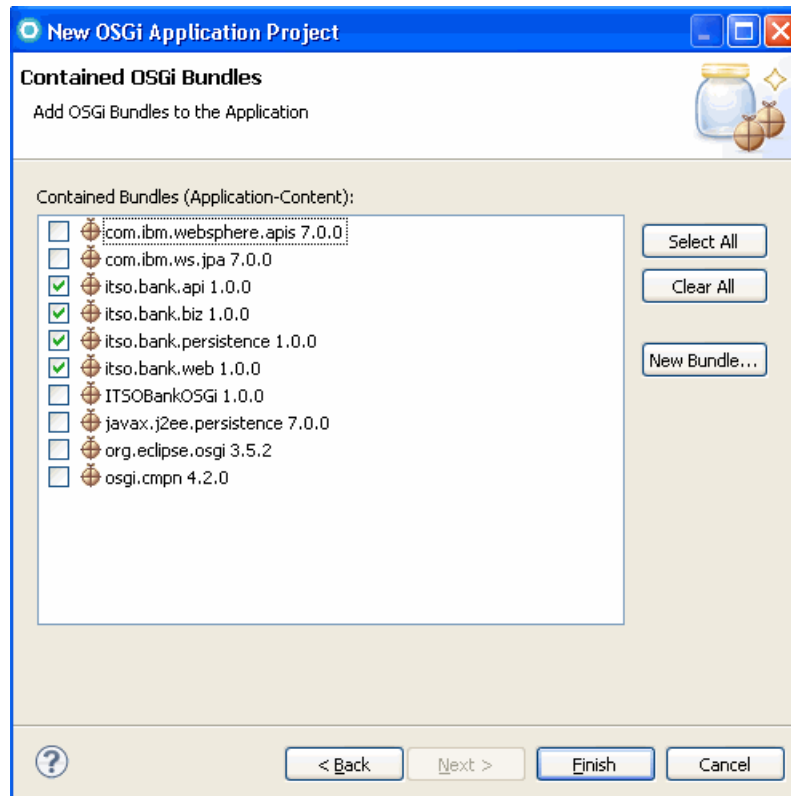


Figure 15-22 ITSO Bank application bundle

15.6.6 Deploying the OSGi application

Before you start deploying the OSGi application, you must configure the JTA and non-JTA data sources in WebSphere Application Server. We have configured the persistence unit during the creation of the persistence bundle in 15.6.2, “Persistence bundle” on page 864. Follow these steps to configure the JTA and non-JTA data sources in WebSphere Application Server:

1. Start the WebSphere Application Server v7 runtime server, open the administrative console, and create the Java Database Connectivity (JDBC) data sources:
 - a. Go to **Resources** → **JDBC** → **JDBC providers**.
 - b. Select **server** scope. Click **New**.
 - c. Set the database to **derby**.
 - d. Set the Provider type to **Derby JDBC Provider 40 (Type 4 JDBC driver)**.

- e. Set the implementation type to **Connection pool data source**.
 - f. Enter a text string as a JDBC provider name.
 - g. Click **Next** and click **Finish**. Save changes.
2. Now go to **Resources** → **JDBC** → **Data sources** and create the JTA and non-JTA data sources:
 - a. Select the server scope, click **New**, and enter DS1 for a data source name.
 - b. Enter a JNDI name, jdbc/bank, for the JTA data source. We have configured this JNDI name in the persistence.xml descriptor during the persistence bundle creation step. Click **Next**.
 - c. Click **Select an existing JDBC provider** and select the Apache Derby JDBC provider that you created. Click **Next**.
 - d. Give the Database name as the location pointing to the path where the database was created, for example:
E:\Program~1\IBM\SDP\runtimes\base_v7\profiles\WTE_APPSRV7_FEP1\ITSOBANK or \${USER_INSTALL_ROOT}/ITSOBANK. Click **Next**.
 - e. Leave the security aliases as is, click **Next**, click **Finish**, and save the changes.
 3. Follow the preceding steps to configure a non-JTA data source with a separate JNDI name of jdbc/banknojta. Then perform the following steps:
 - a. Under Additional properties, click **WebSphere Application Server data source properties**.
 - b. Under General Properties, check **Non-transactional data source** to indicate that this particular data source does not support transaction handling.
 4. To export the OSGi application as an Enterprise Business Archive (EBA), select the application project, and from the context menu, select **Export** → **OSGi application (EBA)**, which generates an archive that you can deploy to any OSGi-aware server.
 5. Deploying the application in Rational Application Developer is simple, because OSGi applications are fully supported in the WebSphere Application Server test environment. In the Servers view, right-click the **WebSphere Application Server V7** server instance that has the OSGi feature installed and select **Add and Remove**. In the resulting dialog window, select **itso.bank.app** and move it to the list of configured applications, as shown in Figure 15-23 on page 881.

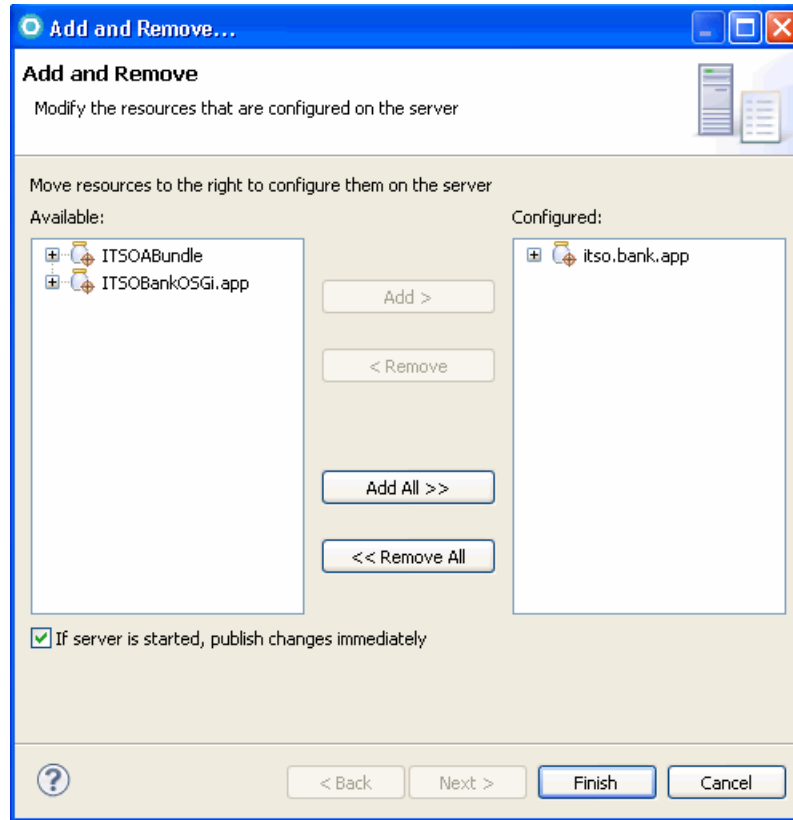


Figure 15-23 Deploying the OSGi application

Testing the application

To test the application, select the **redbank.html** file within the project. From the context menu, select **Run As** → **Run on server** and enter an account number, for example, 111-11-1111. At this stage, the application provides the capabilities to update customers, inspect accounts, view transactions, and create transactions.

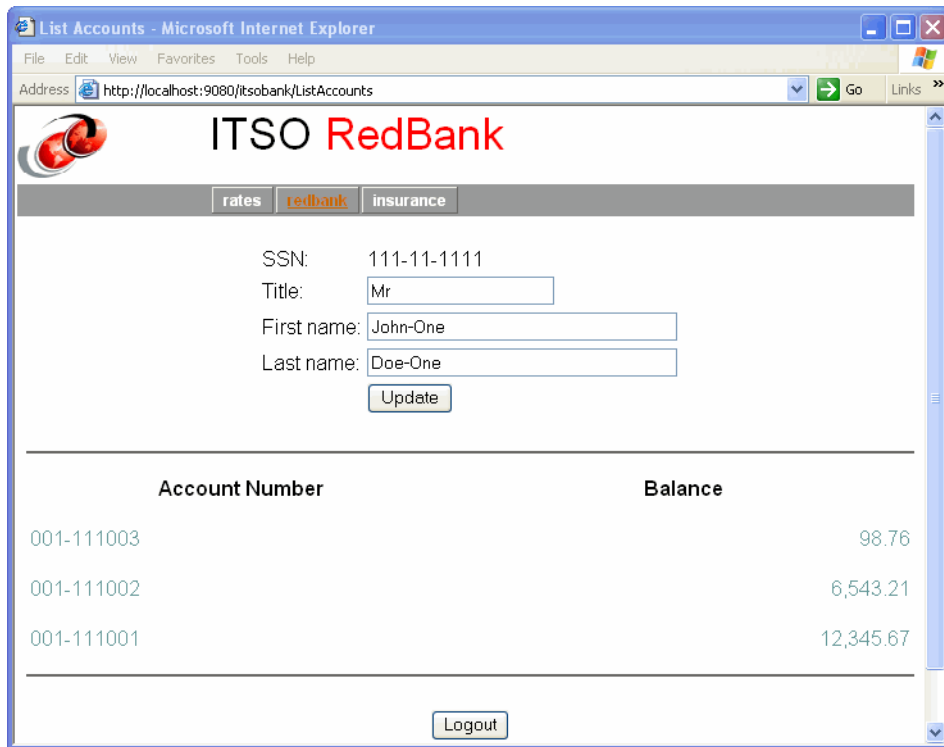


Figure 15-24 OSGi application

Graphical Bundle Explorer

You can view the OSGi bundle that we created and the bundles inside it with all the dependencies among them using the Bundle Explorer. To do that, select the OSGi application in the workspace, and from the context menu, select **Show In → Bundle Explorer**. This selection opens a viewer that shows all the bundles inside the application, and you can view dependencies between them, and dependencies on JARs that are in the target platform, as well.

Additional scenarios

We have created a simple application that we developed and deployed in this chapter. More complex scenarios are available:

- ▶ Connecting two OSGi applications
- ▶ Connecting JEE to OSGi applications
- ▶ Connecting OSGi applications to JEE
- ▶ Connecting Java Message Service (JMS) to OSGi applications

Converting non-OSGi projects to OSGi bundles

Many users start with existing Java or Java EE applications that are running. They want to explore how these applications will function within the OSGi framework and run time, and they want to benefit from the modularity that OSGi offers.

Starting with a Java, Dynamic Web, or JPA Utility project, or even a PDE plug-in project, you can select the project, and from the context menu, select **Configure** → **Convert to OSGi Bundle Project**.

This selection simply adds a Manifest to the project and exports any package within the project that is not marked as internal. It also adds to the Manifest an import packages header to list packages imported by Java classes within the project. The header excludes packages that are defined within the project, or within JARs inside the project.

Java EE web projects

Java EE web projects can be used inside an OSGi application without being converted to an OSGi bundle. WebSphere Application Server converts these Java EE web projects to a web bundle after they are deployed to the server.

You can add a Java EE web project to an OSGi application using the application Manifest Editor, under the Dynamic Web Projects section.

PDE plug-ins can be added to an OSGi application without being converted to an OSGi bundle, as well. PDE plug-ins are added to the application under the application contents with the other bundles.

15.7 Further information

For more information about OSGi and OSGi application development, see *Getting Started with the Feature Pack for OSGi Applications and JPA 2.0*, SG24-7911.



Developing Service Component Architecture (SCA) applications

In this chapter, we describe the Rational Application Developer and WebSphere Application Server implementation for Service Component Architecture (SCA). SCA is a specification that was created by a consortium of companies called the Open Service Oriented Architecture (OSOA) collaboration. We describe examples that show how SCA helps to produce new services, use existing services, connect heterogeneous components, and offer a variety of communication mechanisms to make reuse and interoperability easier.

This chapter discusses the following topics:

- ▶ Introduction to SCA
- ▶ SCA project creation or augmentation
- ▶ Developing a Java component from a WSDL interface
- ▶ Creating a contribution to include the deployable composites
- ▶ Deploying the contribution to WebSphere Application Server
- ▶ Testing the services provided by the SCA application
- ▶ Wiring a component to a service on another component
- ▶ Reusing an existing Java EE application to create a component
- ▶ Adding intents and policies
- ▶ More information

16.1 Introduction to SCA

Service Component Architecture is a programming model for service-oriented architectures (SOA). It is based on the Open Service-Oriented Architecture (<http://osoa.org>) and the deployment of SCA assets to the WebSphere Application Server V7.0 Feature Pack for Service Component Architecture (SCA) and WebSphere Application Server V8.0.

Central to SCA is the concept of *service*, which in this context is much broader than in the concept of a *web service*. A web service in SCA is a *binding* (a communication mechanism) that a service might offer to its clients. There are many other bindings, making the concept of service in SCA much more general. However, SOAP-based web services can be reused or created in SCA, which is an instance of SCA being an *inclusive* programming model.

The other central notion is that of a *component*. SCA is *not bound to a specific type of implementation* (in Java, for example), so components can be implemented with a variety of technologies, including Java Platform, Enterprise Edition (Java EE), Spring, OSGi, and SCA Composite (recursive composition). The fundamental idea informing the SCA programming model is not to replace existing technologies, but to allow applications written in separate languages and residing on remote systems to *coexist and communicate*.

From a technological point of view, SCA makes use of descriptors written in XML, which can be visualized in easily understandable diagrams.

16.1.1 Concepts

In this section, we illustrate the concepts of SCA using images taken while following the SCA Hello World Tutorial contained in the product. This tutorial uses a *bottom-up* approach to generate an SCA service, starting from the Java implementation of the service, which has a Java interface annotated with `@Remoteable`. It might be helpful to follow the SCA Hello World Tutorial while going through this introductory material.

Component

A *component* is the fundamental building block of every SCA application, because it performs a unit of business function. The simplest SCA component in Rational Application Developer is shown in Figure 16-1 on page 887.

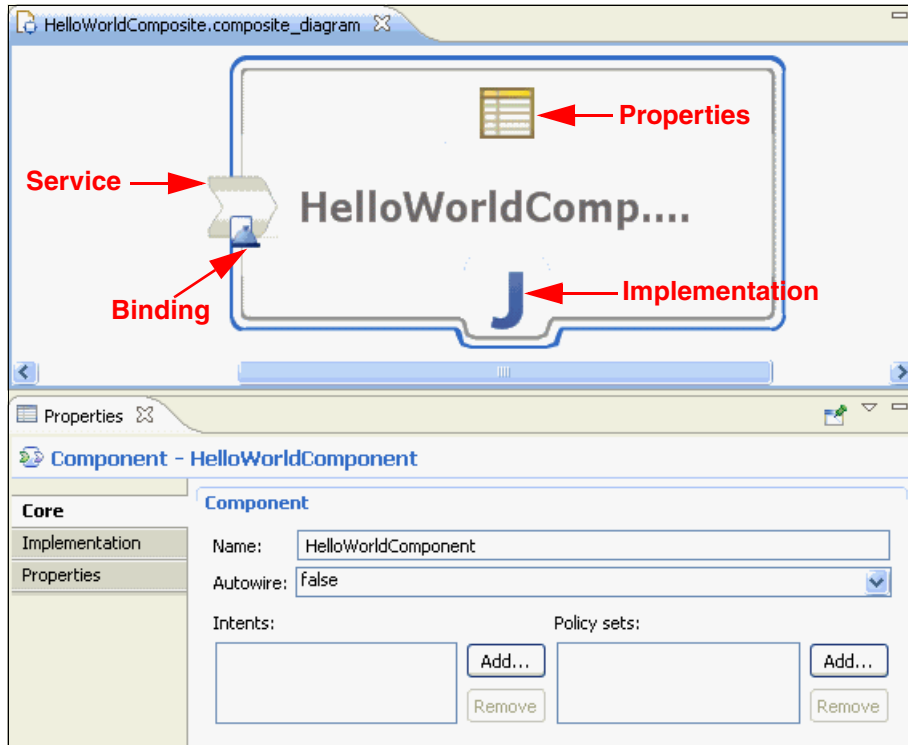


Figure 16-1 Component from SCA Hello World Tutorial

Component implementation

You can implement a component using various technologies, such as a Java class, an SCA composite, a Spring application, or a Java EE application, as shown in Figure 16-2.

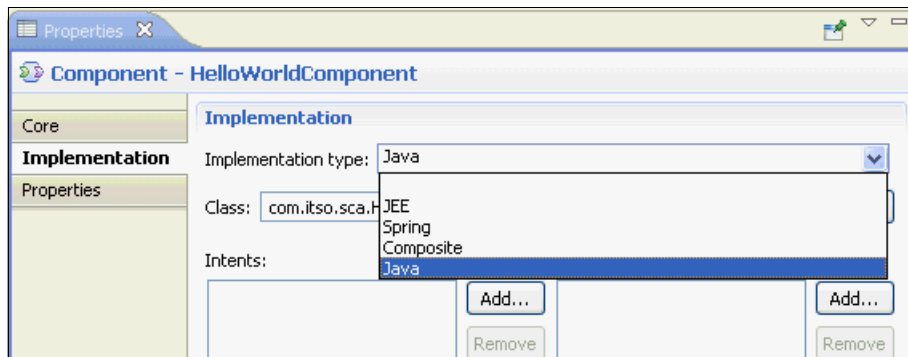


Figure 16-2 Property page showing possible component implementations

Service: Interface and binding

A component provides zero, one, or many services. A service is characterized by an interface and one or more bindings. The interface can be implemented either in Java or as a Web Services Description Language (WSDL) document (Figure 16-3).

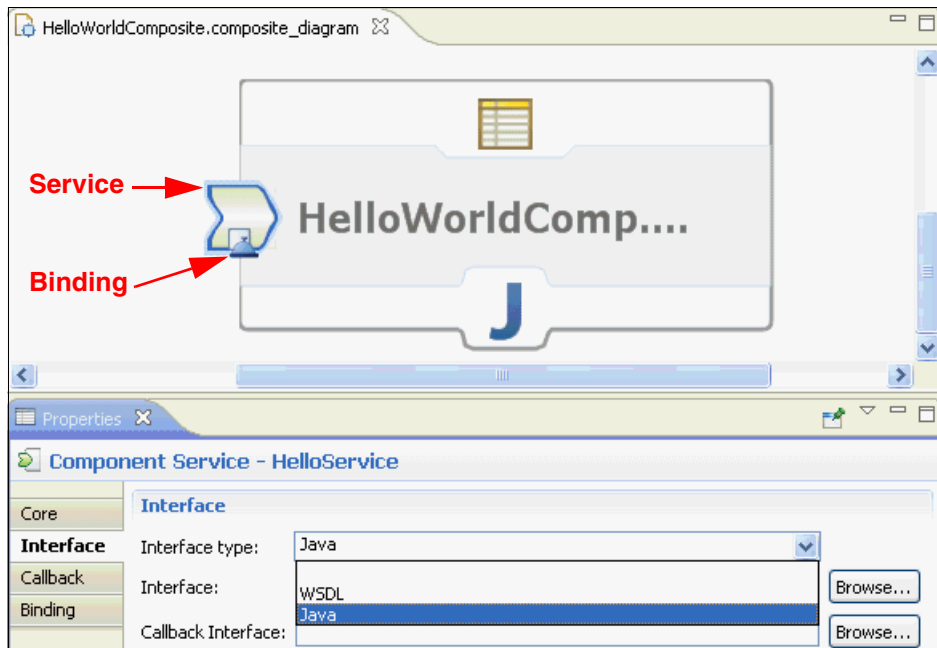


Figure 16-3 Service interface can be Java or WSDL

The bindings of a service specify the many possible communication mechanisms or protocols that a client can use to interact with the service.

Many possible types of bindings are available for the service: Webservice, HTTP, Enterprise JavaBeans (EJB), Java Message Service (JMS), SCA, and Atom, as shown in Figure 16-4 and Figure 16-5 on page 889.

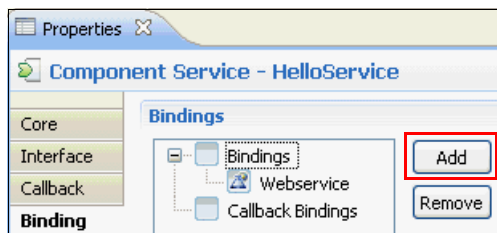


Figure 16-4 Adding bindings to a service

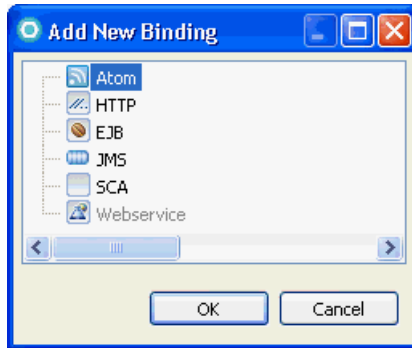


Figure 16-5 Available binding types

Component references

A *reference* is a declaration of a service on which the component implementation depends and might need to invoke to do its job. A reference is characterized by an interface and a set of bindings. The interface of a reference declares a set of business operations or methods on which the component implementation depends (Figure 16-6).

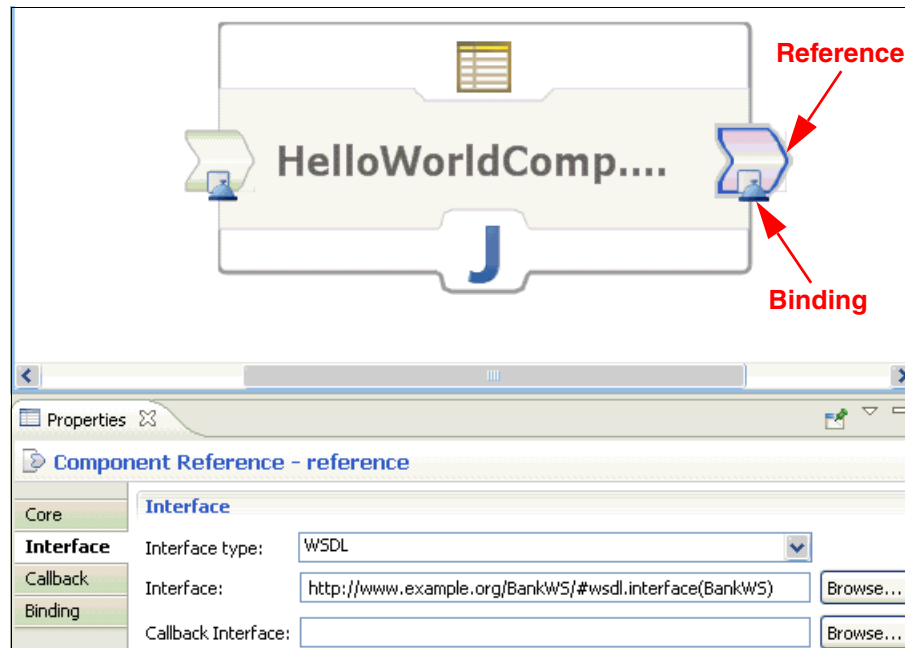


Figure 16-6 Component reference with a WSDL interface

Component properties

Component properties are data values that can be injected into components. Their reference kind can be *type*, in which case you can browse for a list of XML schema types, or *element*, in which case you can browse for schema global elements (Figure 16-7)).

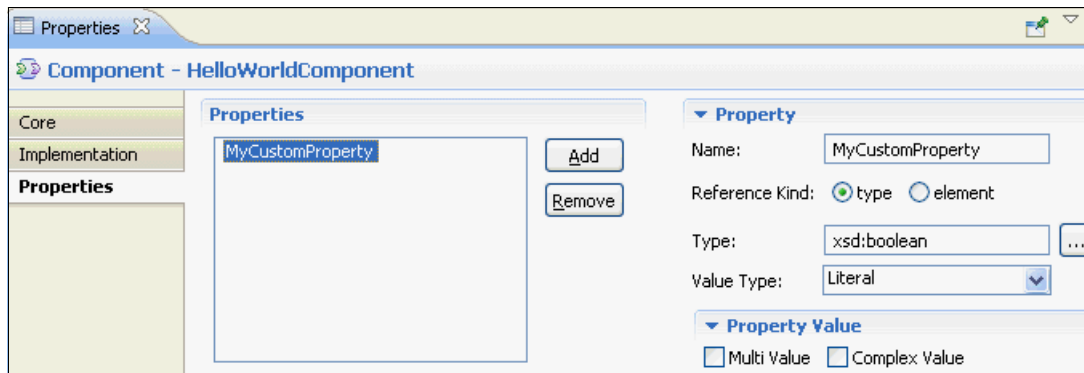


Figure 16-7 Adding properties to components

Figure 16-8 on page 891 shows a schematic representation of a component, summarizing all of the concepts in Figure 16-7.

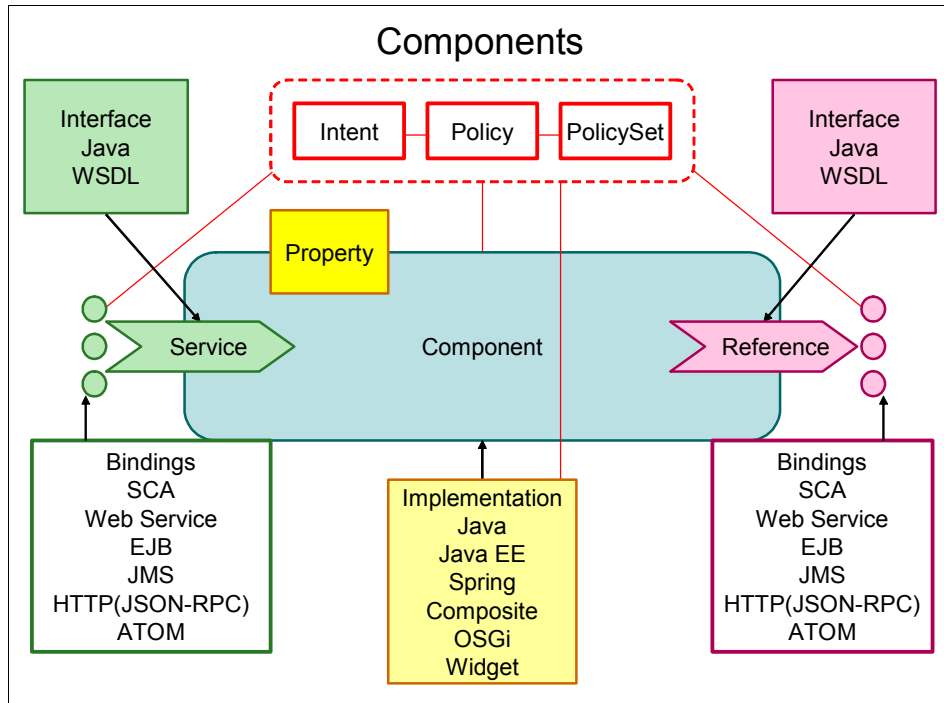


Figure 16-8 Component, services, interfaces, bindings, references, properties, and intents

Intent, policy and policy set

An *intent* is a declaration of an abstract policy or quality of service required of a component, service, or reference. Intents relieve the developers of the burden of having to understand complex, concrete policies and delegate this task to the deployer.

A *policy set* is a collection of mutually compatible concrete policies that can be applied to a specific binding type or implementation type. Although policy sets can be attached to components, services, and references at development time, SCA recommends a late binding approach in which policy sets and bindings are selected at deployment time.

A *policy* is a concrete assertion of a capability, constraint, or other non-functional requirement to be honored by a component, service, or reference.

Composite

A *composite* (Figure 16-10 on page 893) is used to assemble components. It is the smallest unit of deployment in SCA. It contains components, services,

references, and interconnecting wires. There are two kinds of composites: deployable and implementation.

Deployable composites

A *deployable composite*, also known as a *runnable composite* or *top-level composite*, declares itself deployable in its containing contribution. You can see it in the WebSphere Integrated Solutions Console (administrative console), as shown in Figure 16-9.

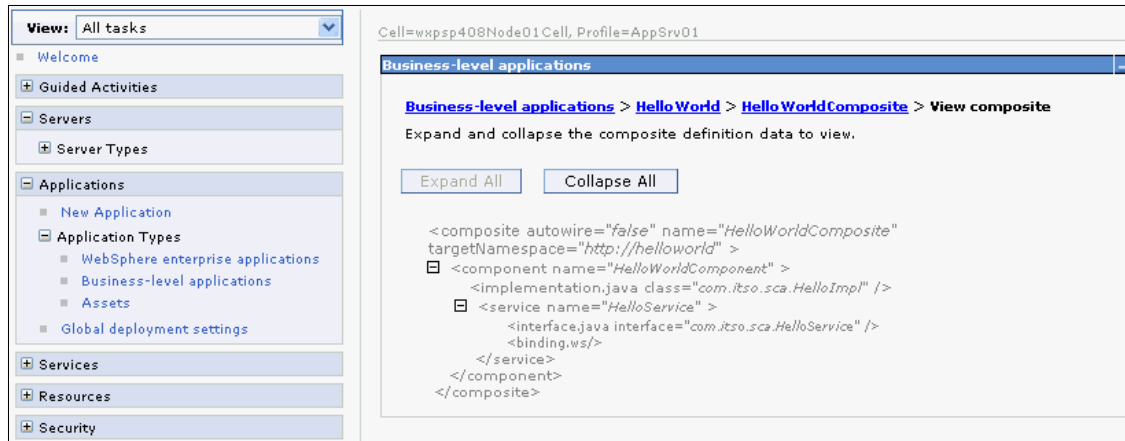


Figure 16-9 Deployable composite as seen in the WebSphere administrative console

Implementation composites

An implementation composite, or *inner composite*, is used as the implementation of a component. When used as an implementation, the components, services, references, and *wires* inside the composite are invisible and not directly accessible outside of the composite. Using a composite as the implementation of a component provides a powerful form of *reuse*. In order to use a composite as an implementation, the desired services and references must be *promoted* from the components within the composite, and they must satisfy the signature of the services and references declared on the component being implemented (Figure 16-10 on page 893).

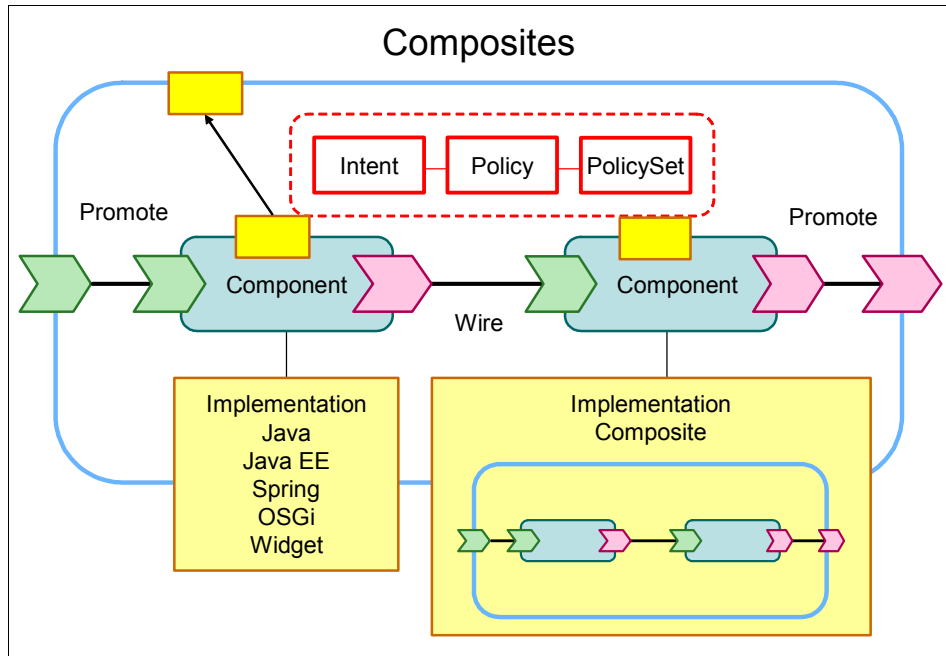


Figure 16-10 Composite with a component implemented by a composite

Remember the following definitions:

- Wire** A dependency from one component reference to a component service
- Promoting** The act of making a component service or reference available on the containing composite when the composite is used to implement a component in another composite

Contribution

An *SCA contribution* is a package of artifacts collected for deployment to an SCA domain (Figure 16-11 on page 894).

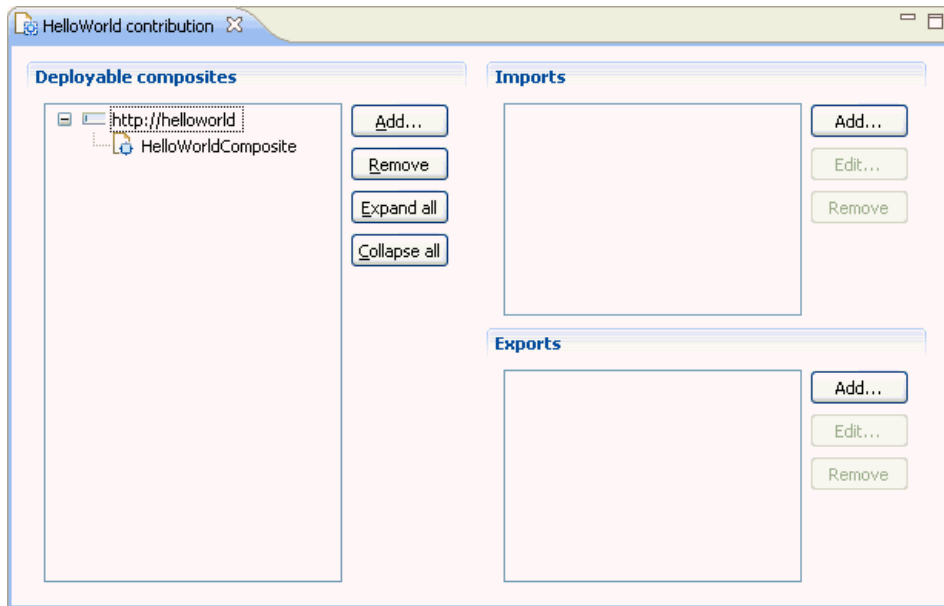


Figure 16-11 Contribution manifest editor

A contribution package can contain multiple resources; however, a typical contribution package consists of the following resources:

- ▶ SCA composites
- ▶ Component implementation artifacts, such as Java classes
- ▶ Service and reference interface artifacts, such as WSDL documents
- ▶ Supporting artifacts, such as XML schema documents
- ▶ A single, distinguished contribution metadata document, `/META-INF/sca-contribution.xml`

A *contribution metadata document* is a manifest file within a contribution package that identifies the deployable composites within the package that have components that are destined to become domain-level components. Deployable composites are distinct from implementation composites that can also reside within the package. A contribution metadata document can export namespace and Java packages for use by other contributions, or import namespace and Java packages that have been exported by other contributions.

Domain

An SCA *domain* is a well-bounded runtime entity that contains a set of service components wired together to provide a realm of business function that is meaningful to a business.

Every domain includes a *single, virtual domain-level composite*. When an actual composite is deployed to a domain, the components within the composite are added to the virtual domain-level composite, and become domain-level components. It is this mechanism that makes it possible to incrementally build up the functionality of a domain by contributing loosely related composites from multiple sources.

In SCA, domain resources, such as components, artifacts, and metadata, are not directly accessible outside of the domain. However, a domain can communicate externally through its services, references, and configured protocol bindings, such as web services.

A domain and the components that it contains can be distributed over many processes and computing systems. Do not misinterpret as domain as being the same as a process or an application server. Using the WebSphere Application Server on a single server, the domain is essentially the scope of the *server*. On a multiple server configuration, the domain is essentially the scope of the *cell*. Using the administrative console, you can visualize the domain, as shown in Figure 16-12.

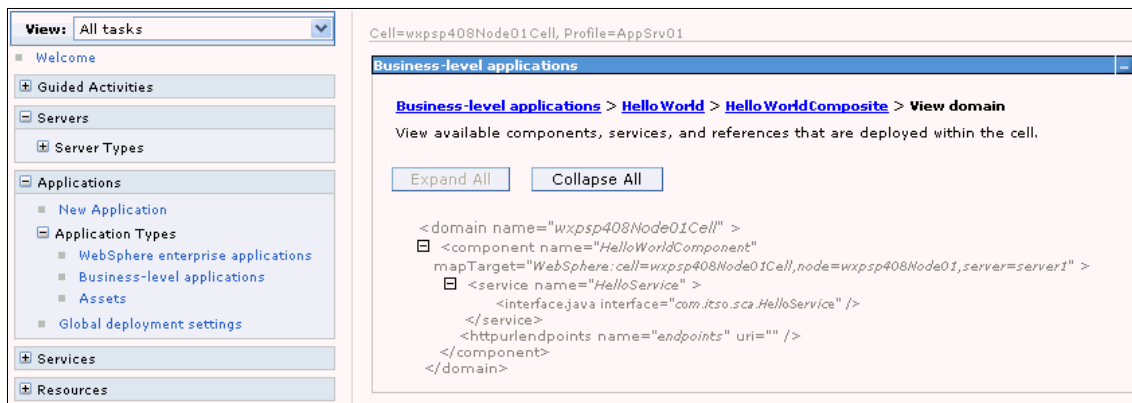


Figure 16-12 Domain as seen in the administrative console

16.1.2 Runtime support

SCA is supported on the following run times, which can all be targeted by projects developed with Rational Application Developer:

- ▶ WebSphere Application Server V7.0 Feature Pack for SCA 1.0
- ▶ WebSphere Application Server V7.0 Feature Pack for SCA 1.0.1
- ▶ WebSphere Application Server V8.0 Beta

16.2 SCA project creation or augmentation

You can create a new SCA project by following these steps:

1. Select **File** → **New** → **Other**. Complete these tasks:
 - a. On the New: Select a wizard window, for Wizards, enter `sca`.
 - b. Select **SCA Project** (Figure 16-13) and click **Next**.

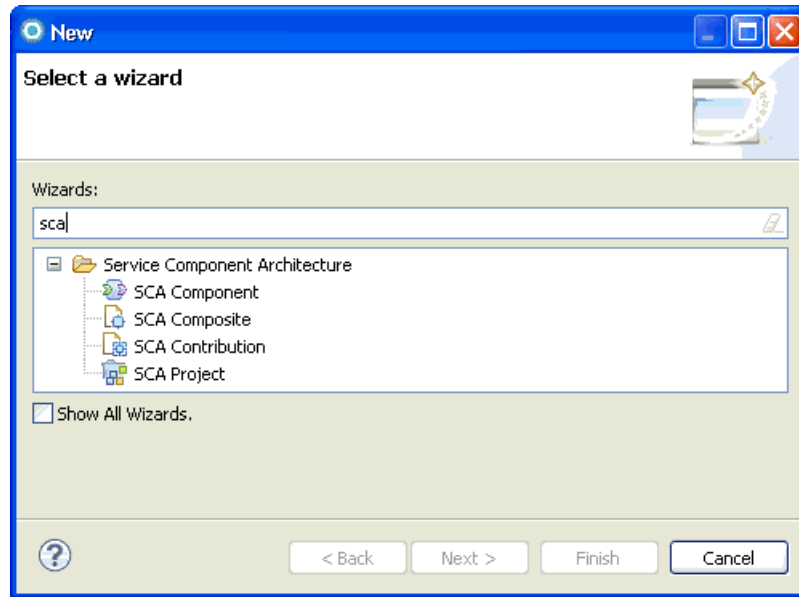


Figure 16-13 New wizard to create new SCA projects

2. In the Create a New SCA Project window, which is shown in Figure 16-14 on page 897 Complete these tasks:
 - a. Enter a project Name: `RAD8TopDownBankSCA`.
 - b. Select the Target Runtime: **WebSphere Application Server v8.0 Beta**.
 - c. Accept the Facet Configuration: **WebSphere v8.0 Beta SCA**.
 - d. Accept all available implementation types: **Composite, Java, JEE, and Spring**.
 - e. Observe that there are other possible implementation types, but they are not available in this context: **EJB, OSGi Application, Web, and Widget**.
 - f. Click **Finish**.

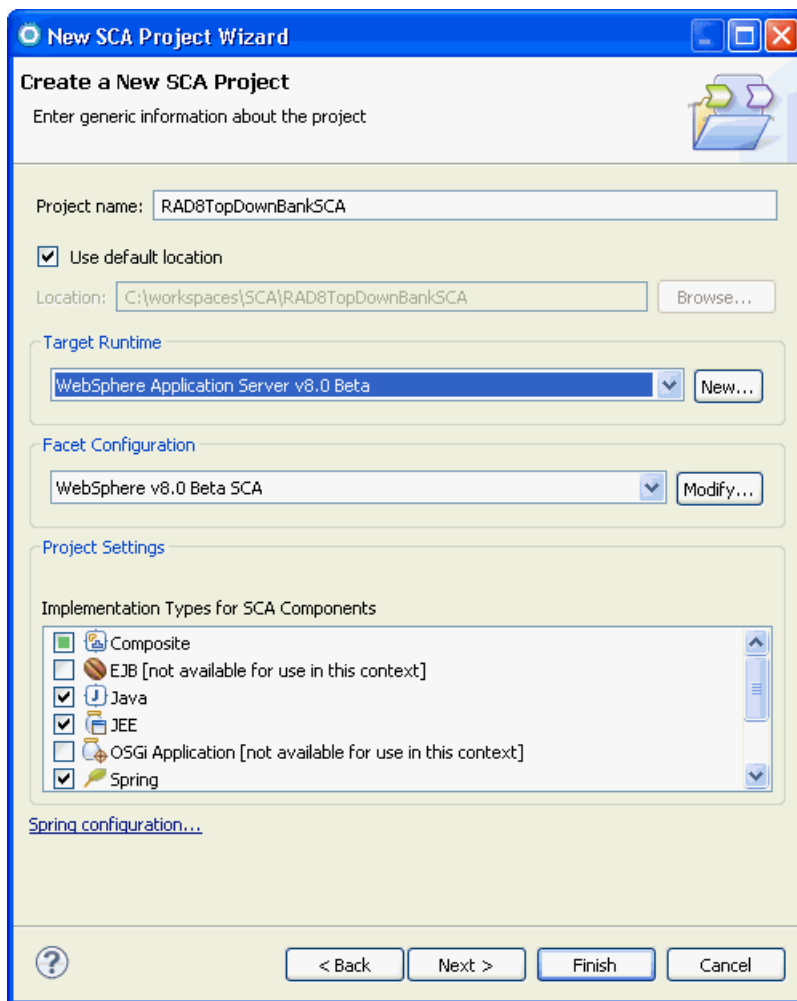


Figure 16-14 Configuration of a new SCA project

You can also augment an existing project with SCA features by using one of the following two ways:

- ▶ Use the Add SCA Support menu:
 - a. Right-click the existing project.
 - b. Select **Configure** → **Add SCA Support**.
- ▶ Add the SCA project facet (valid only for faceted projects, that is, not available for Java projects, for example):
 - a. Right-click the existing project.

- b. Select **Properties**.
- c. Select **Facets**.
- d. Select **Service Component Architecture (SCA)** and select **WebSphere 8.0 Beta SCA**.

16.3 Developing a Java component from a WSDL interface

In this section, we show how you can generate a component implemented in Java and expose a service based on an existing WSDL interface. This *top-down* scenario is the recommended approach to generate Java SCA components.

By contrast, the *bottom-up* scenario corresponds to starting from existing Java Implementations. This approach has less support from SCA tools than the top-down approach. For example, SCA tools do not drive the execution of `wsgen` to generate a WSDL file from a Java implementation class. If the user adopts the bottom-up approach, the user must run `wsgen` and then use the composite editor to create a component with the implementation class and generated WSDL interface.

In the top-down scenario, the product uses the Java API for XML Web Services (JAX-WS) and Java Architecture for XML Binding (JAXB) specifications to map the WSDL interface to Java types. This process is partly equivalent to using the WebSphere Application Server command `wsimport` to generate the Java representations of your business service interfaces and your business data, along with XML Schema Definition Language (XSD) schema definitions of your business data. The result is a Plain Old Java Object (POJO) implementation of the generated interface using the generated JAXB data types. You can recognize these generated files, because they have an initial comment of the following form:

```
// Generated By:JAX-WS RI IBM 2.1.6 in JDK 6 (JAXB RI IBM JAXB 2.1.10  
in JDK 6)
```

The generated annotated Java classes that correspond to your business data contain all the necessary information that the JAXB runtime environment requires to build and parse the XML for marshaling and unmarshaling. You do not need to write code to convert the data between the XML wire format and the Java application. When you develop an SCA service by starting with an existing WSDL file, the interface is considered a *remotable interface*. The remotable interface uses *pass-by-value semantics*, which implies that your data is copied.

Additionally, Rational Application Developer generates a skeleton implementation class annotated with `org.oesa.sca.annotations.Service` for you. This annotation defines the Java implementation as an SCA service implementation.

Avoid in-out and multiple out parameters: The product does not support using a WSDL file when the Java mapping requires *holder classes*. The product uses the JAX-WS specification to define the mapping between WSDL files and Java, including the mapping between a WSDL `portType` object and a Java interface. When you have WSDL `portType` objects with operations that use *in-out parameters* or operations that use *multiple output parameters*, the JAX-WS specification uses instances of the `javax.xml.ws.Holder` class in the mapping of the WSDL `portType` object to a Java interface. When using SCA, do not use a WSDL file when the Java mapping requires holder classes. Instead, use a WSDL file that does not map to holder classes.

Remotable interfaces: The product uses XML marshaling as defined by JAXB to marshal and unmarshal data across a remotable interface. If you start with a *remotable* Java interface for your implementation rather than starting with a WSDL `portType` interface (*bottom-up approach*), be careful when selecting the input and output Java data types and ensure that you understand which data is preserved across JAXB marshaling and unmarshalling.

If you are using a remotable interface, add the `@org.oesa.sca.annotations.Remotable` annotation to the Java interface.

The data marshaling and unmarshaling that is used to instantiate the copying of data over remotable interfaces is defined by the JAXB specification rather than by Java serialization or the `java.io.Serializable` or `java.io.Externalizable` interfaces. Because of this behavior, certain existing Java types are not suitable for use on remotable interfaces, because these types are not serialized using Java serialization. For data types that are not annotated, the class is introspected and its Java properties determine the data that is preserved in the copy. For data types that take advantage of JAXB annotations, you can customize the mapping of Java classes to XSD types and of Java instances to XML documents. Custom Java serialization routines, such as the `readObject()` or `writeObject()`, are not applicable in this scenario. The SCA runtime environment takes an XML-centric view of the business data and uses the JAXB standards to define the mappings between the Java programming model and the XML data format on the wire.

However, when authoring an implementation on a *local* interface, you can use any Java type, because local interfaces use pass-by-reference semantics, which implies that no data is copied.

Copy the wsdl file in 7835code\webservices\topdown\BankWS.wsdl into RAD8TopDownBankSCA\WSDL.

16.3.1 Creating a composite

To create a new composite (Figure 16-15 on page 901), follow these steps:

1. In the Enterprise Explorer, inside the project, right-click **SCA Content** → **Composites**.
2. Select **New** → **SCA Composite**.
3. In the New Composite Wizard window, select **Conventional Composite** (a distinguished application composite can be only added in an SCA enhanced EAR).
4. For Composite name, enter `ITS0BankComposite`.
5. For Target namespace, enter `http://com.ibm.itso.bank`.
6. Accept the automatically generated composite path and select **Finish**.

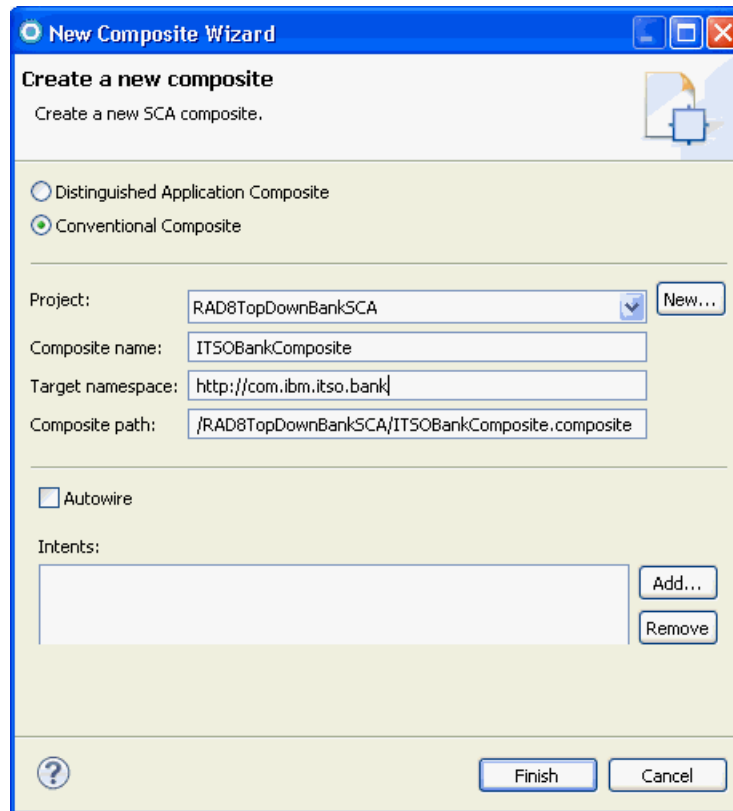


Figure 16-15 Create a new composite window

16.3.2 Creating a component

Create a new component (Figure 16-16 on page 902) with the following steps:

1. In the Enterprise Explorer, inside the project, right-click **SCA Content** → **Composites**.
2. Select **New** → **SCA Component**.
3. In the New Component Wizard window, for Composite, select **ITSOBankComposite**.
4. For Component Name, enter **ITSOBankComponent**.
5. For Interface Type, select **WSDL**.
6. Select **Reuse an existing service interface**.
7. For Interface Name, browse to the **BankWS** portType.
8. For Implementation Type, select **Java**.

9. Select **Create a new implementation** and select **Next**.

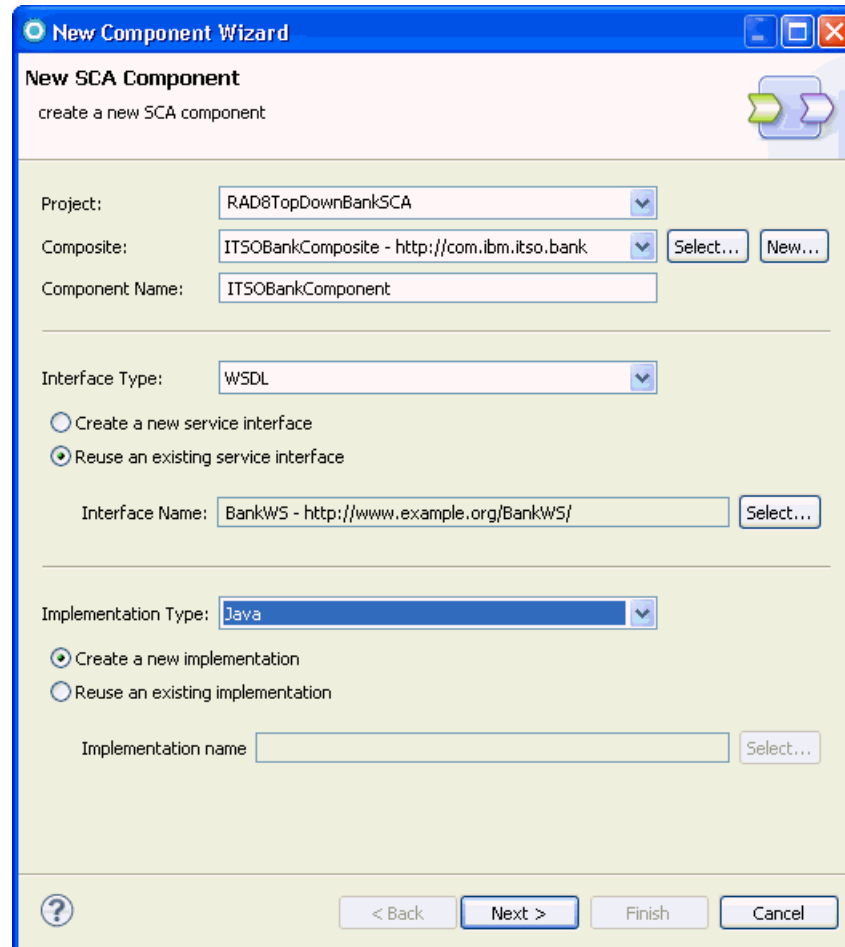


Figure 16-16 Create a new component window

10. In the Java Implementation Configuration window (Figure 16-17 on page 903), accept all defaults and click **Finish**.

The following classes are generated in the `org.example.bankws` package:

Account	Annotated with <code>@javax.xml.bind.annotation.XmlType</code>
BankWS	An interface, which is considered remotable, annotated with <code>@javax.jws.WebService</code>
BankWSImpl	Skeleton implementation class, annotated with <code>@org.oesa.sca.annotations.Service</code>

Customer	Annotated with <code>@javax.xml.bind.annotation.XmlType</code>
GetAccount	Annotated with <code>@javax.xml.bind.annotation.XmlType</code> and <code>@javax.xml.bind.annotation.XmlRootElement</code>
GetAccountResponse	Annotated with <code>@javax.xml.bind.annotation.XmlType</code> and <code>@javax.xml.bind.annotation.XmlRootElement</code>
GetCustomer	Annotated with <code>@javax.xml.bind.annotation.XmlType</code> and <code>@javax.xml.bind.annotation.XmlRootElement</code>
GetCustomerResponse	Annotated with <code>@javax.xml.bind.annotation.XmlType</code> and <code>@javax.xml.bind.annotation.XmlRootElement</code>
ObjectFactory	Annotated with <code>@javax.xml.bind.annotation.XmlRegistry</code>
package-info	Annotated with <code>@javax.xml.bind.annotation.XmlSchema</code>

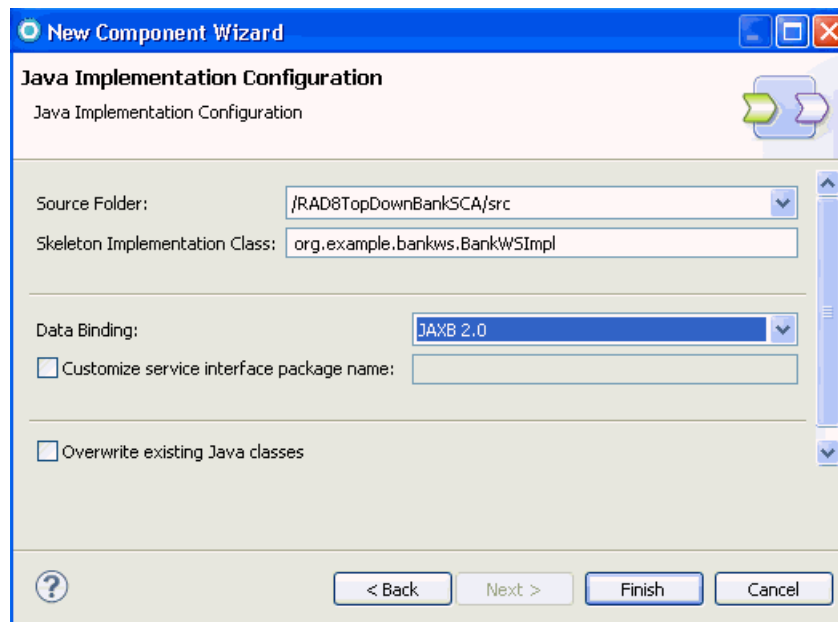


Figure 16-17 Java Implementation Configuration window

16.3.3 Implementing the Java component

We can now supply an implementation for the component, by implementing the methods of the generated class `org.example.bankws.BankWSImpl.java`, as shown in Example 16-1 on page 904.

```
package org.example.bankws;

import java.math.BigDecimal;

import org.osoa.sca.annotations.Service;

@Service (BankWS.class)
public class BankWSImpl implements BankWS {

    public Account getAccount(String accountId) {
        Account account = new Account();
        account.setId(accountId);
        account.setBalance(new BigDecimal(1000.00));
        return account;
    }

    public Customer getCustomer(String customerId) {
        Customer customer = new Customer();
        customer.setFirstName("Lara");
        customer.setLastName("Ziosi");
        customer.setTitle("Mrs");
        customer.setSsn("888-88-8888");
        return customer;
    }
}
```

Observe that the generated implementation class bears the `org.osoa.sca.annotations.Service` annotation, which takes as a parameter the interface class. Now inspect the interface `org.example.bankws.BankWS`. This interface is a JAX-WS annotated interface, annotated with `javax.jws.WebService`. All other generated files (Customer, Account, and so on) are mapped to Java from the WSDL using JAXB 2.0.

16.4 Creating a contribution to include the deployable composites

Next we create a new contribution to hold the composite:

1. In the Enterprise Explorer, inside the project, right-click **SCA Contents** → **Contributions**.
2. Select **New** → **SCA Contribution** (Figure 16-18 on page 905).

3. In the New Contribution Wizard window, select the previously created Composite: **ITSOBankComposite** and click **Finish**.

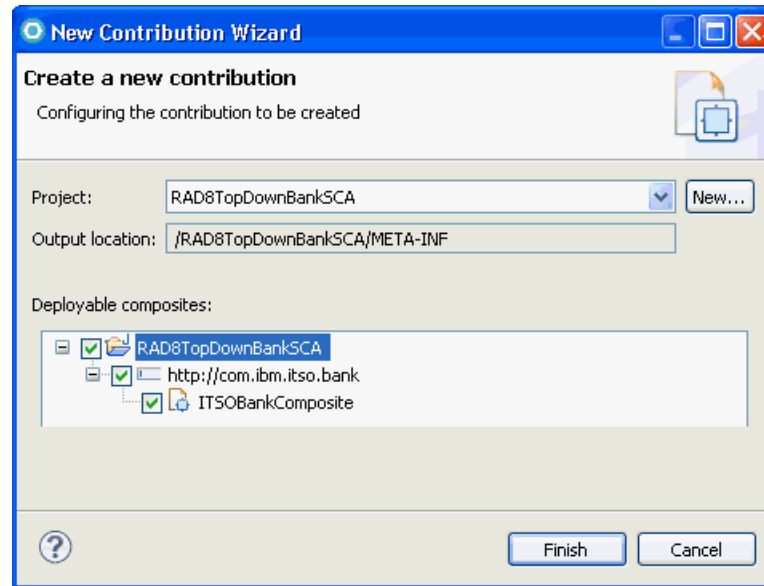


Figure 16-18 Creating a new contribution

The final configuration step consists of adding a binding to the component:

1. In the Enterprise Explorer, double-click **ITSOBankComponent**, which opens the **ITSOBankComposite.composite_diagram** in the editor pane. Complete these tasks:
 - a. Select the **Component Service** icon, which is shown in Figure 16-19 on page 906.
 - b. In the Properties view, make sure that you see **Component Service - BankWS**.
 - c. Select the **Bindings** drawer of the Properties view.
 - d. Select the **Bindings** node in the tree and select **Add**.
 - e. Select **Webservice** and click **OK**.

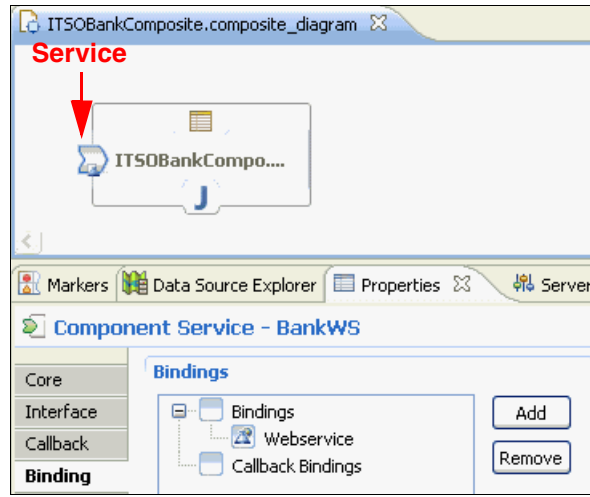


Figure 16-19 Add a Webservice binding to the service

We have completed the creation of the Java component from a WSDL file.

16.5 Deploying the contribution to WebSphere Application Server

We can now proceed to test the component on WebSphere Application Server V8 Beta. Even though contributions are separate artifacts from enterprise archives, Rational Application Developer lets you publish them in exactly the same way as EAR files.

On the server, contributions are managed as business-level applications. A *business-level application* is an administration model that provides the entire definition of an application as it makes sense to the business. It is a WebSphere configuration artifact, similar to a server or cluster, that is stored in the product configuration repository. A business-level application can contain artifacts, such as Java EE applications or modules, shared libraries, data files, SCA contributions, and other business-level applications. You might use a business-level application to group related artifacts or to add capability to an existing application. Follow these steps:

1. In the Servers view, right-click **WebSphere Application Server v8.0 Beta** and select **Add and Remove** (Figure 16-20 on page 907).
2. Add **RAD8TopDownBankSCA** and select **Finish**.

3. Wait for the server to republish (it moves to the state: Started, Synchronized). Depending on the publishing settings, you might need to right-click the server and select **Publish**.

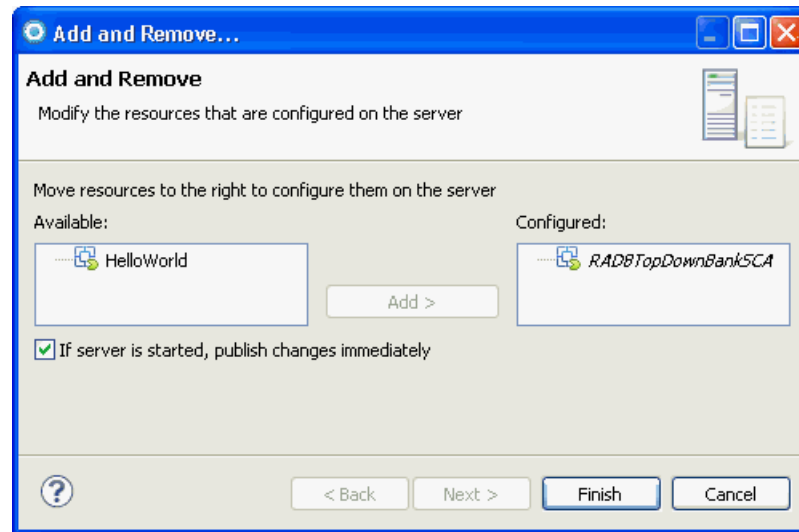


Figure 16-20 Deploying a contribution to WebSphere Application Server

4. Upon successful publication, you see entries similar to the following entries in the Console view:
CWSAM2001I: The ITSOBankComposite composite started successfully.
WSVR0191I: Composition unit WebSphere:cuname=ITSOBankComposite in
BLA WebSphere:blaname=RAD8TopDownBankSCA started.
CWWMH0196I: Business-level application
"WebSphere:blaname=RAD8TopDownBankSCA" was started successfully.

16.6 Testing the services provided by the SCA application

The WSDL file that we used did not contain any binding information for this specific test server. Before we can test the service, we must obtain the correct URL for launching a client. Follow these steps:

1. To obtain the deployed WSDL file for use in the Web Services Explorer or the Generic Service Client, right-click the server and select **Administration** → **Run Administrative Console**.

2. In the administrative console, select **Applications** → **Application Types** → **Business-level applications**.
3. Select **RAD8TopDownBankSCA** from the list of available resources.
4. Select **ITSOBankComposite** from the list of deployed assets.
5. Select **Export WSDL and XSD documents** (Figure 16-21) and save the .zip file to a temporary location that is external to the workspace.

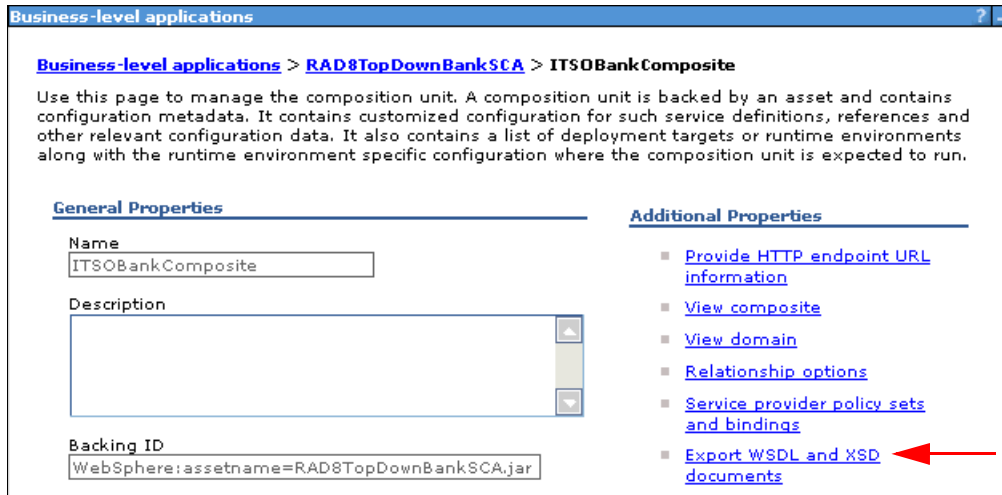


Figure 16-21 Export WSDL and XSD documents

6. After you extract the files (always outside of the workspace), you see `ITSOBankComponent_BankWS_wsdlgen.wsdl` and `WSDL\BankWS.wsdl`.
7. `ITSOBankComponent_BankWS_wsdlgen.wsdl` imports the original `BankWS.wsdl` and contains the actual binding (Example 16-2).

Example 16-2 Generated WSDL file with binding

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="ITSOBankComponent.BankWS"
targetNamespace="http://www.example.org/ITSOBankComponent/BankWS"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://www.example.org/ITSOBankComponent/BankWS"
xmlns:ns0="http://www.example.org/BankWS/"
xmlns:SOAP11="http://schemas.xmlsoap.org/wsdl/soap/">
  <wsdl:import namespace="http://www.example.org/BankWS/"
location="WSDL/BankWS.wsdl">
    </wsdl:import>
  <wsdl:binding name="BankWSBinding" type="ns0:BankWS">
    . . . . .
```



```


</wsdl:binding>
  <wsdl:service name="BankWSService">
    <wsdl:port name="BankWSPort" binding="tns:BankWSBinding">
      <SOAP11:address
location="http://localhost:9080/ITSOBankComponent/BankWS"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>

```

In general, you use this format of the URI to access the wsdl:

`http://<host>:<port>/ComponentName/ServiceName?wsdl`

You can now import the WSDL into the workspace for testing by using the Generic Service Client:

1. Launch the Generic Service Client by selecting this icon  in the toolbar of the Java EE perspective.
2. Select the icon to Add WSDL file (Figure 16-22).

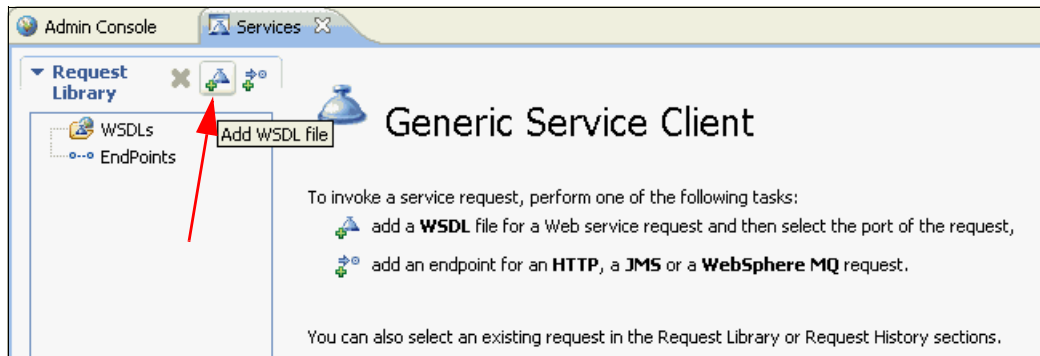


Figure 16-22 Importing a WSDL file with the Generic Service Client

3. Select **Import from File**.
4. In the WSDL field, browse for **ITSOBankComponent_BankWS_wsdlgen.wsdl** and click **OK**. Both `wsdl` files get copied into the root of the project called GSC Store (and the relative path of the `wsdl:import` statement gets adjusted accordingly).
5. The Request Library window now contains the entries that correspond to the binding, as shown in Figure 16-23 on page 910.

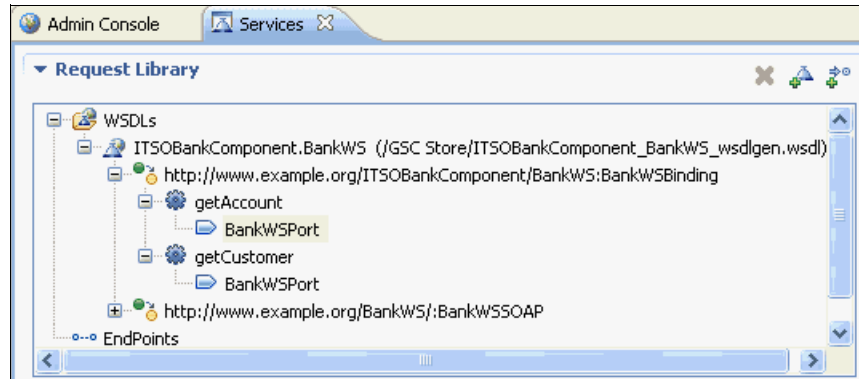


Figure 16-23 Request Library showing imported WSDL

6. You can now select **getAccount** → **BankWSPort** or **getCustomer** → **BankWSPort**.
7. In the right panel of the Generic Service Client, enter data and then select **Invoke**.
8. The results are displayed, as shown in Figure 16-24.

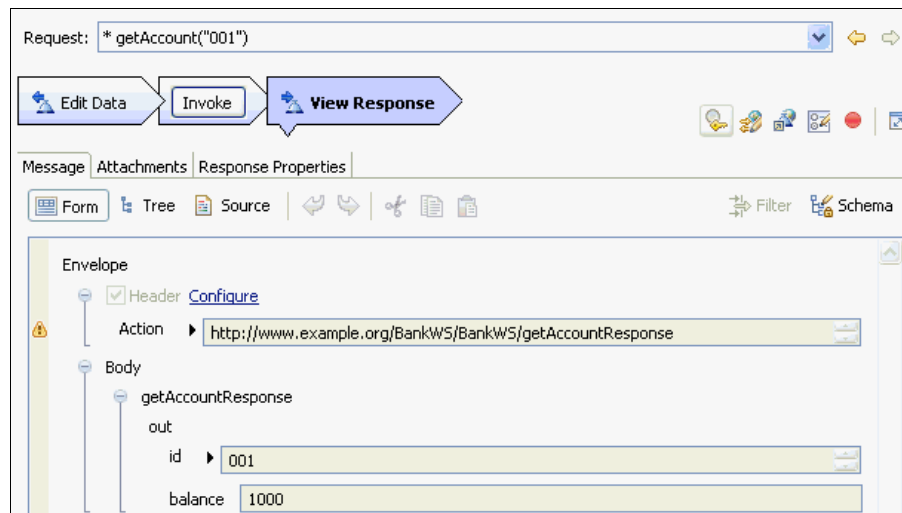


Figure 16-24 Invoking the service from the Generic Service Client

You can also test the generated SCA service with any other web service testing facility that the product provides. For instance, you can generate a web service client by right-clicking **ITSOBankComponent_BankWS_wsdgen.wsdl** and selecting **WebServices** → **Generate Client**. Select to test the Client and

generate it into new projects: RAD8TopDownBankClient and RAD8TopDownBankClientEAR. At the end of the wizard, select **JAX-WS JSPs** for the test facility. The result of running the JavaServer Pages (JSP) can be seen in Figure 16-25. The Servers tools view shows separate icons for deployed EAR files and SCA contributions.

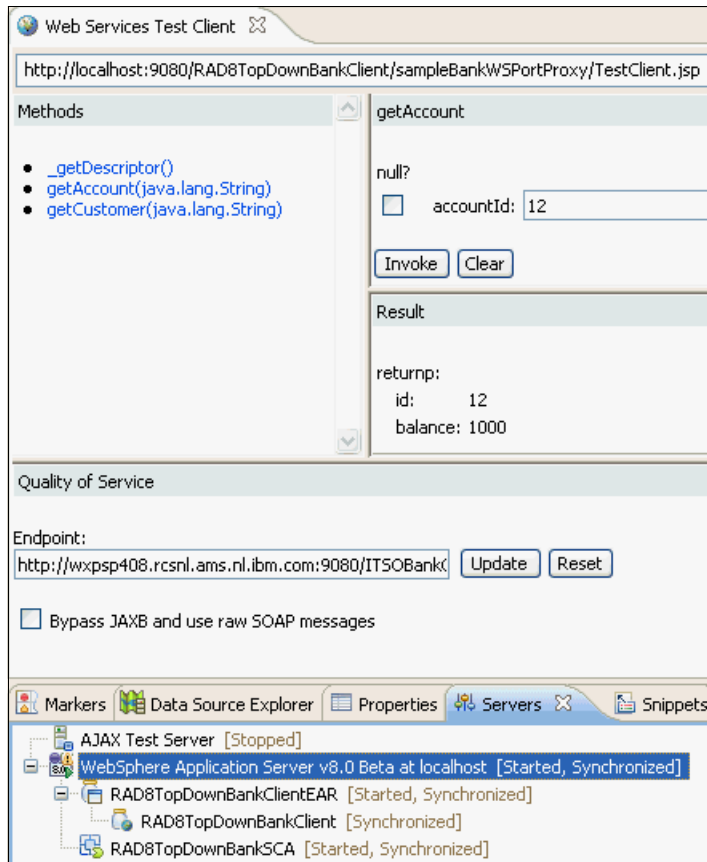


Figure 16-25 Testing a generated JAX-WS JSP client to access an SCA component

16.7 Wiring a component to a service on another component

We show a more complex example, which demonstrates how to perform the following activities:

- ▶ Creating a reference to an external Atom feed provider
- ▶ Exposing a service with an Atom binding

- ▶ Adding a contribution and testing the initial implementation
- ▶ Adding a second component to the composite
- ▶ Wiring the reference on one component to the service on the other component
- ▶ Using a property defined in a component and a composite

Atom is a protocol for publishing feeds. The SCA implementation in WebSphere Application Server and Rational Application Developer offers the Atom binding, which is based on the implementation that is provided by the Apache Tuscany project.

16.7.1 Creating a reference to an external Atom feed provider

We build this example around the following Atom feed for Rational Application Developer support documents:

http://www-947.ibm.com/systems/support/myfeed/xmlfeeder.wss?feeder.requestid=feeder.create_public_feed&feeder.feedtype=Atom&feeder.maxfeed=25&OC=SSRTLW&feeder.subdefkey=swgrat

We suggest that you try first to open this feed in your browser. If this feed is unavailable, reconfigure this sample to use another Atom feed available to you. A number of Atom feeds are offered at this website:

<http://www.ibm.com/developerworks/feeds/index.html>

To configure a reference to this external Atom feed provider, follow these steps:

1. Create a new SCA Project (Figure 16-26 on page 913):
 - a. Select **File** → **New** → **Other**.
2. In the New window, expand **Service Component Architecture**, select **SCA Project** and click **Next**. Complete these tasks:
 - a. Enter a Project name of RAD8AtomFeeds.
 - b. Set the Target Runtime to **WebSphere Application Server v8.0 Beta**.
 - c. For implementation types, leave only **Composite** and **Java** selected.
 - d. Select **Finish**.

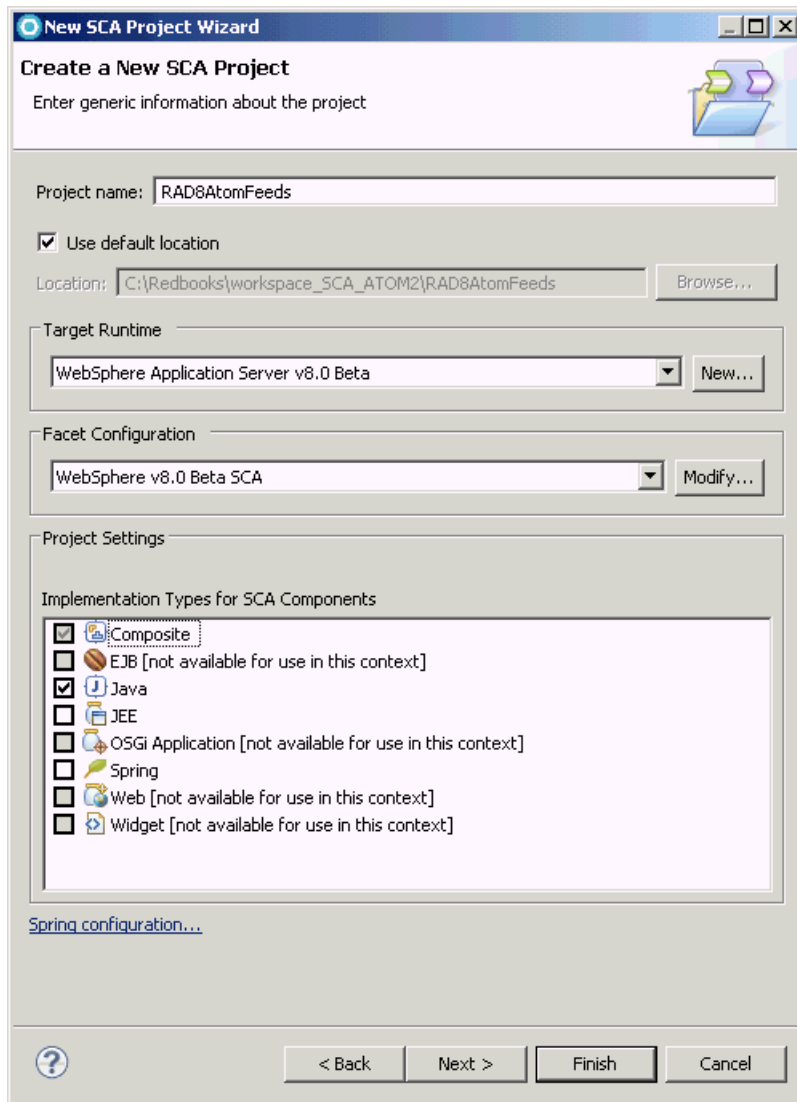


Figure 16-26 Create a new SCA Project window

3. In the RAD8AtomFeeds project, create a new Java package that is called `com.ibm.itso.support.feeds`.
4. Create the Java Interface `com.ibm.itso.support.feeds.Fetcher`, as shown in Example 16-3 on page 914. This interface must extend `org.apache.tuscany.sca.data.collection.Collection<String,Item>`, where the `String` parameter is used as the key of the item in the collection, and `org.apache.tuscany.sca.data.collection.Item` represents an individual

entry in the feed. This interface will be used to type both the reference with the Atom binding and the service with the Atom binding.

Example 16-3 Interface com.ibm.itso.support.feeds.Fetcher

```
package com.ibm.itso.support.feeds;  
  
import org.apache.tuscany.sca.data.collection.Collection;  
import org.apache.tuscany.sca.data.collection.Item;  
  
public interface Fetcher extends Collection<String, Item> {  
}
```

5. Create a new conventional composite:
 - a. In the Enterprise Explorer view, right-click **RAD8AtomFeeds** → **SCA Content** → **Composites** and select **New** → **SCA Composite** (Figure 16-27).

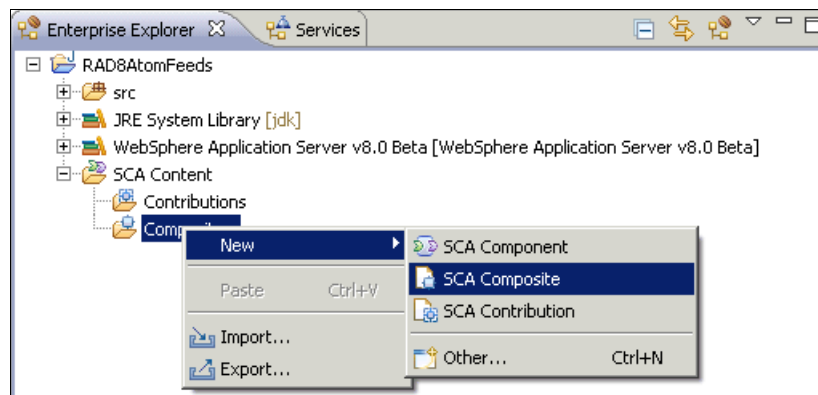


Figure 16-27 Create a new SCA Composite

6. In the New Composite Wizard window, complete these tasks:
 - a. Enter a Composite name of SupportFeeds.
 - b. Select **Conventional Composite**.
 - c. Enter a Target namespace of `http://itso.rad.feeds`.
 - d. Select **Finish** (Figure 16-28 on page 915).

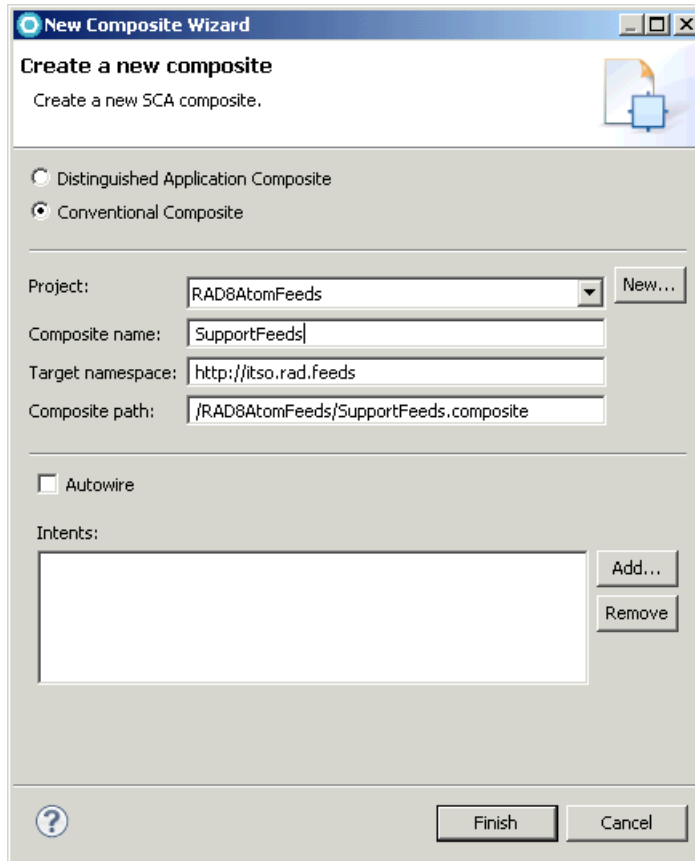


Figure 16-28 Creating a new conventional composite

7. Create a new component:
 - a. In the SupportFeeds.composite_diagram diagram, drag and drop a **Component** from the palette onto the canvas (Figure 16-29 on page 916).
 - b. Name the component AtomFeedFetcher.

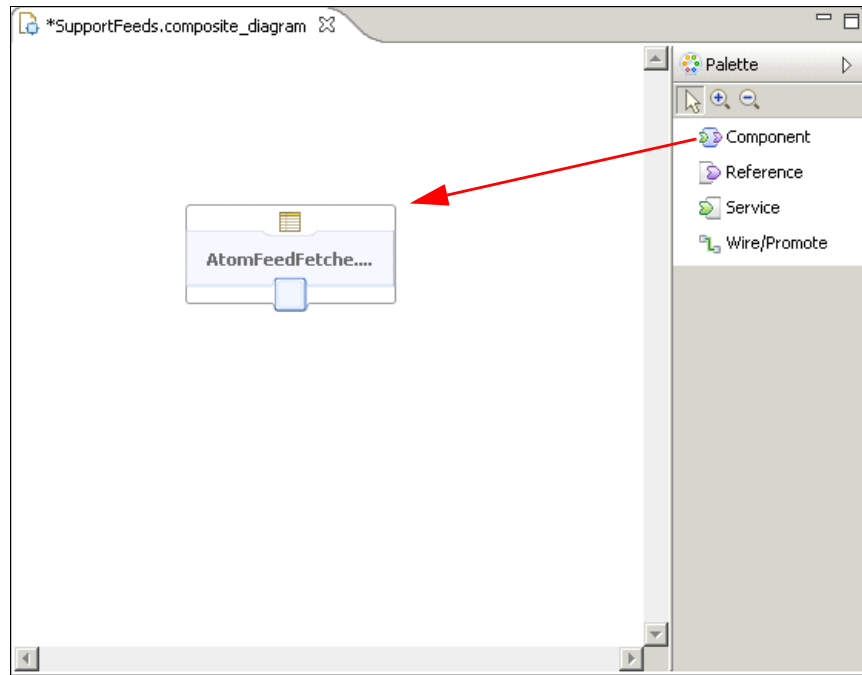


Figure 16-29 Creating a new component from the palette

8. Create a reference:
 - a. From the Palette, drag and drop a **Reference** onto the component.
 - b. Right-click the reference and select **Add Binding** → **Atom** (Figure 16-30 on page 917).

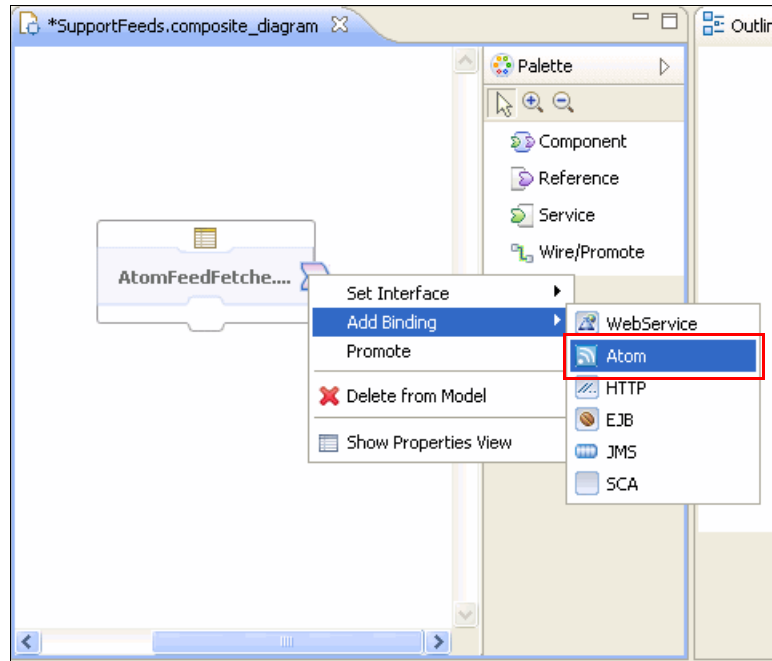


Figure 16-30 Adding a binding

- c. Right-click the reference and select **Show Properties View**. The Properties view appears in the Views section of the workbench.
9. In the **Core** tab, for Reference Name, enter `atomFeeds`. Remember this name, because you must reuse it in the implementation. You will inject this reference using the `@Reference` annotation.

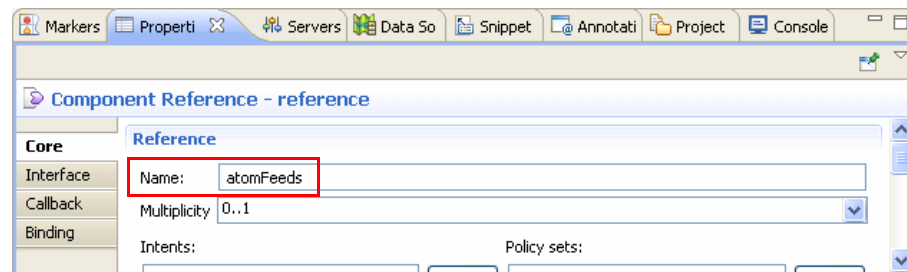


Figure 16-31 Adding the Name to the reference

10. In the Properties view that is related to the reference, click the **Interface** tab (Figure 16-32 on page 918). Complete these steps:
 - a. For Interface type, select **Java**.

- b. Browse for the interface named **com.ibm.itso.support.feeds.Fetcher**.

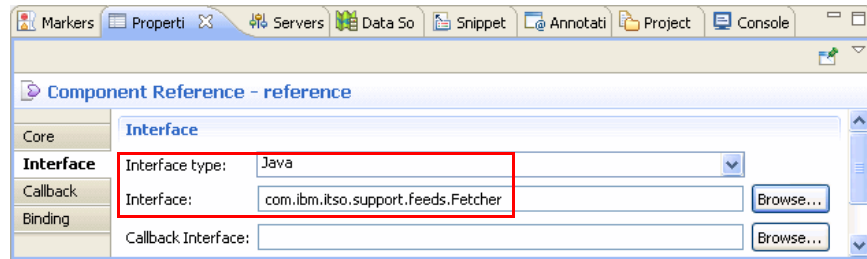


Figure 16-32 Adding the interface to the reference

11. In the Properties view, select the **Binding** tab (Figure 16-33), and in the URI field for the Atom Binding, enter this URI:

`http://www-947.ibm.com/systems/support/myfeed/xmlfeeder.wss?feeder.requireid=feeder.create_public_feed&feeder.feedtype=Atom&feeder.maxfeed=25&OC=SSRTLW&feeder.subdefkey=swgrat`

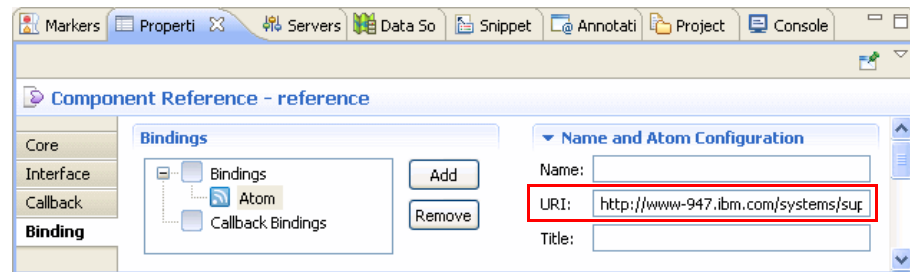


Figure 16-33 Enter the URI for the binding

12. Create the implementation class `com.ibm.itso.support.feeds.FetcherImpl` with the code in Example 16-4. The type of the field must be the interface (Fetcher), and the name of the field *must* be equal to the name of the reference (atomFeeds).

Example 16-4 Initial implementation of the component

```
package com.ibm.itso.support.feeds;

import org.osoa.sca.annotations.Reference;

public class FetcherImpl {
    public @Reference Fetcher atomFeeds;
}
```

13. Right-click the **AtomFeedFetcher** component and select **Set Implementation** → **Java**. Complete these steps:
 - a. Browse for the implementation class named **com.ibm.itso.support.feeds.FetcherImpl**.
 - b. The **J** icon appears on the component.
 - c. Control the settings by right-clicking the component. Select **Show Properties View** and look at the **Implementation** tab (Figure 16-34).

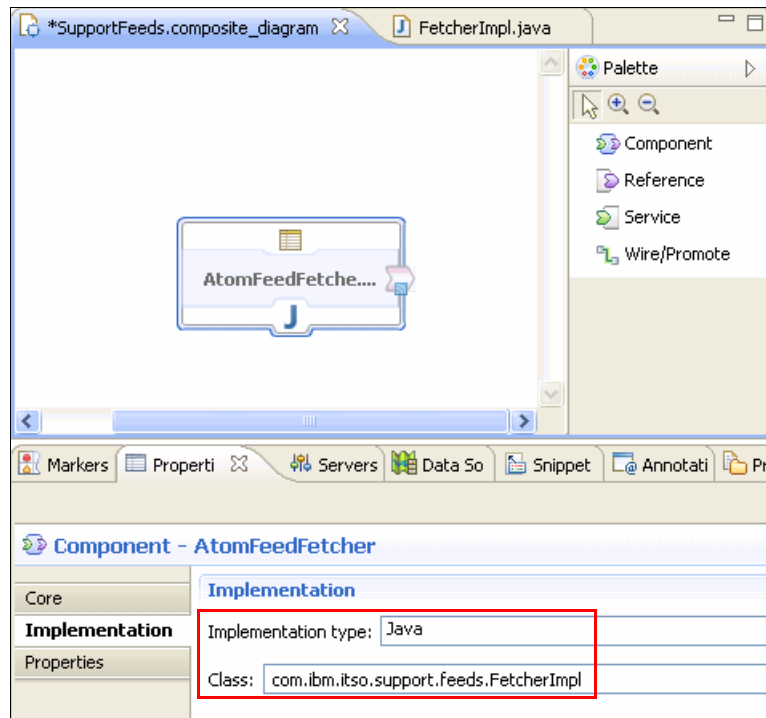


Figure 16-34 Adding a Java implementation to the component

Example 16-5 shows the contents of the `SupportFeeds.composite` file at this point.

Example 16-5 SupportFeeds.composite with Reference and Implementation

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.oso.org/xmlns/sca/1.0"
xmlns:tuscany="http://tuscany.apache.org/xmlns/sca/1.0"
autowire="false" name="SupportFeeds"
targetNamespace="http://itso.rad.feeds">
  <component name="AtomFeedFetcher">
```

```

    <implementation.java
class="com.ibm.itso.support.feeds.FetcherImpl"/>
    <reference name="atomFeeds">
        <interface.java interface="com.ibm.itso.support.feeds.Fetcher"/>
        <tuscany:binding.atom
uri="http://www-947.ibm.com/systems/support/myfeed/xmlfeeder.wss?feeder
.requid=feeder.create_public_feed&feeder.feedtype=Atom&feeder.m
axfeed=25&OC=SSRTLW&feeder.subdefkey=swgrat"/>
        </reference>
    </component>
</composite>

```

16.7.2 Exposing a service with an Atom binding

We add a service to the component that, in its first implementation, simply offers in output the same information that we receive from the Atom binding accessed via the reference. Because we want to expose this service via an Atom binding, the service must also be typed by an interface that extends `org.apache.tuscany.sca.data.collection.Collection<String,Item>`, so that it can reuse the same interface that we used for the `com.ibm.itso.support.feeds.Fetcher` reference.

Therefore, the implementation class `FetcherImpl` must implement `Fetcher`, and it must be annotated with `@Service(Fetcher.class)`. Of the various methods of the interface `Fetcher`, we are only interested in providing a non-default implementation for the method `getAll`, which returns all of the items that are supplied by the reference. We modify the implementation (Example 16-6).

Example 16-6 Adding @Service to the implementation class

```

package com.ibm.itso.support.feeds;

import org.apache.tuscany.sca.data.collection.Entry;
import org.apache.tuscany.sca.data.collection.Item;
import org.apache.tuscany.sca.data.collection.NotFoundException;
import org.osoa.sca.annotations.Reference;
import org.osoa.sca.annotations.Service;

@Service(Fetcher.class)
public class FetcherImpl implements Fetcher{
    public @Reference Fetcher atomFeeds;

    @Override
    public Entry<String, Item>[] getAll() {

```

```

        return atomFeeds.getAll();
    }

    @Override
    public void delete(String arg0) throws NotFoundException {
    }

    @Override
    public Item get(String arg0) throws NotFoundException {
        return null;
    }

    @Override
    public String post(String arg0, Item arg1) {
        return null;
    }

    @Override
    public void put(String arg0, Item arg1) throws NotFoundException {
    }

    @Override
    public Entry<String, Item>[] query(String arg0) {
        return null;
    }
}

```

Rational Application Developer offers a quick way to populate the component from the data that is contained in the implementation class. Follow the steps:

1. Right-click the component **AtomFeedFetcher** on the diagram and select **Refresh from Implementation** (Figure 16-35 on page 922).

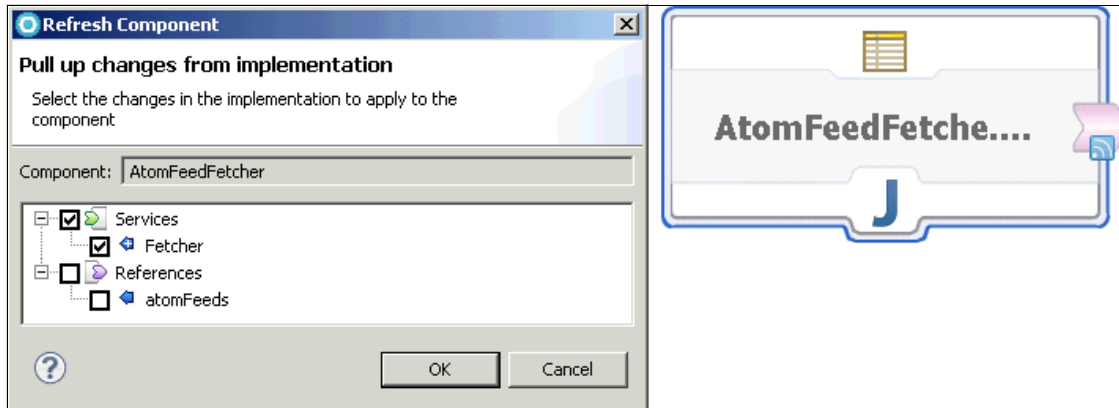


Figure 16-35 Refresh from implementation

2. Rational Application Developer detects that we have added an @Service annotation and proposes to add it to the component. It also detects that the reference atomFeeds already exists in the component, and you might choose to update it by selecting the check box in front of it. We do not need to update it. Click **OK**.
3. The Service icon (arrow) gets added to the component on the diagram.
4. Right-click the **Service** icon (green arrow) and select **Show Properties View**. You see that Service Name in the Core tab is Fetcher. The Service Name must match the name of the interface, which is the type of the service (Figure 16-36).

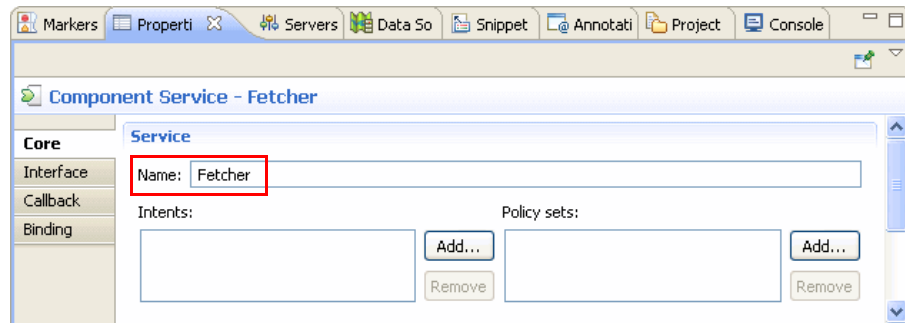


Figure 16-36 The service and its properties

5. Switch to the **Interface** tab and verify that the Interface is of type **Java** and that it has the value `com.ibm.itso.support.feeds.Fetcher`.
6. Right-click the **Service** icon (green arrow) representing the service on the diagram and select **Add Binding** → **Atom**.

7. Right-click the **Service** icon (green arrow) representing the service on the diagram and select **Show Properties View**.
8. Select the **Binding** tab. For the URI, enter /supportFeeds.

Example 16-7 shows the contents of the SupportFeeds.composite file at this point.

Example 16-7 Adding the service to the component

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.oso.org/xmlns/sca/1.0"
xmlns:ns1="http://tuscany.apache.org/xmlns/sca/1.0"
xmlns:sp_0="http://tuscany.apache.org/xmlns/sca/1.0" autowire="false"
name="SupportFeeds" targetNamespace="http://itso.rad.feeds">
  <component name="AtomFeedFetcher">
    <implementation.java
class="com.ibm.itso.support.feeds.FetcherImpl"/>
    <service name="Fetcher">
      <interface.java interface="com.ibm.itso.support.feeds.Fetcher"/>
      <ns1:binding.atom uri="/supportFeeds"/>
    </service>
    <reference name="atomFeeds">
      <interface.java interface="com.ibm.itso.support.feeds.Fetcher"/>
      <ns1:binding.atom
uri="http://www-947.ibm.com/systems/support/myfeed/xmlfeeder.wss?feeder
.requid=feeder.create_public_feed&feeder.feedytype=Atom&feeder.m
axfeed=25&OC=SSRTLW&feeder.subdefkey=swgrat"/>
    </reference>
  </component>
</composite>
```

You might see the following validation error:

“The atom binding cannot be used in this service because it is incompatible with the specified service interface.”

You can relegate this error to a warning by performing these steps:

1. Select **Window** → **Preferences** → **Service Component Architecture** → **Validation Rules**.
2. Click the **WebSphere** tab and expand **Feature Pack for SCA 1.0.1** → **Atom Binding**.
3. Set the Incompatible use of atom bindings to **Warning** (Figure 16-37 on page 924) and click **OK**.

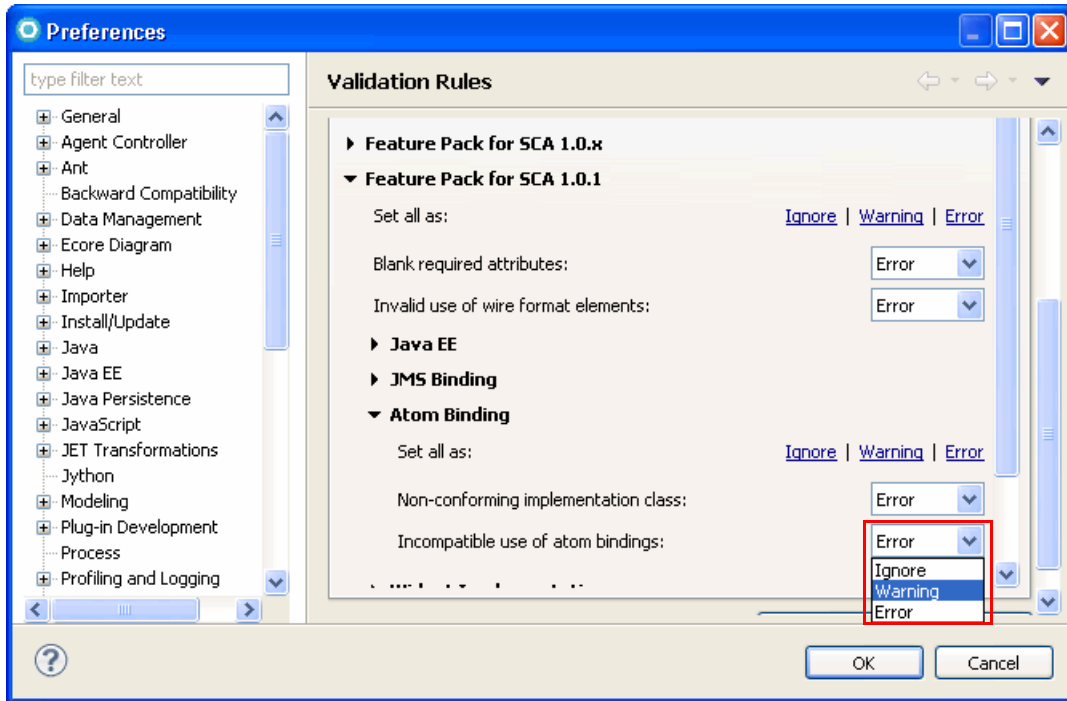


Figure 16-37 Setting the incompatible use of atom bindings to Warning

16.7.3 Adding a contribution and testing the initial implementation

In order to deploy to the server to test this initial implementation, we need a new contribution. Follow these steps:

1. In the Enterprise Explorer, right-click **Contributions** and select **New** → **SCA Contribution**.
2. In the New Contribution Wizard: Create a new contribution window, for Deployable composites, select **SupportFeeds** (Figure 16-38 on page 925) and click **Finish**.

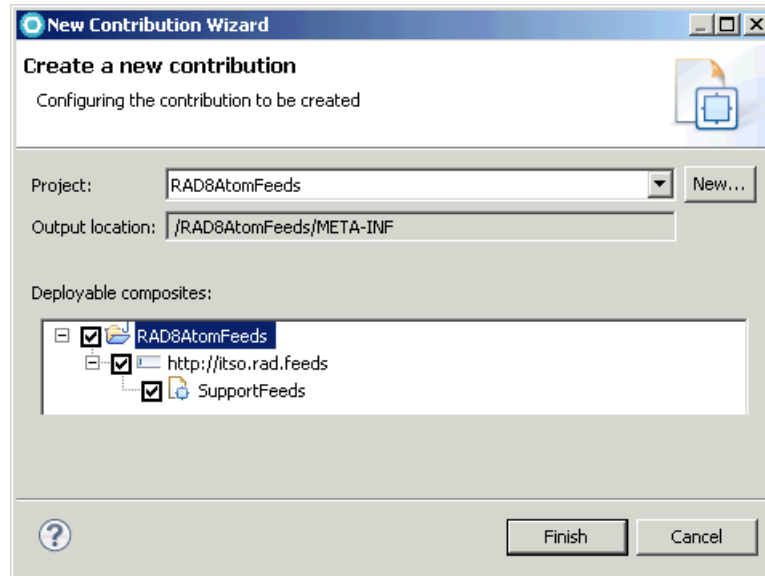


Figure 16-38 Creating a new contribution to test the new service

3. Deploy **RAD8AtomFeeds** to WebSphere Application Server V8.0 Beta.
4. You can now test the offered service from a web browser by entering the URL:
`http://hostname:port/supportFeeds`

Typically, the *hostname* is `localhost` and the *port* has the value `9080`, as seen in Figure 16-39 on page 926.

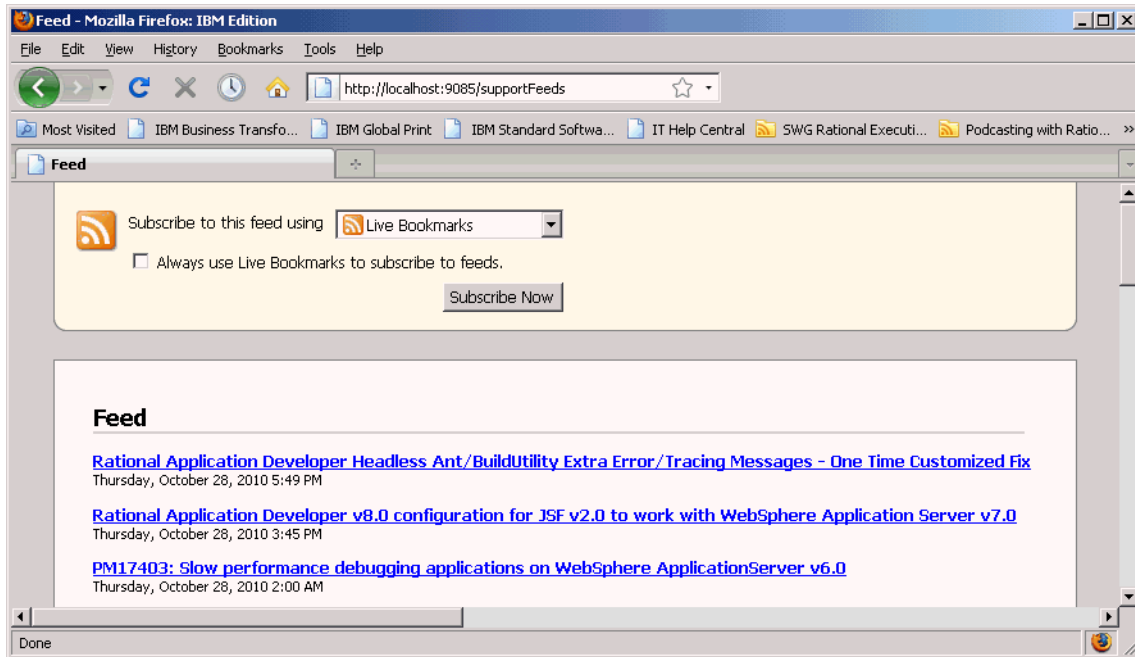


Figure 16-39 Testing the service with Atom binding in the browser

16.7.4 Adding a second component to the composite

Now we can add a second component to the composite. This second component provides a filter function to retrieve only the feeds that contain a keyword in the Title or in the Contents. Follow these steps:

1. The service offered by this new component is typed by the interface `com.ibm.itso.support.feeds.Filter`, as shown in Example 16-8.

Example 16-8 Interface Filter

```
package com.ibm.itso.support.feeds;

import org.apache.tuscany.sca.data.collection.Entry;
import org.apache.tuscany.sca.data.collection.Item;

public interface Filter {

    public Entry<String,Item>[]
    filterByPropertyValue(Entry<String,Item>[] entries);
}
```

2. Example 16-9 shows the implementation of the service, which is the `com.ibm.itso.support.feeds.FilterImpl` class.

Example 16-9 Class FilterImpl

```
package com.ibm.itso.support.feeds;

import java.util.ArrayList;
import java.util.List;

import org.apache.tuscany.sca.data.collection.Entry;
import org.apache.tuscany.sca.data.collection.Item;
import org.osoa.sca.annotations.Property;
import org.osoa.sca.annotations.Service;

@Service(Filter.class)
public class FilterImpl implements Filter {
    @Property
    public String componentFilter;

    @Override
    public Entry<String, Item>[] filterByPropertyValue(Entry<String,
Item>[] entries) {
        //If we have no filter, return the whole list
        if (componentFilter==null || componentFilter.length()==0) return
entries;
        //Create a local List
        List<Entry<String, Item>> filteredEntries = new
ArrayList<Entry<String, Item>>();

        //Convert the String to lower case and then to CharSequence for
use in String.contains
        CharSequence filterChars =
componentFilter.toLowerCase().subSequence(0, componentFilter.length());
        System.out.println(FilterImpl.class.getName()+" : value of
componentFilter Property: "+filterChars);

        if (entries != null) {
            try {
                for (Entry<String, Item> entry : entries) {
                    if (entry != null){
                        Item item = entry.getData();
                        if (item != null) {
                            String title = item.getTitle();

```

```

        //if filter is found in lowered case title, add entry
to the list
        if (title != null &&
title.toLowerCase().contains(filterChars)) {
            filteredEntries.add(entry);
            continue;//if already found, move on
        }
        //if filter is found in lowered case contents, add
entry to the list
        String contents = item.getContents();
        if (contents != null
&&contents.toLowerCase().contains(filterChars)) {
            filteredEntries.add(entry);
        }
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}

return filteredEntries.toArray(new
Entry[filteredEntries.size()]);

}
}

```

3. Create a new component called `AtomContentFilter`.
4. Right-click the component and choose **Set implementation** → **Java**.
5. Browse to the `com.ibm.itso.support.feeds.FilterImpl` class.
6. Right-click the **AtomContentFilter** component and select **Refresh from implementation** (Figure 16-40 on page 929), which shows that the product adds a service called `Filter` and a property called `componentFilter`. Click **OK** to accept these changes. We explain the role of properties in 16.7.6, “Using a property defined in a component and a composite” on page 932.

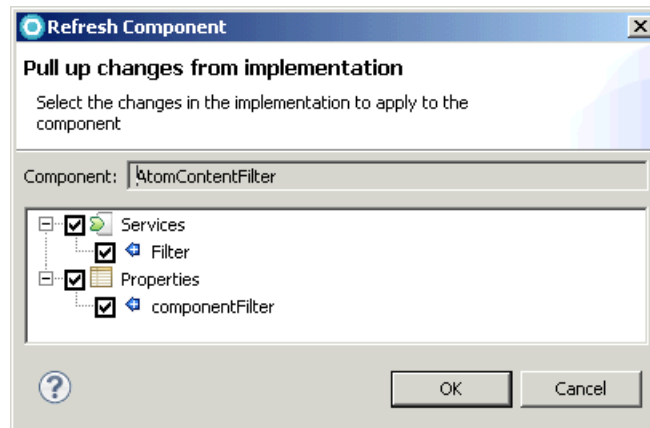


Figure 16-40 Refreshing from implementation

Figure 16-41 shows the aspect of the component now. Note the Service (green arrow) icon and the Property icon.



Figure 16-41 Component with Service and Property icons

16.7.5 Wiring the reference on one component to the service on the other component

Before we can use the Filter service from the AtomFeedFetcher component, we must add a reference to AtomFeedFetcher, typed by the Filter interface. We must add an @Reference annotation to FetcherImpl to receive the injected Filter object at run time:

1. Add a public field of type Filter and name atomFilter with the @Reference annotation to the com.ibm.itso.support.feeds.fetcherImpl class (Example 16-10). Use the atomFilter reference to invoke the filterByPropertyValue method of the FilterImpl class in the implementation of the getAll method.

Example 16-10 Adding a public field and name with an annotation to the class

```
package com.ibm.itso.support.feeds;

import org.apache.tuscany.sca.data.collection.Entry;
import org.apache.tuscany.sca.data.collection.Item;
import org.apache.tuscany.sca.data.collection.NotFoundException;
import org.osoa.sca.annotations.Reference;
import org.osoa.sca.annotations.Service;

@Service(Fetcher.class)
public class FetcherImpl implements Fetcher{
    public @Reference Fetcher atomFeeds;

    public @Reference Filter atomFilter;

    @Override
    public Entry<String, Item>[] getAll() {
        //return atomFeeds.getAll();
        Entry<String, Item>[]
filteredFeeds=atomFilter.filterByPropertyValue(atomFeeds.getAll());
        return filteredFeeds;

    }
    ....
}
```

2. Right-click **AtomFeedFetcher** on the diagram. Select **Refresh from Implementation** (Figure 16-42). The reference `atomFilter` is added.

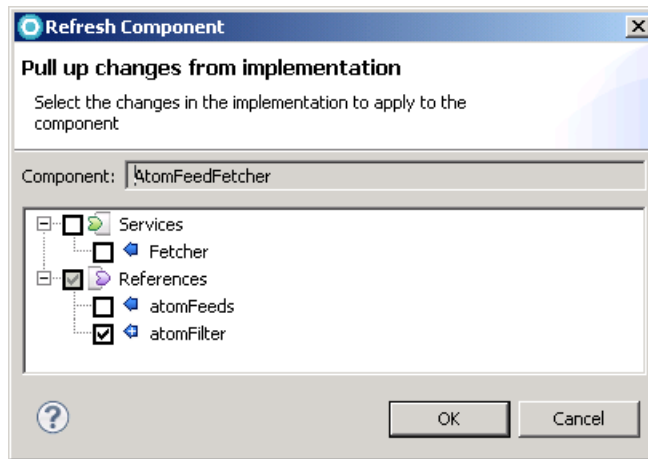


Figure 16-42 Refreshing the reference from the implementation

- From the Palette, select the **Wire/Promote** tool, select the atomFilter reference on the AtomFeedFetcher component and draw a line toward the Filter service on the AtomContentFilter component (Figure 16-43). The reference and service that you have connected with the wire appear to have no binding; therefore, the default *SCA Binding* is used.

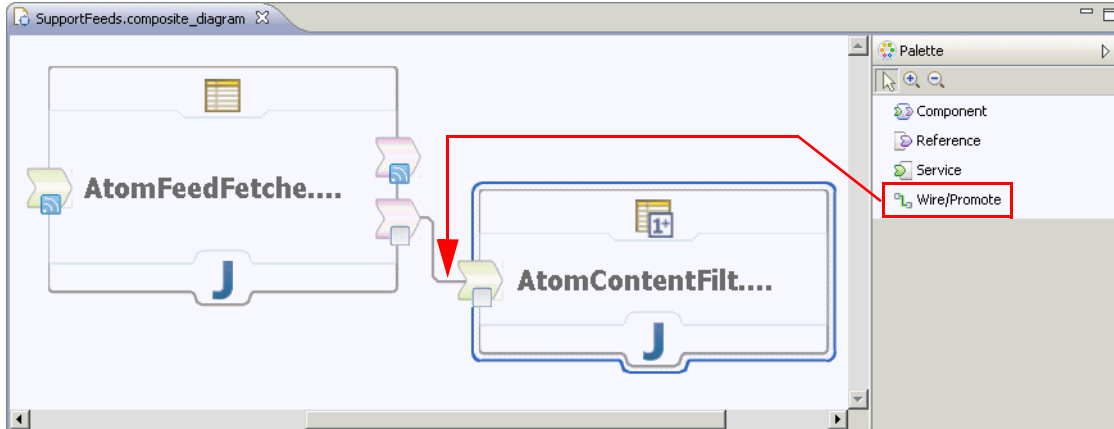


Figure 16-43 Add a wire between a service and a reference

The addition of the wire actually sets the target attribute of the atomFilter reference, as shown in Example 16-11.

Example 16-11 Additional reference

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<composite xmlns="http://www.oxa.org/xmlns/sca/1.0"
xmlns:ns1="http://tuscany.apache.org/xmlns/sca/1.0"
xmlns:sp_0="http://tuscany.apache.org/xmlns/sca/1.0"
xmlns:sp_1="http://tuscany.apache.org/xmlns/sca/1.0" autowire="false"
name="SupportFeeds" targetNamespace="http://itso.rad.feeds">
  <component name="AtomFeedFetcher">
    <implementation.java
class="com.ibm.itso.support.feeds.FetcherImpl"/>
    <service name="Fetcher">
      <ns1:binding.atom uri="/supportFeeds"/>
    </service>
    <reference name="atomFeeds">
      <interface.java interface="com.ibm.itso.support.feeds.Fetcher"/>
      <ns1:binding.atom
uri="http://www-947.ibm.com/systems/support/myfeed/xmlfeeder.wss?feeder
.requid=feeder.create_public_feed&feeder.feedtype=Atom&feeder.m
axfeed=25&OC=SSRTLW&feeder.subdefkey=swgrat"/>
      </reference>
      <reference name="atomFilter" target="AtomContentFilter/Filter">
        <interface.java interface="com.ibm.itso.support.feeds.Filter"/>
      </reference>
    </component>
    <component name="AtomContentFilter">
      <implementation.java
class="com.ibm.itso.support.feeds.FilterImpl"/>
      <service name="Filter">
        <interface.java interface="com.ibm.itso.support.feeds.Filter"/>
      </service>
      <property many="false" name="componentFilter" mustSupply="true"/>
    </component>
  </composite>

```

16.7.6 Using a property defined in a component and a composite

So far, we have introduced a property called `componentFilter` on the component `AtomContentFilter`. Properties can also be defined on the composites, and they can be reused by the components included therein. The following procedure declares the property and its default value in the composite and reuses it in the component:

1. Right-click the white space on the canvas of `SupportFeed.composite_diagram`.
2. Select **Show Properties View**. In the Property window, select **Property**.

3. Enter the property name, type, and default value, as shown in Figure 16-44 on page 933:
 - a. Select **Add**.
 - b. In Property Name field, enter `filter`.
 - c. Select **Must Supply**.
 - d. For Reference kind, select **type**.
 - e. For Type, select **xsd:string**.
 - f. For Simple Value, enter `debug` (or another keyword that you expect to be part of the returned feed).

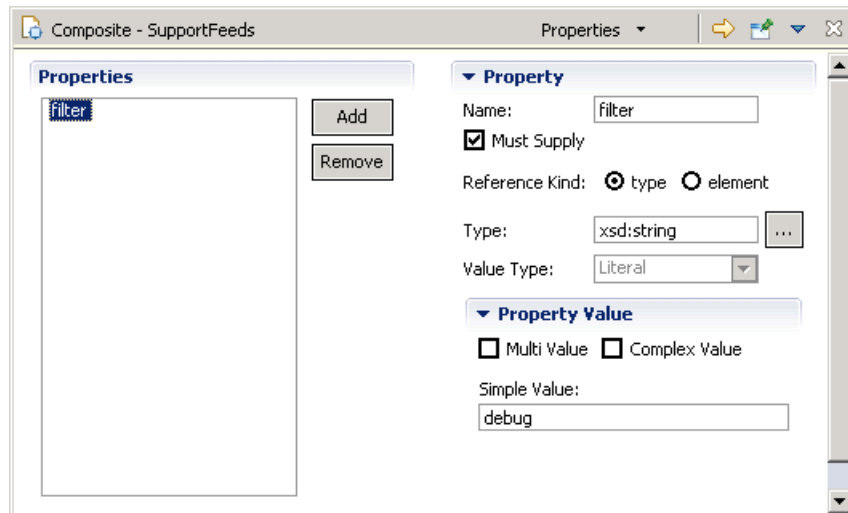


Figure 16-44 Setting the properties of the property filter defined on the composite

To reference this property in the component, follow these steps:

1. Right-click the `AtomContentFilter` component.
2. Select **Show Properties View**.
3. Select the **Properties** tab.
4. Select **componentFilter**.
5. Set the type to **xsd:string**.
6. Set the Value Type to **Source XPath**.
7. In Source, enter `$filter`, which is a special syntax that references the property value of the `filter` property defined in the enclosing composite.
8. Save the composite.

The SupportFeeds.composite file now contains both properties (Example 16-12).

Example 16-12 Final contents of SupportFeeds.composite file

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osea.org/xmlns/sca/1.0"
xmlns:ns1="http://tuscany.apache.org/xmlns/sca/1.0"
xmlns:sp_0="http://tuscany.apache.org/xmlns/sca/1.0"
xmlns:sp_1="http://tuscany.apache.org/xmlns/sca/1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" autowire="false"
name="SupportFeeds" targetNamespace="http://itso.rad.feeds">
  <component name="AtomFeedFetcher">
    <implementation.java
class="com.ibm.itso.support.feeds.FetcherImpl"/>
    <service name="Fetcher">
      <ns1:binding.atom uri="/supportFeeds"/>
    </service>
    <reference name="atomFeeds">
      <interface.java interface="com.ibm.itso.support.feeds.Fetcher"/>
      <ns1:binding.atom
uri="http://www-947.ibm.com/systems/support/myfeed/xmlfeeder.wss?feeder
.requid=feeder.create_public_feed&feeder.feedtype=Atom&feeder.m
axfeed=25&OC=SSRTLW&feeder.subdefkey=swgrat"/>
    </reference>
    <reference name="atomFilter" target="AtomContentFilter/Filter">
      <interface.java interface="com.ibm.itso.support.feeds.Filter"/>
    </reference>
  </component>
  <component name="AtomContentFilter">
    <implementation.java
class="com.ibm.itso.support.feeds.FilterImpl"/>
    <service name="Filter">
      <interface.java interface="com.ibm.itso.support.feeds.Filter"/>
    </service>
    <property name="componentFilter" source="$filter" type="xsd:string"
mustSupply="true"/>
  </component>
  <property mustSupply="true" name="filter"
type="xsd:string">debug</property>
</composite>
```

16.7.7 Testing the implementation by exporting the contribution

The implementation is now complete. You can test it by deploying RAD8AtomFeed to the server. We show how the contribution can be exported from Rational Application Developer and deployed using the administrative console by using the information that is available at this website:

http://www14.software.ibm.com/webapp/wsbroker/redirect?version=matt&product=was-base-dist&topic=trun_app_bla

Perform these steps:

1. Remove the application from the server using Add/Remove projects if it was previously deployed to avoid any possible naming conflicts.
2. Select **File** → **Export**.
3. Select **Service Component Architecture** → **SCA Archive File**. On Figure 16-45, follow these steps:
 - a. Select **Export Contributions**.
 - b. Select **RAD8AtomFeeds**.
 - c. Select a location for the archive file (select a .jar file).
 - d. Select **Export Composite diagram files**.
 - e. Select **Finish**.

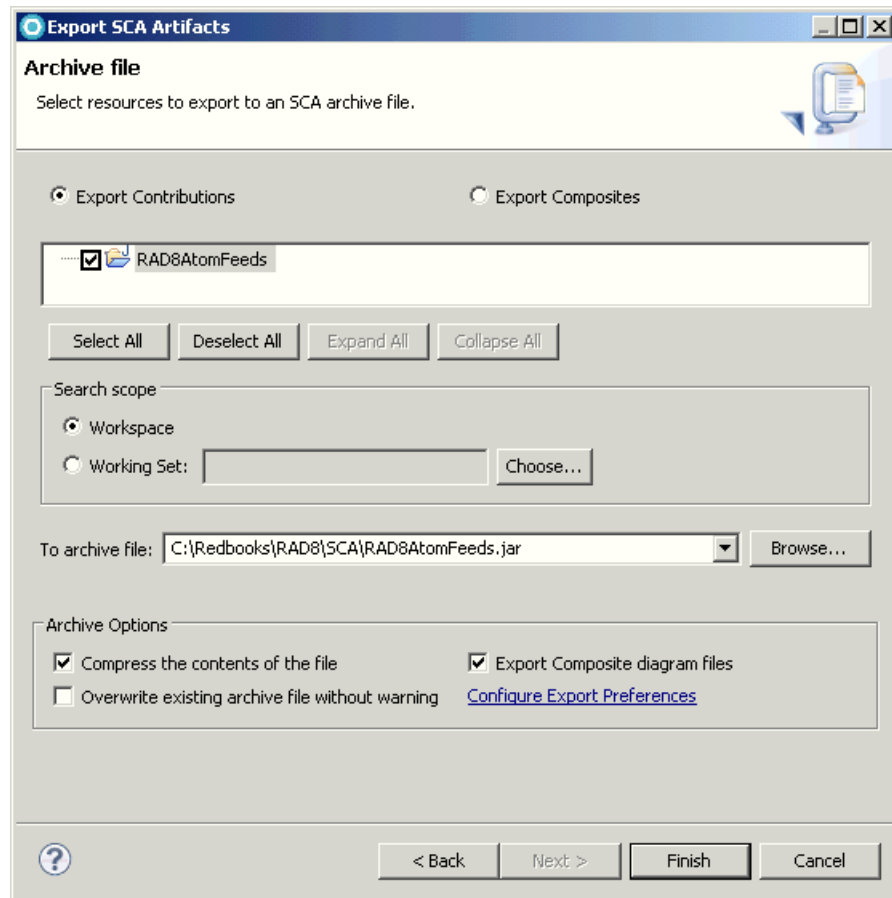


Figure 16-45 Export contribution from Rational Application Developer

4. Right-click the server in the Servers view and select **Administration** → **Run Administrative Console**.
5. Select **Applications** → **New Application** → **New Asset**.
6. Select **Local File System** and browse for the jar file that you exported from Rational Application Developer.
7. Select **Next** until the end of the wizard, accepting all default options.
8. Select **Save**.
9. Select **Applications** → **New Application** → **New Business-Level Application**.
10. Enter a name for the application.
11. Under Deployed Assets, select **Add** → **Add Asset**.

12. Select **Next** until the end of the wizard, accepting all default options.
13. Select **Save**.
14. You are now ready to test the application in the web browser, where you see only the filtered results, as shown in Figure 16-46.

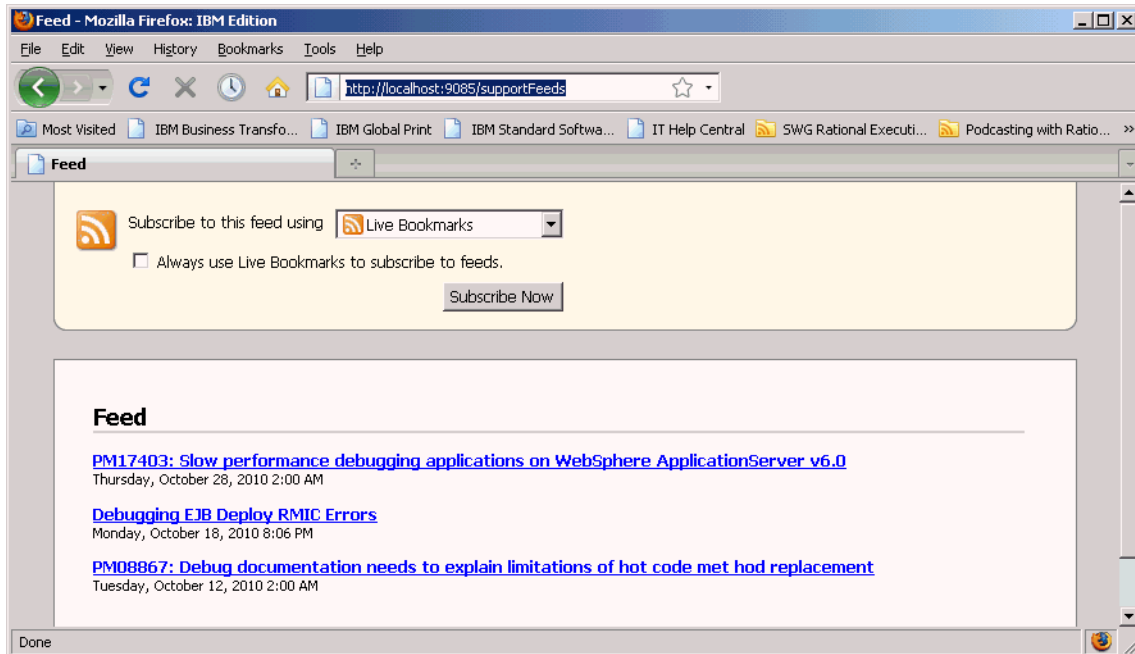


Figure 16-46 Browser showing the output filtered by keyword “debug”

16.8 Reusing an existing Java EE application to create a component

In this section, we show you how to reuse an existing Java EE application to implement an SCA component, without enhancing it. We also show how to create and use a web application, enhancing it to make use of SCA references. This section is conceptually similar to the tutorial that you access by selecting **Help** → **Help Contents** → **Tutorial** → **SCA** → **SCA Java EE Tutorial**.

We use these major steps:

1. Import the projects that are contained in the 7835code\sca\RAD8SimpleBankStart.zip folder. The projects are RAD8SimpleBankEJB and RAD8SimpleBankEAR.

2. Creating a new SCA Enhanced EAR file to hold the web project:
 - a. The EAR will be called RAD8SCAWebEAR and the web project will be called RAD8SCAWeb. The web project will contain a servlet and a copy of the EJB remote interface (so that the servlet compiles).
 - b. Furthermore, this EAR file will be SCA enhanced, so that the servlet contained in the web project can reference the remote interface of the EJB using the @Reference SCA annotation.
 - c. You will add a distinguished application composite (you can only create a distinguished composite in an EAR).
 - d. In the distinguished application composite, define an SCA component and set its implementation to reuse the existing web project.
 - e. Set an SCA reference on the component.
 - f. The SCA reference needs to be promoted so that it can be used outside of the composite where it was declared.
3. Creating a new SCA project with a contribution:
 - a. Add a conventional composite, because this project is a regular SCA project, not an EAR project.
 - b. Add two components, implemented by the non-SCA enhanced EAR RAD8SimpleBankEAR and by the SCA enhanced EAR RAD8SCAWebEAR.
 - c. Add a contribution so that this project can be deployed to the server.
 - d. Copy the EJB remote interface in this project as well, so that the reference and service interfaces can be resolved at run time.

16.8.1 Explore the existing EAR

The existing EAR contains the following projects:

- ▶ The existing EAR contains an EJB project called RAD8SimpleBankEJB.
- ▶ The existing EAR contains an EAR project called RAD8SimpleBankEAR.
- ▶ The EJB project contains the EJB SimpleBankFacadeBean and the remote interface SimpleBankBeanRemote.
- ▶ There are additional classes, but the remote interface only exposes primitive types, `java.math.BigDecimal`, and arrays of primitive types. This approach ensures that we can use the default SCA binding, which relies on JAXB serialization.

Setting the wire format to Java object: If you need to expose, on the interface, additional types that do not conform to the rules for JAXB serialization, consider changing the Wire format of the SCA binding by setting it to the **Java object** value (Figure 16-47 on page 939). For more information, see these websites:

- ▶ http://www14.software.ibm.com/webapp/wsbroker/redirect?version=matt&product=was-base-dist&topic=tzca_default_binding_serial
- ▶ http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.soafep.multiplatform.doc/info/ae/ae/tzca_default_binding_serial.html

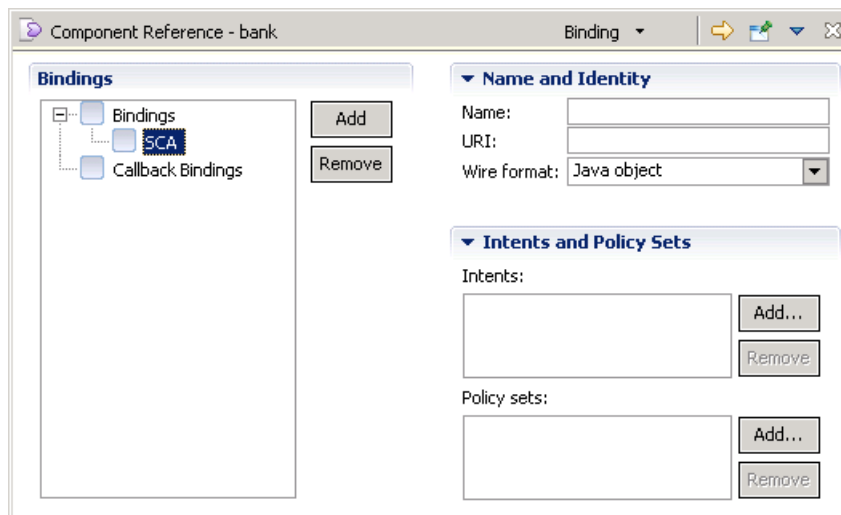


Figure 16-47 Setting the wire to the Java object format for the SCA binding

16.8.2 Creating a new SCA Enhanced EAR file to hold the web project

To create a new SCA Enhanced EAR to hold the web project, perform these steps:

1. Create a new enterprise application project called RAD8SCAWebEAR.
2. For the Target, select **WebSphere Application Server 8**.
3. For the EAR, select version **6** and select **Next**.
4. Create a new Web module called RAD8SCAWeb and select **Finish**.

Next we need to add SCA support to the EAR and web projects:

1. Right-click the project **RAD8SCAWebEAR** and select **Configure** → **Add SCA Support**.
2. Accept all defaults and select **Finish**.
3. Right-click the project **RAD8SCAWeb** and select **Configure** → **Add SCA Support**.
4. Accept all defaults and select **Finish**.

We now create a distinguished application composite in the SCA enhanced EAR:

1. In the Enterprise Explorer, right-click the logical folder **SCA Content** → **Composites** of the project **RAD8SCAWebEAR**.
2. Select **New** → **Composite**.
3. The New Composite Wizard opens (Figure 16-48 on page 941). Complete these steps:
 - a. The New Composite Wizard automatically selects **Distinguished Application Composite**.
 - b. The Composite name has the predetermined name of `application`. You cannot change this name in the case of a distinguished application composite.
 - c. For Target Namespace, enter `http://itso.bank.sca`.
 - d. Select **Finish**.

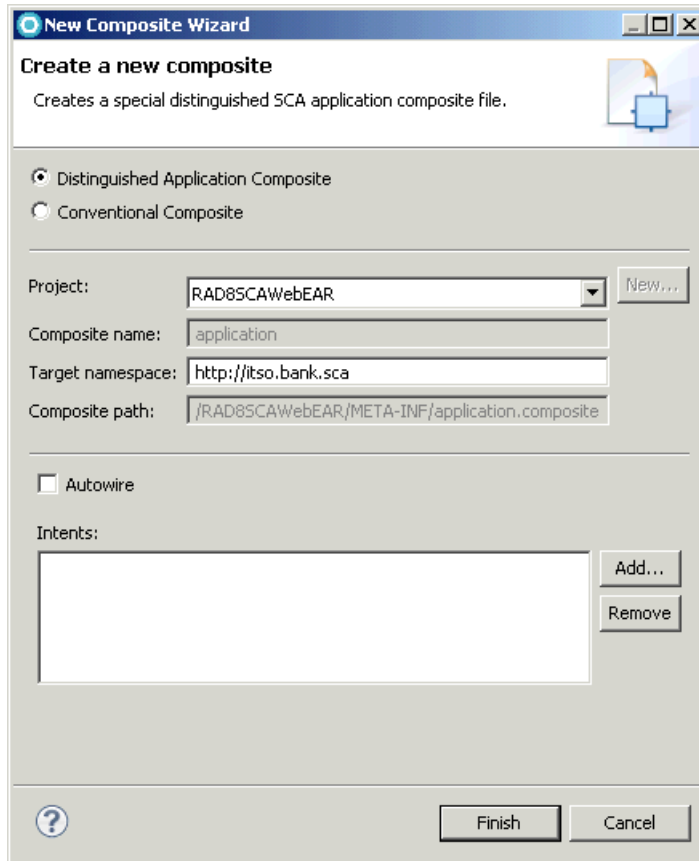


Figure 16-48 Create a new distinguished composite

4. Using the Palette, create a New Component called RAD8SCAWebComponent.
5. Right-click the component and select **Set Implementation** → **Web** (Figure 16-49 on page 942).

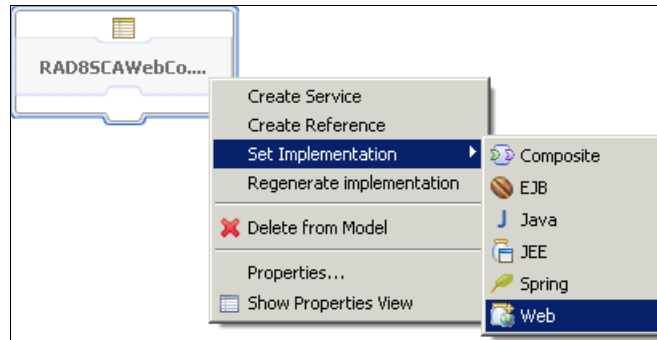


Figure 16-49 Creating a new component with web implementation

6. The **Web module Selection Dialog** box appears and shows the only web project available, which is RAD8SCAWeb (Figure 16-50).

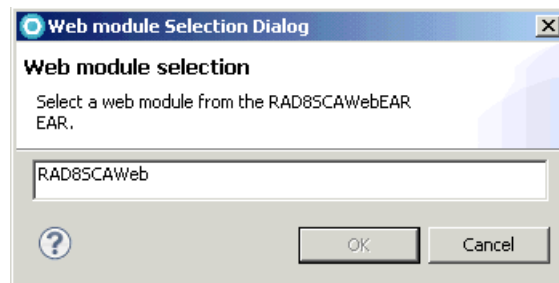


Figure 16-50 Web module Selection Dialog

7. Select **OK**.

Before we can add an SCA reference to this component, we must have the EJB remote interface available in this project:

1. In the Enterprise Explorer, locate the file `RAD8SimpleBankEJB\ejbModule\itso\rad8\bank\ejb\facade\SimpleBankRemote.java` and copy it to the `src` folder of the `RAD8SCAWeb` project. This action causes a compilation error due to the mismatch between the package statement and the current folder. Open the copied file and use the provided quick fix to move the file into the correct package, which has been created automatically.
2. Import the `index.html` file that is located in `7835code\sca\index.html` into `RAD8SCAWeb\WebContent`.
3. In the `RAD8SCAWeb` project, create a new servlet called `ListAccounts` in the `itso.rad8.bank.servlets` package.

4. Overwrite the automatically generated code with the code that was imported from `7835code\sca>ListAccounts.java`.
5. Example 16-13 shows `itso.bank.test.servlet.ListCustomers.java`. Observe the use of the `@Reference` annotation to get the EJB proxy injected by the container into the servlet.

Example 16-13 Servlet code using @Reference annotation

```
package itso.rad8.bank.servlets;

import itso.rad8.bank.ejb.facade.SimpleBankRemote;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.osoa.sca.annotations.Reference;

/**
 * Servlet implementation class ListAccounts
 */
@WebServlet("/ListAccounts")
public class ListAccounts extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @Reference SimpleBankRemote simpleBank;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public ListAccounts() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request,
     HttpServletResponse response)
     */
}
```

```

        protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
            String ssn=request.getParameter("ssn");
            PrintWriter out=response.getWriter();

            String fullName= simpleBank.getCustomerFullName(ssn);
            System.out.println(fullName);
            out.println("<html><body><h2>Customer Listing</h2>");
            out.print("<br> Customer Name: "+fullName+"</br>");
            String[] accounts= simpleBank.getAccounts(ssn);
            for(int i=0;i<accounts.length;i++){
                String account = accounts[i];
                System.out.println(account);
                out.print("<br> "+account+"</br>");
            }
            out.println("</body><html>");

        }

        /**
         * @see HttpServlet#doPost(HttpServletRequest request,
        HttpServletResponse response)
         */
        protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
            doGet(request,response);
        }

    }

```

Now open **application.composite** and update the component with the reference:

1. In the Enterprise Explorer, select **RAD8SCAWebEAR** → **SCA Content** → **Composites** → **http://itso.bank.sca** → **application** and open it (with the SCA Composite Editor).
2. Right-click the **RAD8SCAWebComponent** component and select **Create Reference**.
3. Open the properties of the reference:
 - a. In Core, enter the name `simpleBank` (this name *must* be equal to the name of the reference in the Java code).

- b. In Interface, complete these tasks:
 - i. For the InterfaceType, select **Java**.
 - ii. Browse to the Interface **itso.bank.service.EJBBankService**.
- c. In Binding, you can add a Binding of type SCA, but adding this binding type is unnecessary, because SCA is the default binding. You only add it explicitly if you need to customize it, for example, by changing the default wire format to Java object.

We promote this reference so that it can be used in another composite:

1. Right-click the Reference on the diagram and select **Promote**.
2. Select the promoted interface. Observe that it has the Promote value set to: RAD8SCAWebComponent/simpleBank (Figure 16-51).

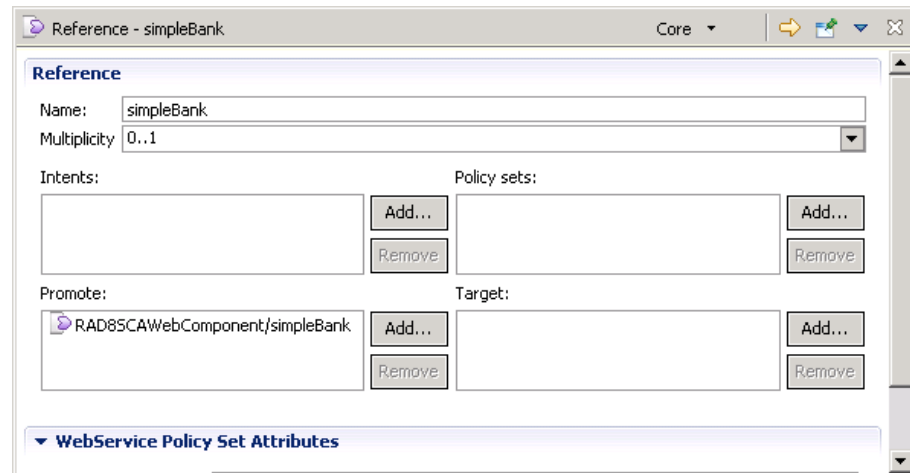


Figure 16-51 Promoted interface core properties

The distinguished composite file, `application.composite`, now has the contents that are shown in Example 16-14.

Example 16-14 Content of the `application.composite` file

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
xmlns:was="http://www.ibm.com/xmlns/prod/websphere/sca/1.0/2007/06"
autowire="false" name="application"
targetNamespace="http://itso.bank.sca">
  <component name="RAD8SCAWebComponent">
    <implementation.web web-uri="RAD8SCAWeb.war"/>
    <reference name="simpleBank">
```

```
<interface.java
interface="itso.rad8.bank.ejb.facade.SimpleBankRemote"/>
</reference>
</component>
<reference name="simpleBank"
promote="RAD8SCAWebComponent/simpleBank"/>
</composite>
```

16.8.3 Creating a new SCA project with a contribution

Next we create an SCA project with a contribution that we can deploy to the server:

1. Create a new SCA project called `RAD8SCAProject`, accepting all defaults, and select **Finish**.
2. Inside the project, create a new composite with the following parameters (Figure 16-52 on page 947):
 - a. For the composite type, select **Conventional Composite** (because this project is an SCA project, we can create a deployable composite).
 - b. For Composite name, enter `RAD8SCAComposite`.
 - c. For Target namespace, enter `http://itso.bank.sca`.
 - d. Accept the default Composite path.
 - e. Select **Finish**.

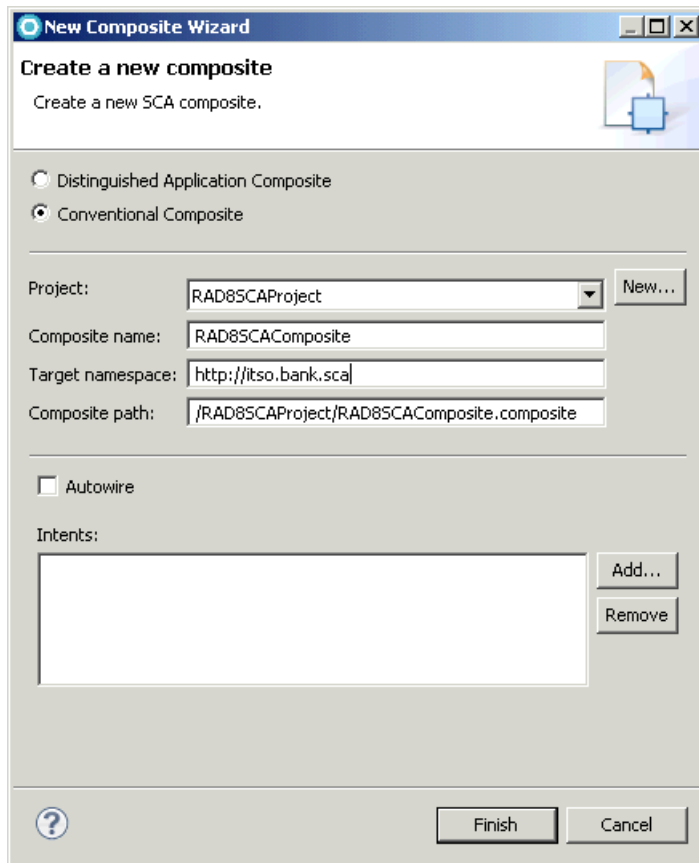


Figure 16-52 Creating a conventional composite in the SCA project

We create the two components and wire them together:

1. Create a new Component called RAD8EJBComponent:
 - a. Set the Implementation to **JEE**. For Project, select **RAD8SimpleBankEAR**.
 - b. Add a service to this component:
 - i. When we declare a service with the @Service annotation inside a Java file, we name the service after the interface declared in the annotation. In this case, we do not use this annotation, because we do not enhance the Java EE application. So, we must use the default name, which is obtained by applying the following pattern:


```
EJBName_EJBInterfaceName
```

 In this case, this pattern resolves to this default name:

SimpleBankFacadeBean_SimpleBankRemote

- ii. Set the Interface type to **Java** and the interface to **itso.rad8.bank.ejb.facade.SimpleBankRemote**.
 2. Create a new component called RAD8SCAWebComponent:
 - a. Set the Implementation to **JEE**, and for the Project, select **RAD8SCAWebEAR**.
 - b. Create a reference on this component:
 - i. For the Name, enter `simpleBank`.
 - ii. Set the Interface type to **Java** and the interface to **itso.rad8.bank.ejb.facade.SimpleBankRemote**.
 3. Wire the reference to the service, which sets the target of the reference to `RAD8EJBComponent/SimpleBankFacadeBean_SimpleBankRemote`.
 4. The resulting diagram looks like Figure 16-53.

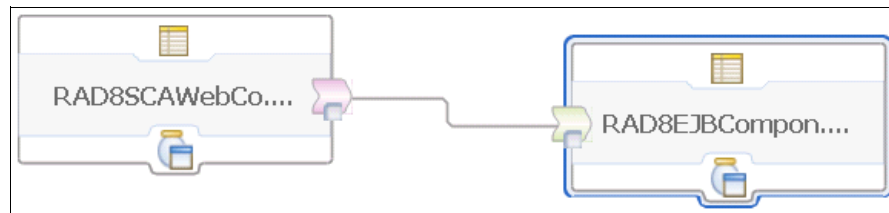


Figure 16-53 Completed conventional composite

5. Example 16-15 shows the `RAD8SCAComposite.composite` file contents.

Example 16-15 Completed conventional composite text

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
xmlns:was="http://www.ibm.com/xmlns/prod/websphere/sca/1.0/2007/06"
autowire="false" name="RAD8SCAComposite"
targetNamespace="http://itso.bank.sca">
  <component name="RAD8EJBComponent">
    <implementation.jee archive="RAD8SimpleBankEAR.ear"/>
    <service name="SimpleBankFacadeBean_SimpleBankRemote">
      <interface.java
interface="itso.rad8.bank.ejb.facade.SimpleBankRemote"/>
    </service>
  </component>
  <component name="RAD8SCAWebComponent">
    <implementation.jee archive="RAD8SCAWebEAR.ear"/>
```



```
<reference name="simpleBank"
target="RAD8EJBComponent/SimpleBankFacadeBean_SimpleBankRemote">
  <interface.java
interface="itso.rad8.bank.ejb.facade.SimpleBankRemote"/>
  </reference>
</component>
</composite>
```

6. To allow the SCA project to resolve the EJB remote reference, in the Enterprise Explorer, locate the file `RAD8SimpleBankEJB\ejbModule\itso\rad8\bank\ejb\facade\SimpleBankRemote.java` and copy it to the src folder of the project `RAD8SCAProject`.
7. This task gives a compilation error due to the mismatch between the package statement and the current folder. Open the copied file and use the provided quick fix to move the file into the correct package, which has been created automatically.
8. Finally, create a contribution and add `RAD8SCAComposite` to it (Figure 16-54 on page 950).

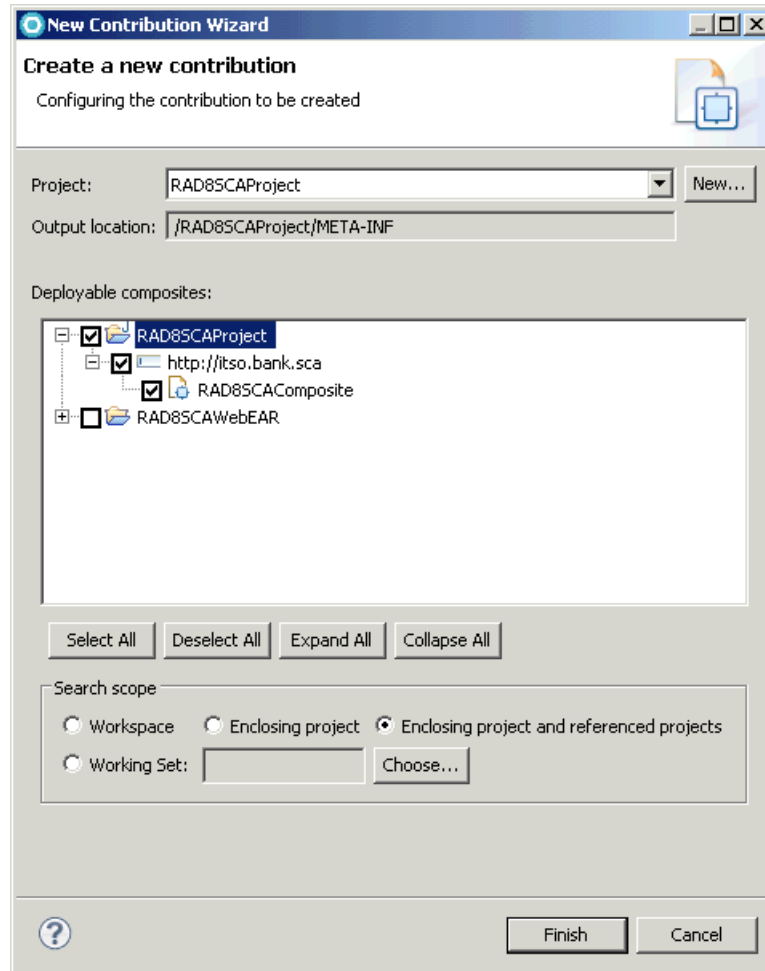


Figure 16-54 Creating a contribution

9. Example 16-16 shows the contents of the sca-contribution file.

Example 16-16 Contents of the sca-contribution file

```

<?xml version="1.0" encoding="UTF-8"?>
<contribution xmlns="http://www.oxa.org/xmlns/sca/1.0">
  <deployable composite="ns1:RAD8SCAComposite"
    xmlns:ns1="http://itso.bank.sca"></deployable>
</contribution>

```

The application is now complete and ready to test. You can locate the completed application in this file:

7835codesolution\sca\RAD8SimpleBankSCA.zip

16.8.4 Testing the completed application

To test the application, right-click the server and select **Add/Remove Projects**. Make sure to select only the SCA contribution, **RAD8SCAProject**, which contains the two EAR files, as shown in Figure 16-55.

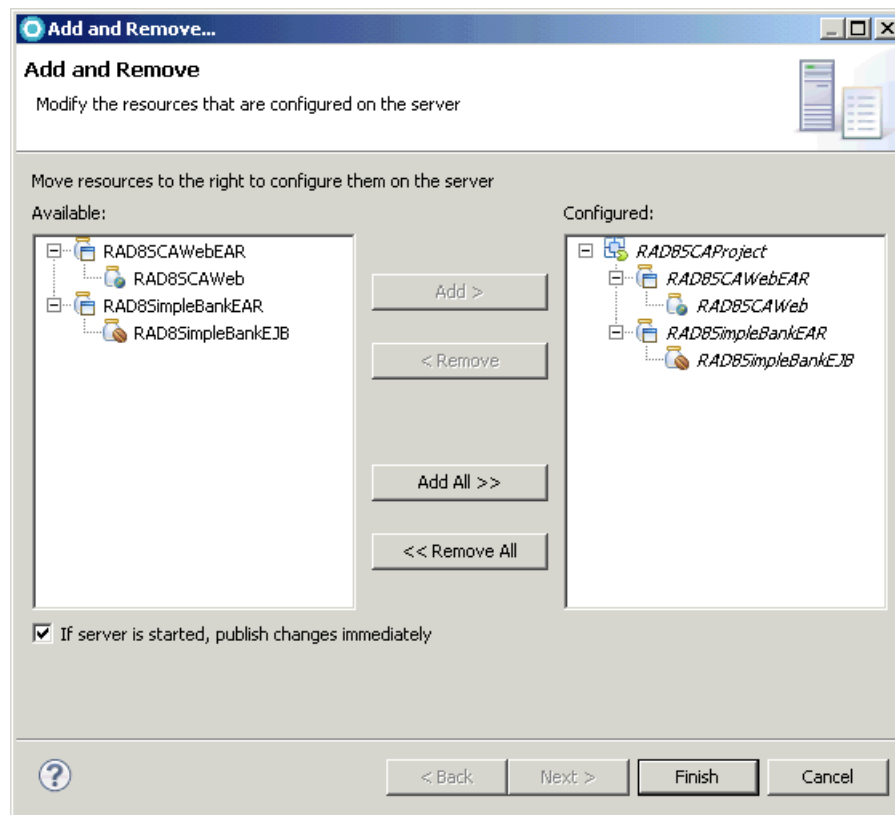


Figure 16-55 Adding the SCA contribution to the server

To test the application, open a web browser and enter the following URL (Figure 16-56 on page 952):

`http://localhost:9080/RAD8SCAWeb/index.html`

As usual, the port number might differ in your profile.

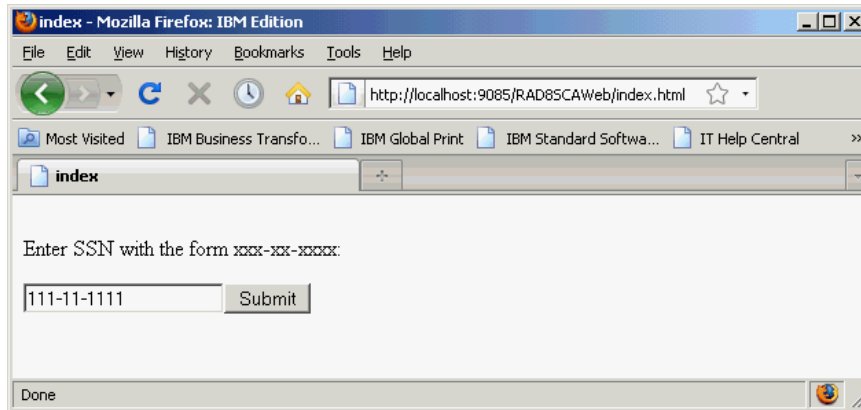


Figure 16-56 Testing index.jsp in the browser

Do not use the Run On Server menu that is available when you right-click `index.html` in Rational Application Developer. If you do, the EAR `RAD8SCAWebEAR` is also added to the server, outside of the context of the SCA contribution, which leads to runtime errors.

In the format that is shown in Figure 16-56, enter one of the SSN numbers, 111-11-1111, 222-22-2222, up to 999-99-9999, and click **Submit**. Figure 16-57 shows the results.

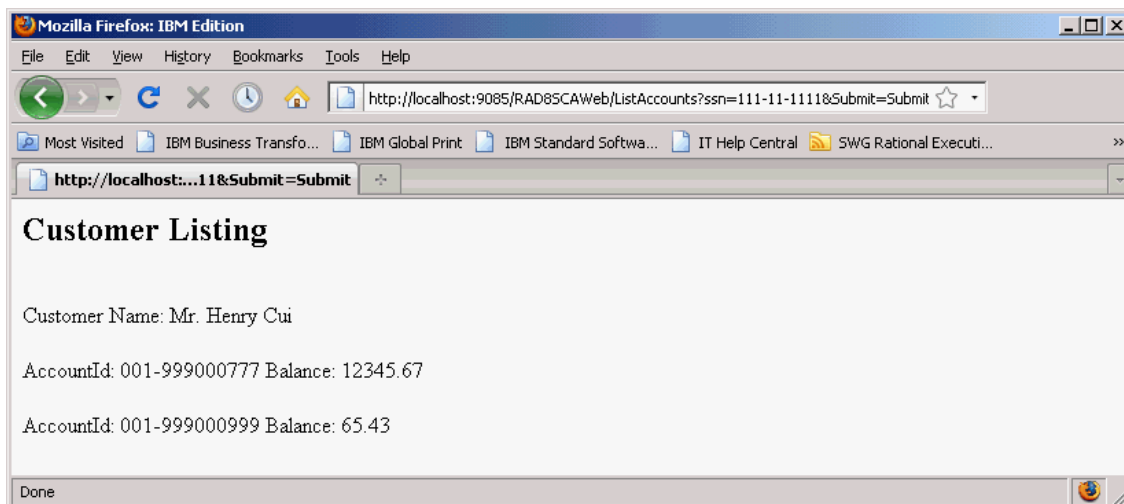


Figure 16-57 Invoking the servlet that contains the `@Reference` annotation

16.9 Adding intents and policies

A detailed description of intents and policies is beyond the scope of this book; however, you can find more information at the following sources:

- ▶ **Feature Pack for SCA, Version 1.0.1 → End-to-end paths → EJB applications → Using the transaction service → Transaction support in WebSphere Application Server: SCA transaction intents**
http://www14.software.ibm.com/webapp/wsbroker/redirect?version=v701sca&product=was-nd-mp&topic=cscsca_global_trans
- ▶ **WebSphere Application Server, Version 8.0 → End-to-end paths → EJB applications → Using the transaction service → Transaction support in WebSphere Application Server: SCA transaction intents**
http://www14.software.ibm.com/webapp/wsbroker/redirect?version=matt&product=was-base-dist&topic=cscsca_global_trans
- ▶ **Feature Pack for SCA, Version 1.0.1 → Securing applications and their environment → Authorizing access to resources: Using SCA authorization and security identity policies**
http://www14.software.ibm.com/webapp/wsbroker/redirect?version=v701sca&product=was-nd-mp&topic=tsec_authsoa_policy
- ▶ **WebSphere Application Server (Distributed platforms and Windows), Version 8.0 → Administering applications and their environment → Administering web services (generally applicable) → Managing web services policy sets: Mapping abstract intents and managing policy sets**
http://www14.software.ibm.com/webapp/wsbroker/redirect?version=matt&product=was-base-dist&topic=twbs_sca_intent_policyset

16.10 More information

- ▶ Open Service Oriented Architecture (OSOA)
<http://www.osoa.org>
- ▶ David Chappell, Introducing SCA
http://www.davidchappell.com/articles/introducing_sca.pdf
- ▶ Apache Tuscany
<http://tuscany.apache.org/sca-overview.html>
- ▶ Tutorials in the product:
 - SCA Hello World

- SCA Java EE
- SCA Spring
- SCA Account Services Application
- ▶ Samples in the product:
 - SCA HelloWorld sample
 - SCA Spring sample
 - SCA Service Data Object (SDO) sample
 - SCA Java EE sample
 - SCA JMS sample
 - SCA Web 2.0 sample
 - SCA AccountServices sample
- ▶ Samples in WebSphere Application Server V7:

http://www14.software.ibm.com/webapp/wsbroker/redirect?version=v701sca&product=was-nd-mp&topic=cscs_samples
- ▶ Samples in WebSphere Application Server V8:

http://www14.software.ibm.com/webapp/wsbroker/redirect?version=matt&product=was-base-dist&topic=cscs_samples
- ▶ IBM Education Assistant: IBM WebSphere Application Server V7.0 Feature Pack for Service Component Architecture

http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/index.jsp?topic=/com.ibm.iea.wasfpsca/plugin_coverpage.html
- ▶ Exploring the WebSphere Application Server Feature Pack for SCA:
 - Part 1: An overview of the Service Component Architecture Feature Pack

http://www.ibm.com/developerworks/websphere/library/techarticles/0812_beck/0812_beck.html
 - Part 2: Web services policy sets

http://www.ibm.com/developerworks/websphere/library/techarticles/0901_coats/0901_coats.html
 - Part 3: Intents and policies available in the SCA feature pack

http://www.ibm.com/developerworks/websphere/library/techarticles/0902_beck/0902_beck.html
 - Part 4: SCA Java annotations and component implementation

http://www.ibm.com/developerworks/websphere/library/techarticles/0902_beck2/0902_beck2.html
 - Part 5: Protocol bindings for Service Component Architecture services

http://www.ibm.com/developerworks/websphere/library/techarticles/0904_beck/0904_beck.html

- Part 6: Using Spring with Service Component Architecture
http://www.ibm.com/developerworks/websphere/library/techarticles/1001_beck6/1001_beck6.html
- Part 7: Using Atom and JavaScript Object Notation (JSON) - Remote Procedure Call (RPC) for Web 2.0 support
http://www.ibm.com/developerworks/websphere/library/techarticles/1001_beck7/1001_beck7.html
- Part 8: Java EE support in the Feature Pack for SCA
http://www.ibm.com/developerworks/websphere/techjournal/1010_ponniah/1010_ponniah.html
- ▶ WebSphere Application Server V7 Feature Pack for SCA 1.0.1 Information Center
http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.soafep.multiplatform.doc/info/ae/ae/welcome_fep_sca.html
- ▶ WebSphere Application Server V8 Beta Information Center
http://www14.software.ibm.com/webapp/wsbroker/redirect?version=matt&product=was-base-dist&topic=welc6tech_sca_dev
- ▶ *Getting Started with the Feature Pack for OSGi Applications and JPA 2.0*, SG24-7911, which shows how to create SCA components that are implemented as OSGi bundles



Developing Modern Batch jobs on computing grids

In this chapter, we discuss the concepts of Modern Batch and cover the Compute-Intensive (CI) sample. We also provide an overview of the transactional capabilities.

We explore the features that are provided by Rational Application Developer for batch job development and deployment. We also demonstrate how Rational Application Developer can help with testing Java batch jobs.

The chapter is organized into the following sections:

- ▶ Introduction to Modern Batch
- ▶ New Modern Batch job tools in Rational Application Developer
- ▶ Working with the Compute-Intensive sample
- ▶ Overview of the Transactional batch capabilities
- ▶ Additional information

17.1 Introduction to Modern Batch

Since the invention of the computer, there has been a continual race between processing power and program complexity. Certain programs perform many sequential tasks, certain programs process many records, and other programs perform tasks that are CPU intensive. In any case, these programs have one thing in common, they take a long time to run and do not require user input.

Batch processing has been available on mainframe systems since the beginning. Unfortunately, batch processing requires a skill set usually not available outside the mainframe community. With the addition of Modern Batch to the Java programming language and to Rational Application Developer, the benefits of batch processing are now available to the client/server model of computing.

17.2 New Modern Batch job tools in Rational Application Developer

IBM has used its extensive experience with batch processing on the mainframe in the Rational Application Developer suite. With Rational Application Developer, you can now design, develop, and deploy Modern Batch applications into WebSphere Application Server V7.0.0.13 with the Feature Pack for Modern Batch.

The Modern Batch tools are now part of the Rational Application Developer installation program. Simply select the features as part of the Installation Manager installation process, as indicated in Figure 17-1 on page 959.

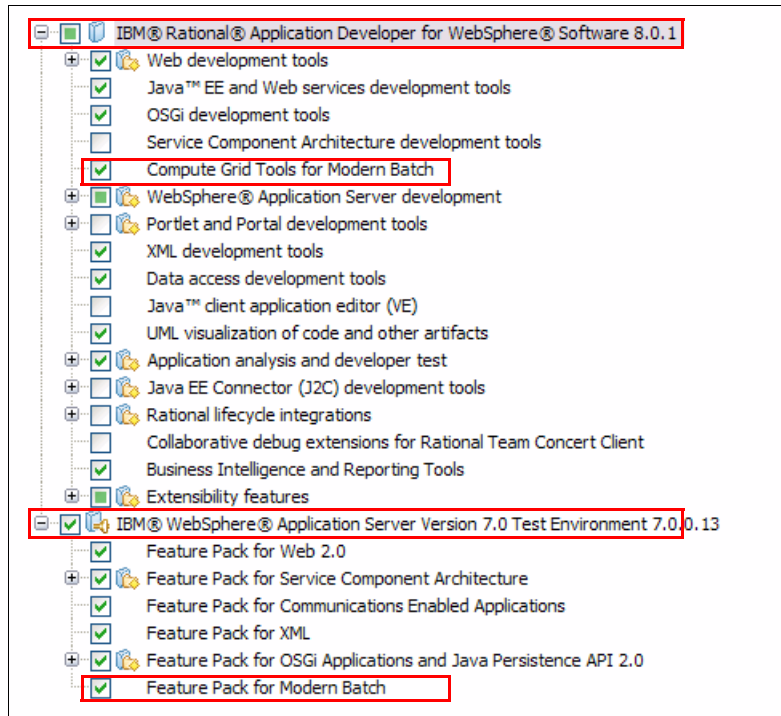


Figure 17-1 Installation features required for Modern Batch

This feature pack installs the wizards, project facets, samples, and run times that are required to develop Modern Batch applications.

17.3 Working with the Compute-Intensive sample

Rational Application Developer contains two samples of Modern Batch applications. One is *Transactional batch*-based and the other is *Compute-Intensive (CI) batch*-based. Transactional batch is intended for high volume record processing. Compute-Intensive batch is intended for low volume, CPU-intensive processing.

In this section, we analyze the CI batch example.

17.3.1 Installing the sample

You can access the CI batch example by following these steps:

1. In Rational Application Developer, select **Help** → **Help Contents**.

2. In the Help - Rational Application Developer for WebSphere Software window, expand **Samples** → **Batch** → **Modern Batch samples** → **Compute-Intensive batch application**.
3. Here, you can import the sample into the workspace. Click **Import Sample**, as shown in Figure 17-2.

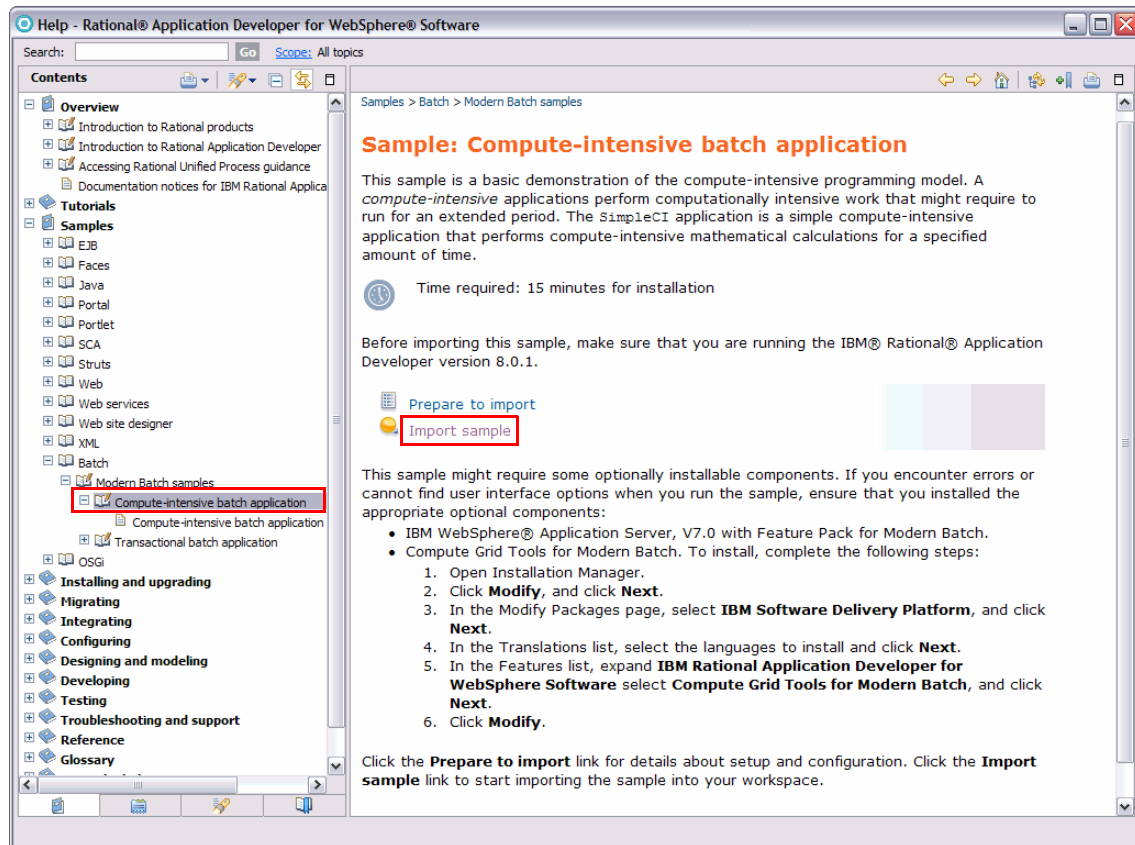


Figure 17-2 Importing the sample application

4. This action opens the Import Project wizard. Accept the default values and click **Finish**. (Figure 17-3 on page 961).

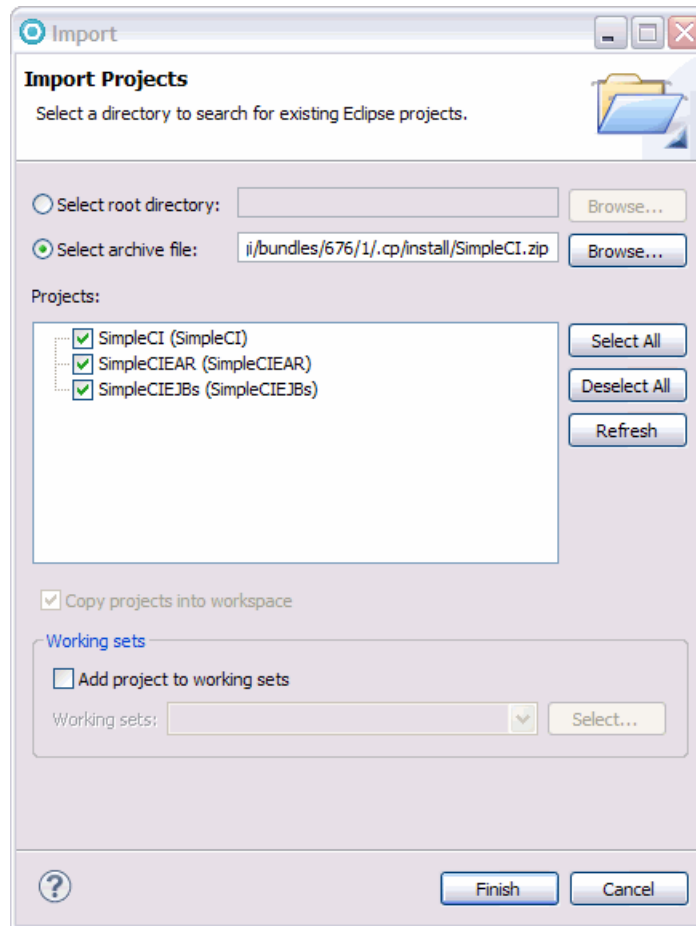


Figure 17-3 Importing the Compute-Intensive sample project

17.3.2 Understanding the sample

The Compute-Intensive (CI) sample consists of three projects:

- ▶ The application project: SimpleCI
- ▶ The EAR project: SimpleCIEAR
- ▶ The EJB project: SimpleCIEJB

The SimpleCI project contains the xJCL batch job definition and the classes that implement the CI pattern and business logic. See Figure 17-4 on page 962.

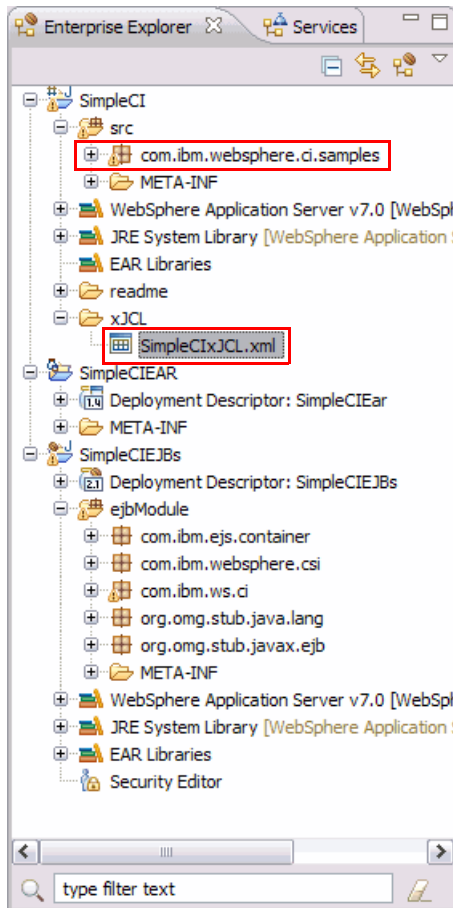


Figure 17-4 Sample project contents

To further understand the example, we examine the xJCL batch descriptor in the XJCL Editor, as seen in Figure 17-5 on page 963.

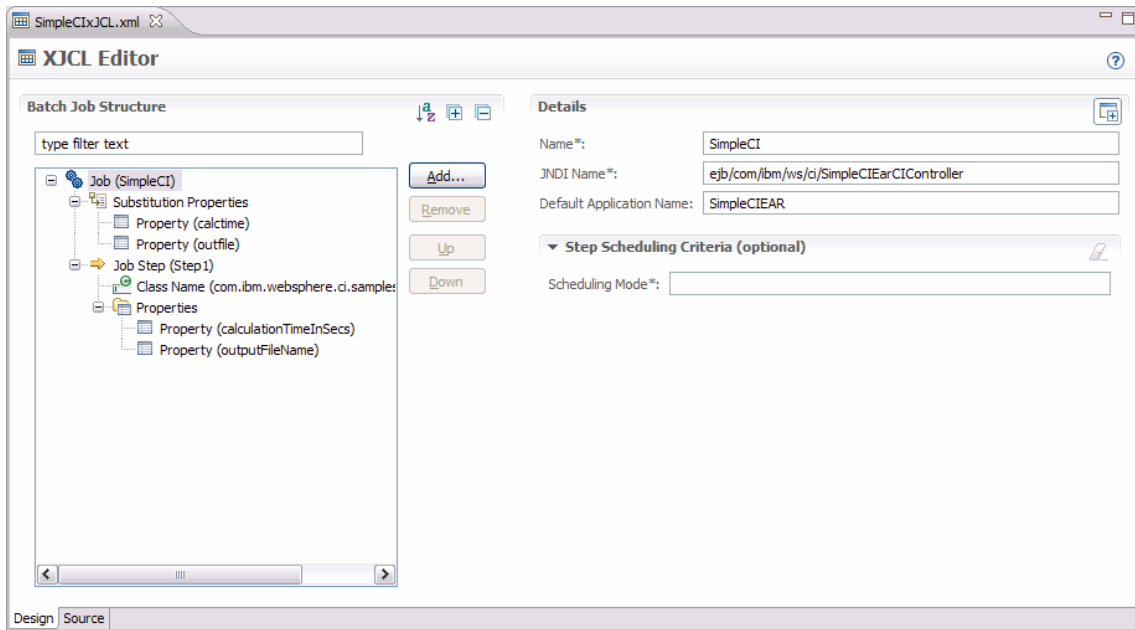


Figure 17-5 The XJCL Editor

The xJCL file is an XML file that assembles the pieces of the batch job. Depending on the type of batch job, CI or Transactional, you can configure the batch job contents by selecting the parent item and clicking **Add**, as shown in Figure 17-6 on page 964.

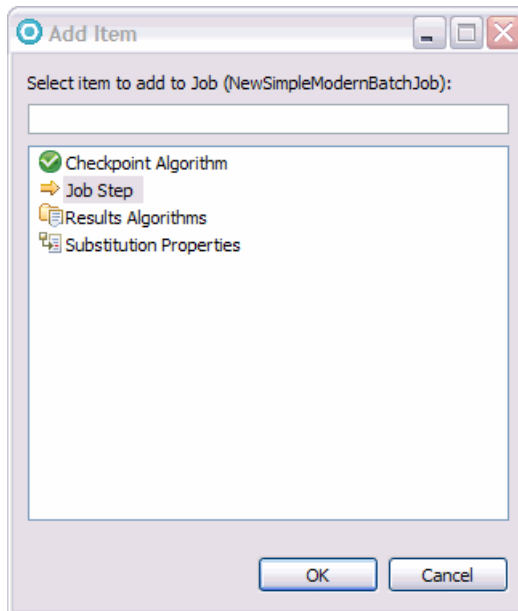


Figure 17-6 XJCL Editor contents

Important: Before the sample can be executed, you need to update one of the properties.

Before the sample can be executed, you need to update one of the properties, Property (*outfile*) and enter a filename for the output. See Figure 17-7 on page 965. The xJCL file allows for simple substitution of variables, which aids in readability and maintenance. The properties are passed to the implementation via the `setProperties` method for CI batch jobs and the `initialize` method for Transactional batch jobs.

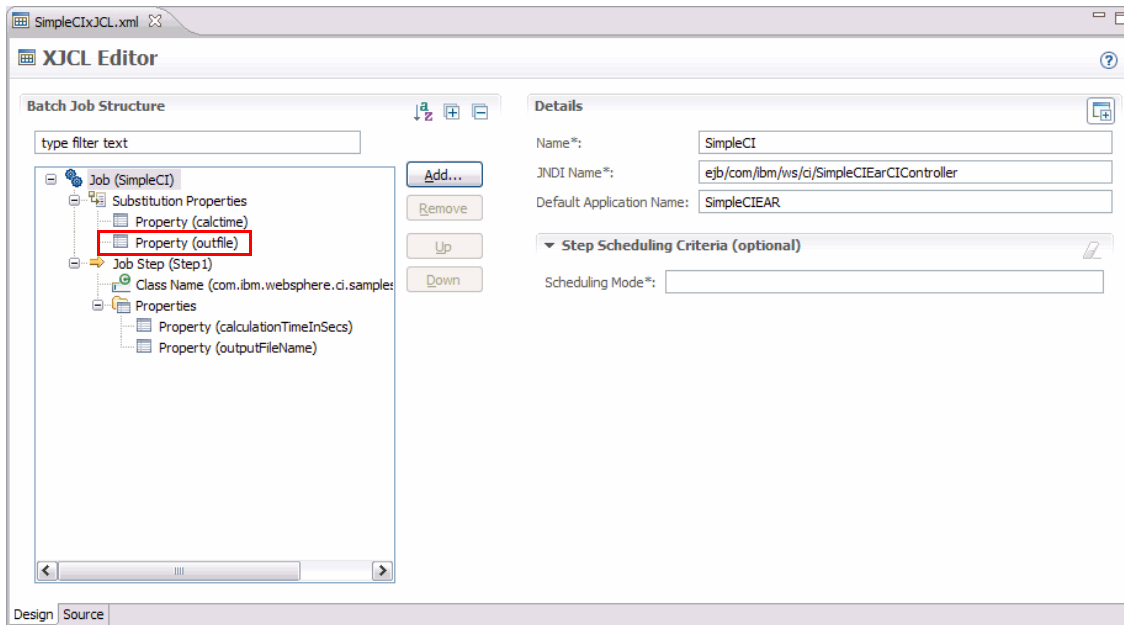


Figure 17-7 Property containing the output filename

The framework for the CI sample is simple. The sample implements the `com.ibm.websphere.ci.CIWork` interface. The run method contains the business logic and gets executed by the batch processing environment. The batch job uses this method to achieve the requirements.

17.3.3 Deploying the sample

After you have updated the *outfile* property to a valid file name, you can deploy the batch job. Batch jobs follow the normal procedure for deploying applications in Rational Application Developer.

WebSphere Application Server V7.0.0.13 has a Feature Pack for Modern Batch. See Figure 17-1 on page 959. After the feature pack has been installed, you can run the sample directly from the WebSphere Application Server Test Environment. Because the batch job runs on the WebSphere Application Server, if you want to debug the batch job, you must start the server in debug mode.

To start, right-click the WebSphere Application Server that has been added to Rational Application Developer and click **Add and Remove**. This step opens the Add and Remove window. See Figure 17-8 on page 966.

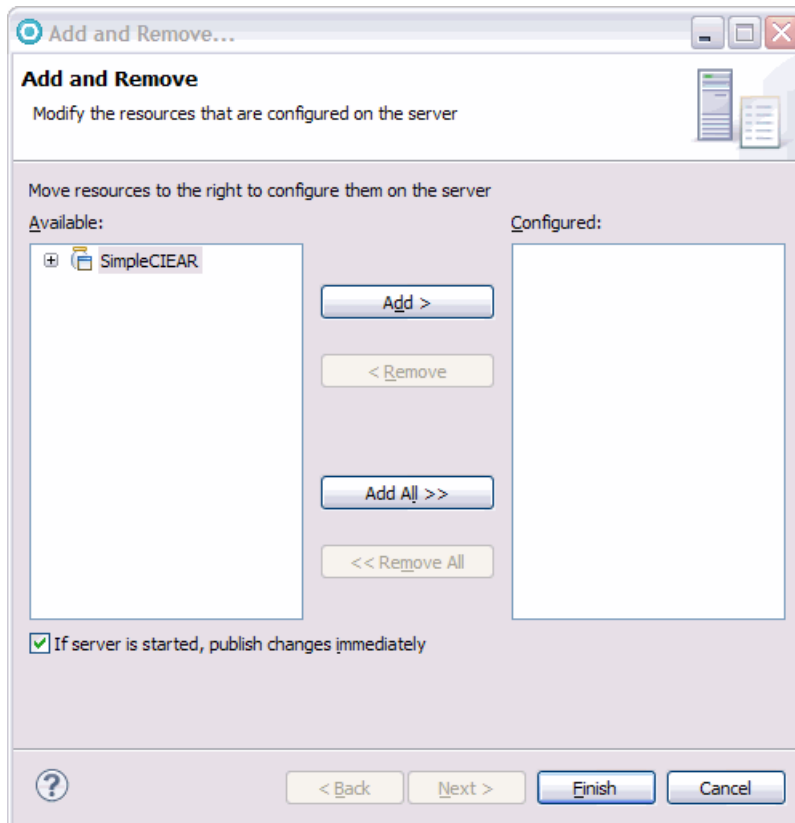


Figure 17-8 Adding the project to the WebSphere Application Server

Move the SimpleCIEAR project to the Configured pane and click **Finish**. The SimpleCIEAR must be in the [Started, Synchronized] state. If not, make sure that the WebSphere Application Server is started and click **Publish** again.

17.3.4 Running the sample

Installing the Compute Grid Tools for Modern Batch adds a new option to the Run As menu: Modern Batch Job. To execute the batch job, right-click the **xJCL** file and select **Run As** → **Modern Batch Job**. See Figure 17-9 on page 967.

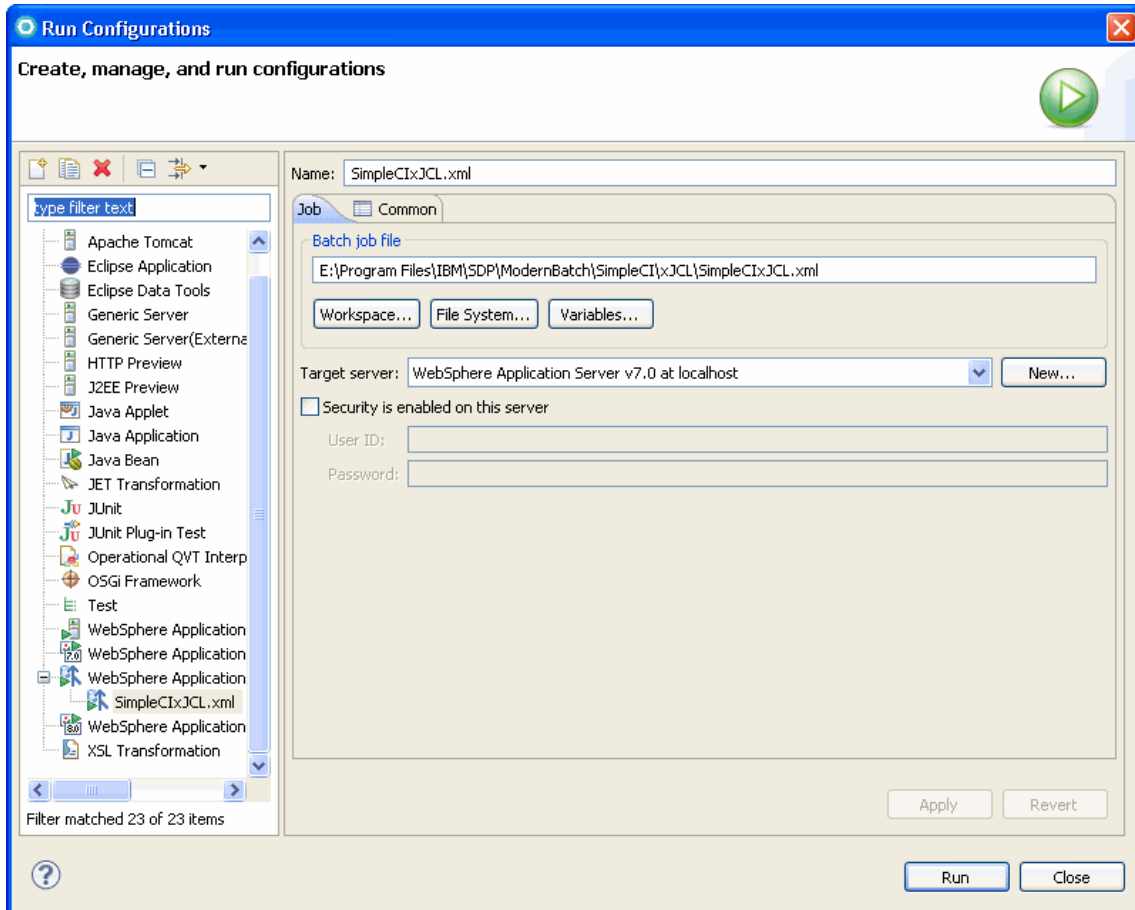


Figure 17-10 Run Configurations dialog window

Click **Run**, and the batch job launches on the WebSphere Application Server. The Modern Batch Job Management Console (JMC) opens. See Figure 17-11 on page 969.

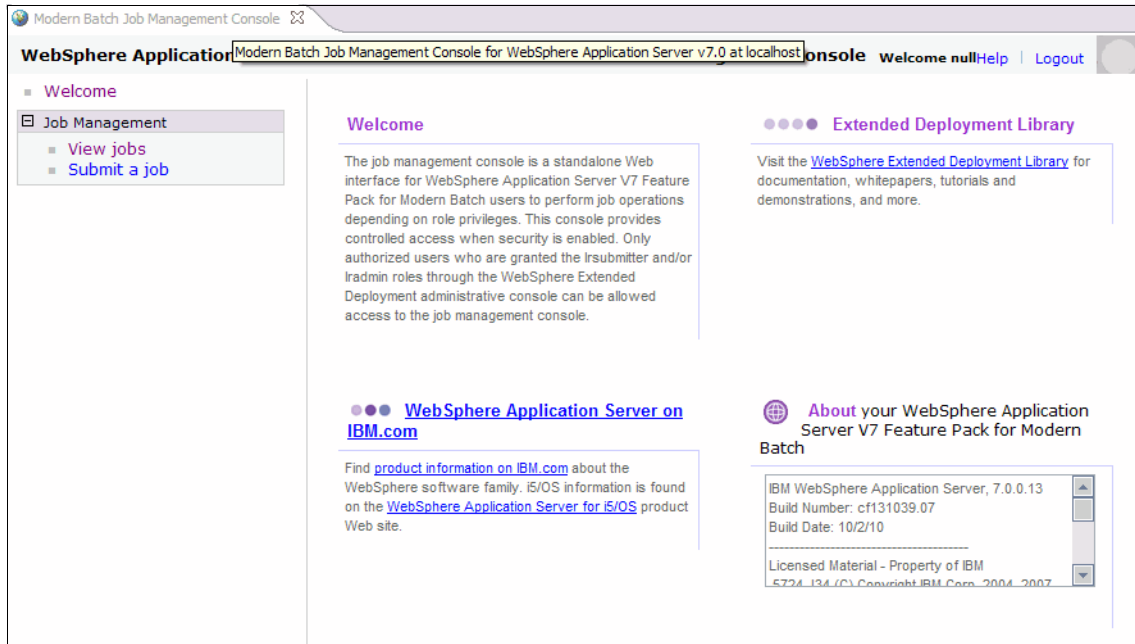


Figure 17-11 Modern Batch Job Management Console (JMC)

You can use the JMC to manage the batch jobs and check status. See Figure 17-12 on page 970 and Figure 17-13 on page 971. You can also view the WebSphere Application Server logs for significant information about the executing job.

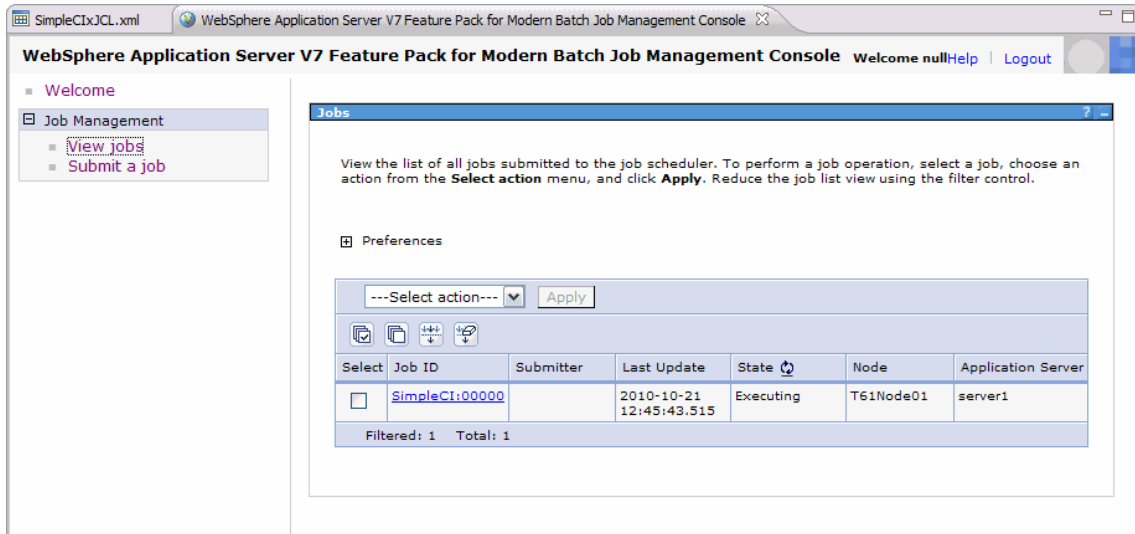


Figure 17-12 Modern Batch Job Management Console viewing jobs

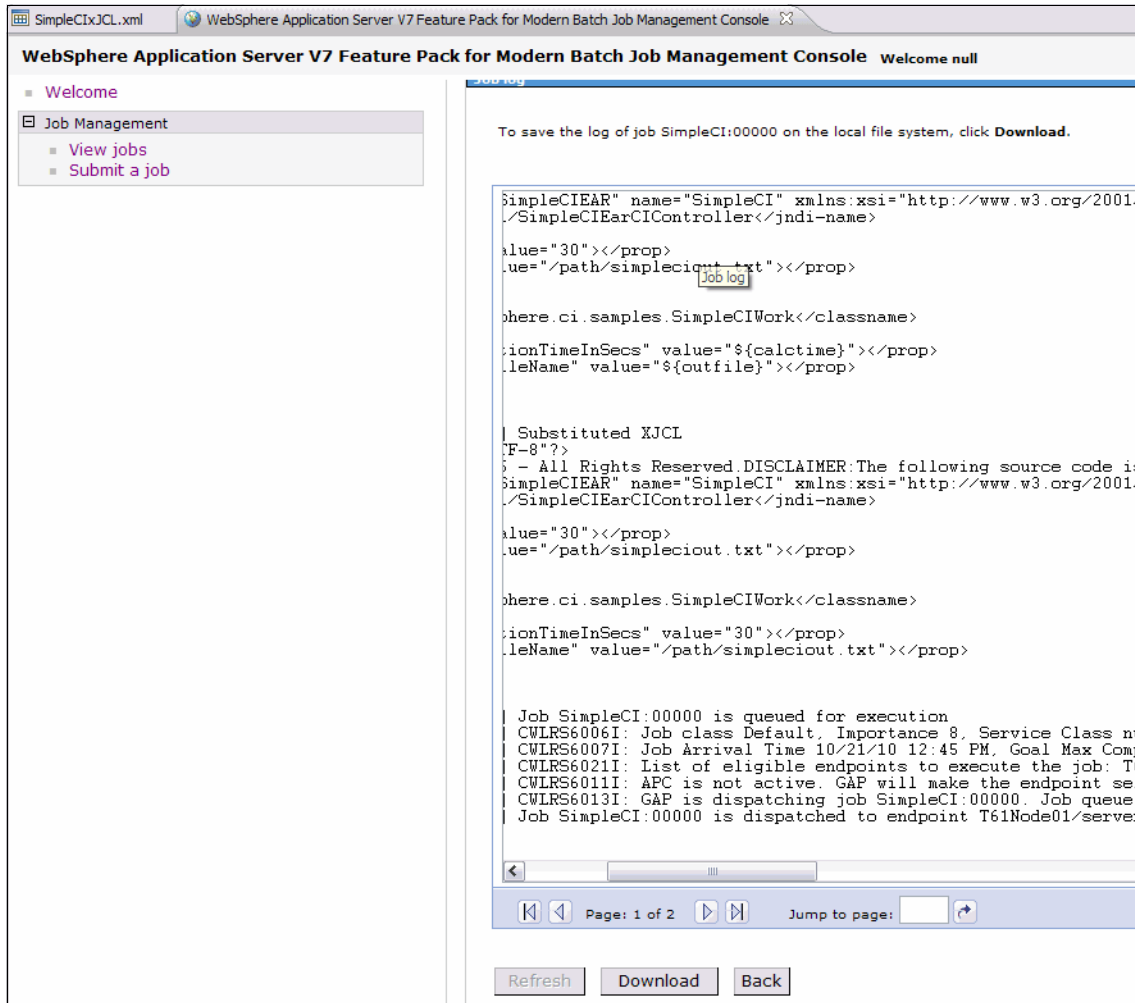


Figure 17-13 Modern Batch Job Management Console viewing log

For more information about running a batch application, see this website:

http://publib.boulder.ibm.com/infocenter/radhe1p/v8/index.jsp?topic=com.ibm.servertools.doc/topics/batch/t_batch_run_config.html

17.4 Overview of the Transactional batch capabilities

If your batch job handles a large volume of records, it is more suited to the Transactional batch pattern. The Transactional batch pattern allows for various

types of input streams for acquiring data and output streams for saving processed records. Also, multiple types of batch processors exist for controlling the execution of the batch job.

17.4.1 Sequence diagram for the Transactional batch pattern

We can best describe the interaction between the classes by the following sequence diagram. See Figure 17-14 on page 973 and Figure 17-15 on page 974. The overall sequence is simple and consists of three phases:

- ▶ Initialization: Sets up the streams and header
- ▶ Record processing: Iterates through the input and performs checkpoints
- ▶ Cleanup: Completes the checkpoints and closes the streams



Figure 17-14 Transactional batch job sequence diagram (Page 1 of 2)

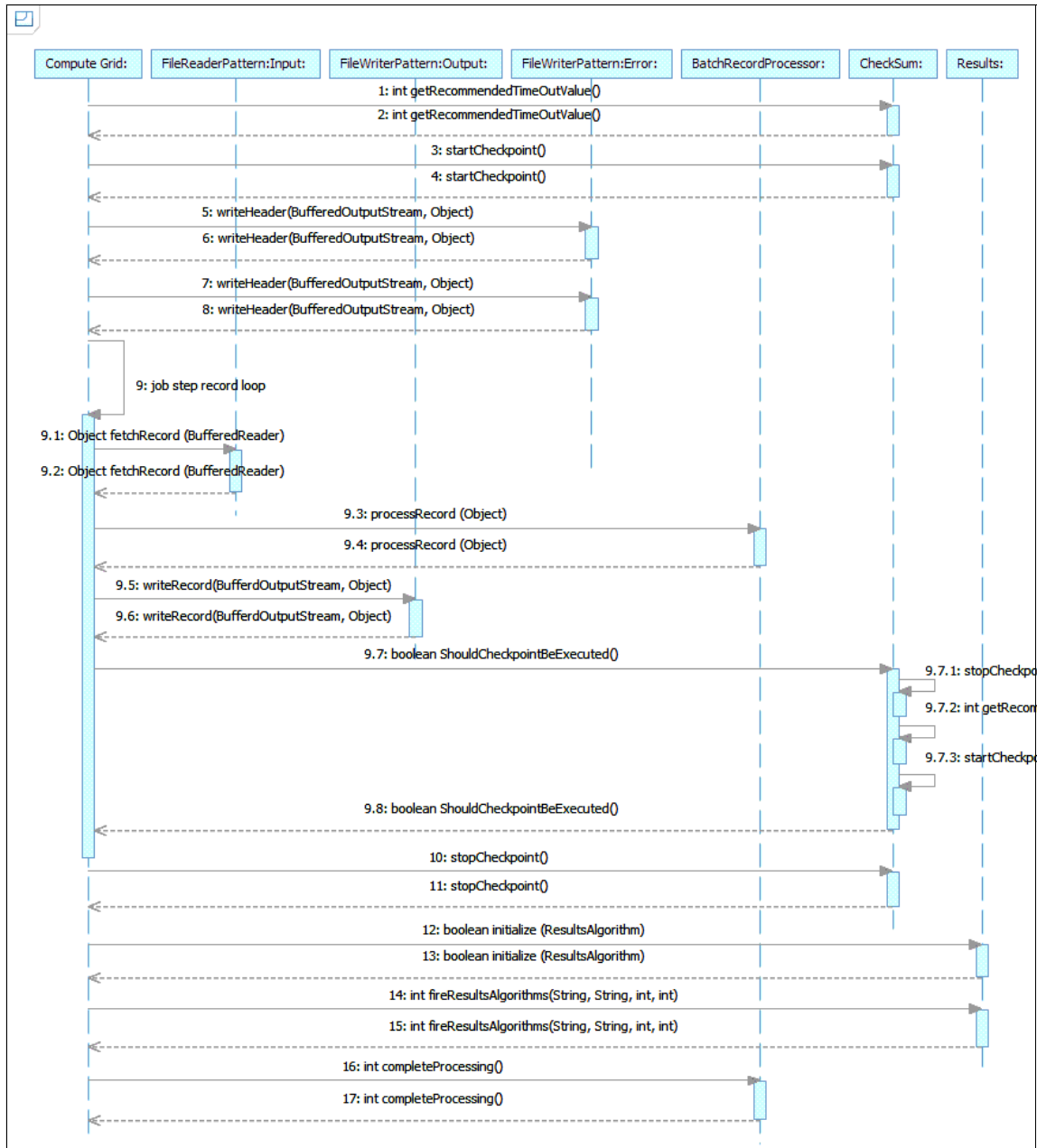


Figure 17-15 Transactional batch job sequence diagram (Page 2 of 2)

17.4.2 Available patterns

Specific patterns are available for batch processors and stream processors when creating a Transactional batch xJCL. See Figure 17-16 and Figure 17-17 on page 976.

Batch Step Creation
Assemble required properties for a batch step in an xJCL job file

Job Step Patterns

Name: StepCreation1A

Select Pattern: Generic

Implementation Class: com.ibm.websphere.batch.devframework.steps.technologyadapters.GenericX

Batch Processor: BatchRecordProcessor

Required Properties:

Name	Value
BATCHRECORDPROCESSOR	com.ibm.websphere.ci.samples.SimpleBatchRecor...

Optional Properties:

Name	Value
debug	
EnablePerformanceMeasurement	

< Back Next > Finish Cancel

Figure 17-16 Configuring the Batch Processor Pattern in a Transactional batch job

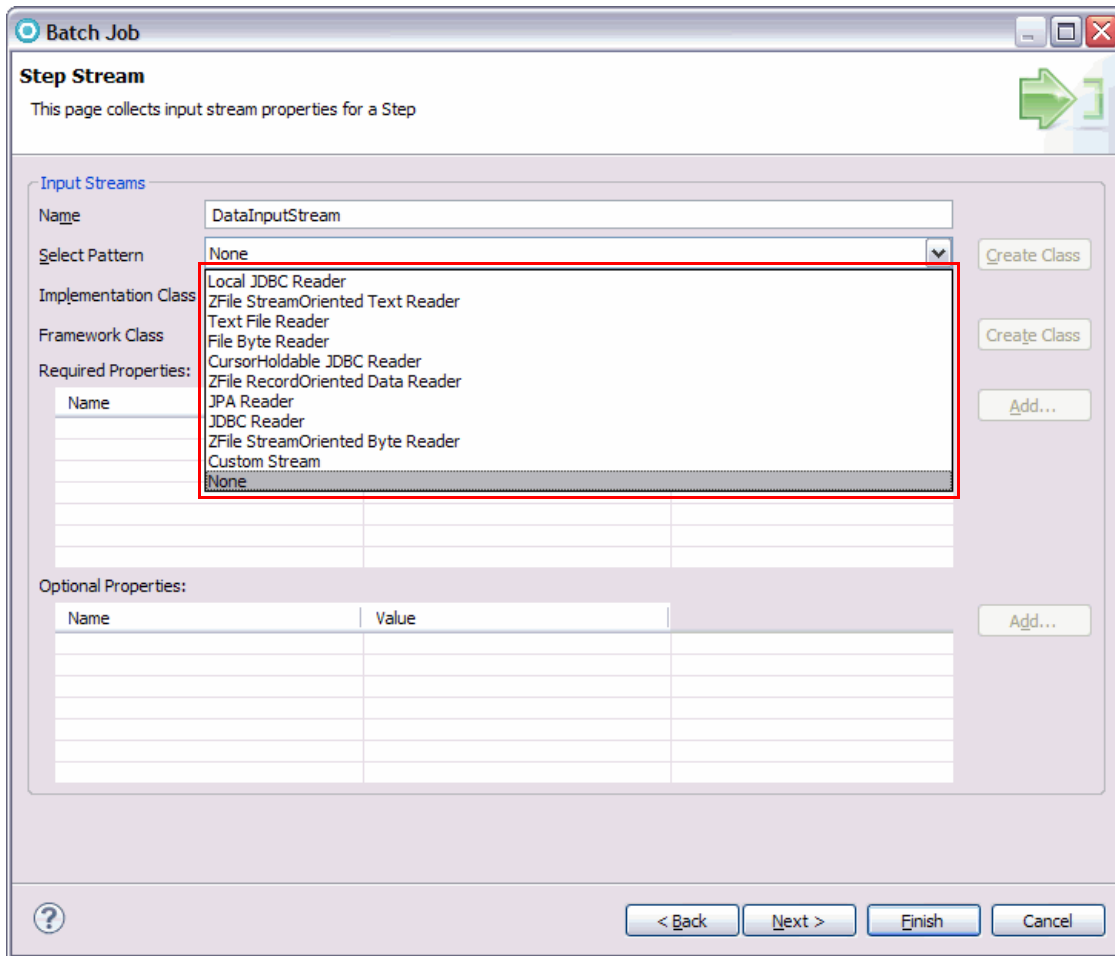


Figure 17-17 Pattern selection

The Modern Batch implementation in Rational Application Developer includes several batch step patterns (Table 17-1) and several input/output stream patterns (Table 17-2 on page 977 and Table 17-3 on page 977). Additionally, patterns are available for checkpoint algorithms and results algorithms.

Table 17-1 Batch step patterns

Batch step pattern	Description
Generic Batch Step	A simple step that uses one input and one output stream.
Error Tolerant Batch Step	A simple step that uses one input, one output, and one error stream.

Table 17-2 Stream processor patterns

Stream processor pattern	Description
JDBCReaderPattern	Used to retrieve data from a database using a Java Database Connectivity (JDBC) connection
JDBCWriterPattern	Used to write data to a database using a JDBC connection
ByteReaderPattern	Used to read byte data from a file
ByteWriterPattern	Used to write byte data to a file
FileReaderPattern	Used to read a text file
FileWriterPattern	Used to write to a text file
RecordOrientedDatasetReaderPattern	Used to read a z/OS data set
RecordOrientedDatasetWriterPattern	Used to write to a z/OS data set
JPAReaderPattern	Used to retrieve data from a database using an OpenJPA connection
JPAWriterPattern	Used to write data to a database using an OpenJPA connection
Custom	Implements the BatchDataStream interface

Table 17-3 Threshold patterns

Threshold pattern	Description
RecordBasedThresholdPolicy	The record based threshold policy of RecordBasedThresholdPolicy is applicable only if the threshold batch step of ThresholdBatchStep is used. It counts the number of error records processed. If the result is greater than the threshold, it forces the job to go into a restartable state.
PercentageBasedThresholdPolicy	The percentageBasedThresholdPolicy is applicable only if the ThresholdBatchStep is used. It calculates the percentage of the number of error records processed to the total number processed. If the result is greater than the threshold, it forces the job to go into a restartable state.

17.5 Additional information

The following resources provide additional information:

- ▶ Information Center for the Feature Pack for Modern Batch (all operating systems)
http://publib.boulder.ibm.com/infocenter/wasinfo/fep/index.jsp?topic=/com.ibm.websphere.bpfep.multiplatform.doc/info/ae/ae/welcome_fepbp.html
- ▶ Configuring a WebSphere Application Server profile for the Feature Pack for Modern Batch
http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/com.ibm.servertools.doc/topics/batch/r_profiletemplate.html
- ▶ Feature Packs by version for WebSphere Application Server
<http://www-01.ibm.com/support/docview.wss?uid=swg27008534>
- ▶ *Batch Processing with WebSphere Compute Grid: Delivering Business Value to the Enterprise*, REDP-4566

Web application development

In this part, we describe how to use Rational Application Developer for web application development.


This part includes the following chapters:

- ▶ Chapter 18, “Developing web applications using JavaServer Pages (JSP) and servlets” on page 981
- ▶ Chapter 19, “Developing web applications using JavaServer Faces” on page 1057
- ▶ Chapter 20, “Developing web applications using Web 2.0” on page 1097
- ▶ Chapter 21, “Developing portal applications” on page 1131
- ▶ Chapter 22, “Developing Lotus iWidgets” on page 1183

Sample code for download: The sample code for all of the applications that are developed in this part is available for download at the following address:

<ftp://www.redbooks.ibm.com/redbooks/SG247835>

See Appendix C, “Additional material” on page 1877, for instructions.



Developing web applications using JavaServer Pages (JSP) and servlets

In this chapter, we develop web applications using JavaServer Pages (JSP), Java Platform, Enterprise Edition (Java EE) servlet technology, and static HTML pages, which are fundamental technologies for building Java EE web applications. Throughout the RedBank example, we guide you through the features that are available in Rational Application Developer to work with these technologies.

First, we describe the major tools that are available for web developers in Rational Application Developer and introduce the new features that are provided with Rational Application Developer v8. Next we present the design of the ITSO RedBank application and then build and test the application using the various tools available within Rational Application Developer. In the final section, we list sources of further information about Java EE web components and the Web Tools within Rational Application Developer.

The chapter is organized into the following sections:

- ▶ Introduction to Java EE web applications
- ▶ Web development tooling
- ▶ Rational Application Developer new features

- ▶ RedBank application design
- ▶ Implementing the RedBank application
- ▶ Web application testing

The sample code for this chapter is in the \7835code\webapp folder.

18.1 Introduction to Java EE web applications

Java EE is an application development framework that is the most popular standard for building and deploying web applications in Java. Two of the key underlying technologies for building the web components of Java EE applications are servlets and JSP. *Servlets* are Java classes that provide the entry point to logic for handling a web request and return a Java representation of the web response. *JSP* are a mechanism to combine HTML with logic written in Java. After they have been compiled and deployed, JSP run as a servlet, where they also take a web request and return a Java object representing the response page.

Typically, in a large project, the JSP and servlets are part of the presentation layer of the application and include logic to invoke the higher level business methods. The core business functions are usually separated into a clearly defined set of interfaces, so that these components can be used and changed independently of the presentation layer (or layers, when using more than one interface).

Enterprise JavaBeans (EJB) are also a key feature included in the Java EE framework and are a popular option to implement the business logic of an application. See Chapter 12, “Developing Enterprise JavaBeans (EJB) applications” on page 577, for details about EJB. The separation of the presentation logic, business logic, and the logic to combine them is referred to as the *model view controller (MVC)* pattern and is described later.

Technologies, such as Struts, JavaServer Faces (JSF), various JSP tag libraries, and even newer developments, such as Ajax, have been developed to extend the JSP and servlets framework to improve various aspects of Java EE web developments. For example, JSF facilitates the construction of reusable user interface (UI) components that can be added to JSP pages. We describe several of these technologies in other chapters of this book. However, the underlying technologies of these tools are extensions to Java servlets and JSP.

When planning a new project, the choice of technology depends on several criteria, such as the size of the project, previous implementation patterns, maturity of technology, and skills of the team. Using JSP with servlets and HTML is a comparatively simple option for building Java EE web applications.

Figure 18-1 shows the relationships among the Java EE, enterprise application, web applications, EJB, servlets, JSP, and additions, such as Struts and JSF.

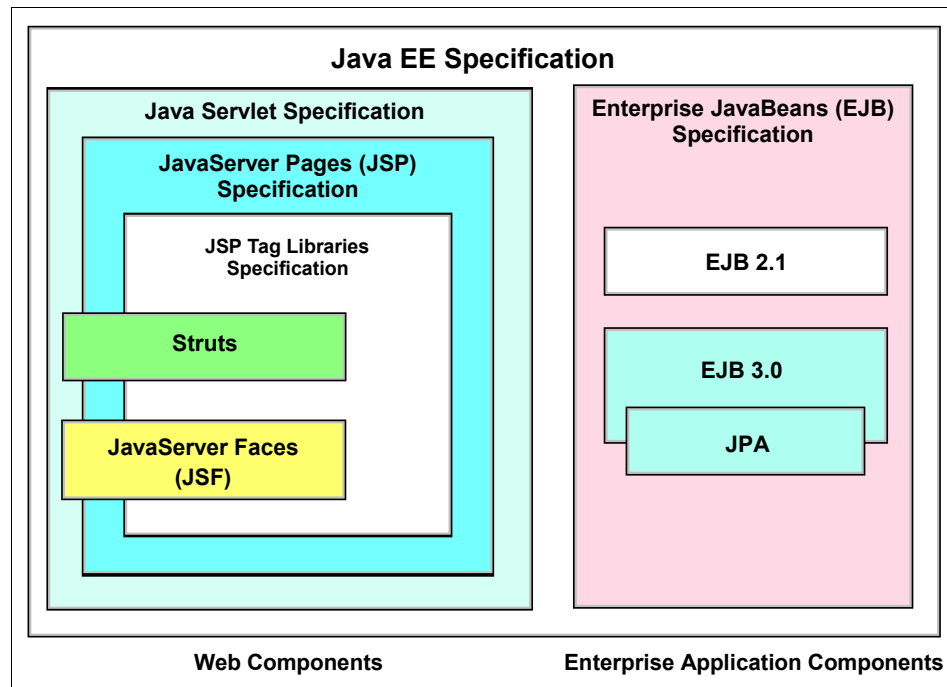


Figure 18-1 Java EE-related technologies

In this chapter, the focus is mainly on developing web applications using JSP, servlets, and static pages using HTML with the tools included with Rational Application Developer. After you master these concepts, you can more easily understand the other technologies that are available.

18.1.1 Java EE applications

At the highest level, the Java EE specification describes the construction of two application types that can be deployed on any Java EE-compliant application server:

- ▶ Web applications, which are represented by a web archive (WAR) file
- ▶ Enterprise applications represented by an enterprise archive (EAR) file

Both files are constructed in a compressed file format, with a defined directory and file structure. Web applications generally contain the web components that are required to build the information presented to the user and lower-level logic. The enterprise application contains an entire application, including the

presentation logic and logic implementing its interactions with an underlying database or other back-end system.

An EAR file can include one or more WAR files where the logic within the web applications usually invokes the application logic in the EAR.

Enterprise applications

An *enterprise application project* contains the collection of resources that are required to deploy an enterprise (Java EE) application to WebSphere Application Server. It can contain a combination of web applications (WAR files), EJB modules, Java libraries, and application client modules (all stored in JAR format). They also must include a deployment descriptor (an `application.xml` file in the `META-INF` directory), which contains meta information to guide the installation and execution of the application.

On deployment, the EAR file is unwrapped by the application server and the individual components (EJB modules, WAR files, and associated JAR files) are deployed individually. The JAR files within an enterprise application can be used by the other contained modules, allowing the sharing of code at the application level by multiple Web or EJB modules.

The use of EJB is not compulsory within an enterprise application. When developing an enterprise application (or even a web application), the developer can write whatever Java logic is the most appropriate for the situation. EJB are the defined standard within Java EE for implementing application logic, but many factors can determine the decision for implementing this part of a solution. In the RedBank sample application, the business logic is implemented using standard Java classes that use HashMaps to store data.

Web applications

A web application server publishes the contents of a WAR file under a defined URL root (called a *context root*) and then directs web requests to the correct resources and returns the appropriate web response to the requestor. Certain requests can be mapped to a simple static resource, such as HTML files and images. Other requests, which are referred to as *dynamic resources*, are mapped to a specific JSP or servlet class. Through these requests, the Java logic for a web application is initiated and calls to the main business logic are processed.

When a web request is received, the application server looks at the context root of the URL to identify for which WAR the request is intended, and the server reviews the contents after the root to identify to which resource to send the request. This resource might be a static resource (HTML file), the contents of which are returned, or a dynamic resource (servlet or JSP), where the processing of the request is handed over to JSP or servlet code.

In every WAR file, descriptive meta information describes this information and guides the application server in its deployment and execution of the servlets and JSP within the web application.

The structure of these elements within the WAR file is standardized and compatible between various web application servers. The Java EE specification defines the hierarchical structure for the contents of a web application that can be used for deployment and packaging purposes. All Java EE-compliant servlet containers, including the test web environment provided by Rational Application Developer, support this structure.

Figure 18-2 shows the structure of a WAR file, an EAR file, and a JAR file.

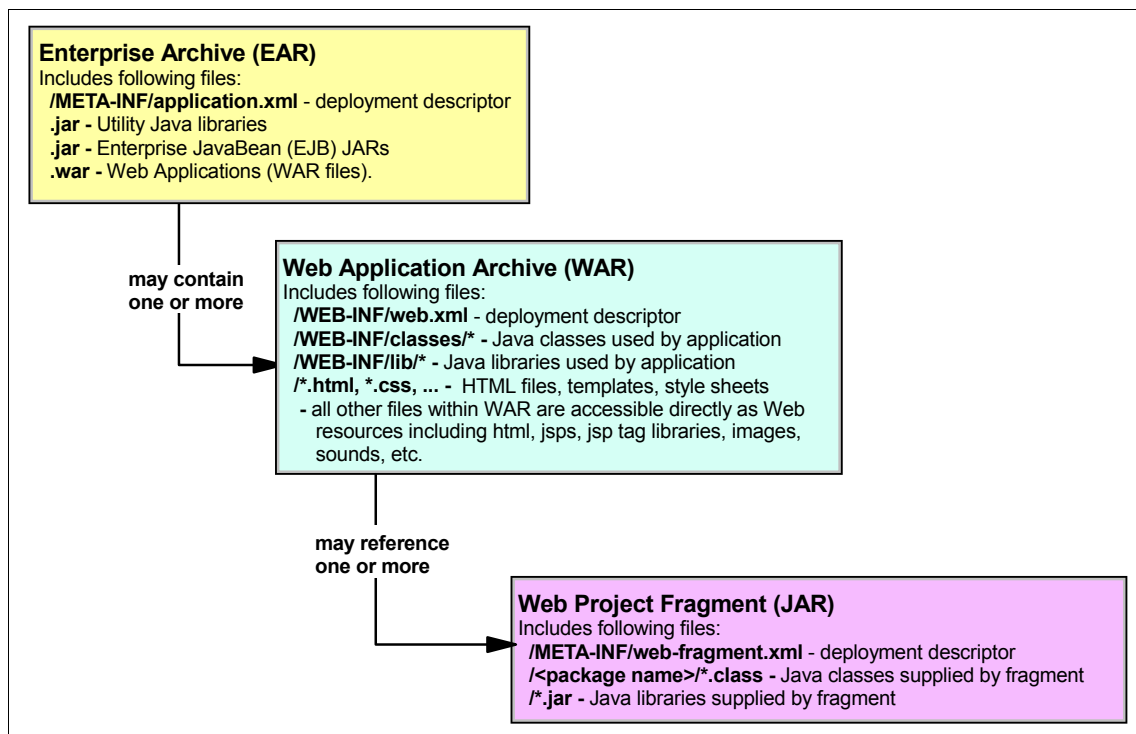


Figure 18-2 Structures of EAR, WAR, and JAR files

The *Java Specification Requests (JSR) 315: Java Servlet 3.0 Specification* (see <http://jcp.org/en/jsr/detail?id=315>) includes a series of Java annotations for declaring the fundamental classes that make up a JEE web application. These classes are used to define servlets, URL mappings to servlets, security (definition of which user groups can access a particular set of URLs), filters (a mechanism to call certain Java code before a request is processed), listeners (a mechanism to call specific Java code on certain events), and configuration

parameters to be passed to servlets or the application as a whole. In previous versions, these classes were defined in the `web.xml` file and the new version removes this dependency. However, it still can be used if required and Rational Application Developer includes tooling support for Version 2.5 and Version 3.0 of the Servlet specification.

A key extension in the latest version of the JEE specification (Version 6) is the addition of Web Fragment projects. Web Fragment projects are a mechanism to partition the web components in a WAR file into separate JAR files (called *Web Fragments*), which enhance or provide utility/framework logic to the main WAR file.

There are no requirements for the directory structure of a web application outside of the `WEB-INF` directory. All these resources are accessible to clients (general web browsers) directly from a URL, given the context root. Naturally, you must structure the web resources in a logical way for easy management. For example, use an `images` folder to store graphics.

Java EE web APIs

Figure 18-3 on page 988 shows the main classes that are used within the Java EE framework and the interactions between them. The application servlet class is the only class outside of the Java EE framework and contains the application logic.

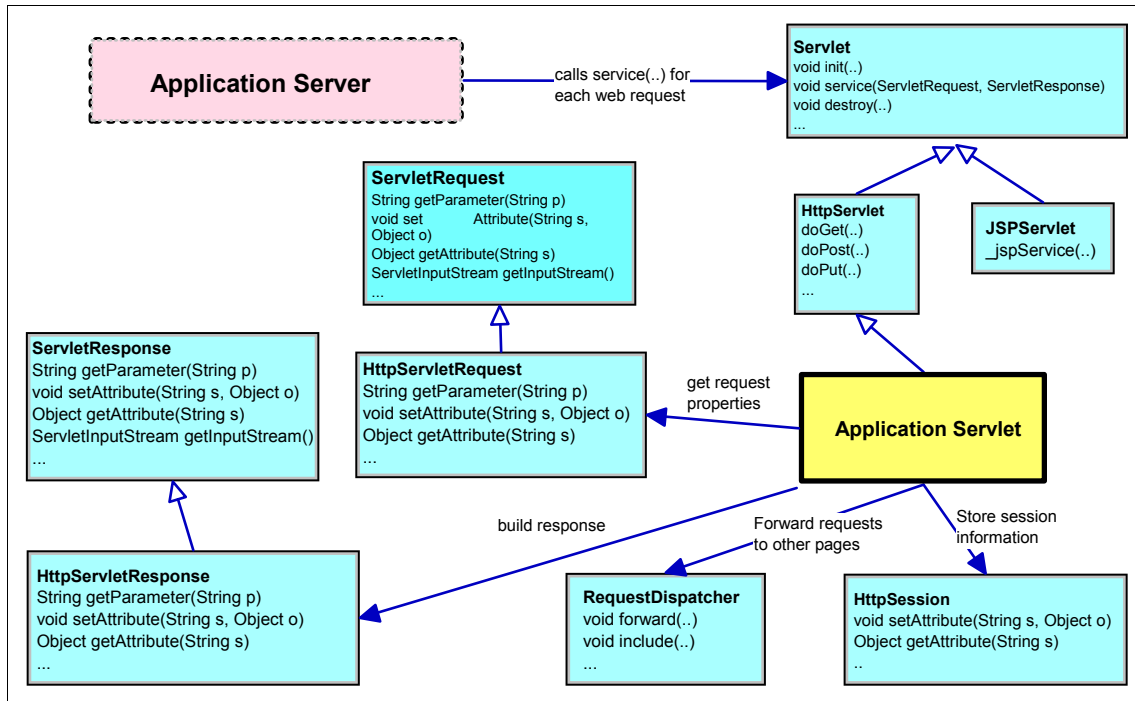


Figure 18-3 Java EE web component classes

The Java EE framework has the following main classes:

- ▶ `HttpServlet` (extends `Servlet`): The main entry point for handling a web request. The `doGet`, `doPost`, and other methods invoke the logic for building the response given the request and the underlying business data and logic. In the *JEE Servlet 3.0 specification*, these classes are identified by the `@WebServlet` annotation.
- ▶ `HttpJSPServlet` (extends `Servlet`): The WebSphere Application Server automatically compiles a JSP page into a class that extends this type. It runs similarly to a normal servlet, and its only entry point is the `_jspService` method.
- ▶ `HttpRequest` (extends `Request`): This class provides an API to access all pertinent information in a request.
- ▶ `HttpResponse` (extends `Response`): This class provides an API to create a response to a request and the application state.
- ▶ `HttpSession`: This class stores any information required to be stored across a user session with the application (as opposed to a single request).

- ▶ `RequestDispatcher`: Within a web application, redirecting the processing of a request to another servlet is required. This class provides methods to redirect the processing of a request to another servlet.

Other classes are available in the Java EE web components framework. For a full description of the classes that are available, see 18.7, “More information” on page 1055, which provides a link to the Java EE servlet specifications.

JSP

You can include any valid fragment of Java code in a JSP page and mix it with HTML. In the JSP file, the Java code is marked by `<%` and `%>`, and on deployment (or sometimes the first page request, depending on the configuration), the JSP is compiled into a servlet class. This process combines the HTML and the scriptlets in the JSP file in the `_jspService` method that populates the `HttpResponse` variable. Combining a lot of complex Java code with HTML can result in a complicated JSP file. Except for simple examples, avoid this practice.

One way around this situation is to use custom *JSP tag libraries*, which are tags defined by developers that initiate calls to a Java class within a JSP page. These classes implement `Tag`, `BodyTag`, or `IterationTag` interfaces from the `javax.servlet.jsp.tagext` package, which is part of the Java EE framework. Each tag library is defined by a `.tld` file that includes the location and content of the `taglib` class file. Although not strictly required, you need to include a reference to this file in the deployment descriptor.

XML-based tag files: JSP 2.0 introduces XML-based tag files. Tag files no longer require a `.tld` file. Tags can now be developed using JSP or XML syntax.

The most widely available tag library is the JavaServer Pages Standard Tag Library (JSTL), which provides simple tags to handle simple operations required in most JSP programming tasks, including looping, globalization, XML manipulation, and even processing of SQL result sets. The RedBank application uses JSTL tags to display tables and add URLs to a page. The final section of this chapter contains references to learn more about JSP and tag libraries.

18.1.2 Model view controller pattern

The model view controller (MVC) concept is a pattern used many times when describing and building applications with a user interface component, including Java EE applications.

Following the MVC concept, a software application or module must have its business logic (model) separated from its presentation logic (view). This separation is desirable, because it is likely that the view will change over time, and it might not be necessary to change the business logic each time. Also, many applications might have multiple views of the same business model, and if the view is not separated, adding an additional view causes considerable disruptions and increases the component's complexity.

You can achieve this separation through the layering of the component into a *model* layer (responsible for implementing the business logic) and a *view* layer (responsible for rendering the user interface to a specific client type). In addition, the *controller* layer sits between those two layers, intercepting requests from the view (or presentation) layer and mapping them to calls to the business model, then returning the response based on a response page selected by the controller layer. The key advantage provided by the controller layer is that the presentation can focus only on the presentation aspects of the application and leave the flow control and mapping to the controller layer.

You can achieve this separation in Java EE applications in several ways. Various technologies, such as JSF and Struts, differ in the ways that they apply the MVC pattern. Our focus in this chapter is on JSP and servlets that fit into the view and controller layers of the MVC pattern. If only servlets and JSP are used in an application, the details of how to implement the controller layer are left to whatever mechanism the development team decides is appropriate and that they can create using Java classes.

In the example later in this chapter, a *command* pattern (see Eric Gamma, et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995, ISBN 0-201-63361-2) is applied to encompass the request to the business logic and interactions made with the business logic through a facade class. In the other interactions, the request is sent directly to a servlet that makes method calls through the facade.

18.2 Web development tooling

Rational Application Developer includes many web development tools for building static and dynamic web applications. Many of these web development tools focus on technologies, such as Web 2.0 technologies, portals, and JSF, which are described in other chapters. In this section, we highlight the following tools and features, which focus on the more fundamental aspects of web development:

- ▶ Web perspective and views
- ▶ Page Designer
- ▶ Page templates

- ▶ CSS Designer
- ▶ Security Editor
- ▶ File creation wizards

18.2.1 Web perspective and views

The Web perspective helps web developers build and edit web resources, such as servlets, JSP, HTML pages, style sheets images, and deployment descriptor files.

To open the Web perspective, from the workbench, select **Window** → **Open Perspective** → **Web**. Figure 18-4 shows the default layout of the Web perspective with a simple `index.html` that has been published to the local test service and is being viewed.

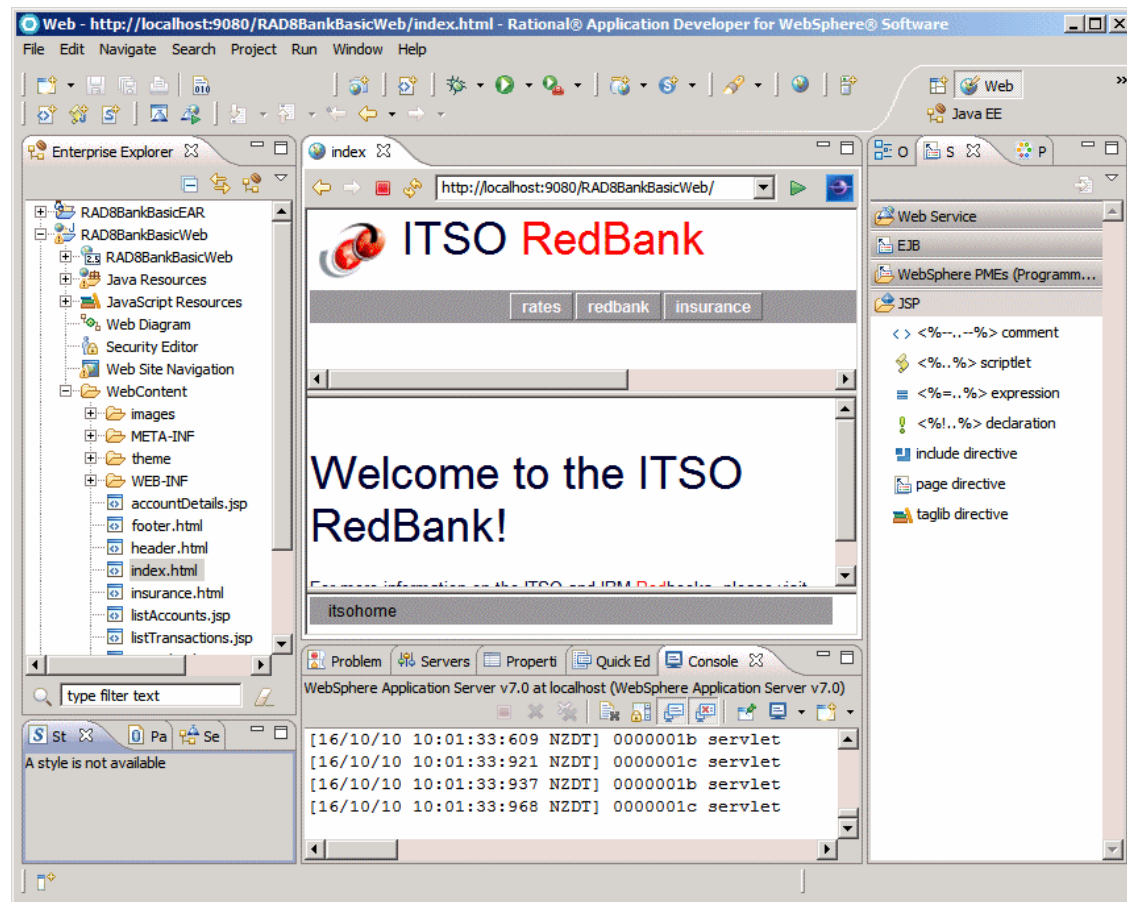


Figure 18-4 Web perspective in Rational Application Developer

In the Web perspective, many views are accessible (by selecting **Window** → **Show View**), several of which are already open in the Web perspective default setting.

By default, the following views are available in the Web perspective:

- ▶ **Console view:** This view shows output to SystemOut from any running processes.
- ▶ **Outline view:** This view shows an outline of the file being viewed. For HTML and JSP, this view shows a hierarchy of tags around the current cursor position. Selecting a tag in this view moves the cursor in the main view to the selected element. This view is particularly useful for moving quickly around a large HTML file.
- ▶ **Page Data view:** When editing JSP files, this view gives a list of any page or scripting variables available.
- ▶ **Page Designer view:** This view is a “what you see is what you get” (WYSIWYG) editor for JSP and HTML that consists of four tabs: Design (where the user can drag components to the page), Source (shows the HTML), Split (shows the Design in the top half and the Source in the bottom half), and Preview (shows how the final page looks in either Internet Explorer or Firefox).
- ▶ **Palette view:** When editing JSP or HTML files, this view provides a list of HTML items (arranged in drawers) that can be dragged and dropped onto pages.
- ▶ **Problems view:** This view shows ding errors, warnings, or informational messages for the current workspace.
- ▶ **Enterprise Explorer view:** This view shows a hierarchy view of all projects, folders, and files in the workspace. In the Web perspective, it structures the information within web projects in a way that makes navigation easier.
- ▶ **Properties view:** This view shows the properties for the item currently selected in the main editor.
- ▶ **Quick Edit view:** When editing HTML or JSP files, this view provides a mechanism to quickly add Java Script to a given window component on certain events, for example, `onClick`.
- ▶ **Servers view:** Use this view if you want to start or stop test servers while debugging.
- ▶ **Services view:** This view shows a summary of all the web services present in the projects on the workspace.
- ▶ **Snippets view:** In this view, you can edit small bits of code, including adding and editing actions assigned to tags. You can drag items from the Snippets view to the Quick Edit view.

- ▶ **Styles view:** With this view, you can edit and apply both pre-built and user-defined styles sheets to HTML elements and files.
- ▶ **Thumbnails view:** Given the selection of a particular folder in the Project Explorer view, this view shows the contents of the folder.

More information: For more information about the Web perspective and views, see 4.3.19, “Web perspective” on page 132.

18.2.2 Page Designer

The Page Designer is the primary editor within Rational Application Developer for building HTML, XHTML, JSP, and JSF source code. Web designers can drag and drop web page items from the Palette view to the desired position on the web page.

It provides the following representations of a page:

- ▶ The *Design tab* provides a WYSIWYG environment to visually design the contents of the page. A good development technique is to work within the Design tab of the Page Designer and build up the HTML contents by clicking and dragging items from the Palette view onto the page and arranging them with the mouse or editing properties directly from the Properties view. Tags can be positioned as absolute instead of relative.
- ▶ The *Source tab* provides access to the page source code showing the raw HTML or JSP contents. You can use the Source tab to change details that are not immediately obvious in the Design tab.
- ▶ The *Split tab* (Figure 18-5 on page 994) combines the Source tab and either the Design tab or the Preview tab in a split-screen view. The Split tab is helpful to see the Design tab and Source tab in one view; the changes are immediately reflected.

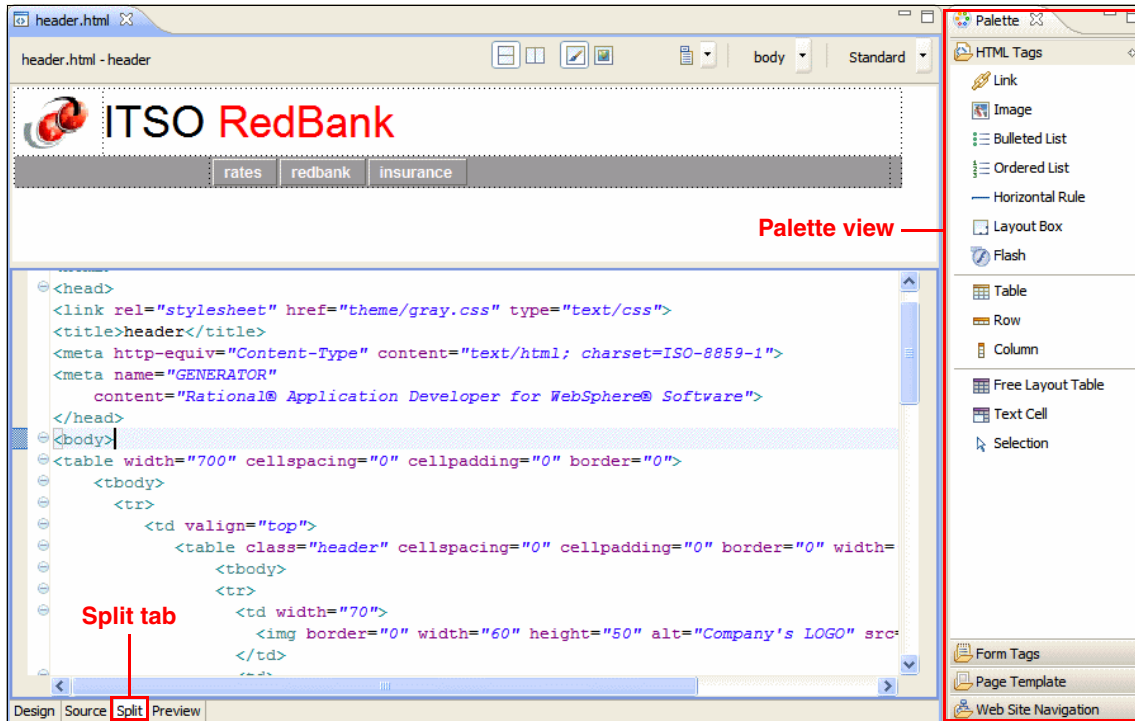


Figure 18-5 Page Designer Split tab

- ▶ The *Preview tab* shows how the page will look when it is displayed in a web browser. You can use the Preview tab throughout the process to verify the look of the final result in either Firefox or Internet Explorer.

Another important feature of Page Designer is the ability to automatically convert HTML elements to related elements. For example, there are menu options to convert Text entry fields to Password entry fields, which you can achieve from the context menu (**Convert Widget** → **HTML Form Widgets**) of the element in the design view (see Figure 18-6 on page 995). Rational Application Developer comments the original element for reference. Rational Application Developer supports conversions between selected HTML and Dojo widgets.

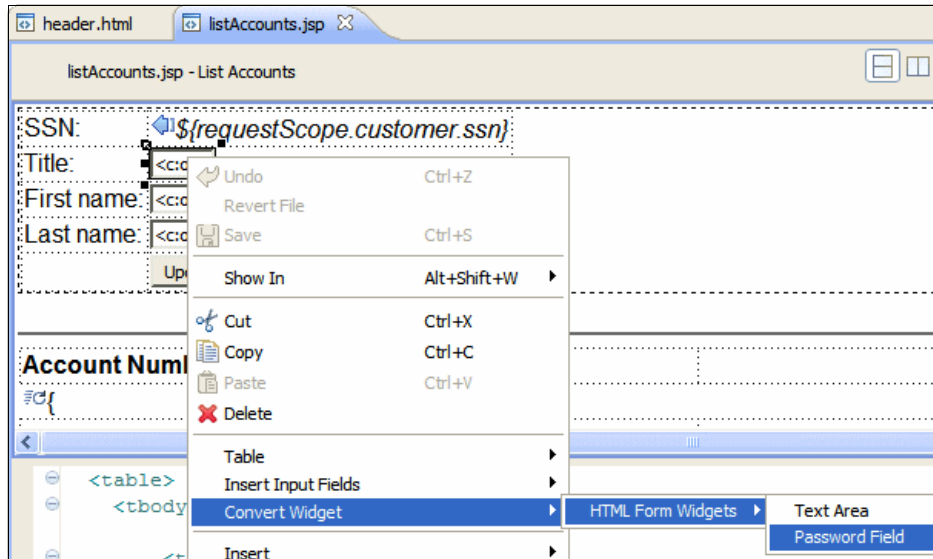


Figure 18-6 Convert Widget in Page Designer

Finally, HTML content is often provided to a development team and created from tools other than Rational Application Developer. These files can be imported by using the context menu on the target directory, selecting **File** → **Import** → **General** → **File System**, browsing to the new file, and clicking **Import**. When an imported file is opened in the Page Designer, all the standard editing features are available.

More information: For a detailed example of using the Page Designer, see 18.5.8, “Developing the static web resources” on page 1022, and 18.5.10, “Working with JSP” on page 1036.

18.2.3 Page templates

A *page template* contains common areas that you want to appear on all pages, and content areas that are intended to be unique on each page. Page templates are used to provide a common look and feel for a web project.

The Page Template File creation wizard is used to create these files. After being created, the file can be modified in the Page Designer. The page templates are stored as *.html files for HTML pages and *.jsp files for JSP pages. Changes to the page template are reflected in pages that use that template. Templates can be applied to individual pages, groups of pages, or applied to an entire web project, and they can be replaced or updated for a given website. Areas can be

marked as read-only, meaning that the Page Designer does not allow the developer to modify those areas, ensuring that certain designated areas are never changed accidentally.

When creating a new page template, you are prompted to choose whether the template will be a dynamic page template or a design-time template:

- ▶ *Dynamic page templates* use Struts-Tiles technology to generate pages on the web server.
- ▶ *Design-time templates* allow changes to be made and applied to the template at design or build time, but after an application is deployed and running, the page template cannot be changed. Design time templates do not rely on any technologies other than the standard Java EE servlet libraries.

18.2.4 CSS Designer

Cascading style sheets (CSS) are used with HTML pages to ensure that an application has consistent colors, fonts, and sizes across all pages. You can create a default style sheet when creating a new project. Several samples are included with Rational Application Developer.

At the start of the development effort, decide on the overall theme (color or fonts) for a web application and create the style sheet. Then as you create the HTML and JSP files, you can select that style sheet to ensure that web pages have a consistent look and feel. Style sheets are commonly kept in the WebContent/theme folder.

The CSS Designer is used to modify cascading style sheet (*.css) files. It provides two panels. On the right side, it shows all the text types and their respective fonts, sizes, and colors, which are all editable. On the left side, you see a sample of how the various settings will look. Any changes that are made are immediately applied to the design in the Page Designer if the HTML file is linked to the CSS file.

More information: For an example of customizing style sheets used by a page template, see 18.5.6, “Customizing a style sheet” on page 1019.

18.2.5 Security Editor

Rational Application Developer features the Security Editor, which provides a wizard to specify security groups within a web application and the URLs to which a group has access. With the Java EE specification, you can define, in the deployment descriptor, security groups and levels of access to defined sets of

URLs. The Security Editor (Figure 18-7) provides an interface for this information.

Selecting an entry in the Security Roles pane shows the resources members of that role in the Resources pane and the constraint rules that are applicable for the role and resource (if one is selected). Each entry in the Constraints window has a list of resource collections that specify the resources available to it and which HTTP methods can be used to access these resources. Using context menus, it is possible to create new roles and security constraints and to add resource collections to these constraints.

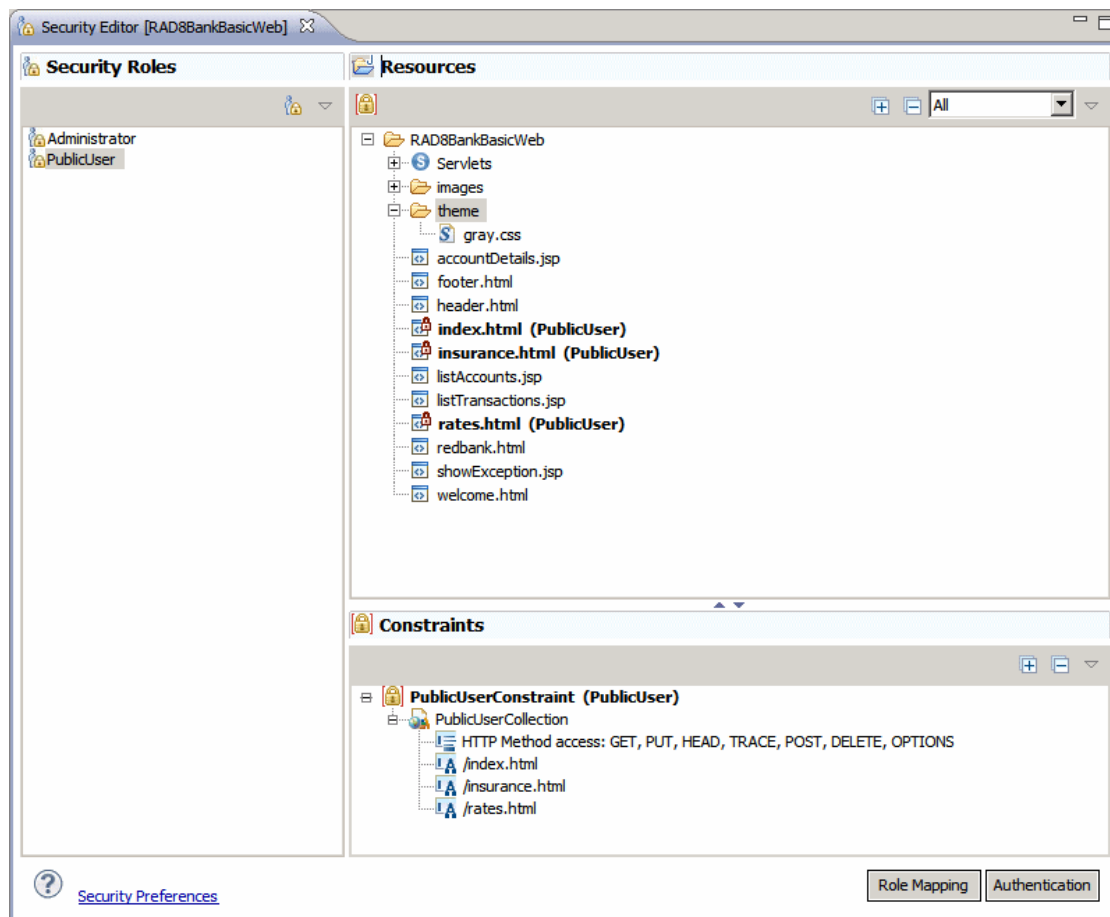


Figure 18-7 Security Editor

The Java EE security specification defines the mechanism for declaring groups and the URL sets that each group can access, but it is up to the Web container to map this information to an external security system. WebSphere's administrative

console provides the mechanism to configure an external Lightweight Directory Access Protocol (LDAP) directory. See the IBM Redbooks publication *Experience J2EE! Using WebSphere Application Server V6.1*, SG24-7297.

18.2.6 File creation wizards

Rational Application Developer provides many web development file creation wizards. You can access them by selecting **File** → **New** → **Other**. Then from the Select a wizard window, expand the **Web** folder and select the type of file required. These wizards prompt you for the key features of the new artifact and can help you to quickly get a skeleton of the component that you need. You can always manipulate the artifact created by the wizard directly, if necessary.

The following wizards are available in the Web perspective:

- ▶ **CSS:** You can use the CSS file wizard to create a new cascading style sheet (CSS) in a specified folder.
- ▶ **Dynamic Web Project:** This wizard steps you through the creation of a new web project, including which features the project uses and any page templates present.
- ▶ **Web Fragment Project:** This wizard guides you through the steps to make a web fragment project, including the link to the target dynamic web project.
- ▶ **Filter:** This wizard constructs a skeleton Java class for a Java EE filter, which provides a mechanism for performing processing on a web request before it reaches a servlet. The wizard also updates the `web.xml` file with the filter details.
- ▶ **Filter Mapping:** This wizard steps you through the creation of a set of URLs with which to map a Java EE filter. The result of this wizard is stored in the deployment descriptor.
- ▶ **HTML:** This wizard steps you through the creation an HTML file in a specified folder, with the option to use HTML Templates.
- ▶ **JSP:** This wizard steps you through the creation of a JSP file in a specified folder, with the option to use JSP Templates.
- ▶ **Listener:** You can use a listener to monitor and react to events in a servlet's or application's life cycle by defining methods that are invoked when life-cycle events occur. This wizard guides you through the creation of such a listener and to select the application life-cycle events to which to listen.
- ▶ **Security Constraint:** You can use this wizard to populate the `<security-constraint>` in the deployment descriptor that contains a set of URLs and a set of http methods, which members of a particular security role are entitled to access.

- ▶ **Security Role:** This wizard adds a `<security-role>` element to the deployment descriptor.
- ▶ **Servlet:** You can use this wizard to create a skeleton servlet class and add the servlet with appropriate annotations or add to the deployment descriptor.
- ▶ **Servlet Mapping:** This wizard steps you through the creation of a new URL to servlet mapping and adds appropriate annotations or information to the deployment descriptor.
- ▶ **Static Web Project:** This wizard steps you through building a new web project containing only static pages.
- ▶ **JSP Tag:** This wizard steps you through creating a new Tag library file.
- ▶ **Web Page:** By using this wizard, you can create an HTML or JSP file in a specified folder, with the option to create from a large number of page templates.
- ▶ **Web Page Template:** The Page Template File wizard is used to create new page template files in a specified folder. You can optionally create from a page template or create as a JSP fragment and define the markup language (HTML, HTML Frameset, Compact HTML, XHTML, XHTML Frameset, and Wireless Markup Language (WML) 1.3). You can select from one of the following models: Template containing Faces Components, Template containing only HTML, or Template containing JSP.

Dojo, Struts, and JSF: Several wizards are available in the Web perspective specifically for Dojo, widgets, and JSF, which we discuss in other chapters.

18.3 Rational Application Developer new features

Significant improvements to the previous version of Rational Application Developer have been made in the tools that are available for creating artifacts within a web application:

- ▶ **Support for JEE Servlet 3.0 specification:** The Servlet 3.0 specification revision is a major revision of the specification and includes the following changes:
 - **Use of annotations versus deployment descriptor:** Under the servlet 3.0 specification, servlets, servlet mappings, filters, and listeners can be declared by using annotations rather than being declared in the deployment descriptor. This feature reduces the amount of configuration required for each web application. The tooling support in Rational Application Developer has been changed to cater for declarations made in annotations or the deployment descriptor.

- Support for Web Fragment projects: A Web Fragment project is a new project type that allows logical partitioning of the web application in such a way that the utility projects or frameworks being used within the web application can define all the artifacts without requiring you to edit or add information in `web.xml`. There is a reference inside the web project to the Web Fragment project, which contributes to it. You do not need to update the deployment descriptor to declare the web fragment project used.
- Ability to store an EJB directly in a WAR: New to the Servlet 3.0 specification and Rational Application Developer is the ability to include EJBs directly in the WAR.
- Ability to programatically add and remove servlets: The Servlet 3.0 specification includes API changes to allow developers to add and remove servlets while an application executes. However, there are no specific features to achieve this capability in Rational Application Developer.
- ▶ HTML support: Page Designer does not support the new elements and attributes that are defined in the HTML5 specification. HTML5 is only supported in our source editors. Page Designer support for earlier versions of HTML remains. Elements, such as frameset, and the height/width, spacing, and padding attributes within tables are not recommended in HTML5. If these elements and attributes are used, Page Designer underlines the attribute or element and raises a warning.
- ▶ Page Designer enhancements:
 - The preview tab now includes options for previewing a web page in Internet Explorer or Mozilla Firefox.
 - There is a feature to convert HTML widgets to related widgets through a context menu.
 - Users can show all pages affected by a CSS style rule. When a CSS file is edited, users can see the affected HTML or JSP files in the search view and decide whether to keep the modified style rule or not.
 - There is support for Scalable Vector Graphics (SVG) images and Flash widgets in Page Designer.
 - *JSP fragment files* (JSPF) are a mechanism to break up JSP pages into reusable blocks (fragments) that can be assembled on a standard JSP page. Page Designer now includes additional tooling to understand how a fragment will be used when it is included in separate contexts.

18.4 RedBank application design

In this section, we describe the design for the ITS0 RedBank web application. We outline the design of the RedBank application and explain how it fits into the Java EE web framework, particularly with regard to JSP and servlets.

18.4.1 Model

The model for the RedBank project is implemented by using a simple Java project and exposed to other components through a facade interface (called ITS0Bank). The main ITS0Bank object is a Singleton object, accessible by a single static public method called getBank.

The ITS0Bank object is composed of the other business objects that make up the application, including Customer, Account, and Transaction. The facade into the bank object includes methods, such as getCustomer, getAccounts, and withdraw, deposit, and transfer. Figure 18-8 shows a simplified Unified Modeling Language (UML) class diagram of the model. We describe the model in detail in Chapter 7, “Developing Java applications” on page 229.

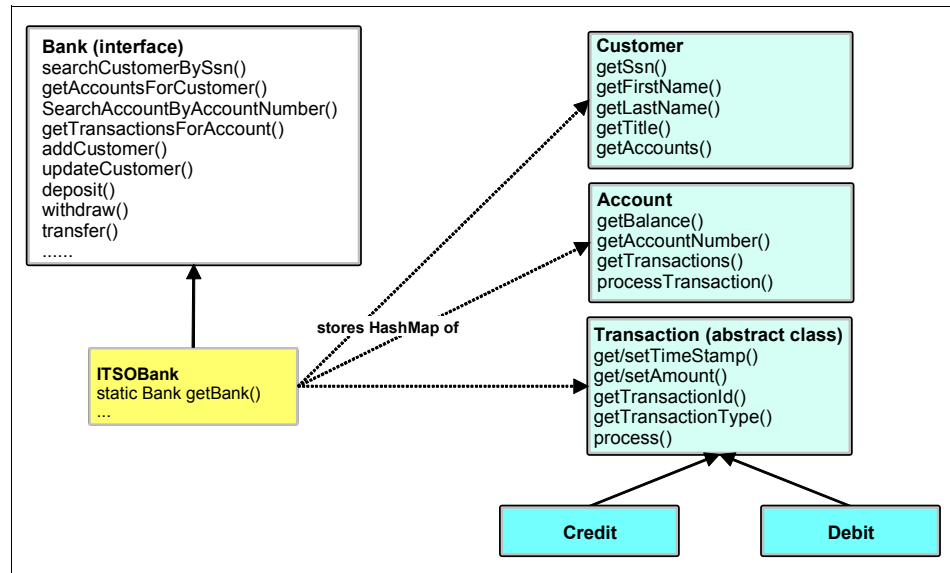


Figure 18-8 Class diagram for RedBank model

The underlying technology to store data that is used by the ITS0Bank application involves Java HashMaps. These Java HashMaps are populated at startup in the constructor, and obviously, the data is lost every time that the application is

restarted. In an actual client example, the data might be stored in a database, but for the purposes of this example, HashMaps are acceptable. In Chapter 12, “Developing Enterprise JavaBeans (EJB) applications” on page 577, the ITS0Bank model is modified to run as EJB and Java Persistence API (JPA) entities, and the application data is stored in a database.

18.4.2 View layer

The view layer of the RedBank application consists of four HTML files and four JSP files. The application home page is the `index.html` file that contains a link to four HTML pages (the `welcome.html`, `rates.html`, `insurance.html`, and `redbank.html` pages). The `welcome.html`, `rates.html`, and `insurance.html` pages are simple static HTML pages that show information without forms or entry fields.

The `redbank.html` page contains a single form in which a user can type the customer ID to access customer services, such as accessing a balance and performing transactions. Although the account number is verified, we do not cover security issues (logon and password) in this example.

From the `redbank.html` page, the user sees the `listAccounts.jsp` page, which shows the customer’s details, a list of accounts, and a button to log out.

Selecting an account opens the `accountDetails.jsp` page, which shows the balance for the selected account and a form through which a transaction can be performed. This page also shows the current account number and balance, which are both dynamic values. A simple JavaScript code controls whether the amount and destination account fields are available, depending on the option selected. One of the transaction options on the `accountDetails.jsp` page is List Transactions, which invokes the `listTransactions.jsp` page.

If anything goes wrong in the regular flow of events, the exception page (`showException.jsp`) is displayed to inform the user of the error.

The `listAccounts.jsp`, `accountDetails.jsp`, `listTransactions.jsp`, and `showException.jsp` JSP pages make up the dynamic pages of the RedBank application.

18.4.3 Controller layer

The controller layer was implemented using two strategies, one straightforward strategy and one complex strategy, which is more applicable to an actual client situation.

The application has the following servlets:

- ListAccounts** Gets the list of accounts for one customer.
- AccountDetails** Shows the account balance and the selection of operations: list transactions, deposit, withdraw, and transfer.
- Logout** Invalidates the session data.
- PerformTransaction** Performs the selected operation by calling the appropriate control action: ListTransactions, Deposit, Withdraw, or Transfer.
- UpdateCustomer** Updates the customer information.

The first three servlets use a simple function call from the servlet to the model classes to implement their controller logic and then use `RequestDispatcher` to forward control to another JSP or HTML resource. Figure 18-9 shows the pattern that is used in the sequence diagram.

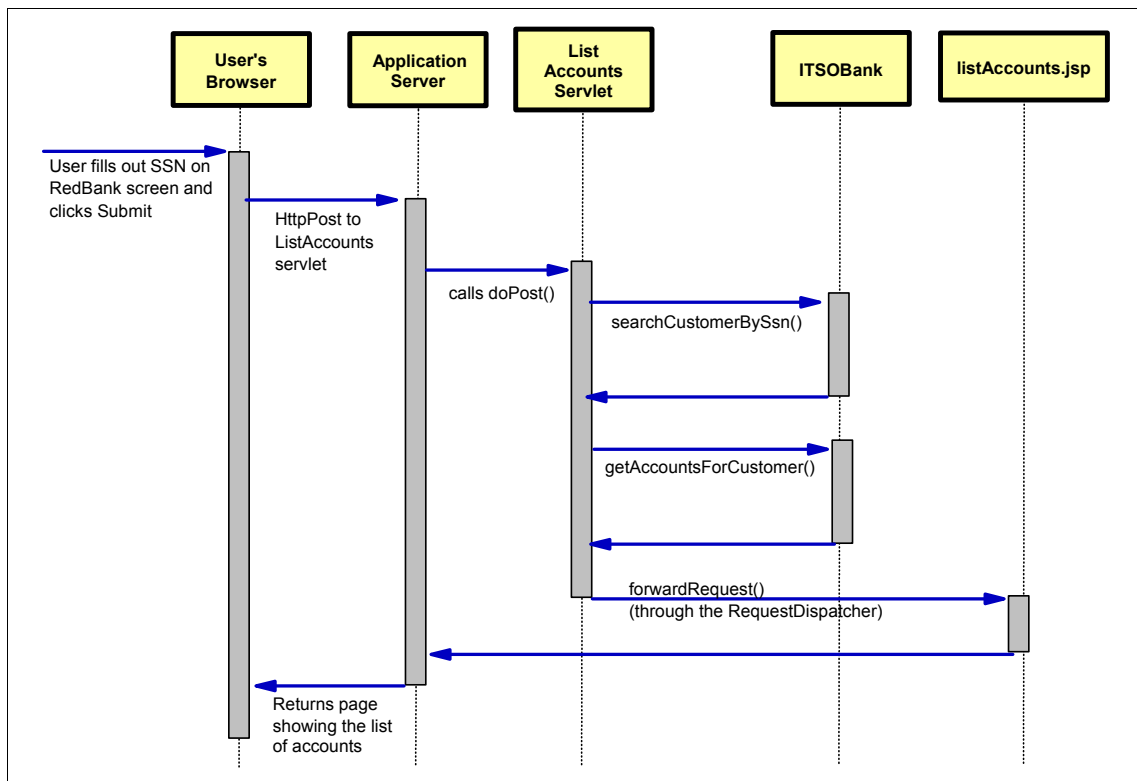


Figure 18-9 ListAccounts sequence diagram

PerformTransaction uses a separate implementation pattern. It acts as a front controller, receiving the HTTP request and passing it to the appropriate control action object. These objects are responsible for carrying out the control of the application. Figure 18-10 shows a sequence diagram for the list transaction operation from the account details page, including the function calls through PerformTransaction, the ListTransactionsCommand class, onto the model classes, and forwarding to the appropriate JSP.

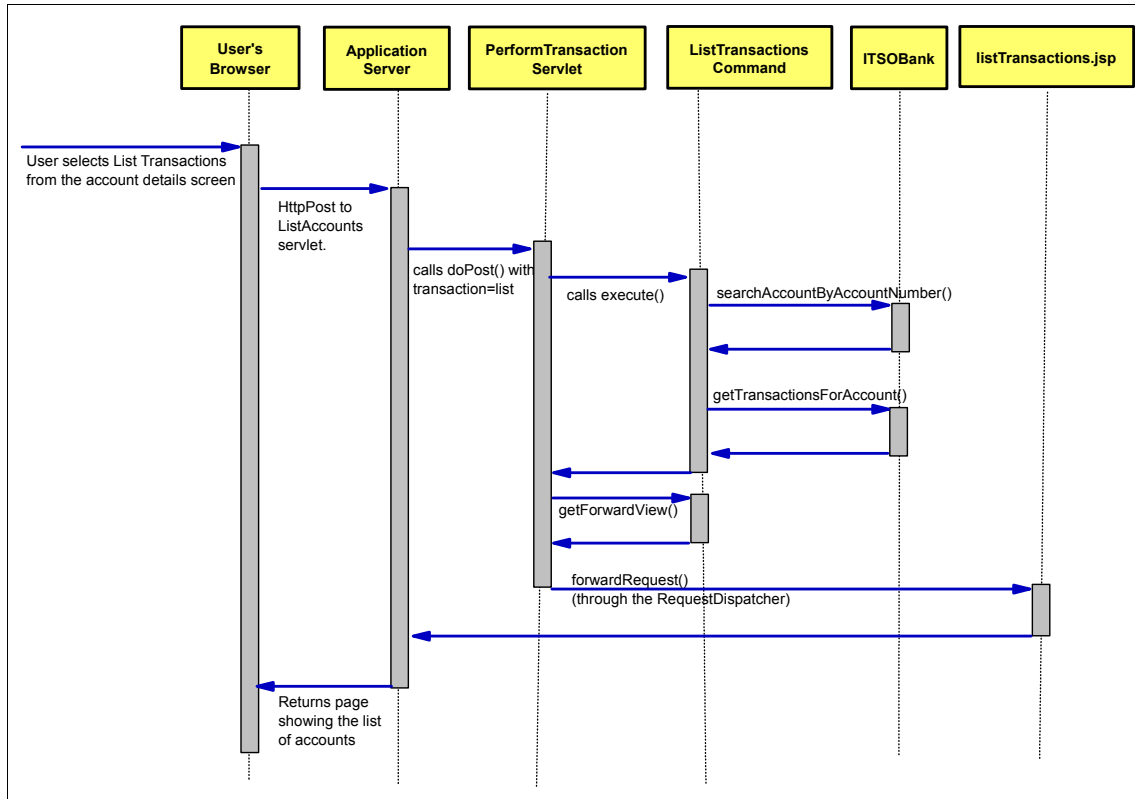


Figure 18-10 PerformTransaction sequence diagram

The Struts framework provides a much more detailed implementation of this strategy and in a standardized way.

Action objects: Action objects, or commands, are part of the Command design pattern. For more information, see Eric Gamma, et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995, ISBN 0-201-63361-2.

18.5 Implementing the RedBank application

In this section, we use an example to introduce you to the tools within Rational Application Developer that facilitate the development of web applications. In the example, we create separate web artifacts (including page templates, HTML, JSP, and servlets) and demonstrate how to use the available tools.

The section is organized in the following way:

- ▶ Creating the web project
- ▶ Importing the Java RedBank model
- ▶ Defining the empty web pages
- ▶ Creating frameset pages
- ▶ Customizing frameset web page areas
- ▶ Customizing a style sheet
- ▶ Verifying the site navigation and page templates
- ▶ Developing the static web resources
- ▶ Developing the dynamic web resources
- ▶ Working with JSP

At the end of this section, the RedBank application will be ready for testing.

18.5.1 Creating the web project

The first step is to create a web project in the workspace.

Enabling web capabilities: Before you begin, ensure that web capabilities are enabled. Select **Windows** → **Preferences**, expand **General** → **Capabilities**, and ensure that the **Web Developer** options (including basic, typical, and advanced) are selected.

Two types of web projects are available in Rational Application Developer: static and dynamic. Static web projects contain static HTML resources and no Java code and thus are comparatively simple. To demonstrate as many features of Rational Application Developer as possible, and because the RedBank application contains both static and dynamic content, we use a dynamic web project for this example.

In Rational Application Developer, perform the following steps:

1. Select **Window** → **Open Perspective** → **Web** to open the Web perspective.
2. To create a new web project, select **File** → **New** → **Dynamic Web Project**.

3. In the New Dynamic Web Project window (Figure 18-11 on page 1007), enter the following items:
 - a. In the Name field, type `RAD8BankBasicWeb`.
 - b. For Project location, select **Use Default** (default). This setting specifies where to place the project files on the file system. It is also acceptable to use the default option of leaving them in the workspace.
 - c. The Target Runtime option shows the supported test environments that have been installed. Select **WebSphere Application Server v8.0 Beta**.

Target Server Environment: Although it is possible to re-create the sample using WebSphere Application Server V7.0 Beta, the sample code provided in this chapter does not work if WebSphere Application Server V7 is chosen. It is written in Servlet Version 3.0, which is only supported by WebSphere Application Server V8 Beta and later.

- d. For the Dynamic Web Module version, select **3.0**.
- e. For Configuration, define the configuration as the last item.
- f. For the EAR Membership, select **Add module to an EAR** (default). Dynamic web projects, such as the one we are creating, run exclusively within an enterprise application. For this reason, you must either create a new EAR project or select an existing project.
- g. For the EAR Project Name, enter `RAD8BankBasicWebEAR` (the default). Because we select **Add module to an EAR**, the wizard creates a new EAR project.

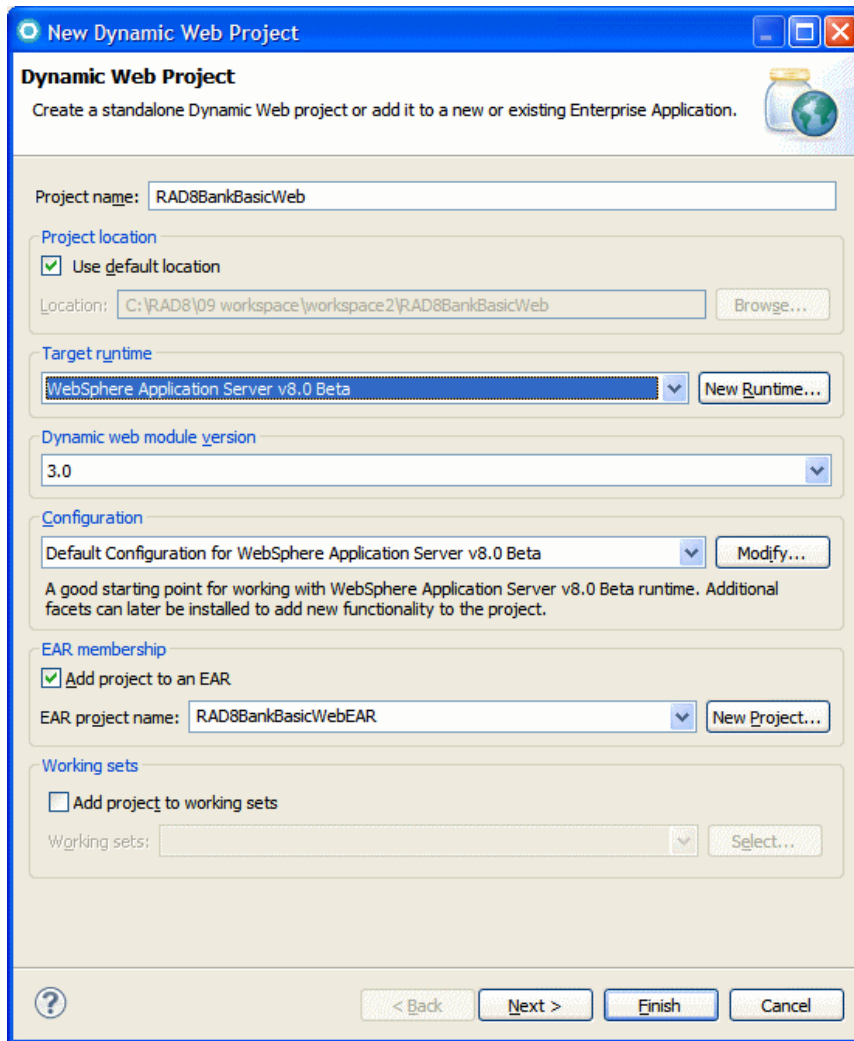


Figure 18-11 New Dynamic Web Project window

- h. For the Configuration section, click **Modify**.
4. In the Project Facets window (Figure 18-12 on page 1008), select the additional features **Default Style Sheet** and **JSTL**. Click **OK** and the configuration changes to <custom>.

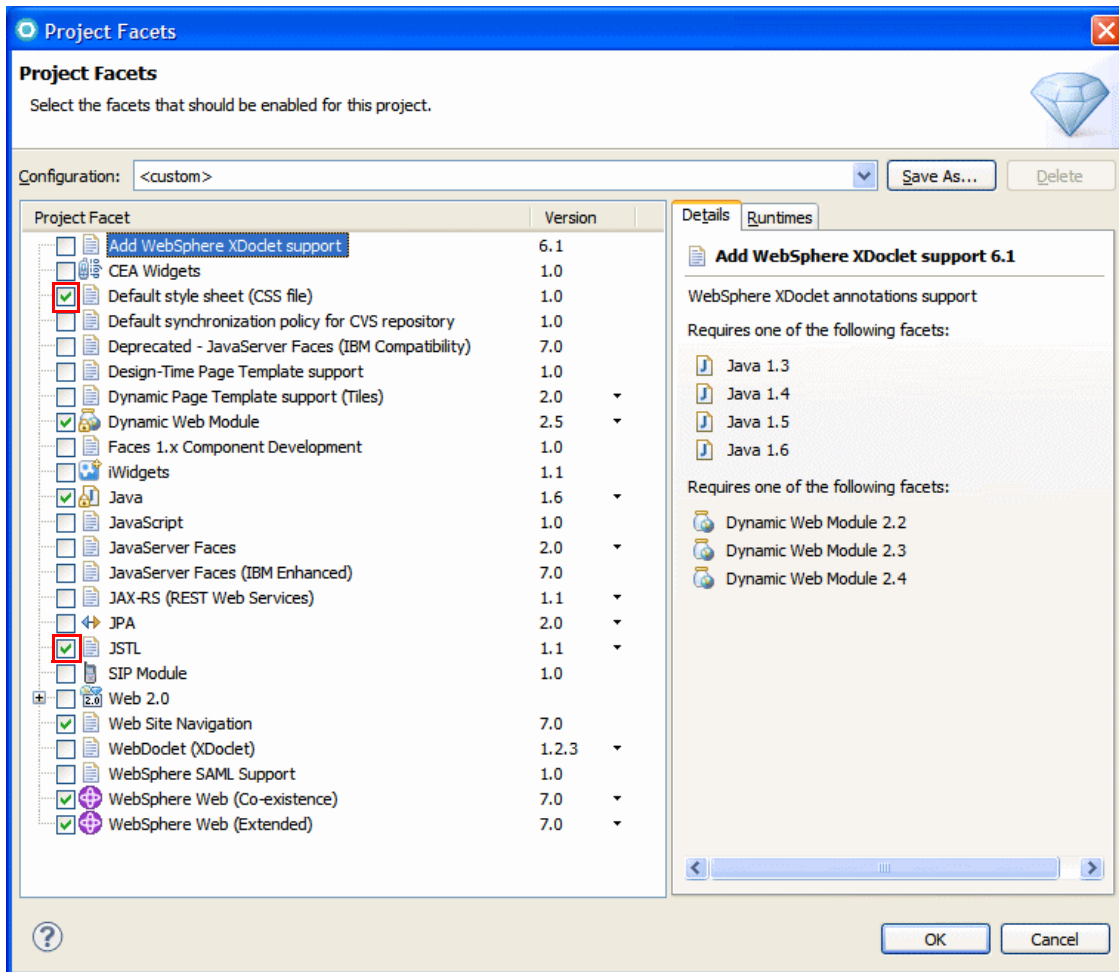


Figure 18-12 Dynamic Web Project facets

Tips:

- ▶ For the options that have a down arrow, you can alter the underlying version of the feature that is selected. By default, the latest version that is available is selected.
- ▶ From this window, to save the configuration for future projects, you can click **Save** and enter a configuration name and a description.

5. Click **Next**. In the Java window for the New Dynamic Web Project (Figure 18-13 on page 1009), accept the value src.

This action specifies the directory where any Java source code used by the web application is stored. The default value of `src` is sufficient for most cases.

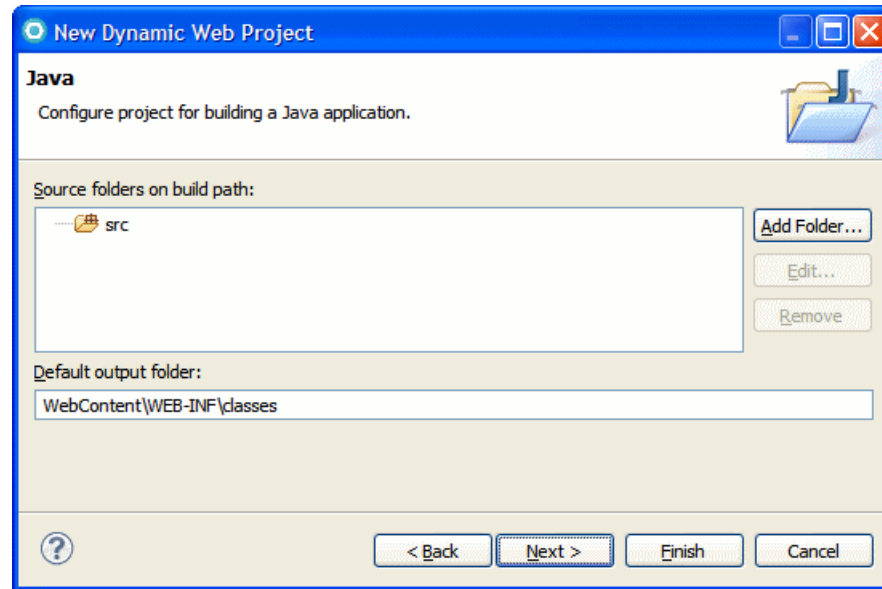


Figure 18-13 New Dynamic Web Project: Java window

6. In the Web Module window (Figure 18-14 on page 1010), accept the following default options:
 - a. For the Context Root, accept the value `RAD8BankBasicWeb`.

The *context root* defines the base of the URL for the web application. The context root is the root part of the URI under which all the application resources are going to be placed, and by which they are referenced later. It is also the top-level directory for your web application when it is deployed to an application server.
 - b. For the Content Directory, accept the value `WebContent`.

This value specifies the directory where the files intended for the WAR file are created. All the contents of the `WEB-INF` directory, HTML, JSP, images, and any other files that are deployed with the application are contained under this directory. Usually, the folder `WebContent` is sufficient.
 - c. Select **Generate web.xml deployment descriptor** to create the `web.xml` file and IBM extensions.
 - d. Click **Finish**.

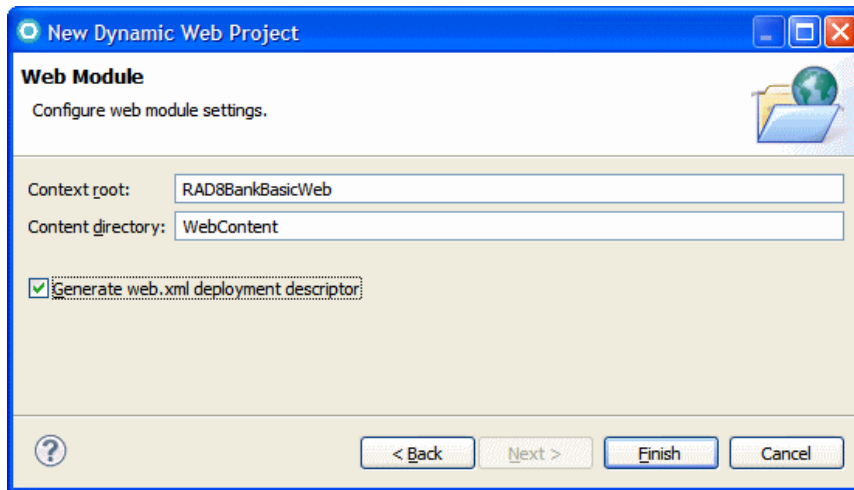


Figure 18-14 New Dynamic Web Project: Web Module window

The Technology Quickstart window opens. You can browse the Help topics for the features. When you finish, close the Technology Quickstart.

Figure 18-15 on page 1011 shows the web project directory structure for the newly created RAD8BankBasicWeb project and its associated RAD8BankBasicEAR project (enterprise application).

The following folders are shown under the web project:

- ▶ Deployment Descriptor
This folder shows an abstracted view of the contents of the project's web.xml file. It includes sub-folders for the major components that make up a web project configuration, including servlets and servlet mappings, filters and filter mappings, listeners, security roles, and references.
- ▶ Web Site Navigation
Clicking this folder starts the tool for editing the page navigation structure.
- ▶ Java Resources: src
This folder contains the Java source code for regular classes, JavaBeans, and servlets. When resources are added to a web project, they are automatically compiled, and the generated class files are added to the WebContent\WEB-INF\classes folder.

► WebContent

This folder holds the contents of the WAR file that is deployed to the server. It contains all the web resources, including compiled Java classes and servlets, HTML files, JSP, and graphics needed for the application.

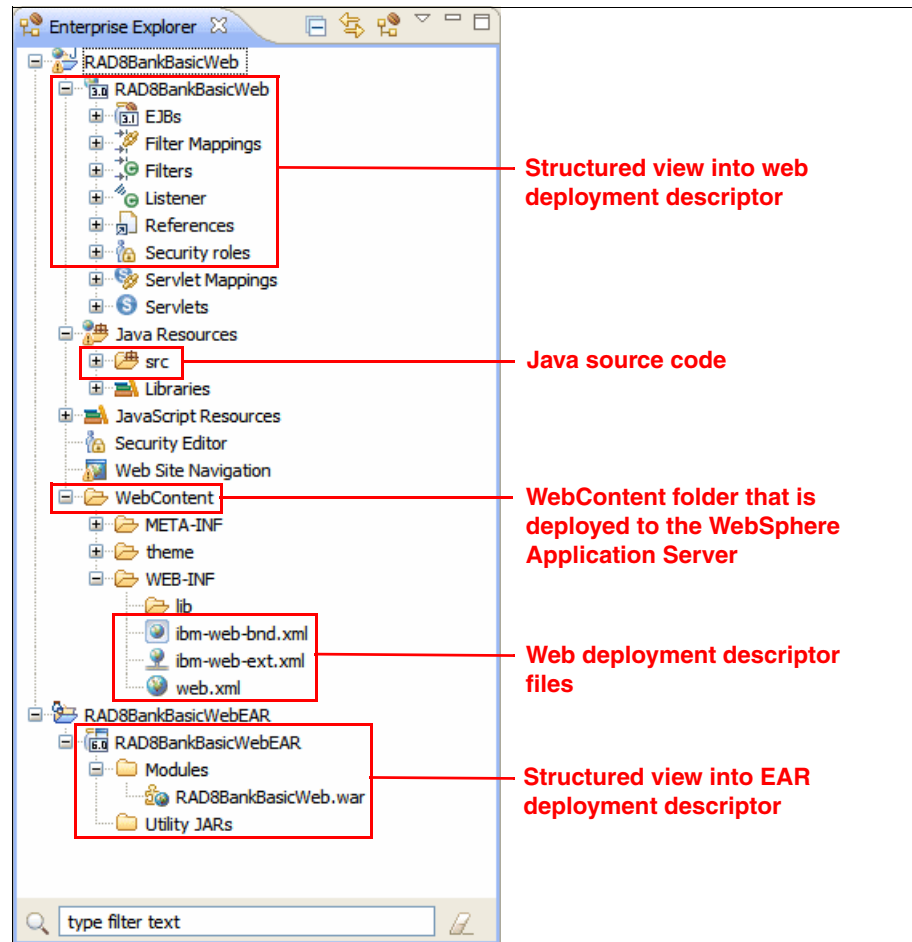


Figure 18-15 Web project directory structure

Important: Files that are not under WebContent are not deployed when the web project is published. Typically, these files include Java source and SQL files. Make sure that you place everything that is to be published under the WebContent folder.

18.5.2 Importing the Java RedBank model

The ITS0Bank web application requires the classes that we created in Chapter 7, “Developing Java applications” on page 229. In this section, we explain how to import the project files. If you already have the final project in the workspace, you can skip this import.

To import the RedBank model, follow these steps:

1. Locate the file `c:\7835codesolution\jsp\RAD80Java.zip`.
2. From the Import menu option (**File** → **Import**), select **General** → **Existing Projects into Workspace** and click **Next**.
3. Choose the **Select archive file** option that is selected and click **Browse**. Navigate to `RAD80Java.zip` and click **OK**.
4. Click **Finish**. The Java project is imported into the workspace.

To verify that the model is running, you can run the main method of the `itso.rad80.bank.client.BankClient` class, which invokes the bank facade classes directly, makes simple transactions directly, and prints the results to the console.

We add a reference to the `RAD80Java` project in the `RAD8BankBasicWeb` web application in “Adding the `RAD80Java` JAR to the web project” on page 1027.

18.5.3 Defining the empty web pages

In this section, we show how to define the skeleton pages for both the static and dynamic web pages that make up the RedBank application. We use the New Web Page wizard. When we finish, we will have a working application that will demonstrate the page navigation of the application.

We create an HTML or JSP page for each of the rows shown in Table 18-1.

Table 18-1 Web pages of the RedBank application

HTML or JSP file	HTML version
<code>welcome.html</code>	5
<code>index.html</code>	4.01
<code>rates.html</code>	5
<code>insurance.html</code>	5
<code>redbank.html</code>	5

HTML or JSP file	HTML version
listAccounts.jsp	5
accountDetails.jsp	5
listTransactions.jsp	5
showException.jsp	5

Importing web resources for the RedBank application

Prior to creating the web pages, we must import resources to provide the correct look and feel for web pages used in our example, such as images and CSS files. To import the resources, follow these steps:

1. Expand **RAD8BankBasicWeb** → **WebContent**, and from the context menu, select **Import**.
2. Select **General** → **File System** and click **Next**.
3. In the From directory, type `c:\7835code\webapp`, select the **images** and **theme** folders and click **Finish**.

The `itso_logo.gif` and `c.gif` images and the `gray.css` file are imported.

Defining the empty HTML pages

To create the `welcome.html` web page, complete the following steps:

1. From the Menu, select **New** → **Web Page**.
2. In the New Web Page window, enter the following values:
 - a. For the File name, type `welcome.html`.
 - b. For the Folder, type `/RAD8BankBasicWeb/WebContent`.
 - c. For the Template, expand **Basic Templates** and select **HTML/XHTML**.
 - d. Configure the document markup options by clicking **Options**. Ensure that the Markup Language is set to **HTML**. Ensure that the Document Type is set to **HTML 5**.
 - e. Select **Style Sheets**, enter `/theme/gray.css` as the style sheet, and remove the `Master.css`.
3. Click **Close** and then click **Finish** to create the HTML page.
4. If the Split view does not happen automatically, open the new file in the Page Designer view and move to the **Split** view.

5. Locate the title tag and change the value stored inside it to this value:

```
<title>ITS0 Home</title>
```

The title that is shown in the top half reflects the new title.

6. Locate the body tag and enter text to identify the page:

```
<body>Welcome to Redbank</body>
```

6) Save the web page.

Complete the previous steps to create HTML pages for the other HTML pages (rates.html, insurance.html, and redbank.html).

Defining the empty JSP pages

To create the listAccounts.jsp web page, complete the following steps:

1. From the top Menu, select **New** → **Web Page**.
2. In the New Web Page window, complete the following actions:
 - a. For the File name, type listAccounts.jsp.
 - b. For the Template, expand **Basic Templates** and select **JSP**.
 - c. Click **Options**. Verify the gray.css cascade style sheet and that the document type is **HTML 5**.
 - d. Click **Close**.
3. Click **Finish** to create the page. Close the editor that opens.

Complete the previous steps to create HTML pages for the other JSP pages (accountDetails.jsp, listTransactions.jsp, and showException.jsp).

Important: The spelling and capitalization of the JSP file names must be typed exactly as shown.

18.5.4 Creating frameset pages

The RedBank user interface (view) is made up of a combination of static HTML pages and dynamic JSP. In this section, we explain how to create an HTML frameset page (index.html) that will define the layout of the web pages created in 18.5.3, “Defining the empty web pages” on page 1012.

index.html: By default, a web server looks for the index.html (or index.htm) page when a web project is run. Although you can change this behavior, use index.html as the top-level page name.

Frameset pages provide an efficient method for creating a common layout for the user interface of the web application. A frameset page has the same structure as a table, where the rows are defined in a tag element called `<frameset>` and the columns are individually defined in a tag element called `<frame>`. An alternative approach is to use Page Templates for which Rational Application Developer also has tooling support.

In our example, we only have three areas and no column separations:

- ▶ Header area: With the company logo and the navigation bar to other pages
- ▶ Workspace area: Where the rest of the operational pages are displayed
- ▶ Footer area: With the option to return to the main menu

Creating an HTML frameset page

To create a new HTML frameset page (`index.html`) to use for the RedBank layout, perform the following steps:

1. Right-click **WebContent** and select **New** → **Web Page**.
2. In the New Web Page window (Figure 18-16 on page 1016), enter the following values:
 - a. For the File name, type `index.html`.
 - b. For the Template, select **Basic Templates** → **HTML/XHTML**.
 - c. Click **Options**. In the New Web Page Options window, for the Markup Language, select **XHTML Frameset**. For the **Document Type**, select **HTML 4.01 Frameset**. Click **Close**.

Important: In this example, the `index.html` page is HTML 4.01, and all other pages are HTML5. This is to demonstrate the support for both standards and also because the `index.html` page uses frameset tags. Frameset tags are not recommended for HTML5. There is no support for building these frameset tags in Rational Application Developer.

-
-
-
- d. Click **Finish** to create the `index.html` frameset.

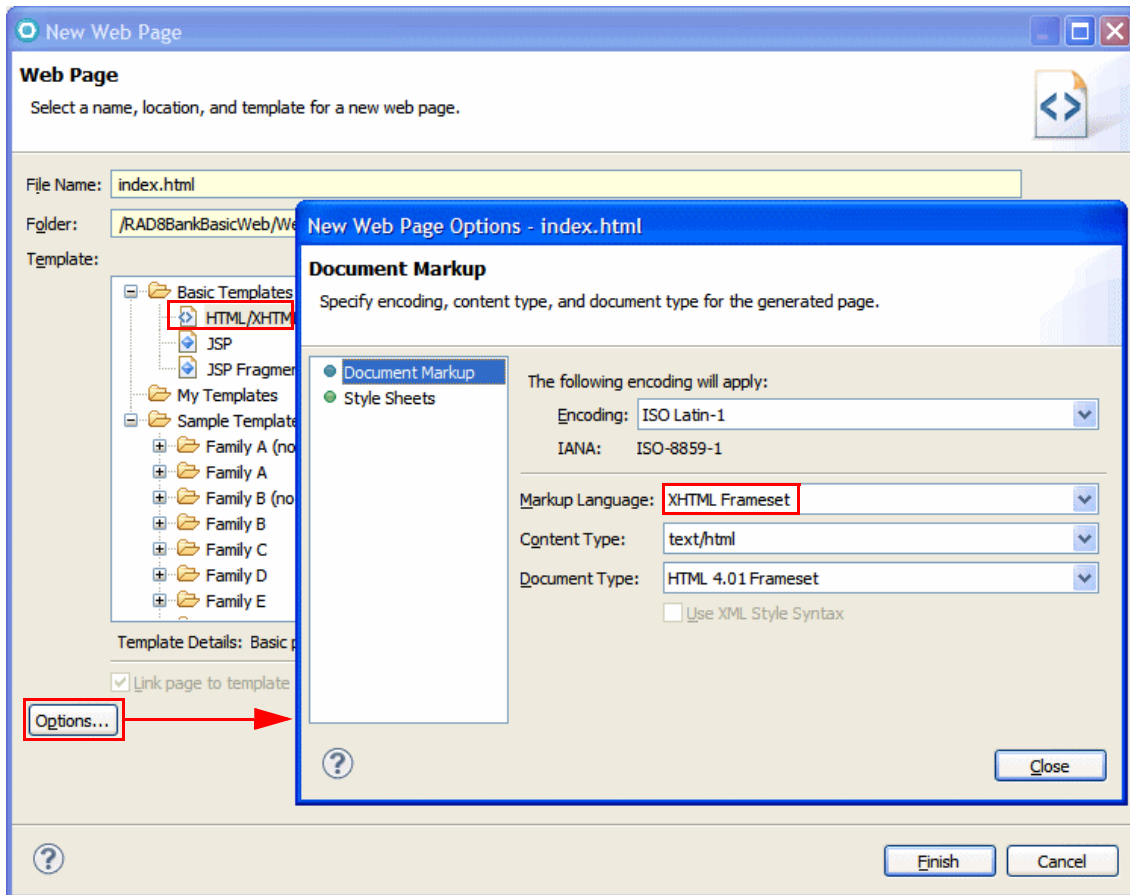


Figure 18-16 Creating a frameset page

Creating an HTML header for all web pages

Create the static HTML web page for the header area that will show the logo and heading information:

1. Right-click **WebContent** and select **New** → **Web Page**.
2. In the New Web Page window, enter the following values:
 - a. For the File name, type header.html.
 - b. For the Template, select **Basic Templates** → **HTML/XHTML**.
 - c. Click **Options**. In the New Web Page Options window, for the Markup Language, select **HTML**. For the Document Type, select **HTML 5** and click **Close**.
 - d. Click **Finish** to create the static header.html.

3. Import the code for the logo, title, and action bar into the `header.html` file:
 - a. Locate the `c:\7835code\webapp\html\SnippetForHeaderHTML.txt` file and open it in a simple text editor (for example, Notepad).
 - b. Open the **header.html** file in the Page Designer and select the **Source** tab.
 - c. Paste the code from `SnippetForHeaderHTML.txt` between the `<body>` and `</body>` tags.
 - d. Save the `header.html` file. Select the **Preview** tab to verify that the page has the ITSO RedBank text and logo as desired.

Warnings: The sample provided uses tags that are invalid for HTML5, and Rational Application Developer generates warnings indicating that these attributes and tags are obsolete. To be fully HTML5-compliant, refactor this information. However, these warnings do not affect the operation of the RedBank application, because HTML5-compliant web browsers are backward-compatible.

Creating an HTML footer for all web pages

Create the web page that holds the source of the footer area in the same way as for the header page:

1. Create a new web page named `footer.html` under `WebContent`.
2. Copy and paste the code from `SnippetForFooterHTML.txt` between the `<body>` and `</body>` tags.
3. Save the `footer.html` file. Select the **Preview** tab to see the newly created link.

18.5.5 Customizing frameset web page areas

Next we add frame references to the pages that are part of the user interface frame areas. We explain how to customize the following elements of the HTML page template (`index.html`, `header.html`, and `footer.html`).

Defining the areas in the frameset

To create the mentioned areas/frames in the frameset and link the areas with the previously created `header.html` and `footer.html` web pages, follow these steps:

1. In the Enterprise Explorer, expand **RAD8BankBasicWeb** → **WebContent** and open the **index.html** web page.
2. In the Page Designer, select the **Split** tab to work simultaneously with the source code and interface design.

3. To define the frameset areas, add a rows attribute to the frameset tag by following these steps:
 - a. In the Outline view, right-click the **frameset** tag and select **Add Attribute** → **New Attribute**.
 - b. In the New Attribute window, for the Name, type rows. For the value, type 20%,70%,10% and click **OK**.
 - c. Verify in the **Source** tag the value `<frameset rows="20%,70%,10%">`.
4. Link the header area with the header.html web page by following these steps on index.html:
 - a. In the Outline view, expand **html** → **frameset** and select the **frame** node.
 - b. In the Properties view (Figure 18-17), select the following attributes on the frame tab:
 - i. For the URL, type header.html. Or, click the **Browse** icon (📁) and select **File**. Then in the File Selection window, select **header.html** and click **OK**.
 - ii. For the Frame name, type headerArea.
 - iii. Leave the rest of the values by default, and save the changes.

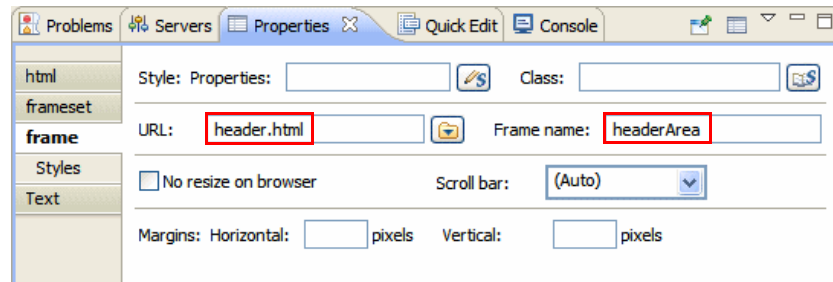


Figure 18-17 Frameset properties for index.html

- c. On the Split tab of the Page Designer, verify that the `<frame>` code was replaced by `<frame src="header.html" name="headerArea">`.
5. To link the workspace area, create a new frame and link it to the welcome.html web page by using the following steps:
 - a. In the Outline view, right-click the **frame** tag and select **Add After** → **frame**.
 - b. In the Properties view, set the URL to **welcome.html** and the Frame name to **workspaceArea**.
6. To link the footer area, create a new frame and link it to the **footer.html** web page and frame name **footerArea**.

7. From the **Design** tab, click the **Show frames** icon () to see the frames in position (Figure 18-18).

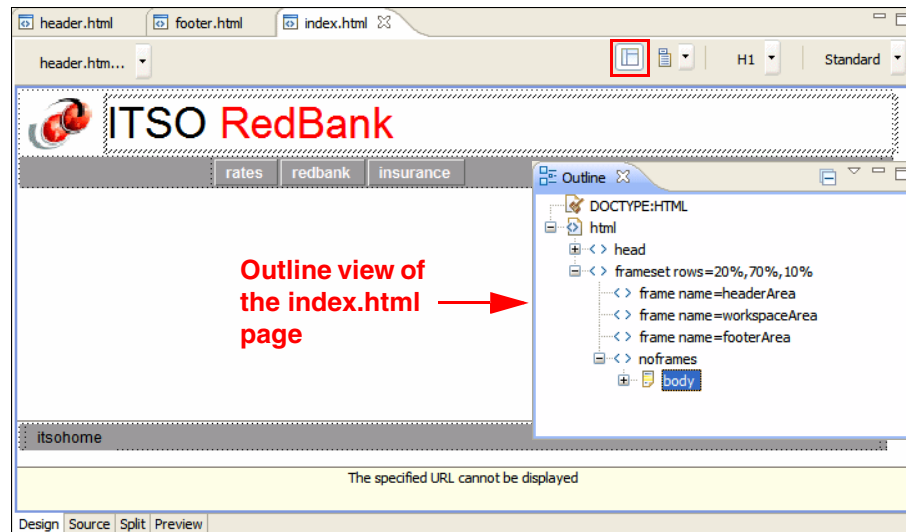


Figure 18-18 Frame design

18.5.6 Customizing a style sheet

You can create style sheets when a web project is created (by selecting the **Default style sheet (CSS File)** option in the Project Facets window), when a page template is created from a sample, or at any time by starting the CSS File creation wizard.

In the RedBank example, a style sheet named `gray.css` was imported as part of the process of “Importing web resources for the RedBank application” on page 1013. Both the HTML and the JSP web pages that we created reference the `gray.css` style sheet to have a common appearance of fonts and colors.

In the following example, we customize the colors that are used on the navigation bar links when you hover over a link. By default, the link text in the navigation bar is orange (`#cc6600`) when hovering. We customize this color to red (`#ff0000`).

To customize the `gray.css` style sheet, follow these steps:

1. Open the **theme/gray.css** file in the CSS Designer (Figure 18-19 on page 1020).

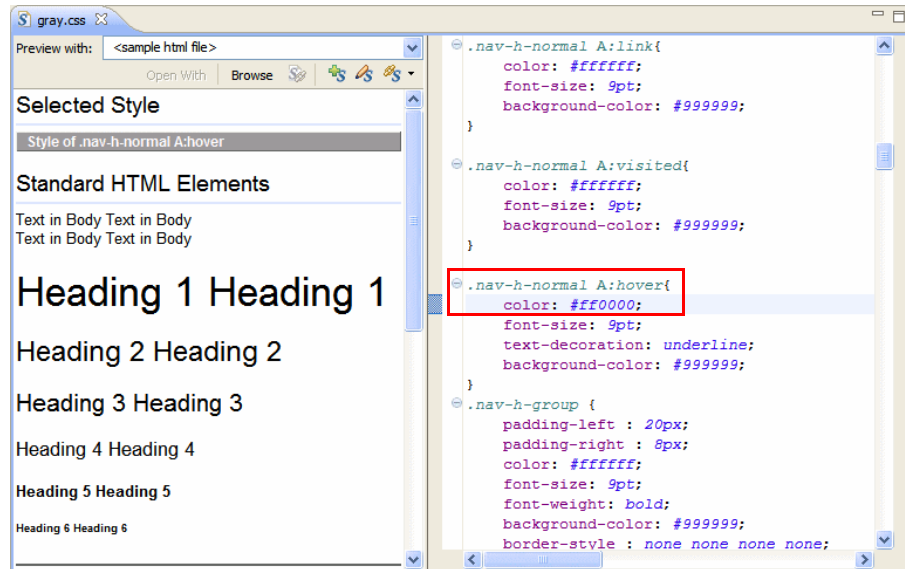


Figure 18-19 CSS Designer: gray.css

By selecting the text style **.nav-h-normal A: hover** in the right pane (scroll down to locate the style, or find the style in the Styles view at the bottom left), the text in the left pane is displayed and highlighted. This makes it easy to change the settings and see the change immediately.

2. Change the Hex HTML color code for `.nav-h-normal A: hover` from `color: #cc6600`; (orange) to `color: #ff0000`; (red).
3. Customize the footer highlighted link text. Locate the `.nav-f-normal A: hover` style, and change the color from `#ff6600` (orange) to `#ff0000` (red).
4. Save the file.

Now when you hover over the links in the header and footer, the color changes to red, which can be demonstrated on the Preview tab of `index.html`. Obviously, any number of changes can be applied to the style sheets to change the look and feel of the application.

18.5.7 Verifying the site navigation and page templates

At this stage, although the pages have no content, verify that the page templates look as expected and that the navigation links in the header and footer navigation bars work as required:

1. Add text to identify each of the web pages created in the Web Site Navigation:

- a. Open the **welcome.html** page in the Page Designer and go to the **Source** tab.
 - b. Type `Welcome Page` between the tags `<body></body>`. The final code is displayed:

```
<body>Welcome Page</body>
```
 - c. Repeat these steps indicating the names in the body page for `rates.html`, `redbank.html`, and `insurance.html`.
 - d. Start the WebSphere Application Server v8.0 Beta server in the Servers view if it is not running. On the Servers view, from the context menu, select **Start** and wait until the status indicator says **Started**.
2. In the Enterprise Explorer view, right-click **index.html** in **RAD8BankBasicWeb** and select **Run As** → **Run on Server**.
 3. In the Run Server window, select **Choose an existing server** and choose **WebSphere Application Server v8.0 Beta**. Click **Finish**.

After the application is published to the server, the browser pane shows the index page. You can click the tabs for `rates`, `redbank`, and `insurance` to move between these pages (Figure 18-20).

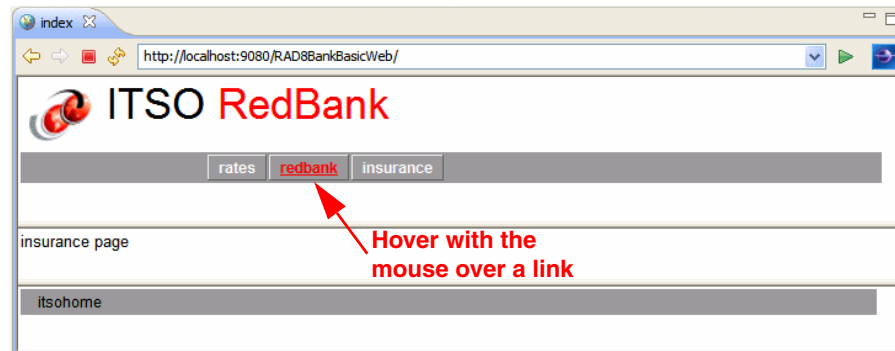


Figure 18-20 ITSO RedBank website

4. To remove the project from the test server, in the Servers view, right-click **WebSphere Application Server v8.0 Beta**, select **Add Remove Projects**, and remove **RAD8BankBasicEAR**.

Alternatively, expand **WebSphere Application Server v8.0 Beta**, right-click the **RAD8BankBasicEAR** project and select **Remove**.

18.5.8 Developing the static web resources

In this section, we create the content for the four static pages of our sample with the objective of highlighting several of the features of the Page Designer. The Page Designer facilitates the building of HTML pages by allowing the user to add HTML elements from the Pallet view using the drag-and-drop method. HTML fragments can also be imported directly into the source tab, as we also demonstrate.

This section covers the following topics:

- ▶ Creating the welcome.html page content (text and links)
- ▶ Creating the rates.html page content (tables)
- ▶ Importing the insurance.html page contents
- ▶ Importing the redbank.html page contents

Creating the welcome.html page content (text and links)

The RedBank home page is `index.html`. The links to the child pages are included as part of the header and footer of our page template. In the following example, we add static text to the page and add a link to the page to the Redbooks website:

1. Open the **welcome.html** file in Page Designer.
2. Select the **Design** tab.
3. Insert the welcome message text:
 - a. Delete the `Welcome Page` text.
 - b. Insert two line breaks:
 - i. In the Context Area, right-click and select **Insert** → **Line Break**. Leave Type as **Normal** (default).
 - ii. Repeat these steps for the second line break.
 - c. In the Context Area, right-click and select **Insert** → **Paragraph** → **Heading 1**.
 - d. Enter the text `Welcome to the ITSO RedBank!` at the current cursor position, which will place the text between the `<h1>` tags.
4. Insert a link to the Redbooks website:
 - a. Add an empty line after the heading.
 - b. From the menu bar, select **Insert** → **Paragraph** → **Normal**.
 - c. In the new area, enter the text `For more information on the ITSO and IBM Redbooks, please visit our Internet site.`
 - d. Highlight the text **Internet site**, right-click, and select **Insert** → **Link**.

- e. In the Insert Link window, select **HTTP**. In the URL field, type `http://www.ibm.com/redbooks` and click **OK**.
5. Customize the text font face, size, and color. In the Properties view, perform these steps:
 - a. Select **Red** in Redbooks from the text created in the previous step (use the keyboard Shift and arrow keys).
 - b. On the Text tab of the Properties view, select the color **Red** to make this partial word stand out. The source changes to `...IBM Redbooks, ...`
6. Save the page.
7. Select the **Preview** tab to view the page (Figure 18-21). Verify that the page looks correct in both Firefox and Internet Explorer.



Figure 18-21 Preview of the `welcome.html` page

Creating the `rates.html` page content (tables)

In this example, you add a static table that contains interest rates by using the Page Designer:

1. Open the **rates.html** file in Page Designer and select the **Design** tab.
2. Delete the **Rates Page** text.
3. In the Palette, expand **HTML Tags**.
4. Select and drag a **Table** from the Palette to the content area.
5. In the Insert Table window, for the Rows, Columns, and Padding inside cells fields, enter 5. Then click **OK**.
6. Resize the table as desired.
7. Enter the descriptions and rates (as seen in Figure 18-22 on page 1024) in the table.

8. Select each heading text, and in the Properties view, click the **Bold** icon ().

Table option: You can add additional table rows and columns or delete rows and columns by using the Table menu option.

9. Save the page.
10. Select the **Preview** tab to view the page (Figure 18-22).

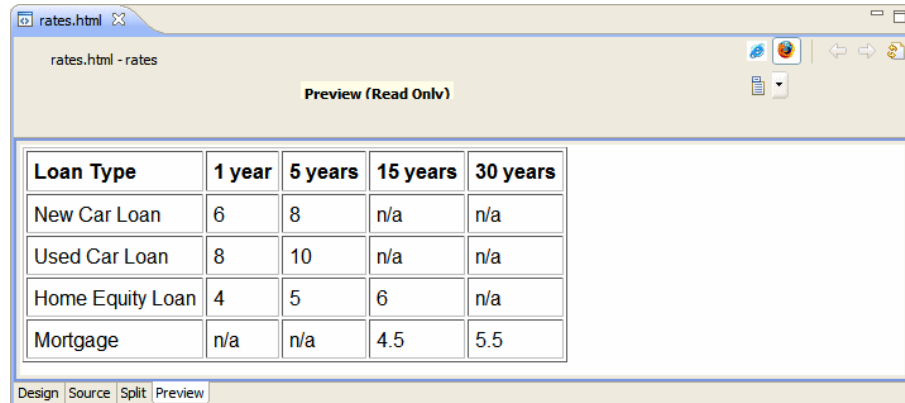


Figure 18-22 Preview of the rates.html page

Importing the insurance.html page contents

Next import the body of the `insurance.html` file:

1. Locate the `c:\7835code\webapp\html\SnippetForInsuranceHTML.txt` file and open it in a simple text editor (for example, Notepad).
2. Open the **insurance.html** file in Page Designer and select the **Source** tab.
3. Select the text between the tags: `<body></body>`.
4. Insert the text from the `SnippetForInsuranceHTML.txt` file.
5. Save the file and switch to the **Preview** tab (Figure 18-23 on page 1025).

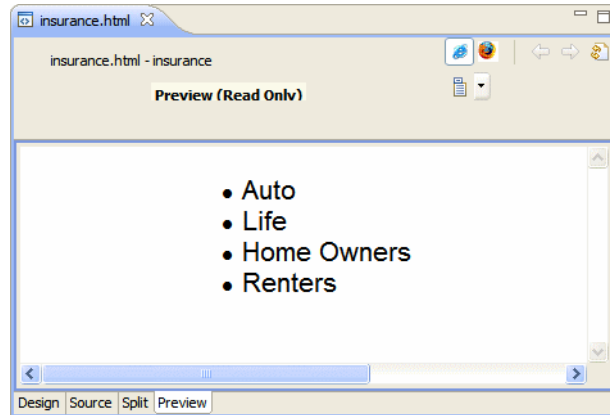


Figure 18-23 Preview of the `insurance.html` page

Importing the `redbank.html` page contents

Repeat the import of the body of the `redbank.html` file:

1. Locate the `c:\7835code\webapp\html\SnippetForBankHTML.txt` file.
2. Replace the existing content area text with the text from the snippet.
3. Switch to the **Preview** tab (Figure 18-24).

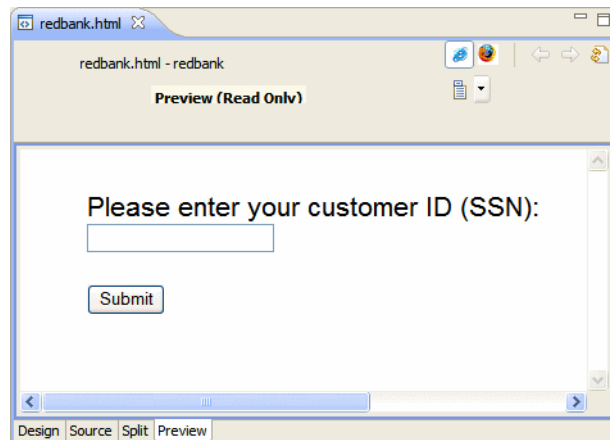


Figure 18-24 Preview of the `redbank.html` page

The static HTML pages for the RedBank application are now complete. You can navigate the site by using the header action bar and the footer `itsohome` link.

18.5.9 Developing the dynamic web resources

In addition to the tools created for building HTML content and designing the flow and navigation in a web application, Rational Application Developer provides several wizards to help you quickly build JavaServer Pages (JSP) and Java servlets. You can use the products of these wizards as is or modify them to fit specific needs.

The wizards support the creation of servlets and JSP. They also compile the Java code and store the class files in the correct folders for publishing to your application servers. As the wizards generate project resources, the appropriate annotations are added to the generated classes (Servlet 3.0 and up) or the deployment descriptor file (`web.xml`) is updated automatically with the appropriate configuration information for the servlets that are created (versions earlier than Servlet 3.0).

In the previous section, we explained how to create each of the static web pages. In this section, we demonstrate the process of creating and working with servlets. The example servlets are first built using the wizards. Then the code contents are imported from the sample solution. In 18.5.10, “Working with JSP” on page 1036, the JSP pages, which invoke the logic in these servlets, are created.

Working with servlets

As described in 18.1, “Introduction to Java EE web applications” on page 983, servlets are flexible and scalable server-side Java components based on the Java Servlet API, as defined in the JEE Servlet Specification. Servlets generate dynamic content by responding to web client requests. When an HTTP request is received by the application server, the web server determines, based on the request URI, which servlet is responsible for answering that request and forwards the request to that servlet. The servlet then performs its logic and builds the response HTML that is returned back to the web client, or forwards the control to a JSP page.

Rational Application Developer supports Version 3.0 of the JEE servlet specification which means that servlets are defined by the `WebServlet` annotation in the servlet class rather than by an entry in the `web.xml` file.

Rational Application Developer provides the features to make servlets easy to develop and integrate into your web application. From the workbench, you can develop, debug, and deploy servlets. You can also set breakpoints within servlets and step through the code in a debugger. Any changes made are dynamically folded into the running web application, without restarting the server each time.

In the sections that follow, we implement the `ListAccounts`, `UpdateCustomer`, `AccountDetails`, and `Logout` servlets. Then the command or action pattern is applied in Java to implement the `PerformTransaction` servlet.

Adding the RAD80Java JAR to the web project

Before the implementation of the servlet classes can proceed, we must add a reference from the `RAD8BankBasicWeb` project to the `RAD80Java` project, because the servlets call the methods from classes in this project. This is achieved in Rational Application Developer using a feature called *Deployment Assembly*, which is available in the Properties window for each web project.

Creating the reference: In previous versions of Rational Application Developer, this reference was created in the J2EE Module Dependencies dialog window. The new Deployment Assembly dialog window can be used to achieve the same result.

To add `RAD80Java` to the Deployment Assembly for `RAD8BankBasicWeb`, complete the following steps.

To add the JAR file to the class path of the `RAD8BankBasicWeb` project, follow these steps:

1. Highlight the **RAD8BankBasicWeb** project, right-click and select **Properties**.
2. On the Properties for `RAD8BankBasicWeb` dialog window, select the **Deployment Assembly** link and click **Add**.
3. From the New Assembly Directive: Select Directive Type window, select **Project** and click **Next**.
4. From the New Assembly Directive: Projects window, highlight the **RAD80Java** project and click **Finish**.

`RAD80Java.jar` now appears in the project's Java Build path under Web App Libraries (see Figure 18-25 on page 1028), which indicates that the Java classes are available at compile time. `RAD80Java.jar` also appears in the Web Deployment Assembly list (see Figure 18-26 on page 1028), which indicates that the JAR file will be added to a WAR file generated from `RAD8BankBasicWeb`. Click **OK** to close the Properties dialog window.

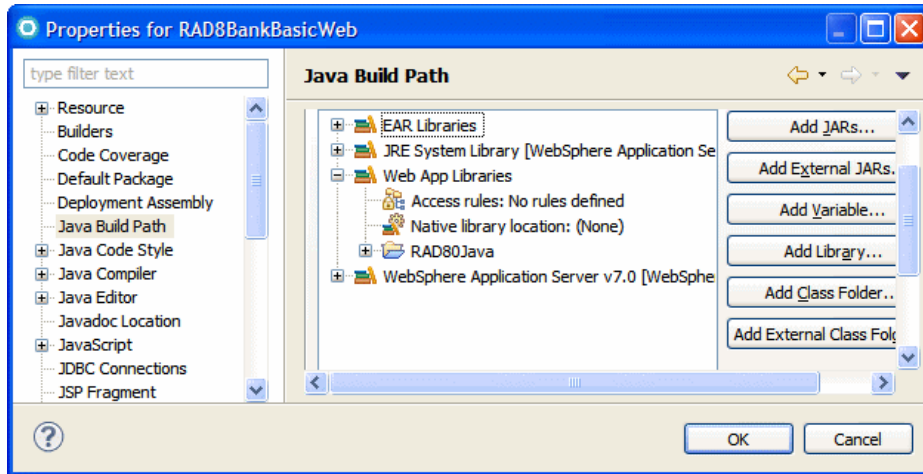


Figure 18-25 Updated Java Build Path for RAD8BankBasicWeb

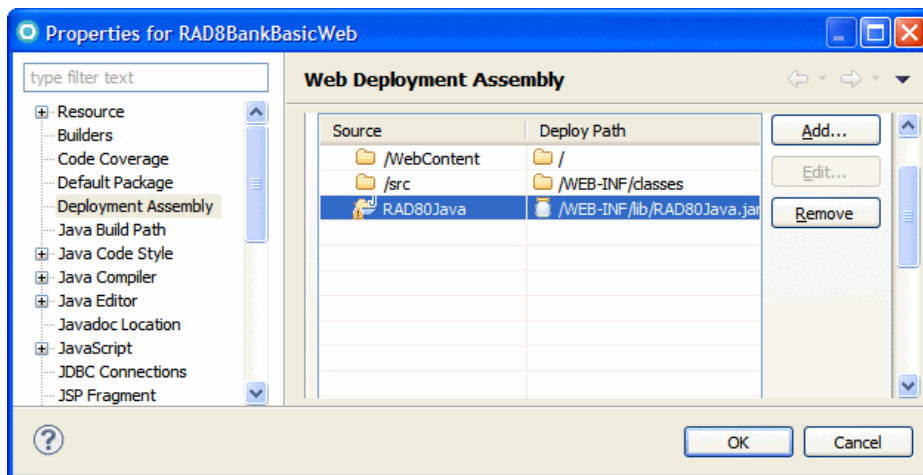


Figure 18-26 Updated Web Deployment Assembly for RAD8BankBasicWeb

Adding the ListAccounts servlet to the web project

Rational Application Developer provides a servlet wizard to assist you in adding servlets to your web application:

1. Select **File** → **New** → **Other** and then select **Web** → **Servlet**. Click **Next**.

Tip: You can also access the Create Servlet wizard by right-clicking the project and selecting **New** → **Servlet**.

2. In the first window of the Create Servlet wizard (Figure 18-27), for the Java package, type `itso.rad8.webapps.servlet`. For the Class name, type `ListAccounts`. Click **Next**.

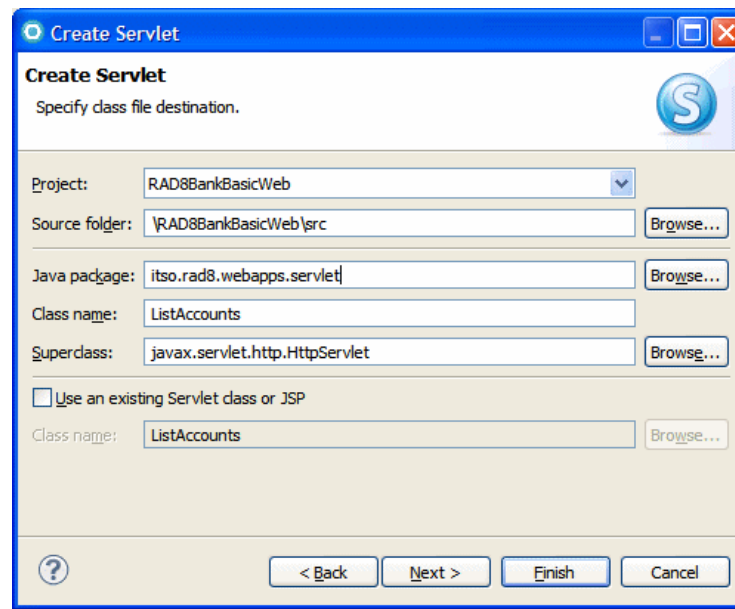


Figure 18-27 New Servlet wizard: Create Servlet window (part 1 of 3)

The second window (Figure 18-28 on page 1030) provides space for the name and description of the new servlet.

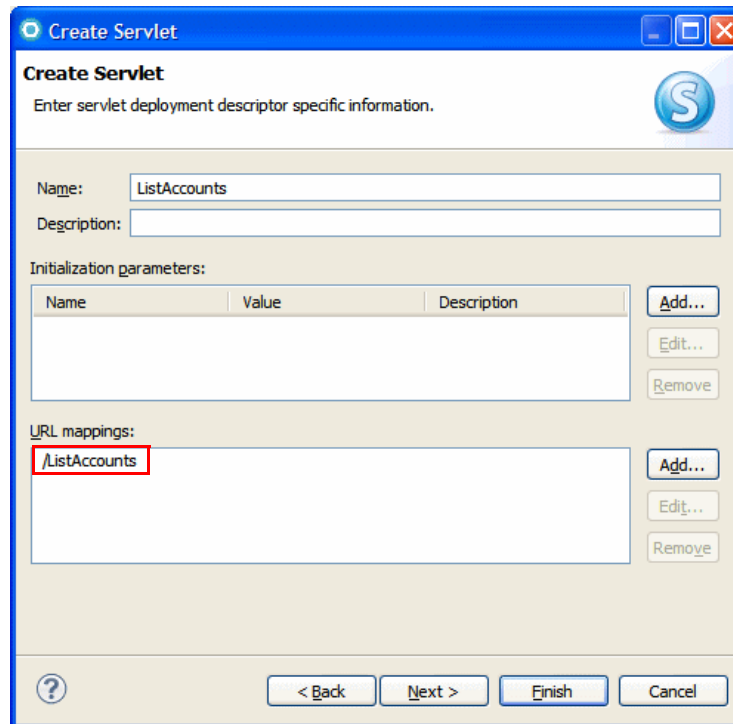


Figure 18-28 New Servlet wizard: Create Servlet window (part 2 of 3)

On this window, you can also add servlet initialization parameters, which are used to parameterize a servlet. You can change servlet initialization parameters at run time from within the WebSphere Application Server administrative console.

The wizard automatically generates the URL mapping `/ListAccounts` for the new servlet. If other, or additional, URL mappings are required, you can add them here. In our sample, we do not require additional URL mappings or initialization parameters. Click **Next**.

3. In the third window (Figure 18-29 on page 1031), click **Finish**. In this window, you can have the wizard create method stubs for methods that are available from the `HttpServlet` interface. The `init` method is called at start-up, and `destroy` is called at shutdown.

The `doPost`, `doGet`, `doPut`, and `doDelete` methods are called when an HTTP request is received for this servlet. All of the `do` methods have two parameters: an `HttpServletRequest` and an `HttpServletResponse`. These methods are responsible for extracting the pertinent details from the request and for populating the response object.

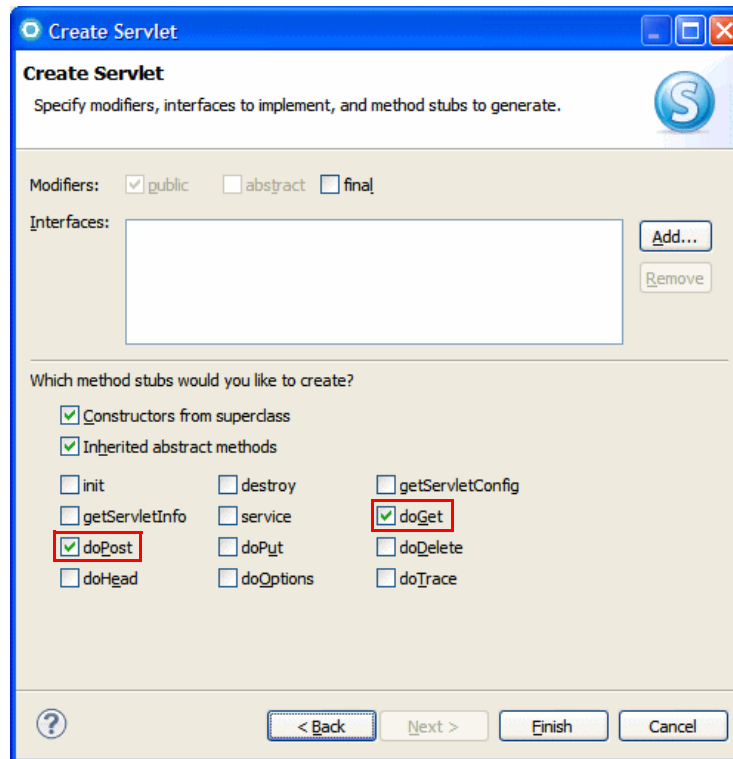


Figure 18-29 New Create Servlet wizard page (part 3 of 3)

For the `ListAccounts` servlet, ensure that only `doGet` and `doPost` are selected. Usually, HTTP gets are used with direct links, when no information has to be sent to the server. HTTP posts are typically used when information in a form has to be sent to the server.

No initialization is required for the new servlet. Therefore, the `init` method is not selected.

The servlet is generated and added to the project. You can find the source code in the Java Resources folder of the project.

4. Expand the deployment descriptor list for the **RAD8BankBasicWeb** (immediately under the project in the Enterprise Explorer), and you see that the `ListAccounts` servlet is shown.

Implementing the `ListAccounts` servlet

A skeleton servlet now exists, but it does not perform any actions when it is invoked. Now add code to the servlet to implement the required behavior. The `ListAccounts.java` code of the servlet is already opened.

Follow these steps:

1. Locate the `c:\7835code\webapp\servlet>ListAccounts.java` file.
2. Replace the contents of `ListAccounts.java` with the sample file. The file compiles successfully with no errors.
3. Examine the source code for the `ListAccounts.java` servlet. Note the annotation before the class definition:

```
@WebServlet("/ListAccounts")
```

This annotation declares the class as a servlet for the web project and describes the URL mapping.

This class implements the `doPost` and `doGet` methods, both of which call the `performTask` method.

The `performTask` method does the following tasks:

- a. The method deals with the HTTP request parameters supplied in the request. This servlet expects to either receive a parameter called `customerNumber` or none at all. If the parameter is passed, we store it in the HTTP session for future use. If it is not passed, we look for it in the HTTP session, because it might have been stored there earlier.
- b. The method implements the control logic. Access to the Bank facade is obtained through the `ITS0Bank.getBank` method, and it is used to get the customer object and the array of accounts for that customer.
- c. The third section adds the customer and account variables to the `HttpRequest` object so that the presentation renderer (`listAccounts.jsp`) gets the parameters that it requires to perform its job. The control of processing the request is then passed through to `listAccounts.jsp` using the `RequestDispatcher.forward` method, which builds the response to be shown on the browser.
- d. The final part of the method is the error handler. If an exception is thrown in the previous code, the catch block ensures that control is passed to the `showException.jsp` page.

Figure 18-9 on page 1003 shows a sequence diagram of the design of this class.

The `ListAccounts` servlet is now complete.

4. Save the changes and close the source editor.

Implementing the UpdateCustomer servlet

The `UpdateCustomer` servlet is used for updating the customer information and is invoked from the `ListAccounts` JSP through a push button.

The servlet requires that the Social Security number (SSN) of the customer that is to be updated is already placed on the session (as must be done in the `ListAccounts` servlet). It extracts the `title`, `firstName`, and `lastName` parameters from the `HttpServletRequest` object, calls the `bank.getCustomer(String customerNumber)` method, and uses the simple setters on the `Customer` class to update the details.

Follow the procedures in “Adding the `ListAccounts` servlet to the web project” on page 1028 and “Implementing the `ListAccounts` servlet” on page 1031 to build the servlet, including the `doGet` and `doPost` methods. The code to use for this class is in the `c:\7835code\webapp\servlet\UpdateCustomer.java` file.

Implementing the `AccountDetails` servlet

The `AccountDetails` servlet retrieves the account details and forwards control to the `accountDetails.jsp` page to show these details. The servlet expects the parameter `accountId`, which specifies the account for which data must be shown, in the request. The servlet calls the `bank.getAccount(...)` method, which returns an `Account` object and adds it as a variable to the request. It then uses the `RequestDispatcher` to forward the request onto the `accountDetails.jsp`.

Security and authorization: An actual client implementation performs security and authorization where the current user has the required access rights to the requested account. You can implement security and authorization by using the Security Editor tool, as described in 18.2.5, “Security Editor” on page 996.

Follow the procedures in “Adding the `ListAccounts` servlet to the web project” on page 1028 and “Implementing the `ListAccounts` servlet” on page 1031 to build the servlet. The code to use for this class is in the `c:\7835code\webapp\servlet\AccountDetails.java` file.

Implementing the `Logout` servlet

The `Logout` servlet is used for logging the customer off from the `RedBank` application. The servlet requires no parameters. The only logic performed in the servlet is to remove the SSN from the session, simulating a logoff action. You remove the SSN by calling the `session.removeAttribute` and `session.invalidate` methods. Finally, the servlet uses the `RequestDispatcher` class to forward the browser to the `index.html` page.

Follow the procedures in “Adding the `ListAccounts` servlet to the web project” on page 1028 and “Implementing the `ListAccounts` servlet” on page 1031 to build the servlet. The code to use for this class is in the `c:\7835code\webapp\servlet\Logout.java` file.

Implementing the PerformTransaction command classes

In the `PerformTransaction` servlet, a `Command` design pattern is used to implement it as a front controller class that forwards control to one of the four command objects: `Deposit`, `Withdraw`, `Transfer`, and `ListTransactions`.

You can find this design in 18.4.3, “Controller layer” on page 1002. This implementation is based on the sequence diagram (Figure 18-10 on page 1004).

Next we import the code for the commands package. The source is in the `C:\7835code\webapp\command` folder. Follow these steps:

1. Create the `itso.rad8.webapps.command` package. In the Enterprise Explorer, right-click the **Java Resources: src** folder and select **New** → **Package**.
2. For the package name, type `itso.rad8.webapps.command` and click **Finish**.
3. From the context menu of the new package, click **Import** and select **General** → **File system**. Click **Next**.
4. Click **Browse** and navigate to the `C:\7835code\webapp\command` folder. Click **OK**.
5. Select the following Java files and click **Finish**:
 - `Command.java`
 - `DepositCommand.java`
 - `ListTransactionsCommand.java`
 - `TransferCommand.java`
 - `WithdrawCommand.java`

The command classes perform the operations on the `RedBank` model classes (from the `RAD80Java` project) through the `Bank` facade. They also return the file name for the next page to be displayed after the command has executed.

Implementing the PerformTransaction servlet

Now that all the commands for the `PerformTransaction` framework have been realized, you can create the `PerformTransaction` servlet. The servlet uses the value of the transaction request parameter to determine which command to execute.

You can use the `Create Servlet` wizard to create a servlet named `PerformTransaction`. The servlet class must be placed in the `itso.rad8.webapps.servlet` package.

Follow the procedures in “Implementing the `ListAccounts` servlet” on page 1031 to prepare the servlet, including the `doGet` and `doPost` methods. The code to use for this class is in the `c:\7835code\webapp\servlet\PerformTransaction.java` file.

PerformTransaction stores a HashMap of the action strings (deposit, withdraw, transfer, and list) to instances of Command classes. Both the doGet and doPost methods call performTask. In the performTask method, the execute method is called on the appropriate Command class that performs the transaction on the Bank classes. After the execute method completes, the getForwardView method is called on the Command class, which returns the next page to display, and PerformTransaction uses the RequestDispatcher to forward the request to the next page.

After this step, all the servlets required for the sample application have been built. The Enterprise Explorer view shows a list of all the servlets in the web project and the URL mappings to these servlets (Figure 18-30 on page 1036).

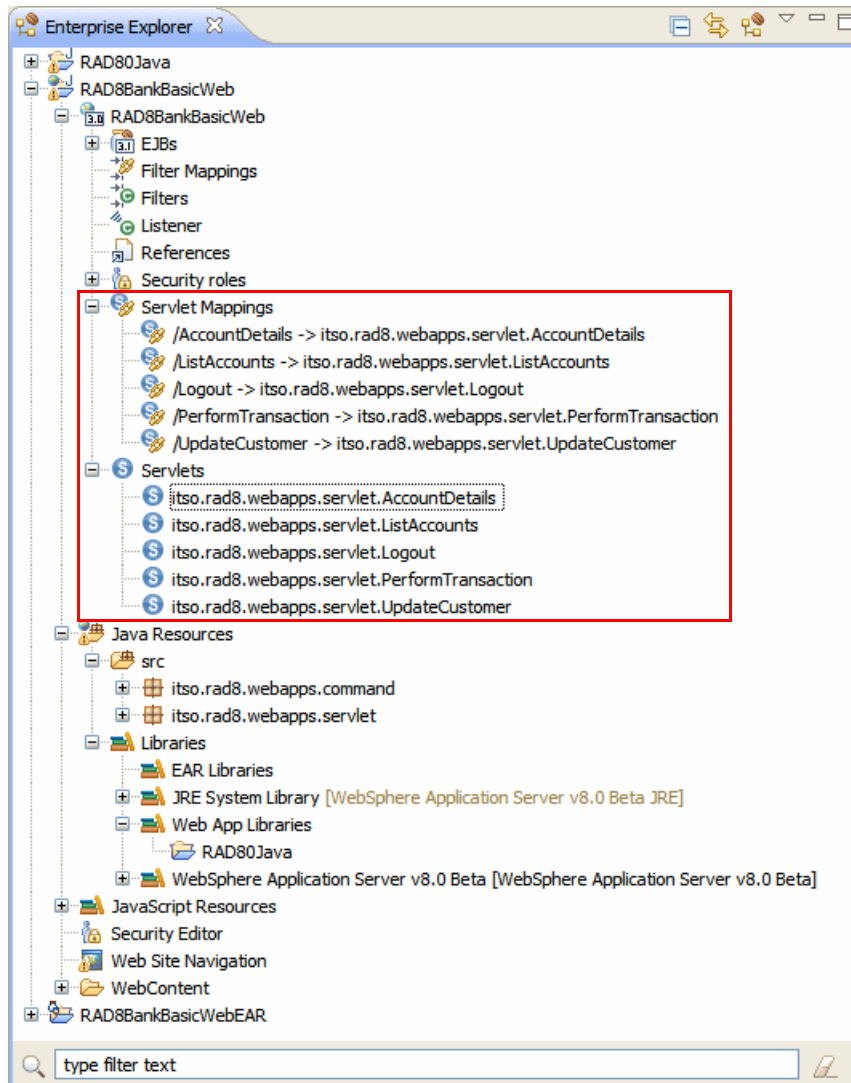


Figure 18-30 Servlets and Servlet Mappings in Enterprise Explorer

18.5.10 Working with JSP

JSP files are edited in the Page Designer, which is the same editor that is used to edit the HTML page. When working with a JSP page in the Page Designer, the Palette view has a separate drawer for JSP Tags, which includes elements, such as JavaBean references, JavaServer Pages Standard Tag Library (JSTL) tags, and scriptlets containing Java code.

In this section, we describe the implementation of `listAccounts.jsp` in detail, and the other JSP (`accountDetails.jsp`, `listTransactions.jsp`, and `showException.jsp`) are imported from the solution.

Implementing the List Accounts JSP

Customizing a JSP file by adding static content is done in the Page Designer tool in the same way that an HTML file can be edited. You can also add the standard JSP declarations, scriptlets, expressions, tags, or any other custom tag that is developed or retrieved from the Internet.

In this example, the `listAccounts.jsp` file is built by using page data variables for *customer* and *accounts* (and an array of `Account` classes for that customer). These variables are added to the page by the `ListAccounts` servlet and are accessible to the Java code and tags used in the JSP.

To complete the body of the `listAccounts.jsp` file, perform the following steps:

1. Open the **listAccounts.jsp** in Page Designer and select the **Design** tab.
2. Add the *customer* and *accounts* variables to the page data meta information in the Page Data view (by default, one of the views in the upper-left corner). These variables are added to the request object in the `ListAccounts` servlet, as discussed in “Implementing the ListAccounts servlet” on page 1031. Page Designer must be aware of these variables. Complete these steps:
 - a. In the Page Data view, expand **Scripting Variables**, right-click **requestScope**, and select **New** → **Request Scope Variable**.
 - b. In the Add Request Scope Variable window, complete the following tasks:
 - i. For the Variable name, select **customer**.
 - ii. Type `itso.rad80.bank.model.Customer`.
 - iii. Click **OK**.

Tip: You can use the browse button (marked with an ellipsis (...)) to find the class that is using the class browser.

- c. Repeat this procedure to add the following request scope variable:
 - i. For the Variable name, select **accounts**.
 - ii. Type `itso.rad80.bank.model.Account []`.

Important: The square brackets indicate that the variable `accounts` is an array of accounts.

3. In the Palette view, select **Form Tags** → **Form** and click anywhere on the JSP page in the content table. A dashed box appears on the JSP page, representing the new form.
4. In the Properties view for the new Form element, enter the following items:
 - a. For the Action, type UpdateCustomer.
 - b. For the Method, select **Post**.

Tip: You can use the Outline view to navigate to the form tag quickly.

5. Add a table with the customer information:
 - a. In the Page Data view, expand and select **Scripting Variables** → **requestScope** → **customer** (`itso.rad80.java.model.Customer`).
 - b. Select and drag the customer object to the form that was previously created.
 - c. In the Insert JavaBean window (Figure 18-31 on page 1039), follow these steps:
 - i. Select **Displaying data (read-only)**.
 - ii. Use the arrow up and down buttons to arrange the fields in the order shown and overwrite the labels.
 - iii. Clear the **accounts** field (we do not display the accounts).

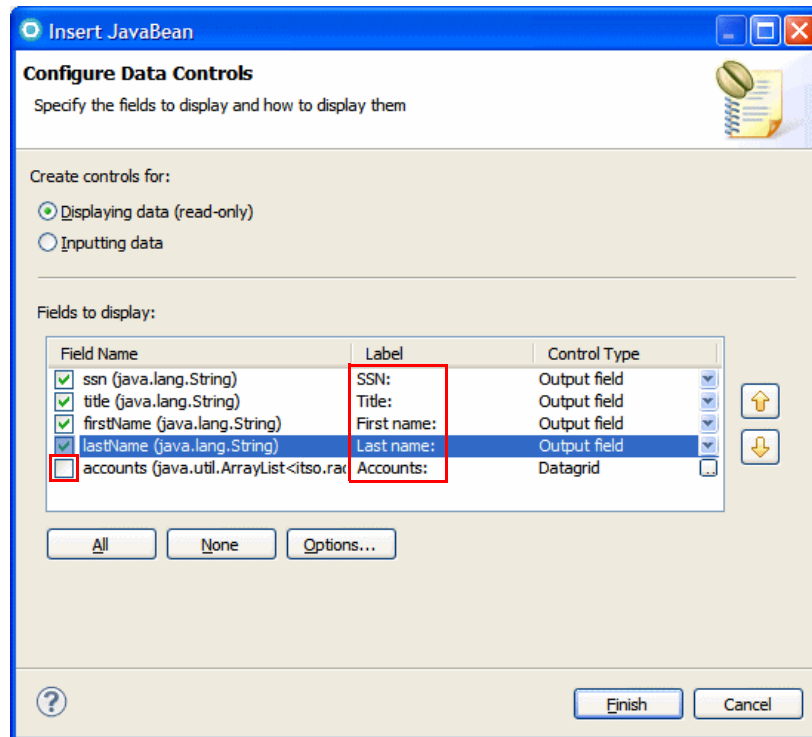


Figure 18-31 Inserting the customer JavaBean

- iv. Click **Finish** to add the Data control for the Customer.

Customer data in table: The newly created table with customer data is changed in a later stage to use input fields for the title, first name, and last name fields. Creating an editable version of this information is not possible with the available wizards.

- d. Right-click the last row of the newly created table (select the **LastName** cell) and select **Table** → **Add Row Below**.
6. In the Palette view, select **Form Tags** → **Submit Button** and click in the right cell of the new row. In the Label field, enter Update and click **OK**. You can leave the Name field empty.
7. In the Palette view, select **HTML Tags** → **Horizontal Rule** and click in the area immediately beneath the form.
8. In the Page Data view, expand **Scripting Variables** → **requestScope** → **accounts** (itso.rad8.java.model.Account[]).
9. Drag the accounts object beneath the Horizontal Rule that was created.

10. In the Insert JavaBean: Configure Data Controls wizard (Figure 18-32), follow these steps:
 - a. Clear **transactions**. We do not display the transactions.
 - b. For both fields, in the Control Type column, select **Output link**.
 - c. In the Label field for the accountNumber field, enter AccountNumber.
 - d. Ensure that the order of the fields is accountNumber and balance.
 - e. Click **Finish**. The accounts bean is added to the page and is displayed as a list.

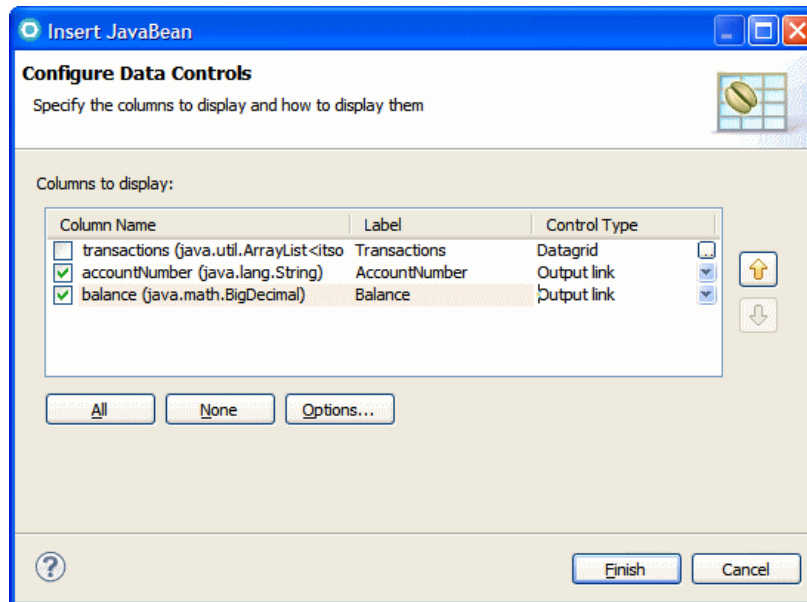


Figure 18-32 Inserting the accounts JavaBean

11. The wizard inserts a JSTL `c:forEach` tag and an HTML table with headings, as entered in the Insert JavaBean window. Because we selected **Output link** as the Control Type for each column, corresponding `c:url` tags have been inserted. We now have to edit the URL for these links to make sure that they are identical and to pass the `accountId` variable as a URI parameter:
 - a. From the Design view, select the first `<c:url>` tag under the heading AccountNumber, which has the text `${varAccounts.accountNumber}`. In the Properties view (Figure 18-32), in the Value field, enter AccountDetails.
The tag changes to `AccountDetails`.

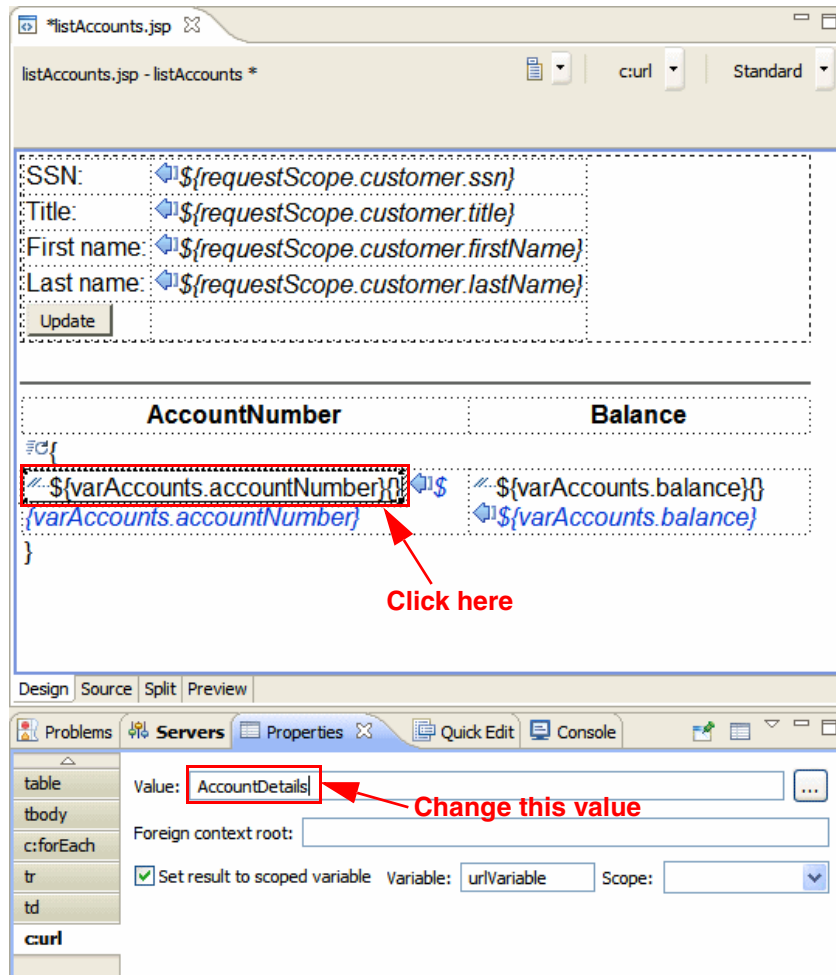



Figure 18-33 Configuring the AccountDetails URL

- b. Under the heading Balance, which has the text `${varAccounts.balance}`, select the second `<c:url>` tag. In the Properties view, in the Value field, enter `AccountDetails`. This value specifies the target URL for the link, which, in this case, maps to the `AccountDetails` servlet.
- c. Add a parameter to this URL to ensure that the link goes to the correct account. In the Palette view, select **JSP Tags** → **Parameter** () and click the first `<c:url>` in the AccountNumber column. This has the text `AccountDetails{}` (Figure 18-34 on page 1042).

- d. In the Properties view, on the **c:param** tab, in the Name field, enter `accountId`, and in the Value field, enter `${varAccounts.accountNumber}`. This action adds a parameter to the account URL with a name of `accountId` and the value of the `accountNumber` request variable.

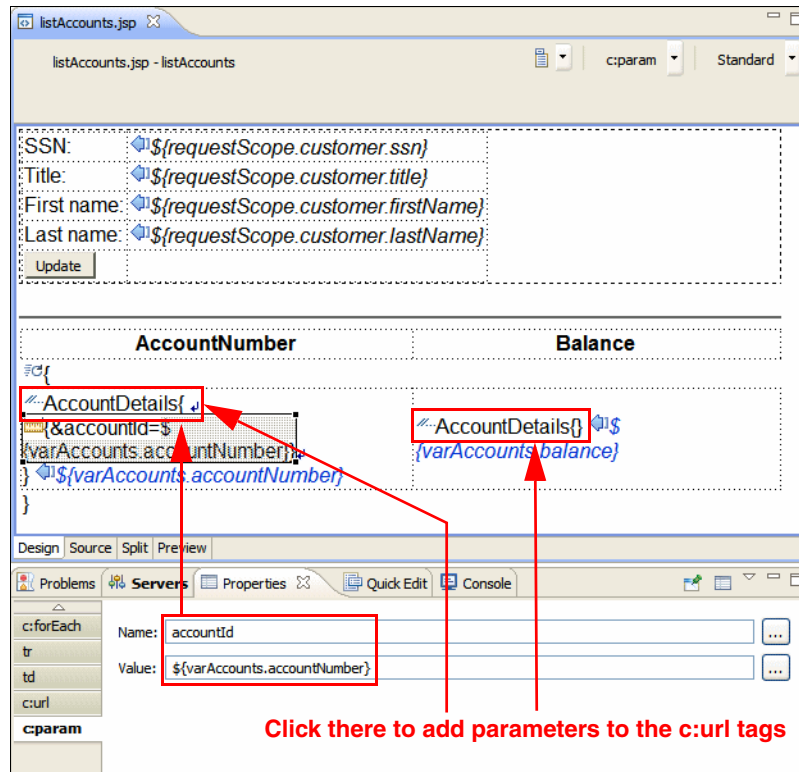


Figure 18-34 Adding parameters to `c:url` tags

- e. Repeat the two previous steps to add a parameter to the second `<c:url>` tag in the Balance column, showing the text `Account Number{}</code>. In the Name field, type accountId, and in the Value field, type ${varAccounts.accountNumber}.`
12. Click anywhere in the Balance column and select the **td** tag in the Properties view. In the Horizontal alignment list box, select **Right** to right-align the contents of the Balance cells.
 13. Select the **Source** tab and compare the code to display the accounts to the code that is shown in Example 18-1 on page 1043. This JSP code shows the accounts as a list, using the `<c:forEach>` tag to loop through each account, and the `<c:url>` tag

builds a URL to AccountDetails and the <c:param> tag adds the accountId parameter (with the account number value) to that URL.

Example 18-1 JSP code with JSTL tags to display accounts (formatted)

```
<c:forEach var="varAccounts" items="{requestScope.accounts}">
  <tr>
    <td>
      <c:url value="AccountDetails" var="urlVariable">
        <c:param name="accountId"
          value="{varAccounts.accountNumber}"></c:param>
      </c:url>
      <a href="{c:out value='{urlVariable}' /}">
        <c:out value="{varAccounts.accountNumber}"></c:out>
      </a>
    </td>
    <td align="right">
      <c:url value="AccountDetails" var="urlVariable">
        <c:param name="accountId"
          value="{varAccounts.accountNumber}"></c:param>
      </c:url>
      <a href="{c:out value='{urlVariable}' /}">
        <c:out value="{varAccounts.balance}" />
      </a>
    </td>
  </tr>
</c:forEach>
```

14. Select the **Split** tab. From the Palette view, select **HTMLTags** → **Horizontal Rule** and click in the area beneath the account details table.
15. Add a logout form:
 - a. In the Palette view, select **Form Tags** → **Form** and click beneath the new horizontal rule. A dashed box is displayed on the JSP page, representing the new form.
 - b. In the Properties view for the new form tag, enter the following items:
 - i. For the Action, type Logout.
 - ii. For the Method, select **Post**.
 - c. In the Palette view, select **Form Tags** → **Submit Button** and click in the new form. When you click the Logout button, the doPost method is called on the Logout servlet.
 - d. In the Insert Submit Button window, in the Label field, enter Logout and click **OK**.

16. Change the title, first name, and last name to entry fields, so that the user can update the customer's details. To convert the Title, First name, and Last name text fields to allow text entry, follow these steps:

a. Select the `${requestScope.customer.title}` field.

b. Select the **Source** tab and you can see the following code:

```
<td><c:out value="${requestScope.customer.title}" /></td>
```

c. Change the code to this code:

```
<td><input type="text" name="title"
        value="<c:out value='${requestScope.customer.title}' />"
/></td>
```

d. Repeat these steps for the first name and last name fields:

```
<td><input type="text" name="firstName"
        value="<c:out value='${requestScope.customer.firstName}' />"
/></td>
```

.....

```
<td><input type="text" name="lastName"
        value="<c:out value='${requestScope.customer.lastName}' />"
/></td>
```

The customer fields change from display-only fields to editable fields, so that the details can be changed.

You can change the length of the three input fields in the Properties view. For example, you can specify 6 columns and 3 as the maximum length for the title, and 32 columns for the names.

You can also change the width of the content areas in the source code.

17. Format the account balance, which is a `BigDecimal`. Otherwise, it is displayed with many digits. Complete these steps:

a. In the Balance column, click the balance field `${varAccounts.balance}`.

b. From the context menu, select **JSP** → **Insert Custom**.

c. In the Insert Custom Tag window (Figure 18-35 on page 1045), click **Add** to add another tag library.

d. Locate and select the `http://java.sun.com/jsp/jstl/fmt` URI and click **OK**.

e. Select the new tag library and select `formatNumber` as the custom tag. Click **Insert** and click **Close**.

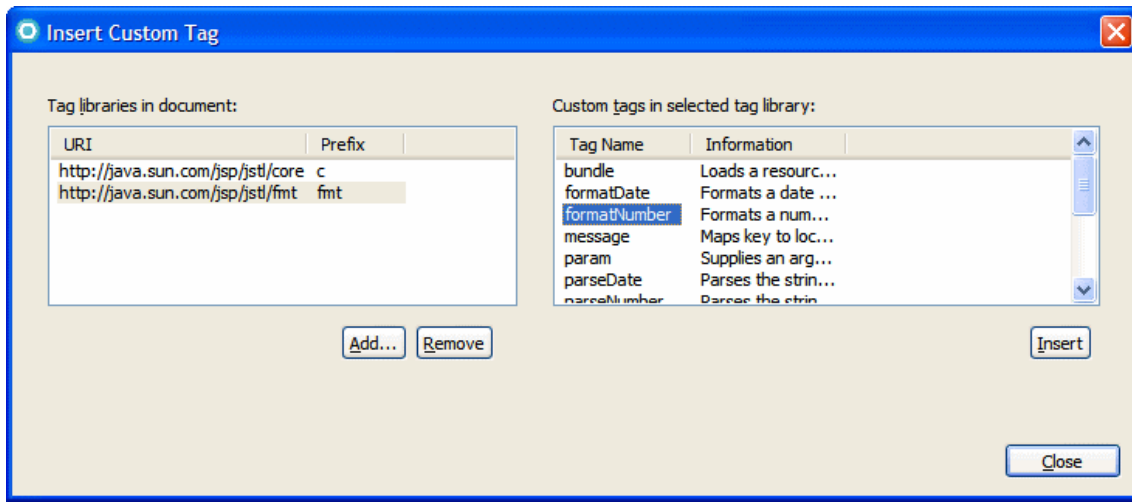


Figure 18-35 Inserting a custom tag

- f. On the Source tab, select the **<fmt:formatNumber>** tag. In the Properties view, set `maxFractionDigits` and `minFractionDigits` to **2**. For the value, type `${varAccounts.balance}`.

- g. Remove the `<c:cout value=... >` and `</c:cout>` tags:

```

<a href="<c:cout value='${urlVariable}' />">
  <del>c:cout value="${varAccounts.balance}" />
  <fmt:formatNumber maxFractionDigits="2" minFractionDigits="2"
    value="${varAccounts.balance}"></fmt:formatNumber>
</del>c:cout>
</a>

```

18. Select any field in the accounts table. In the Properties view, select the **table** tab. Set the width to **100** and select **%**.

19. Select the **Account Number** heading. In the Properties view, set the horizontal alignment to **Left**. Select the **Balance** heading and set the horizontal alignment to **Right**.

20. Save the file.

Figure 18-36 on page 1046 shows the JSP in the **Design** tab.

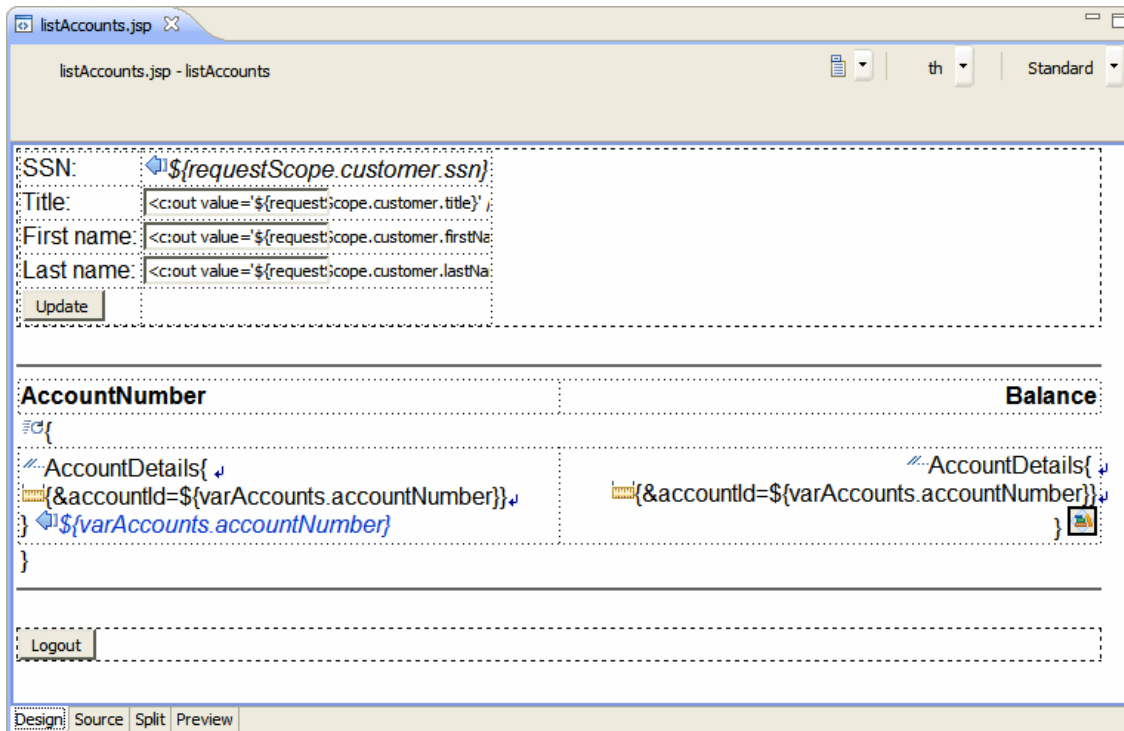


Figure 18-36 List Accounts JSP finished

JSP source code for the listAccounts.jsp file: The JSP source code for the listAccounts.jsp is in the c:\7835code\webapp\jsp\ directory. You can import the code into the WebContent folder, or copy and paste it directly into Rational Application Developer.

Implementing the other JSP

The other JPS are already created as part of the model solution. We built them by using a similar process of adding request beans to the JSP and building HTML and JSP elements around them.

To import the other JSP files and to view them in Page Designer, perform the following steps:

1. Select the **WebContent** folder, click **Import**, and select **General** → **File System**. Click **Next**.
2. Click **Browse** and navigate to the **c:\7835code\webapps\jsp** file.

3. Select all the JSP files, except `listAccounts.jsp` (which has already been completed). Click **Finish**.
4. When prompted whether to override the existing files, click **Yes to All**.

Associated request beans of the imported pages: The imported pages do not have the associated request beans showing in the Page Data view, because they are maintained in the `.jspPersistence` file immediately under the web project directory. You have to specifically add them to the Page Data view.

Add the required variables to the appropriate Page Data view by following the next steps for each JSP file in Table 18-2:

1. Open each JSP file in the Page Designer by double-clicking the file from the Project Explorer view.
2. In the Page Data view, select **Scripting Variables** → **New** → **Request Scope Variable** and add the variables for each JSP, as specified in Table 18-2.

Table 18-2 Request scope variables for each JSP

JSP file	Variable	Type
<code>accountDetails.jsp</code>	<code>account</code>	<code>itso.rad80.bank.model.Account</code>
<code>listTransactions.jsp</code>	<code>account</code>	<code>itso.rad80.bank.model.Account</code>
<code>listTransactions.jsp</code>	<code>transactions</code>	<code>itso.rad80.bank.model.Transaction[]</code>
<code>showException.jsp</code>	<code>message</code>	<code>java.lang.String</code>
<code>showException.jsp</code>	<code>forward</code>	<code>java.lang.String</code>

The following sections describe briefly the logic that is contained within each JSP.

Account Details JSP

The `accountDetails.jsp` page shows the details for a particular customer account and gives options to execute a transaction:

- ▶ The JSP uses a single request variable called `account` to populate the top portion of the body of the page, which shows the account number and balance.
- ▶ The middle section is a simple static form, which provides fields for the details of a transaction (transaction type, amount, and destination account) and posts the request to the `PerformTransaction` servlet for processing.
- ▶ The Customer Details button navigates the user to the `listAccounts.jsp` page.

Figure 18-37 shows the Page Designer view of this page.

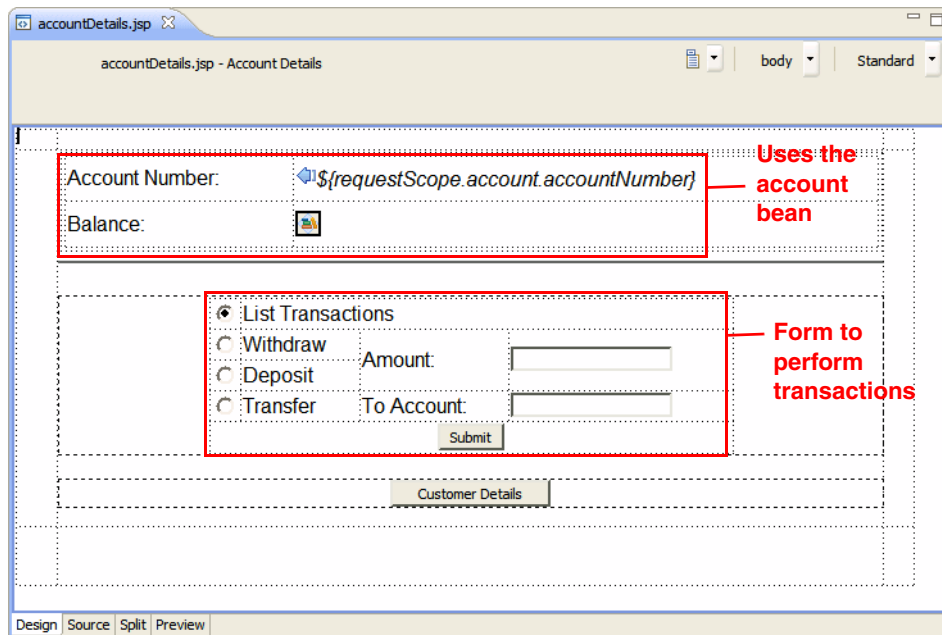


Figure 18-37 Completed `accountDetails.jsp` in a preview

List Transactions JSP

The `listTransactions.jsp` page shows a read-only view of the account, including the account number and balance, plus a list of all transactions:

- ▶ The JSP uses two request variables called `account` and `transactions`. The first section of the page uses the `account` request bean to populate a table showing the account number and balance.
- ▶ The middle section uses the `transactions[]` request bean to show a list of transactions. This transaction array bean uses the JSTL tag library to iterate through the transaction list and build up an HTML representation of the transaction history.
- ▶ The Account Details button returns the browser to the `accountDetails.jsp` page.

Figure 18-38 on page 1049 shows the Page Designer view of this page.

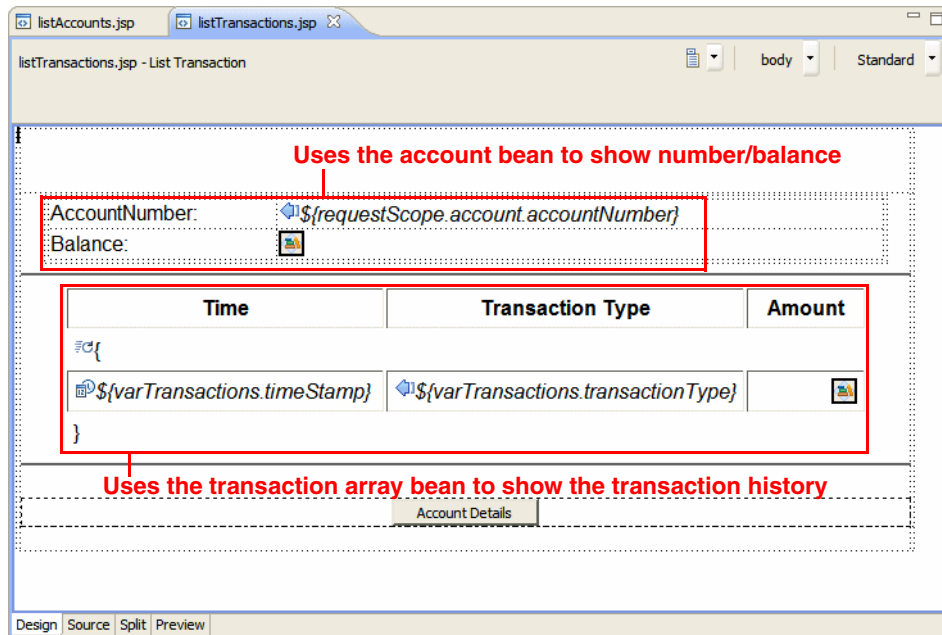


Figure 18-38 Completed `listTransactions.jsp` in a preview

Show Exception JSP

The `showException.jsp` page is displayed when an exception occurs in the processing of a request:

- ▶ The JSP shows a simple error message and gives a link to another page within the RedBank application to allow the user to continue.
- ▶ Two request beans are used on this page. The message bean stores the text to display to the user, and the forward bean stores a URL for the next page to continue. The URL is hidden behind the text `Click here to continue`.

Figure 18-39 shows this page in the Design view of the Page Designer.

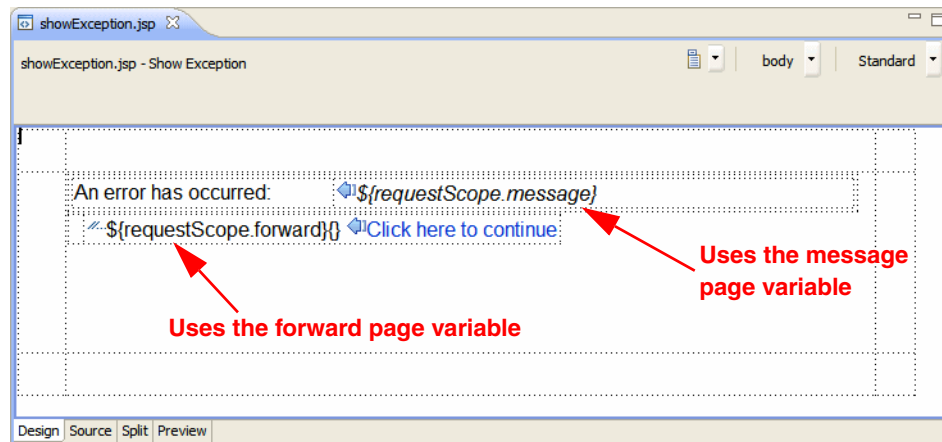


Figure 18-39 Completed `showException.jsp` in a preview

The RedBank application is finished and ready to be tested.

18.6 Web application testing

In this section, we demonstrate how to run the sample RedBank application that we built in the previous sections.

18.6.1 Prerequisites to run the sample web application

To run the RedBank application, you must choose one of the following actions:

- ▶ Complete the sample following the procedures that are described in 18.5, “Implementing the RedBank application” on page 1005.
- ▶ Import the completed projects from `\7835codesolution\jsp\RAD8Web-JSP.zip`.

18.6.2 Running the sample web application

To run the RedBank web application in the test environment, follow these steps:

1. Right-click **RAD8BankBasicWeb** in the Enterprise Explorer view and select **Run As** → **Run on Server**.
2. In the Server Selection window, select **Choose an existing server** and select **WebSphere Application Server v8.0 Beta**. Select the **Always use this**

server when running my project option so that this step can be skipped next time. Click **Finish**.

The main page of the web application is displayed in a web browser inside Rational Application Developer.

18.6.3 Verifying the RedBank web application

After you start the application by running it on the test server, verify that the web application works properly:

1. From the main page, select the **redbank** menu option.
2. On the RedBank page (Figure 18-40), type a customer's Social Security number (SSN), for example, 222-22-2222.

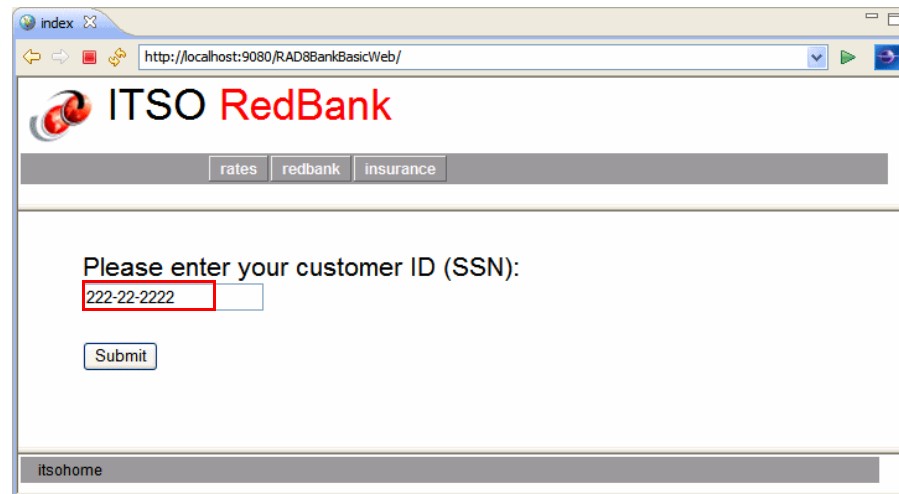


Figure 18-40 ITSO RedBank login page

3. Click **Submit**. The page now lists the customer and accounts (Figure 18-41).

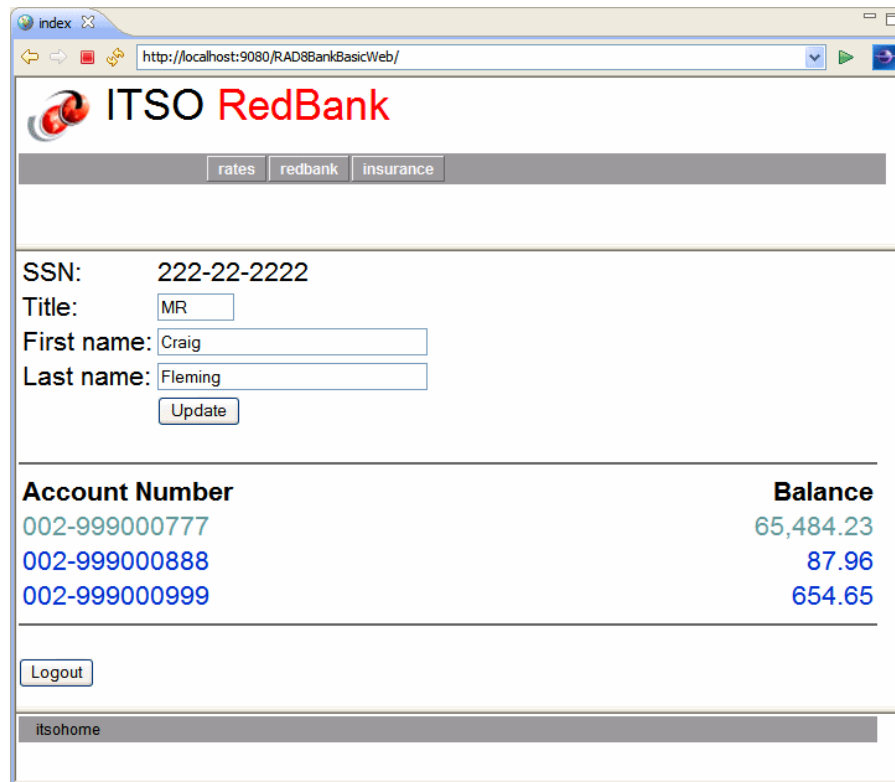


Figure 18-41 Listing of customer accounts

4. From the list of accounts, choose one of the following actions. In this example, we click an account to view the account information.
- Change the customer title or name fields. For example, change the title to **Sir**. Then click **Update** to perform the `doPost` method of the `UpdateCustomer` servlet.
 - Click **Logout** to return to the Login page.
 - Re-enter the Social Security number (SSN), click **Submit**, and verify that the Customer name has changed.
5. Click the link for one of the accounts, and from the account view (Figure 18-42 on page 1053), choose one of the following actions:
- Select **List Transactions** and click **Submit**. There are no transactions yet.

- Select **Deposit** or **Withdraw**, enter an amount, and click **Submit** to execute a banking transaction. The page is redisplayed with the updated balance.
- Select **Transfer** and enter an amount and a target account. Then click **Submit**. The page is redisplayed with the updated balance.
- Click **Customer Details** to return to the account listing.

In this example, we run a few transactions (deposit, withdraw, and transfer). Then we select **List Transactions** and click **Submit**.

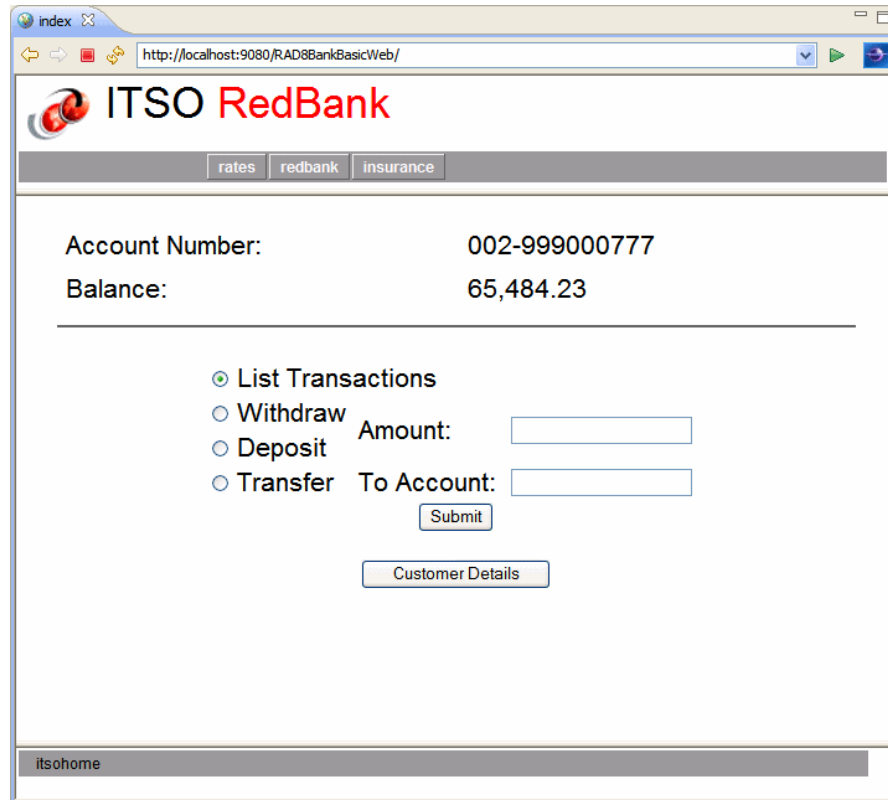
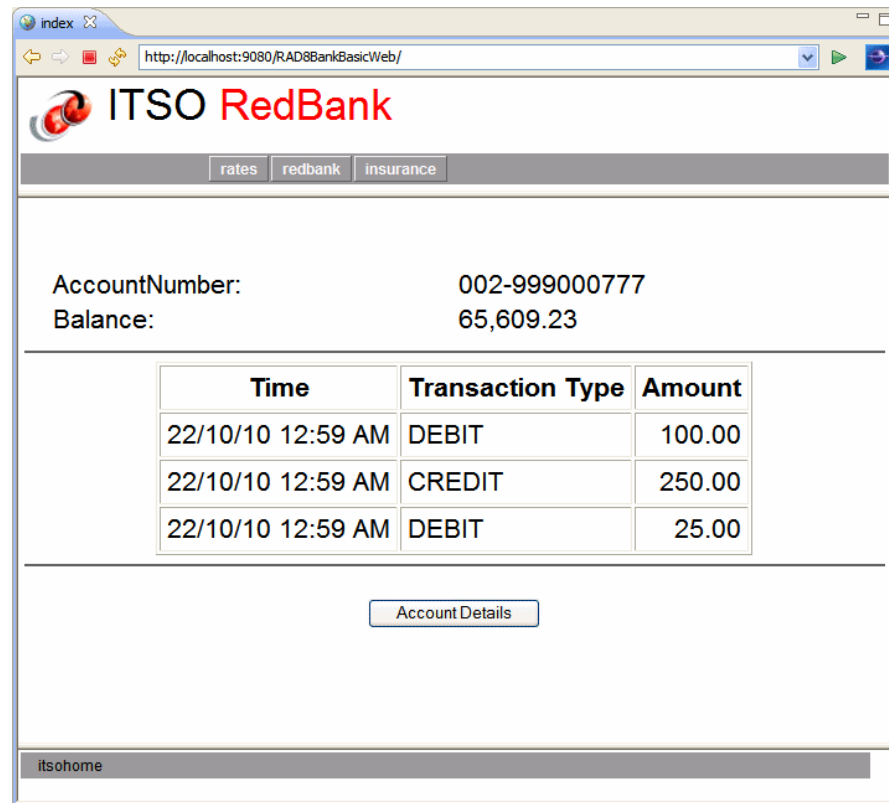


Figure 18-42 Details for a selected account

Figure 18-43 shows the transaction listing.



The screenshot shows a web browser window with the URL `http://localhost:9080/RAD8BankBasicWeb/`. The page header features the ITSO RedBank logo and navigation tabs for `rates`, `redbank`, and `insurance`. The account information is displayed as follows:

AccountNumber: 002-999000777
Balance: 65,609.23

Time	Transaction Type	Amount
22/10/10 12:59 AM	DEBIT	100.00
22/10/10 12:59 AM	CREDIT	250.00
22/10/10 12:59 AM	DEBIT	25.00

Below the table is an `Account Details` button. The footer of the page contains the text `itsohome`.

Figure 18-43 List of transactions for an account

6. Try a withdrawal of an amount greater than the balance. The Show Exception JSP shows an error message (Figure 18-44 on page 1055).

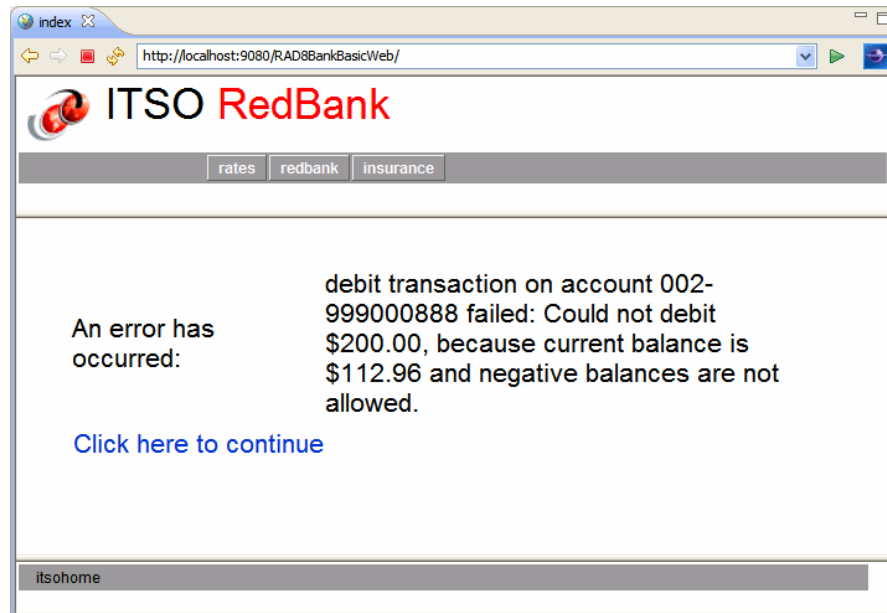


Figure 18-44 Withdrawal over the limit error

18.7 More information

The RedBank application can be improved in many ways, for example, by adding features or using other technologies. We explain these methods in the following chapters:

- ▶ To use a database rather than HashMaps, see Chapter 9, “Developing database applications” on page 393.
- ▶ To use EJB to store the model, see Chapter 12, “Developing Enterprise JavaBeans (EJB) applications” on page 577.
- ▶ To use JSF components rather than JSTL, see Chapter 19, “Developing web applications using JavaServer Faces” on page 1057.
- ▶ To debug the application, see Chapter 28, “Debugging local and remote applications” on page 1461.

The Help feature provided with Rational Application Developer has a large section about developing websites and applications. It contains reference information for all the features presented in this chapter and further information about topics only covered briefly here, including JSP tag libraries and security.

See the following web addresses for further information about the topics in this chapter:

- ▶ Oracle Java Servlet Technology home page, which contains links to the specification, API Javadoc, and articles about servlets:
<http://www.oracle.com/technetwork/java/javaee/servlet/index.html>
- ▶ Oracle JavaServer Pages Technology home page for technical information about JSP:
<http://www.oracle.com/technetwork/java/javaee/jsp/index.html>
- ▶ JavaServer Pages Standard Tag Library (JSTL) home page for technical information about JSTL:
<http://www.oracle.com/technetwork/java/index-jsp-135995.html>
- ▶ “JSP and Servlets best practices,” article that articulates clearly the various ways of applying an MVC pattern to JSP and servlets:
<http://www.oracle.com/technetwork/articles/javase/servlets-jsp-140445.html>
- ▶ *IBM WebSphere Application Server V6.1 Security Handbook*, SG24-6316
<http://www.redbooks.ibm.com/abstracts/sg246316.html?Open>



Developing web applications using JavaServer Faces

JavaServer Faces (JSF) is a framework that simplifies building user interfaces for web applications. In this chapter, we introduce the features, benefits, and architecture of JSF and demonstrate the Rational Application Developer support and tooling for JSF. The chapter includes an example web application using JSF, with persistence implemented in the Java Persistence API (JPA). See Chapter 10, “Persistence using the Java Persistence API” on page 443.

The chapter is organized into the following sections:

- ▶ Introduction to JSF
- ▶ Developing a web application using JSF and JPA
- ▶ More information

The sample code for this chapter is in the `C:/7835codesolution/jsf` folder.

19.1 Introduction to JSF

The JSF framework allows users to make use of pre-built components and easily create web applications. For example, JSF provides an Input Text, Output Text, and a Data Table component. You can add these components easily to a JSF web page and connect them to your project's data. JSF technology and the JSF tooling provided by Rational Application Developer enable even inexperienced developers to quickly develop web applications.

This section provides an overview of the following aspects of JSF:

- ▶ JSF 1.x features and benefits
- ▶ JSF 2.0 features and benefits
- ▶ JSF 2.0 application architecture
- ▶ JSF features in Rational Application Developer

19.1.1 JSF 1.x features and benefits

The JSF 1.x specification is defined in *Java Specification Request (JSR) 127: JavaServer Faces*.

The following list describes the key features and benefits of using JSF for web application design and development:

- ▶ Standards-based web application framework: JSF technology is the result of the Java Community process. JSF makes use of the model view controller (MVC) pattern; it addresses the view or presentation layer through user interface (UI) components, and addresses the model through managed beans.
- ▶ Event-driven architecture: JSF provides server-side rich UI components that respond to client events.
- ▶ UI development:
 - UI components are decoupled from their rendering, which means that they can be extended to use other technologies, such as Wireless Markup Language (WML).
 - JSF allows direct binding of UI components to model data.
 - Developers can use extensive libraries of prebuilt UI components that provide both basic and advanced web functionality. In addition, custom UI components can be created and customized for specific uses.
- ▶ Session and object management: JSF manages designated model data objects by handling their initialization, persistence over the request cycle, and cleanup.

- ▶ Validation and error feedback: JSF allows direct binding of reusable validators to UI components. The framework also provides a queue mechanism to simplify error and message feedback to the application user. These messages can be associated with specific UI components.
- ▶ Globalization: JSF provides tools for the globalization of web applications, including supporting number, currency, time, and date formatting, and the externalization of UI strings.

19.1.2 JSF 2.0 features and benefits

The JSF 2.0 specification is defined in *JSR 314: JavaServer Faces 2.0*. It builds on and extends the features that are available in JavaServer Faces 1.x.

This section describes the major features of JSF 2.0:

- ▶ Facelet usage
- ▶ Built-in Ajax support
- ▶ Annotation usage
- ▶ Creating templates
- ▶ New components
- ▶ Custom components

Facelet usage

JSF 2.0 uses *Facelets*, which are XHTML pages instead of JSP pages, as the view layer. Facelets relieve JSF of the restrictions that are imposed by JSP technology. For more information, see “Improving JSF by Dumping JSP” by Hans Bergsten in O’Reilly on Java.com, 9 June 2004:

<http://onjava.com/pub/a/onjava/2004/06/09/jsf.html>

Facelets: Facelets are the standard view decoration language for JSF application development in JSF 2.0. We do not recommend combining Facelets and Faces JSP pages in the same project.

Built-in Ajax support

In JSF V1.x, it was possible to use Ajax with JSF, but this capability required additional component libraries. Now with JSF 2.0, a JavaScript library for performing simple Ajax operations is automatically provided. No additional libraries are required. The example application in this chapter uses Ajax.

Annotation usage

With JSF 2.0, Java classes can be directly annotated, which eliminates the need to register these classes in `faces-config.xml`. For example, `@ManagedBean`

indicates that this class is a Faces Managed Bean. The `@ManagedBean` annotation can be used in place of a managed-bean entry in `faces-config.xml`.

Other annotations, such as `@FacesComponent` and `@FacesRenderer`, are also available.

Creating templates

Facelet pages can be created from templates to allow a more uniform look across your project and to aid with future maintenance. We describe the creation of these templates in 19.2.3, “Creating Facelet templates” on page 1069.

New components

JSF provides a variety of components for use in your web pages. In addition to the components that were already available in JSF 1.x (Input Text, Output Text, Data Table, and so on), JSF 2.0 includes two new components that simplify GET navigation: `<h:link>` and `<h:button>`.

For more information about components that are available on Facelet pages, see this website:

<http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.etools.jsf.doc/topics/tcrtfaceletspgcontent.html>

Custom components

Composite components build on Facelets' templating features so that you can implement custom components. The advantage to custom components is that you can implement them without any configuration and without any Java code.

More detailed information and examples are available using the following link:

<http://www.ibm.com/developerworks/java/library/j-jsf2fu2/index.html>

The steps for creating and customizing custom components with the Rational Application Developer are described in detail in the information centers:

- ▶ <http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.etools.jsf.doc/topics/tjsfover.html>
- ▶ <http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.etools.jsf.doc/topics/tcrtfaceletcomposite.html>

19.1.3 JSF 2.0 application architecture

You can extend the JSF application architecture easily in a variety of ways to suit the requirements of your particular application. You can develop custom

components, renderers, validators, and other JSF objects and register them with the JSF run time.

This section highlights the JSF 2.0 application architecture, as shown in Figure 19-1.

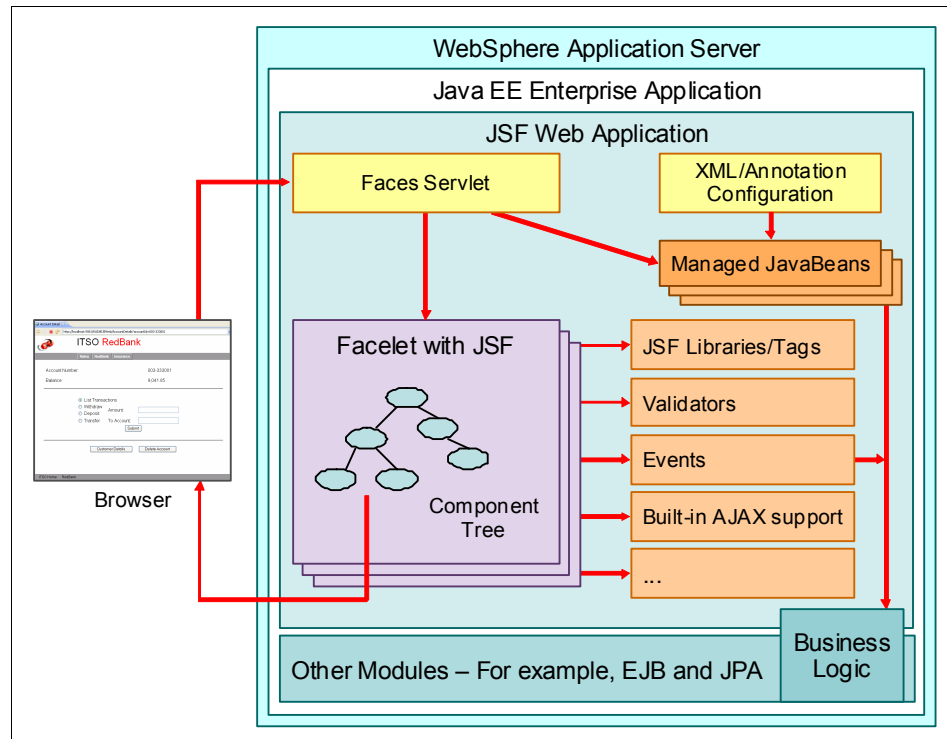


Figure 19-1 JSF application architecture

The JSF application architecture includes these components:

- ▶ **Facetlet pages:** These pages are built from JSF components, where each component is represented by a server-side class.
- ▶ **Faces servlet:** One servlet (`FacesServlet`) controls the execution flow.
- ▶ **XML/Annotation Configuration:** The configuration of validators and Faces managed beans can be defined either with the XML file `faces-config.xml` or by using annotations.
- ▶ **Tag libraries:** The JSF components are implemented in tag libraries.
- ▶ **Validators:** Java classes are used to validate the content of JSF components. For example, user input can be validated according to specific business logic.

- ▶ Faces managed beans: JavaBeans are defined in the configuration file to hold the data of JSF components. Faces managed beans represent the data model and are passed between the business logic and user interface.
- ▶ Events: Java code is executed in the server for events, such as pushing a button and invoking business logic.
- ▶ Ajax: Ajax is supported by JSF 2.0 as a built-in feature.

Figure 19-2 represents the structure of a simple JSF 2.0 application, in this case, our RAD8JSFWeb project.

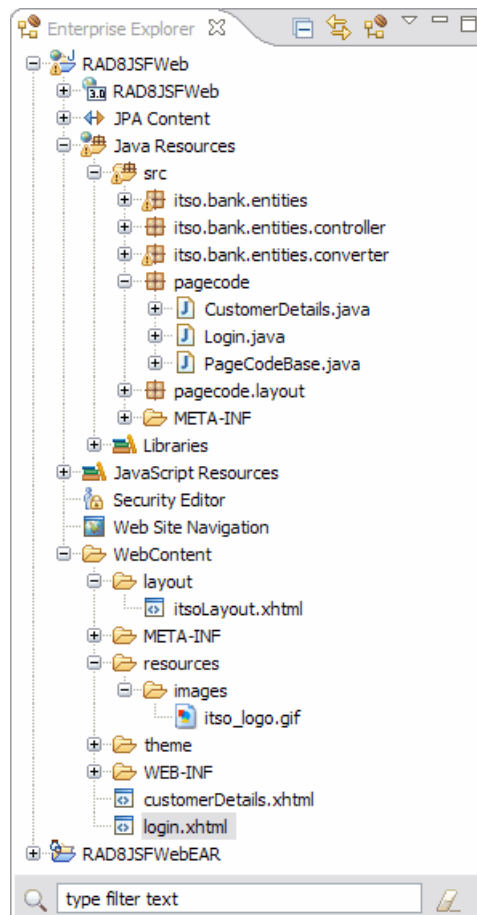


Figure 19-2 JSF 2.0 application structure within Rational Application Developer

19.1.4 JSF features in Rational Application Developer

Rational Application Developer provides a number of rich features to enhance your usage of the JSF framework:

- ▶ JSF Trace
- ▶ Integration of third-party JSF tag libraries
- ▶ Customizable data templates

JSF Trace

For help debugging your JSF application or reviewing the phases of the JSF life cycle, you can use JSF Trace. This feature visually displays information about your application at run time, including incoming requests, error messages, and objects in the request, session, and application scopes.

JSF Trace is covered in detail in “Debug and troubleshoot JavaServer Faces applications by using JSFTrace in Rational Application Developer” by Yury Kats in IBM developerWorks, 24 September 2009:

<http://www.ibm.com/developerworks/rational/library/09/debugjavaserverfaces/traceapplicationdeveloper/index.html>

Integration of third-party JSF tag libraries

JSF has a set of standard components for building web pages. These standard components can be supplemented with components from JSF tag libraries that have been created by other companies. Rational Application Developer makes it easy to integrate these third-party tag libraries, and even to customize the tooling for these tags.

For more information, see “Faces library definitions for third-party JavaServer Faces controls” by Scott Paxton in IBM developerWorks, 18 June 2009:

<http://www.ibm.com/developerworks/rational/library/09/faceslibrarydefinitionrationalapplicationdeveloper/index.html>

Customizable data templates

Rational Application Developer makes it easy to generate UI components based on your project’s data structures. Simply by adding a data source, such as a Faces managed bean, to the Page Data view and then dragging it onto your web page, you can create controls that are connected to that data source.

For even more control over the controls that are generated on your web page, see “Introduction to JavaServer Faces data templates” by Christie Rice in IBM developerWorks, 25 September 2009:

http://www.ibm.com/developerworks/rational/library/09/intro_jfs_data_templates_rad/index.html

19.2 Developing a web application using JSF and JPA

In this section, we describe a web application that is implemented with JavaServer Faces (JSF) and Java Persistence API (JPA). For each Facelet that we create, a managed bean class is generated. For each action in the Facelet, a method in the managed bean class is invoked. In those methods, we will use the JPA Manager Beans to retrieve the necessary data.

Rational Application Developer provides tooling to interact directly with the JPA entities without using a session bean. A JPA Manager Bean is created for each JPA entity with methods, such as find and update.

Structure of the JSF web application

The sample application consists of the following pages:

- ▶ Login page (login): Validates a customer’s unique ID. If the ID is valid, display the customer details page.
- ▶ Customer details page (customerDetails): Shows details (title, first name, and last name) of a customer and the associated account balances.

You can find a completed version of the web application in the `c:\7835codesolution\jsf\RAD8JSFWeb.zip` project file.

19.2.1 Setting up the ITSOBANK database

The JPA entities are based on the ITSOBANK database. Therefore, we must define a database connection within Rational Application Developer that the mapping tools use to extract schema information from the database.

See “Setting up the ITSOBANK database” on page 1880 for instructions to create the ITSOBANK database. For the JPA entities, we can either use the DB2 or Derby database. For simplicity, we use the built-in Derby database in this chapter. In addition, you have to create a connection to the database ITSOBANKderby, as described in 9.2.2, “Creating a connection to the ITSOBANK database” on page 395.

Configuring the data source

You can choose from one of the following methods to configure the data source:

- ▶ Use the WebSphere administrative console.
- ▶ Use the WebSphere Enhanced EAR, which stores the configuration in the deployment descriptor and is deployed with the application.

While developing JSF and JPA web applications with Rational Application Developer, the data source is created automatically when you add JPA-managed data to a Facelet file. The data source configuration is added to the EAR deployment descriptor.

However, if you have already defined the ITS0BANKderby data source on the server (see “Configuring the data source in WebSphere Application Server” on page 1882), you might experience problems, because you can only have one active connection to the database. To clear the connection, remove all applications from the server and then restart the server.

19.2.2 Creating the JSF Project

In this section, we describe how to create a dynamic web JSF project.

This application consists of RAD8JSFWebEAR (the enterprise application) and RAD8JSFWeb (the web application). Follow these steps:

1. In the Enterprise Explorer view, right-click and select **New → Project**.
2. In the New Project wizard, select **Web → Dynamic Web Project** and click **Next**.
3. In the New Dynamic Web Project wizard, define the project details (Figure 19-3 on page 1066):
 - a. For the Project name, type RAD8JSFWeb.
 - b. Leave **Use default location** checked.
 - c. For the Target Runtime, select **WebSphere Application Server v8.0 Beta**.
 - d. Leave the Dynamic web module version at **3.0**.
 - e. For the Configuration, select **JavaServer Faces v2.0 Project** and click **Modify**.

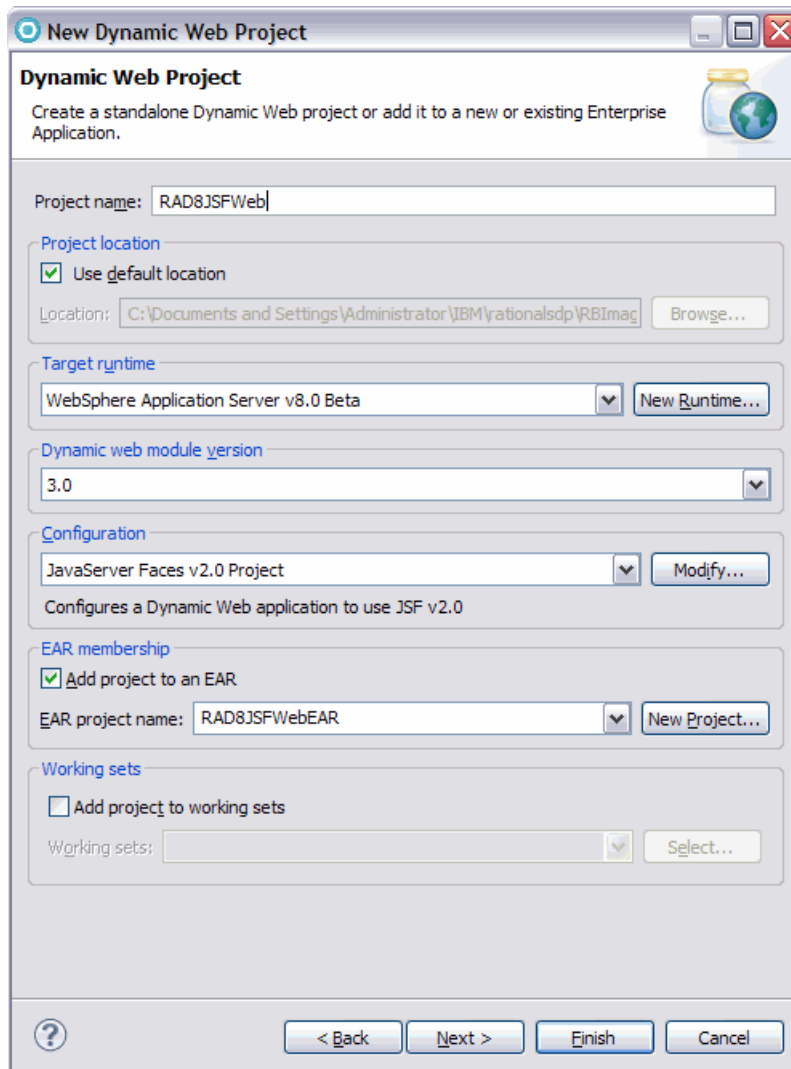


Figure 19-3 New Dynamic Web Project wizard

- f. In the Project Facets window, select the following Project Facets (Figure 19-4 on page 1067):
- **Dynamic Web Module:** Preselected
 - **Java:** Preselected
 - **JavaServer Faces:** Preselected
 - **JPA:** Select this facet
 - **WebSphere Web (Co-existence):** Preselected
 - **WebSphere Web (Extended):** Preselected

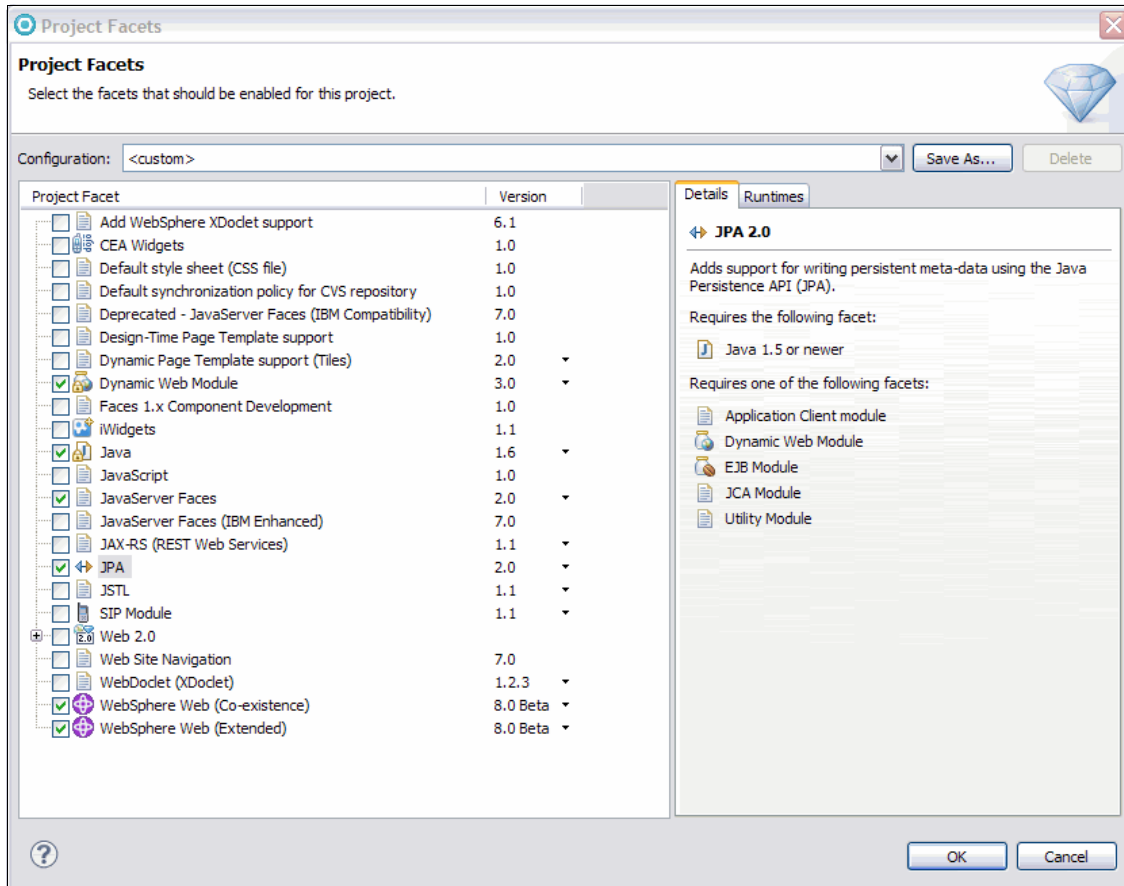


Figure 19-4 Web project facets for JSF

- g. Click **OK**. The value of Configuration in the Dynamic Web Project window changes to *<custom>*.
 - h. Click **Next**.
4. Accept the source folder `src` and click **Next**.
 5. Select **Generate web.xml deployment descriptor** and click **Next**.
 6. In the JPA Facet page, complete these steps:
 - a. Leave **RAD JPA 2.0 Platform** selected for the Platform.
 - b. Leave **Library Provided Target Runtime** selected for the Type.
 - c. For the connection, select the **ITSOBANKderby** that was defined before. Complete these tasks:
 - Click **Connect** if the connection is not active.

- In case you have not created the connection to the ITSOBANK database yet, click **New Connection** to define it. See 9.2.2, “Creating a connection to the ITSOBANK database” on page 395 for details about creating this connection.
- d. Select **Override default schema from connection** and select **ITSO**.
- e. For Persistence class management, select **Discover annotated classes automatically**.
- f. Select **Create mapping file (orm.xml)**. See Figure 19-5.

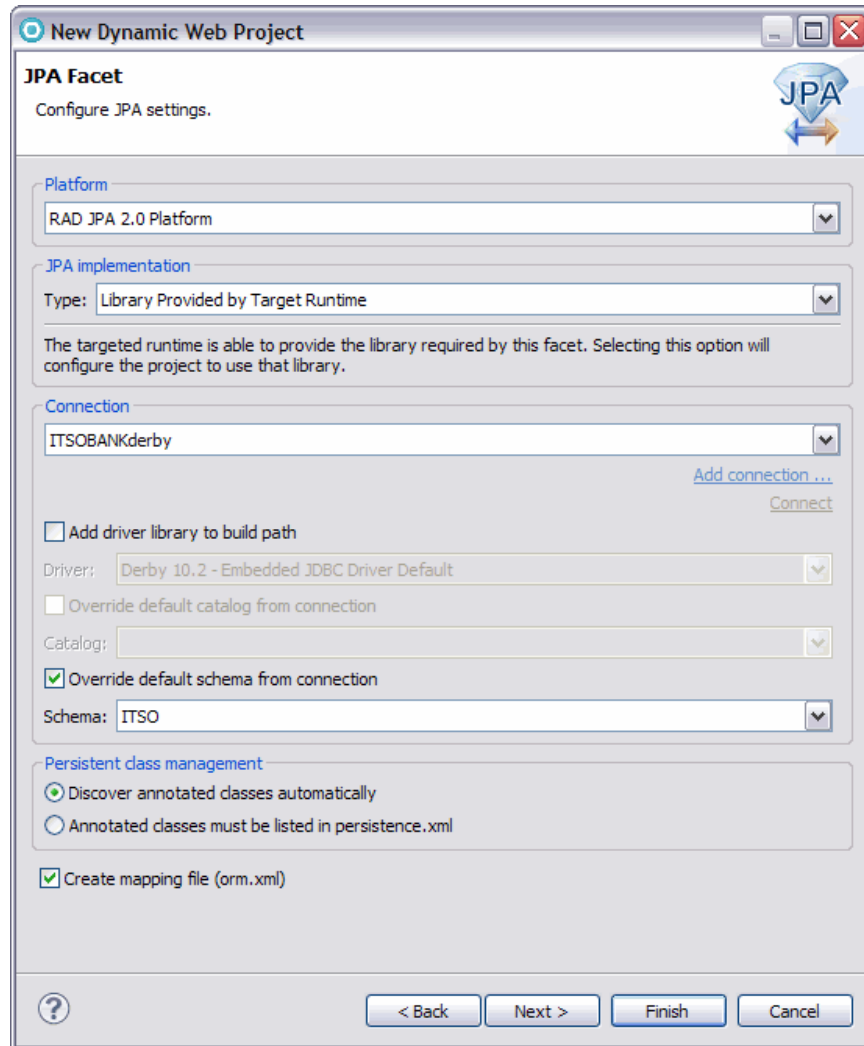


Figure 19-5 JPA Facet settings

- g. Click **Next**.
7. In the JSF Capabilities page, leave **Default JSF Implementation** selected.
8. Click **Finish** and the project will be created for you.
9. Switch to the **Web perspective** when prompted.
10. **Close** the Technology Quickstarts.

19.2.3 Creating Facelet templates

Before we create the Facelet web pages, `login` and `customerDetails`, we define the template that these pages will use. This template helps to provide a consistent look and feel to the application by creating universal header and footer sections.

Creating the template

Follow these steps to create the template:

1. In the Enterprise Explorer, expand **RAD8JSFWeb** and select the folder **WebContent**.
2. Right-click folder **WebContent**. Select **New** → **Folder** and type `layout` as the name.
3. Right-click folder **layout** and select **New** → **Web Page**.
4. In the New Web Page window, specify the details for this web page (Figure 19-6 on page 1070):
 - a. Type `itsLayout` for the File Name.
 - b. Select **Facelet** as the type of Template.

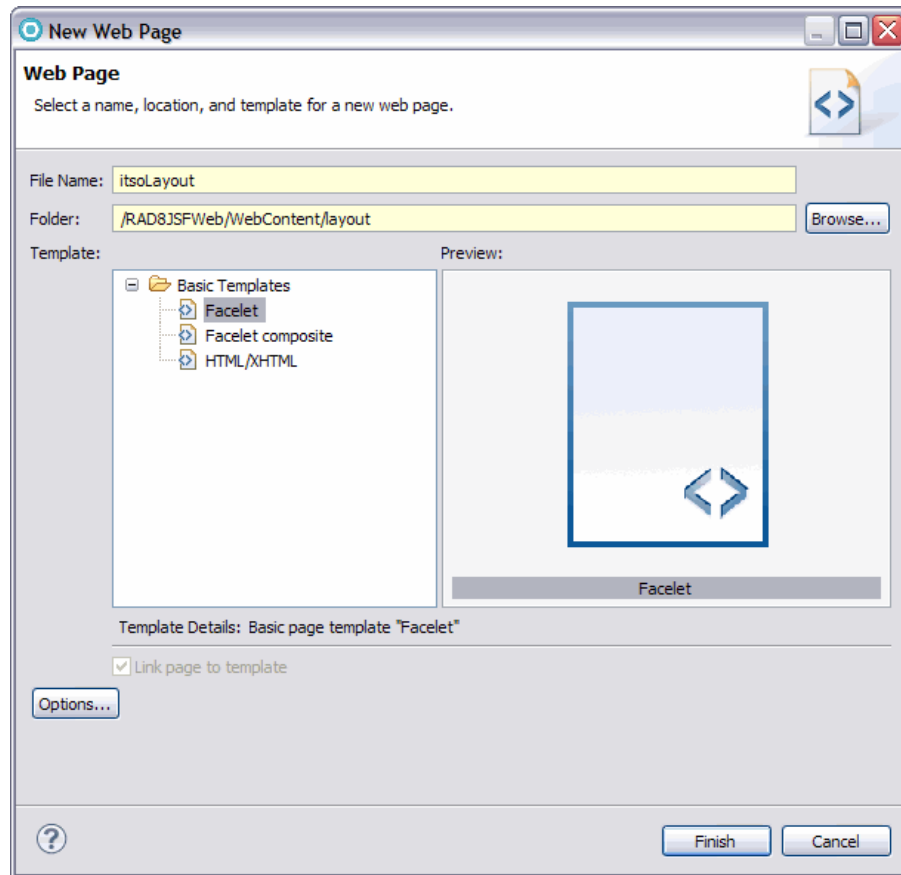


Figure 19-6 Create itsoLayout

5. Click **Finish** and the `itsoLayout.xhtml` file opens.

Creating the header

First, we create the header for our template:

1. In the Palette view, go to the Standard Faces Components drawer and select **Panel - Grid**.
2. Drag and drop this Insert tag to the `itsoLayout` page (Figure 19-7 on page 1071):
 - a. In the Design view, you see a visualization of the new tag with the text:

`grid1: Drag and Drop Items to this area to populate this region.`
 - b. Click this text and open the **Properties** view.

- c. The **h:panelGrid** tab is selected in the Properties view. Enter 2 for the number of Columns.

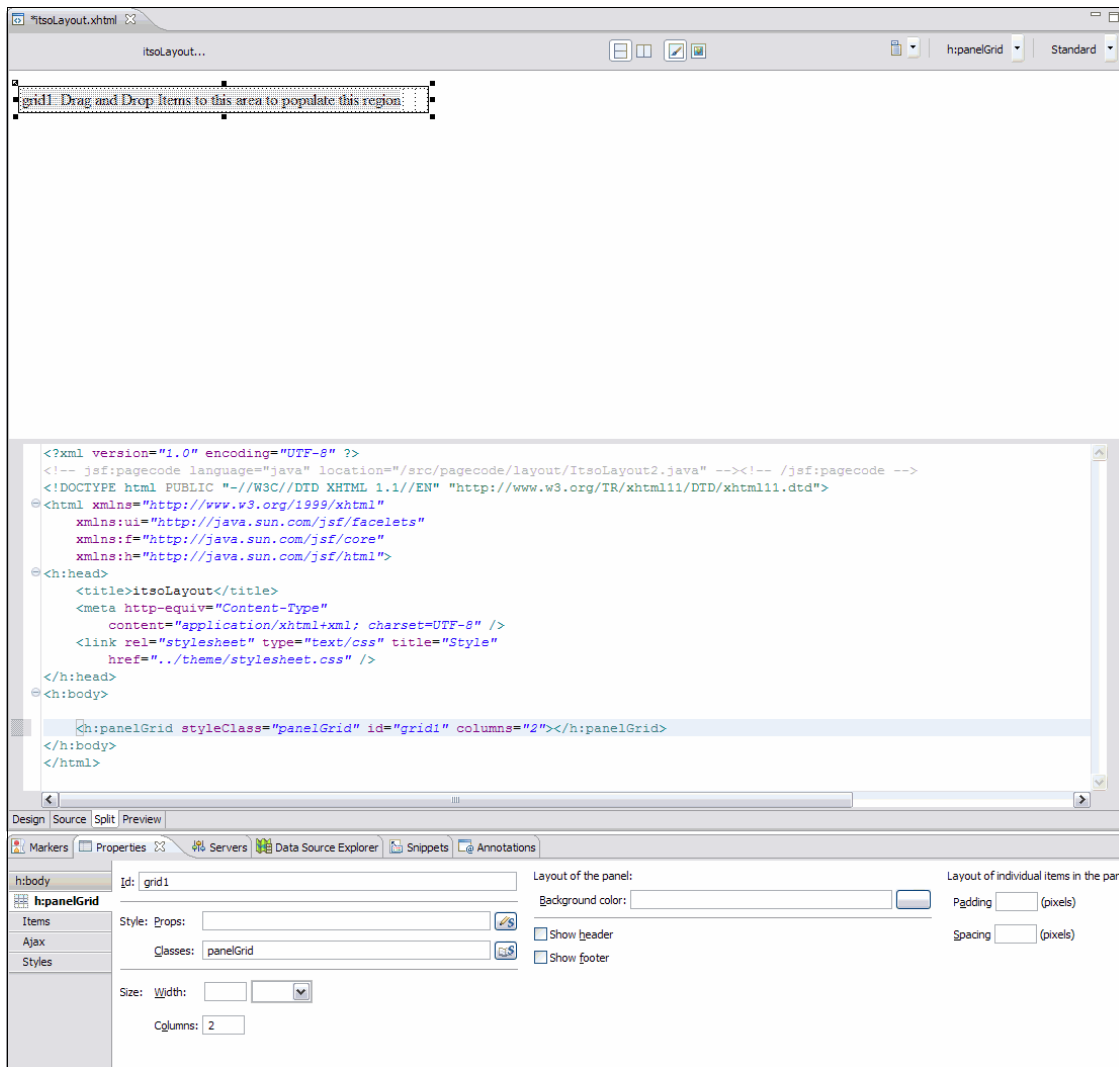


Figure 19-7 Properties view for **h:panelGrid**

3. We import the logo that we use in the page header. In the Enterprise Explorer, right-click **WebContent** under **RAD&JSFWeb** and select **New** → **Folder**.
4. Enter resources for the folder name and click **Finish**.
5. We create a folder called **images** inside the resources folder. Right-click the new **resources** folder and select **New** → **Folder**.

6. Enter images for the folder name and click **Finish**.
7. Now that the folders have been created, we can import the logo. Right-click **images** and select **Import**. Click **General** → **File System**. Click **Next**.
8. Locate **itso_logo.gif** on your file, which is in C:/7835code/jsf. Browse to the location of this file. Click **Finish**.
9. The file is imported (Figure 19-8).

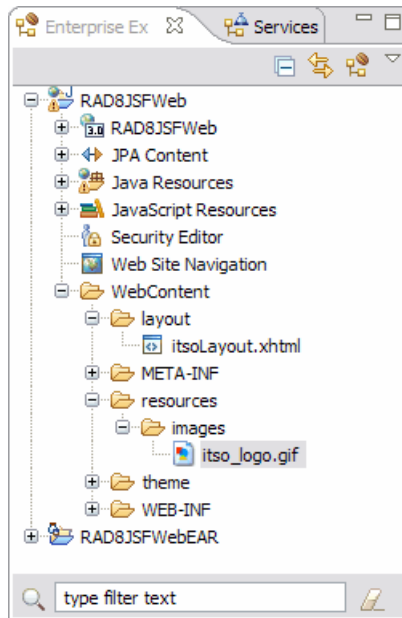


Figure 19-8 *itso_logo* imported into the project

10. Drag an image from the Standard Faces components drawer of the Palette, dropping it on top of the text **grid1**.
11. Go to the **Properties** view. The **h:graphicImage** tab is selected. Click **Browse** next to Name. In the Select an image resource dialog window, select **itso_logo.gif** from under **images** and click **OK**. The logo is visible on your page.
12. Switch to the **Accessibility** tab of the Properties view, which is located under **h:graphicImage**. Enter ITS0 1ogo for the Alternate Text.
13. Drag an Output from the Palette and drop it on the right side of the logo.
14. Go to the **Properties** view. The **h:outputText** tab is selected. Type ITS0 RedBank for the Value.
15. Look at the source for *itsoLayout*. Find the title tag. Change the text from *itsoHeader* to ITS0 RedBank.

16. We have now created the header for our page, showing the bank's logo and a title. Your page looks like Figure 19-9.



Figure 19-9 *itsoLayout* with header

17. Save the page.

Creating the content area

Complete these steps to create the content area:

1. We create a content area to hold the content of our login and customerDetails pages. Expand the **HTML Tags** drawer in the Palette. Drag a **Horizontal Rule** to the page and drop it beneath the Panel Grid.
2. Expand the **Facelet Tags** drawer. Drag an **Insert** to the page and drop it beneath the Horizontal Rule.
3. Go back to the **HTML Tags** drawer. Drag a second **Horizontal Rule** and drop it at the bottom of the page.
4. Click the text **Drop controls here for content area “< no name >”** and go to the **Properties** view. The **ui:insert** tab is selected. Enter `pageContent` as the Name, as shown in Figure 19-10 on page 1074.

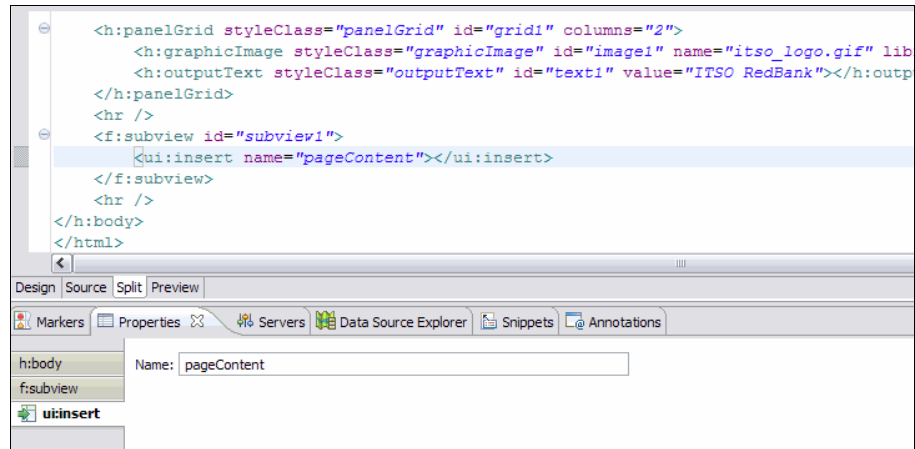


Figure 19-10 its0Layout with content area

Creating the footer

Complete these steps to create the footer:

1. We create a footer for the page, which will consist of a simple text string. Drag an **Output** from the Standard Faces Components drawer of the Palette and drop it at the bottom of the page.
2. Go to the **Properties** view. Enter Created by ITS0, 2010 for the Value (Figure 19-11 on page 1075).

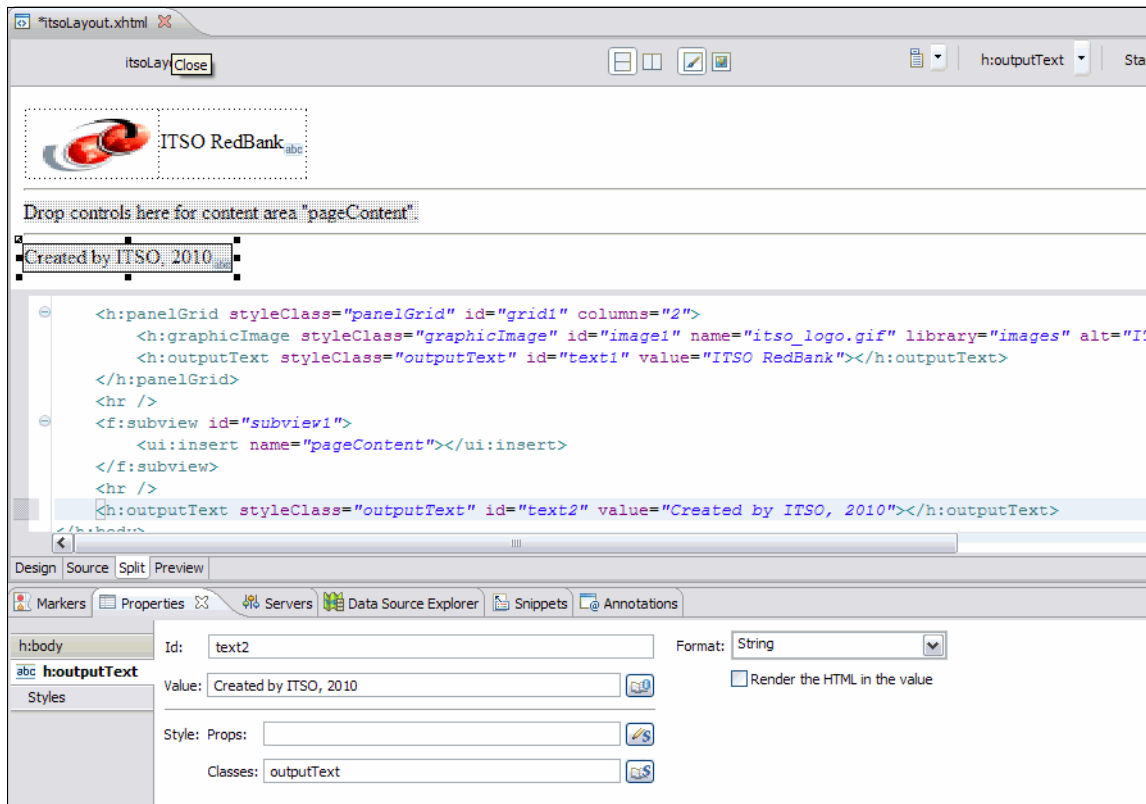


Figure 19-11 *itsoLayout with footer*

3. Save the page.

19.2.4 Creating Facelets

In this section, we create the `login.xhtml` and `customerDetails.xhtml` Facelets based on our layout template, `itsoLayout.xhtml`:

1. In the Enterprise Explorer, expand **RAD8JSFWeb** and select the folder **WebContent**.
2. Right-click **WebContent** and select **New** → **Web Page**. Complete these tasks:
 - a. Type `login` for the File Name.
 - b. Select your defined template **itsoLayout.xhtml** in the folder layout under **MyTemplates**.

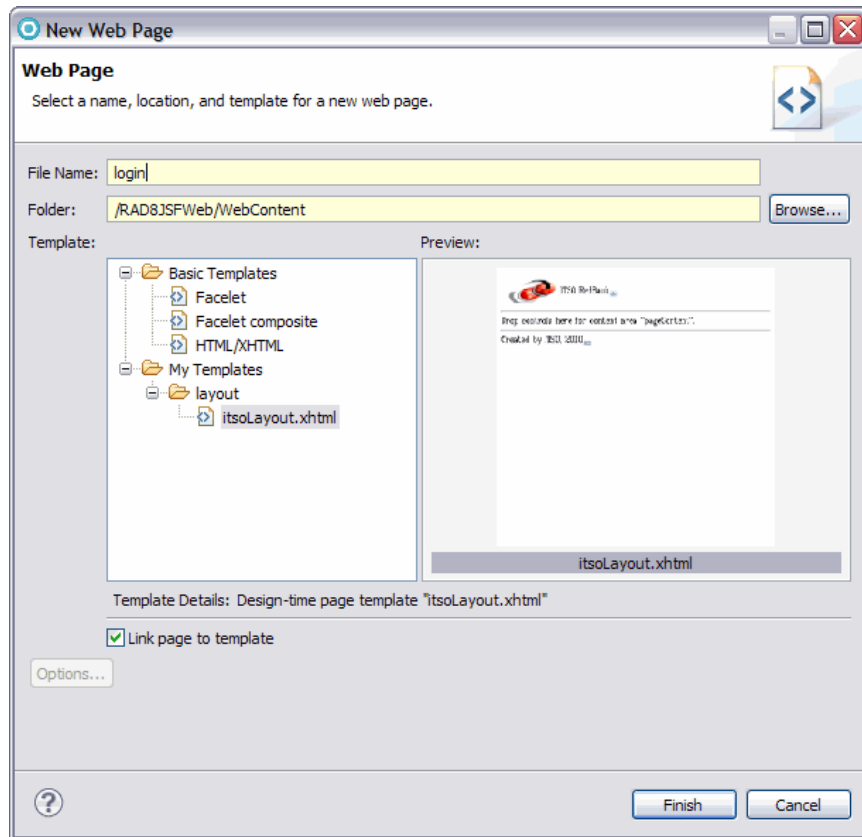


Figure 19-12 Create login.xhtml based on a template

3. Click **Finish**, and the login.xhtml file opens.
4. Use the same steps to create the customerDetails.xhtml Facelet.

19.2.5 Creating JPA Manager Beans

In this section, we create JPA Manager Beans and JPA entities for the ITS0BANK database.

Creating entities

Complete these steps to create the entities:

1. If the server is running and is connected to the ITS0BANK database, stop the server.
2. Open **customerDetails.xhtml** and go to the **Page Data** view.

3. Expand **JPA**. Right-click **JPA Manager Beans** and select **New** → **JPA Manager Bean** (Figure 19-13).

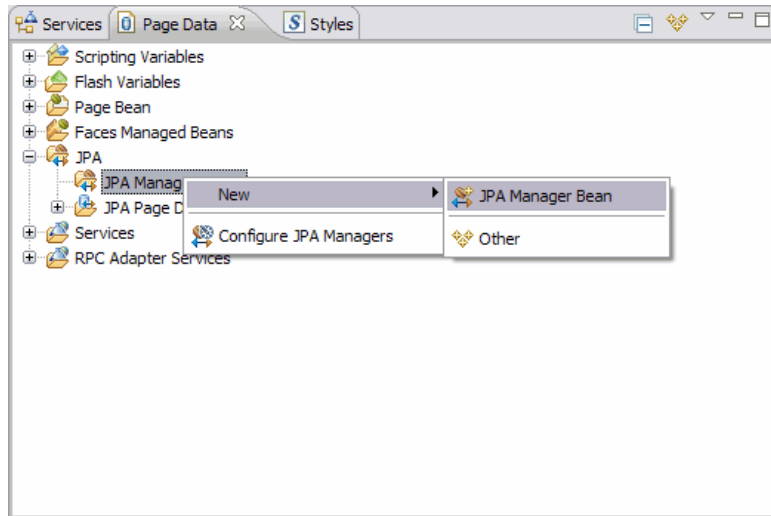


Figure 19-13 Creating a new JPA Manager Bean from the Page Data view

4. In the JPA Manager Bean Wizard, click **Create New JPA Entities**.

Important: To retrieve records from the relational database, we require a connection. We use the `ITSOBANKderby` connection that was created in 19.2.2, “Creating the JSF Project” on page 1065.

5. In the Generate Custom Entities wizard, define the connection, schema, and tables (Figure 19-14 on page 1078):
 - a. For the Connection, select **ITSOBANKderby**.
 - b. For the Schema, select **ITSO** (click **Connect** if you do not see this schema listed).
 - c. Click the **Select All** icon to select the four tables.
 - d. Select **Update class list in persistence.xml** so that the generated classes are added to the file.

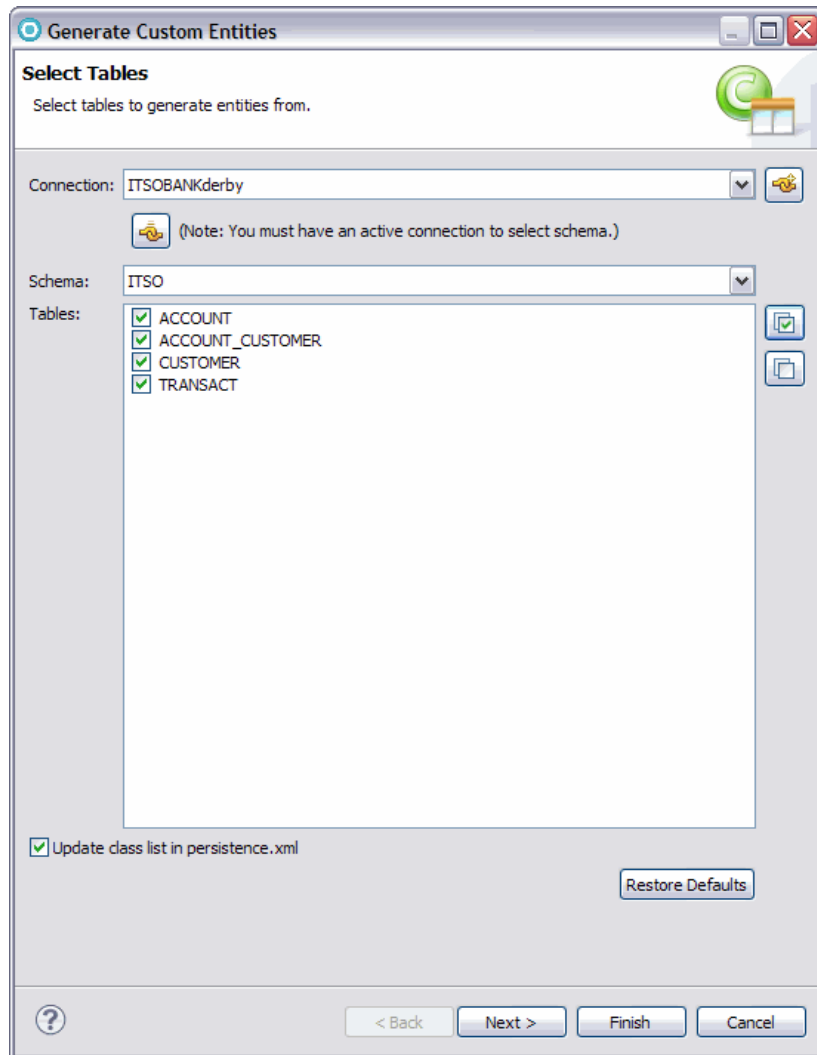


Figure 19-14 Select tables

- e. Click **Next**.
6. Click **Next** on the Table Associations page.
7. In the Customize Default Entity Generation page, define the Table mapping and the package name (Figure 19-15 on page 1079):
 - a. For the Table mapping definition, for the Key generator, select **none**.
 - b. For the Entity access, select **Field**.
 - c. For the Associations fetch, select **Default**.

- d. For the Collection properties type, select **java.util.List**.
- e. Clear **Always generate optional JPA annotations and DDL parameters**.
- f. For the Package, type `itso.bank.entities`.

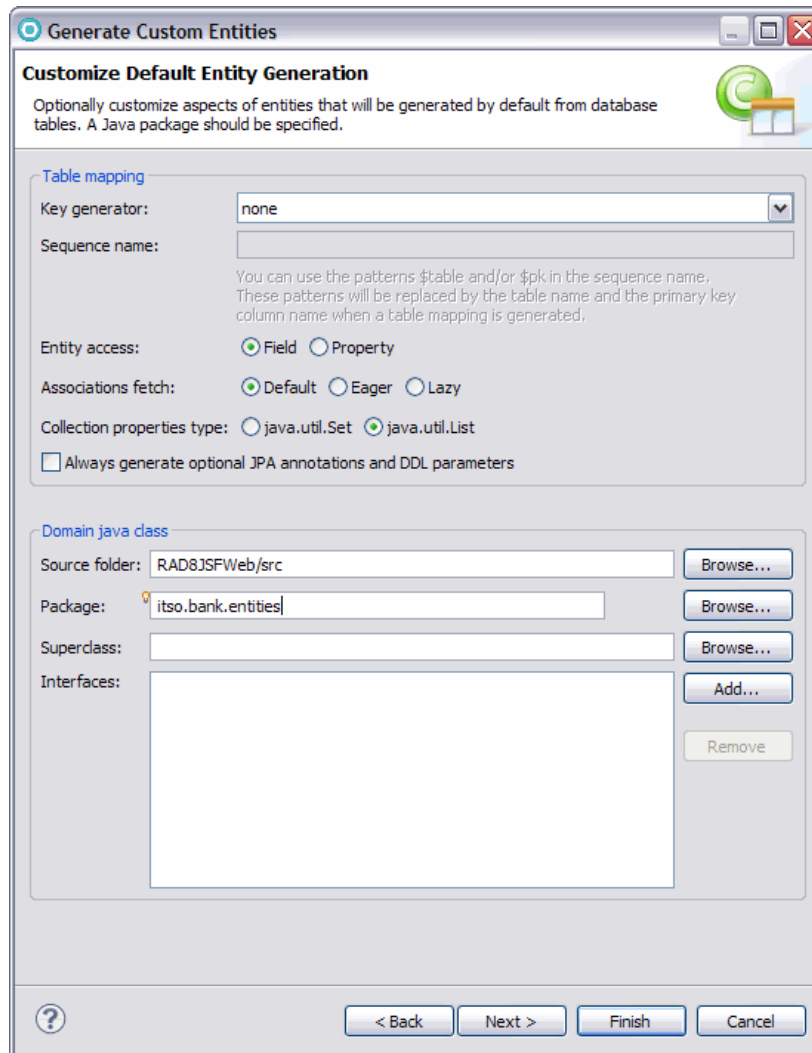


Figure 19-15 Customize Default Entity Generation

- g. Click **Next**.

8. In the Generate Custom Entities: Customize Individual Entities window, define the class names (Figure 19-16):
 - a. Select **TRANSACTION** in the Tables and columns pane.
 - b. The default class name is Transact. Change the class name to Transaction.

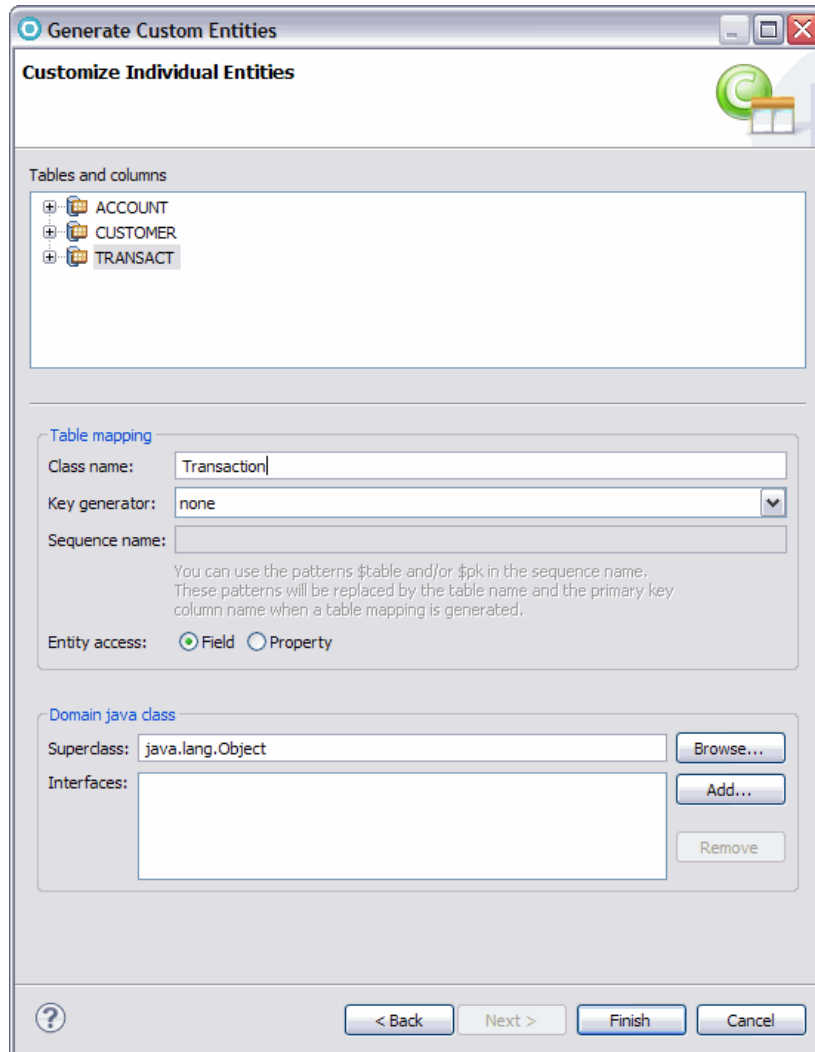


Figure 19-16 Customize Individual Entities

9. Click **Finish**.

Editing the Customer entity

Complete these steps to edit the Customer entity:

1. In the JPA Manager Bean Wizard window, the Account, Customer, and Transaction entities are displayed.
2. Select the **Customer** entity and click **Edit Selected Entities**.
3. In the Tasks page, go through the tasks, and make the following selections:
 - a. For the Primary key, ensure that **ssn** is selected.
 - b. For the Relationships, ensure that **Account** is selected.
 - c. For Named Queries, click **Add**.
 - d. In the Add Named Query dialog box, ensure that **getCustomer** is the Named Query Name and then click **OK**.
 - e. For Concurrency Control, ensure that **No Concurrency Control** is selected.
 - f. Leave all check boxes unchecked for the Other task.
 - g. Select **Automatically set up initial values for JDBC Deployment**.
 - h. Click **Finish** to return to the JPA Manager Bean Wizard window.
4. Click **Next** in the JPA Manager Bean Wizard window.
5. In the Tasks window, click the **Other** task.
6. Select **I want the container to inject the persistence unit into my beans** and check **Generate JSF Converter** for target entity.
7. Click **Finish**.

Editing the Account entity

We add additional JPA managers. We create a JPA Manager for Accounts:

1. In the Page Data view on `customerDetails.xhtml`, right-click **JPA Manager Beans** and select **Configure JPA Manager Beans**.
2. In the JPA Manager Bean wizard, click **Create new JPA Manager**.
3. Select **Account** and then click **Edit Selected Entities**.
4. Click **Named Queries**.
5. Click **Add**.
6. On the Add Named Query dialog window, change the name to `getAccountBySSN` and change the query statement to `select a from Account a, in(a.customers) c where c.ssn =:ssn order by a.id`. This code snippet is available in `C:/7835code/jsf/getAccount.txt`.
7. Click **OK**, **Finish**, and then **Finish** again.

8. Click **Cancel**.
9. The `AccountManager` bean has now been created for you.

19.2.6 Creating JPA page data

Now we add JPA page data so that the JSF components can interact with the JPA data. We want a single customer record and a list of accounts.

Customer record

Follow these steps:

1. Open `customerDetails.xhtml`. In the Page Data view, right-click **JPA** and select **New** → **JPA Page Data**.
2. In the Add JPA data to page dialog window, select **CustomerManager** and select **Retrieve a single record**.
3. Click **Next** twice.
4. On the Set Filter Values page, change the primary key value to `#{sessionScope.customerId}`. Click **Finish**.

Account list

Follow these steps:

1. In the Page Data view, right-click **JPA** and select **New** → **JPA Page Data**.
2. In the Add JPA data to page dialog window, select **AccountManager** and select **Retrieve a list of data**.
3. Click **Next** twice.
4. On the Set Filter Values page, change the primary key value to `#{sessionScope.customerId}`. Click **Finish**.
5. Save the page.

19.2.7 Editing the login page

Next we complete our login page. We add UI components, simple validation, and navigation to go from `login.xhtml` to `customerDetails.xhtml`.

Adding UI components

Instead of adding each UI component individually, we instead define data for the customer ID and have Rational Application Developer generate the necessary UI for us:

1. Open **login.xhtml**. Open the **Page Data** view. Right-click **Scripting Variables** and select **New** → **Session scope variable**.
2. In the Add Session Scope Variable dialog window, enter `customerId` for the Variable name and `java.lang.String` for the Type (Figure 19-17).

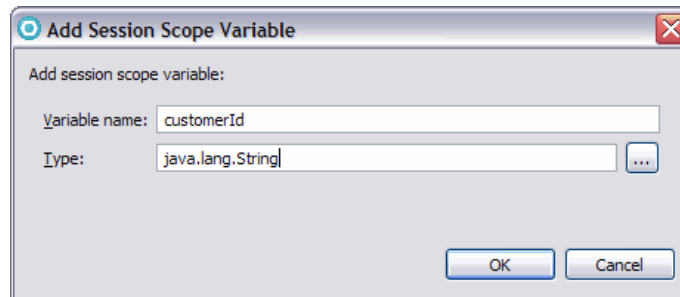


Figure 19-17 Add Session Scope Variable

3. Click **OK**.
4. In the Page Data view, expand **Scripting Variables** → **sessionScope**. Click **customerId** and drag it to the page, dropping it on top of “Drop controls here”.
5. In the Insert JavaBean wizard, select **Inputting data**.
6. Change the label from `CustomerId` to `Enter your customer ID:` (Figure 19-18 on page 1084).

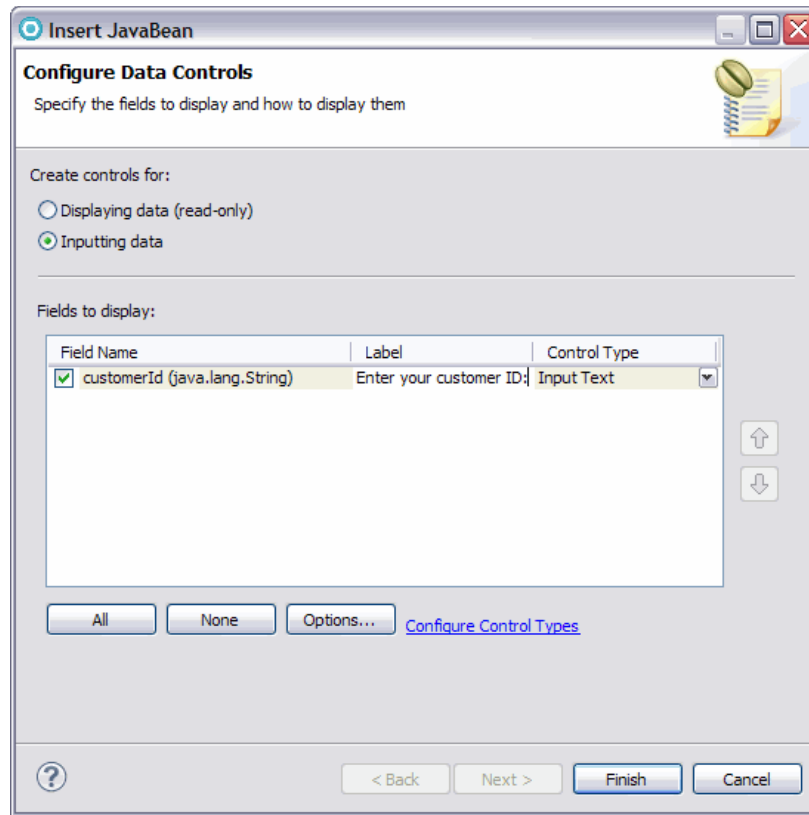


Figure 19-18 Create a login input field

7. Click **Options**. Select the **Buttons** tab of the Options dialog window, and change the label to Login.
8. Click **OK** and then click **Finish**. Figure 19-19 on page 1085 shows the login page.

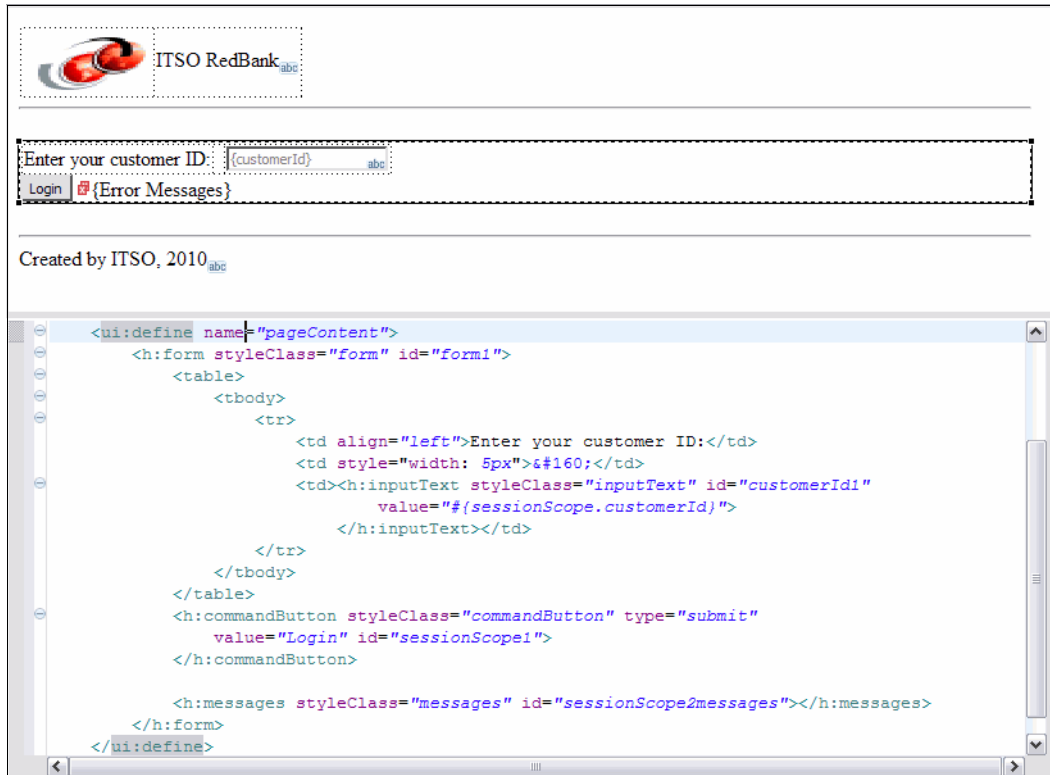


Figure 19-19 Login page with login input text

9. Save the page.

Adding validation

JSF allows you to add simple validation to your web page easily. Here, we ensure that the user types an 11-digit number for the customerId:

1. Click the **customerId** Input Text.
2. Go to the **Properties** view and switch to the **Validation** tab (under **h:inputText**).
3. Check **Value is required**.
4. Enter 11 for both the Minimum length and the Maximum length (Figure 19-20 on page 1086).

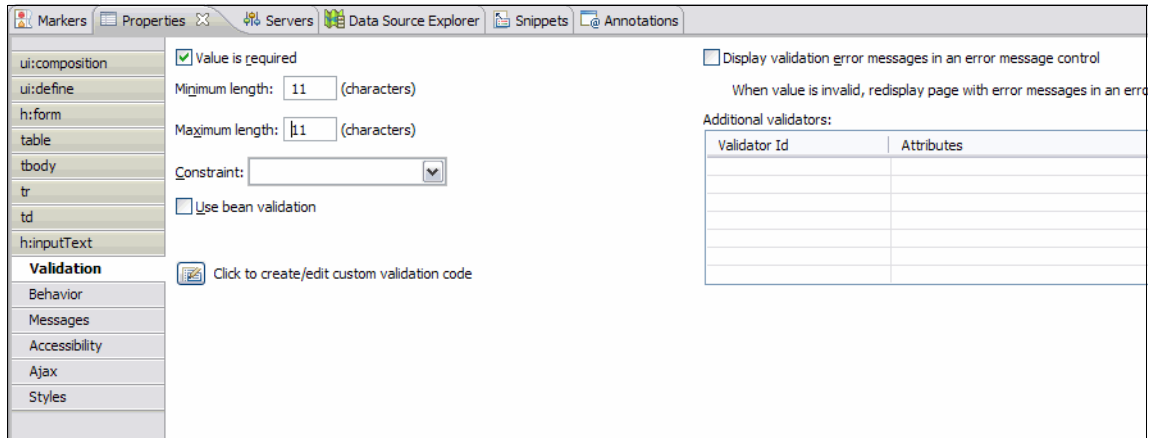


Figure 19-20 Validation tab of the Properties view

5. Save the page.

Verifying the customer ID

We want to ensure that the customerId is valid and that our RedBank application has a customer with that ID:

1. In the Enterprise Explorer, select **RAD8JSFWeb** → **Java Resources** → **src** → **pagecode** → **Login.java**. This is a Faces managed bean that was created automatically by Rational Application Developer. We add a method that will run when the button is clicked.
2. Paste in the code that is shown in Example 19-1. This code is available in C:/7835code/jsf/LoginButton.txt.

Example 19-1 LoginButton

```
public String doLoginAction() {
    try {
        String id = (String) getSessionScope().get("customerId");
        System.out.println("Logon id: " + id);
        CustomerManager customerManager =
        (CustomerManager)getManagedBean("CustomerManager");
        Customer customer = customerManager.findCustomerBySsn(id);
        if (customer == null) {
            throw new Exception("Customer " + id + " was not found.");
        }
        return "login";
    } catch (Exception e) {
```

```
        getFacesContext().addMessage("id", new
FacesMessage(e.getMessage()));
        return null;
    }
}
```

3. If you get errors for unknown imports, right-click the new text and select **Source** → **Organize Imports**. The following import statements are automatically added for you:
 - `import itso.bank.entities.Customer;`
 - `import itso.bank.entities.controller.CustomerManager;`
 - `import javax.faces.application.FacesMessage;`
4. Save the `login.xhtml` and `Login.java` files.
5. Open **login.xhtml**. Click the button.
6. Go to the **Properties** view. Click **Select or code an action button** (next to Action or outcome) and then choose **Select an action**.
7. In the Faces Action selection dialog window, click **Page Code** → **doLoginAction** and click **OK**.
8. Save `login.xhtml`.

Adding navigation

We add navigation so that the customer's details can be displayed if the customer ID is valid:

1. Open **login.xhtml**. Click the button.
2. Go to the **Properties** view. Click **Add Rule**.
3. In the **Add Navigation Rule** dialog window, select **customerDetails.xhtml** for the page.
4. Select **The outcome is** and then type `login` (Figure 19-21 on page 1088).

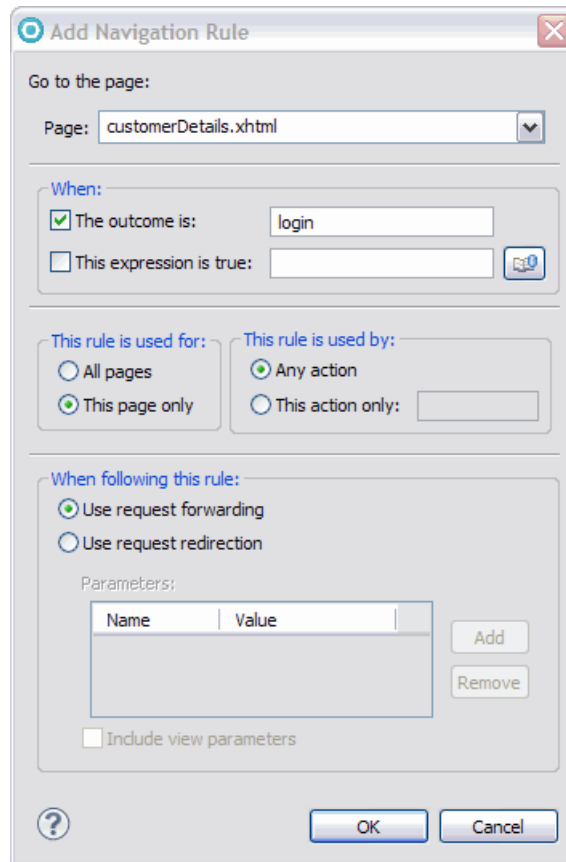


Figure 19-21 Login page

5. Click **OK**.

The login.xhtml page is now complete.

19.2.8 Editing the customer details page

For our customerDetails page, we display the information about a particular customer and show the associated bank account balances.

Displaying customer information

Follow these steps:

1. Open **customerDetails.xhtml**. Go to the **Page Data** view.

2. Expand **JPA** → **JPA Page Data**. Click **customer**. Drag **customer** to the page, and drop it on top of the Drop Controls here text.
3. In the Add JPA data to page wizard, make the following changes (Figure 19-22):
 - a. Clear **customer.accounts**.
 - b. Move **customer.title** up so that it is over `customer.firstName`.
 - c. Change the label of `customer.ssn` to “Customer ID:”.
 - d. Change the label of `customer.firstname` to “First Name:”.
 - e. Change the label of `customer.lastname` to “Last Name:”.
 - f. Change the control type of `customer.ssn` to **Display Text**.

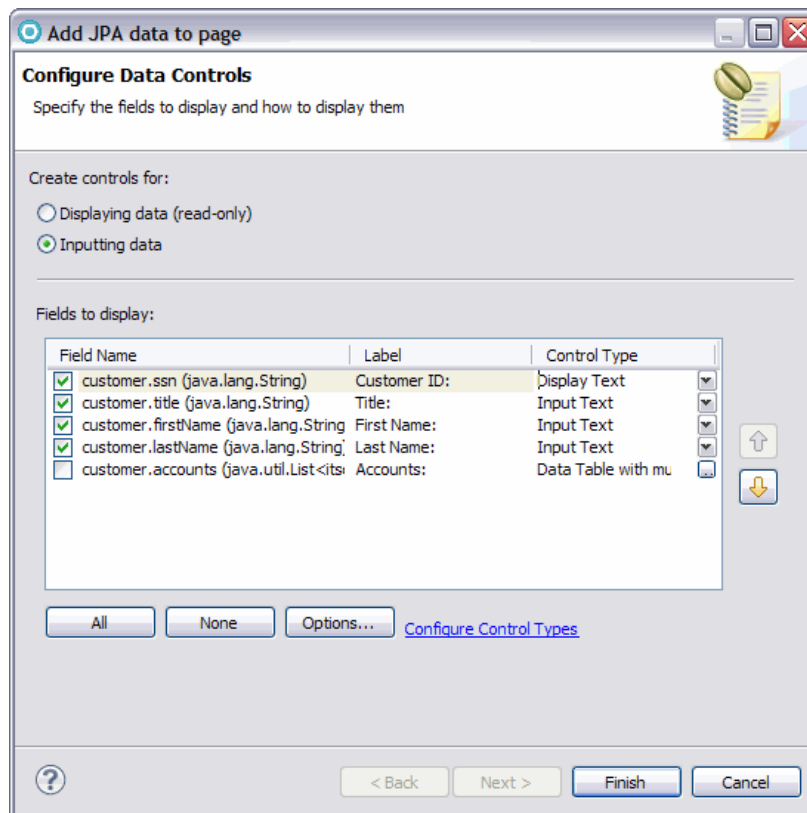


Figure 19-22 Add JPA data to page wizard for customer

4. Click **Options**.
5. On the Buttons tab, change the label to Update.

6. Click **OK** and click **Finish**.

Displaying account information

Follow these steps:

1. In the Page Data view, expand **JPA** → **JPA Page Data** → **accountList** → **accountList**. Click the inner **accountList** and drag it to the page. Drop it after Error Messages.
2. In the Add JPA data to page wizard, make the following changes (Figure 19-23):
 - a. Clear **customers** and **transacts**.
 - b. Change the id label to Account Number.

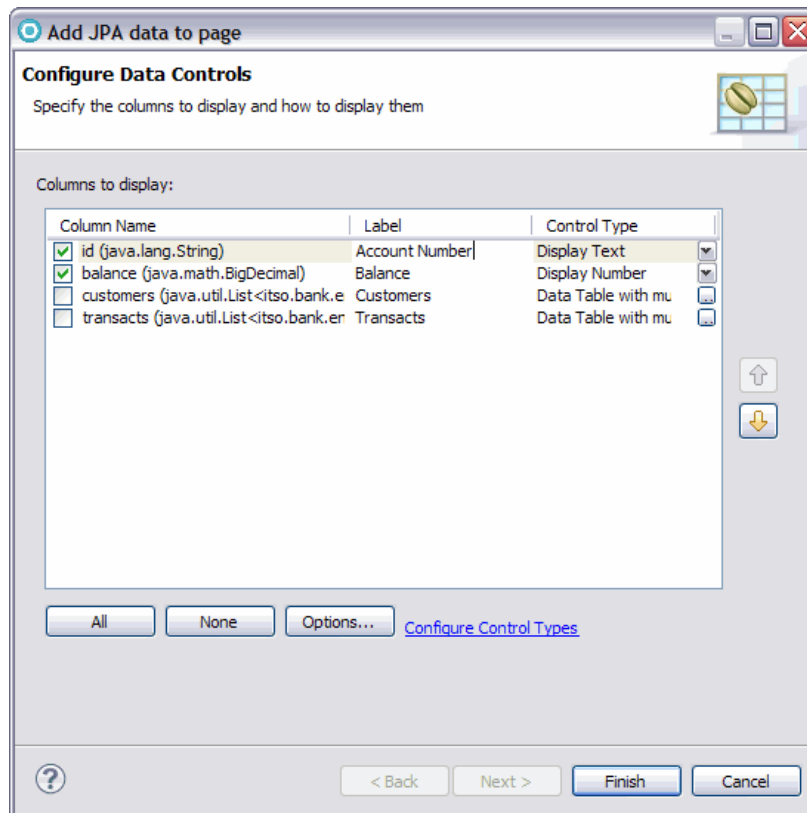


Figure 19-23 Add JPA data to page wizard for accounts

3. Click **Finish**.

The customerDetails page contains customer and account information (Figure 19-24).

The screenshot shows a web page for ITSO RedBank. At the top left is the bank's logo. Below it, there is a form for customer details with fields for Customer ID (ssn), Title, First Name, and Last Name. Below the form is an 'Update' button and an error message placeholder. Underneath is a table with two columns: 'Account Number' and 'Balance'. The table contains one row with values '{id}' and '{balance}'. At the bottom of the page, it says 'Created by ITSO, 2010'.

Figure 19-24 customerDetails page with customer and account information

4. Save customerDetails.xhtml.
5. In the Enterprise Explorer view, open **RAD8JSFWeb** → **Java Resources** → **src** → **pagecode** → **CustomerDetails.java**.
6. Find the method **getAccountList()**.
7. Replace the line `Object ssn =` with `Object ssn = getSessionScope().get("customerId");`

This code is available in `C:/7835code/jsf/getAccountList.txt`. See Figure 19-25.

```
@JPA(targetEntityManager = itso.bank.entities.controller.AccountManager.class, targetNamedQuery = "getAccountB
@JPAFilter(name = "ssn", value = "#{sessionScope.customerId}")
public List<Account> getAccountList() {
    if (accountList == null) {
        AccountManager accountManager = (AccountManager) getManagedBean("AccountManager");
        Object ssn = getSessionScope().get("customerId");
        accountList = accountManager.getAccountBySSN(ssn);
    }
    return accountList;
}
```

Figure 19-25 New getAccountList() method

8. Save `CustomerDetails.java`.

Updating customer information

Our page has an Update button. We ensure that it can update simple customer information:

1. In the Enterprise Explorer view, open **RAD8JSFWeb** → **Java Resources** → **src** → **pagecode** → **CustomerDetails.java**.
2. Paste in the code that is shown in Example 19-2, which is available in `C:/7835code/jsf/UpdateButton.txt`.

Example 19-2 updateButton

```
public String doUpdateAction() {
    CustomerManager customerManager =
    (CustomerManager)getManagedBean("CustomerManager");
    try {
        customerManager.updateCustomer(customer);
    } catch (Exception e) {
        logException(e);
    }
    return "update";
}
```

3. Save `CustomerDetails.java`.
4. Open **customerDetails.xhtml**. Click **Update** and go to the **Properties** view.
5. Click **Select or Code an action** and then choose **Select an action**.
6. In the Faces Action selection dialog window, click **Page Code** → **doUpdateAction** and click **OK**.
7. In the Properties view, click **Add Rule**.
8. In the Add Navigation Rule dialog window, select **CustomerDetails.xhtml** for the Page. Select **The outcome is** and type `update`.
9. Click **OK**.
10. Save the page.

19.2.9 Using Ajax

We already have an Update button on our `customerDetails.xhtml` page to update customer information. We decide to add a second update button that uses Ajax.

Follow these steps:

1. Open **customerDetails.xhtml**. Drag a **Button – Command** from the Standard Faces Components drawer of the palette and drop it next to the existing button.
2. In the source, change the value of the new button to Ajax Update, which changes the button's label.
3. Go to the **Properties** view. The **h:commandButton** tab is selected.
4. Go to the **Ajax** tab.
5. Click **Support Ajax**.
6. We want to allow the user to update the customer's first name, last name, and title. Type `firstName1 lastName1 title1` into the Execute combination box.
7. Select **form1** for Render.
8. Select **Click to create/edit custom code**.
9. The Quick Edit view automatically opens. Ensure that **listener** is selected.
10. Paste in the code that is shown in Example 19-3, which is also available at `C:/7835code/jsf/ajaxButton.txt`.

Example 19-3 ajaxButton

```
CustomerManager customerManager =
(CustomerManager)getManagedBean("CustomerManager");
try {
    customerManager.updateCustomer(customer);
} catch (Exception e) {
    logException(e);
}
```

The code is shown in Figure 19-26 on page 1094.

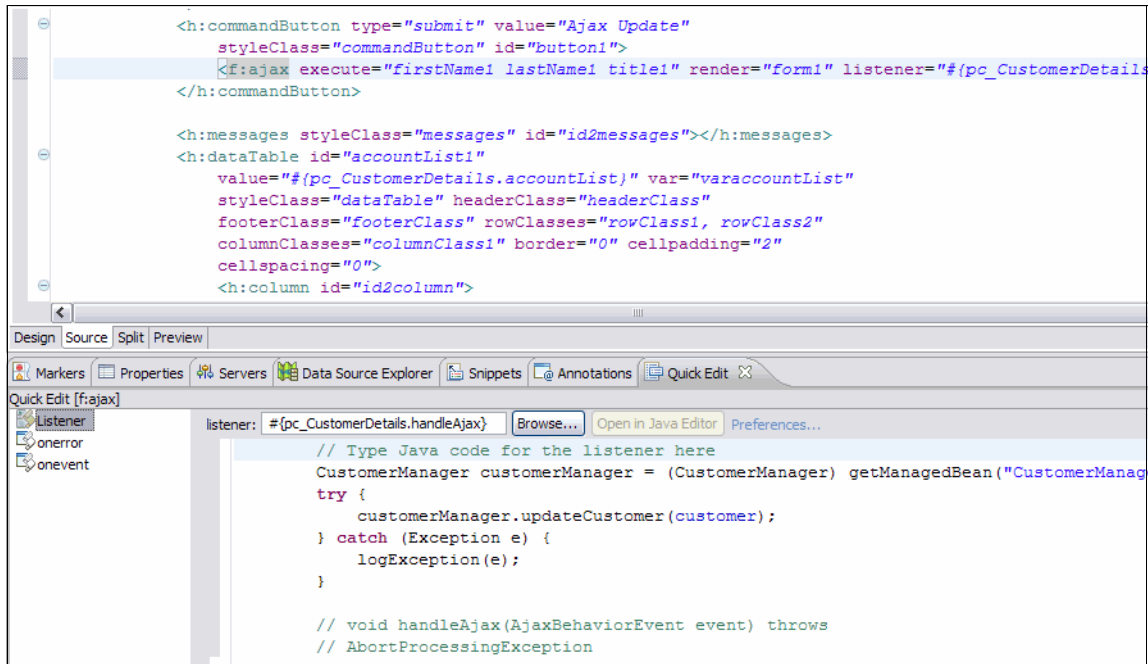


Figure 19-26 Button with Ajax

11. Save the page.

19.2.10 Running the JSF application

Now we run our web application and see the results:

1. In the Enterprise Explorer view, right-click **login.xhtml** and select **Run As** → **Run on server**.
2. Select a server and make any other necessary selections.
3. When the login page appears, you can interact with your web application.
4. If you enter a user ID that is too short, such as 1234, or that does not have a customer associated with it, an error message appears. See Figure 19-27 on page 1095.



Figure 19-27 Login page with error

5. If you enter a valid user ID, such as 444-44-4444, the details for that customer are shown (Figure 19-28).

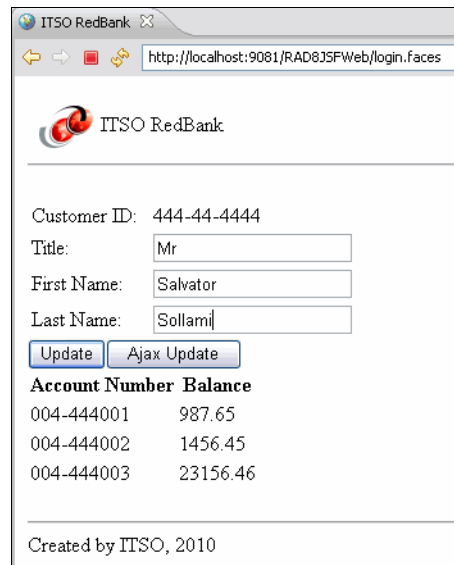


Figure 19-28 Customer details page

6. On the customer details page, you can make changes to the customer's first name, last name, and title. Then click either **Update** or **Ajax Update** to save those changes.

19.2.11 Final code

To run the web application, you must have completed the previous steps or have imported the sample from C:/7835codesolution/jsf/RAD8JSFWeb.zip.

You also must have set up the ITS0BANK database, as described 19.2.2, “Creating the JSF Project” on page 1065.

19.3 More information

For more information about JSF 2.0, see the following resources:

- ▶ *JSR 314: JavaServer Faces 2.0*
<http://www.jcp.org/en/jsr/detail?id=314>
- ▶ Rational Application Developer Information Center:
 - <http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.e.tools.jsf.doc/topics/tjsfover.html>
 - <http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.e.tools.jsf.doc/topics/tcrtfaceletcomposite.html>
- ▶ IBM developerWorks:
 - <http://www.ibm.com/developerworks/web/library/wa-aj-gmaps/>
 - <http://www.ibm.com/developerworks/java/library/j-jsf2fu2/index.html>
 - <http://www.ibm.com/developerworks/java/library/j-jsf2fu3/>
 - <http://www.ibm.com/developerworks/java/library/j-facelets2.html>
 - http://www.ibm.com/developerworks/wikis/download/attachments/140051369/radjsffacelet_template.swf?version=1
 - <http://www.ibm.com/developerworks/java/library/j-jsf2fu-0410/index.html>



Developing web applications using Web 2.0

The term *Web 2.0* implies a new online application that brings users into the creation of the application, which is usually referred to as the *social* aspect of Web 2.0 applications. Wikis and blogs are examples of users playing a big part in the overall value of the application.

Web 2.0 is also used to refer to the architecture and technologies that make these applications possible. In this chapter, we use the term Web 2.0 in this context.

Web 2.0 represents a shift in designing and developing web applications. In this chapter, we introduce the features, benefits, and architecture of Web 2.0, with a focus on demonstrating the Rational Application Developer support for Web 2.0.

The chapter is organized into the following sections:

- ▶ Introduction to Web 2.0 architecture and development practices
- ▶ Overview of Web 2.0 tooling features
- ▶ Developing the Web 2.0 sample application

The sample code for this chapter is in the 7835code\web20 folder.

20.1 Introduction to Web 2.0 architecture and development practices

In this section, we introduce the Web 2.0 architecture, technologies, and practices.

20.1.1 Web 2.0 architecture

The browser and Representational State Transfer (REST) services are the major components in Web 2.0 architecture that differentiate it from previous designs. HTML5 is gaining unprecedented adoption from all browser vendors, mainly because of the Web 2.0 trend. REST services are widely deployed on the web for a variety of purposes ranging from performing searches, viewing maps, looking up dictionaries, managing emails, and making secured commercial transactions. The interactions between the browser and REST services enable the highly interactive user experience that is the hallmark of most successful Web 2.0 implementations.

In Web 2.0 architecture, the browser takes over many responsibilities that used to be placed on the back-end programs that run on the server. The best example of this shift is how model view controller (MVC) is implemented. In the past, all components in MVC were implemented with server-based technologies using model beans, JavaServer Pages (JSP), and servlets. Now all components can be implemented inside the browser. The server's major responsibility is to provide services to the application running inside the browser.

Figure 20-1 on page 1099 shows the server-centered web application architecture.

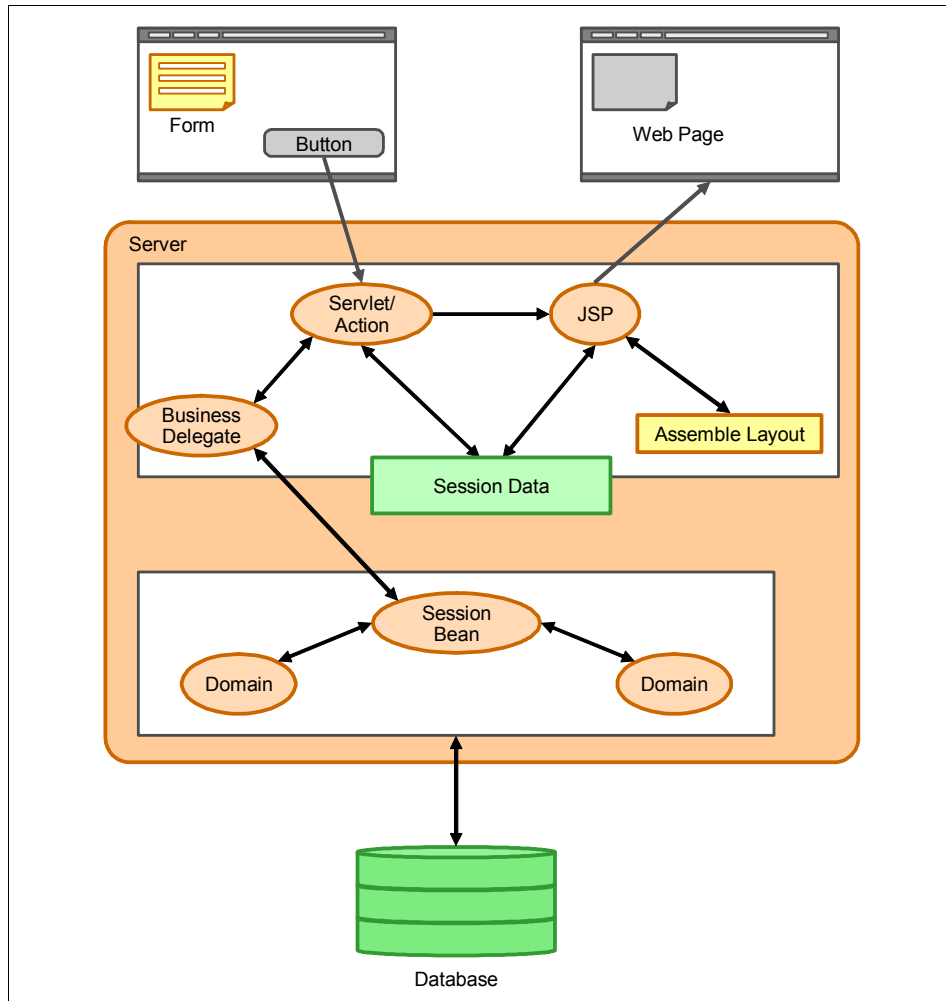


Figure 20-1 Server-centered web applications

Figure 20-2 on page 1100 shows the browser-centered web application architecture with REST services.

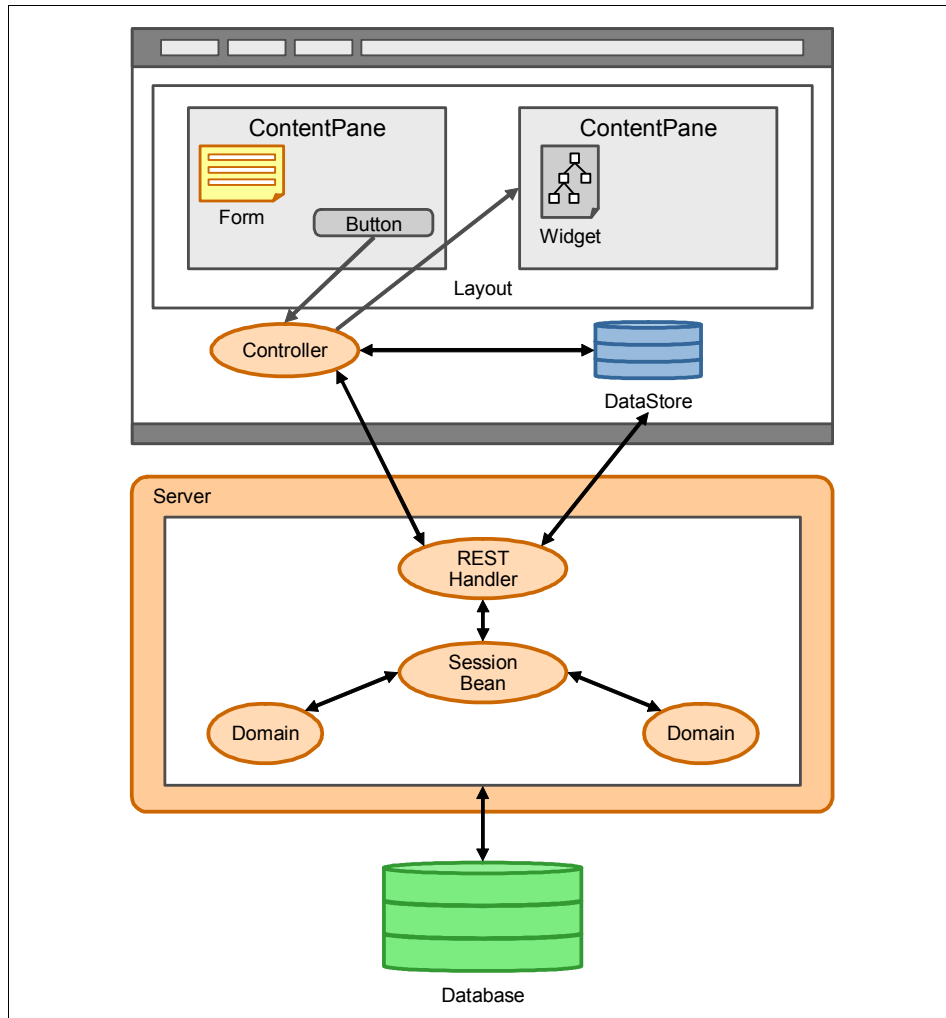


Figure 20-2 Browser-centered web applications with REST services

20.1.2 Technologies used in Web 2.0 applications

As discussed in 2.2.5, “Web 2.0 development” on page 41, the primary technologies of choice to implement Web 2.0 applications are Ajax, REST services, and JavaScript Object Notation (JSON). In addition, *syndicated feeds* in RSS or Atom formats also can get data into the browser. Certain applications require the user interface to be refreshed based on events that originated on the server side, for instance, a stock ticker widget. The *Bayeux protocol*, which is often referred to as *pub/sub* or *publish/subscribe*, is designed to address these

interaction patterns. In order for the Ajax code to invoke services in other domains, a *server-side proxy* is needed to broker the interactions between the application domain and the service domain. Without the proxy, the Ajax code is unable to access the services due to the browsers' same-origin policy for script execution.

In the following sections, we discuss the content of the Web 2.0 Feature Pack for WebSphere Application Server that delivers all these technologies in a single package.

Dojo: IBM Ajax toolkit of choice

Dojo is an open source project to which IBM has contributed significantly. Dojo has all the necessary features to build web applications with enterprise-strength UI and scalability. The powerful UI building blocks that are called *widgets*, which are fully accessible, are Dojo's greatest advantage over other Ajax toolkits. Another feature that makes Dojo stand out is the object-oriented heritage found everywhere in the system, such as its *class* sub-system that supports inheritance. Inheritance makes developing with Dojo a much smoother process than developing with raw JavaScript or many other Ajax toolkits.

Dojo Toolkit is divided into *modules*, which are used to group classes of similar functions. The IBM Dojo Toolkit includes the following modules:

dojo	A single tiny library, which contains XMLHttpRequest support, Document Object Model (DOM) manipulation, event handling, effects, Cascading Style Sheet (CSS) queries, globalization utilities, and a lot more. For certain applications, simply using the Dojo base is sufficient.
dijit	The widget system, which contains a rich set of pre-packaged widgets. It offers a convenient and flexible way to build custom widgets.
dojox	A collection of extensions. Each extension is independent of the other extensions. Certain extensions are mature and stable, such as the widgets in the <i>grid</i> folder. Other extensions are experimental.
ibm_soap	An IBM extension that can invoke web services via SOAP. This extension makes it possible to consume web services completely with JavaScript, without having to write any server-side Java code.
ibm_atom	An IBM extension that supports Atom format for feed syndication, as well as AtomPub for consuming REST services.

- ibm_opensearch** An IBM extension that includes a data store that can be initialized with a URL to an open search description document. The store can then be used to perform searches against the target server.
- ibm_gauge** An IBM extension that includes analog gauge widgets.

IBM JAX-RS: Building standard-based REST services

Java application programming interface (API) for RESTful Web Services (JAX-RS) is part of Java platform, Enterprise Edition (JEE) 6. JAX-RS specifies a collection of interfaces and annotations that simplifies the development of RESTful services. The following key features are provided by JAX-RS:

- ▶ Collection of annotations for declaring resource classes and the data types that they support
- ▶ Set of interfaces that allow application developers to gain access to the runtime context
- ▶ Extensible framework for integrating custom content handlers

The IBM implementation is based on Apache Wink, which is an open source project that was developed at the Apache Software Foundation to implement JAX-RS.

IBM JAX-RS includes the following features:

- ▶ JAX-RS server run time
- ▶ Stand-alone client API with the option to use Apache HttpClient 4.0 as the underlying client
- ▶ Built-in entity provider support for JSON4J
- ▶ Atom Java Architecture for XML Binding (JAXB) model in addition to Apache Abdera support
- ▶ Multipart content support
- ▶ Handler system to integrate user handlers into the processing of requests and responses

JSON4J: Processing JSON with Java

JSON4J is a Java library for processing JSON. Think of this library as the equivalent of JAXB for processing XML and Java. With JSON4J, you can produce JSON objects or arrays from Java objects easily, or you can parse JSON strings into Java objects.

JSON4J is developed by IBM independently.

Ajax Proxy: Proxy solution implemented with Java

The Ajax Proxy that is included in the Web 2.0 feature pack is a reverse proxy. Install it near the server that hosts the Ajax client, where outgoing connections are forwarded through the proxy to the requested server. From the Ajax client's point of view, the requests target services on the same domain, even though the reverse proxy might forward requests to servers on other domains.

IBM Web Messaging: Connecting the Ajax client to server-side events

The *web messaging service* is an implementation of the pub/sub pattern that pushes server-side events from the WebSphere Application Server service integration bus (SIB) to the Ajax client in the browser. The implementation is based on the Bayeux protocol. Any Ajax toolkit that supports the Bayeux protocol, such as the Dojo Toolkit, can communicate with the IBM web messaging service.

Figure 20-3 shows the overall architecture of the web messaging service.

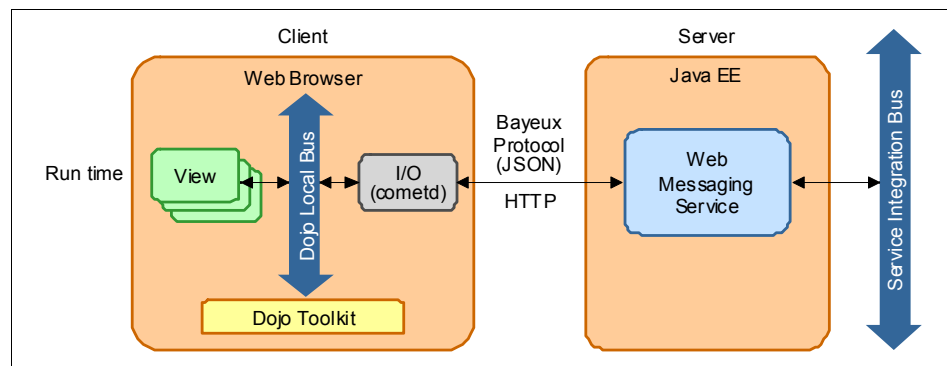


Figure 20-3 Overall architecture of IBM web messaging service

In this architecture, because the Ajax clients connect to the server via the SIB, multiple ways exist for server-side components to publish messages to the Ajax. These options include standard Enterprise JavaBeans (EJB) publishing to a topic, a Java Message Service (JMS) client publishing to a topic, a web service, or the Ajax client publishing to other Ajax clients.

20.2 Overview of Web 2.0 tooling features

Many technologies are available for building Web 2.0 applications. Rational Application Developer provides a rich set of tools that help in the development process.

20.2.1 JavaScript editing

JavaScript is at the center of the Ajax programming model. It is a powerful language known for its dynamicity and flexibility. However, it also presents a challenge to developers who are used to programming with a strong-typed language, such as Java or C#. The following features in Rational Application Developer make developing in JavaScript a much smoother experience:

- ▶ Syntax highlighting makes JavaScript code easy to read in the editor
- ▶ Code assist brings APIs to the programmer's fingertip, eliminating the need to flip through documentation
- ▶ Code formatting enhances code readability
- ▶ Syntax validation catches coding mistakes
- ▶ Code outline enables fast browsing
- ▶ JSON editing with dedicated editor for .json files:
 - Syntax highlighting
 - Syntax validation
 - Code formatting
 - Bracket matching
 - Code compression removes white spaces and puts the entire content in a single line to optimize runtime download

20.2.2 Dojo development

The object-oriented style of Dojo's type system provides opportunities to further streamline the programming experience. Furthermore, the widget system enables the visual construction of Dojo web pages. The following features in Rational Application Developer take advantage of Dojo's architectural characteristics to provide a cohesive development environment:

- ▶ Project setup to access the Dojo toolkit locally or remotely (using Content Delivery Networks, from another project, or through a URI)
- ▶ Class creation wizard to promote object-oriented design principles

- ▶ Custom widget creation wizard to develop reusable extensions
- ▶ Palette showing Dojo widgets for drag-and-drop to assemble web UI
- ▶ Properties view for configuring Dojo widgets
- ▶ Visual editor with preview for “what you see is what you get” (WYSIWYG) experience
- ▶ Custom build wizard to author Dojo layers to optimize performance
- ▶ Dojo Firebug extension to gain valuable insight into runtime characteristics

20.2.3 Testing and debugging

Speed and accuracy are the main concerns during unit testing and debugging. Rational Application Developer provides the following features to meet those requirements:

- ▶ Ajax Test Server:
 - Lightweight, fast, and ideal for unit testing Ajax applications
 - Automatically installed into the Servers view
 - Built-in Ajax Proxy for invoking services across network domains
- ▶ Firebug:
 - Premier JavaScript debugging tools that are widely used by the development community
 - Debugging tools ship as ready to use and are supported by IBM
 - Tools automatically installed when you launch the browser type *Firefox with Firebug*
 - Firebug integrated with the Debug perspective for breakpoints, variables, and call stack

20.2.4 JAX-RS services development

JAX-RS is a standard specification for building REST services. Rational Application Developer allows any JAX-RS implementation to be used and, in particular, provides the IBM JAX-RS library ready to use as the officially supported option:

- ▶ Project setup to easily configure dynamic web projects to use the IBM JAX-RS library
- ▶ Validation and quick fix to configure the JAX-RS servlet
- ▶ Annotation support for source code editing

- ▶ Deployment support for WebSphere Application Server to ensure that the library is available at run time

20.2.5 Using other server-side technologies

The following features assist in developing server-side code that uses other libraries that are included in the Web 2.0 feature pack:

- ▶ Project set up to use the following libraries:
 - JSON4J
 - Abdera for RSS and Atom feeds support
 - IBM Web Messaging
 - Remote Procedure Call (RPC) Adapter
- ▶ Deployment support on WebSphere Application Server to ensure that the libraries are available at run time
- ▶ RPC Adapter wizard to create HTTP RPC services based on Plain Old Java Objects (POJOs) and EJBs

20.3 Developing the Web 2.0 sample application

In this section, you develop a Web 2.0 application using the Dojo and Ajax tooling that is provided in Rational Application Developer. The application displays account balances and transactions for a bank customer.

20.3.1 Setting up the project

Begin by importing an initial version of the project into your workspace. The project contains a servlet to simulate a back-end service that provides data for your web application. The focus of this tutorial is on building the front-end web page to display the data:

1. Import the project archive file from `C:\7835code\web20\RAD8DojoInitial.zip`.
2. Select **File** → **Import**.
3. In the tree, select **General** → **Existing Projects into Workspace** and click **Next**.
4. Click **Select archive file** and then browse to the compressed file.
5. Select the project named **RAD8Dojo** and click **Finish**.

6. Select **Window** → **Open Perspective** → **Other** → **Web** to open the Web Perspective.
7. Configure the project for Dojo development:
 - a. Right-click the project and select **Properties**.
 - b. Click the **Project Facets** node in the tree on the left.
 - c. In the list of facets shown, expand the **Web 2.0** node and check the box next to **Dojo Toolkit**.

By enabling the Dojo Toolkit facet, your web project is configured to develop Dojo web applications. The Dojo Toolkit that is included in Rational Application Developer includes additional IBM extensions to the base Dojo Toolkit, including libraries for ATOM (ATOM Syndication Format) data access, analog and bar gauges, and simplified access for SOAP web services.

- d. Click the **Further configuration available** link at the bottom of the dialog window (Figure 20-4).

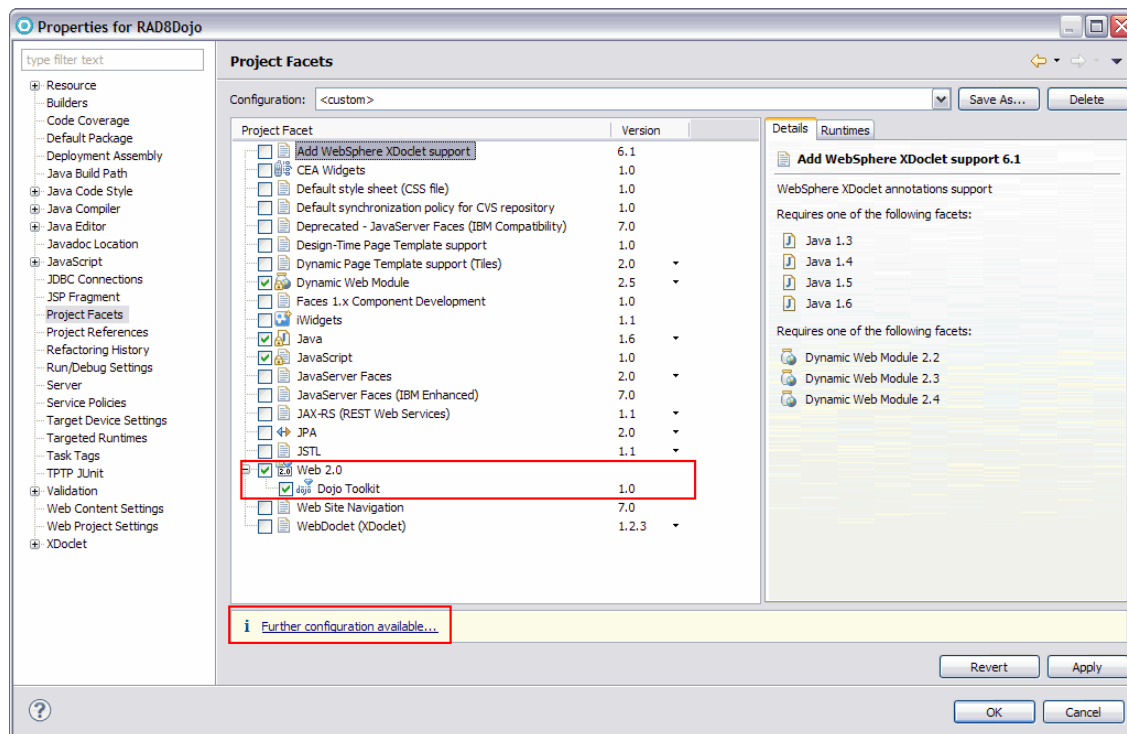


Figure 20-4 The Project Facets page in the Project Properties dialog window

8. The Dojo Project Setup dialog window that opens provides a summary of how Dojo will be incorporated into your project (Figure 20-5). By default, the latest level of Dojo that is supported by IBM is copied into your web project.

If you want more information about the Dojo project setup options, continue with the following optional steps. Otherwise, you can click **OK** on both open dialog windows and move ahead to 20.3.2, “Creating the web page” on page 1111 of the tutorial.

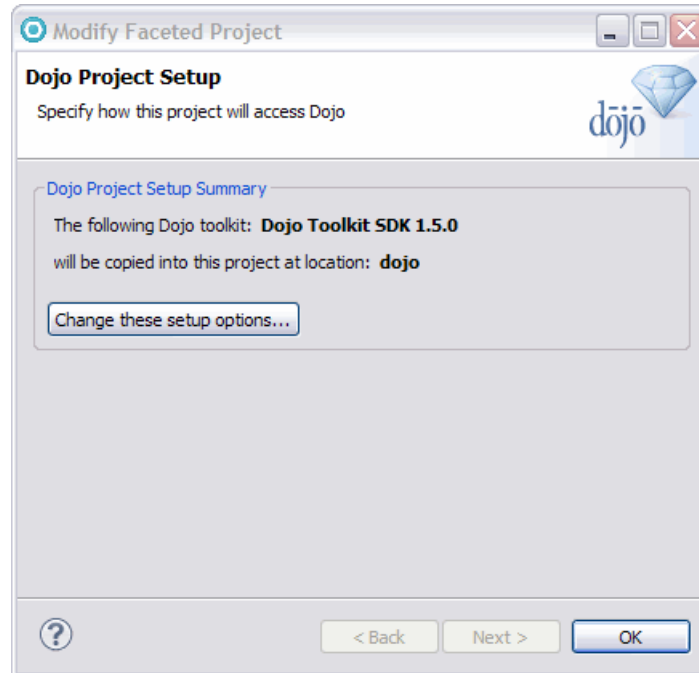


Figure 20-5 The Dojo Project Setup dialog

9. Optional: To modify the Dojo setup, click **Change these setup options**, and follow these steps:
 - a. The Dojo Project Setup Options dialog box provides you with three options for configuring Dojo in your web application. Select the third option, **Dojo is remotely deployed or is on a public CDN** (Figure 20-6 on page 1109). Click **Next**.

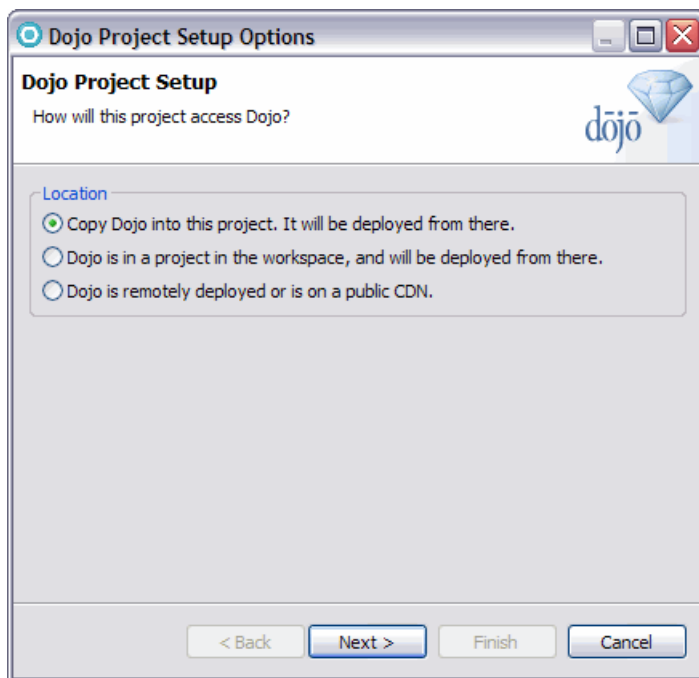


Figure 20-6 This dialog is used to modify the Dojo Project Setup options

You use this third option, Dojo is remotely deployed or is on a public CDN, if your application uses a remotely hosted public content delivery network (CDN) or an existing copy of Dojo that is already deployed on your network. CDNs provide geographically distributed hosting for open source JavaScript libraries. When a browser resolves the URL in your web application, the browser automatically downloads the file from the closest available server. You need to provide the URL or URI to the appropriate location. If Dojo is not contained in your project, the Dojo tools must reference a corresponding copy of Dojo to provide content assist and validation. The wizard gives you the option of selecting a default version or selecting your own Dojo from disk. This option does not copy Dojo into your project or workspace. Click **Back**.

- b. Select the second option, **Dojo is in a project in the workspace, and will be deployed from there** and click **Next**. On this page, you can browse to the root Dojo folder in another project in your workspace. The copy of Dojo is not copied into your project. It is deployed from the project where it is currently located. Click **Back**.
- c. Select the first option, **Copy Dojo into this project. It will be deployed from there** and click **Next**. On this page, you can specify the location in your project where Dojo will be copied. At the bottom of the page, you can

select one of the default versions of Dojo that ships with Rational Application Developer or browse for a copy on your disk. Leave the default values and click **Finish**.

10. Click **OK** on the Dojo Project Setup dialog window.
11. Click **OK** on the Project Properties dialog window. A progress monitor displays while your project is configured to work with Dojo. After your project is configured to work with Dojo, you see a sub-folder named `dojo` under the `WebContent` folder (Figure 20-7 on page 1111).

Types: You can visualize the types and methods that are available in your JavaScript libraries by expanding the JavaScript Resources node in the Enterprise Explorer. Further expand the Dojo Toolkit node, and see your types broken down by namespace (Figure 20-7).

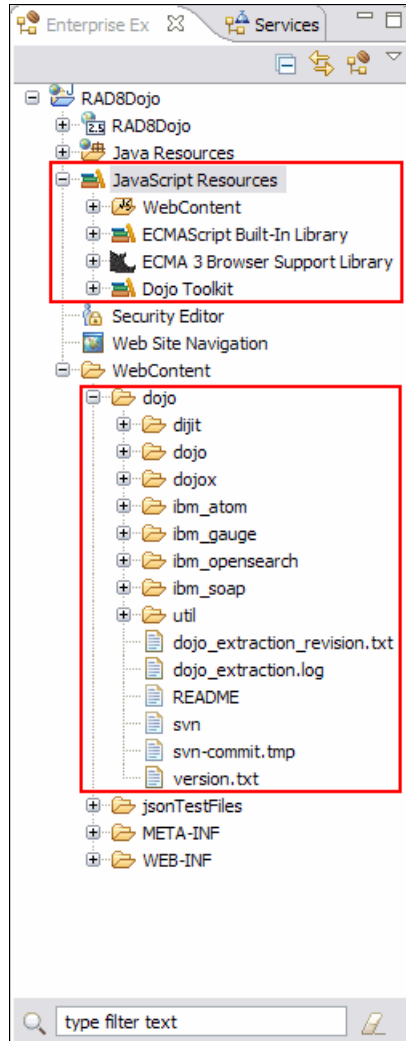


Figure 20-7 Enterprise Explorer view: JavaScript Resources and imported Dojo files

20.3.2 Creating the web page

In this section, you create the web page for your application. You use the Dojo Widget Palette, Page Designer, and Property views to add and configure a Dojo Layout Widget.

Complete these tasks:

1. In the Enterprise Explorer, right-click the **WebContent** folder of your project and select **New** → **Web Page**.
2. In the New Web Page wizard, enter index as the File Name.
3. For the Template, select **HTML/XHTML** and click **Finish**. Your new web page appears in the Page Designer view.
4. At the bottom of the Page Designer view, click the **Design** tab. The Design view provides a visual representation of your web page.
5. In the rightmost column, show the palette by clicking the **Palette** tab. It might be hidden behind other tabs (Figure 20-8).

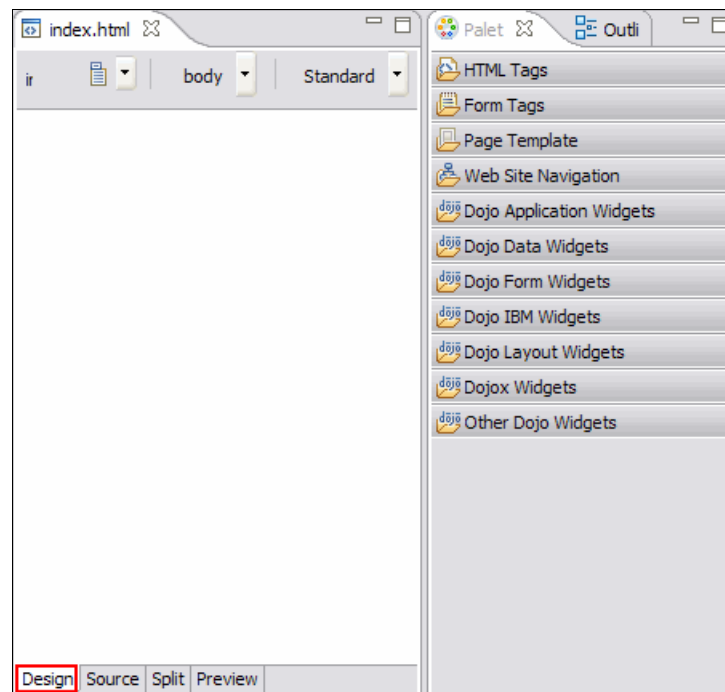


Figure 20-8 Page Designer opened to the Design view and the Palette surfaced

Dojo palette drawers: The palette drawers labeled Dojo are dynamically generated based on the Dojo files in your project. As new widgets are added to the project, they are added to the palette.

6. Expand the **Dojo Layout Widgets** palette drawer by clicking it. You see a list of available Dojo widgets, which can be dropped onto your page.
7. Click **BorderContainer**, and drag and drop it onto the open editor. Complete these steps:
 - a. A dialog window opens, allowing you to perform additional configuration tasks of the BorderContainer widget. Click the **Top**, **Bottom**, and **Center** check boxes (Figure 20-9).

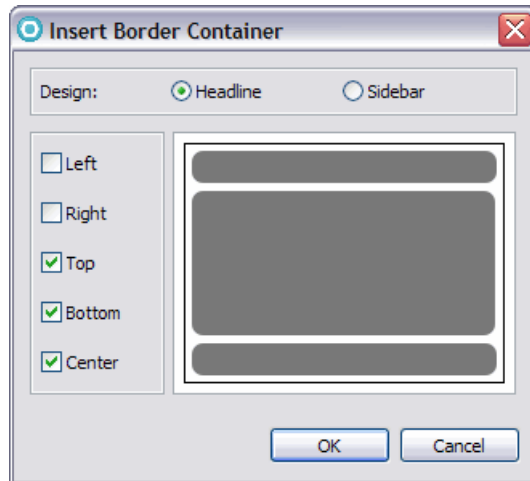


Figure 20-9 The Insert Border Container dialog box

- b. Click **OK**.
8. A visualization of the BorderContainer is now displayed in the Design view, and the appropriate markup has been added to the source. Click the BorderContainer visualization. Then click the **Properties** tab in the view beneath the editor. If not already selected, click the **BorderContainer** tab (Figure 20-10 on page 1114).

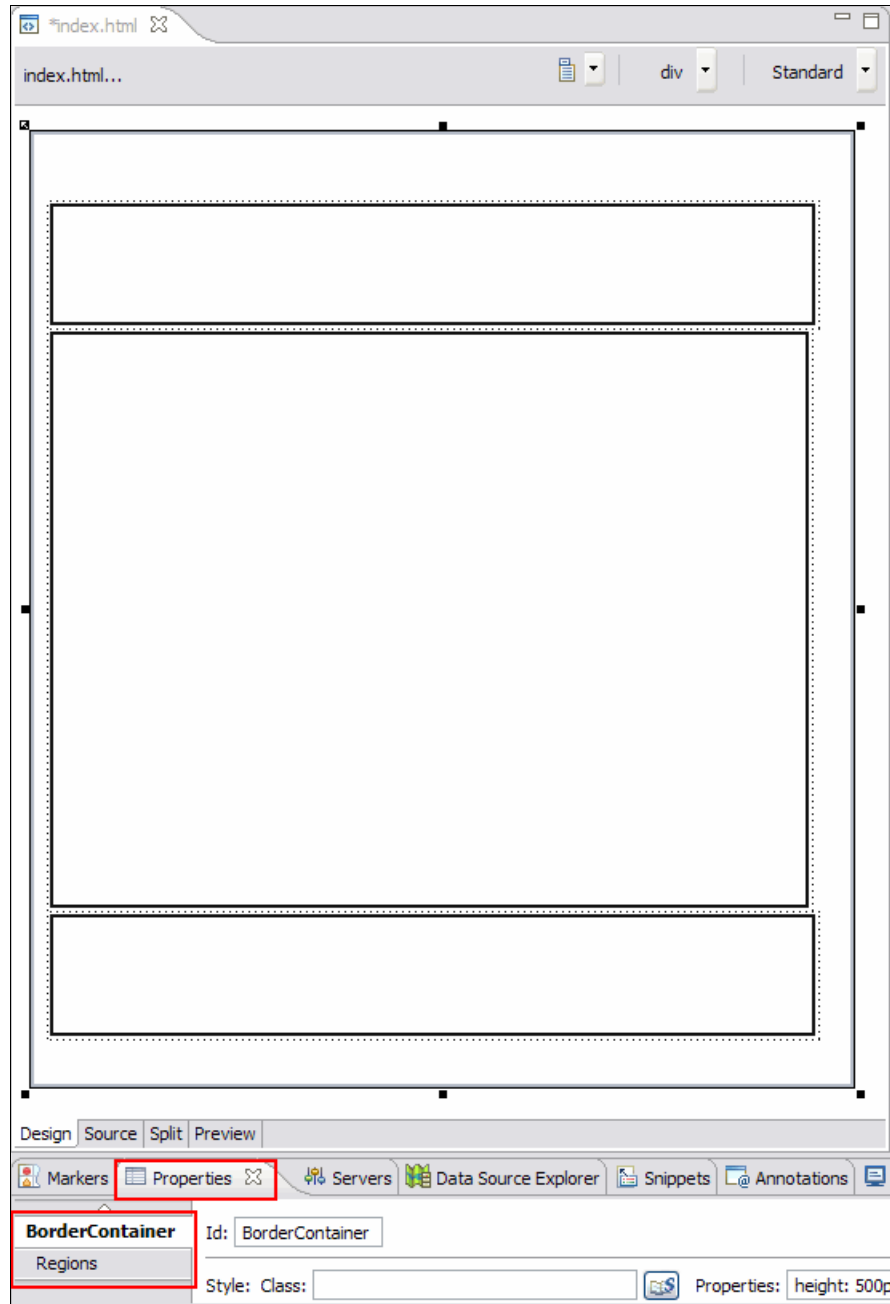


Figure 20-10 *BorderContainer visualization in Page Designer and Properties tab selected*

Follow these steps:

- a. Change the height of the BorderLayout in the Property tab from 500px to **100%**. Change the width to **640px**.
 - b. Click in each of the three BorderLayout regions in the Page Designer editor and type labels in each region: Logon, Accounts, and Transactions.
9. Switch to the **Source** view by clicking the tab at the bottom of the editor.
10. For each BorderLayout region, add the attribute `splitter="true"`. This attribute allows the user to change the size of the regions:
- ```
<div dojoType="dijit.layout.ContentPane" region="top"
splitter="true">
```
11. Switch to the **Preview** view, and verify that you see the BorderLayout (Figure 20-11).

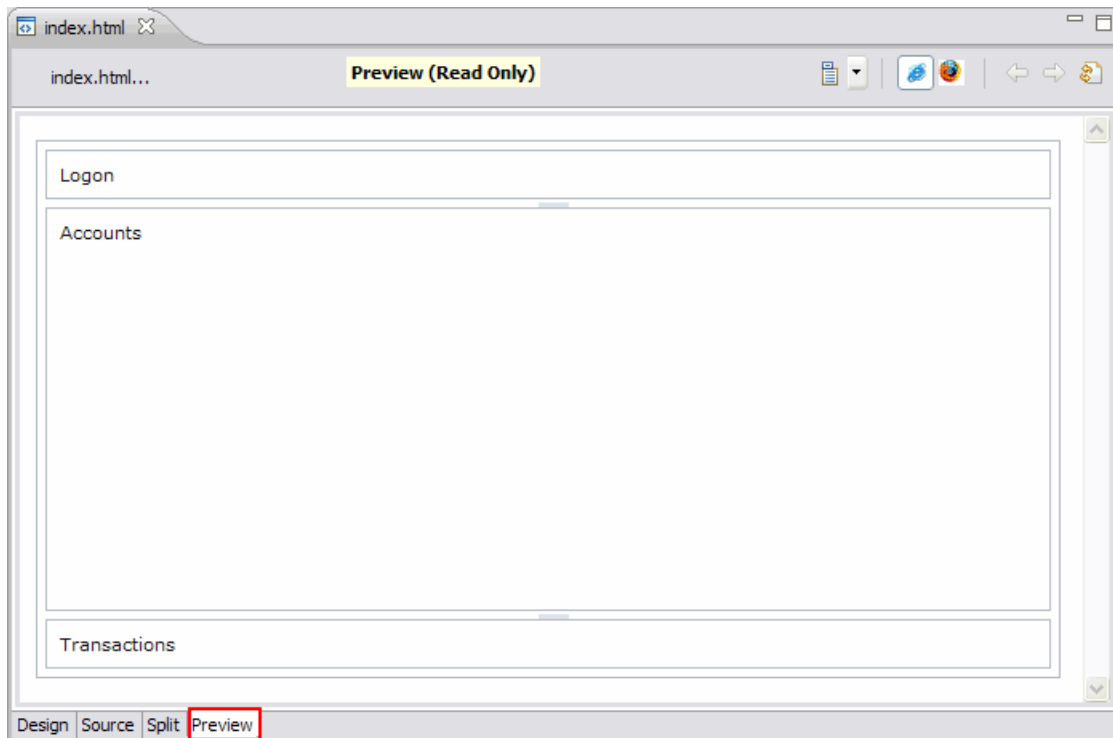


Figure 20-11 A preview of the BorderLayout

12. Save and close `index.html`.

### 20.3.3 Building a custom Dojo widget

Dojo ships with dozens of standard widgets, including input fields, combination boxes, radio buttons, and so forth. You can create custom widgets to encapsulate reusable UI elements or a specific piece of functionality. In this section, you create a custom Dojo widget using the New Dojo Widget wizard that is provided in Rational Application Developer.

The new widget allows a user to select a Social Security number and click a Submit button. We test the widget using the AJAX Test Server and a simple JSON file. Later, we replace the JSON file with a call to our back-end service. Follow these steps:

1. Right-click the **WebContent/dojo** folder and select **New** → **Dojo Widget**.
2. Enter bank as the Module Name. Enter CustomerLogon as the Widget Name.
3. Leave the defaults for the rest of the fields (Figure 20-12) and click **Finish**.

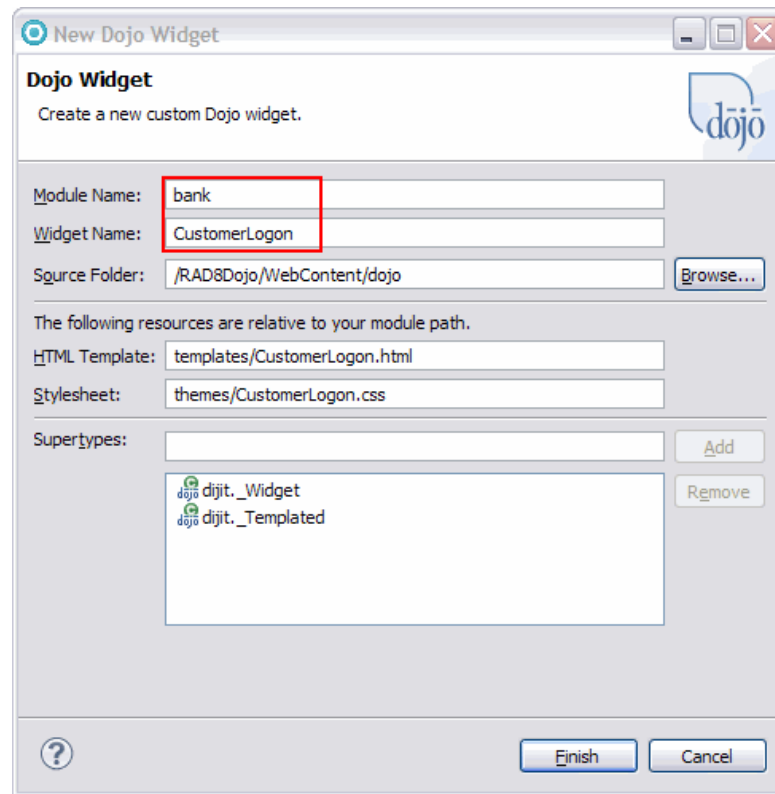


Figure 20-12 New Dojo Widget wizard with Module Name and Widget Name fields entered



Three files are created under a folder named `dojo/bank`:

- `templates/CustomerLogon.html` file that is the UI template for the widget
  - `themes/CustomerLogon.css` file, which provides the styling for the widget
  - `CustomerLogon.js` file, which provides the JavaScript back end and business logic portion of the widget
4. Double-click **CustomerLogon.js** if the file is not already open.
  5. Change the `widgetsInTemplate` field from `false` to **true**. This field indicates that our custom widget will contain other Dojo widgets as part of its UI.
  6. Directly beneath the `widgetsInTemplate` field, add a new field that will be used to hold the user selection, `currentSSN`. It needs to have a default value of `null`. Be sure to add a comma after the field. See Example 20-1.

*Example 20-1 The attribute code in the CustomerLogon.js file*

---

```
// Set this to true if your widget contains other widgets
widgetsInTemplate : true,
// the currently selected social security number
currentSSN: null,
```

---

7. Directly beneath the `postCreate` function, add a new function named `logon`. Leave the function empty for now. See Example 20-2

*Example 20-2 The function code in the CustomerLogon.js file*

---

```
postCreate : function() {
},
// the logon function will store the social security number
// selected by the user when they press the submit button.
logon: function() {
}
```

---

8. Double-click the **templates/CustomerLogon.html** file to open it in the editor.
9. At the bottom of the editor, click the **Source** tab to display the page in the Source view.
10. Between the existing **div** tags, type a label:  
Select a Customer SSN:
11. Add Dojo widgets for the input field and submit button:
  - a. In the rightmost column, display the **Palette** by clicking the appropriate tab.
  - b. Expand the **Dojo Form Widgets** drawer.

- c. Select the **FilteringSelect** widget, and drop it to the right of the label that you typed in the previous step, between the existing div tags. See Figure 20-13.
- d. Click **OK** on the Insert Filtering Select dialog box that appeared. We will populate our FilteringSelect widget later with a JSON file.
- e. In addition to palette drops, you can add widgets to a page by using content assist. Directly beneath the closing select tag just added, type <but and invoke content assist (press Ctrl+Spacebar). You see the button tag as a choice. Double-click it or press Enter to insert the tag on your page.
- f. Put your cursor inside the opening button tag, where the asterisk (\*) is located in the following example: <button \*></button>.
- g. Type `dojo` and invoke content assist. Select the **dojoType** attribute, and insert it into your page.

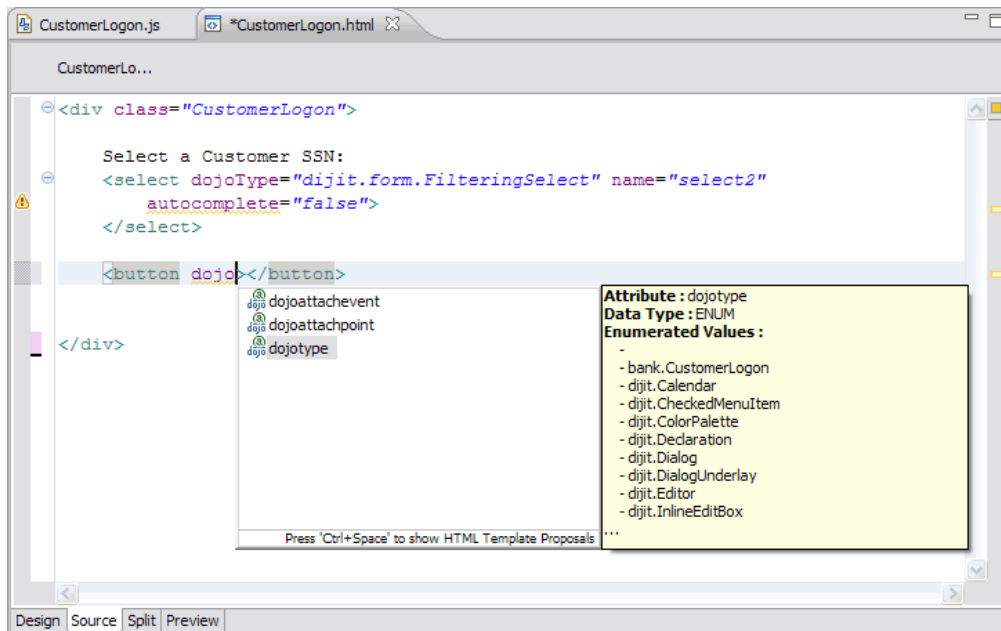


Figure 20-13 Using content assist to add a dojotype attribute to the button tag

- h. To set the value of the dojotype attribute, place your cursor inside the double quotation marks, and invoke content assist again. You see a list of available dojo widgets. Begin typing `dijit.form.B` until you see `Button`. Insert it onto your page.

- i. Add the following attributes to the FilteringSelect widget:
  - i. Put your cursor after the autoComplete attribute. Type invalid and invoke content assist. Insert the **invalidmessage** suggestion. Enter "You must select a valid Customer SSN" as its value.
  - ii. Add a **dojoattachpoint** attribute and set its value to ssnSelect.
- j. Add the following attributes to the Button widget:
  - i. label="Logon"
  - ii.dojoattachevent="onClick: logon"
- k. Example 20-3 shows the complete CustomerLogin.html code.

*Example 20-3 The finished CustomerLogon.html markup*

---

```
<div class="CustomerLogon">
 Select a Customer SSN:
 <select dojoType="dijit.form.FilteringSelect"
name="select2"autocomplete="false" invalidmessage="You must enter
a valid Customer SSN." dojoattachpoint="ssnSelect">
 </select>
 <button dojotype="dijit.form.Button" label="Logon"
dojoattachevent="onClick: logon"></button>
</div>
```

---

**Attributes:** dojoattachpoint and dojoattachevent are attributes that are specified by all Dojo widgets. The value that is specified for the dojoattachpoint is the name by which that widget instance can be referenced from the CustomerLogon.js file.

The dojoattachevent attribute adds event handling to widgets. It specifies the action to listen for and the function to execute when that action takes place. In this example, it executes the logon function in CustomerLogon.js when the button widget is clicked.

12. Save and close CustomerLogon.html.
13. Open the **CustomerLogon.js** file.
14. Add dojo.require statements for the two widgets that are used in the html file. The dojo.require statements load the necessary resources to create those widgets when the page is loaded. Complete these steps:
  - a. Beneath the existing require statements, type dojo.re and invoke content assist (press Ctrl+Spacebar).
  - b. Select **dojo.require(moduleName)** from the pop-up list.

- c. Type "dijit.form.FilteringSelect" as the function attribute.
  - d. Repeat steps b and c for "dijit.form.Button".
15. In the logon function, add the following line of JavaScript code:

```
this.currentSSN = this.ssnSelect.get("displayedValue");
```

This line of code uses the `dojoattachpoint` attribute that is specified in the `CustomerLogon.html` page to reference the `FilteringSelect` widget and return its selected value. The value is assigned to the widget attribute `currentSSN` that you defined in a previous step.

**Content assist:** Content assist is also available for widget attributes. For example, you can type `this.cur` and invoke content assist. You then see `this.currentSSN` in the list of available proposals.

16. Rational Application Developer V8 provides templates for commonly used Dojo functions. Inside the `postCreate` function, type `dojo.xhr`, and invoke content assist. You see a list of available methods and template proposals. Select the template proposal for **dojo.xhrGet** (Figure 20-14).

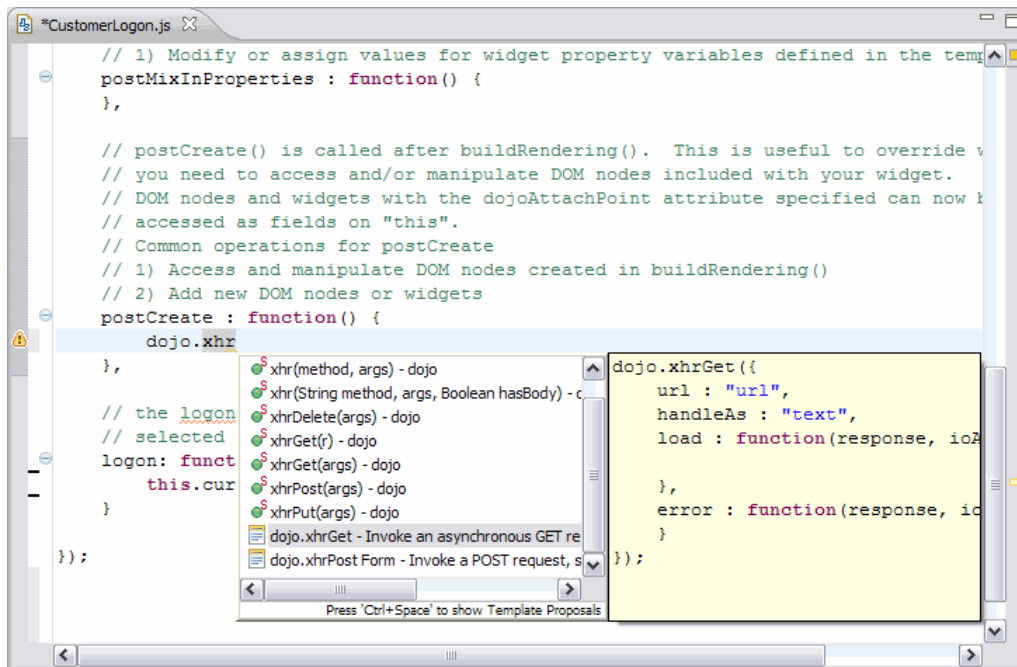


Figure 20-14 Selecting the `dojo.xhrGet` template

17. Skip the `url` field, for now, by pressing the `Tab` key.

18. In the content assist pop-up list for `handleAs`, select **json**.
19. In the error function, add the following code:

```
alert("Could not load Customer data.");
```
20. You need to wrap your load function in a call to `dojo.hitch`, which allows you to make calls to the `this` object from within the function. Copy and paste the code that is shown in Example 20-4 for the load function.

*Example 20-4 The load function code*

---

```
load : dojo.hitch(this, function(response, ioArgs) {
 // Un-comment the following line or re-type it using content assist
 // var ssnStore = new dojo.data.ItemFileReadStore({data: response});
 this.ssnSelect.searchAttr = "ssn";
 this.ssnSelect.set("store", ssnStore);
}),
```

---

21. Declared Dojo types are also available in content assist. Beneath the commented out line of code, type `var ssnStore = new dojo.` and then invoke content assist.
22. Filter down the list by continuing to type `dojo.data.Item`.
23. By now you see `dojo.data.ItemFileReadStore` in the list of proposals. Insert it into the function.
24. Add the constructor parameters: `{data: response}` as is shown in the commented line of code.
25. At the top of the file, beneath the existing `dojo.require` statements, add a new `dojo.require` statement for `"dojo.data.ItemFileReadStore"`.
26. Save the `CustomerLogon.js` file.

### 20.3.4 Adding to a page and testing a custom Dojo widget

In this section, you add your custom widget to a web page and test it on the AJAX test server. First, you test using a simple JSON file, which you modify using the JSON editor. Then you change to using a servlet to provide the back-end data. Complete these tasks:

1. Open `index.html` and switch to the **Design** view.
2. Surface the **Palette** and expand the **Other Dojo Widgets** drawer.
3. Drag and drop your widget, **CustomerLogon**, into the **Logon** section of the **BorderContainer**. Add it beneath the label (Figure 20-15 on page 1122).

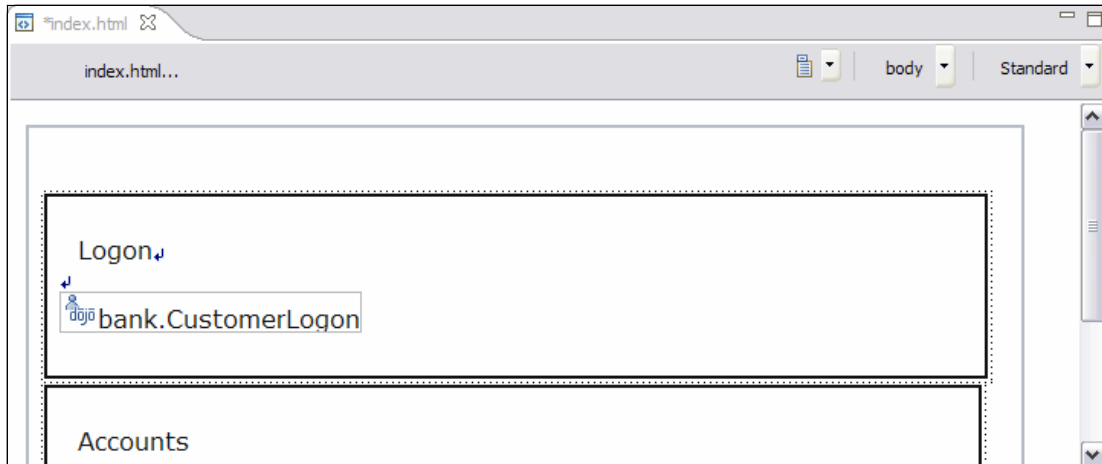


Figure 20-15 Adding the CustomerLogon widget to the BorderContainer

4. Test to make sure that your widget shows up correctly in a browser. Click the **Preview** tab at the bottom of the Page Designer editor (Figure 20-16). You can flip between Firefox and Internet Explorer previews by using the buttons at the top of the editor.

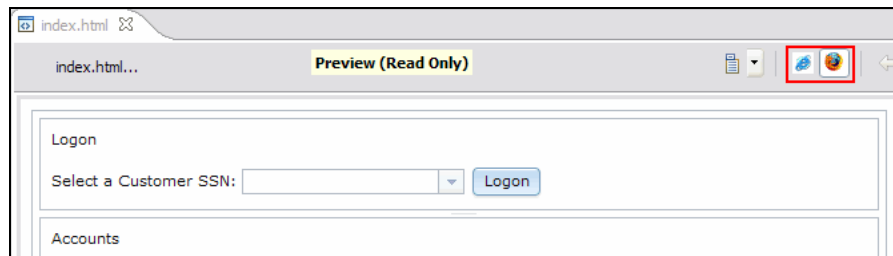
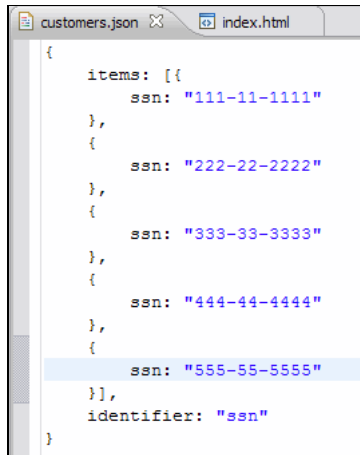


Figure 20-16 Preview of index.html

5. You receive a pop-up alert when loading the page in Preview mode, because you have not attached any data to the FilteringSelect widget. To add data to your widget, perform these steps:
  - a. Open the file **WebContent/jsonTestFiles/customers.json**. It opens in the JSON editor.
  - b. The file is initially compressed, with all of the white space removed. You can uncompress the file for editing by right-clicking in the editor and selecting **Source** → **Format**.
  - c. Add an additional entry to the JSON file with an ssn of 555-55-5555 (Figure 20-17 on page 1123).



```
{
 items: [
 {
 ssn: "111-11-1111"
 },
 {
 ssn: "222-22-2222"
 },
 {
 ssn: "333-33-3333"
 },
 {
 ssn: "444-44-4444"
 },
 {
 ssn: "555-55-5555"
 }
],
 identifier: "ssn"
}
```

Figure 20-17 The JSON editor displaying uncompressed contents

- d. Recompress the content of the file. Right-click and select **Source** → **Compress**.
- e. Save and close the JSON file.
6. Next you need to modify your widget to point to this JSON file. Open **CustomerLogon.js** and scroll down to the **postCreate** function.
7. Find the `url` attribute in the `dojo.xhrGet` function call and change its value to the path of the JSON file, which is `"jsonTestFiles/customers.json"`.
8. Now test your widget using the AJAX Test Server:
  - a. Right-click **index.html** in the Enterprise Explorer.
  - b. Select **Run As** → **Run on Server**. If you see **Ajax Test Server** listed in the list of available servers, select it and click **Finish**. (If you do not see it listed, click **Manually define a new server**. Select **AJAX Test Server** from the list and click **Finish** (Figure 20-18 on page 1124).)

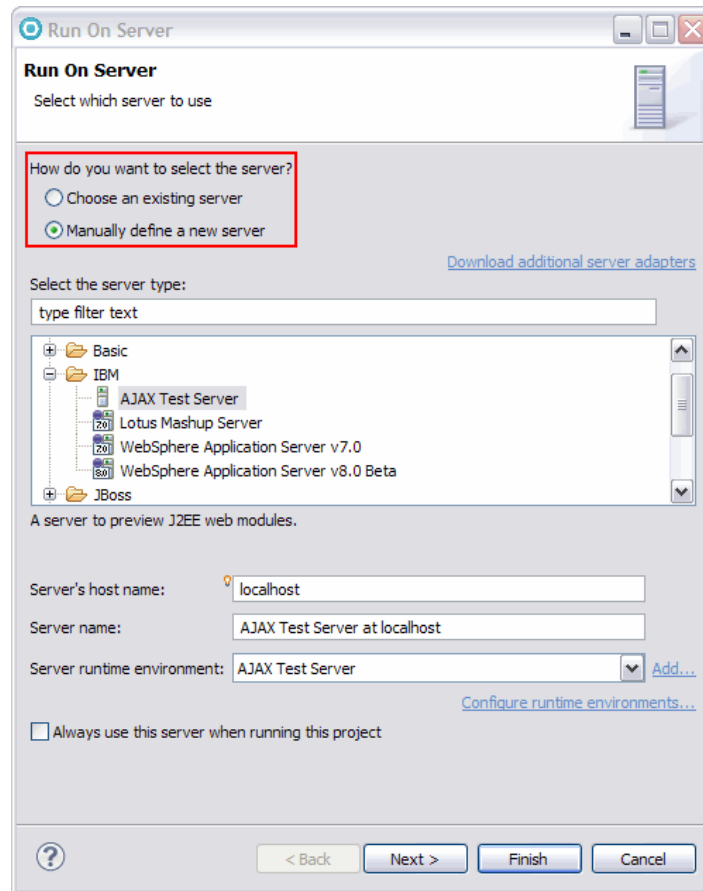


Figure 20-18 Creating a new Ajax Test Server

- c. By default, your page launches in the internal web browser (Figure 20-19 on page 1125). You can select another browser by clicking **Window** → **Web Browser**. If you want to use the integrated Firebug debugging support, select the **Firefox with Firebug** browser.



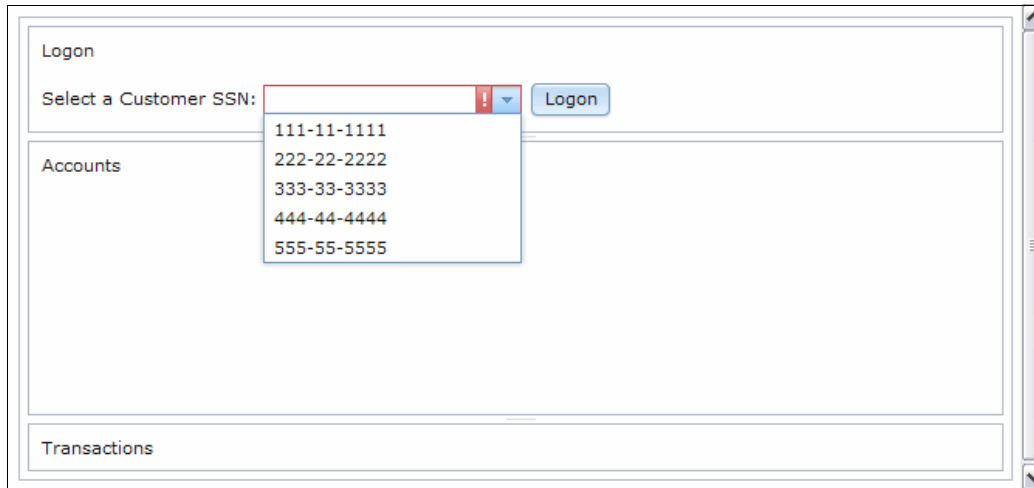


Figure 20-19 Web page loaded with sample JSON data displayed in the FilteringSelect widget

9. Now that you have verified that your widget works with the JSON file, connect the widget to the actual back-end service that will feed data to your application:
  - a. Open **CustomerLogon.js**.
  - b. Change the `url` attribute in the `postCreate` function to `"/RAD8Dojo/BankDataService?type=getCustomers"`, which is the path to your back-end servlet.
  - c. Save and close `CustomerLogon.js`.
10. Run the page on the server again to verify that your `FilteringSelect` widget is still being populated.

### 20.3.5 Adding a Dojo DataGrid to your web page

In this section, you add two `DataGrid`s to your web page to display customer accounts and account transactions. After the `Logon` button is clicked, the accounts grid loads. After a specific account is selected from the accounts grid, the transaction grid loads.

You use the `DataGrid` wizard in this section. Complete these steps:

1. Open **index.html** and switch to the **Design** view.
2. Expand the **Dojo Data Widgets** palette drawer.
3. Drag and drop the **DataGrid** widget onto the page beneath the **Accounts** label. The Dojo `DataGrid` wizard opens (Figure 20-20 on page 1126).

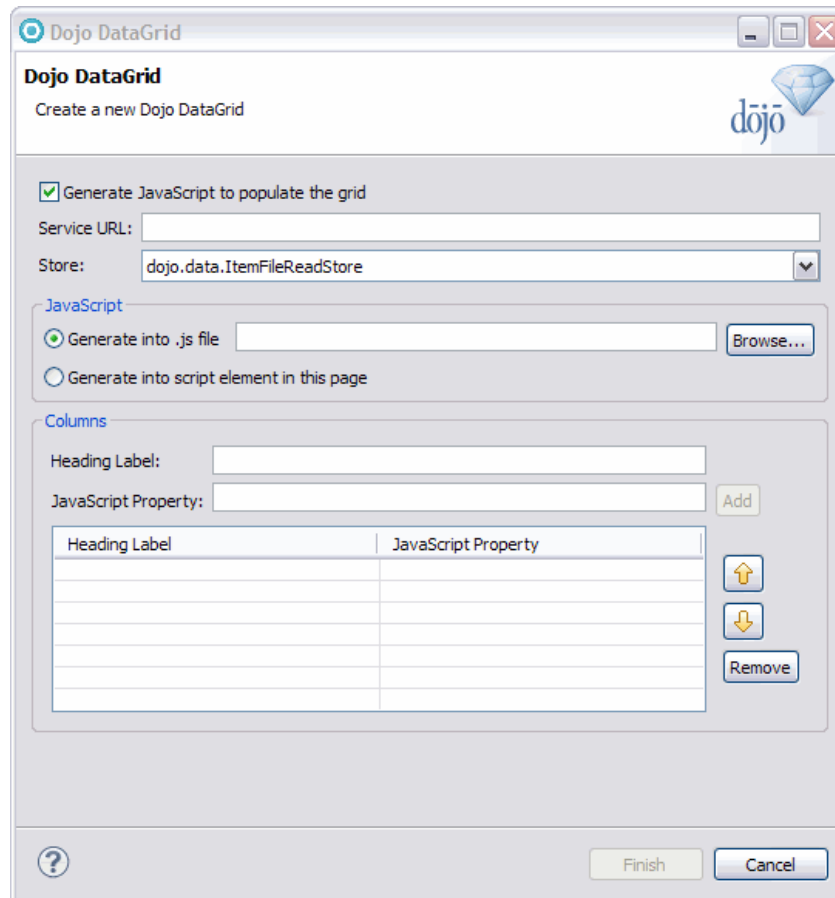


Figure 20-20 Dojo DataGrid wizard

4. On the Dojo DataGrid wizard, perform these steps:
  - a. Enter `/RAD8Dojo/BankDataService?type=getAccountsBySsn&id=` as the Service URL. After the code is generated, you will manually add the necessary URL parameter.
  - b. Click **Generate into .js file**.
  - c. Enter `/RAD8Dojo/WebContent/gridCode.js` as the path to the .js file.
  - d. Add two columns:
    - i. Enter `ID` into the Heading Label text field.
    - ii. Enter `id` into the JavaScript Property text field.
    - iii. Click **Add**.

- e. Repeat the previous steps for the heading Balance and the property balance.
5. Finish the wizard.

A table has been inserted into your web page, and the JavaScript code that is necessary to populate a DataGrid has been generated into a file named `gridCode.js`. In general, treat this generated code as a template that will be modified to suit the specific needs of your application. Complete these steps:

1. Switch to the **Source** view of your web page.
2. Find the script tag - `<script src="gridCode.js"></script>` - and delete it.
3. In the table that was created, add a style attribute and change the `id` to `accountGrid`:

```
<table id="accountGrid" dojotype="dojox.grid.DataGrid"
autowidth="true" rowselector="20px" style="height: 400px">
```

4. In the existing script region on your page, beneath the `dojo.require()` statements, type `dojo.add`, and invoke content assist.
5. Select the template for **dojo.addOnLoad**, which will be inserted into your page.
6. The `dojo.addOnLoad` function is a life-cycle method that runs after the DOM of a page has finished loading. The `dojo.connect` event handling method executes a given function when a specific action takes place. Inside the `dojo.addOnLoad` function, you add a call to `dojo.connect` to populate the account grid when the Logon button is clicked in the CustomerLogon widget. Follow these steps:
  - a. Type `dojo.co`, and invoke content assist. Here, you can select another of the default templates for `dojo.connect`. Insert it into your page.
  - b. For the `domNode` attribute, type `dijit.byId("CustomerLogon")`.
  - c. Change the value of the second attribute to `logon`, which is the name of the function in your widget that will process the event.
  - d. Next you need to write the code to grab the selected SSN from your custom widget. Inside the function that is generated by the `dojo.connect` template, add the code that is shown in Example 20-5.

*Example 20-5 Add code to `dojo.connect`*

---

```
var ssn = dijit.byId("CustomerLogon").currentSSN;
if(ssn == null || ssn == undefined || ssn == "") {
 alert("You must select a SSN.");
}
```

---

- e. Now we bring in our generated DataGrid template code. Open the **gridCode.js** file.
- f. Copy all of the code inside the addOnLoad function in the gridCode.js file, and paste it into the dojo.connect function, beneath the existing code, in index.html (Figure 20-21).

```

// Load the grid when the page initially loads
dojo.addOnLoad(function(){
 // If the data in the response does not conform to the Data Store's
 // structure, massage the response and return it in the proper format
 var filterData = function(response) {
 return response;
 };
 // Set up all grid event handlers
 var addListeners = function(dataStore) {
 };

 // Populate the Data Grid
 dojo.xhrGet({
 url: "/RAD8Dojo/BankDataService?type=getAccountsBySsn&id=",
 handleAs: "json",
 headers: {"If-Modified-Since":0},
 load: function(response){
 var filteredData = filterData(response);
 var dataStore = new dojo.data.ItemFileReadStore({data: filteredData, id:"dataStoreId"});
 var grid = dijit.byId("gridId");
 grid.setStore(dataStore);
 addListeners(dataStore);
 }
 });
});

```

Figure 20-21 The selected code to be copied and pasted into index.html

- g. Press Ctrl+Shift+F to format the copied code.
- h. At the end of the URL string, add + ssn:
 

```
url : "/RAD8Dojo/BankDataService?type=getAccountsBySsn&id=" + ssn,
```
- i. Change the line `var grid = dijit.byId("gridId");` to `var grid = dijit.byId("accountGrid");`
7. Run on the server, and verify that your page loads. Select an SSN from the FilteringSelect, click **Logon**, and verify that account data loads in the accounts grid.
8. Add a second DataGrid to display account transactions:
  - a. Switch the index.html editor back to the **Design** view.
  - b. Drag and drop the **DataGrid** widget onto the page beneath the **Transactions** label.
  - c. Clear **Generate JavaScript to population the grid**.

- d. Add two columns:
    - i. Enter ID into the Heading Label text field.
    - ii. Enter id into the JavaScript Property text field.
    - iii. Click **Add**.
  - e. Repeat for the heading Transaction Type and the property transType.
  - f. Repeat for the heading Amount and the property amount.
  - g. Finish the wizard.
9. Switch back to **Source** view. Complete these steps:
- a. In the table that was created, add a style attribute and change the id to transactionGrid:
 

```
<table id="transactionGrid" dojotype="dojox.grid.DataGrid"
autowidth="true" rowselector="20px" style="height: 400px">
```
  - b. Inside the existing **dojo.addOnLoad** function call, add a second call to `dojo.connect` using the provided content assist templates.
  - c. For the `domNode` attribute, type `dijit.byId("accountGrid")`.
  - d. Change the second attribute to `"onRowClick"`.
  - e. Add the parameter `row` to the function:
 

```
dojo.connect(dijit.byId("accountGrid"), "onRowClick",
function(row) {
```
  - f. Inside the function, add the line: `var accountId = row.grid.store.getValue(row.grid.getItem(row.rowIndex),"id");`
  - g. Copy all of the code inside the `addOnLoad` function in the `gridCode.js` file and paste it into the `dojo.connect` function in `index.html`.
  - h. Press `Ctrl+Shift+F` to format the copied code.
  - i. Change the URL string to:
 

```
"/RAD8Dojo/BankDataService?type=getTransactionsById&id=" +
accountId,
```
  - j. Change the line `var grid = dijit.byId("gridId");` to `var grid = dijit.byId("transactionGrid");`
10. Run on the server. You see similar results to Figure 20-22 on page 1130.

Logon

Select a Customer SSN:

---

Accounts

ID	Balance
002-222001	65484.23

---

Transactions

ID	Transactio	Amount
0000001	Credit	2222.22
0000002	Debit	800.8
0000003	Credit	21.5
0000004	Debit	1000.11
0000005	Debit	876.54
0000009	Credit	700.77

Figure 20-22 Completed Web 2.0 application



## Developing portal applications

In this chapter, we introduce support of the portal development tools that are included in IBM Rational Application Developer, with special focus on the features that have been added to Rational Application Developer. We also highlight how you can use the portal tools in Rational Application Developer to develop a portal and associated portlet applications for WebSphere Portal. In addition, we include a development scenario to demonstrate how to use the new integrated portal tooling to develop a portal, customize the portal, and develop two portlets.

The chapter includes the following sections:

- ▶ Introduction to portal technology
- ▶ Developing applications for WebSphere Portal
- ▶ New WebSphere portal and portlet development tools in Rational Application Developer
- ▶ Developing portal solutions using portal tools

The sample code for this chapter is in the 7835code\portal folder.

## 21.1 Introduction to portal technology

As Java 2 Platform, Enterprise Edition (J2EE) technology has evolved, much emphasis has been placed on the challenges of building enterprise applications and bringing those applications to the web. At the core of the challenges currently being faced by web developers is the integration of disparate user content into a seamless web application and well-designed user interface. Portal technology provides a framework to build these applications for the web.

Because of the increasing popularity of portal technologies, the tooling and frameworks that are used to support the building of new portals has evolved. The major job of a portal is to aggregate content and functionality. Portal servers provide the following advantages:

- ▶ A server to aggregate content
- ▶ A scalable infrastructure
- ▶ A framework to build portal components and extensions

Additionally, many portals offer personalization and customization features. Personalization enables the portal to deliver user-specific information targeting a user based on that user's unique information. Through customization features, users can organize the look and feel of the portal to suit their individual needs and preferences.

Portals deliver e-business applications over the web to many types of client devices from PCs to PDAs. Portals provide site users with a single point of access to multiple types of information and applications. Regardless of where the information resides or the format that it is in, a portal aggregates all of the information in a way that is relevant to the user.

The goal of implementing an enterprise portal is to enable a working environment that integrates people, their work, personal activities, and supporting processes and technology.

### 21.1.1 Portal concepts and definitions

Before beginning development for portals, familiarize yourself with the common definitions and descriptions of portal-related terminology.

#### **Portal page**

A *portal page* is a single web page that can be used to display content that is aggregated from multiple sources. The content that is displayed on a portal page is shown by an arrangement of one or more portlets. For example, a world stock market portal page might contain two portlets that display stock tickers for



popular stock exchanges and a third portlet that shows the current exchange rates for world currencies.

## Portlet

A *portlet* is an individual application that shows content on a portal page. To a user, a portlet is a single window on the portal page that provides information or web application functionality. To a developer, portlets are Java-based pluggable modules that can access content from a source, such as another website, an XML feed, or a database, and show this content to the user as part of the portal page.

Figure 21-1 shows a portal welcome page and its contained portlets.

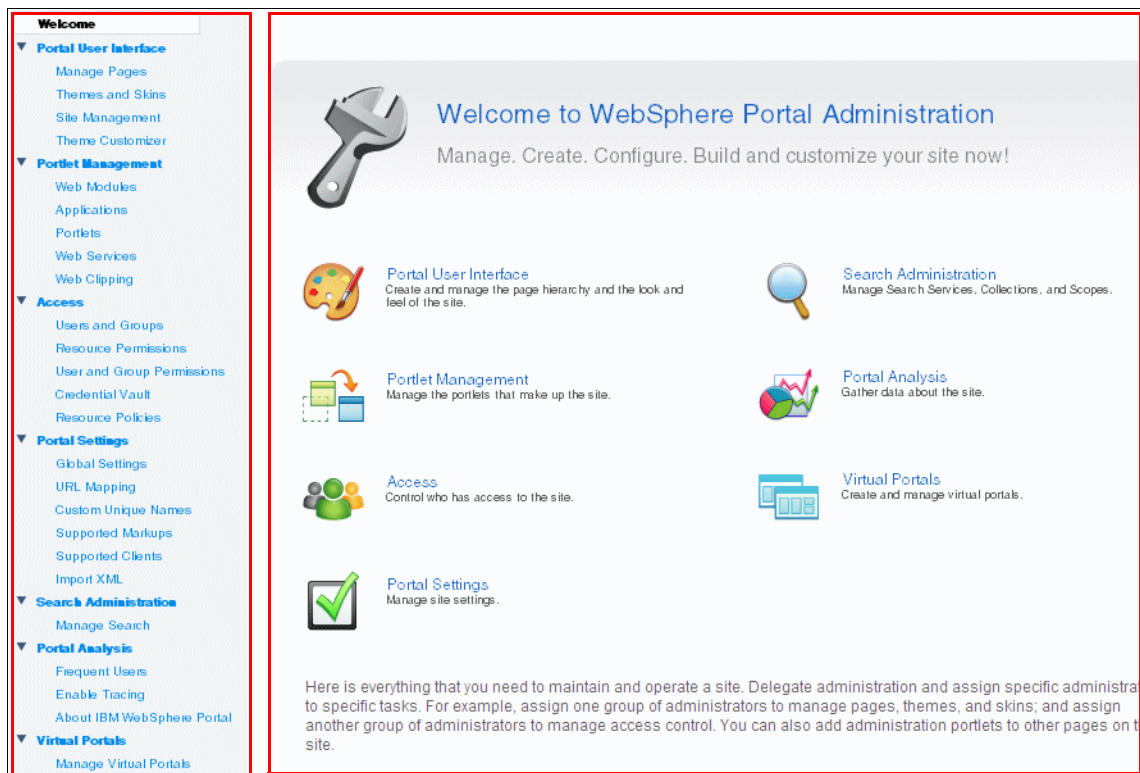


Figure 21-1 Portlets laid out on the Portal Welcome page

## Portlet application

A *portlet application* is a deployable unit that contains one or more portlets. It encapsulates all the resources required by the portlets, such as Java classes, JavaServer Pages (JSP) files, images, deployment descriptors, libraries, and other resources.

## Portlet states

*Portlet states* determine how individual portlets look when a user accesses them on the portal page. These states are similar to minimize, restore, and maximize window states of applications run on any popular operating system in a web-based environment.

The state of the portlet is stored in the `PortletWindow.State` object and can be queried for changing the way a portlet looks or behaves based on its current state. The IBM portlet application programming interface (API) defines the following possible states for a portlet:

- ▶ Normal: The portlet is displayed in its initial state, as defined when it was installed.
- ▶ Minimized: Only the portlet title bar is visible on the portal page.
- ▶ Maximized: The portlet fills the entire body of the portal page, hiding all other portlets.

## Portlet modes

With *portlet modes*, the portlet can display a separate *face* depending on how it is being used. Separate content can be displayed within the same portlet, depending on its mode. Modes are most commonly used to allow users and administrators to configure portlets or to offer help to the users. The IBM Portlet API has the following modes:

- ▶ View: The initial face of the portlet when created. The portlet normally functions in this mode.
- ▶ Edit: In this mode, the user can configure the portlet for that user's personal use (for example, specifying a city for a localized weather forecast).
- ▶ Help: If the portlet supports the help mode, this mode shows a help page to the user.
- ▶ Configure: If provided, this mode shows a face through which the portal administrator can configure the portlet for a group of users or a single user.

## Portlet events

Certain portlets only display static content in independent windows. To allow users to interact with portlets and to allow portlets to interact with each other, portlet events are used. *Portlet events* contain information to which a portlet might need to respond. For example, when a user clicks a link or button, an *action event* is generated. To receive notification of a given event, the portlet must also have the appropriate *event listener* implemented within the portlet class.

There are three commonly used types of portlet events:

- ▶ **Action:** Generated when an HTTP request is received by the portlet that is associated with an action, such as when a user clicks a link.
- ▶ **Message:** Generated when one portlet within a portlet application sends a message to another portlet.
- ▶ **Window:** Generated when the user changes the state of the portlet window.

## 21.1.2 IBM WebSphere Portal

IBM WebSphere Portal provides an extensible framework through which users can interact with enterprise applications, people, content, and processes. They can personalize and organize their own view of the portal, manage their own profiles, and publish and share documents. WebSphere Portal provides additional services, such as single sign-on (SSO), security, credential vault, directory services, document management, web content management, and personalization. Other services provided include search, collaboration, search and taxonomy, support for mobile devices, accessibility support, globalization, e-learning, integration to applications, and site analytics. Clients can further extend the portal solution to provide host integration and e-commerce.

With WebSphere Portal, you can plug in new features or extensions using portlets. In the same way that a servlet is an application within a web server, a portlet is an application within WebSphere Portal. Developing portlets is the most important task when providing a portal that functions as the user's interface to information and tasks.

Portlets are an encapsulation of content and functionality. They are reusable components that combine web-based content, application functionality, and access to resources. Portlets are assembled into portal pages that, in turn, make up a portal implementation.

Portal solutions, such as IBM WebSphere Portal, are proven to shorten the development time. Pre-built adapters and connectors are available so that developers can use the company's existing investment by integrating with the existing systems without re-inventing the wheel.

### **New features in WebSphere Portal V7.0**

With the following new features in WebSphere Portal V7.0, you can develop more robust enterprise solutions:

- ▶ Improved installation, migration, configuration, and security.
- ▶ A new task has been added to aid in automated log collection for problem reports opened with IBM Support.

- ▶ Easily add support for tagging, rating, blogs, blog archives, and wikis.
- ▶ Page Builder and Unified Theme Architecture:
  - The Page Builder theme is now the default theme.
  - A new unified architecture supporting portlets and widgets.
  - Both server-side and client-side page aggregation.
- ▶ The Unified Task List portlet provides a single point of integration with workflow management systems.
- ▶ Enhanced virtualization through VMware.

For more information about the WebSphere Portal V7.0 new features, see the IBM WebSphere Portal V7.0 Information Center at the following address:

<http://www-10.lotus.com/ldd/portals/wiki.nsf/xpViewCategories.xsp?lookupName=IBM%20WebSphere%20Portal%20%20Product%20Documentation>

**Portal projects:** The portal projects targeting IBM WebSphere Portal V7.0 are not supported. You cannot create and work with the portal projects targeting IBM WebSphere Portal V7.0, or later. Use the web-based Site Designing Portlet feature to manage portal projects.

### 21.1.3 Portal and portlet development features in Rational Application Developer

Rational Application Developer provides development tools for portal and portlet applications destined to WebSphere Portal. Several portal tools are bundled with IBM Rational Application Developer that allow you to create, test, debug, and deploy portal and portlet applications. Rational Application Developer supports portlet development using the Standard and IBM portlet APIs.

The following features are provided to support the development of portlet applications:

- ▶ New Portlet Project wizard
- ▶ Support for creating and publishing iWidget projects
- ▶ Support for iWidgets in portlet projects
- ▶ Dojo tools
- ▶ Advanced Dojo tools for client side Inter-Portlet Communications (IPC)
- ▶ “What you see is what you get” (WYSIWYG) Portlet deployment descriptor editor
- ▶ *Java Specification Request (JSR) 286: Event Publish and Subscribe Wizards*

- ▶ Cooperative portlet wizards
- ▶ Ajax proxy tooling
- ▶ Remote Procedure Call (RPC) tooling
- ▶ Person menu and person menu extension support
- ▶ Client-side programming model support
- ▶ Client-side click-to-action support
- ▶ Client Side Aggregation (CSA 2.0) support
- ▶ Personalization wizard
- ▶ Person tagging support
- ▶ Portlet application samples
- ▶ Portal server configuration
- ▶ Portal server test environment
- ▶ Portal server test, debug, and deploy
- ▶ Remote Portal server start-up option
- ▶ Hot deployment
- ▶ Portal theme support enhancement
- ▶ Import and export web archive (WAR) file
- ▶ JavaServer Faces (JSF) 1.2 features support
- ▶ Java Persistence API (JPA)-enabled portlets
- ▶ Static page aggregation
- ▶ Friendly URL support
- ▶ Business process message access
- ▶ Site Designing Portlet
- ▶ Visual tooling to insert portlet objects into JSP files, using Page Designer

### **Portal test environments**

Rational Application Developer supports the following versions of integrated test environments to run and test your portal and portlet projects from within the Rational Application Developer workbench:

- ▶ IBM WebSphere Portal Server V6.1
- ▶ IBM WebSphere Portal Server V6.1 on WebSphere Application Server V7
- ▶ IBM WebSphere Portal Server V7.0

In this chapter, we use Version 7.0 of WebSphere Portal running under WebSphere Application Server V7.0.

### **Enabling the portal development capability**

By default, Rational Application Developer portal development capability is not enabled. To enable portal development capability, follow these steps:

1. Select **Window** → **Preferences**.
2. In the Preferences window, expand **General** → **Capabilities** and click **Advanced**.
3. In the Advanced window, expand **Web Developer (advanced)**, select **Portal Development** and click **OK**.
4. Back in the Preferences window, click **OK**. Portal development is now enabled.

Follow the instructions in the Help topics about Developing Portal Applications to ensure that the configuration in the development environment accurately reflects the configuration of the staging or runtime environment. If you do not do this step, you might experience compilation errors after the product is imported or encounter unexpected portal behaviors.

## **21.1.4 Setting up Rational Application Developer with the Portal test environment**

Setting up a Portal test environment in Rational Application Developer is now a much easier and more streamlined task. We perform the following high-level activities to complete the setup of the Portal test environment in Rational Application Developer, as explained in “Installing WebSphere Portal V7” on page 1806:

- ▶ Installing WebSphere Portal V7
- ▶ Adding WebSphere Portal V7 to Rational Application Developer
- ▶ Optimizing the WebSphere Portal Server for development

## **21.2 Developing applications for WebSphere Portal**

Rational Application Developer includes many tools to help you quickly develop portals and individual portlet applications. In this section, we cover simple portlet development strategies and provide an overview of the tools that are included in Rational Application Developer to aid with the development of WebSphere Portal applications.

## 21.2.1 Portal samples and tutorials

Rational Application Developer also comes with several samples and tutorials to aid you with the development of WebSphere Portal applications. The Samples Gallery provides example portlet applications to illustrate portlet development.

To access portlet samples, click **Help** → **Help Contents** → **Samples**. Then expand **Portlet** (Figure 21-2). Here, you can select from a number of Portlet examples.

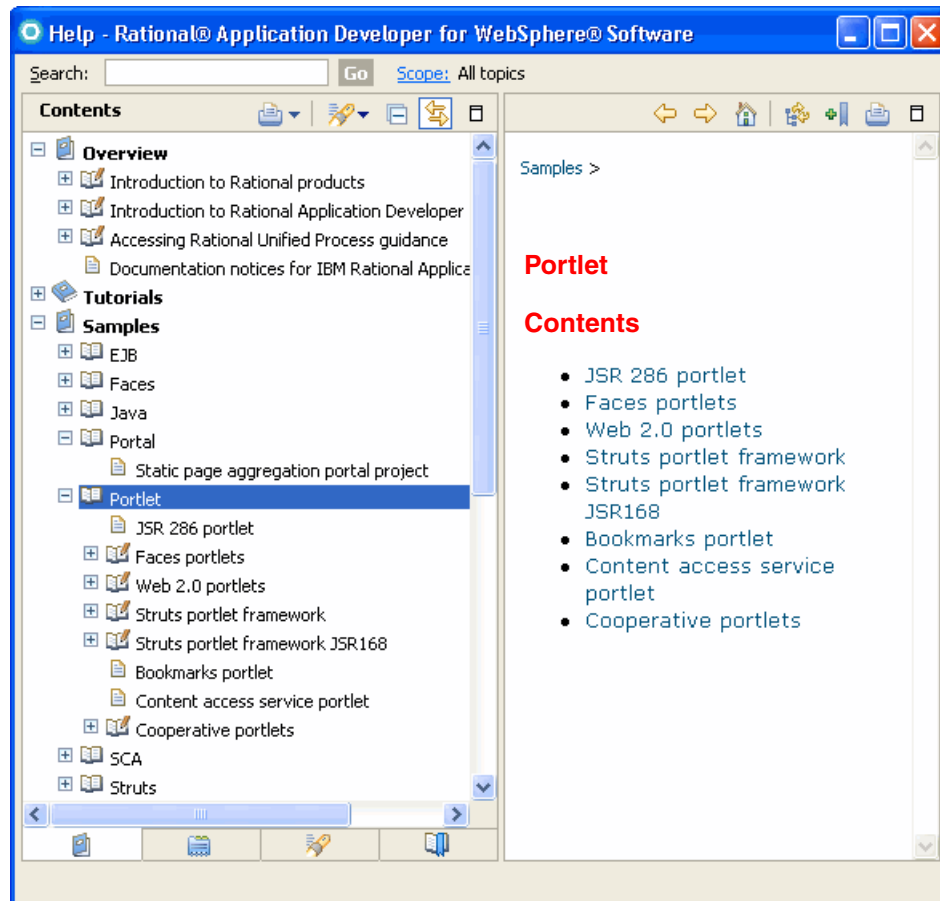


Figure 21-2 Portal and Portlet development technology samples

The Tutorials Gallery provides detailed tutorials to illustrate portlet development. To access the tutorials, select **Help** → **Help Contents** → **Tutorials**. Then expand **Portal**. You can select **Explore Portal Designer**.

## 21.2.2 Development strategy

A portlet application consists of Java classes, JSP files, and other resources, such as deployment descriptors and image files. Before beginning development, you must make several decisions regarding the development strategy and technologies that are used to develop a portlet application.

### Choosing a portlet API: JSR 168, JSR 286, or IBM

Rational Application Developer supports the development of portlets using the JSR 168 portlet API, the JSR 286 portlet API, and the IBM Portlet API. All three portlets can be deployed to WebSphere Portal.

In this section, we provide information to help you decide which API to use when you develop portlets:

- ▶ JSR 168 portlet API is a Java specification from the Java Community Process that addresses the requirements of aggregation, personalization, presentation, and security for portlets running in a portal environment. Portlets that conform to the JSR 168 specification are more portable and reusable, because they can be deployed to any JSR 168-compliant portal.

Rational tools support portlet development based on the JSR 168 specification. For more information about this API, see the following web address:

<http://www.jcp.org/en/jsr/detail?id=168>

- ▶ JSR 286 portlet API is a Java specification from the Java Community Process that has improved upon the JSR 186 portlet API by providing additional capabilities, such as filters, events, and public render parameters. These improvements have necessitated changes to the XML Schema Definition (XSD) for the Portlet Deployment Descriptor (PDD) that adds new elements to it.

For more information about this API, see the following web address:

<http://www.jcp.org/en/jsr/detail?id=286>

- ▶ IBM Portlet API was initially supported for WebSphere Portal V4.x, and in subsequent versions of WebSphere Portal V5.x and V6.x. The IBM Portlet API is deprecated in V6.x, but is still supported. No new functionality will be added. Use the standard Portlet API.

### Deciding which API to use

The IBM Portlet API extends the servlet API and many of the major interfaces (request, response, and session). JSR 168 API does not extend the servlet API, but shares many of the same characteristics. JSR 168 uses much of the functionality that is provided by the servlet specification, such as deployment,



class loading, web applications, web application life-cycle management, session management, and request dispatching.

For new portlets, consider using JSR 286 to take advantage of its additional capabilities. If you cannot use JSR 286, consider using JSR 168 when its functionality is sufficient for the portlets or when the portlet is expected to be published as a Web Services for Remote Portlets (WSRP) service. WSRP is another portal-based standard that is used to integrate the presentation of remote portlets provided as web services into the local portal page. The concepts in JSR 168 and WSRP have been aligned to allow JSR 168 portlets to be published as web services. Several of these concepts include portlet modes and states, URL and namespace encoding, and the handling of transient and persistent information.

## Choosing markup languages

WebSphere Portal supports multiple client types by generating pages in multiple markup languages. By using Rational Application Developer tools, you can develop portlet applications that support these markup languages. The following markup languages are officially supported:

- ▶ cHTML is a markup language for mobile devices in the NTT DoCoMo i-mode network.
- ▶ HTML is a markup language for desktop computers. All portlet applications must support HTML, at a minimum.
- ▶ Wireless Markup Language (WML) is a markup language for Wireless Application Protocol (WAP) devices, which are typically mobile phones.

To edit WML files and cHTML files, you can use Page Designer, as you do when editing other web content.

To run or debug a portlet application that supports WML or cHTML, you can use a device emulator that is provided by a device vendor. To add a device emulator, follow these steps:

1. Select **Window** → **Preferences**.
2. In the Preferences window, select **General** → **Web browser**.
3. Click **New** to locate and define a web browser type that is appropriate for the device that you want to test and debug.

For more information about markup languages, see the Markup guidelines topic in the WebSphere Portal Information Center:

[http://infolib.lotus.com/resources/portal/7.0.0/doc/en\\_us/pt700abd001/html-wrapper.html](http://infolib.lotus.com/resources/portal/7.0.0/doc/en_us/pt700abd001/html-wrapper.html)

## Choosing other frameworks

JavaServer Faces (JSF), Struts technology, iWidgets, and Dojo can be incorporated into a portlet development strategy easily. Rational Application Developer provides extensive tooling support to help in the creation and code generation for creating a JSF portlet, Struts portlet, iWidget portlet, or Dojo portlet.

### ***JavaServer Faces***

Faces-based application development can be applied to portlets, similar to the way that Faces development is implemented in web applications. Faces support in Rational Application Developer simplifies the process of writing Faces portlet applications and eliminates the need to manage many of the underlying requirements of portlet applications.

Rational tools provide a set of wizards that help you create Faces portlet-related artifacts. In many cases, these wizards are identical to the wizards that are used to create standard Faces artifacts.

See the Rational Application Developer Faces documentation in the information center for usage details at the following address:

<http://publib.boulder.ibm.com/infocenter/radhelp/v7r0m0/topic/com.ibm.e.tools.jsf.doc/topics/cjsfover.html>

Also, see Chapter 19, “Developing web applications using JavaServer Faces” on page 1057, for more detailed information about application development using the JSF framework.

### ***Struts***

Struts-based application development can also be applied to portlets, similar to the way that Struts development is implemented in web applications. The Struts Portal Framework (SPF) was developed to merge these two technologies. SPF support in Rational Application Developer simplifies the process of writing Struts portlet applications and eliminates the need to manage many of the underlying requirements of portlet applications.

The Struts portlet tooling supports the development of portlet applications based on both the JSR 168 API and the IBM Portlet API. There are differences in the runtime code that is included with projects, tag libraries supported, Java class references, and configuration architecture. Unless otherwise noted, these differences are managed by the product tooling.

In addition, multiple wizards are present to help you create Struts portlet-related artifacts. These wizards are the same wizards that are used in Struts development. See the Rational Application Developer Struts documentation for usage details.

You can find more information about Struts at the following web addresses:

- ▶ <http://struts.apache.org/>
- ▶ <http://publib.boulder.ibm.com/infocenter/radhelp/v7r0m0/topic/com.ibm.etools.struts.doc/topics/cstrdoc007.html>

We explain web development using iWidgets in Chapter 22, “Developing Lotus iWidgets” on page 1183.

### 21.2.3 Portal tools for developing portals

A *portal* is essentially a J2EE web application. It provides an aggregation framework where developers can associate many portlets and portlet applications by using one or more portal pages.

Rational Application Developer includes several new portal site creation tools that enable you to visually customize portal page layout, themes, skins, and navigation.

#### Portal Import wizard

One way to create a new Portal project is to import an existing portal site from a WebSphere Portal server into Rational Application Developer. Importing is also useful for updating the configuration of a project that already exists in Rational Application Developer.

The portal site configuration on WebSphere Portal server contains the following resources: global settings, resource definitions, portal content tree, and page layout. Importing these resources from WebSphere Portal server to Rational Application Developer overwrites duplicate resources within the existing portal project. Non-duplicate resources from the server configuration are copied into the existing portal project. Resources that exist only in the portal project are not affected by the import.

Rational Application Developer uses the XML configuration interface to import a server configuration and optionally retrieves files from the `installedApps/node/wps.ear` file of the WebSphere Portal Server installation. These files include the JSP, Cascading Style Sheet (CSS), and image files for themes and skins. When creating a new portal project, retrieving files is mandatory. To retrieve files, Rational Application Developer must have access to this directory, as specified when you define a new server for this project.

#### New Portal Project wizard

The New Portal Project wizard guides you through the process of creating a portal project within Rational Application Developer.

During this process, you are able to perform the following actions:

- ▶ Specify a project name.
- ▶ Select the version of the portal server.
- ▶ Select a default theme.
- ▶ Select a default skin for the chosen theme.

**Important:** Do not name your project *wps* or anything that resembles this string to avoid internal naming conflicts.

The project that you create with this wizard does not have any portlet definitions, labels, or pages. The themes and skins that are available in this wizard are the same as if you had imported a portal site from a WebSphere Portal server. To create a new Portal Project, follow these steps:

1. Select **File** → **New** → **Project** → **Portal**.
2. In the Select a wizard window, expand **Portal** and select **Portal Project**. Click **Next**.
3. In the New Portal Project window (Figure 21-3 on page 1145), complete these actions:
  - a. In the Project Name field, type a name, for example, MyPortal.
  - b. Select **Use default**.
  - c. For Select the portal server version, select **6.1.0.1**.
  - d. For the Target Runtime, select **WebSphere Portal v6.1 on WAS 7**.
  - e. For the EAR Project Name, type a name, for example, MyPortalEAR.
  - f. Click **Next**.

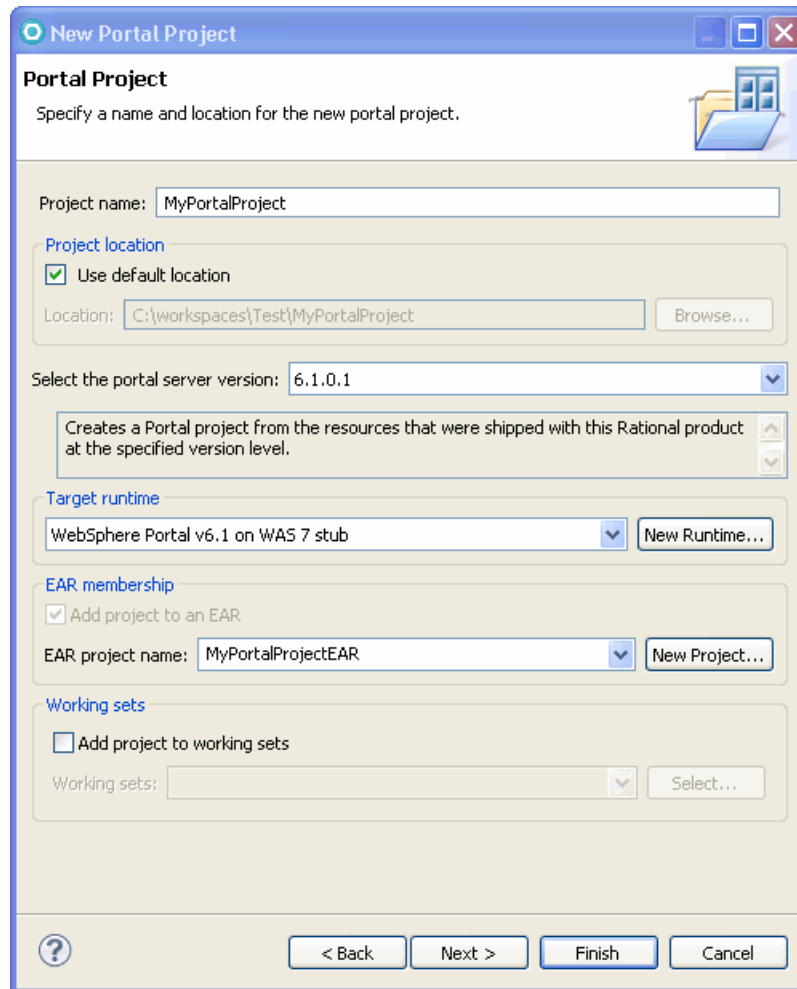


Figure 21-3 New Portal Project wizard

4. In the Select Theme window, select the default theme (**Portal**) and click **Next**.
5. In the Select Skin window, select the default skin (**IBM**) and click **Finish**.

The Portal Designer editor opens with the selected theme and skin.

## Portal Designer

For portal site layout and appearance, you can think of Portal Designer as a “what you see is what you get” (WYSIWYG) editor. It renders the graphic interface of items, such as themes, skins, page layouts, and simple portlets.

Portal Designer also shows the initial pages of JSF and Struts portlets within your portal pages. It does not display the content for WSRP. For more information about how to display simple portlets, see “Viewing portlets in Portal Designer” in the Rational Application Developer Help.

Use Portal Designer to customize both the graphic design of your portal and the layout of your portal pages. Use it as you might use any WYSIWYG web editor. You can perform the following capabilities, among other capabilities, with this editor:

- ▶ Right-clicking a design element. You can select an insert menu item.
- ▶ Clicking a design element to edit its properties on the Properties tab.
- ▶ Using Page Designer to alter the graphic design of themes, skins, and styles. These editors are available in the Edit menu.

PortalConfiguration is the name of the layout source file that resides in the root of the portal project folder (Figure 21-4 on page 1147). To open Portal Designer, double-click the **ibm-portal-topology.xml** file under the PortalConfiguration folder in the Enterprise Explorer.

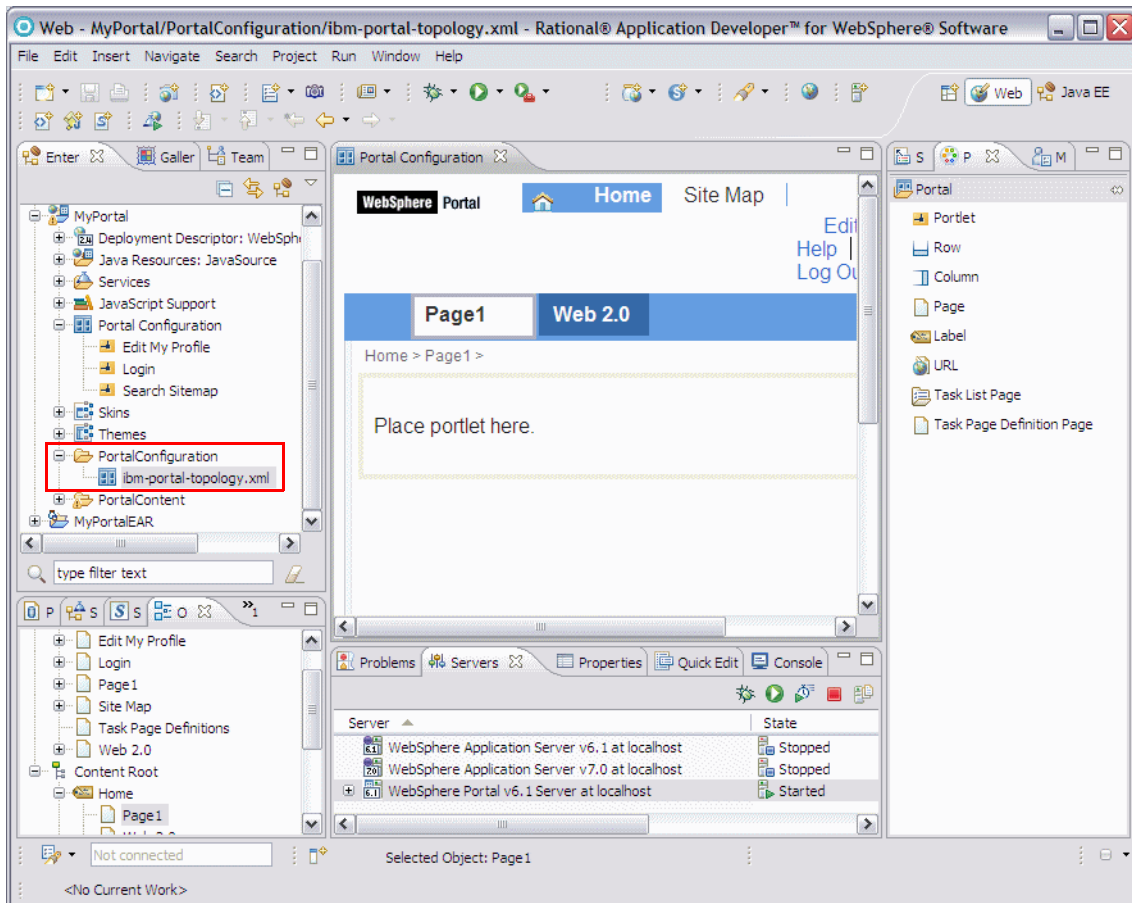


Figure 21-4 Portal Designer workbench

## Skin and theme: Designing and editing

A *skin* is the border around each portlet within a portal page. Unlike themes, which apply to the overall look and feel of the portal, skins are limited to the look and feel of each portlet that you insert into your portal application.

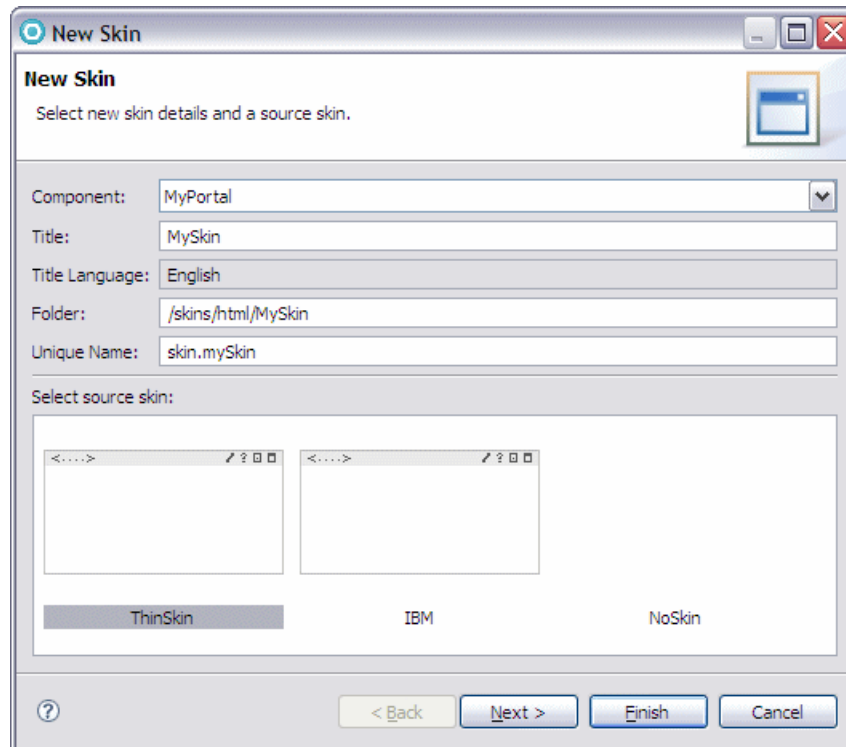
Rational Application Developer installation includes pre-built themes and skins to use with portal projects. Also, wizards can create new themes and skins. Changing themes and skins was previously done through portal administration. In addition to these wizards for creating new skins and themes, there are tools that can be used to change or edit skins and themes.

After you create them, the skins and themes are displayed in the Enterprise Explorer view. Double-click a skin or theme to manually edit it.

## ***New Skin wizard***

In addition to using the pre-defined skins that came with the installation, you can use the New Skin wizard to create customized skins for your project:

1. Right-click the Portal project in the Enterprise Explorer view and select **New** → **Skin**. The New Skin window opens (Figure 21-5).



*Figure 21-5 New Skin wizard*

2. Type the name of the new skin to be created and click the source skin from which the new skin will be copied. Using a source skin allows you to take an existing skin and add customizations while still retaining the original skin. After you choose the source skin, click **Next**.
3. In the Select Themes window, select the themes that allow the skin that you created and click **Finish**.
4. The new skin is saved in the Skins folder of the portal project. From this point, you can use the Page Designer to edit the skin and apply the customizations.



## New Theme wizard

Themes provide the overall look and feel of your portal application. In addition to using the pre-existing themes, you can use the New Theme wizard to create customized themes for your project:

1. Right-click the portal project in the Enterprise Explorer view and select **New** → **Theme**. The New Theme window opens (Figure 21-6).

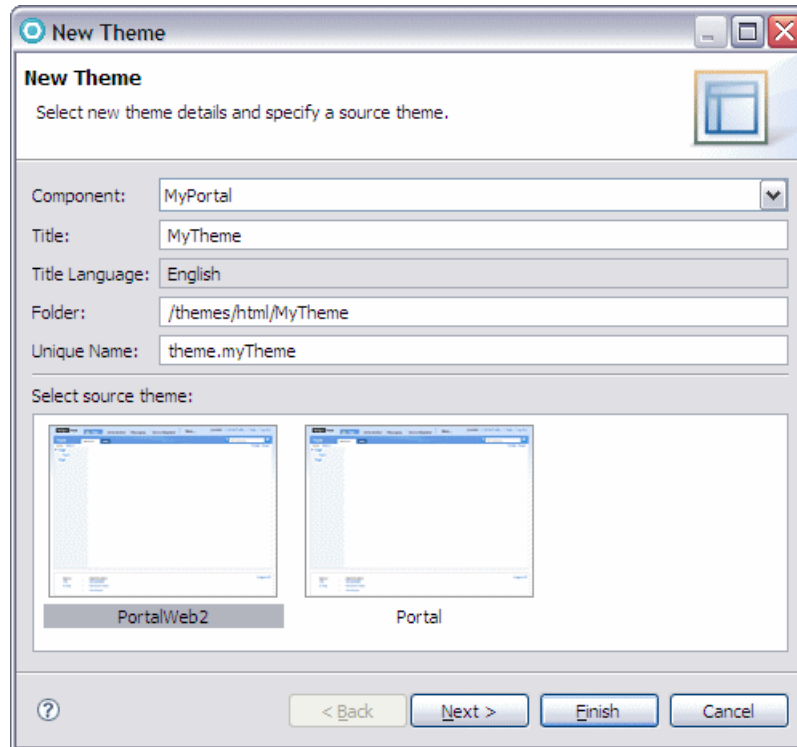


Figure 21-6 New Theme wizard

2. Type the name of the new theme to be created and click the source theme from which the new theme will be copied. Using a source theme allows you to take an existing theme and add customizations while still retaining the original theme. After you choose the source skin, click **Next**.
3. In the Select Skins window, select the allowed skins for the theme and click **Finish**.
4. The new theme is saved in the Themes folder of the portal project. From this point, you can use the Page Designer to edit the theme and apply the customizations.

## 21.3 New WebSphere portal and portlet development tools in Rational Application Developer

For those who are already familiar with portal and portlet development tools in the previous version of the Rational Application Developer, we briefly introduce the new portal and portlet tooling in Rational Application Developer.

### 21.3.1 Support for WebSphere Portal Server V7

In Rational Application Developer, you can create a portal-based project that is targeted to WebSphere Portal V7.0 at run time. You can target a local run time if you have a portal run time installed on your local machine. Otherwise, you can target server stubs to create the portlet project. To test the project, you can create the server instance of the desired run time and then publish the portlet project by using either the Run On Server or the Deploy option.

**Important:** Tool support for portal projects does not exist for IBM WebSphere Portal V7.0 run time.

### 21.3.2 Site Designing Portlet

The Site Designing Portlet is a web-based user interface that is used to customize and manage the portal site. You can use this user interface to change the appearance and layout of your portal site. You can add pages to a portal site, add portlets to a portal site, and add wire between portlets on a portal page.

### 21.3.3 New portlet project features

Two new portlet project features have been added into Rational Application Developer. These new portlet project features include the Dojo-enabled portlets and the creation of iWidgets inside a portlet project. The Dojo-enabled portlets run the portal runtime environment, with the option to create custom widgets that can be reused and enhanced further for WebSphere Portal V6.1 and later. You can also create and publish iWidget projects or create widgets inside a portlet project on the WebSphere Portal Version 6.1.5 and later runtime environment.

## 21.3.4 RPC tooling for portlet projects

The Remote Procedure Call (RPC) adapter for portlet projects allows JavaScript or client-side code, such as the Dojo library, to directly invoke and consume server-side logic.

## 21.4 Developing portal solutions using portal tools

In this section, we provide an example to develop eventing portlets.

### 21.4.1 Developing event handling portlets

*Events* are a powerful and flexible mechanism for communication between JSR 286 portlets. Events can be used to exchange complex data between portlets and to trigger portlet activity, such as updates to back-end systems. In the portal, they can also work with other communication mechanisms, such as cooperative portlets and click-to-action portlets.

Rational Application Developer provides wizards and user friendly editors to create and configure events. Wizards help portlets publish and receive events. You specify a unique name for the event that has to be published or processed, using either the default namespace or a custom namespace. In addition, the alias address in the namespace indicates that these events are compatible with any events of another portlet that has the same alias and can, therefore, work with input or output values of the provided address type.

Events subscribe to a server-side model or to a client-side model (IBM API and JSR API portlets targeted to WebSphere Portal V6.1) for declaring, publishing, and sharing information. Events can be distributed between local and remote portlets.

Server-side model portlets communicate with each other using the WebSphere Portal property broker. These portlets subscribe to the broker by publishing typed data items, or properties, that they can share as a provider or as a recipient.

### Project setup

We use two projects for this application:

- ▶ RAD80PortletEventEAR: Enterprise application
- ▶ RAD80PortletEvent: Application with two portlets (based on JSR 286)

Import these projects from the

C:\7835code\portal\RAD80PortletEventStart.zip project archive file.

## Structure of the sample application

The sample application consists of the following portlets that generate a `processAction` method:

- ▶ `CityPortlet` contains a list of cities for the user to select.
- ▶ `CityInfoPortlet` shows detailed information about the city that is selected.

## Creating an event to connect the portlets

Create an event to connect the `CityPortlet` and `CityInfoPortlet`:

1. In the Enterprise Explorer, expand **RAD80PortletEvent**.
2. From the Palette, expand the Portlet drawer. Drag the **Event Source Trigger** palette item onto the portlet JSP file where you define the events for the portlet application (Figure 21-7).

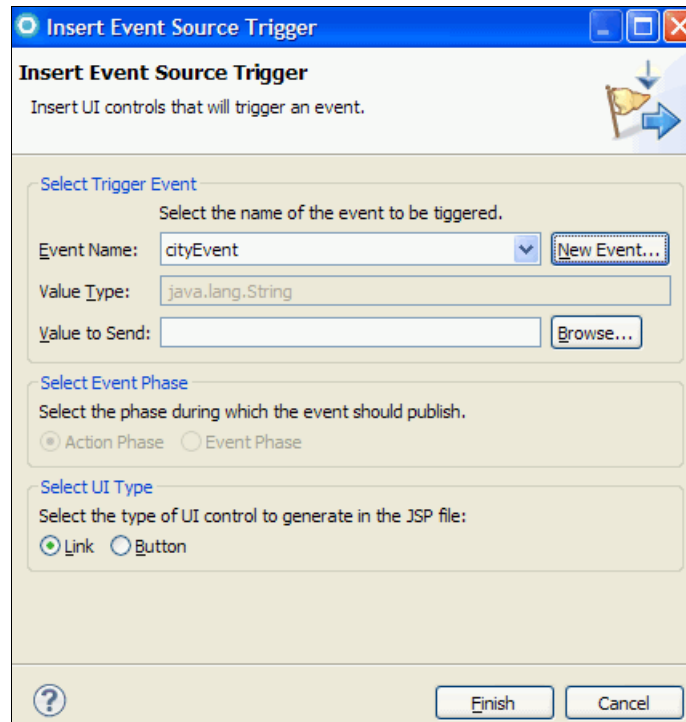


Figure 21-7 Inserting the event source trigger

3. Click **New Event**, and the Enable this Portlet to Publish events window opens.
4. Enter `cityEvent` in the Event name field and click **Finish**.

To enable a portlet to process an event, follow these steps:

1. In the Enterprise Explorer, expand **RAD80PortletEvent** → **Portlet Deployment Descriptor**.
2. Right-click **CityInfoPortlet** and select **Events** → **Enable this portlet to process events**.
3. In the Event: Enable this Portlet to Process events window, for the Event name, select **cityEvent** and click **Finish**.

As a result, a processEvent method is added to the CityInfoPortlet class (Example 21-2).

*Example 21-1 CityInfoPortlet processEvent method*

---

```
public void processEvent(EventRequest request, EventResponse response)
throws PortletException, java.io.IOException {
 Event sampleEvent = request.getEvent();
 if(sampleEvent.getName().toString().equals("cityEvent")) {
 Object sampleProcessObject = sampleEvent.getValue();
 }
}
```

---

## Adding the event logic to the two portlets

You have to put logic into the processAction method of the CityPortlet class and the processEvent method of the CityInfoPortlet class. When the user submits the CityPortlet, the processAction method is called. You have to trigger an event with the value of the selected city to the CityInfoPortlet to process this event.

To add the event logic to the portlets, follow these steps:

1. Open the **CityPortlet** class and change the processAction method (Example 21-2).

*Example 21-2 CityPortlet class processAction method (updated)*

---

```
public void processAction(ActionRequest request, ActionResponse
response)
 throws PortletException, java.io.IOException {
 //Initialize the fields in the class as per your requirement
 java.lang.String sampleObject = new java.lang.String();
 response.setEvent("cityEvent", sampleObject);
 String city = request.getParameter("cityCombo");
 response.setEvent("cityEvent", city);
}
```

---

2. Open the **CityInfoPortlet** class and change the `processEvent` method to get the value of the event received. Delegate this value to the `CityDB` helper class, which returns a `CityInfoBean` object with information about the selected city (Example 21-3).

*Example 21-3 CityInfoPortlet class processEvent method (updated)*

---

```
public void processEvent(EventRequest request, EventResponse
response)
 throws PortletException, IOException {

 Event sampleEvent = request.getEvent();
 if(sampleEvent.getName().toString().equals("cityEvent")) {
 Object sampleProcessObject = sampleEvent.getValue();
 String city = (String) sampleEvent.getValue();
 cityInfoBean = CityDB.getCityInfo(city);
 }
}
```

---

The `doView` method of the `CityInfoPortlet` class puts the `cityInfoBean` on request scope and forwards the processing to a JSP page to display the result (Example 21-4).

*Example 21-4 CityInfoPortlet class doView method*

---

```
public void doView(RenderRequest request, RenderResponse response)
 throws PortletException, IOException {
 // Set the MIME type for the render response
 response.setContentType(request.getResponseContentType());
 request.setAttribute("info", cityInfoBean);
 // Invoke the JSP to render
 PortletRequestDispatcher rd =
getPortletContext().getRequestDispatcher
 (getJspFilePath(request, VIEW_JSP));
 rd.include(request, response);
}
```

---

The `CityInfoPortletView.jsp` in the `CityInfoPortlet` retrieves the `cityInfoBean` and shows information about the selected city.

## Deploying and running the event handling portlets

In the Enterprise Explorer, right-click **RAD80PortletEvent** → **Run As** → **Run on Server**. In the subsequent window, click **Finish** to complete publishing to the WebSphere Portal server. After the application is deployed, Rational Application Developer opens a browser in the workbench (Figure 21-8 on page 1155).

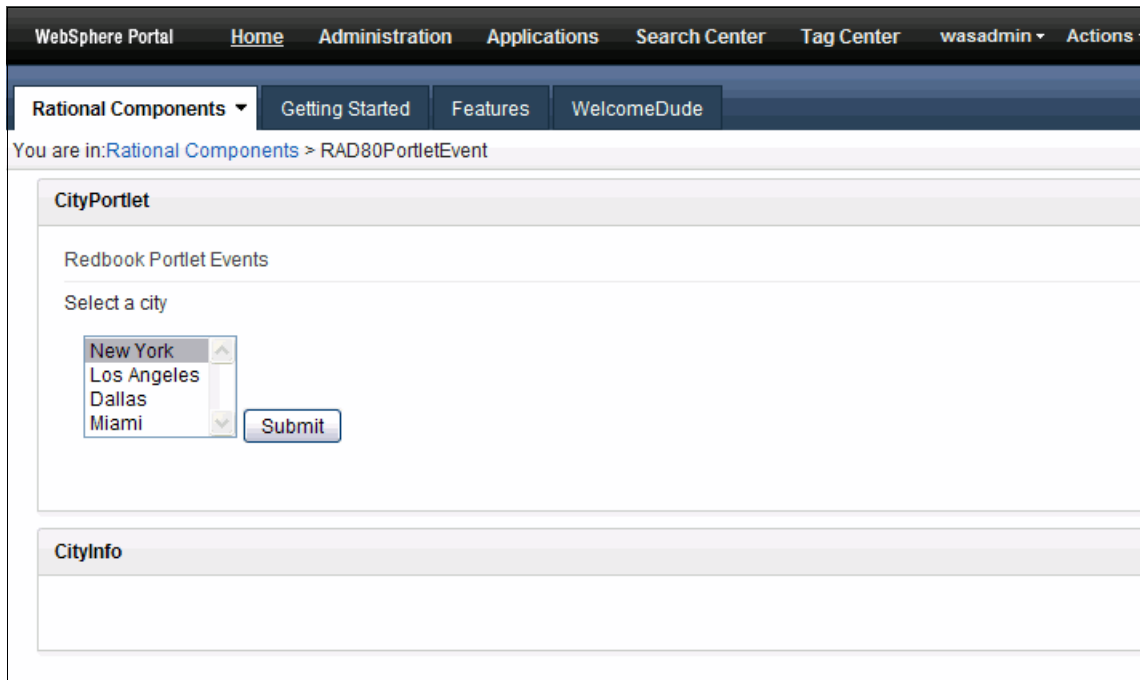


Figure 21-8 Browser with CityPortlet

## Connecting the portlets

You need to use wires to exchange information or actions between portlets. To access the wiring tool, follow these steps:

1. With the browser opened with the CityPortlet, click the **Actions** menu option. When the pop-up menu appears, select **Edit Page Properties** (Figure 21-9).

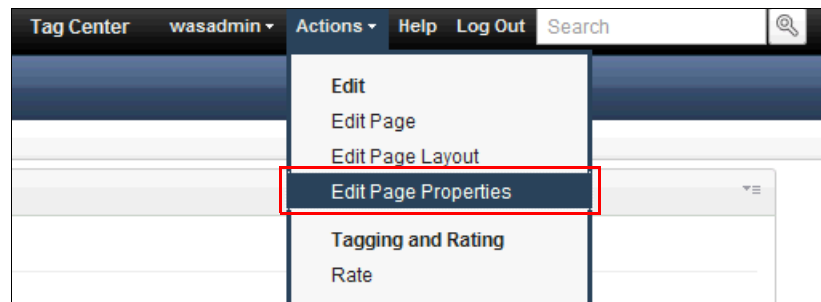


Figure 21-9 Edit Page Properties

2. In the Page Properties window, for the Unique name, type `rad80.portlet.page` and click **OK**.
3. Again, click the **Actions** menu option and select **Edit Page Layout**.
4. In the Page Customizer window, select the **Wires** tab (Figure 21-10).

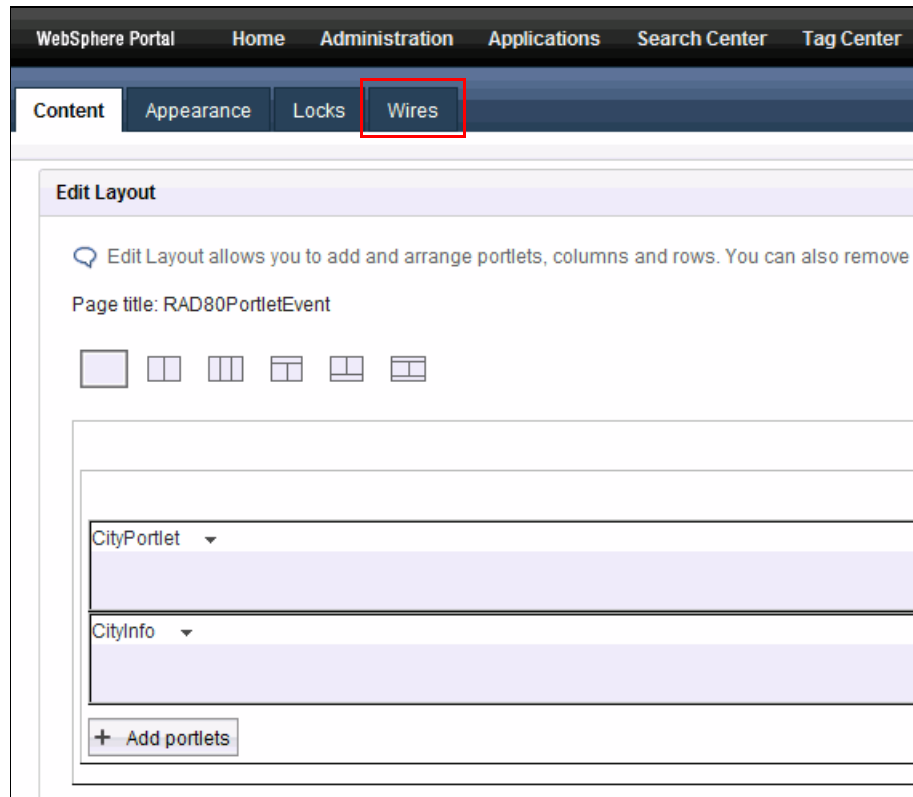


Figure 21-10 Edit Page Layout

5. On the Portlet Wiring Tool page, connect the portlets (Figure 21-11 on page 1157):
  - a. For the Source Portlet, select **CityPortlet**.
  - b. For Sending, select **publish.{http://RAD80Portlet/}cityEvent**.
  - c. For the Target Page, select **RAD80PortletEvent (rad80.portlet.page)**.
  - d. For the Target portlet, select **CityInfo**.
  - e. For Receiving, select **process.{http://RAD80Portlet/}cityEvent**.
  - f. Click the plus icon (+) to add the wire.
  - g. Click **Done** and then click **Home**.



WebSphere Portal   Home   Administration   Applications   Search Center   Tag Center

Content   Appearance   Locks   **Wires**

### Portlet Wiring Tool

**i** EJPART050I: No wires have been created on the page.

Wires are connections between portlets. The portlet wiring tool allows you to view, add, and delete wires. To add a new wire, click the Add Wire button, enter the wire details, and click Add Wire.

Wires for page: **RAD80PortletEvent**

Source portlet	Sending	Target page	Switch Target page
CityPortlet	publish.{http://rad80Portlet}/cityEvent	RAD80PortletEvent (rad80.portlet.page)	CityInfo

Consider semantic types only for matching of sources and targets  
 Consider payload type only for matching of sources and targets  
 Consider semantic types or payload type for matching of sources and targets

**Done**

Figure 21-11 Portlet Wiring Tool (compressed)

## Testing the application

To test the event link between the two portlets, select **New York** in the first portlet and click **Submit**. The second portlet shows information about the selected city (Figure 21-12).

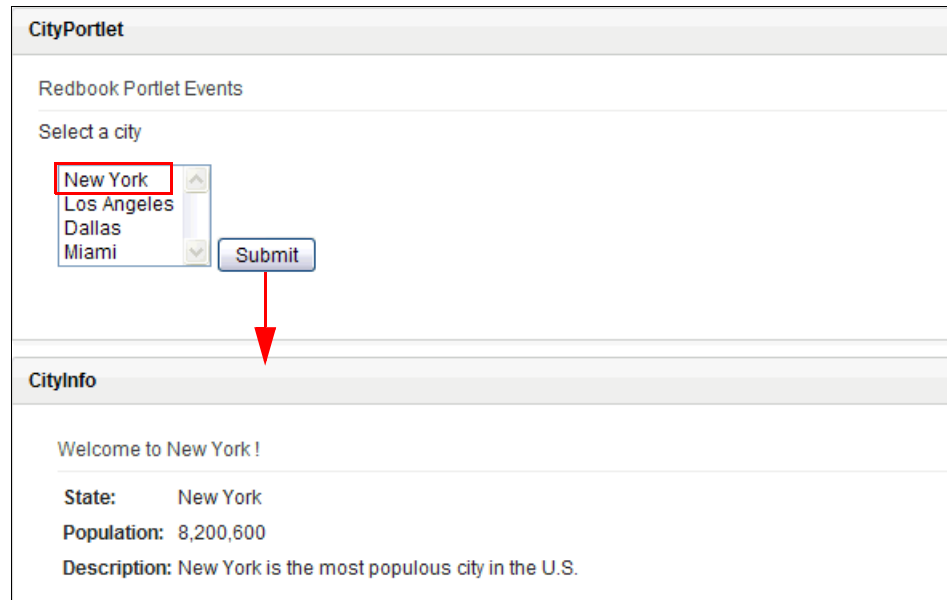


Figure 21-12 Application with event handling between portlets

### 21.4.2 Creating Ajax and Web 2.0 portlets

The new JSR 286 contains improvements to develop Ajax portlets. One of the new features is called *resource serving*.

#### Ajax using JSR 286 resource serving

With resource serving, portlets can serve resource requests in resource URLs. With WSRP 2.0, requests for resources, such as PDF documents, are addressed within the WSRP protocol. Formerly, an out-of-band connection was required for such resources, for example, HTTP. Resources are now portal context aware, because the context information is passed directly over the WSRP protocol.

Resource serving allows Ajax support inside the portlets, and the portlets have a way to return XML, JavaScript Object Notation (JSON), HTML fragments, or other content. Only the portlet that made the requisition is updated, not the entire page (Figure 21-13 on page 1159).

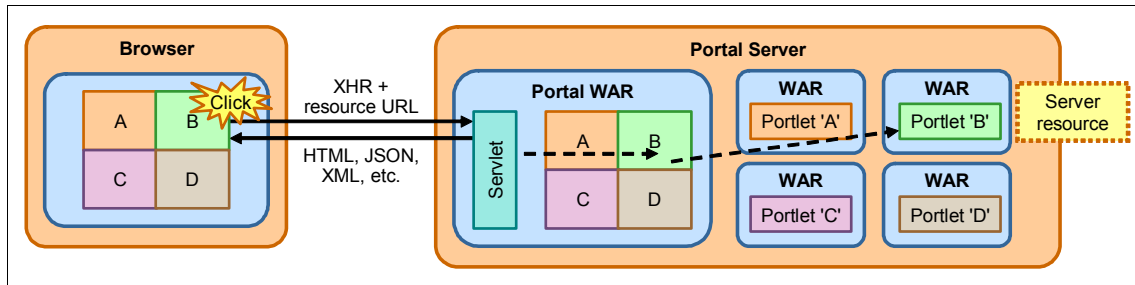


Figure 21-13 Portlet with Ajax support

A resource request is a new callback interface that is triggered by so-called *resource URLs*. To make a portlet become a resource serving portlet, you have to implement the `ResourceServingPortlet` interface and implement the `serveResource` method. The output for the `serveResource` method must have a separate content page.

### Resource serving example

Here, we adapt the previous portlet events example to add Ajax behavior. The `CityInfoPortlet` will have a button to retrieve a list of hotels about the current city (Figure 21-14).

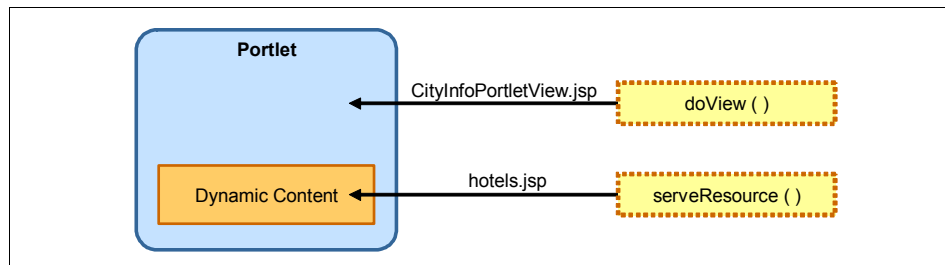


Figure 21-14 Resource serving portlet to add city hotels

### Project setup

We have two projects:

- ▶ `RAD80PortletEventAjaxEAR`: Enterprise application
- ▶ `RAD80PortletEventAjax`: Application with two portlets (based on JSR 286)

Import these projects from the

`C:\7835code\portal\RAD80PortletEventAJaxStart.zip` project archive file.

To import files before developing the other artifacts, follow these steps:

1. Select **File** → **Import**.

2. In the Import window, select **General** → **File System**. Click **Next**.
3. In the File system window, perform these tasks:
  - a. For From directory, click **Browse** and locate **C:\7835code\portal**.
  - b. In the right frame, select **hotel.jsp**.
  - c. For Into folder, click **Browse** and locate **RAD80PortletEvent/WebContent/\_CityInfo/jsp/html**.
  - d. Click **Finish**.
4. Repeat the import to place **HotelInfoBean.java** into the **com.ibm.rad80portlet.bean** package.
5. Repeat the import to place **HotelDB.java** into the **com.ibm.rad80portlet.db** package.

**Tip:** You can also drag a file from a Microsoft Windows Explorer folder directly to the appropriate folder or package in Rational Application Developer.

## Developing the Ajax code

Open the **CityInfoPortlet** class in the editor and perform these steps:

1. Select **Source** → **Organize Imports** (or press Ctrl+Shift+O) to resolve the import.
2. Right-click in the editor window and select **Source** → **Override/Implement Methods**. Select the **serveResource** method, and for the Insertion point, select **After processEvent(...)**. Click **OK**.
3. When the **serveResource** method is called, find the hotels of the selected city and set a parameter with the hotel list to the JSP file. Add the code to the **serveResource** method, as shown in Example 21-5.

*Example 21-5 CityInfoPortlet serveResource method*

---

```
public void serveResource(ResourceRequest request,
ResourceResponse response) throws PortletException,
IOException {
 String cityId = request.getParameter("cityId");
 request.setAttribute("hotels", HotelDB.getHotels(cityId));
 PortletRequestDispatcher rd = getPortletContext()
 .getRequestDispatcher(JSP_FOLDER +
 "/html/hotel.jsp");
 rd.include(request, response);
}
```

---

4. Select **Source** → **Organize Imports** (or press Ctrl+Shift+O) to resolve the import.
5. From the Palette, expand the Portlet drawer. Drag the **Serve Resource** palette item onto the portlet JSP file to add the portlet resource URL tag and `serveResource` method in portlet class.
6. In the Insert a portlet resource URL window, complete the following fields and click **Finish** (Figure 21-15):
  - a. For the Package prefix, type a prefix to the package.
  - b. For the Portlet Resource URL attributes, enter the resource ID and URL.

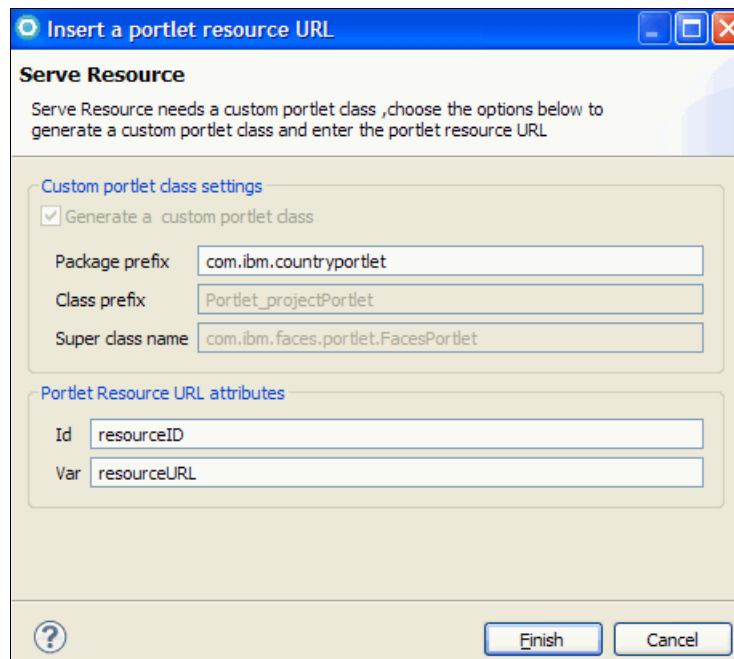


Figure 21-15 Portlet resource URL

7. Locate the `C:\7835code\portal\SnippetJavaScriptAjax.txt` JavaScript snippet file and copy its contents.
8. Paste the code into the `CityInfoPortletView.jsp` after the tags `<portlet-client-model:init>.....</portlet-client-model:init>`. This code has JavaScript functions that call Ajax functions.
9. Create a Hotels button with an event click that targets the resource URL and a `<div>` tag where the list of hotels will appear. Add the code to `CityInfoPortletView.jsp` after the `</table>` tag, as shown in Example 21-6 on page 1162.

*Example 21-6 Hotels button and div tag*

---

```
</table>

 <===== existing tag
<form>
<input type="button"
onclick="<portlet:namespace/>getHotels('${requestScope.resourceUrl}'
,
'hotels','${requestScope.info.cityId}')"
value="Hotels" >
</form>
</c:if> <===== existing tag
<div id="hotels"> </div>
```

---

### 21.4.3 Deploying and running the application

In the Enterprise Explorer, right-click **RAD80PortletEvent** → **Run As** → **Run on Server**. In the subsequent window, click **Finish** to publish the application to the WebSphere Portal server. After the application is deployed, Rational Application Developer opens a browser in the workbench.

**Connecting the portlets:** After deploying the application, connect the portlets. Follow the instructions in “Connecting the portlets” on page 1155.

#### Testing the application

To test the Ajax functionality, perform these steps:

1. In the first portlet, select **New York** from the list box and click **Submit**. The second portlet shows the information about New York.
2. Click **Hotels** to call an Ajax function that retrieves a list of hotels (Figure 21-16 on page 1163).

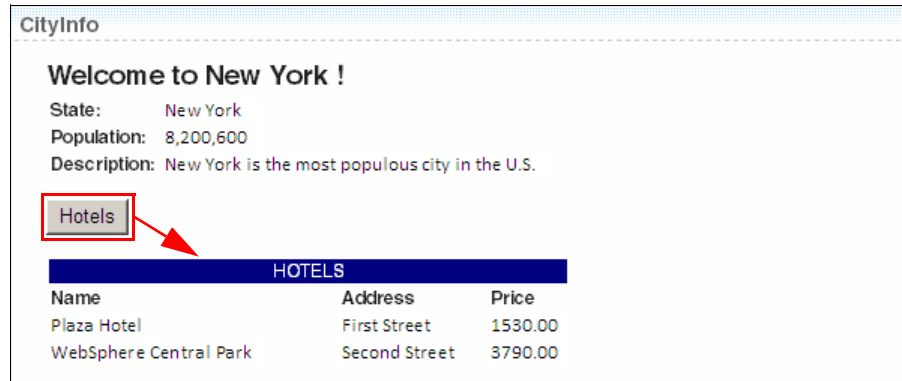


Figure 21-16 Portlet application with Ajax

#### 21.4.4 Creating a portal site with the Site Designing Portlet feature

The Site Designing Portlet is a web-based user interface that is used to create and manage the portal site. You can use this user interface to change the appearance and layout of your portal site. It is available to use after the product is installed and started with WebSphere Portal server.

You must first understand the composition of the portal site and how the pages are aggregated before creating your portal site. Site navigation uses portal's hierarchical structure information about various pages, labels, URLs, and so on to generate a navigation tree that is similar to the site map. The topmost node in the navigation hierarchy tree is the *content root*.

After you define a portal site with a content root, you can manage your site by adding pages and labels in your site navigation tree. You can plan and lay out your whole portal site and then create pages and fill them with portlets for the site. You can rename the pages, create easy to remember URLs, wire portlets inside them, and edit the page metadata. You can also search for a portlet, add it to a page, and also edit the existing site layout. The home page of the Site Designing portlet uses the portal theme, by default. This feature is supported with WebSphere Portal Server V6.1.5, and later. To populate your portal site structure, add a new page to a portal site:

1. Select a **WebSphere Portal Server** in the Server view to open the site navigation.
2. Right-click and select **Site Designing portlet**.
3. After the Site Designing portlet is launched in the browser, in the site navigation, select a page and click the drop-down arrow in the tree where you want to attach the new page (Figure 21-17 on page 1164).

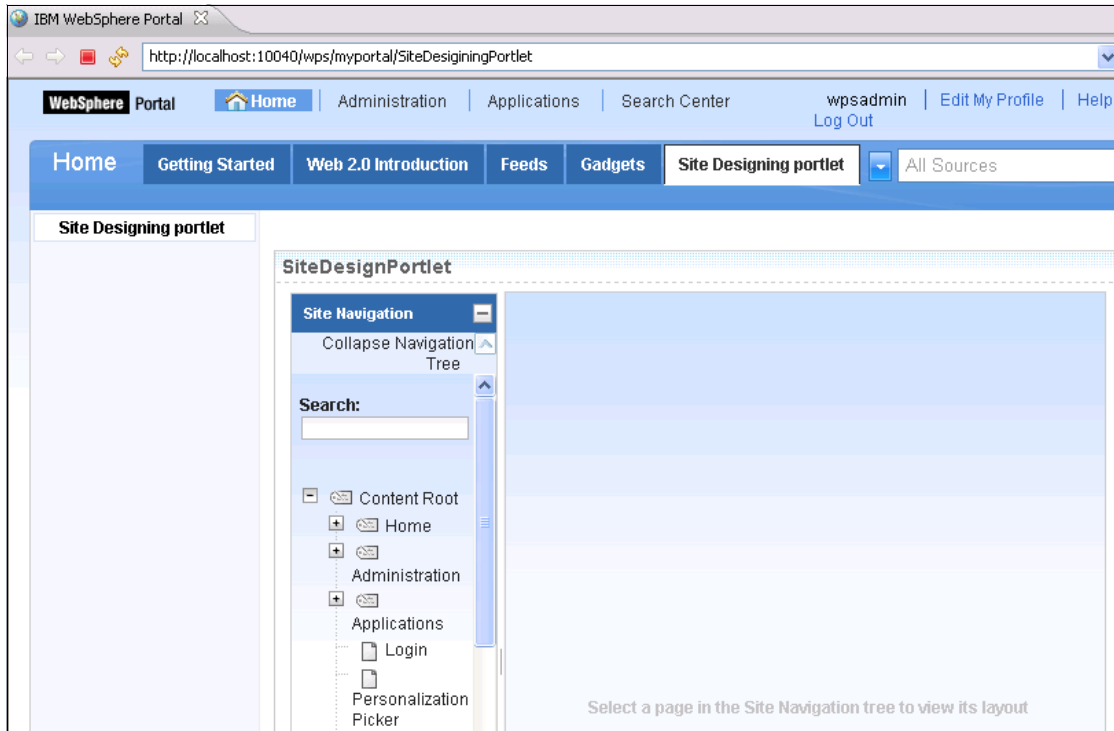


Figure 21-17 Site Designing portlet launched in a browser

4. Select **Create Page** and **Before**, **After**, or **As Child**, depending on where you want to put the new page in relation to the page that you selected.
5. In the Page Metadata window, edit the following metadata for the page and click **Submit** (Figure 21-18 on page 1165):
  - a. Friendly URL: Make sure that your URL for the page is easy to remember and consistent with the rest of the site.
  - b. Theme: Select the theme of the page from the drop-down list.
  - c. Page Cache Options: You can specify a cache scope that indicates whether cached content is public (shared) or per user. A portlet that caches its output can change its content immediately to provide additional output. You can indicate how long, in seconds, to cache portlet output. The page expiration time enables the portlet developer to inform the container when to invalidate the current cache entry for the portlet and therefore when to refresh the portlet's content.
  - d. Page Properties:
    - This page can be added to the user's favorites or bookmark list.



- Other pages can share the contents of this page.

Figure 21-18 Site Designing portlet: Page Metadata

6. Click **Submit**.

## Adding portlets to the site page

Follow these steps to add each portlet to the site page:

1. Select a page in the site navigation tree where you want to add a portlet.
2. Select the appropriate page layout by clicking one of the predefined page layouts inside the site layout (Figure 21-19 on page 1166).

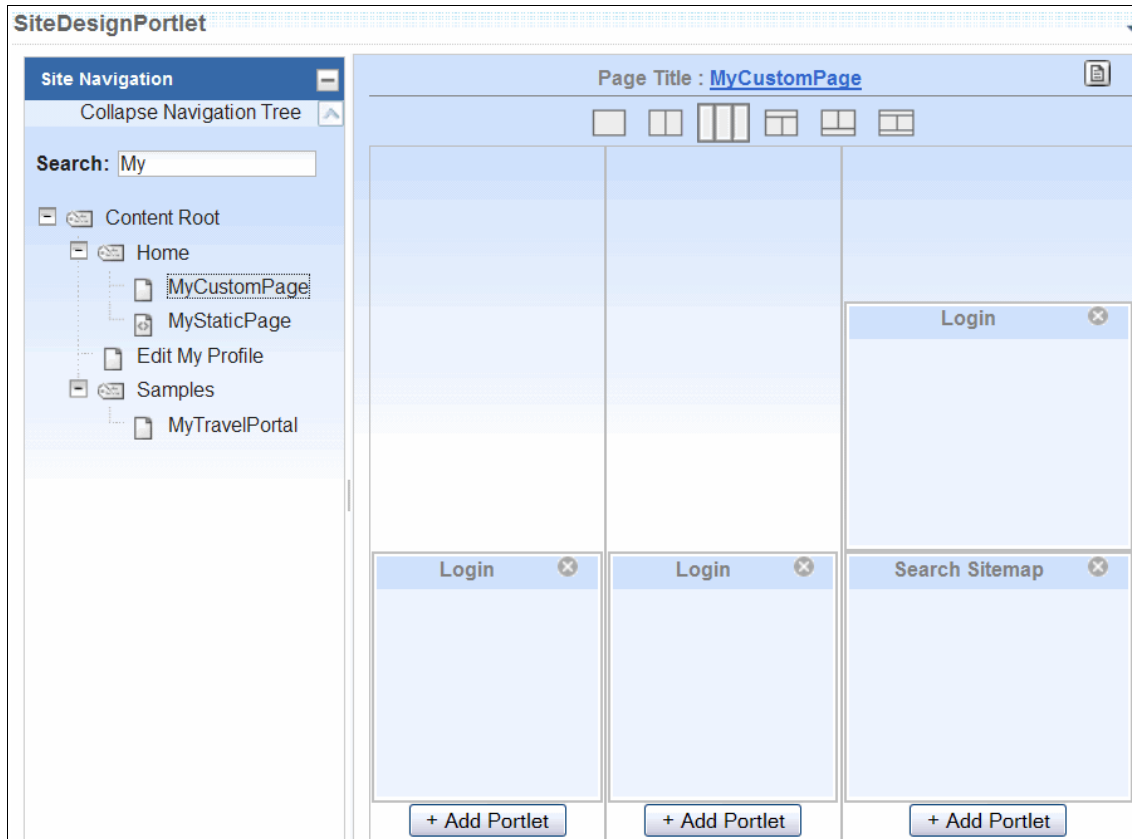



Figure 21-19 Site Designing portlet: Page layout

3. Select the row or column of the page where you want to add a portlet and click **Add Portlet**.
4. In the Universal Catalog window, select the **Portlet** check box to limit the list of page components.
5. Search for the portlet that you want to add, preview it, and click **Add**.
6. Click **OK** to see the page with the portlet that you added.

### Creating a wire

Perform these steps to create a wire:

1. Select a WebSphere in the Server view to open the site navigation.
2. Right-click and select **Site Designing Portlet**.
3. Select a page that you want to wire in the site navigation tree.

4. Click the **Create Wire**  icon in the upper-right corner of the window.
5. In the Portlet Wiring Tool wizard, select the Target page, which, by default, is set to the current page.
6. The wiring tool scans both the current page and the target page to list all of the source portlets (portlets that can publish events) and target portlets (portlets that can process events). It also displays the existing wires, if any, between the source and target events. See Figure 21-20.

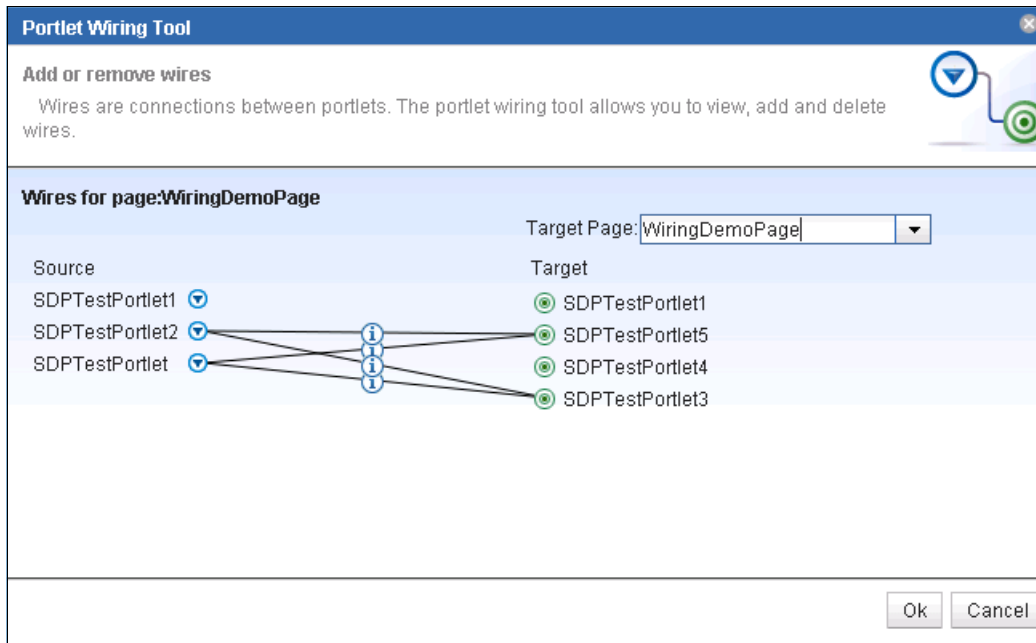



Figure 21-20 Site Designing portlet: Portlet Wiring Tool

7. Hover over a source portlet to list all of the events that the portlet publishes. Select an event and click **Add** beside it to create possible wires from the current source to all target portlets that declare to receive the same event.
8. Click the exclamation mark () icon on the wire to remove a wire.
9. Click **OK**.

### 21.4.5 Developing Dojo-based inter-portlet communication

The Dojo-based portlet events provide a powerful and flexible publish/subscribe (pub/sub) mechanism for communication between portlets without a page refresh. This communication method is based on directed communication links that pass information from a source portlet to a target portlet. When data is sent

across a link, the target portlet is invoked to process the received information and performs arbitrary updates.

You can configure the Dojo capabilities for a project from the Properties view. The Palette view consists of Dojo drawers from where you can drag Dojo widgets onto the portlet JSP file. The Dojo Event Publisher and Dojo Event Subscriber palette items are available in the Portlet drawer. These items are used to add publisher and subscriber events for a dojo widget to achieve inter-portlet or intra-portlet communication through JavaScript events.

This task displays the Dojo inter-portlet communication mechanism. With this task, you can verify that the text entered in one portlet is also displayed on another portlet without any page refresh. Follow the steps to run a portlet project that has two portlets interacting with each other:

1. Create a portlet project with Dojo on WebSphere Portal with facet enabled.
2. Drag a **Dojo TextBox** widget onto the portlet JSP file (Figure 21-21).

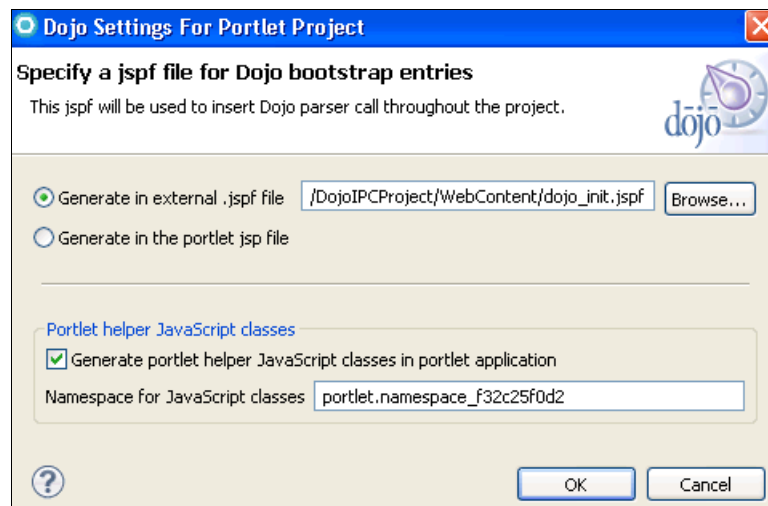


Figure 21-21 Dojo settings for portlet project

3. In the Dojo Settings For Portlet Project window, specify the JavaScript namespace for the Dojo classes as **com.amazing.portlet** and click **Finish**.
4. Drag a **Button Dojo** widget from the Dojo Form Widgets drawer.
5. Drag the **Dojo Event Publisher** palette item from the portlet drawer onto the **Button Dojo** widget.
6. In the Insert Dojo Event Publisher window, complete the following field for an event publisher (Figure 21-22 on page 1170):

- a. Object ID: Specify how you want to connect the publish calls. The `dojo.byId` function returns the Document Object Model (DOM) node, whereas the `dijit.byId` function gives reference to the `Dijit` widget node.
- b. Event name: Specify the name of the event for the widget. After this event is completed, the `dojo.publish()` method is called.
- c. Publish function name: Specify the publisher function name. For example, name the function as `F1()`. This function is used to construct a message for publishing and is called before the event is triggered.
- d. Topic name: Specify the topic name to which you want this publisher event to publish the message. A topic acts as a broker to connect a publisher to various subscribers. Enter the topic name as `dojo.ipc`. The topic name must be the same for the event publisher and the subscriber for the event to take place.

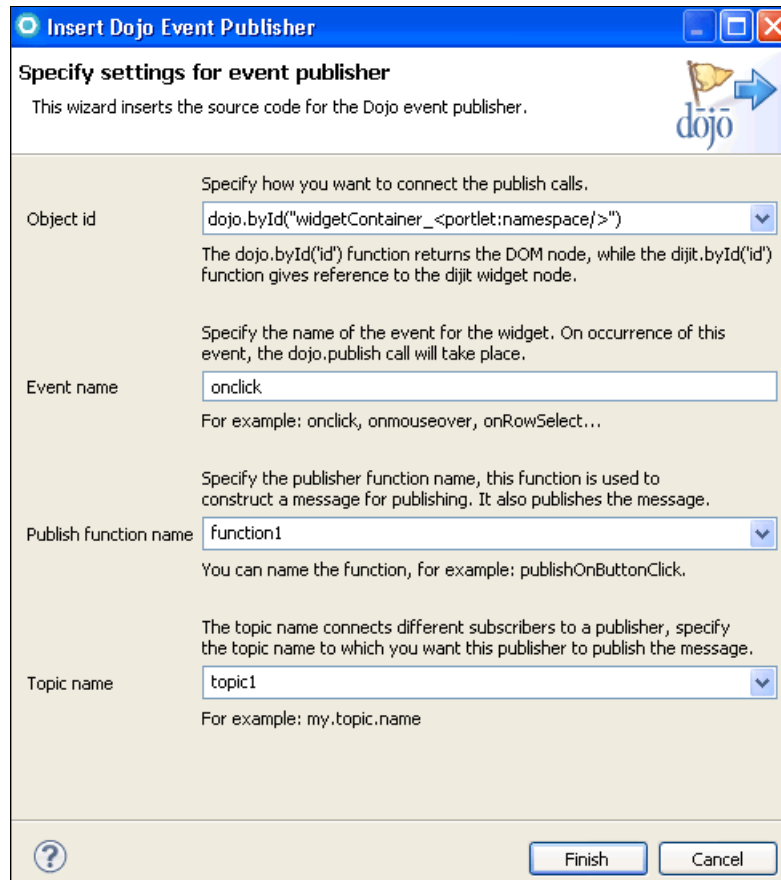


Figure 21-22 Specifying settings for Dojo event publisher

7. Click **Finish**.
8. Open the **[PortletView1].js** file in the portletHelper folder under the /project/WebContent/js/com/ibm/amazing directory.
9. To the function F1() that was auto-generated, add the logic `this.portlet.byId("textBox").value = args;` in the .js file and save the file.
10. Create another portlet in the same portlet project.
11. Drag a **Dojo TextBox** widget onto the portlet JSP file of the newly created portlet.
12. Drag the **Dojo Event Subscriber** palette item from the portlet drawer onto the portlet JSP file.

13. In the Insert Dojo Event Subscriber window, complete the following fields for an event subscriber (Figure 21-23):
- Topic name: Specify the topic name to which you want this portlet to subscribe. Select the already listed topic **dojo.ipc**. The topic name connects separate subscribers to a publisher. This topic name must be the same topic name as that of the event publishing dojo widget.
  - Subscriber function name: Specify the subscriber function name that will process all the messages published on the specified topic. This function is called when the event is received by the target portlet.
  - Number of Arguments: Specify the number of arguments that the subscriber function has. These arguments are equal to the size of the message array that is published by the publisher on the specified topic. Select 0 if this function does not have any arguments. Select 1 for one argument, 2 for two arguments, and so on.

**Insert Dojo Event Subscriber**

**Specify settings for event subscriber**

The wizard inserts the source code for the Dojo event subscriber.

The topics name connects different subscribers to a publisher, specify the topic name to which you want this subscriber to subscribe.

Topic name:

For example: my.topic.name

Specify the function name that processes all the messages published on the topic specified above.

Subscriber function name:

Subscriber function name can be of your choice, for example: handlerToInterestingEvent.

Specify the number of arguments that the subscriber function will have. These arguments are equal to the size of the message array published by the publisher on the specified topic.

Number of Arguments:

Select 0 if this function does not have an argument. Select 1 for one argument, 2 for two arguments, and so on.

Figure 21-23 Dojo Event Subscriber

14. Click **Finish**.

15. Open the file **[PortletView2].js** under the **portletHelper** folder.
16. To the generated function `F2()` in the `.js` file, add the logic `this.portlet.byId("textBox").value = args;` in the `.js` file and save the file. Run the project on a WebSphere Portal server.
17. Enter text in the text box and click the button.
18. Verify that the text that you entered is also displayed on another portlet without any page refresh.

## 21.4.6 Consuming RPC adapter services

The Remote Procedure Call (RPC) adapter service for portal provides a mechanism to consume server-side Java objects, such as EJB session bean methods or plain old Java object (POJO) services, to AJAX-based user interfaces.

The portal tools provide support for the RPC adapter service by enabling you to consume classes and methods as an RPC adapter service. To use the RPC adapter tooling, you need to ensure that your project has the Web 2.0 server-side technologies project facet enabled. Complete these steps to consume an RPC adapter service:

1. Create a portlet project with a V6.1 or later target run time, and with the RPC adapter and Dojo on WebSphere Portal Facet installed.
2. Open the portlet project JSP file and select the **Page Data** view.
3. Right-click **RPC Adapter services for Portal** and select **New** → **RPC Adapter Service**.
4. The Expose RPC Adapter Service wizard opens. Select **POJO Class** and click **Browse**.
5. The Select a Java class to expose window opens. In the Select entries field, type the class name to list matching classes.
6. Select a class and click **OK**.
7. The selected POJO class lists the class methods.
8. Select the required methods and click **Finish**.
9. Refresh the **Page Data** view. The RPC methods created are listed under the service name in the RPC Adapter Services for Portal node.
10. Expand the service node under the RPC Adapter Services for Portal node.
11. Select a method and drag it on the **Design** view of the portlet JSP file to generate Dojo data and JavaScript.



12. Under the JavaScript group, select **Generate into script element in this page**.
13. Enter the required data, keep the other options as is, and click **Finish** to generate the code.
14. Publish the portlet project and verify that the Dojo data is rendered properly.

Complete these steps to generate Dojo data inside the portlet JSP file and its corresponding JavaScript in an external JavaScript file:

1. Expand the service node under the RPC Adapter Services for Portal node.
2. Select a method and drag it onto the **Design** view of the portlet JSP file to generate Dojo data and JavaScript.
3. Under the JavaScript group, select **Generate into .js file**.
4. Click **Browse** and select a **.js** file.
5. Enter the required data, keep the other options as is, and click **Finish** to generate the code.
6. Publish the portlet project and verify that the Dojo data is rendered properly.

## 21.4.7 Creating iWidget projects

An *iWidget* is an XML file that contains markup that is rendered and can be supported by JavaScript files for dynamic client-side scripting and CSS files for styling the markup. JSP, HTML, or HTML fragments can also contain markups, which can also be written in Ajax.

You can create an iWidget either in a portlet or in an iWidget project. When you complete this task, you create an iWidget XML file using a template. You can use the Edit mode to display a markup alternative to the View mode. Follow the steps to create an iWidget in an iWidget project:

1. Select **File** → **New** → **Project** → **Web** → **iWidgets Project**.
2. In the New iWidgets Project wizard, select the option **Web and Java EE technologies (Ajax, HTML, CSS, JSP, Servlet etc.)**. Click **Next**.
3. Enter a project name and accept the default directory for your iWidget.
4. Select a target run time. WebSphere Portal Server Version 6.1.5 and later are supported target run times for an iWidget project. This server is the server to which you will deploy your widget.
5. By default, the wizard associates your project with a project EAR. Click **Next**.
6. In the New iWidgets Project wizard page, specify a name for the iWidget.

7. Select the iWidget type from the drop-down list (Figure 21-24). The templates provide you with a working copy of an iWidget XML file and a JavaScript file with stubbed functions. The following three templates are available:
  - Simple iWidget: Hello World iWidget with no JavaScript
  - Event publisher iWidget: iWidget with an event that sends data
  - Event subscriber iWidget: iWidget with an event that receives data

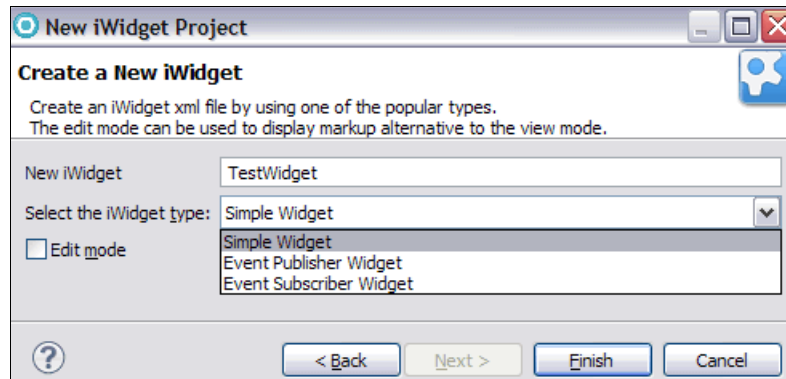


Figure 21-24 Types of iWidgets

8. Select the **Edit Mode** check box to add an Edit mode to the iWidget. A *mode* is a section of markup that an iWidget renders at run time. When an iWidget is running, you can switch between modes to display separate markups.
9. Click **Finish** to create an iWidget in an iWidget project.

If you want to add more iWidgets to the project, follow these steps:

1. Open the Enterprise Explorer view, right-click the iWidget project created, and select **New** → **iWidget**.

**Creating an iWidget in a portlet project:** To create an iWidget in a portlet project, right-click the portlet project in your workspace and select **New** → **Other**. Select **iWidget** from the Web folder in the New window.

2. In the New iWidget wizard, type the widget name.
3. Select the iWidget type from the drop-down list and then click **Finish**. The templates provide you with a working copy of an iWidget XML file and a JavaScript file with stubbed functions. Four templates are available during a new iWidget creation:
  - Simple iWidget: Hello World iWidget with no JavaScript.
  - Event publisher iWidget: iWidget with an event that sends data.

- Event subscriber iWidget: iWidget with an event that receives data.
- iWidget with JSP content: iWidget with its markup defined in an external JSP file. The JSP file is inserted into the iWidget definition file by the JavaScript. (This option is available only while creating a new iWidget in an existing iWidget project.)

## 21.4.8 JPA tooling support for portlet projects

The JPA API simplifies object relational mapping and data persistence tasks. Using the JPA tooling support, you can create the database connection and JPA entities that are object representations of database tables. This toolkit also helps you to create JPA Manager Beans and to expose methods to manage and persist JPA entities. The JPA query builder automates the generation of JPA query methods.

With the JPA tools in the product, you can use wizards to create and automatically initialize mappings. You can create new database tables from existing entity classes or new entity beans from existing database tables. You can also use the tools to create mappings between existing database tables and entity beans, where names or other attributes differ.

In this example, the JPA-enabled portlet application uses *Derby 10.1- Embedded JDBC Driver Default* at the back end and JPA for database connectivity. Using the Portal Toolkit and JPA tools support, you can create an end-to-end JPA portlet application quickly. The toolkit simplifies the database management, query generation, and code generation tasks.

Import these projects from the C:\7835code\portal\HealthCareDataBase.zip project archive file. The following target run times for JPA portlet projects are supported:

- ▶ IBM WebSphere Portal V6.1 on WebSphere Application Server V6.1 with EJB3.0 front-end programming interface (FEP)
- ▶ IBM WebSphere Portal V6.1 on IBM WebSphere Application Server V7.0
- ▶ IBM WebSphere Portal V7.0

### Creating a new portlet project

To create a new portlet project, perform the following steps:

1. Select **File** → **New** → **Portlet Project**.
2. Enter CountryPortlet as the Portlet Project Name.
3. Select **WebSphere Portal 6.1 (WebSphere Portal 6.1 on WebSphere Application Server 6.1 with EJB 3.0 Fep)**.

4. Click **Modify** beside Configuration, select **JSR 286** as the API, and select **Faces** as the Portlet Type from the Portlet Project Configuration dialog window. Click **OK**.
5. Click **Finish**. When prompted, switch to the **Web** perspective.

### **Establishing the database connection**

Follow these steps:

1. Copy and extract the attached database file HealthCareDatabase.zip to your local directory C:\db.
2. Open the **Database Development** perspective.
3. Right-click **Database Connections** in the Data Source Explorer view, and select **New**.
4. When the New Connection wizard appears, select **Derby** as the database manager.
5. Select **Derby 10.1- Embedded JDBC Driver Default** as the Java Database Connectivity (JDBC) driver.
6. Browse and select **C:\db\HealthCareDatabase** in the Database Location field (Figure 21-25 on page 1177).

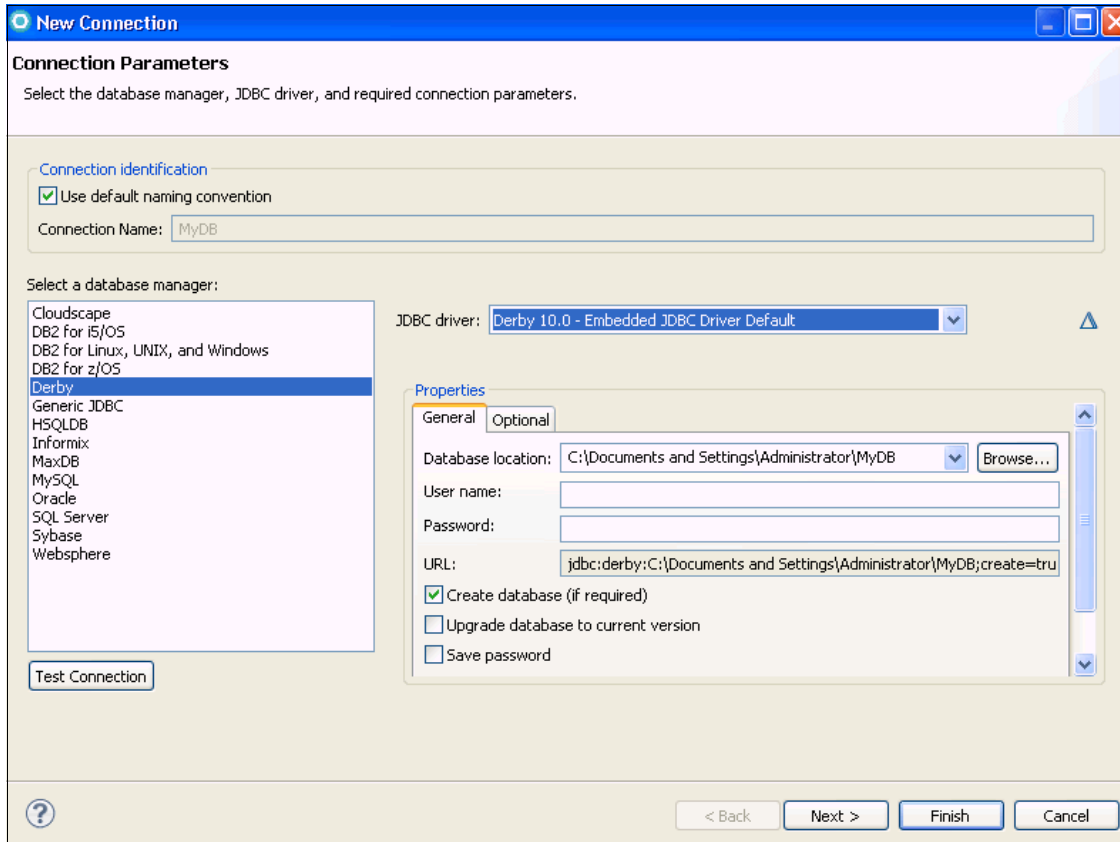


Figure 21-25 Derby database connection

7. Click **Finish**. Ensure that the connection works.

## Creating JPA entities and Manager Beans

Follow these steps:

1. In the Enterprise Explorer view, select the **CountryPortlet** project.
2. Right-click and select **JPA Tools** → **Configure JPA Entities**. The Configure JPA Entities window installs the JPA 1.0 facet if it is not installed.
3. Click **Create New JPA Entities** to open the Generate Custom Entities wizard. Complete these tasks:
  - a. Select the **HealthCareDatabase** database and the **APP** schema (Figure 21-26 on page 1178).

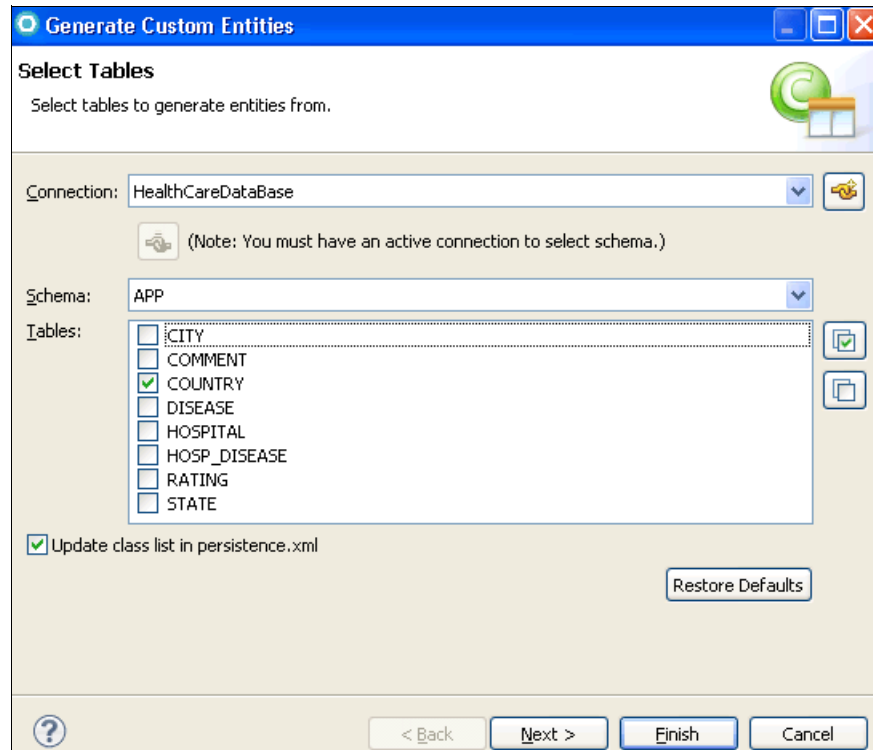


Figure 21-26 Database entities

- b. Select the **COUNTRY** table and click **Finish**.
4. On the Configure JPA Entities window, select the **Country** entity as a primary key and click **Finish** to create the country entity.

To configure this entity, select the **Country Entity** on the Configure JPA Entities wizard.

## Creating JPA Manager Beans

Complete these steps to create the JPA Manager Bean (CountryManager bean) for the Country Entity:

1. Right-click the Portlet project and select **JPA Tools** → **Add JPA Manager beans**.
2. Select the **Country** entity from the list of available entities.
3. Click **Next** and click **Launch Entity Configuration Wizard**.
4. To add the getCountry query to the CountryManager bean on the Configure JPA Entities window, select the **Named Queries** task and click **Add**.

5. Accept the default values on the Add Named Query window and click **OK** (Figure 21-27).

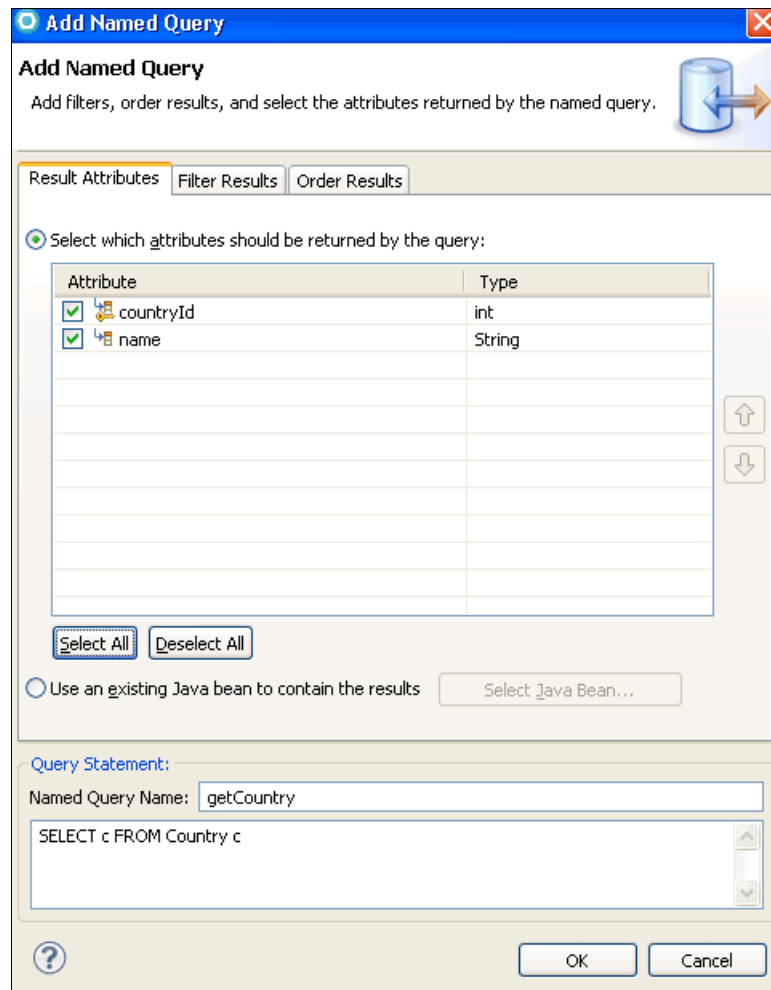


Figure 21-27 Adding the named query

6. Click **Finish**.

The Country JPA entity and CountryManager bean are created. Verify that the JPA entity is created in the `src/entities` directory and that the manager bean is created in the `src/entities/controller` directory.

## JPA data consumption

Follow these steps:

1. Open the **CountryPortletView.jsp** in the Page Designer view in the Design view.
2. From the Page Data view, expand **JPA** → **JPA Manager Beans** → **entities.controller.CountryManager** and select the **getCountry** query (Figure 21-28).

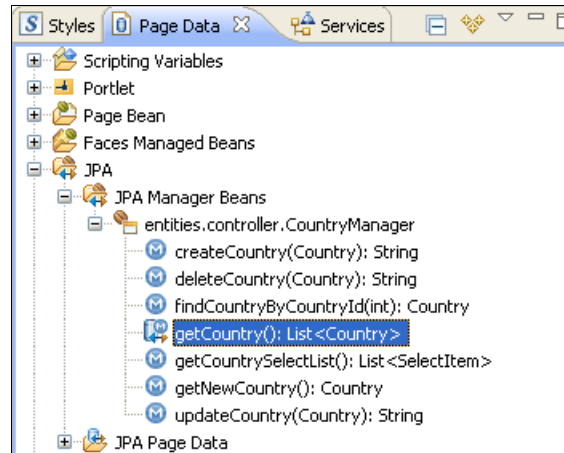


Figure 21-28 Data query consumed by JPA

3. Drag the **getCountry** query from the Page Data view onto the **CountryPortletView.jsp** file.
4. Select **Retrieve a list of data** on the Add JPA data to page window and click **Next**.
5. Set the order of the fields and click **Finish** (Figure 21-29 on page 1181).



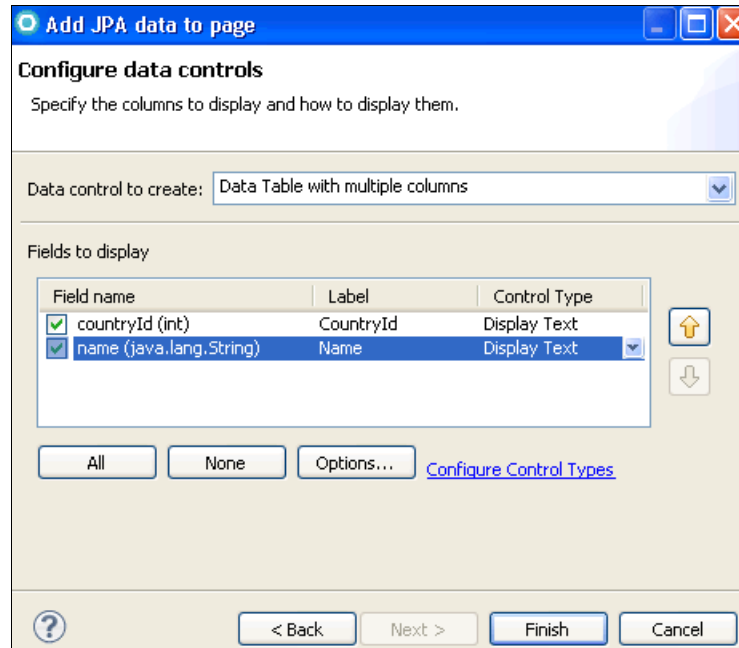


Figure 21-29 Added JPA data fields

The code to display the Country list is generated in the `CountryPortletView.jsp` file.

6. Right-click **CountryPortlet** and select **Run as** → **Run on Server**.
7. Select **WebSphere Portal 6.1 on WebSphere Application Server v7 with EJB 3.0 Fep** and click **Finish**. The list of countries is displayed on the browser.

## 21.5 More information

For more information about portal technology, see the following resources:

- ▶ The following IBM Redbooks publications cover the Portlet Application Development for the older versions of WebSphere Portal:
  - *Building Composite Applications*, SG24-7367
  - *IBM Rational Application Developer V6 Portlet Application Development and Portal Tools*, SG24-6681
  - *IBM WebSphere Portal V5 A Guide for Portlet Application Development*, SG24-6076

– *Portal Application Design and Development Guidelines*, REDP-3829

- ▶ WebSphere Portal 6.1 Information Center:  
<http://www.ibm.com/developerworks/websphere/zones/portal/proddoc.html#v61infocenters>
- ▶ WebSphere Portal zone  
<http://www.ibm.com/developerworks/websphere/zones/portal/>
- ▶ WebSphere Portal family wiki  
<http://www-10.lotus.com/ldd/portalwiki.nsf>
- ▶ What's new in the *Java Portlet Specification V2.0 (JSR 286)*  
[http://www.ibm.com/developerworks/websphere/library/techarticles/0803\\_hepper/0803\\_hepper.html](http://www.ibm.com/developerworks/websphere/library/techarticles/0803_hepper/0803_hepper.html)
- ▶ What's new in WebSphere Portal V6.1: JSR 286 features  
[http://www.ibm.com/developerworks/websphere/library/techarticles/0809\\_hepper/0809\\_hepper.html](http://www.ibm.com/developerworks/websphere/library/techarticles/0809_hepper/0809_hepper.html)
- ▶ What's new in IBM Rational Application Developer V7.5 Portal Toolkit  
<http://www.ibm.com/developerworks/rational/library/09/rationalapplicationdeveloperportaltoolkit1/>
- ▶ WebSphere Portal Update Installer  
<http://www-1.ibm.com/support/docview.wss?rs=688&uid=swg24006942>
- ▶ WebSphere Application Server V6.0.2 Fix Pack 17 (WebSphere Application Server V6.0.2.17) for Microsoft Windows platforms  
<http://www-1.ibm.com/support/docview.wss?rs=180&uid=swg24014309>
- ▶ Update Installer for WebSphere Application Server V6.0 releases  
<http://www-1.ibm.com/support/docview.wss?rs=180&uid=swg24008401>
- ▶ Troubleshooting: Rational Application Developer V7.x cannot sense the proper state of WebSphere Portal Server V6.x  
[http://www-1.ibm.com/support/docview.wss?rs=2042&context=SSRTLW&context=SSJM4G&context=SSCG7C&dc=DB520&uid=swg21258582&loc=en\\_US&cs=utf-8&lang=en](http://www-1.ibm.com/support/docview.wss?rs=2042&context=SSRTLW&context=SSJM4G&context=SSCG7C&dc=DB520&uid=swg21258582&loc=en_US&cs=utf-8&lang=en)



## Developing Lotus iWidgets

In this chapter, we introduce one particular type of widgets supported by Rational Application Developer: *iWidgets*. Rational Application Developer uses the IBM iWidget Specification V2.0 as the basis for its iWidget development.

A sample Stock Quote iWidget is used in this chapter to cover the various topics relating to iWidget development in Rational Application Developer.

This chapter covers the following topics:

- ▶ Introduction to iWidgets
- ▶ Developing iWidgets in Rational Application Developer
- ▶ Working with the sample iWidget application
- ▶ Additional resources

## 22.1 Introduction to iWidgets

A *widget* is a small, stand-alone application that can be run on the desktop or within a web page. Widgets are intended to be simple, single purpose applications that can be combined or “mashed together” to provide added functionality and content for the users. Widgets are particularly useful when creating portal sites. Portal web pages are designed to allow users to pick and choose the content they want to view on a single web page for convenient access.

Widgets, by definition, are designed to be reusable components. They fit perfectly within the current information technology objectives of creating reusable, web-based components that encapsulate and isolate the information that they provide. The following examples are widgets:

- ▶ Stock ticker
- ▶ Sports ticker
- ▶ Local weather
- ▶ Social networking
- ▶ World clock
- ▶ Games
- ▶ Advertising

*iWidgets* are widgets that are based on the IBM iWidget Specification. The *IBM iWidget Specification* is a framework that defines characteristics of HTML markup, metadata formats, and JavaScript services for enabling the aggregation of iWidget components into a single Web application. Rational Application Developer supports the iWidget specification V2.0. There are many other competing standards for widget development and deployment.

In Rational Application Developer, iWidgets are developed as part of a static web project, dynamic web project, or portal project depending on the requirements of your application and the intended web server to which it will be deployed. Typically, the dynamic web project is used.

iWidgets are configured in an XML file. The XML file contains several parts that describe the widget, including content, events, event descriptions, itemsets, items, and resources.

### 22.1.1 Content

You can configure widgets with four distinct modes:

- ▶ View: Contents displayed during the normal running state of a widget
- ▶ Edit: Contents displayed during the edit/configure mode of a widget

- ▶ Help: Contents displayed to provide help to the user by the widget
- ▶ Print: Contents displayed when the user wants to print the widget

iWidgets are able to display separate content based on the selected mode, which allows you to configure various views for your widget. It is also possible to create custom views if the standard views are insufficient for your requirements. The content is basically an HTML fragment that is displayed inside the widget display area (Figure 22-1).

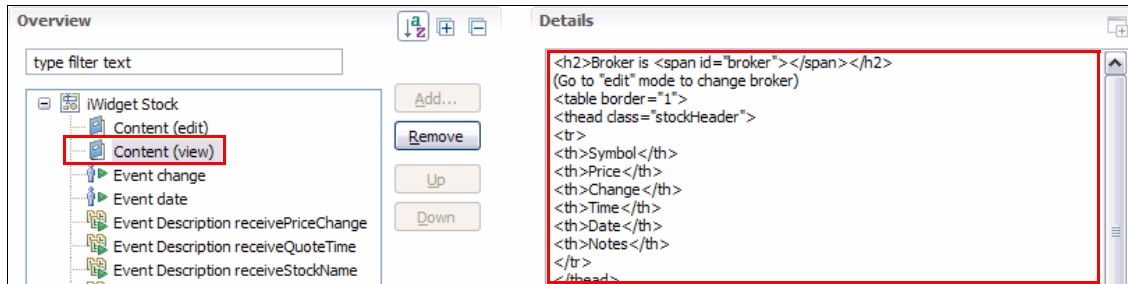


Figure 22-1 iWidget Editor: Content (view) Details

## 22.1.2 Events and event descriptions

*Events* are used to exchange information between the separate content views of the widget, as well as between other widgets. As with other web technologies, events are useful for providing data to your widget and changing state.

Event descriptions are the payload of the event. They describe the type of data that is being exchanged and descriptive text that can be displayed to the user. Events are associated with event descriptions so that when an event is triggered, the widget knows the data that will be passed with it.

## 22.1.3 Itemsets and items

An *itemset* is a simple definition of the data that is used by the widget. It provides an abstraction of a data store for the data. More complex data stores can be configured, but the itemset is the simplest implementation available.

*Items* detail the individual attributes of the itemset. Items can be the stock price, company symbol, and time last updated. Item descriptions also maintain the data type of the attribute and descriptive text that can be displayed to the user.

## 22.1.4 Resources

*iWidget resources* allow you to include external source files into the iWidget. This feature allows you to set up Cascading Style Sheet (CSS) files, JavaScript files, and other file resources that are used by the iWidget. Providing for separate files supports good coding standards, maintainability, and team development.

## 22.2 Developing iWidgets in Rational Application Developer

Rational Application Developer provides several samples and tutorials relating to widgets.

### 22.2.1 Accessing the tutorials and samples

You can access the iWidget sample by following these steps:

1. Select **Help** → **Help Contents**.
2. In the Help: Rational Application Developer window, expand **Samples** → **Web** → **iWidget samples**.
3. Select the sample that you want to install:
  - Stock widget
  - Color switch with button widget
  - Color switch with timer widget

A sample for creating an iWidget portlet is also available by following these steps:

1. Select **Help** → **Help Contents**.
2. In the Help: Rational Application Developer window, expand **Samples** → **Portlet** → **Web 2.0 portlets** → **iWidget**.

You can access the iWidget tutorial by following these steps:

1. Select **Help** → **Help Contents**.
2. In the Help: Rational Application Developer window, expand **Tutorials** → **Web** → **Create an iWidget**.

The tutorial takes approximately 60 minutes to finish and explores creating the project, creating the iWidget, modifying the iWidget xml, configuring test events, and deploying the iWidget.

## 22.2.2 Configuring Rational Application Developer for iWidget development tools

Install the iWidget development tools with the IBM Installation Manager if you have not enabled the functionality before. The iWidget development tools are included, by default, in Rational Application Developer, and you can enable them by selecting the feature called **Web Development Tools** → **Ajax, Dojo Toolkit and HTML development tools**.

Create an AJAX Test Server if one is not configured yet. An AJAX Test Server is created, by default, in Rational Application Developer. For more information about the AJAX Test Server, see 23.10, “AJAX Test Server” on page 1271.

## 22.3 Working with the sample iWidget application

We use the Stock widget for our example. This widget displays the company name, stock quote, price change, and the broker name. The Stock widget demonstrates how you can create an iWidget application that switches modes, consumes events, and generates events. After the sample is imported into the workspace, you can run the example on the AJAX Test Server.

### 22.3.1 Preparing the sample iWidget application

The first step is to load the sample application from the Help feature. To access the Stock widget sample, click **Help** → **Help Contents**. Click **Samples**. Then expand **Web** and **iWidget** and select **Stock widget** (Figure 22-2 on page 1188). Here, you can import the sample into the workspace.

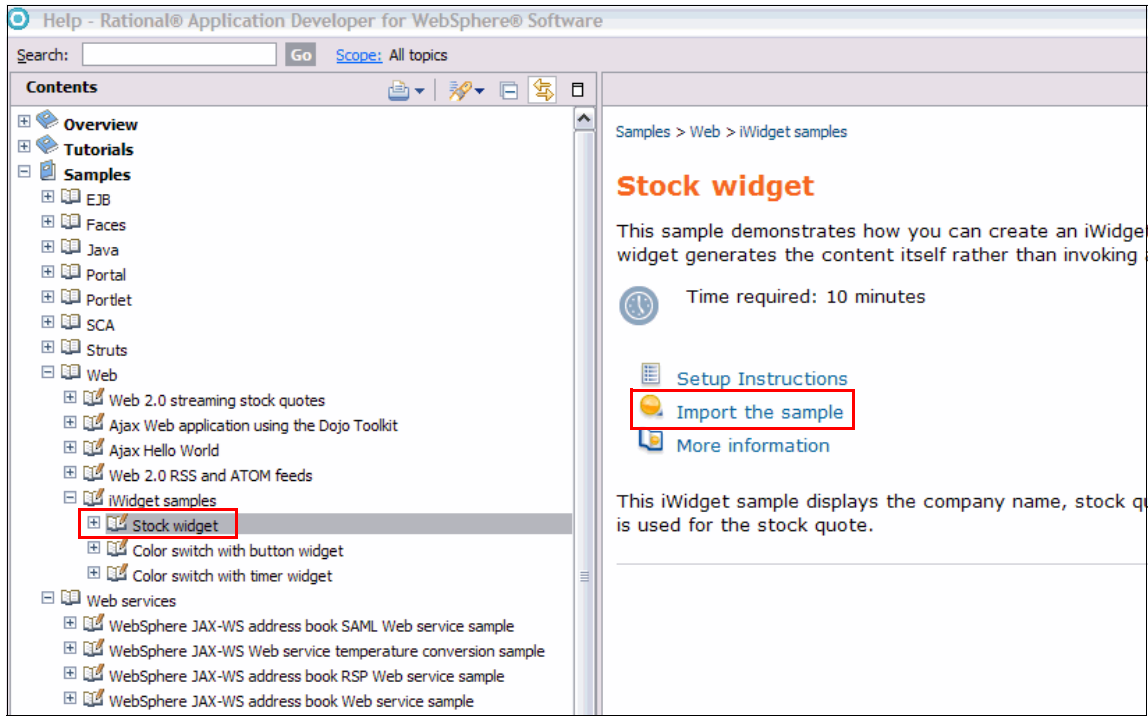


Figure 22-2 Loading the iWidget sample: Stock widget

This step opens the Import Project wizard window. Accept the default values and click **Finish** (Figure 22-3 on page 1189).



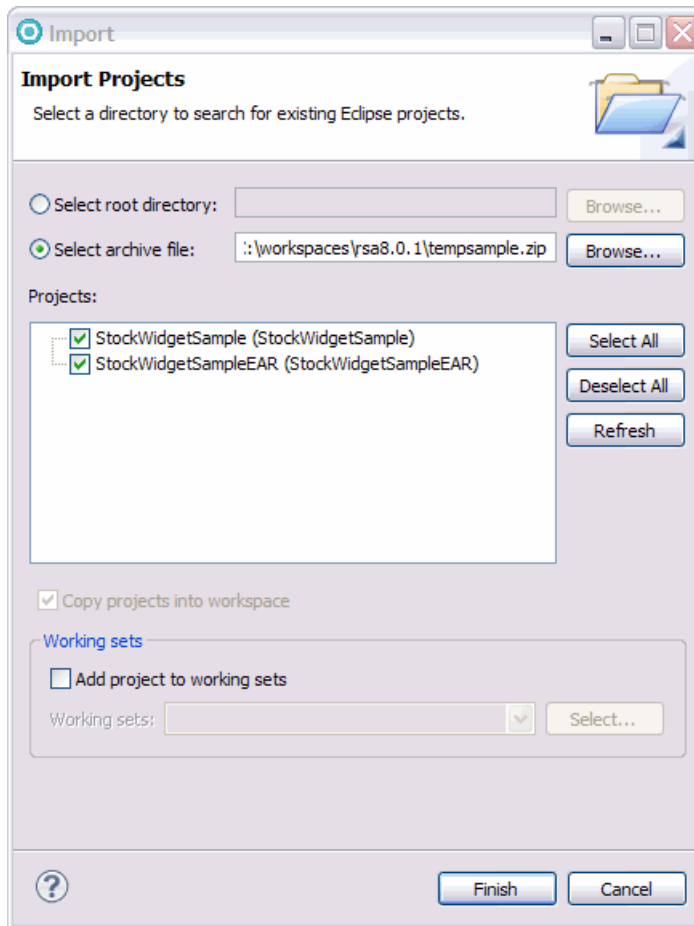


Figure 22-3 Importing the sample Stock widget project

## 22.3.2 Developing the sample iWidget application

The Stock iWidget sample is a complete working solution. There are no necessary modifications to get it to run on the AJAX Test Server. In this example, we add a new stock quote to the iWidget to demonstrate the functionality of the new iWidget Editor. We use the iWidget Editor to modify the iWidget definition file, `stock.xml`. See Figure 22-4 on page 1190. There are three additional files that are utilized by the Stock iWidget sample:

- ▶ `stock.css`: Style sheet definitions for the HTML fragments
- ▶ `stock.js`: JavaScript functions supporting the iWidget
- ▶ `stockData.js`: The dojo script that defines the data model that is used by the iWidget

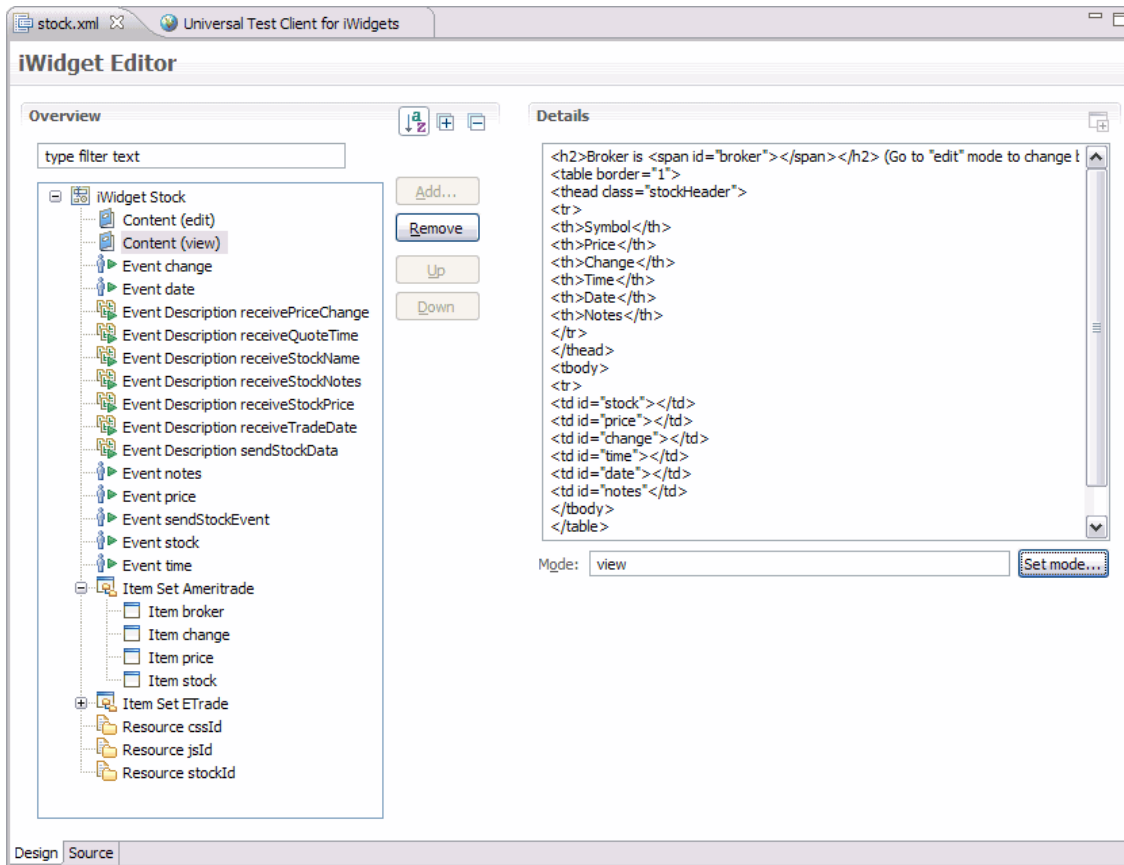


Figure 22-4 iWidget Editor

You can add new sections of the iWidget by right-clicking the top level of the tree and clicking **Add** (Figure 22-5). You can also edit the XML file directly when you click the **Source** tab.

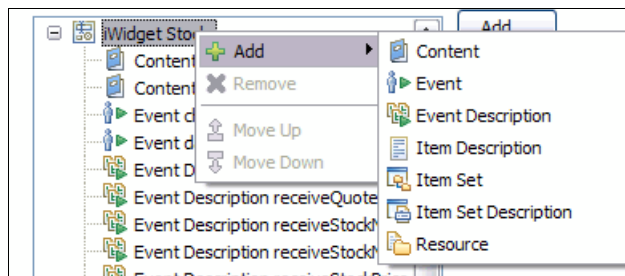


Figure 22-5 Adding sections to the iWidget Stock sample

The sample provides two itemsets: one itemset is a stock quote from Ameritrade and the other itemset is a stock quote from E\*Trade. Create a third itemset as a stock quote from TD Waterhouse. Follow these steps:

1. Click iWidget **Stock**.
2. Select **Add Item Set**.
3. For the Id, type TDWaterhouse. For the description, type Broker is TD Waterhouse. See Figure 22-6.

The screenshot shows a 'Details' dialog box with three input fields. The first field, labeled 'Id\*', contains the text 'TDWaterhouse'. The second field, labeled 'onItemsetChanged:', is empty. The third field, labeled 'Description:', contains the text 'Broker is TD Waterhouse'. There are small icons in the top right corner of the dialog box.

Figure 22-6 Adding the itemset

Items will contain each of the attributes and the values that make up the itemset. In this example, there are four items that are defined in the itemset: broker, change, price, and stock. These items correspond to the information that will be displayed by the iWidget.

Complete these tasks:

1. Select the Item Set **TDWaterhouse**.
2. Select **Add → Item**.
3. For the Item Id, enter broker.
4. For the Value, enter TDWaterhouse.
5. Repeat the previous steps and add Items with these IDs: change, price, and stock, and values of +1.00, \$52.00, and DELL, respectively.

See Example 22-1.

*Example 22-1 Source code snippet for the TD Waterhouse broker*

```
<iw:itemSet id="Ameritrade" description="Broker is Ameritrade">
 <iw:item id="stock" value="MSFT"/>
 <iw:item id="broker" value="Ameritrade"/>
 <iw:item id="price" value="$15"/>
 <iw:item id="change" value="-1.23"/>
</iw:itemSet>

<iw:itemSet id="TDWaterhouse" description="Broker is TD Waterhouse">
 <iw:item id="broker" value="TD Waterhouse" />
 <iw:item id="stock" value="DELL" />
```

```
<iw:item id="price" value="$52.00" />
<iw:item id="change" value="+1.00"/>
</iw:itemSet>
```

When you run the sample, you can switch to the **Edit** view and the TD Waterhouse entry appears in the drop-down list (Figure 22-7).



Figure 22-7 Stock Widget with the TD Waterhouse Itemset

Next you need to configure the event so that your View mode will see the event and display the new values. Because we have done this step for the other two entries, no changes are required. But, to illustrate how we configured the event so that View mode will see the event and display the new values, you can open the **stock.xml** file in the iWidget Editor again and click the **Content (edit)** tree node. You can see the script that is associated with the Edit view of the widget (Figure 22-8).

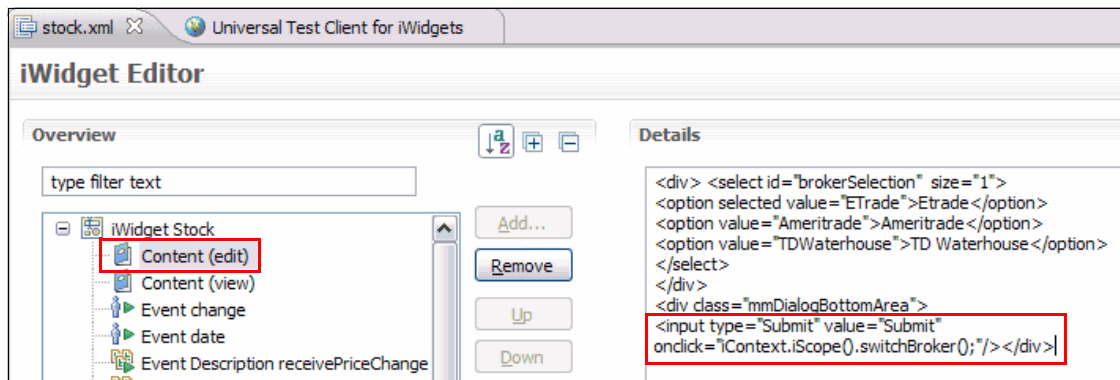


Figure 22-8 Edit view

Notice that the Submit button's onclick event is configured to run the `iContext.iScope().switchBroker()` method. The source for the method is in the `stock.js` JavaScript file. The function retrieves the current selection from the drop-down list, assigns the broker for this widget, and fires a new event signaling the change to the View mode. See Example 22-2.

```
switchBroker: function() {
 var element = this.iContext.getElementById("brokerSelection");
 this.broker = element.options[element.selectedIndex].value;

 this.iContext.iEvents.fireEvent("onModeChanged", null,
 "{newMode:'view'}");
}
```

---

### 22.3.3 Testing the sample iWidget application

You can test the sample application using the AJAX Test Server that is configured in Rational Application Developer. To launch the iWidget, follow these steps:

1. Open the **StockWidgetSample** project.
2. Expand the **WebContent** folder.
3. Right-click the **stock.xml** file.
4. Select **Run As** and **Run On Server**, as depicted in Figure 22-9 on page 1194.

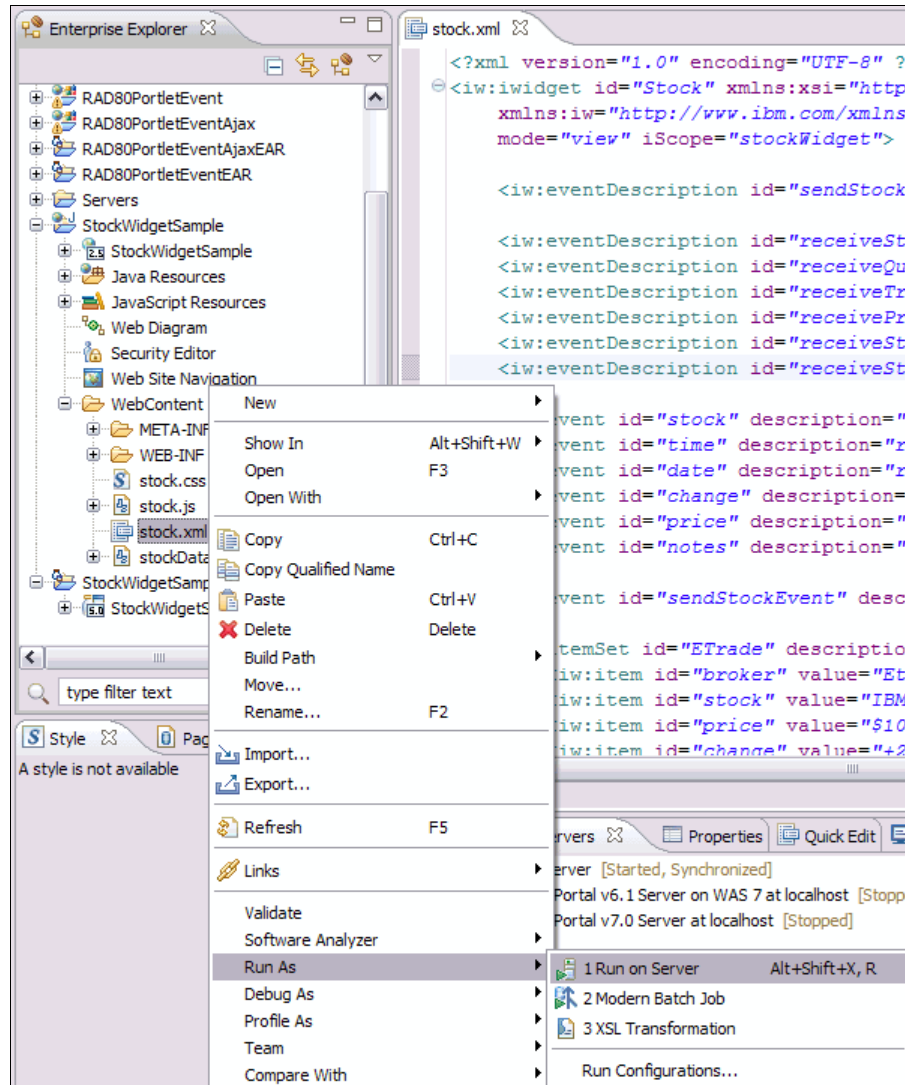


Figure 22-9 Launching the StockWidgetSample iWidget application

5. The Run On Server window opens with the list of installed servers. Select the **AJAX Test Server** from the list of servers and click **Finish**.
6. The StockWidgetSample application launches in the Universal Test Client. You can send events to the iWidget by entering the values in the Send events area, as shown in Figure 22-10 on page 1195, to your iWidget window.

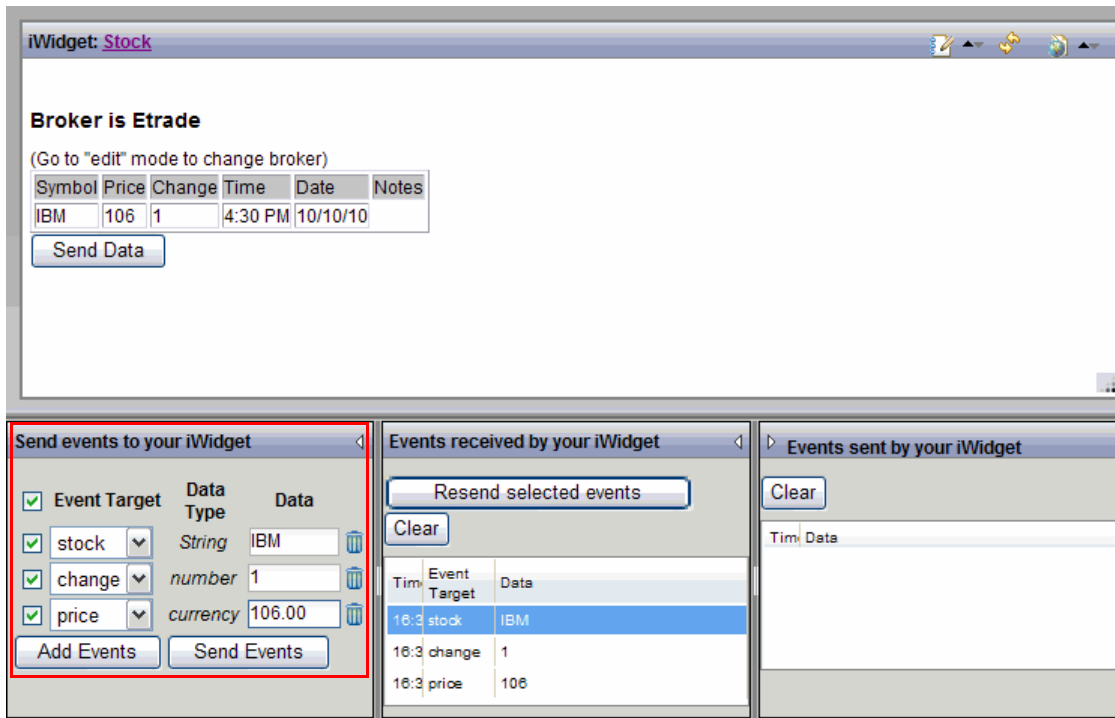


Figure 22-10 Sending test events to the StockWidgetSample iWidget

7. You can test your changes to the widget by switching to the **Edit** mode. See Figure 22-11.

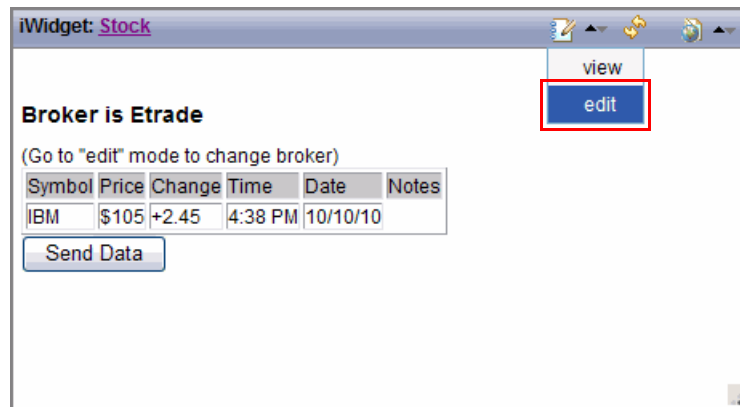


Figure 22-11 Switching to the Edit mode of an iWidget

- Simply select the TD Waterhouse itemset from the drop-down list and click **Submit** (Figure 22-12). Because the new event is handled by the existing code, the iWidget switches to the View mode and displays the values that you entered into the stock.xml file (Figure 22-13).



Figure 22-12 Selecting the TD Waterhouse itemset from the Edit mode

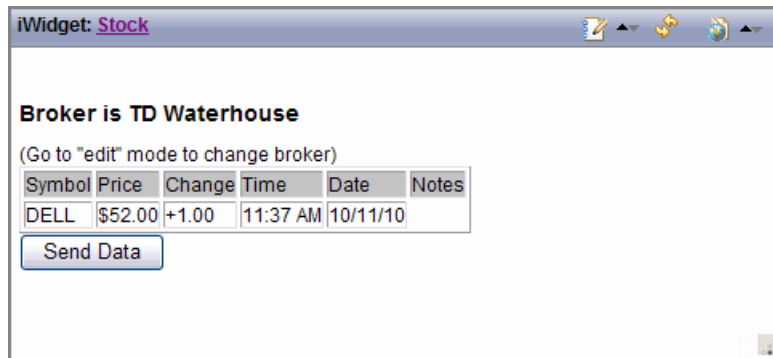


Figure 22-13 View mode updated with the TD Waterhouse selection

### 22.3.4 Deploying into WebSphere Portal V7

WebSphere Portal V7 has built-in support for running iWidgets. It has never been easier to publish your iWidget. Follow these steps:

- To start, right-click the **Enterprise Archive Project**, select **Properties** → **Targeted Runtimes**, and select **WebSphere Portal Server 7**. Right-click the WebSphere Portal V7 server that has been added to Rational Application Developer and click **Add and Remove**. The Add and Remove window opens. Move the **StockWidgetSampleEAR** project to the Configured: pane and click **Finish**. The StockWidgetSampleEAR is in the [Started, Synchronized] state. If not, make sure that the Portal Server is started and click **Publish** again.
- After the iWidget has been deployed, you can simply add it to your portal page like any other portlet or widget. Log in to your portal page and click **Action** → **Edit Page**. Next click **Customize** (Figure 22-14 on page 1197).



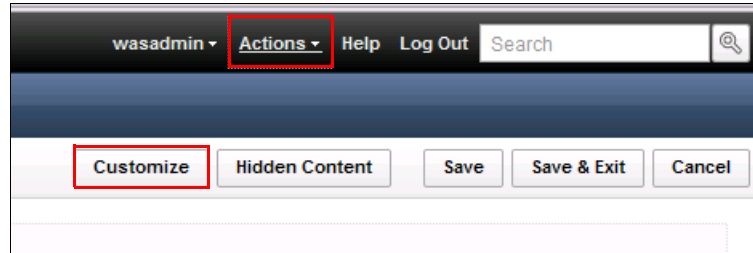


Figure 22-14 Adding the iWidget to the portal page

3. The portal page switches to the Add Content view. Click **All**, enter the name of the iWidget into the search box, and click the **magnifying glass icon** (🔍). See Figure 22-15.

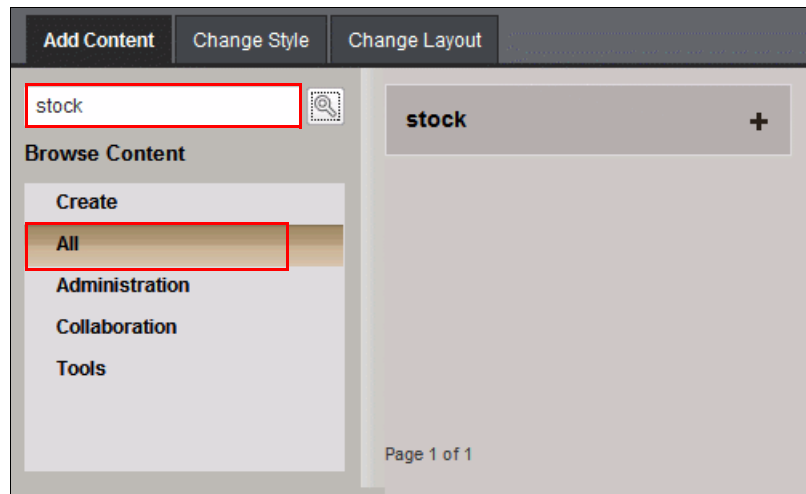


Figure 22-15 Searching for the iWidget from the list of available portlets

4. You can drag and drop the **stock** iWidget onto the preview pane and then click **Save and Exit**. Your new StockSampleWidget displays and is ready to use, as seen on Figure 22-16 on page 1198. You can click the drop-down list arrow in the upper right of the iWidget and click **Edit Shared Settings** to switch the broker, as described in 22.3.3, “Testing the sample iWidget application” on page 1193.

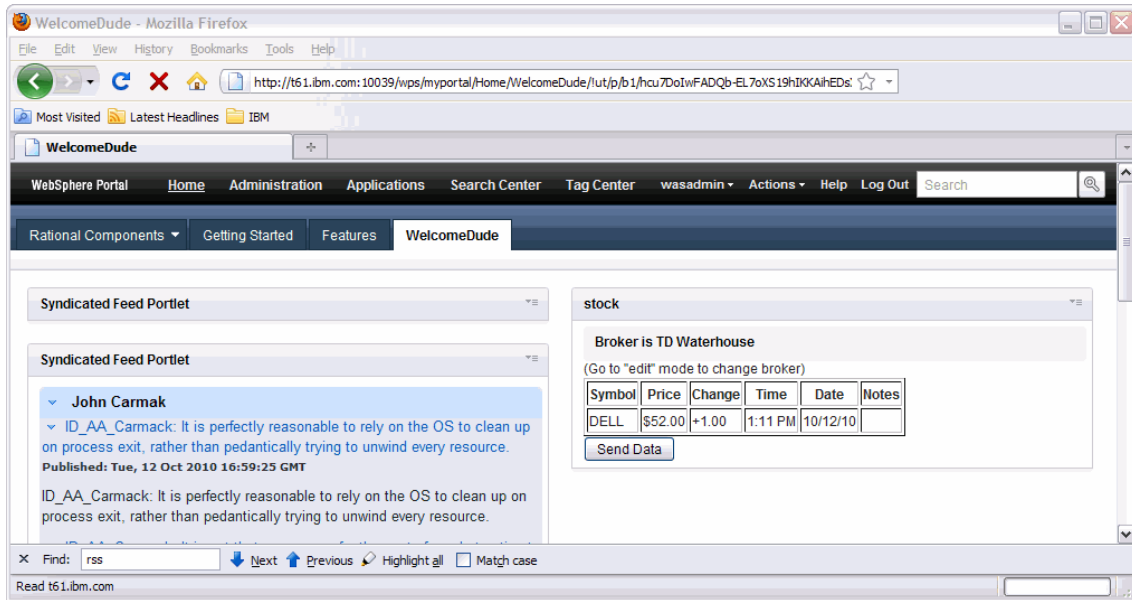


Figure 22-16 Viewing the iWidget sample on the portal web page

## 22.4 Additional resources

Rational Application Developer includes several iWidgets as part of the standard distribution. Several iWidgets are simple samples, and several iWidgets are specific to vertical markets. Table 22-1 on page 1199 shows a list of these iWidgets.

Table 22-1 iWidgets included in Rational Application Developer

iWidget	Description	Help topic
Stock widget	Simple stock ticker sample with two stock quote providers with fictional data	Samples → Web → iWidget samples
Color switch with button widget	Simple colored square that switches the color from blue to red when the button is clicked. Color switch widgets are used to show how to write HTML markup in a separate HTML or JSP file and then use AJAX to put that file into the iWidget's CONTENT element.	Samples → Web → iWidget samples
Color switch with timer widget	Simple colored square that switches the color from blue to red after receiving external events a given interval.	Samples → Web → iWidget samples
Dojo widgets	Widgets based on the Dojo JavaScript library.	Tutorials → Web → Create a Web 2.0 application with Dojo
Portlet widgets	iWidgets designed to run in the WebSphere Portal environment.	Samples → Portlet → Web 2.0 portlets → iWidget
Communications Enabled Application widgets	Widgets designed for the Telephony Vertical Market that provide interfaces into automatic call distributor (ACD) equipment.	Developing → Developing Web applications → Creating Communications Enabled Applications → Adding communications widgets to a Web page
IBM Enterprise Content Management widgets	iWidgets designed for the Enterprise Content Management and Business Process Management vertical markets that provide interfaces into IBM FileNet® and IBM Content Manager systems.	<a href="http://publib.boulder.ibm.com/infocenter/p8docs/v4r5m1/index.jsp?topic=/com.ibm.p8.doc/ecmwidgets_help/intro/intro_overview.htm">http://publib.boulder.ibm.com/infocenter/p8docs/v4r5m1/index.jsp?topic=/com.ibm.p8.doc/ecmwidgets_help/intro/intro_overview.htm</a>

## 22.4.1 Further information

See these resources for more information:

- ▶ Create and test IBM iWidgets using Rational Application Developer  
<http://www.ibm.com/developerworks/rational/library/10/create-and-test-ibm-iwidgets-using-rational-application-developer/index.html>
- ▶ iWidget viewlet  
<http://www.ibm.com/developerworks/wikis/download/attachments/129008975/iwidgets.swf?version=1>
- ▶ IBM iWidget V1.0 Specification: PDF Version  
[http://www-10.lotus.com/ldd/mashupswiki.nsf/dx/iwidget-spec-v1.0.pdf/\\$file/iwidget-spec-v1.0.pdf](http://www-10.lotus.com/ldd/mashupswiki.nsf/dx/iwidget-spec-v1.0.pdf/$file/iwidget-spec-v1.0.pdf)
- ▶ IBM iWidget V2.0 Specification: PDF Version  
<http://download.boulder.ibm.com/ibmdl/pub/software/dw/lotus/mashups/developer/iwidget-spec-v2.pdf>
- ▶ Introduction to creating mashups using IBM Mashup Center 2.0  
[http://www-10.lotus.com/ldd/mashupswiki.nsf/dx/Tutorial\\_Introduction\\_to\\_creating\\_mashups\\_using\\_IBM\\_Mashup\\_Center\\_2.0](http://www-10.lotus.com/ldd/mashupswiki.nsf/dx/Tutorial_Introduction_to_creating_mashups_using_IBM_Mashup_Center_2.0)
- ▶ *Lotus Connections iWidget Development Guide*  
<http://www-10.lotus.com/ldd/lcwiki.nsf/dx/development-guide>

# Deploying, testing, profiling, and debugging applications

In this part, we describe the tooling and technologies that are provided by Rational Application Developer for testing, deploying, profiling, and debugging.

This part includes the following chapters:

- ▶ Chapter 23, “Cloud environment and server configuration” on page 1203
- ▶ Chapter 24, “Building applications with Apache Ant” on page 1279
- ▶ Chapter 25, “Deploying enterprise applications” on page 1309
- ▶ Chapter 26, “Testing using JUnit” on page 1365
- ▶ Chapter 27, “Profiling applications” on page 1419
- ▶ Chapter 28, “Debugging local and remote applications” on page 1461

**Sample code for download:** The sample code for all of the applications that are developed in this part is available for download at the following address:

<ftp://www.redbooks.ibm.com/redbooks/SG247835>

See Appendix C, “Additional material” on page 1877, for instructions.



## Cloud environment and server configuration

Rational Application Developer provides support for testing, debugging, profiling, and deploying enterprise applications to local, remote test environments, and instances in Cloud servers. To run an enterprise application or web application in Rational Application Developer, the application must be published (deployed) to the server. We accomplish this task by deploying the EAR project, for the application, to an application server. With the server started, the application can be tested by using a web browser or the Universal Test Client (UTC) if it includes Enterprise JavaBeans (EJB).

In Rational Application Developer, the new IBM Rational Desktop Connection Toolkit for Cloud Environments is available. It allows you to manage resources on, and deploy to, the IBM Cloud.

In this chapter, we describe the features and concepts of server configuration. We also demonstrate how to configure a server to test applications.

## 23.1 Introduction to server configurations

Rational Application Developer includes support for many third-party servers obtained separately. One of the great features of the server tooling is the ability to simultaneously run multiple server configurations and test environments on the same development node where Rational Application Developer is installed.

When using Rational Application Developer, it is common for a developer to have multiple test environments or server configurations, which are made up of workspaces, projects, preferences, and supporting test environments (local or remote).

The server environment configuration includes the following key features, among others:

- ▶ Multiple workspaces with separate projects, preferences, and other configuration settings defined
- ▶ Multiple test environment servers configured for Rational Application Developer
- ▶ When using WebSphere Application Server v8.0 environments, multiple profiles, each potentially representing a separate server configuration

For example, a developer might want to have a separate server configuration for WebSphere Application Server v8.0 Beta with a unique set of projects and preferences in a workspace, and a server configuration pointing to a newly created and customized WebSphere Application Server v8.0 Beta profile. On the same system, the developer might create a separate portal server configuration, with unique portal workspace projects and preferences, and a WebSphere Portal V6.1 Test Environment. In the following topics, we explain how to create, configure, and run multiple WebSphere Application Server instances.

### 23.1.1 Application servers that are supported by Rational Application Developer

The most commonly used application server with Rational Application Developer is WebSphere Application Server. WebSphere Application Server is tightly integrated with Rational Application Developer, which offers tooling to test, run, and debug applications from the workbench, for example, by using the run-on-server functionality. You can specify server-specific configurations, such as extensions and bindings for a WebSphere Application Server, from the workbench.



You can start tooling, such as the WebSphere administrative console and the Profile Management Tool, for WebSphere Application Server from within the workbench. In addition, you can develop, run, and debug administrative scripts against a WebSphere Application Server. You can use these servers:

- ▶ IBM WebSphere Application Server versions 6.0, 6.1, 7.0, and 8.0 Beta
- ▶ IBM WebSphere Portal Version 6.1.x and 7.0
- ▶ J2EE Publishing Server (publish EAR to a server)
- ▶ Static Web Publishing Server (HTTP server for static web projects)

You can install the integrated test environments for WebSphere Application Server V6.1 and V7.0 using the IBM Installation Manager (Figure 23-1).

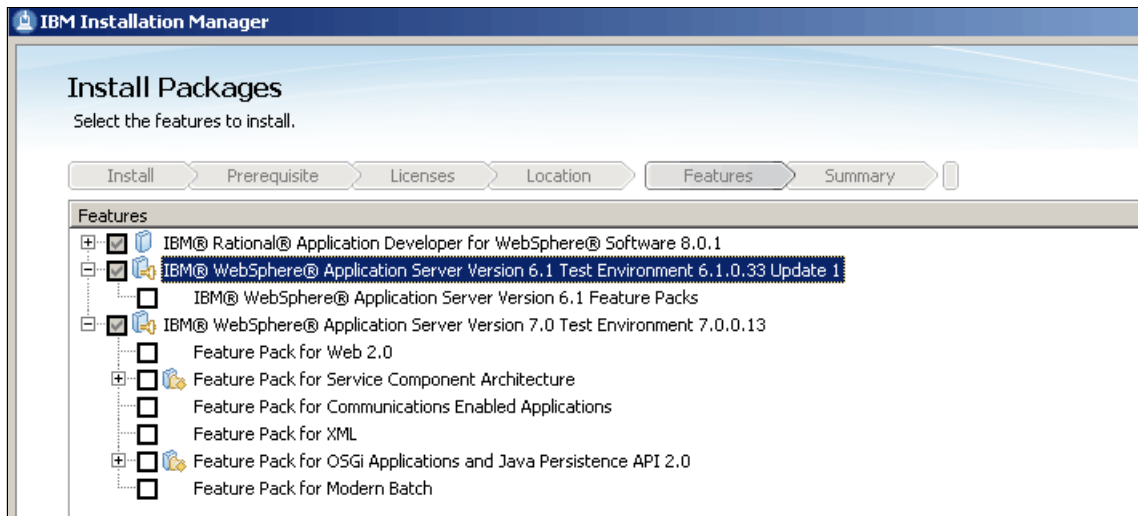


Figure 23-1 Servers available to install using IBM Installation Manager

Rational Application Developer server tools are based on the Eclipse Web Tools Platform (WTP) project. WTP provides a facility for publishing an enterprise application project and all of its modules to a runtime environment for testing purposes.

*Server adapters* are tools installed into the workbench that support a particular server. The server adapters in the following list are included, by default, in the Web Tools Platform installed with Rational Application Developer:

- ▶ Apache Tomcat versions 3.2, 4.0, 4.1, 5.0, 5.5, 6.0, and 7.0
- ▶ IBM WebSphere Portal Server versions 6.1 and 7.0
- ▶ IBM WebSphere Application Server V7.0 and V8.0 Beta
- ▶ JBoss versions 3.2, 3, 4.0, 4.2, and 5.0
- ▶ ObjectWeb Java Open Application Server (JOnAS) Version 4

- ▶ Oracle Containers for J2EE (OC4J) Standalone Server Version 10.1.3 and 10.1.3.x

To obtain additional server's adapters, follow these steps:

1. Open the New Server window, right-click the **Servers** window and select **New** → **Server**.
2. In the New Server window (Figure 23-2), click **Download additional server adapters**.

You can obtain the following additional server adapters at the time of writing this book:

- Apache Geromino Server versions 1.0, 1.1.x, 2.0, and 2.1
- IBM WebSphere Application Server Community Edition versions 1.1.x, 2.0, and 2.1

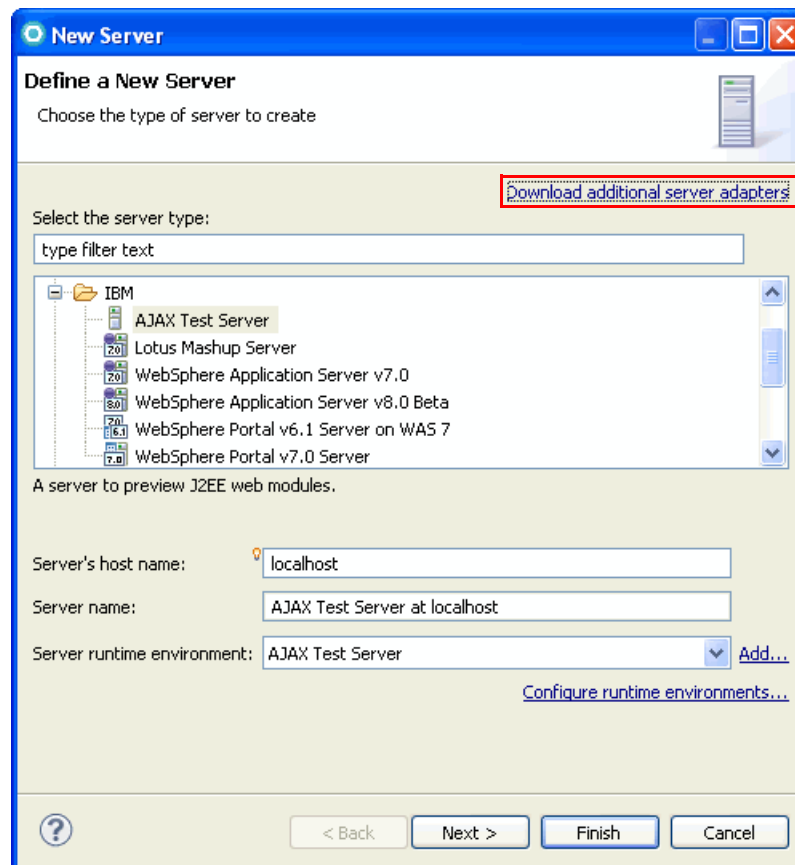


Figure 23-2 Downloading additional server adapters

### 23.1.2 Local and remote test environments

When configuring a test environment, the server can be either a local server (integrated with Rational Application Developer) or a remote server. After the server is installed and configured, the server definition within Rational Application Developer is similar for local and remote servers.

In both the local and remote configurations, Remote Method Invocation (RMI) or SOAP connectors can be used by Rational Application Developer to control the server using Java Management Extensions (JMX). The RMI (ORB bootstrap) port is designed to improve performance and communication with the server. The SOAP connector port is designed to be more firewall compatible and uses HTTP as the base protocol. In the case of local configurations, the interprocess communication (IPC) connector is also available and is the recommended connector.

## 23.2 Cloud extensions: Developing and testing applications on the IBM Smart Business, Development, and Test Cloud

New in Rational Application Developer is support for cloud computing environments. This new service for cloud computing simplifies the process of testing and deploying software by providing a flexible hosting environment that can be configured to the needs of each user. Using the tools in the IBM Rational Desktop Connection Toolkit for Cloud Environments, which is also referred to as the *Cloud Toolkit*, you can manage resources on and deploy them to the IBM Smart Business Development and Test Cloud, which is also referred to as the *IBM Cloud*.

In addition to the IBM Cloud web client, you can install the IBM Rational Desktop Connection Toolkit for IBM Cloud Environments, which allows you to work with the IBM Cloud images through your workspace. Using the tools in the IBM Rational Desktop Connection Toolkit for Cloud Environments, you can manage resources on, and deploy to, the IBM Cloud.

Clouds rely on *virtual systems*, which are computer systems that are insulated from their hosts by a layer of virtualization. Each virtual system is a complete computer system, typically including an operating system and one or more applications running on that operating system. The cloud provides a *virtual server*, a simulated host, for each virtual system. In this way, many virtual systems can run simultaneously on a single cloud. These virtual systems can be

created and destroyed much more easily than installing and uninstalling software on physical hosts, providing a flexible system that is responsive to your needs.

The IBM Cloud consists of a Rational Asset Manager repository and a *hypervisor*, which hosts virtual systems. The repository contains a library of virtual images, each of which defines a virtual system, including an operating system and software, such as application servers, database systems, and other IBM software products. Each virtual image is available in graduated sizes of Copper, Bronze, Silver, and Gold, referring to the relative capacity of the virtual server, its storage capacity, and other resources that are devoted to it.

The server tools for the Cloud Toolkit can directly request and consume a *subset* of the library of virtual images. The subset includes support for WebSphere Application Server V7.0.x servers. Several of the newer features of the Cloud Toolkit are only available in the Cloud Toolkit V1.0.0.1 or later, for example, cloud connection sharing. You are encouraged, therefore, that when updating your Rational Application Developer to Version 8.0.1 that you also update your Cloud Toolkit, which is a separate optional feature for Rational Application Developer and needs to be updated separately.

## 23.2.1 Installing IBM Rational Desktop Connection Toolkit for Cloud Environments

In addition to the IBM Cloud web client, you can install the IBM Rational Desktop Connection Toolkit for IBM Cloud Environments, which allows you to work with the IBM Cloud images through your workspace.

### Installing the cloud tooling

There are three installable components to the IBM Rational Desktop Connection Toolkit for Cloud Environments. Several of these options might not be available depending on the product that you install.

- ▶ Cloud Client for Eclipse  
Provides tools for interacting with the IBM Cloud environment from within Rational Application Developer or Rational Software Architect, such as browsing the IBM Cloud's asset catalog, requesting instances, and managing running instances.
- ▶ Server Tools Extension for the Cloud  
Provides tools for deploying applications to virtualized WebSphere Application Server instances on the IBM Cloud. This feature requires the WebSphere Application Server development feature to be installed if installing on Rational Application Developer, or the Extension for service-oriented

architecture (SOA) and WebSphere feature if installing on Rational Software Architect v8.0.

► Deployment Modeling Extension for the Cloud

Enables the Rational Software Architect Extension for Deployment Planning and Automation to interact with IBM Cloud environments, for purposes such as designing and cataloging virtual assets. This feature requires the Extension for Deployment Planning and Automation feature in Rational Software Architect v8.0.

These components can be installed through IBM Installation Manager.

If you did not install the Rational Desktop Connection Toolkit for Cloud Environments extension at the same time that you installed Rational Application Developer, follow these instructions to install it:

1. If Installation Manager is running, stop it.
2. Change to the *RAD\_SETUP* subdirectory of the directory where you extracted the disks for Rational Application Developer.
3. Start the Launchpad program:
  - For Microsoft Windows: Run **1launchpad.exe**.
  - For Linux: Run **1launchpad.sh**.
4. In the Launchpad dialog box, click **Install IBM Rational Application Developer V8.0**. IBM Installation Manager starts.
5. Click **Install packages**. The Install Packages wizard window opens.
6. On the first page of the Install Packages wizard, select **IBM Rational Desktop Connection Toolkit for Cloud Environments** (Figure 23-3 on page 1210) and then click **Next**.

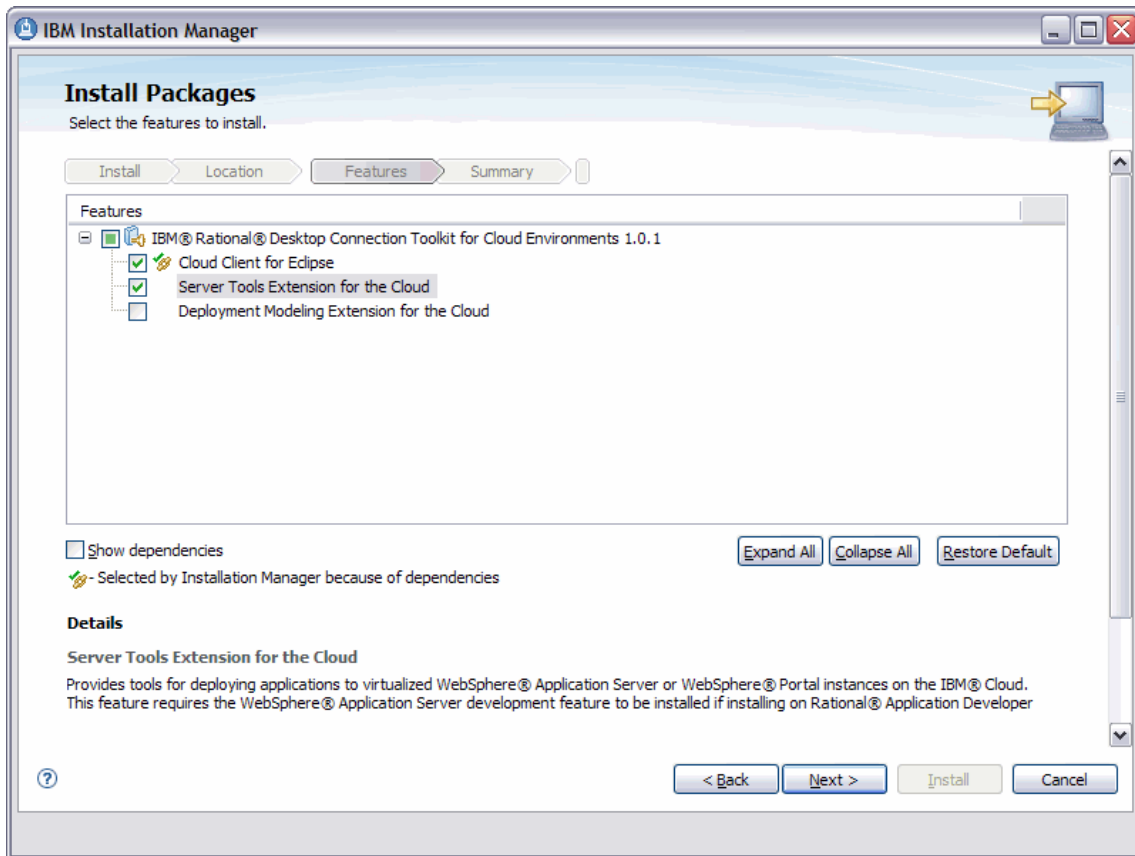


Figure 23-3 Installing IBM Rational Desktop Connection Toolkit for Cloud Environments

7. On the Licenses page, read the license agreement. If you agree to the terms of all of the license agreement, click **I accept the terms in the license agreements** and click **Next**.
8. On the Features page, select any additional features that you want to install and then click **Next**.
9. On the Summary page, review your choices before starting the installation process. If you want to change your selections, click **Back** to return to the previous pages. When you are satisfied with your installation choices, click **Install**.
10. When the installation process completes, click **Finish**.
11. Close the Installation Manager.

## 23.2.2 Working with the IBM Development and Test Cloud

You can create a server to specify a WebSphere Application Server runtime environment for testing or publishing your project resources. Example uses of a runtime environment are compiling an application, connecting to a server, and publishing applications on a server. Currently, several WebSphere Application Server V7.0.x images are available on the IBM Cloud. The images differ mostly in the particular licensing and service agreements that correspond to each image.

Typically, a Rational Application Developer user works with a WebSphere Application Server on the cloud in one of three ways:

- ▶ Creating a new server that points to a supported and active existing WebSphere Application Server cloud instance (using an existing instance option)
- ▶ Creating a new server that, as part of a Run on Server action, when run will make a new request for provisioning, prepare an instance, and start it (new request)
- ▶ Requesting a new instance on the web portable whose image type is consumable later from the server tools, specifically, the supported set of WebSphere Application Server-based images that the server tools can consume

The following sections describe each of these ways.

### Before you begin

You must perform these tasks:

- ▶ You must have registered with the IBM Smart Business Development and Test Cloud:  
<https://www.ibm.com/cloud/enterprise/dashboard>
- ▶ You must have installed the Cloud Client for Eclipse and Server Tools Extension for the Cloud.

Optionally, you can provision a WebSphere Application Server instance using the IBM Cloud Web client, or the cloud tooling in the workbench. If you choose to provision a WebSphere Application Server instance, you must generate a key pair on the Account page of the IBM Cloud Web client, and download the private key to a location that the workbench can access. If you do not have a WebSphere Application Server provisioned, you can do so through the Server Creation wizard.

Several of the features described require IBM Rational Desktop Connection Toolkit for IBM Cloud Environments V1.0.0.1, for example, support for connection

sharing with the Cloud Client view. Update your offering to this level if you want to make use of these features.

You will have a set of public and private keys; typically, you can generate a key set from the Account tab on the web client at this web address (<https://www.ibm.com/cloud/enterprise/dashboard>). Save the private key to a known path where it can be accessed later.

### **Generating key sets on the IBM Cloud**

You must save the private key to your local system, and this point in the process is the only time that you can. If you misplaced or failed to download the private key, you can simply generate a new key pair. Note, however, that instances provisioned with a key pair for which you do not have the matching private key will not be able to be remotely stopped and started from the server tools. The path to the correct, matching, private key on the local system is *required* to support remote starting and stopping of the WebSphere Application Server.

The public key is required for provisioning new image requests, and the private key is used for Secure Shell (SSH) connections from a stand-alone client or when remote start is issued from the server tools.

Figure 23-4 on page 1213 shows a portion of the Web Client wizard to generate and persist a key pair. Note the location where you save the private key, because you will need to specify or browse to the path when you create a server from Rational Application Developer.



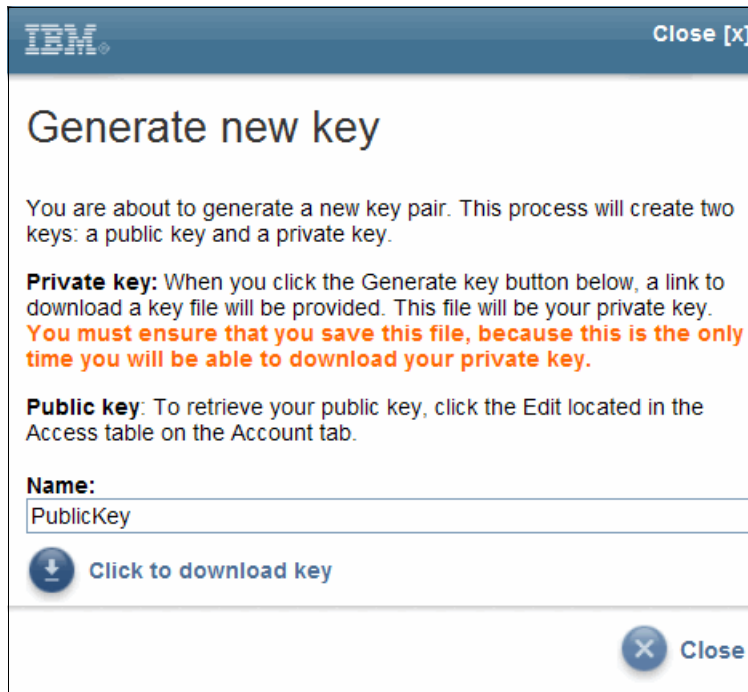


Figure 23-4 Generating a new key pair from the web client

## Creating a WebSphere Application Server on the IBM Cloud

Creating a WebSphere Application Server on the IBM Cloud is similar to creating a standard remote WebSphere Application Server with the addition of certain cloud-specific information. Follow these steps:

1. In the Servers view of Rational Application Developer, right-click and select **New** → **Server**. Follow these steps:
  - a. On the New Server: Define a New Server page, in the Select the server type list, under the IBM folder, select the version of the server that you want to create. Only WebSphere Application Server V7.0 is supported by the IBM Developer Cloud at the time of writing this book.
  - b. Select **Server to be defined here is for use in a cloud environment**. The server's host name field becomes read-only and is automatically filled with Cloud Services (Figure 23-5 on page 1214).

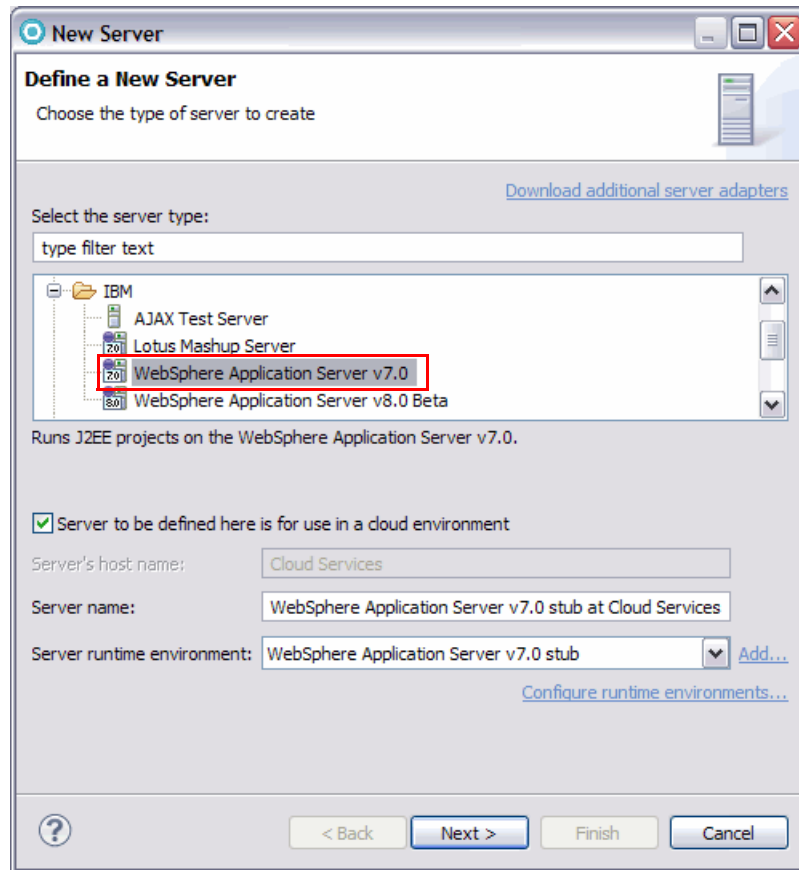


Figure 23-5 Defining the server for use in a cloud environment

- c. Optional: In the Server name field, type a label to identify this server entry in the Servers view. By default, this field is completed with the following naming conventions: *server type @ host name*, for example, WebSphere Application Server v7.0 at Cloud Services.
- d. If the workbench has a reference to a WebSphere Application Server runtime environment, a Server runtime environment list is available for you to select a runtime environment. In addition, you can add additional runtime environments by clicking the **Add** link, or you can modify the runtime environments defined in the workbench by clicking the **Configure runtime environments** link.
- e. If the workbench does not have a reference to the server runtime environment, the New Server wizard prompts you for this information in the WebSphere Application Server Runtime Environment page. Otherwise, this page does not display in the New Server wizard.

2. Optional: In the Name field, type a label to identify this server entry in the Server Runtime Environments preference page (**Window** → **Preferences** → **Server** → **Runtime Environments**):
  - a. In the Installation directory field, type or browse to the path where the WebSphere Application Server is installed. This path is the same as the *WAS\_ROOT* path mappings as defined by the WebSphere Application Server configuration. For example, if you have installed WebSphere Application Server in the *c:/WebSphere/AppServer* directory, specify this path in the Installation directory field.
  - b. Click **Next** to configure WebSphere Application Server settings.
3. You can connect to the cloud in the New Server wizard in a variety of ways:
  - Supply the IBM Cloud user name and password, as well as a unique connection name. Select the **Save cloud connection** check box (Figure 23-6 on page 1216). Then the connection information is available on subsequent invocations of this wizard, and you can opt to use that existing connection.

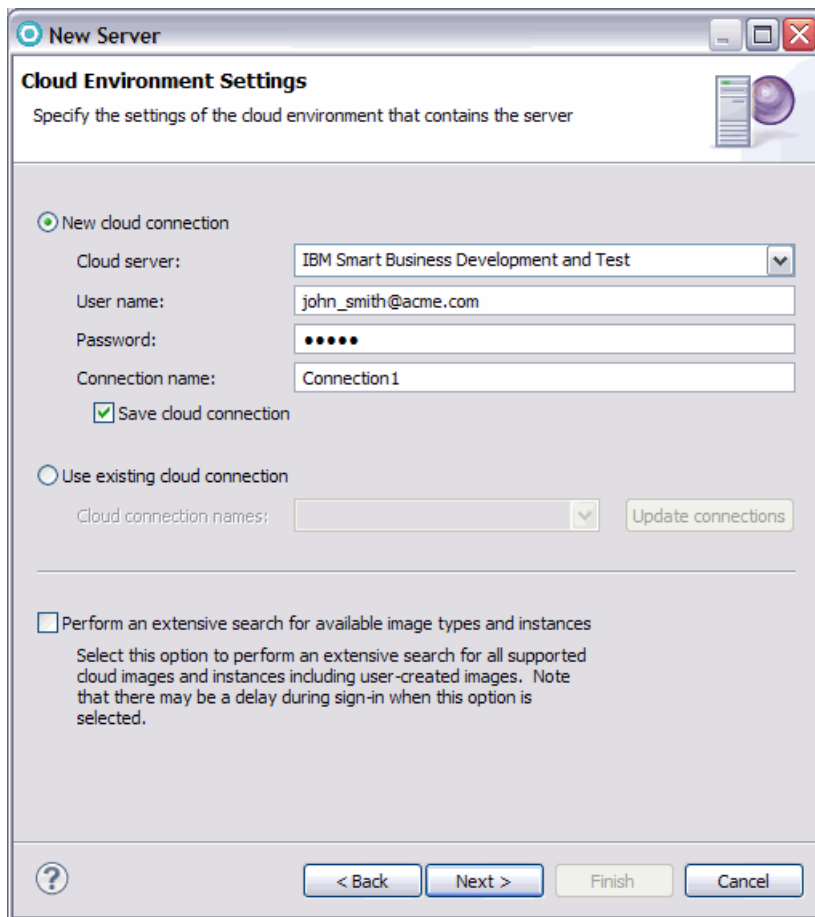


Figure 23-6 New Server wizard: Server to be used on the cloud

- Supply the IBM Cloud user name and password and do *not* provide a connection name. This method is appropriate if you do not want to persist and reuse the connection credentials later. In this case, you must clear **Save cloud connection**. If it is selected, it requires a unique connection name that has not been used already.
- Choose **Use existing cloud connection** to use an existing cloud connection (see Figure 23-7 on page 1217).

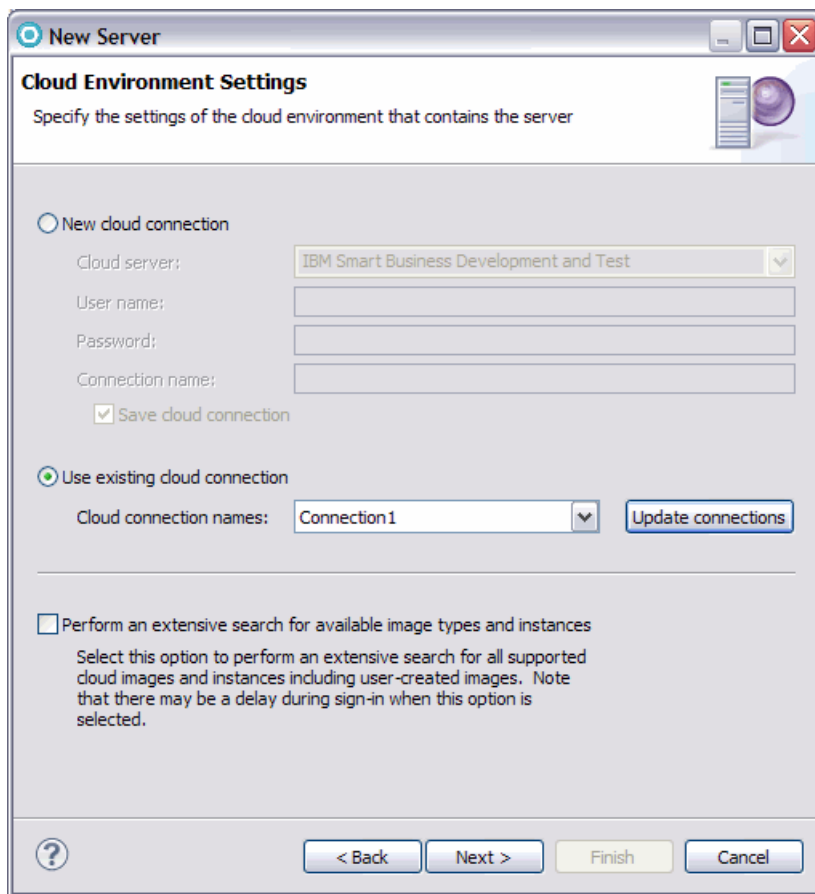


Figure 23-7 Using an existing cloud connection

4. On the Cloud Environment Settings page, enter the user name and password that you use to access the IBM Cloud, or select to use a pre-existing connection to the cloud. Optionally, you can choose to perform a more extensive search, which returns all available images, including user-created images.

Two-way connection sharing is supported with the Cloud Client. Any connections that were created from the Cloud Client views are available in the New Server wizard. In this way, you do not need to enter your credentials repeatedly if you rerun the wizard multiple times. Similarly, any connections created (with the option to save them) in the wizard are immediately available from the Cloud Client.

5. You have the option to request a new instance or use an existing, active instance if an instance is available. If an instance is available and active, it shows in the list of active instance names.

If you choose to request a new instance (Figure 23-8 on page 1219), enter a name for the server; select a location, server template, and the size of the instance. Select a key from the server, or specify a local public key file and path. You can generate a key pair using the IBM Cloud Web client on the Account page.

**New Server**

**Cloud Environment Settings**  
Specify the settings of the cloud environment that contains the server

**Request a new instance**

New request name: MyNewInstance

Location: RTP

Image template: IBM WebSphere Application Svr SLES 7.0.0.11 - DUO

Image Description: IBM WebSphere Application Server Base 7.0.0.11 with feature packs XML v1.0.0.3, Web 2.0 v1.0.0.2, SCA v1.0.1.1, CEA v1.0.0.3 for SUSE Linux Enterprise Server 11.0 (32-bit) for limited development use only

System size: Silver 32 bit

Quantity: 1

Price: \$0.265 US dollars per hour

Use public key from the repository server, if available

Available key names: publicKey3

Specify a local public key file

Local public key file path: Browse...

Use a static IP address

IP Addresses:

Use an existing, active instance, if available

Active instance names: AppServer 7.0.0.9 on SuSE, App Server 7.0.0.11 on Red Hat

Private key file: E:\Redbook\Dev&Test\publicKey3\ibmcloud\_s Browse...

Private key passphrase:

? < Back Next > Finish Cancel

Figure 23-8 Requesting a new cloud instance with WebSphere Application Server

If you choose to use an existing active instance that has already been provisioned and is running on the cloud, you can use that instance to point to the new server, too. In that case, you choose an existing instance from a list and provide the path to the appropriate private key, whose public key is

associated with that cloud instance and optionally a passphrase. See Figure 23-9.

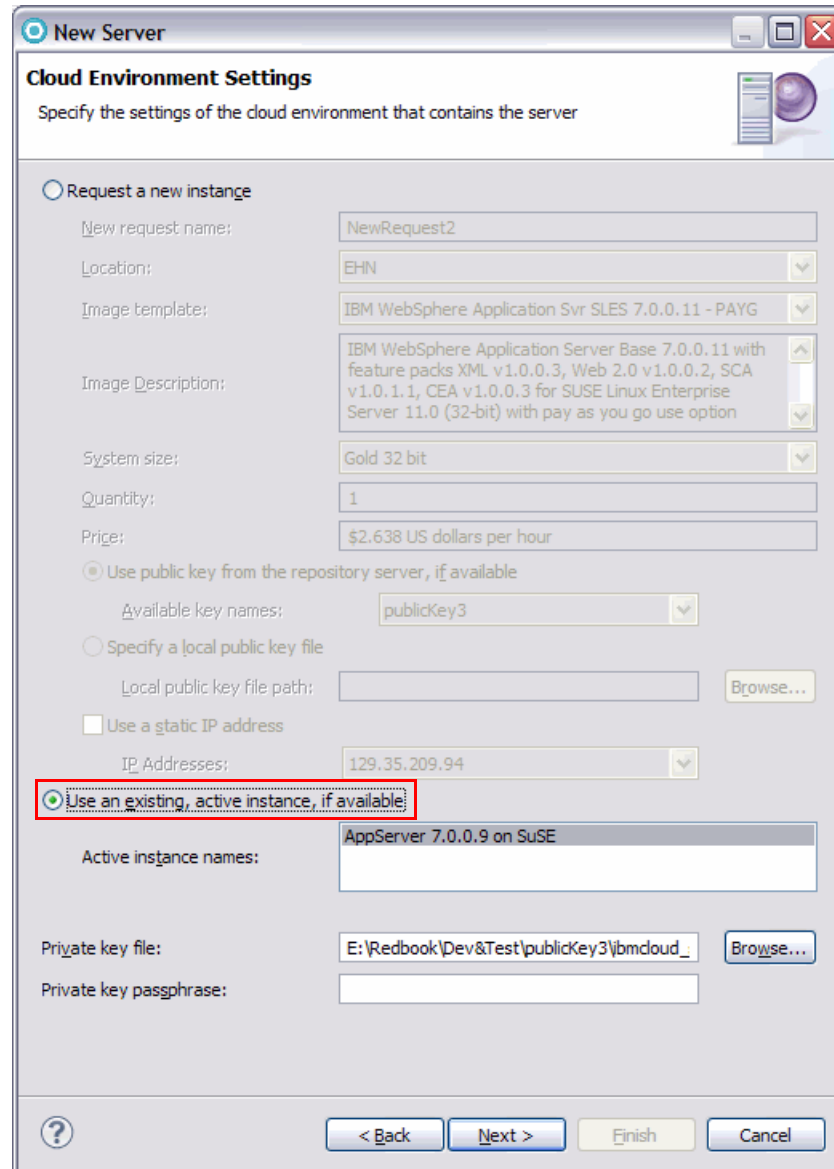


Figure 23-9 Using a suitable cloud instance that has been provisioned and is active

6. The WebSphere Application Server Settings are displayed (Figure 23-10 on page 1221). Table 23-1 on page 1221 shows the default settings. Certain fields cannot be edited and are read-only.



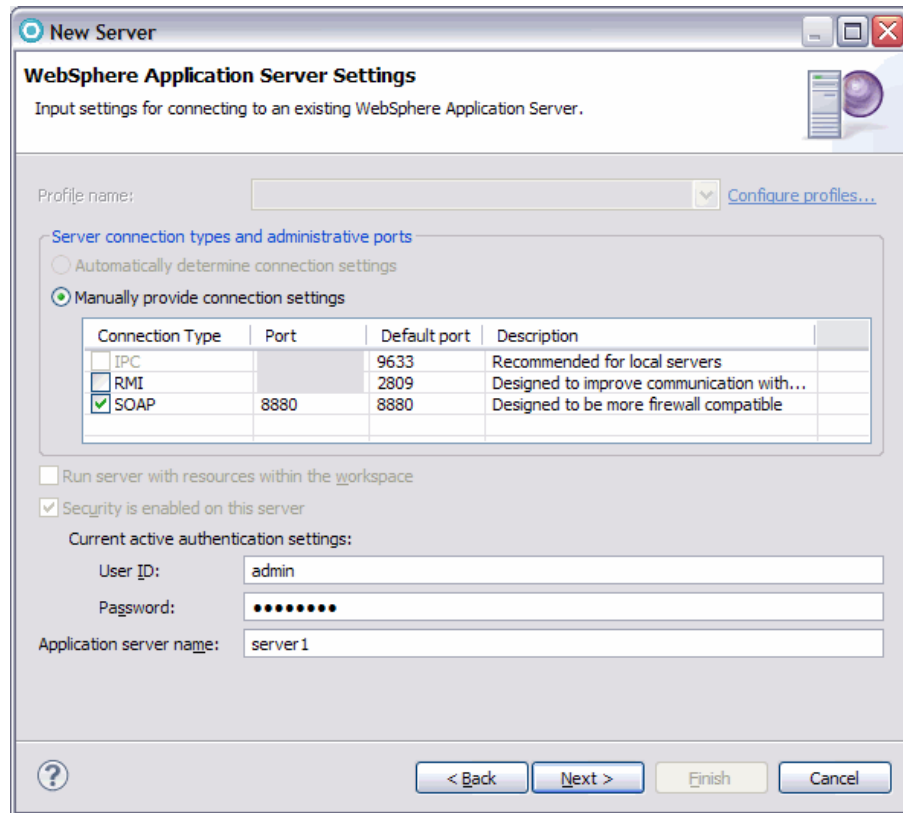


Figure 23-10 WebSphere Application Server Settings page

Table 23-1 Default setting for WebSphere Application Server Settings

Option	Description
Profile name	This option is available only when running a local server and is not available when running a remote server; this field cannot be edited and is disabled.

Option	Description
Server connection type and administrative port	<p>The administrative ports are used to communicate requests between the workbench and the server. The following ports are used for making Java Management Extensions (JMX) connections with the server:</p> <ul style="list-style-type: none"> <li>▶ The Remote Method Invocation (RMI) port, which is also known as the Object Request Broker (ORB) bootstrap, is designed to improve performance and communication with the server. The default setting of the RMI port is 2809 and is not enabled by default for cloud instances.</li> <li>▶ The SOAP connector port is designed to be more firewall compatible. The default setting of the SOAP port is 8880.</li> <li>▶ The interprocess communication (IPC) port is available only for a local WebSphere Application Server V7.0 or v8.0 Beta and cannot be edited.</li> <li>▶ The “Manually provide connection settings” option allows you to select which connection types you want to use. It is the only option available for cloud server instances. The default is port 8080, but you can edit it. You must provide the correct port numbers for each connection type that you choose.</li> </ul>
Run server with resources within the workspace	<p>This option is available only when running a local server and is not available when running a cloud server instance. It is therefore disabled.</p>
Security is enabled on this server	<p>Enables the security feature that comes with WebSphere Application Server. When security is not enabled, all other security settings are ignored.</p> <p>Instances of WebSphere Application Server on the cloud require that security is enabled. The option to deselect security is therefore disabled.</p> <p>By default, the User ID will be <code>virtuser</code>, and the password is the password that you entered when creating the server. Or, if you are requesting an instance, you can enter a password of your choice, provided that it meets the password requirements.</p>

Option	Description
Application server name	Specifies a logical name for the application server. For WebSphere Application Server, the logical name is unique and assigned to a server that distinguishes it from all other server instances within the node. This server name must already be created in the application server, and its default setting is server1.

The only mandatory, not pre-filled field that requires input from you, if you choose to use the defaults for the other settings, is the authentication password. This password field must contain the following information:

- If you use an existing instance, this password must match identically to the WebSphere administration password that was supplied at the time that the instance was requested. There is no validation or password rule enforced so ensure that the password is the correct password.
  - For a new cloud instance, this password must meet the required password rules for instancing this image, which are, at minimum, one uppercase character, one number, and one lowercase character. If the entered password does not meet this requirement, it is flagged with an error message and not accepted until it does meet the rule.
7. On the Remote WebSphere Application Server Settings page, you can select to enable the server to start remotely (Figure 23-11 on page 1224). Follow these steps:
- The platform and SSH information must be selected correctly and read-only. Enter the location of the server profile.
  - The private key path is the path that you entered on a previous wizard page, and the SSH user is always `idcuser`.

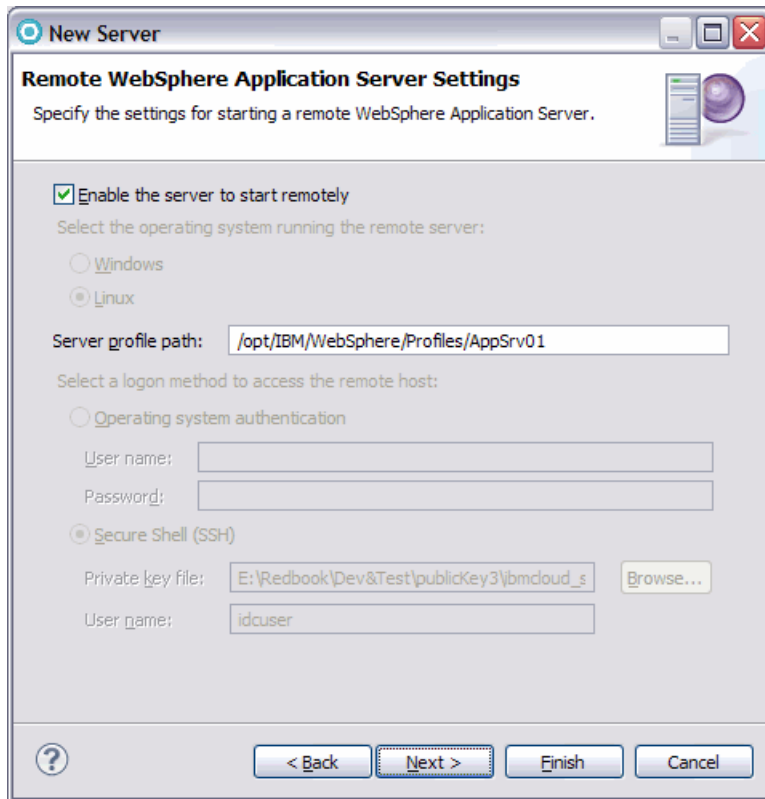


Figure 23-11 Remote server settings page

8. Optional: Click **Next** to add the projects of your application to the server. On the Add and Remove Projects page, under the Available projects list, select the project that you want to test and click **Add**. The project appears in the configured projects list.
9. Optional: A **Next** button might be available to click, depending on what type of projects you are adding to the server. If the **Next** button is available and you select it, the Select Task page appears. In the Select Task page, use the check boxes to select tasks to be performed on the server, such as create tables and data source.
10. Click **Finish**.

The server cloud server instance will be created.

When the wizard is complete, a new server is visible in the Servers view. Figure 23-12 on page 1225 shows a server that has been created but that has not yet been run. The tool tip shows that the instance that it represents has not

yet been provisioned. This server requests a new instance when it is first started as part of its start-up sequence. If it is a server that represents an existing cloud instance, the tool tip shows the assigned IP address.

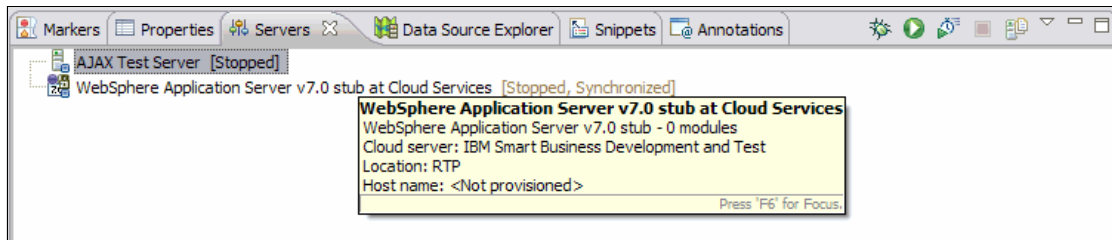


Figure 23-12 A server that will use a cloud instance that has not yet been provisioned

## Next steps

After you have created a server on the cloud, you can proceed as with any remote server instance. The server can be managed, started, stopped, and published using the Servers view.

If the server has been configured to request a new instance in which to start a WebSphere Application Server, it requests that a new instance is provisioned as part of starting the server. In that case, the status in the Servers view shows *Provisioning* during this time (Figure 23-13) and changes to *Starting* and then *Started*.

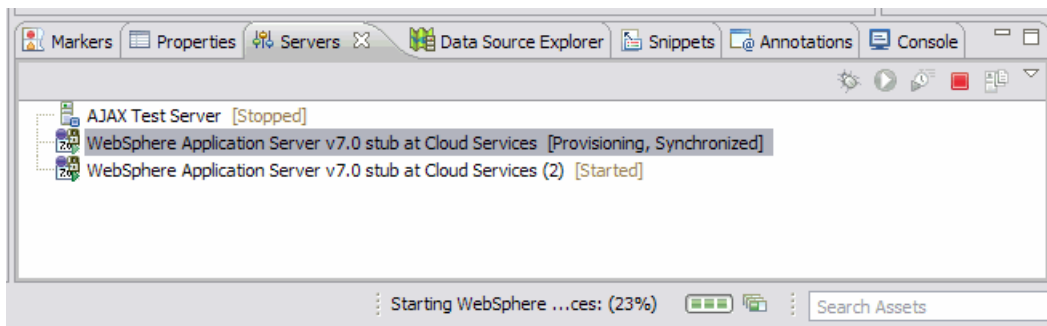


Figure 23-13 Managing, starting, stopping, and publishing using the Servers view

When the provisioning is complete, the system assigns an IP address to the instance. That IP address is used to update the host name for this server. If a static IP address was selected in the wizard, the system uses that IP address; if not, a dynamically generated IP address is assigned. Then the system starts the remote server in the cloud instance.

If the server was configured to use an existing instance, it does not need to make a new request first for provisioning but instead begins to start that server immediately. In that case, the server status does not show *Provisioning* but immediately shows *Starting*.

After it is configured, the server can be used as any other remote or local server. You can publish applications to it, run the administrative console, and so forth. The server provides a high degree of location transparency; you do not need to know from where the cloud instance is really running.

A server editor page is available to modify most cloud options after a server is first created by the New Server wizard. From this editor page, you can connect to the cloud to update your available assets. For example, the editor page lists any unused static IP addresses that are available and your current list of public keys. With this information, you can, for example, make a change to use an existing instance that has become available instead of requesting a new instance, as it is currently configured to do. When that server is started, it does not request a new instance to be provisioned, but instead, it uses an existing instance. In this way, it is possible to alter the configuration at any time after a server was first created, eliminating the need to create a new server.

The server editor page, like the New Server wizard, supports using a shared connection as well as creating a new cloud connection (Figure 23-14 on page 1227). This connection is used to connect to the cloud to get the latest information about available images that are consumable from Rational Application Developer, their prices and descriptions, as well as any active instances that might be used. It also supports the option to perform an extensive search.

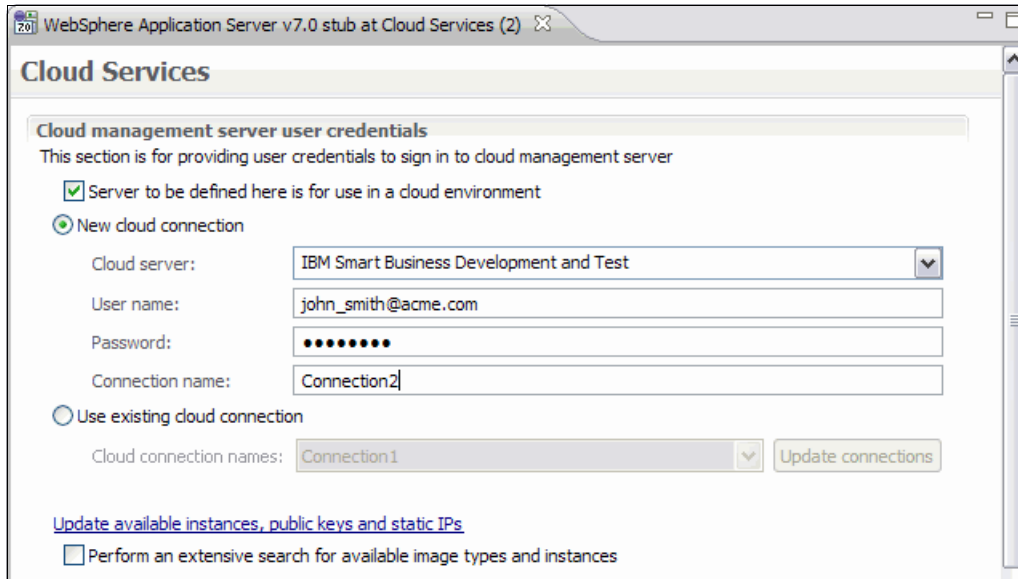


Figure 23-14 Choice of connection type: A new connection or a shared, existing connection

As with the New Server wizard, the editor page supports choices of image template, instance type (size), and will provide a description and pricing for each image and its size (Figure 23-15 on page 1228).

**▼ Create and reuse existing instances**

This section is for defining new instances or reusing existing instances and public keys

Request a new instance

New request name:

Location:

Image template:

Image Description:

System size:

Quantity:

Price:

Use public key from the repository server, if available

Available key names:

Local public key file path:

Use a static IP address

IP Addresses:

Figure 23-15 Image types available in the New Server wizard can also be used from the server editor later

The editor supports toggling between using an existing instance and making a new request (Figure 23-16 on page 1229). For example, a server that was originally created in the wizard to use an existing instance can be modified to instead make a new request when it is run the next time. The editor page provides the same functionality that the wizard provides, except that the server page cannot be used to create a new server. To create a new server, you must use the New Server wizard.



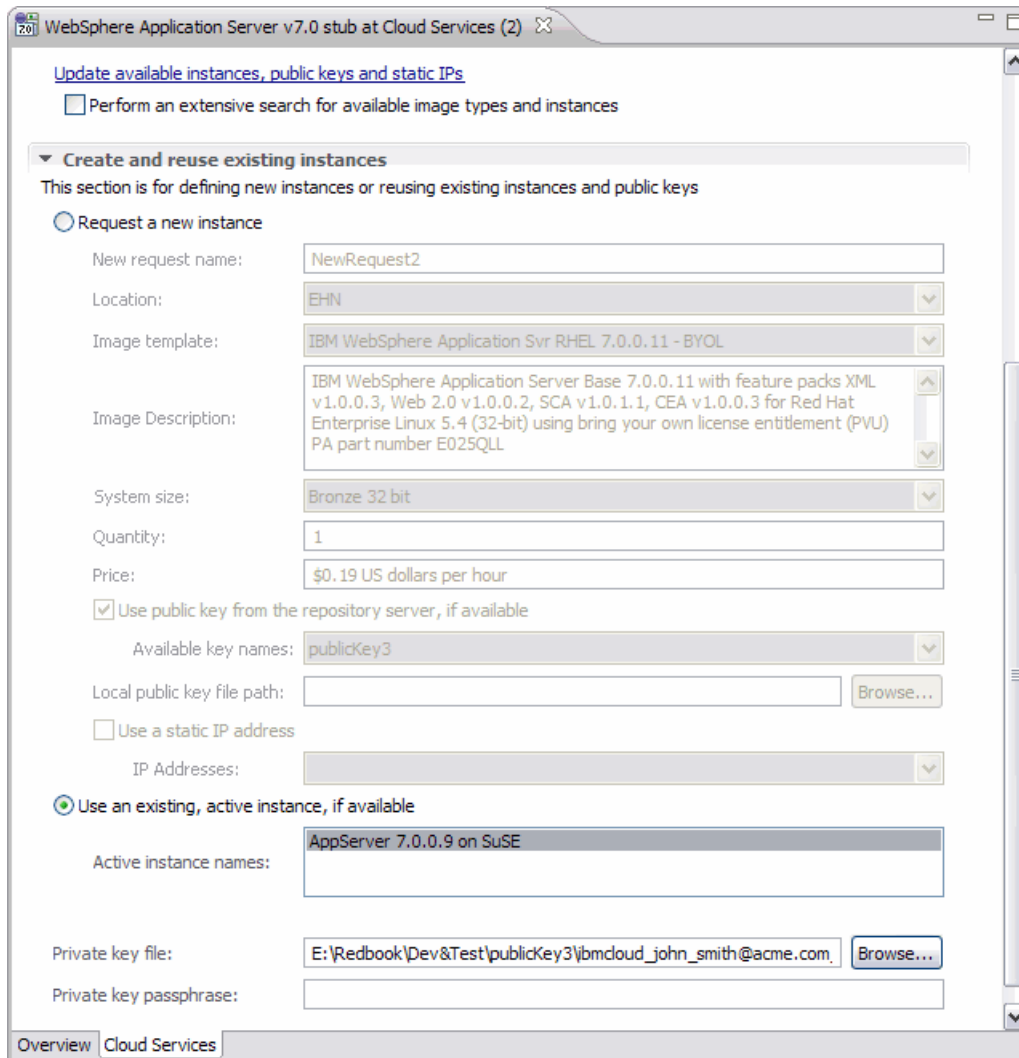


Figure 23-16 Setting a server created by the wizard to request a new instance to reuse an existing instance

### 23.2.3 Working with the Cloud Client for Eclipse

Use the Cloud Client to browse the IBM Cloud asset catalog, request instances, and manage running instances. Connections made from the Cloud Client are usable from the server wizard and the opposite is true as well; connections made from the server wizard are available from the connections that are made in the Cloud Client.

## Cloud views in the workbench

The cloud management view is available through **Windows** → **Show View** → **Cloud Explorer** (Figure 23-17).

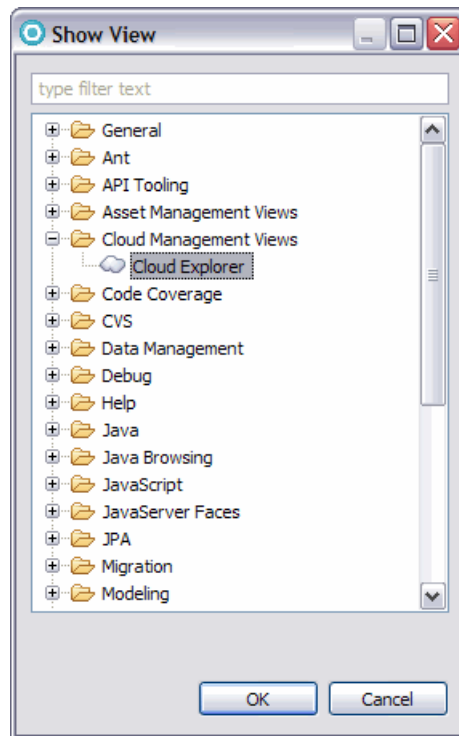


Figure 23-17 Launching the Cloud Client Explorer

## Requesting resources on the cloud from the Cloud Client

To request resources from the cloud server, you first browse or search through the virtual images that are available on the server. Then you select one or more images and specify the quantity and size of each image. The cloud server creates resources from the images, instantiated virtual systems from the images according to the quantity and size that you specify.

### Before you begin

Ensure that you have completed the following steps:

1. You must have registered with the IBM Smart Business Development and Test Cloud (<https://www-147.ibm.com/cloud/enterprise/dashboard>). For instructions to request a contract to use the IBM cloud, refer to *IBM Smart Business Cloud Development*:


<http://www.ibm.com/services/us/igs/cloud-development/>

2. You must have installed the Cloud Client for Eclipse.

There are two ways that you can request resources from the cloud:

- ▶ If you have only the Cloud Client installed, you can use the images on the server as they are.
- ▶ If you have the Deployment Modeling Extension for the Cloud installed, you can import the images into your workspace and construct a custom topology to request from the cloud.

The following steps show the simpler path: using the images on the server as they are. Follow these steps to request an image on the cloud:

1. Open the Cloud Explorer view:
  - a. Click **Window** → **Show View** → **Other**.
  - b. In the Show View window, expand the **Cloud Management Views** folder.
  - c. Click **Cloud Explorer** and then click **OK**.
2. In the Cloud Explorer view, click the **Request Instance** icon ()
3. If you have not yet created a connection to the cloud, you are prompted for the user ID and password that you use to sign in to the IBM Cloud.
4. The Select Image window lists all of the available images on the cloud. Select the image that you want and click **Add**.
5. On the Create a New Cloud Request window, enter a name for the image that you request. Depending on the type of image, your options and requirements in the parameter panel might differ.
6. After you have completed your selections, click **Finish** and read the license terms.

## What to do next

Now that you have requested an image from the cloud, you can use the software on that image like any other system. For example, if the image contains a Rational Asset Manager repository, you can connect to that server in the Asset Explorer view with the IP address of the image, like you connect to any Rational Asset Manager repository.

The Client URLs section of the Cloud Management view contains links that simplify using the image. For example, each image has a link labeled SSH login. Clicking this link opens a terminal window that is connected to the image. The first time that you connect to the image over SSH, you see a message asking if you want to accept the authenticity of the host and continue connecting; type yes, and press Enter. In the terminal window, you can examine the image, verify that it is running, and make changes to the image if necessary.

If the software in the image contains web-based access controls, such as the Rational Asset Manager Web client or the Rational Build Forge® console, links to those pages are listed in the Client URLs section. You can click these links to open a web browser to those pages.

Figure 23-18 shows the Cloud Explorer view.

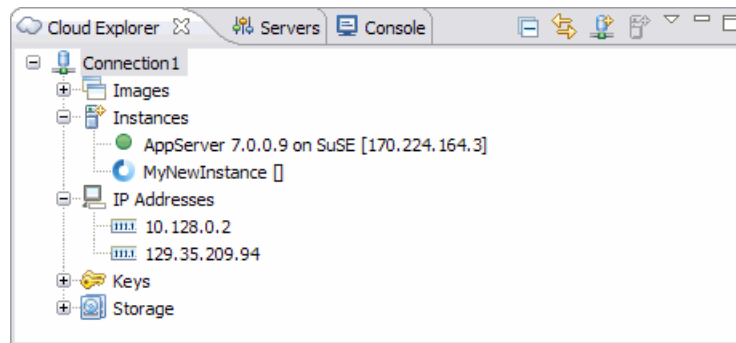


Figure 23-18 Cloud Explorer view

The Cloud Client (Figure 23-19) has a number of useful views and can be especially helpful as an alternative to using the web client when making instance requests for instances of other types of applications to use with the WebSphere Application Server. For example, it is convenient to use the Cloud Client to request that a DB2 instance is provisioned if you are developing applications that need database connectivity.

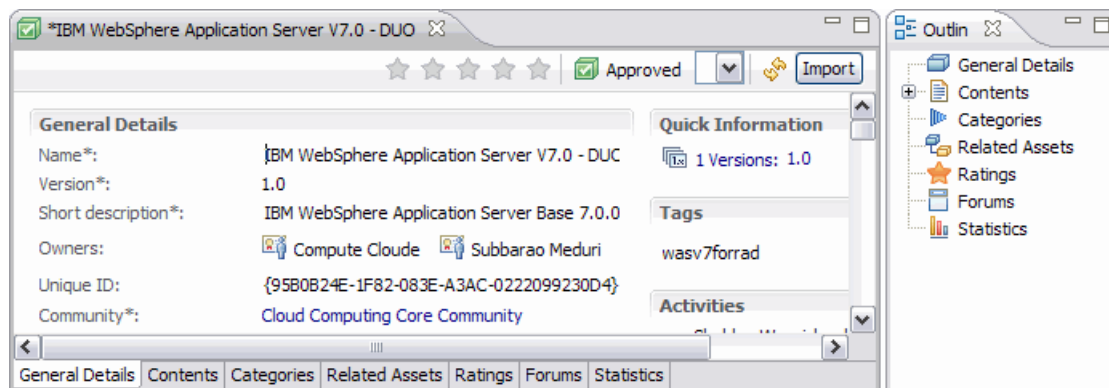


Figure 23-19 General Details tab from the Cloud Client

You can search for assets from the Cloud Client Asset Search view (Figure 23-20 on page 1233).

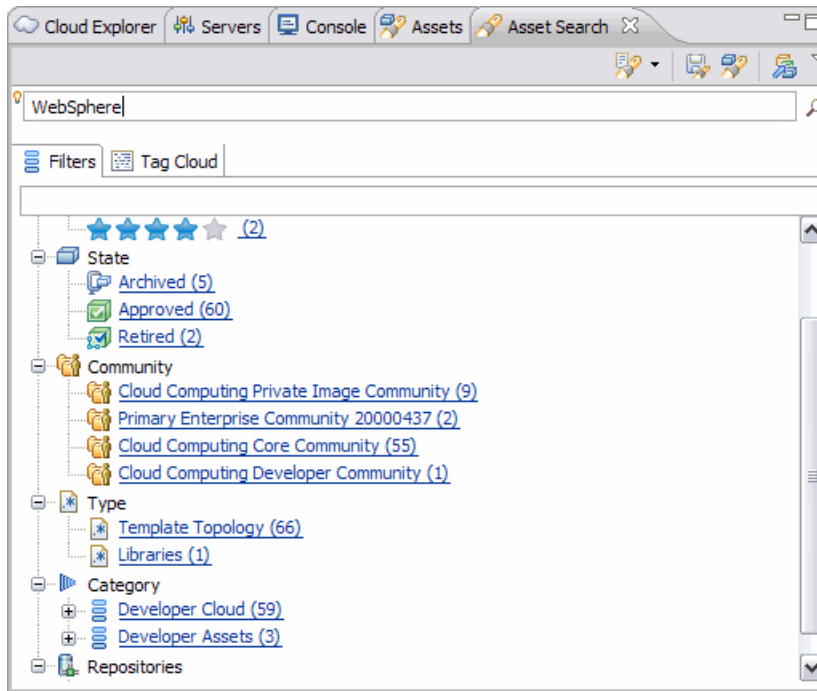


Figure 23-20 Searching assets from the Cloud Client Asset Search view

## 23.2.4 Requesting instances from the web client

Instances can be provisioned by the web client and then used from the Cloud Toolkit server tools, which can be useful when you already have a suitable cloud instance that might have the applications that you want to test already installed on it. It is also useful when setting up a server from a captured image that is available to you that might have been configured a specific way with a given set of deployed applications running on it.

To request an instance from the web client, select one of the images that contains the WebSphere Application Server that can be consumed by the Cloud Toolkit and that matches the specific licensing terms and agreements you are implementing. For example, Figure 23-21 on page 1234 shows selecting an image for provisioning that will contain WebSphere Application Server V7.0.0.11 on Red Hat Enterprise Linux (RHEL) with a Bring Your Own License (BYOL), which expects you to bring your own license entitlement.



### IBM WebSphere Application Svr RHEL 7.0.0.11 - BYOL

IBM WebSphere Application Server Base 7.0.0.11 with feature packs XML v1.0.0.3, Web 2.0 v1.0.0.2, SCA v1.0.1.1, CEA v1.0.0.3 for Red Hat Enterprise Linux 5.4 (32-bit) using bring your own license entitlement (PVU) PA part number E025QLL (Red Hat Enterprise Linux/5.4) [\[less\]](#)

Figure 23-21 Requesting a WebSphere Application Server V7.0.0.11 on Red Hat Enterprise Linux Server

The pop-up wizard guides you through making this request. If you have a static IP address that is unbound to any instance, you can elect to use that IP address at this time. If you do not elect to use a static IP address, a dynamically generated IP address will be assigned when the provisioning is complete. The specific format of the fields for which to supply values can differ from one type of image to another type of image, by data center location and by WebSphere Application Server version.

Figure 23-22 on page 1235 shows page 2 of 4 of the request for a WebSphere Application Server RHEL 7.0.0.11 with a BYOL agreement. The user has elected to use a static IP address that it has been assigned, and the image type is Bronze 32 bit. The usage price per hour in American dollars is provided. An instance of this type and this size costs USD.021 per usage hour.

**IBM** Close [x]

## Add instance

Step 2 of 4: configure image

**You selected: IBM WebSphere Application Svr RHEL 7.0.0.11 - BYOL**  
 IBM WebSphere Application Server Base 7.0.0.11 with feature packs XML v1.0.0.3, Web 2.0 v1.0.0.2, SCA v1.0.1.1, CEA v1.0.0.3 for Red Hat Enterprise Linux 5.4 (32-bit) using bring your own license entitlement (PVU) PA part number E025QLL

Complete the fields below to configure your instance selection. Required fields are indicated with an asterisk (\*).

**Request Name:** App Server 7.0.0.11 on Red Hat

**Quantity:** 1

**Server Size:** Bronze 32 bit

**Root only:**  Yes

**Expires on:** 11/17/12

**Key:** MyPublicKey2 [+ Add Key](#)

**VLAN:** Public Internet

**Select IP:** 129.35.209.94 [How do I add an IP?](#)

**Virtual IP:** none [+ Add IP](#)

**Persistent disk:** [How do I add Storage?](#)

**Image ID:** 20010556

**Price:** \$0.21 / UHR

[← Previous](#)
[Next →](#)
[× Cancel](#)

Figure 23-22 Requesting a new instance for a specific image type

When the request has been successfully submitted, you can view its current status from the control panel. Figure 23-23 on page 1236 shows two instances that have been requested and that are at separate stages in the provisioning process. The first instance, which is set to use a static IP address, is in the *Request* stage. The second instance, which is on a separate data center, that does not use a static IP address, but that will let the system dynamically allocate an IP address, is in the *Provisioning* stage.

Instance name	IP Address	Created on	Running	Data Center	Price/unit	Status
App Server 7.0.0.11 on Red Hat	129.35.209.94	Nov 18, 2010	0 Hour	EHN	\$0.21 / UHR	Requesting
AppServer 7.0.0.9 on SuSE		Nov 18, 2010	0 Hour	RTP	\$0.15 / UHR	Provisioning

Figure 23-23 Viewing the status of the requests for provisioning from the control panel

Figure 23-24 shows that both requests are completed and that the status for both is *Active*. *Active* implies that the instance is now provisioned and is starting.

Instance name	IP Address	Created on	Running	Data Center	Price/unit	Status
App Server 7.0.0.11 on Red Hat	129.35.209.94	Nov 18, 2010	1 Hour	EHN	\$0.21 / UHR	Active
AppServer 7.0.0.9 on SuSE	170.224.164.3	Nov 18, 2010	1 Hour	RTP	\$0.15 / UHR	Active

Figure 23-24 Status when provisioning is complete

## 23.2.5 Resources for additional information

The following resources are useful references about cloud computing:

- ▶ developerWorks video about developing with Rational Application Developer and the IBM Cloud  
[http://www.ibm.com/developerworks/wikis/download/attachments/129008975/rational\\_application\\_developer\\_8\\_with\\_cloud\\_toolkit.swf?version=1](http://www.ibm.com/developerworks/wikis/download/attachments/129008975/rational_application_developer_8_with_cloud_toolkit.swf?version=1)
- ▶ IBM Cloud Computing home page  
<http://www.ibm.com/ibm/cloud/>
- ▶ Smart Business Development and Test on the IBM Cloud website  
<http://www.ibm.com/services/us/igs/cloud-development/>
- ▶ IBM Smart Business Development and Test Cloud home page  
<https://www.ibm.com/cloud/enterprise/dashboard>
- ▶ IBM Smart Business Development and Test Cloud support page  
<https://www.ibm.com/cloud/enterprise/support>
- ▶ IBM Smart Business Development and Test on the Cloud (Enterprise): Support Forum  
<https://www.ibm.com/developerworks/mydeveloperworks/groups/join/1dba2e59-05da-4b9a-84e4-2444a6cac251>
- ▶ developerWorks Cloud Computing home page  
<http://www.ibm.com/developerworks/cloud/index.html>



## 23.3 Understanding WebSphere Application Server v8.0 profiles

The concept of *server profiles* was introduced starting with WebSphere Application Server V6.0. The WebSphere Application Server installation process lays down a set of core product files required by the runtime processes. After installation, you have to create one or more profiles that define the runtime settings for a functional system. The core product files are shared between the runtime components that are defined by these profiles.

With WebSphere Application Server Base Edition, you can only have stand-alone application servers (Figure 23-25). Each application server is defined within a single cell and node. The administrative console is hosted within the application server and can only connect to that application server. No central management of multiple application servers is possible. An application server profile defines this environment.

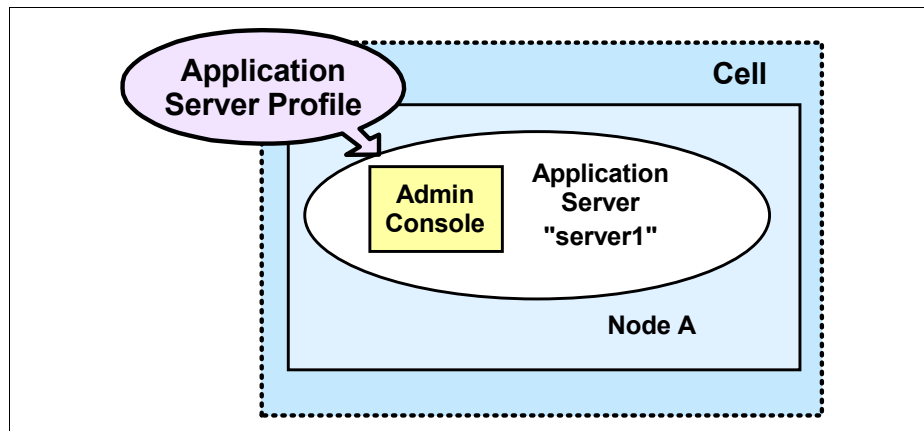


Figure 23-25 System management topology: Stand-alone server (Base)

You can also create stand-alone application servers with the Network Deployment package, although you might do so with the intent of federating that server into a cell for central management at a later point.

With the Network Deployment package, you have the option of defining multiple application servers with central management capabilities. For more information about profiles for the IBM WebSphere Application Server V7.0 Network Deployment Edition, see *WebSphere Application Server V7 Administration and Configuration Guide*, SG24-7615.

## 23.3.1 Types of profiles

The following types of profiles are used when defining the run time of an application server:

- ▶ Application server profile

**Server profile:** Rational Application Developer uses this application server profile for WebSphere Application Server v8.0.

- ▶ Deployment manager profile
- ▶ Custom profile

## 23.3.2 Using the profiles

Here, we discuss situations where you might use the various types of profiles.

### Application server profile

The application server profile defines a single stand-alone application server. Using a profile gives you an application server that can run stand-alone (unmanaged) with the following characteristics:

- ▶ The profile consists of one cell, one node, and one server. The cell and node are not relevant in terms of administration, but you will see them when you administer the server through the administrative console.
- ▶ The name of the application server is server1.
- ▶ The server has a dedicated administrative console.

The primary use for this type of profile can be any of the following possibilities:

- ▶ To build a server in a Base installation, including a test environment within Rational Application Developer.
- ▶ To build a stand-alone server in a Network Deployment installation that is not managed by the deployment manager, for example, to build a test machine.
- ▶ To build a server in a distributed server environment to be federated and managed by the deployment manager. Choose this option if you are new to WebSphere Application Server and want a quick way to get an application server complete with samples. When you federate this node, the default cell becomes obsolete, and the node is added to the deployment manager cell. The server name remains as server1, and the administrative console is removed from the application server.

## Deployment manager profile

The deployment manager profile defines a deployment manager in a Network Deployment installation. Although you can conceivably have the Network Deployment package and run only stand-alone servers, this approach might bypass the primary advantages of Network Deployment, which are workload management, failover, and central administration.

In a Network Deployment environment, create one deployment manager profile, which gives you the following items:

- ▶ A cell for the administrative domain
- ▶ A node for the deployment manager
- ▶ A deployment manager with an administrative console
- ▶ No application servers

After you have the deployment manager, you can federate nodes built either from existing application server profiles or custom profiles. You can also create new application servers and clusters on the nodes from the administrative console.

## Custom profile

A custom profile is an empty node that is intended for federation to a deployment manager. This type of profile is used when you build a distributed server environment. You might use a custom profile in the following ways:

- ▶ Create a deployment manager profile.
- ▶ Create one custom profile on each node on which you will run application servers.
- ▶ Federate each custom profile of the deployment manager, either during the custom profile creation process or later using the **addNode** command.
- ▶ Create new application servers and clusters on the nodes from the administrative console.

## 23.4 WebSphere Application Server v8.0 Beta installation

The IBM WebSphere Application Server v8.0 Beta environment is an installation option that is available in IBM Installation Manager when installing Rational Application Developer. For details about how to install the WebSphere Application Server v8.0 Beta Environment, see “Installing Rational Application Developer” on page 1788.

The stub files for the IBM WebSphere Application Server v8.0 Beta environment are in the directory <rad\_home>\runtime\base\_v8\_stub.

The stub folder contains minimal sets of compile-time libraries with which you can build applications for a server when it is not installed locally.

## 23.5 Using WebSphere Application Server profiles

The Profile Management tool is a WebSphere Application Server tool that creates the profile for each runtime environment. It is also a graphical user interface to the WebSphere Application Server command-line tool, **wasprofile**. You can use the tools in Rational Application Developer to start the WebSphere Application Server Profile Management tool.

### 23.5.1 Creating a new profile using the WebSphere Profile wizard

To create a new WebSphere Application Server v8.0 Beta profile using the WebSphere Profile Management Tool, follow these steps from within the Rational Application Developer environment:

1. In the workbench, select **Window** → **Preferences**.
2. In the Preferences window (Figure 23-26 on page 1241), complete these actions:
  - a. Expand **Server** → **WebSphere Application Server**.
  - b. Under the WebSphere Application Server local server profile management list, select **WebSphere Application Server v8.0 Beta**.
  - c. Click **Run Profile Management Tool** (next to the “WebSphere profiles defined in the runtime selected above” list), which lists all the profiles defined for the runtime environment that you selected in the previous step.

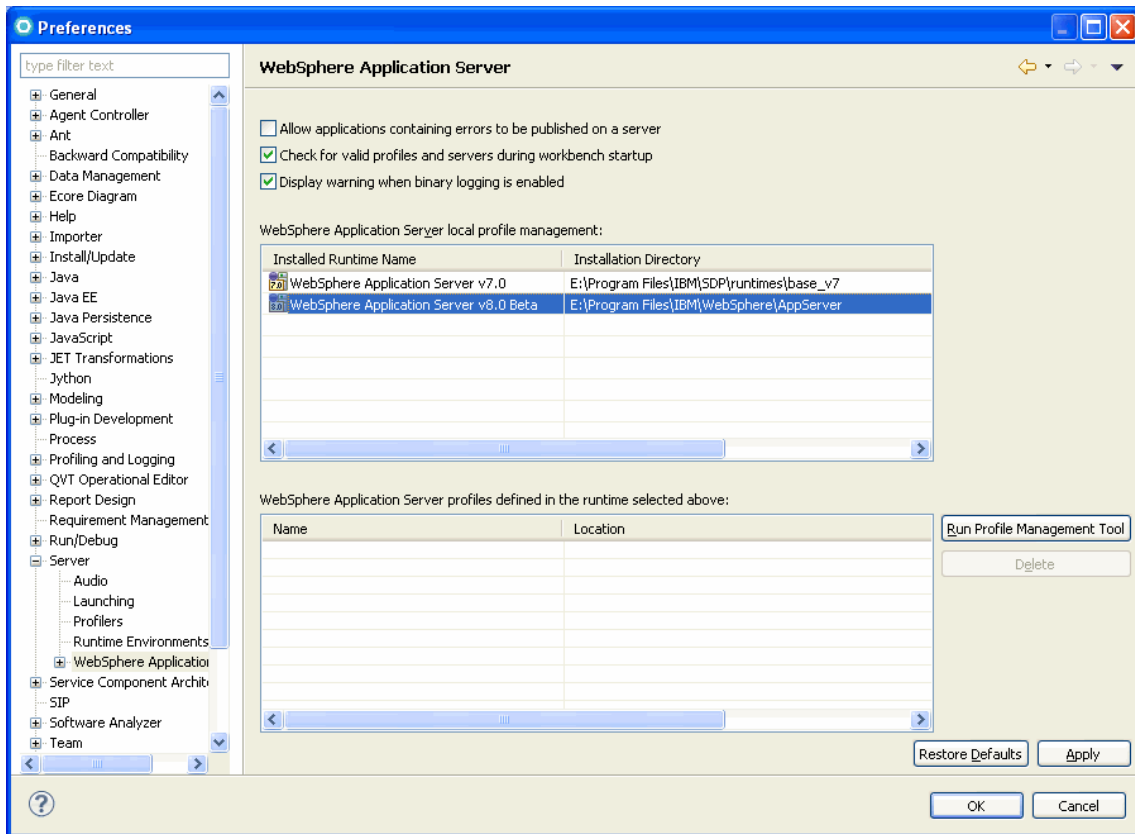


Figure 23-26 Server Preferences page

3. The WebSphere Customization Tools 8.0 window opens. Select **Profile Management Tool** in the list of provided tools and click **Launch Selected Tool**.
4. In the Profile Management Tool window, any existing profiles are listed. Click **Create** to start the new profile creation process (Figure 23-27 on page 1242).

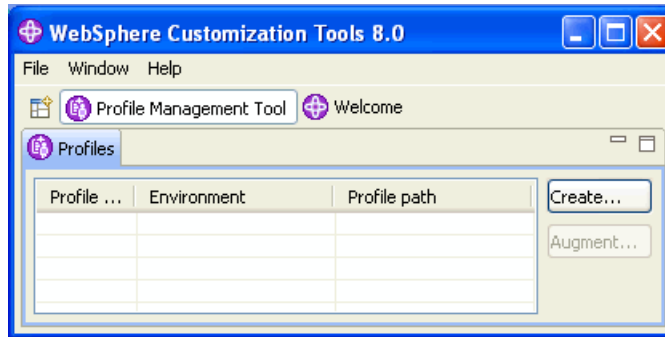


Figure 23-27 Profile Management Tool window

5. In the Environment selection window, in which **Application Server** is preselected, click **Next**.

6. In the Profile Creation Options window (Figure 23-28), select **Advanced profile creation** and click **Next**.

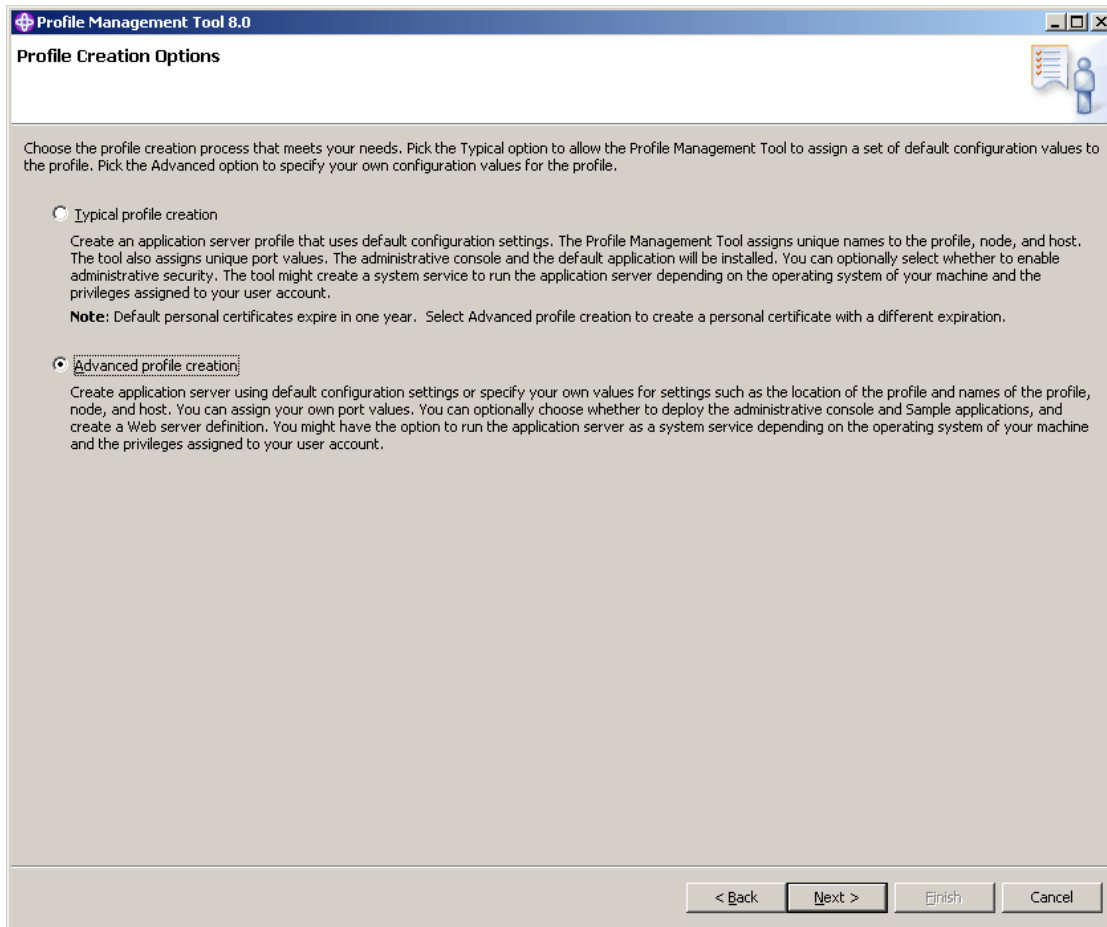


Figure 23-28 Advanced profile creation

7. In the Optional Application Deployment window (Figure 23-29), clear **Deploy the default application** and click **Next**.

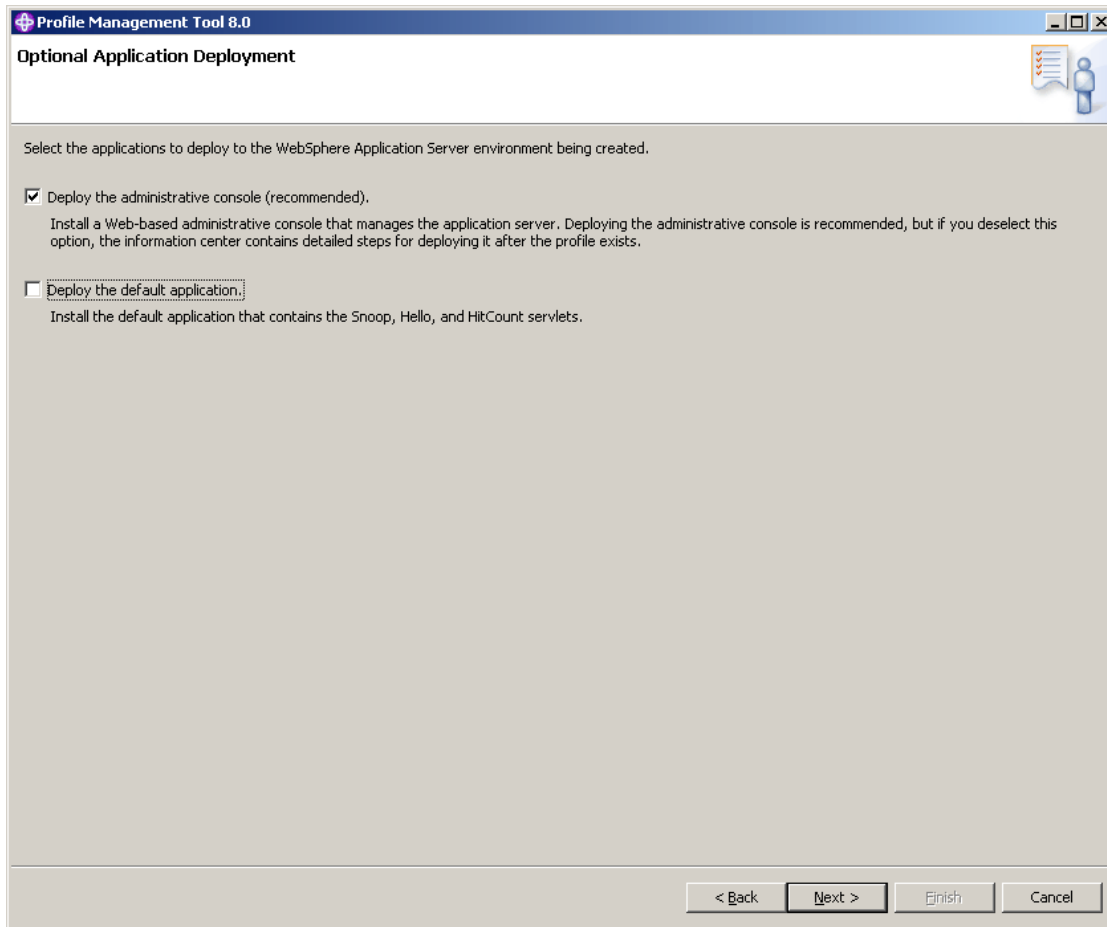


Figure 23-29 Optional Application Deployment



8. In the Profile Name and Location window (Figure 23-30), enter a profile name. For this example, we used the profile name AppSrv01.

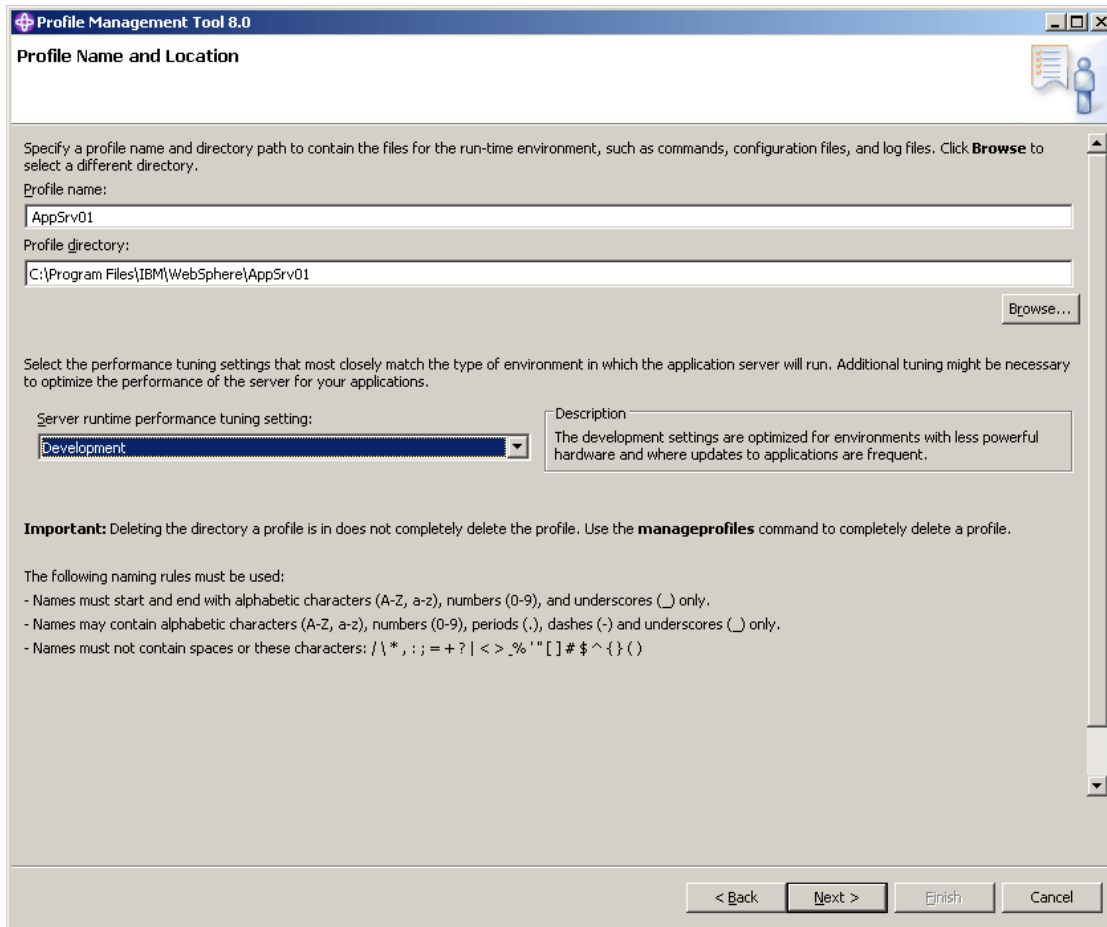


Figure 23-30 Profile Name and Location

9. To increase performance, choose the **Development** option in the Server runtime performance tuning setting (Figure 23-30 on page 1232) and click **Next**.
10. In the Node and Host Names window, you can use the default settings. Click **Next**.
11. In the Administrative Security window, enter a user name and password. Click **Next**.
12. In the Security Certificate (part 1) window, you can use the default settings. Click **Next**.

13. In the Security Certificate (part 2) window, you can use the default settings. Click **Next**.
14. In the Port Values Assignment window, you can use the default settings. Click **Next**.
15. In the Window Service definition window (or Linux Service definition window on Linux), clear **Run the application server process as a Windows service**. On Linux, **Run the application server process as a Linux service** is disabled, by default.
16. In the Web Server Definition window, you can use the default settings. Click **Next**.
17. In the Profile Creation Summary window, review the profile settings. Click **Next**.
18. In the Profile Creation Complete window, clear **Launch the First steps console** and click **Finish**.

Back in the Profile Management Tool window, you can see that the new profile is present in the Profiles list with the name AppSrv01.

When you close the Profile Management tool window and return to the Preferences window, you can also see that the new profile is listed under the “WebSphere profiles defined in the runtime selected above” list.

## 23.5.2 Deleting a WebSphere profile

If you need to delete a WebSphere profile, use the following steps.

**Important:** Do *not* perform these steps now, because we want to retain the newly created profile. These steps are provided here *only* for future reference.

1. From the menu bar of the workbench, select **Window** → **Preferences** → **Server** → **WebSphere Application Server**.
2. Under the WebSphere Application Server local server profile management list, select the installed runtime environment that contains the profile to be deleted.
3. Under the “WebSphere profiles defined in the runtime selected above” list, select the profile to be deleted and click **Delete**. The registry and configuration files associated with the profile selected for deletion are removed from the file system; however, any log files remain on the file system and can be removed manually.

### 23.5.3 Defining the new server in Rational Application Developer

After you create a WebSphere profile, you can define the new server in Rational Application Developer to use for the deployment of applications. A server definition points to a server that is defined within the specific selected WebSphere profile, such as the default profile created during installation or a profile created using the Profile Management Tool. Remember the following considerations regarding the definition of a new server in Rational Application Developer:

- ▶ The AppSrv01 profile was created in the previous section.
- ▶ A Rational Application Developer server definition is essentially a pointer to a WebSphere profile.

#### Creating a server definition in Rational Application Developer

To create a server definition in Rational Application Developer, follow these steps:

1. Select the **Servers** view in the Java Platform, Enterprise Edition (Java EE) or Web perspective.
2. Right-click in the **Servers** view and select **New** → **Server**.
3. In the Define a New Server window (Figure 23-31 on page 1248), follow these steps:
  - a. Leave the Server's host name field at its default value of localhost.

**Tip:** If you specify a remote host name in the Server's host name field, the New Server wizard prompts if you want to enable the server to start remotely. New for Rational Application Developer is support for starting a WebSphere Application Server on a remote computer. For more details, see the *Starting a remote WebSphere Application Server* topic:

[http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/com.ibm.servertools.doc/topics/tremote\\_start.html](http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/com.ibm.servertools.doc/topics/tremote_start.html)

- b. For the Server runtime environment, select **WebSphere Application Server v8.0 Beta** and click **Next**.

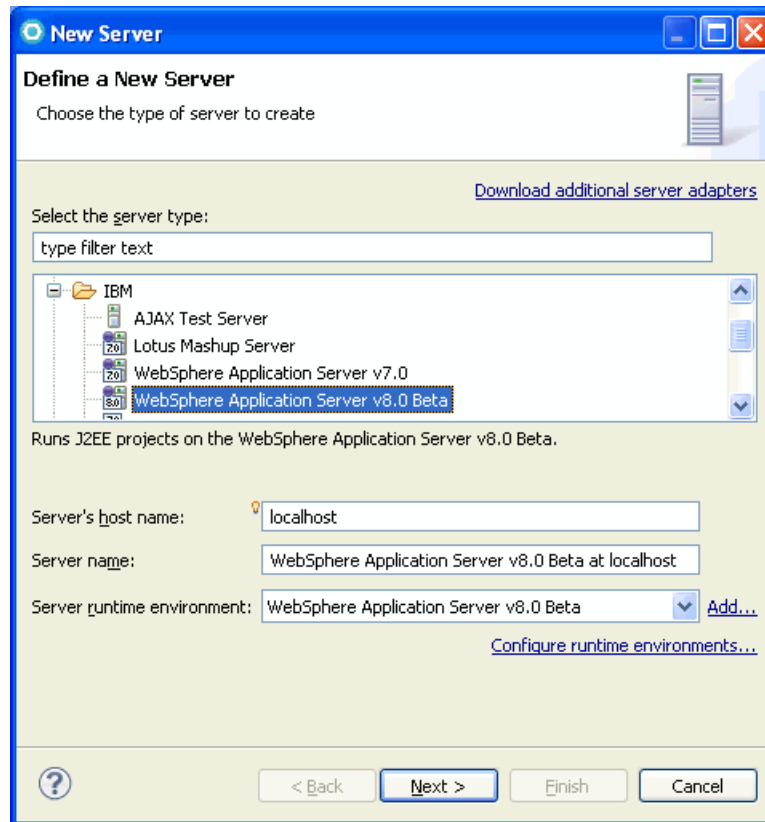


Figure 23-31 Define a New Server window

4. In the New Server: WebSphere Server Settings window (Figure 23-32 on page 1249), follow these steps:
  - a. For the WebSphere profile name, select the WebSphere profile that we created previously. The profile to select is **AppSrv01**.
  - b. Under Server connection types and administrative ports, select **Automatically determine connection settings**.
  - c. Select **Run server with resources within the workspace**.
  - d. Select **Security is enabled on this server**.
  - e. Enter the user and password that were configured in the profile creation.
  - f. For the Application server name, enter server1.
  - g. Click **Next**.

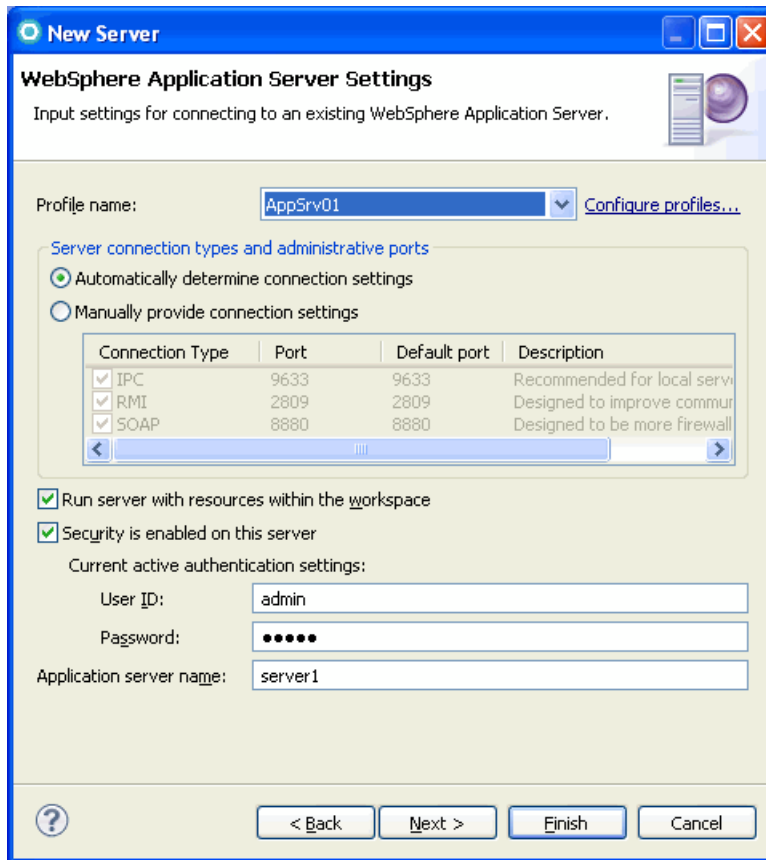


Figure 23-32 New Server: WebSphere Application Server Settings page

5. In the Add and Remove Projects window, click **Finish**. We do not select a project at this time.

The server definition is created and displayed in the Servers view. In our example, we created the server definition WebSphere Application Server v8.0 Beta at localhost.

**Tip:** Open the new server configuration by double-clicking it so that you can change the configuration settings. For example, you can change the name of the server, but we leave the name for now.

## Starting and stopping the server

Follow these steps to start the server:

1. In the Servers view, right-click **WebSphere Application Server v8.0 Beta at localhost** and select **Start**.
2. The status of the server changes to *Started*. If the status of the server does not change to Started, see the Console view for the log output. Verify if the server connection is correct.

For more details, see the *Setting the connection to the WebSphere Application Server* topic:

<http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/com.ibm.servertools.doc/topics/tsoapv6.html>

Follow these steps to stop the server:

1. In the Servers view, right-click **WebSphere Application Server v8.0 Beta at localhost** and select **Stop**.
2. The status of the server changes to *Stopped*.

### 23.5.4 Customizing a server

After the server has been defined in Rational Application Developer, you can easily customize its settings. To do this, in the Servers view, double-click the server that you want to customize. The server Overview window (Figure 23-33 on page 1251) opens in which you can modify the server configuration.

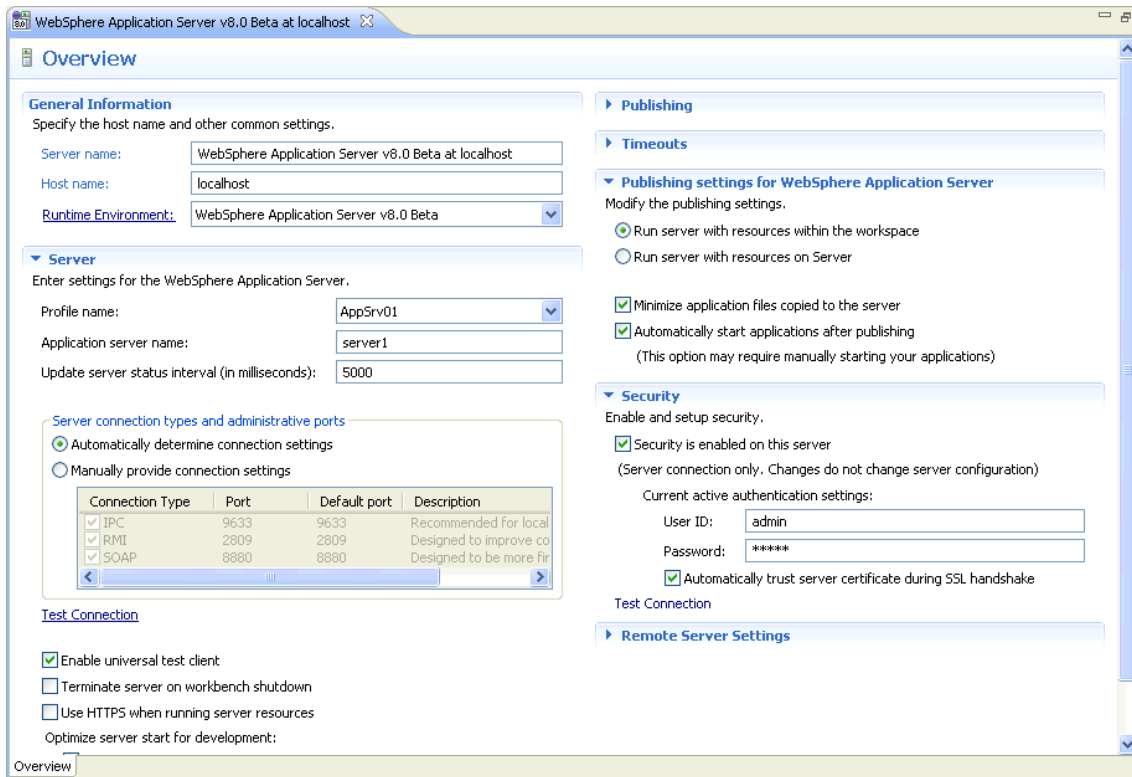


Figure 23-33 Server overview window

The following key settings are worth noting:

► Server:

- WebSphere profile name

Use this field to select the desired WebSphere Application Server profile.

- Server connection types and administrative ports

In this box, you select the method that is used by Rational Application Developer to communicate with the server. The radio buttons provide options for choosing the connection settings automatically or manually. If manual connection settings are chosen, you can select whether to use IPC, RMI, or SOAP as the communication channel between the development environment and the server.

By default, the automatic setting radio button is active. When working with a local server, IPC is the recommended connection setting for WebSphere Application Server V7.0 and v8.0 Beta, although all settings will work. If you are working with a remote server, SOAP or RMI can be used.

- Terminate server on workbench shutdown

If you want the server to be terminated after the workbench is shut down, select this option. Otherwise, the server continues to run after you shut down the development environment. The next time that you start the integrated development environment (IDE), the server is found again in its current state.

- ▶ Publishing:

- Never publish automatically

Select this option to specify that the workbench must not automatically publish files to the server. A developer can still publish to the server, but the developer must publish to the server manually.

- Automatically publish when resources change

Select this option to specify that a change to the files running on the server must automatically be published to the server. The “Publishing interval (in seconds)” option specifies how often the publishing takes place. If you set the publishing interval to 0 seconds, a change to the files running on the server automatically causes publishing to occur.

- ▶ Publishing settings for WebSphere Application Server:

- Run server with resources on Server

This option installs and copies the full application and its server-specific configuration from the workbench to the directories of the server. The advantage of selecting this option is that you run your application from the directories of your server and edit advanced application-specific settings for your application by using the WebSphere administrative console. However, when you choose to add your application to the server by using the Add and Remove Projects wizard, this option takes a longer time to complete than the “Run server with resources within the workspace” option, because it involves more files being copied to the server.

- Run server with resources within the workspace

Use this option to request the server to run your application from the workspace. This setting is useful when developing and testing your application. It is designed to operate faster than the “Run server with resources on Server” option, because fewer files are involved when copying over to the server.

- Minimize application files copied to the server

Use this option to optimize the publishing time on the server by reducing the files copied to the server. In addition to the application files not being copied into the `installedApps` directory of the server, the application is also not copied into your server configuration directory. As a result, you



cannot use the WebSphere administrative console to edit the deployment descriptor. Clear this option if you must use the WebSphere administrative console to edit the deployment descriptor.

- ▶ Security

We discuss the security settings in 23.9, “Configuring security” on page 1268.

- ▶ Remote Server setting

New for Rational Application Developer is support for starting a WebSphere Application Server on a remote machine. For more details, see the *Starting a remote WebSphere Application Server* topic:

[http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/com.ibm.servertools.doc/topics/tremote\\_start.html](http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/com.ibm.servertools.doc/topics/tremote_start.html)

## 23.5.5 Sharing a WebSphere profile between developers

The configuration of a server can be time consuming and error-prone. After you configure the server resources, you might want to let other members of the team use the same configuration from their local environments, without duplicating the same effort.

To replicate server configurations across multiple profiles, you can use the Server Configuration Backup wizard to create a backup of a WebSphere Application Server v8.0 Beta profile in a configuration archive (CAR) file. This wizard performs the same functionality as the **wsadmin** command:

```
wsadmin -user user -password password -c “$AdminTask exportWasprofile
{-archive configuration_archive_path}”
```

Be sure to have the server started, and if security was enabled, be sure to enter the security credentials when prompted. You can also use the Server Configuration Restore wizard to restore a WebSphere Application Server v8.0 Beta profile from a CAR file. This wizard performs the same functionality as the **wsadmin** command:

```
wsadmin -user user -password password -c “$AdminTask importWasprofile
{-archive configuration_archive_path}”
```

### Backing up the server configuration

To back up the server configuration from WebSphere Application Server v8.0 Beta at the localhost, follow these steps:

1. Configure the data source using WebSphere Application Server v8.0 Beta, as described in “Configuring the data source in WebSphere Application Server” on page 1882.

2. Create a new project to store the configuration archive file:
  - Select **File** → **New** → **Project**.
  - On the New Project dialog window, Select **General** → **Project** and click **Next**.
  - For the project name, enter WAS80Car and click **Finish**.
3. In the Servers view, right-click **WebSphere Application Server v8.0 Beta at localhost** and select **Server Configuration** → **Backup**.
4. In the Server Configuration Backup window (Figure 23-34), for the Parent folder, enter /WAS80Car. For the File name, enter WAS80AppSrv1. Then click **OK**.

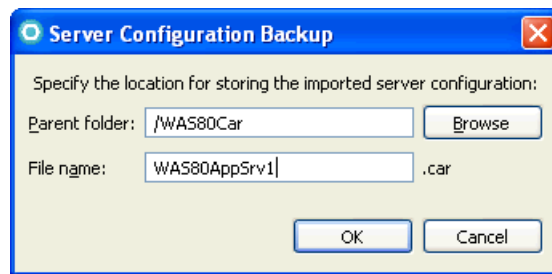


Figure 23-34 Server Configuration Backup window

After the server configuration backup process completes, you see the WAS80AppSrv1.car file under the WAS80Car project.

## Restoring the server configuration

To restore the server configuration to a separate server, in this case, WebSphere Application Server v8.0 Beta at the localhost, follow these steps:

1. Make sure that WebSphere Application Server v8.0 Beta at the localhost is not running. If it is running, stop it.
2. In the Servers view, right-click **WebSphere Application Server v8.0 Beta at localhost** and select **Server Configuration** → **Restore**.
3. In the Server Configuration Restore window (Figure 23-35 on page 1255), for the Parent folder, enter /WAS80Car. For the File name, enter WAS80AppSrv1.car. Then click **OK**.

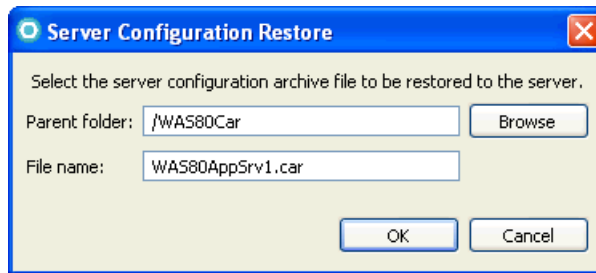


Figure 23-35 Server Configuration Restore window

4. After the server configuration restore process completes, start **WebSphere Application Server v8.0 Beta at localhost**.

The server is configured with the CAR file data.

### 23.5.6 Defining a server for each workspace

To switch workspaces and keep applications deployed on the server, you can create a new WebSphere profile for each new Rational Application Developer workspace, as explained in 23.5.1, “Creating a new profile using the WebSphere Profile wizard” on page 1240. Then define that server in the Servers view, as explained in “Creating a server definition in Rational Application Developer” on page 1247.

This approach requires more disk space for each WebSphere profile and server configuration. It also requires additional memory if the servers run concurrently.

## 23.6 Migrating the server resources from Rational Application Developer V7.0 or V7.5 to V8.0

You can migrate server resources that were configured in previous versions of Rational Application Developer, such as V7.0 or V7.5, to Rational Application Developer v8.0. *Server resources* are files with information that is required to set up and publish artifacts from the workbench to a server.

Refer to the following information center for more help performing this task:

<http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.servertools.doc/topics/tmigserverview.html>

## 23.7 Adding and removing applications to and from a server

After the server is configured, you can configure it further with server resources and use it to run applications by adding applications to it.

In this section, we explain how to add an enterprise application to a server. You can only add enterprise application EAR projects, not web or EJB projects, to a server.

### 23.7.1 Adding an application to the server

To add an application to the server, follow these steps:

1. Verify that the server has started.
2. In the Servers view, right-click a server and select **Add and Remove Projects**.
3. In the Add and Remove Projects window (Figure 23-36 on page 1257), select one of the listed EAR projects and click **Add**. After you click Add, the project is displayed in the Configured projects box. Click **Finish**.

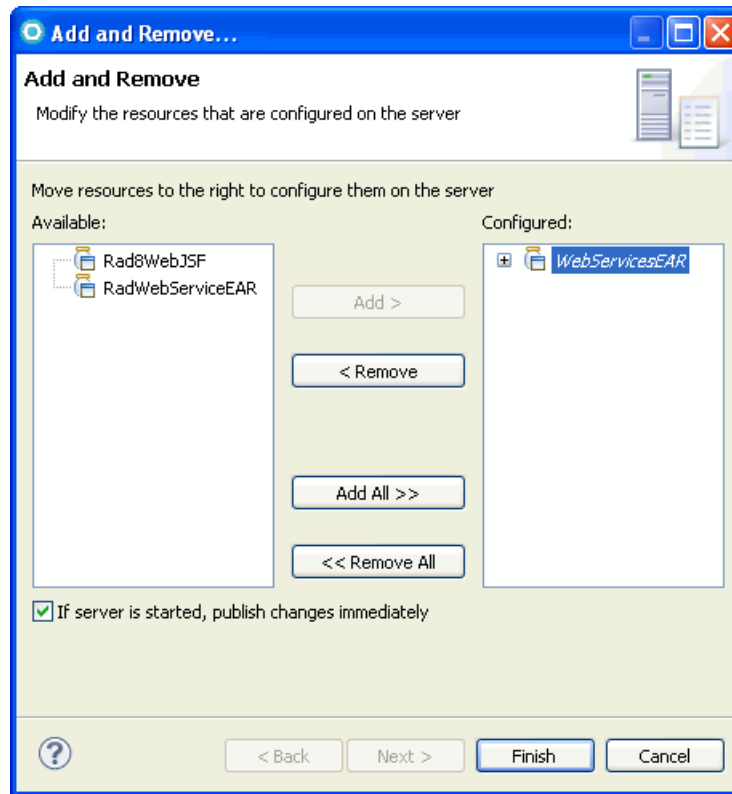


Figure 23-36 Add and Remove Projects window

After an application is added to the server, you can run any of the HTML pages or JavaServer Pages (JSP).

## 23.7.2 Removing an application from a server

A Rational Application Developer server configuration is essentially a pointer to a server defined in a WebSphere profile. In this section, we present two scenarios for removing published projects from the server.

### Removing an application using Rational Application Developer

In most cases, you can remove a project from the test server within Rational Application Developer using either of the following options:

- ▶ In the Servers view, right-click the server where the application is published and select **Add and remove projects**. In the Add and Remove Projects

window, select the project in the Configured projects list, click **Remove**, and click **Finish**.

- ▶ Expand the server in the Servers view, right-click the EAR project to be removed and select **Remove**.

Either option uninstalls the application from the server.

### Removing an application using the administrative console

In certain cases, you might decide to uninstall the application by using the WebSphere administrative console. For example, if you published a project in Rational Application Developer to the test server, it is deployed to the server that is defined in the WebSphere profile. If you then switch workspaces without first removing the project from the server, you break the association between Rational Application Developer and the server.

To address this scenario, uninstall the enterprise application from the WebSphere administrative console:

1. Start the WebSphere administrative console by right-clicking the server and selecting **Administration** → **Run administrative console**. If necessary, click **Log in**.
2. Select **Applications** → **Application Types** → **WebSphere Enterprise Applications**.
3. Select the desired application to uninstall and click **Uninstall**. When prompted, click **OK**.
4. When the uninstallation is complete, save the changes.

**Tip:** After you uninstall the application using the WebSphere administrative console, if you start the Add and Remove Projects wizard, you can see that the project is still displayed in the Configured projects section. Click **Remove** to move the project to the Available projects section.

## 23.8 Configuring application and server resources

In Rational Application Developer, you can define application-related properties and data sources within a WebSphere specific deployment plan called *Enhanced* (Figure 23-38 on page 1259). It simplifies the application deployment by defining the configurations in the enterprise archive that are automatically deployed on the server when the application is deployed.

**Note:** In order to have the enhanced EAR available, the project must have the WebSphere Application Server v8.0 facet enabled. To enable this facet, open the enterprise project properties and select **Project Facets**. In the Project Facets configuration, select the **WebSphere Application (Co-existence)** and **WebSphere Application (Extended)** facets Version 8.0 Beta, as shown in Figure 23-37.

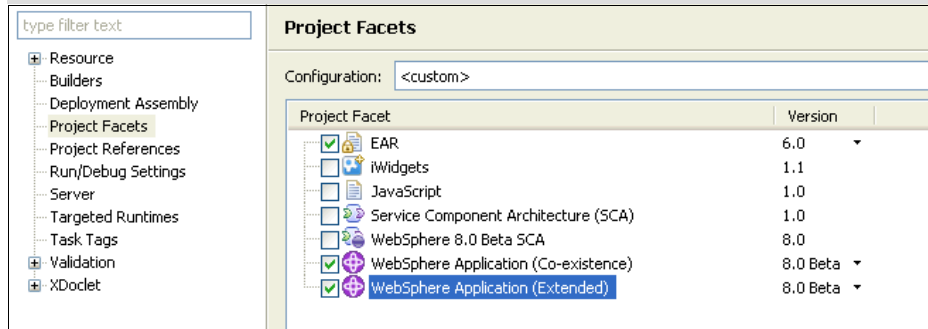


Figure 23-37 WebSphere facets for a EAR project

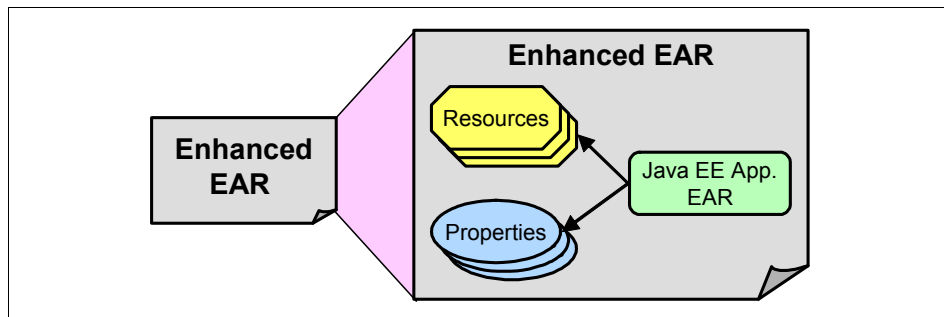


Figure 23-38 Enhanced EAR file

The enhanced EAR tooling is available in the WebSphere Application Server Deployment Editor, as shown in Figure 23-39 on page 1261. Deployment information is saved under the application `/META-INF/ibmconfig` directory.

This file is not generated by default when creating an EAR project. In order to generate it, follow the steps:

1. Right-click over your EAR project in the Enterprise Explorer view.
2. Select **Java EE** → **Open WebSphere Deployment Descriptor**.

**Enhanced EAR editor:** You use the Enhanced EAR editor to edit several WebSphere Application Server v8.0 specific configurations, such as data sources, class loader policies, substitution variables, shared libraries, virtual hosts, and authentication settings. With this editor, you can configure these settings with little effort and publish them every time that you publish the application.

The advantage of using the Enhanced EAR editor is that it makes the testing process simpler and easily repeatable, because the configurations are saved to files that are usually shared in the team repository. Thus, although you cannot configure all the runtime settings, the tool is beneficial for development purposes, because it eases the process of configuring the most common settings.

The disadvantage of this editor is that the configuration settings are stored in the EAR file and are not visible from administrative console. You can use the console to only edit settings that belong to the cluster, node, and server contexts. When you change a configuration using the Enhanced EAR editor, the change is made at the application context. Furthermore, in most cases, these settings depend on the node in which the application server will be installed anyway. Therefore, it is not necessary configure them at the application context for deployment purposes.



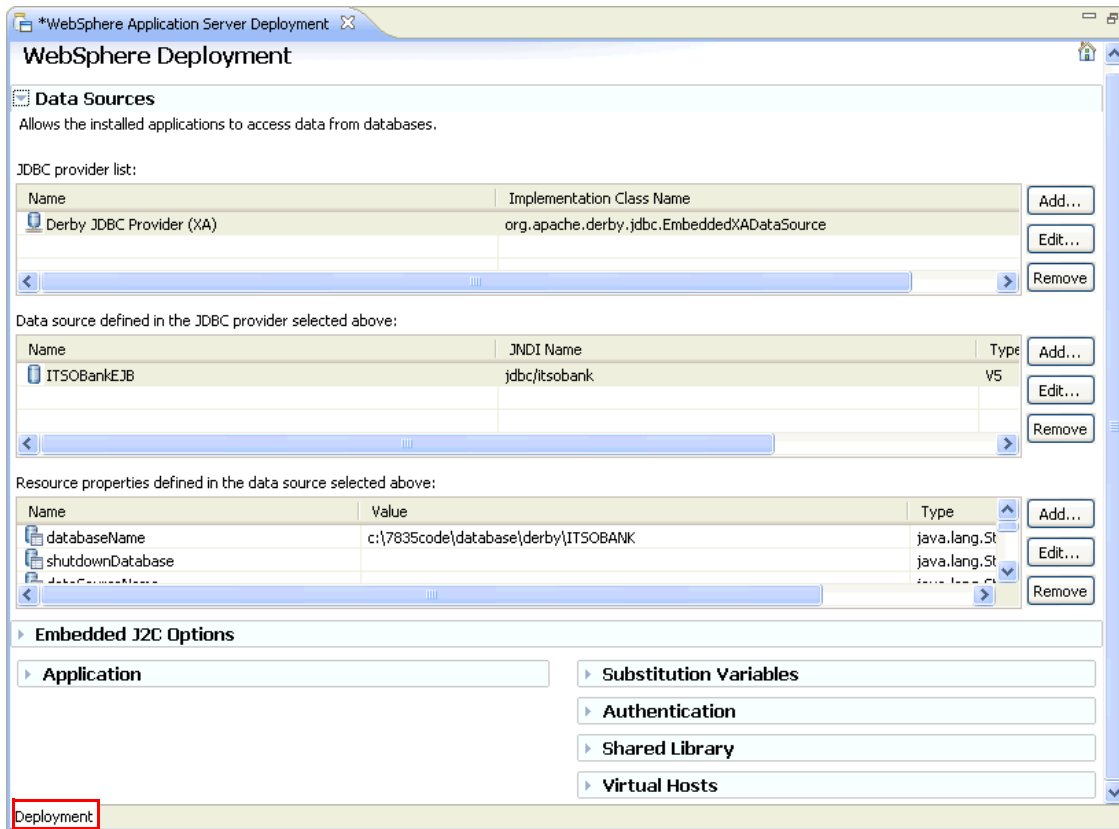


Figure 23-39 Enterprise application deployment descriptor: Enhanced EAR

The server configuration data that you specify in the Enhanced EAR editor is embedded within the application. The embedded data improves the administration process of publishing to WebSphere Application Server v8.0 Beta when installing a new application to an existing local or remote WebSphere Server by preserving the existing server configuration.

You can add the following resource types to the enhanced EAR file:

- ▶ Application class loader settings (Application section)
- ▶ Java Authentication and Authorization Service (JAAS) authentication entries
- ▶ Java Database Connectivity (JDBC) resources (Data Sources section)
- ▶ Resource adapters (Embedded J2EE Connector (J2C) architecture Options section)
- ▶ Shared libraries

- ▶ Substitution variables
- ▶ Virtual hosts

## 23.8.1 Creating a data source in the Enhanced EAR editor

You must specify the data sources that support EJB entity beans before the application can be started. You can specify the data sources using one of several ways, but the easiest way is to use the Enhanced EAR editor.

For an example of configuring the enhanced EAR against the Derby database, see “Configuring the data source for the ITSOBANK” on page 609. In this section, we explain how to create a data source for a DB2 database using enhanced EAR settings:

1. In the Enterprise Explorer, right-click **RAD8EJBEAR** and select **Java EE** → **Open WebSphere Application Server Deployment**.
2. Scroll down the window until you find the **Authentication** section, in which you can define a login configuration that is used by JAAS. Click **Add** to create a new configuration.
3. In the Add JAAS Authentication Entry window (Figure 23-40), for the Alias, type `dbuser`. Enter the user ID and password for your configuration. Click **OK** to complete the creation of the configuration.

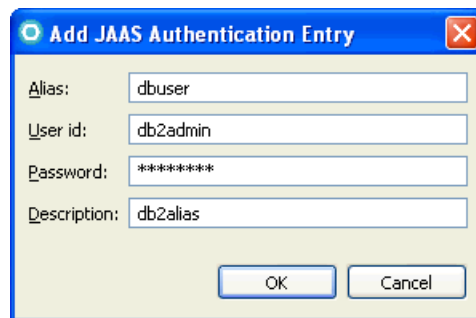


Figure 23-40 JAAS Authentication Entry

4. In the Enhanced EAR editor, scroll back up to the Data Sources, JDBC provider list section. By default, the **Derby JDBC Provider (XA)** is predefined.
5. Because we use DB2 for this example, click **Add** (next to the provider list) to add a DB2 JDBC provider.

6. In the Create JDBC Provider window (Figure 23-41 on page 1263), complete these steps:
  - a. For the Database type, select **IBM DB2**.
  - b. For the JDBC Provider type, select **DB2 Universal JDBC Driver Provider (XA)** and click **Next**.

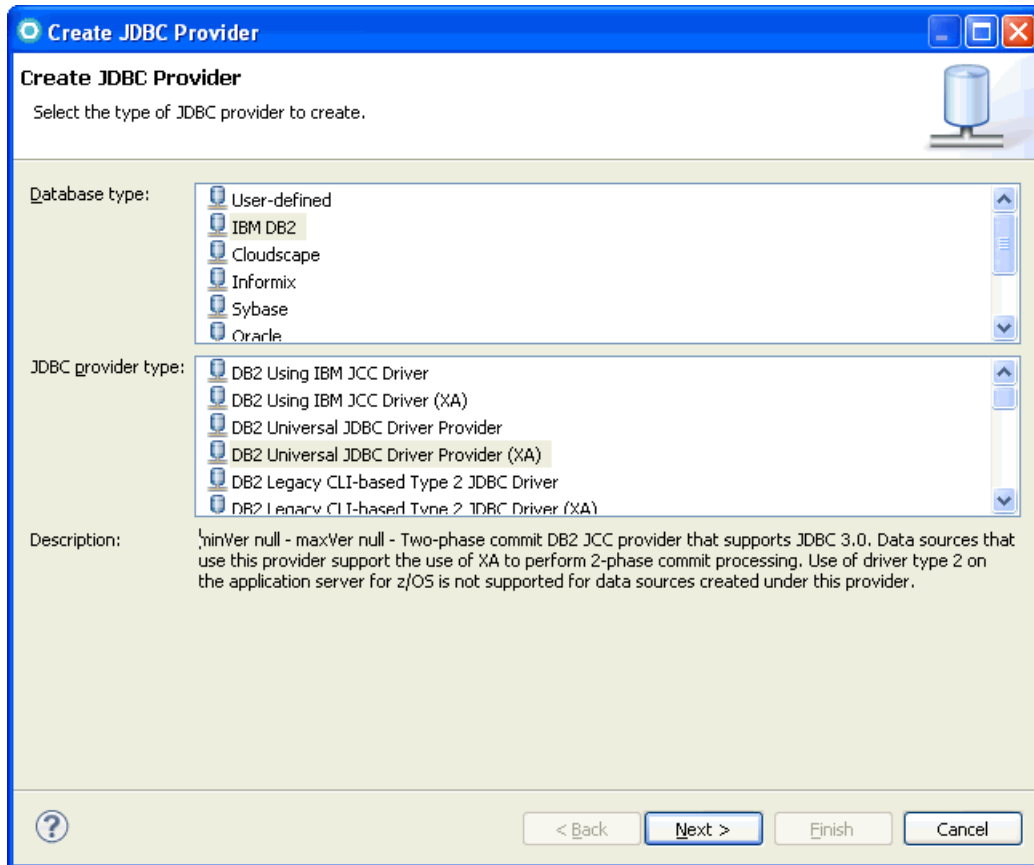


Figure 23-41 Create JDBC Provider window (part 1 of 2)

**DB2 Universal JDBC Driver Provider:** For our development purposes, we can use the DB2 Universal JDBC Driver Provider (non-XA), because we do not require XA (two-phase commit) capabilities.

7. In the next Create JDBC Provider window (Figure 23-42), follow these steps:
  - a. For the Name, type DB2 XA JDBC Provider.  
 Notice the variables that are used to locate the JDBC driver:
 

```

 ${DB2UNIVERSAL_JDBC_DRIVER_PATH}
 ${UNIVERSAL_JDBC_DRIVER_PATH}
 ${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}

```

 If these variables are not set globally for the WebSphere Application Server, you can set them under Substitution Variables.
  - b. Click **Finish**.

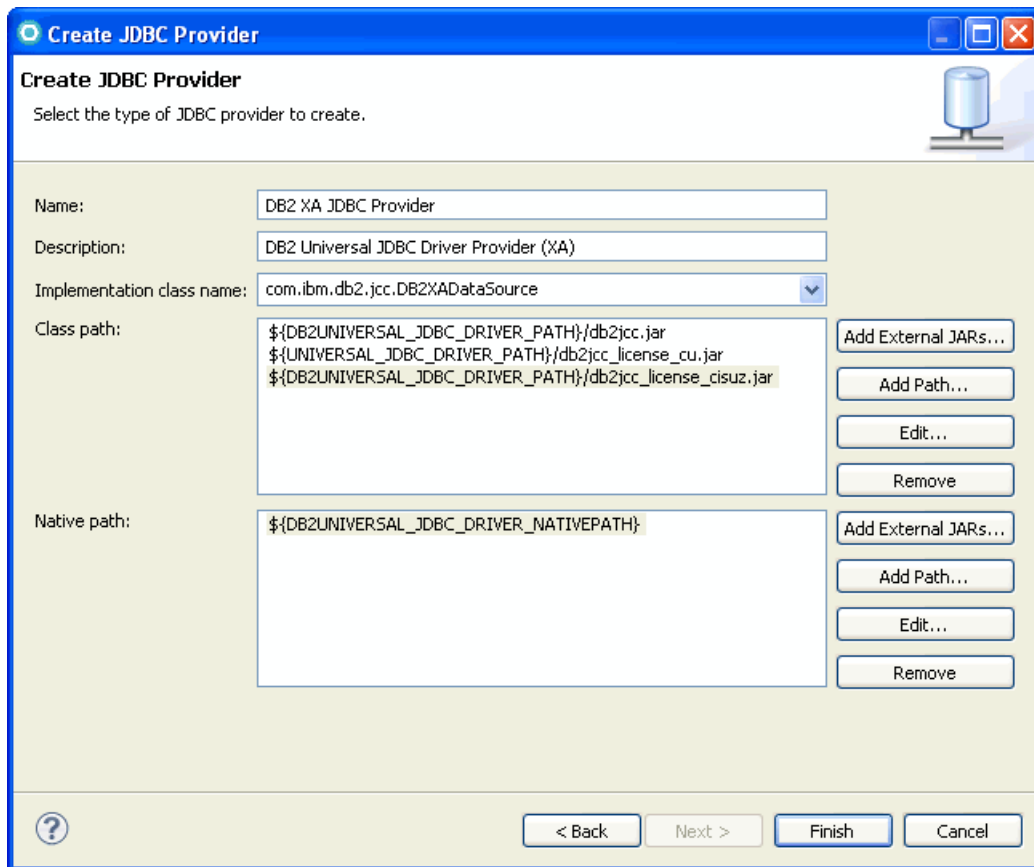


Figure 23-42 Create JDBC Provider window (part 2 of 2)

8. With the new DB2 provider selected, click **Add** next to the defined data sources list.

9. In the Create Data Source window (Figure 23-43), complete these actions:
  - a. From the JDBC provider type list, select **DB2 Universal JDBC Driver Provider (XA)**.
  - b. Select **Version 5.0 data source**.
  - c. Click **Next**.

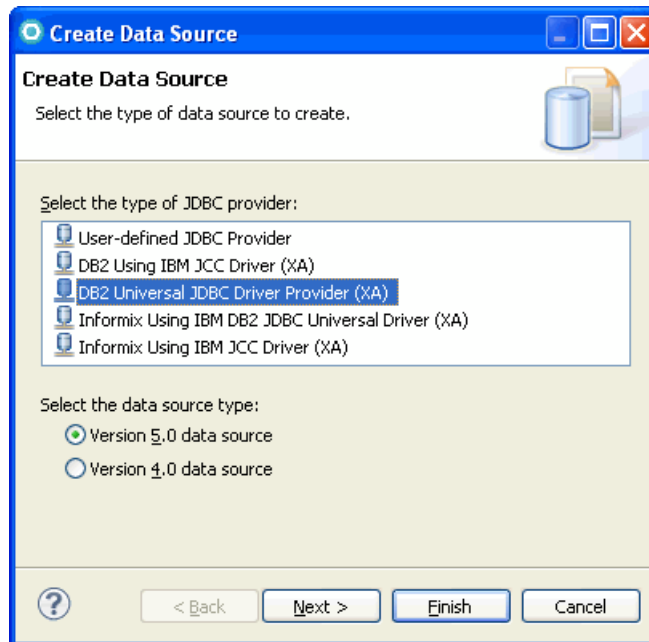


Figure 23-43 Create Data Source window (part 1 of 3)

10. In the next Create Data Source window (Figure 23-44 on page 1266), complete these steps:
  - a. For the Name (of the data source), type RAD8DS. For the Java Naming and Directory Interface (JNDI) name, type jdbc/itsobankdb2.
  - b. For the Component-managed authentication alias, select **dbuser**.
  - c. Clear **Use this data source in container managed persistence (CMP)**. We are using Java Persistence API (JPA) entities, not EJB 2.1 entity beans.
  - d. Click **Next**.

**Create Data Source**

Select the type of data source to create.

Name: RAD8D5

JNDI name: jdbc/itsobankdb2

Description: DB2 Universal Driver Datasource

Category:

Statement cache size: 10

Data source helper class name: com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper

Connection timeout: 180

Maximum connections: 10

Minimum connections: 1

Reap time: 180

Unused timeout: 1800

Aged timeout: 0

Purge policy: EntirePool

Component-managed authentication alias: dbuser

Container-managed authentication alias:

Use this data source in container managed persistence (CMP)

\* Required field.

< Back   Next >   Finish   Cancel

Figure 23-44 Create Data Source window (part 2 of 3)

11. In the last Create Data Source window (Figure 23-45), set the `databaseName` property value to `ITSOBANK`. Then click **Finish** to conclude the wizard.

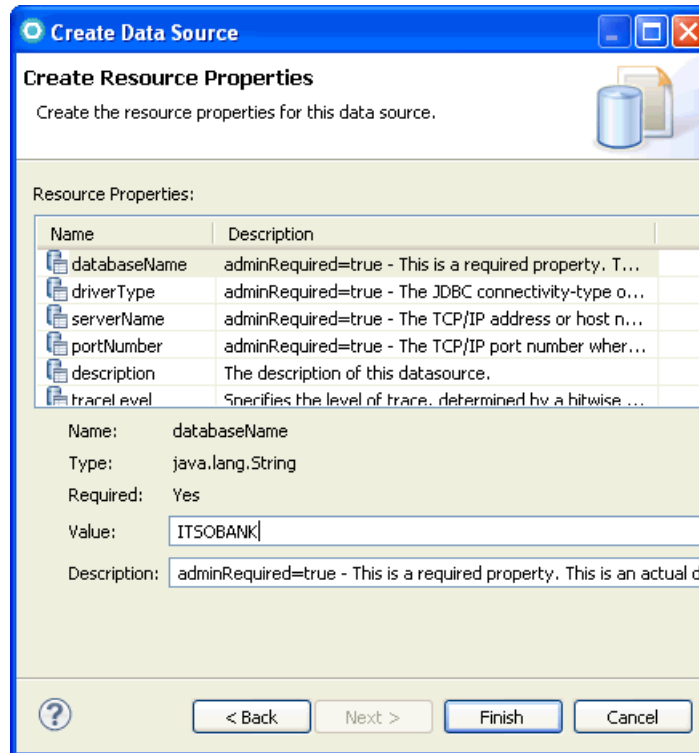


Figure 23-45 Create Data Source window (part 3 of 3)

12. Save the deployment descriptor.

## 23.8.2 Setting the substitution variable

If the variables to access the DB2 JDBC drivers are not set server-wide (using the administrative console), you can expand the Substitution Variable section and define the variables:

1. Click **Add**.
2. In the Add Variable window, define the name of the variable (for example, `${DB2UNIVERSAL_JDBC_DRIVER_PATH}`) and set the value to the location where the DB2 JDBC drivers are installed, for example, `c:\SQLLIB\java`.

### 23.8.3 Configuring server resources

Within Rational Application Developer, the WebSphere administrative console is the primary interface for configuring both local and remote servers of WebSphere Application Server v8.0 Beta. You can only configure complicated resource configurations, such as messaging resources, by using the WebSphere administrative console.

Start **WebSphere Application Server v8.0 Beta at localhost**. And after the test server starts, right-click the server and select **Administration** → **Run administrative console**.

## 23.9 Configuring security

During the WebSphere Application Server v8.0 Beta at localhost profile creation, the administrative security was enabled. In order to allow Rational Application Developer to manage the server, you have to specify that security is enabled in the runtime environment and provide the user ID and password in the server editor for the secured server. We have already done this part during the creation of the server.

However if you are working with a new secured WebSphere Application Server server or you change the authentication method, you must establish a trust between the Rational Application Developer development environment and the server.

**More information:** For more information about configuring WebSphere Application Server security, see *IBM WebSphere Application Server V6.1 Security Handbook*, SG24-6316.

In this section, we show how to enable administrative security, using the local operating system registry for authentication, by using the WebSphere administrative console. We also show you how to pass the administrative settings from the development environment to the runtime server.

### 23.9.1 Configuring security in the server

Follow these steps to configure the security using the server administrative console:

1. Start the test server called **WebSphere Application Server v8.0 Beta at localhost**.



2. Right-click the server and select **Administration** → **Run administrative console**.
3. Click **Log in**.
4. Expand **Security** → **Global Security** (Figure 23-46):
  - a. Select **Enable administrative security** and click **Enable application security**.
  - b. Clear **Use Java 2 security to restrict application access to local resources**.

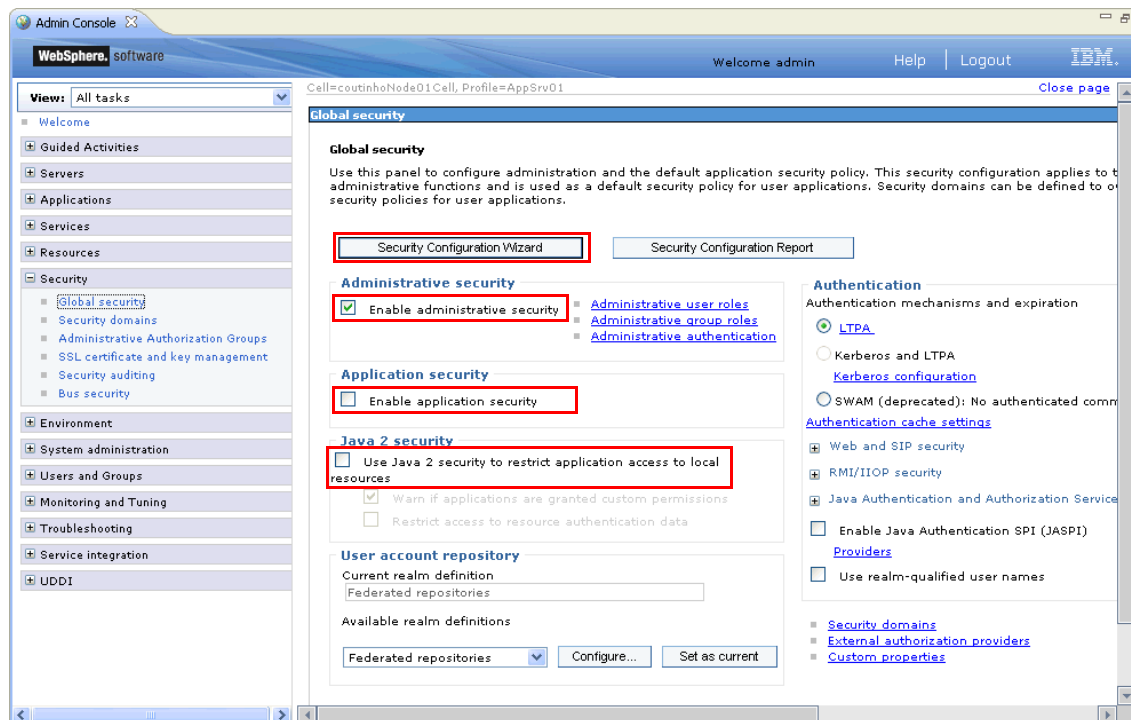


Figure 23-46 Configuring security

5. Click **Security Configuration wizard**.
6. In the Specify extent of protection window, click **Next**.
7. In the Select user repository window, select **Local operating system**.
8. In the Configure user repository window, enter the primary administrative user name and click **Next**.

**Required privileges:** The specified user must have the required privileges, such as the permission to log on as a service, in Windows. For more information, see “Required privileges in Windows” in *IBM WebSphere Application Server V6.1 Security Handbook*, SG24-6316.

9. Click **Finish** and then click **Save**.
10. Click **Logout** to log off from the administrative console.
11. Stop the server.

## 23.9.2 Configuring security in the workbench

Edit the server configuration to specify that security is enabled:

1. In the Servers view, double-click **WebSphere Application Server v8.0 Beta at localhost**.
2. In the server configuration editor (Figure 23-47), complete the following steps:
  - a. Expand the **Security** section.
  - b. Select **Security is enabled on this server**.
  - c. Verify that the values in the User ID and Password fields are the same as those values entered in the Security Configuration wizard window, as explained in “Configuring security in the server” on page 1268. The User ID and Password fields specify the administrator user of the WebSphere administrative console.
  - d. Select **Automatically trust server certificate during SSL handshake**.



Figure 23-47 Security settings in the server editor

3. Save and close the server configuration editor.
4. Start the server and run the administrative console.

5. In the secured administrative console (Figure 23-48), enter the user ID and password. Then click **Log in**.

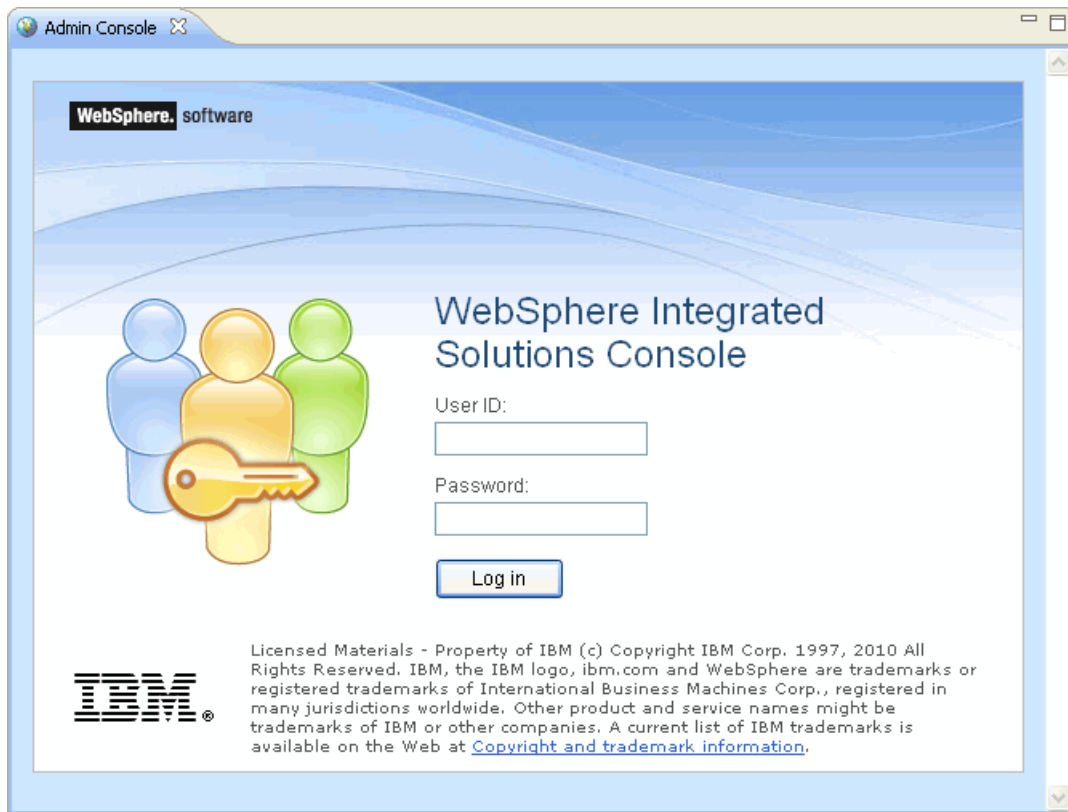


Figure 23-48 Secured administrative console

## 23.10 AJAX Test Server

The AJAX Test Server is a lightweight server that is suited for developing and testing AJAX applications. The server performs module publishing and server restart and contains an AJAX proxy that creates requests to remote domains.

### 23.10.1 Configuring the AJAX Test Server

The following steps create and configure a new AJAX Test Server Configuration:

1. Right-click in the **Servers** view and select **New** → **Server**.

2. In the New Server dialog window, select the **AJAX Test Server** under the **IBM** folder and click **Finish** (Figure 23-49).

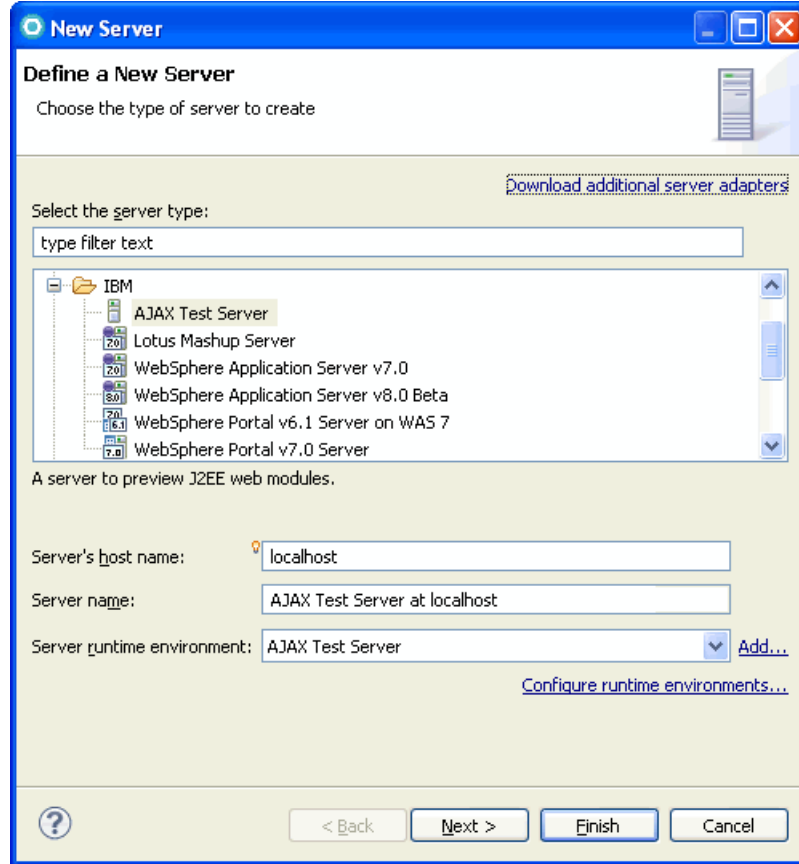


Figure 23-49 New AJAX Test Server

3. The new AJAX Test Server is created. By double-clicking it in the Servers view, its configuration is available, which is similar to any other servers, as shown in Figure 23-50 on page 1273.

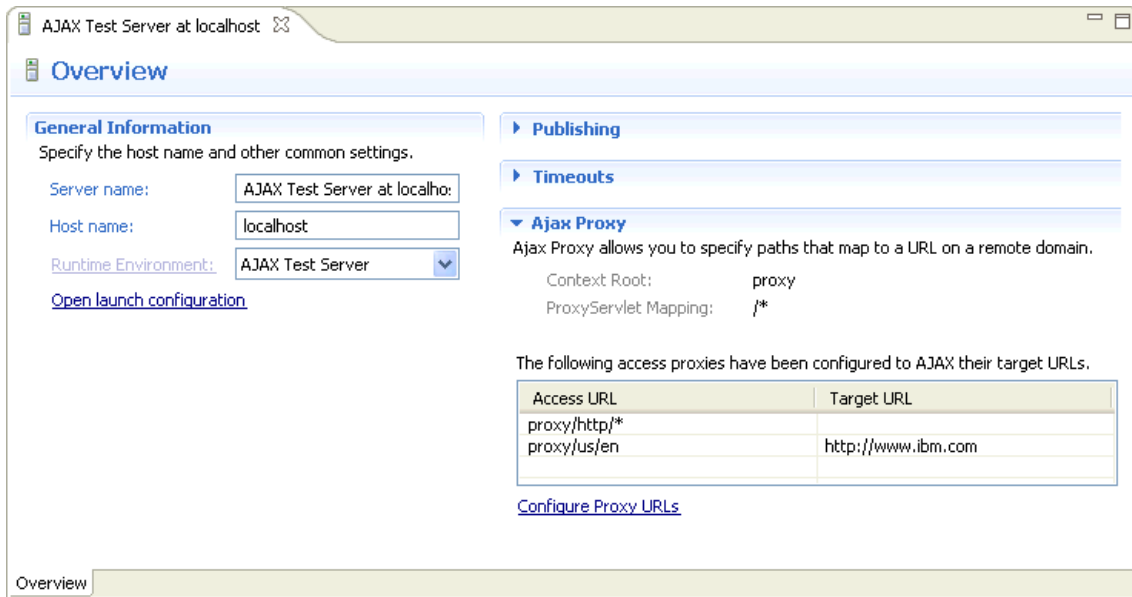


Figure 23-50 AJAX Test Server configuration

## 23.10.2 Configuring the AJAX Proxy

A powerful use of the AJAX servers is to use it as a proxy server for developing client-side Web 2.0 applications. This development usually only consumes web services output that is already deployed on an external server. To avoid deploying the server side on local development machines, it is better to use the AJAX Proxy to redirect the requests to the remote server.

To configure the AJAX Proxy, open the **AJAX Test Server** configuration and click the link **Configure Proxy URLs**.

The AJAX Proxy Configuration Editor window opens, as shown in Figure 23-51 on page 1274.

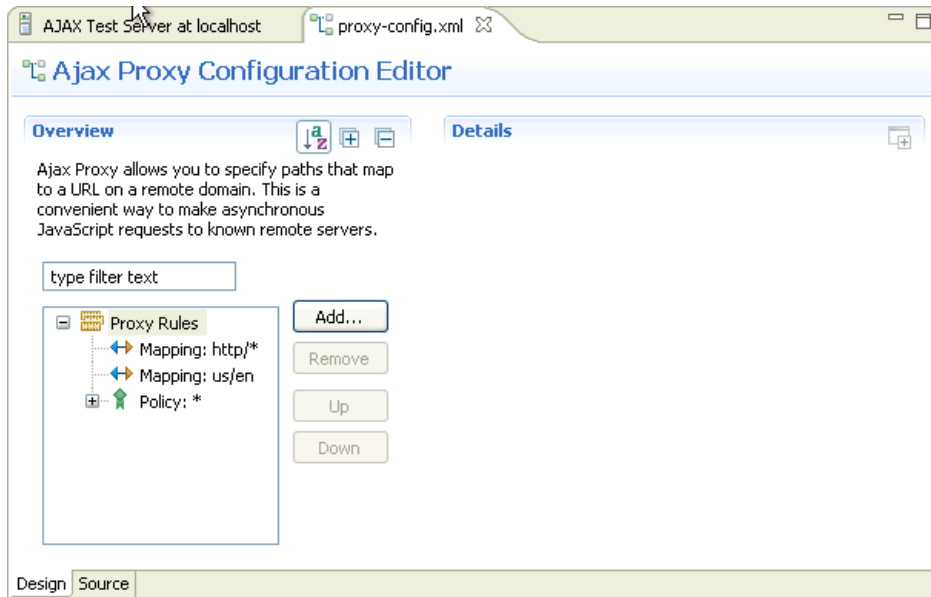


Figure 23-51 AJAX Proxy Configuration Editor

This editor allows you to add the following proxy rules:

### Mapping rules

A *mapping* specifies a local context path, which will be serviced by an external URL. For example, if you set the Context Path to `br/pt` and the URL to `http://www.ibm.com` server, all the requests to `http://localhost:8080/proxy/br/pt` are going to be redirected to the `www.ibm.com` server.

### Metadata

The *metadata elements* add proxy configuration parameters. For example, you can set the following metadata name `retry` with the value `3` to configure the proxy to retry three times when having problems trying to reach a server.

### Access policy

You use the `policy` element to define an access policy for a specific URL pattern. For example, you can configure a policy that only allows you to make POST HTTP requests to a URL, as shown in Figure 23-52 on page 1275. If any HTTP GET request is issued to that URL mapping, the response is HTTP 403 error.

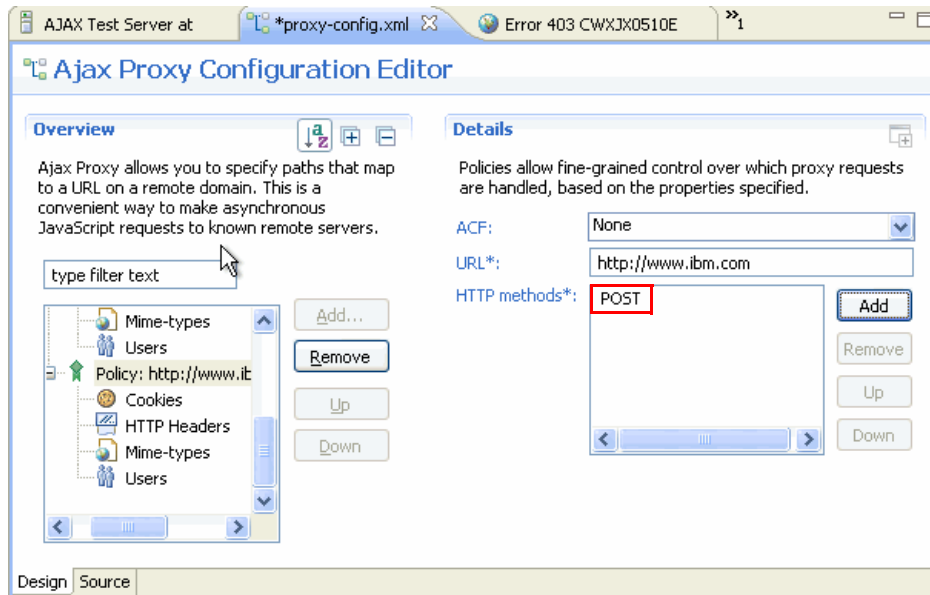


Figure 23-52 Restricting the proxy to only POST HTTP requests

## 23.11 Developing automation scripts

*Scripting* is a non-graphical alternative that you can use to configure and manage WebSphere Application Server. The WebSphere administrative scripting tool, **wsadmin**, is a non-graphical command interpreter environment with which you can run administrative operations on a server in a scripting language.

For more information about how to create automation scripts for an application and how to execute the scripts, refer to 25.5, “Automated deployment using Jython-based wsadmin scripting” on page 1345.

## 23.12 Tips: Enhancing server interaction performance

This section describes tips to speed up server start time and application publishing time.

## 23.12.1 Speeding up server start time

Use these tips to speed up server start time:

- ▶ Customize a server (see 23.5.4, “Customizing a server” on page 1250) to select **Start server with a generated script** in the Server section. This action speeds up the server start time if there are no applications added to the server.
- ▶ Reduce the number of unnecessary applications on the server by removing unused user applications from the server.

## 23.12.2 Speeding up application publishing time

Use these tips to speed up application publishing time:

- ▶ Customize a server (see 23.5.4, “Customizing a server” on page 1250) to select **Run server with resources within the workspace** and **Minimize application files copied to the server** in the Publishing settings for WebSphere Application Server section. This action eliminates the need to publish for changes in static files, such as HTML and JSP files. This action also speeds up the application installation and application update.
- ▶ Customize a server (see 23.5.4, “Customizing a server” on page 1250) to select **Never publish automatically**, and group the application changes in one manual publish process if the class files are changed frequently. This action reduces the number of automatic publishes when resources change.
- ▶ Start the server in debug mode to eliminate the need to republish for certain class file changes.
- ▶ Select the **Development** setting if manually creating a new WebSphere Application Server profile (see 23.5.1, “Creating a new profile using the WebSphere Profile wizard” on page 1240). The Development profile has specific tuning that speeds up the server interaction performance. Also, do not install the default applications, and do not create the profile to start as a Microsoft Windows service.
- ▶ Avoid using non-single-root project structure, because it increases publishing time significantly. A Project Structure Marker warning message shows up in the Problems view or Markers view if the project is not single-root: *“Broken single-root rule: A root folder may not contain linked resources”*.
- ▶ Avoid long start-up operations to speed up application start-up time; for example, having long running processes in the servlet’s start-up code can greatly increase the whole application’s start-up time.



## 23.13 More information

For more information, consult the following IBM Redbooks publications:

- ▶ *WebSphere Application Server V6.1: System Management and Configuration*, SG24-7304
- ▶ *IBM WebSphere Application Server V6.1 Security Handbook*, SG24-6316





# Building applications with Apache Ant

Traditionally, you build applications by using UNIX or Linux shell scripts or Microsoft Windows batch files in combination with tools, such as *make*. Although these approaches are still valid, new challenges exist when developing Java applications, especially in a heterogeneous environment. Traditional tools are limited in that they are closely coupled with a particular operating system. With Apache Ant, you can overcome these limitations and perform the build process in a standardized fashion regardless of the platform.

In this chapter, we describe the existing concepts and new features of Ant in Rational Application Developer. We demonstrate how to use the Ant tooling to build applications.

The chapter is organized into the following sections:

- ▶ Introduction to Ant
- ▶ Ant features in Rational Application Developer
- ▶ New Ant features in Rational Application Developer
- ▶ Building a Java EE application
- ▶ Running Ant outside of Rational Application Developer
- ▶ Using the Rational Application Developer Build Utility

The sample code for this chapter is in the 7835code\ant folder.

## 24.1 Introduction to Ant

*Ant* is a Java-based, platform-independent, open source build tool. It was formerly a sub-project in the Apache Jakarta project, but in November 2002, it was migrated to an Apache top-level project. Ant's function is similar to the *make* tool. Because it is Java-based and does not use any operating system-specific functions, it is platform independent, so that you can build your projects by using the same build script on any Java-enabled platform.

The Ant build operations are controlled by the contents of the XML-based script file. This file defines the operations, the order in which to run them, and the dependencies among them.

Ant comes with several built-in tasks that are sufficient to perform many common build operations. However, if the tasks that are included are insufficient, you can extend Ant's functionality by using Java to develop your own specialized tasks. These tasks can then be plugged into Ant.

Not only can you use Ant to *build* your applications, you also can use Ant for many other operations, such as retrieving source files from a version control system, storing the result back in the version control system, transferring the build output to other machines, deploying the applications, generating *Javadoc*, and sending messages when a build is finished.

### 24.1.1 Ant build files

Ant uses XML *build files* to define the operations that must be performed to build a project. A build file has the following major components:

<b>Project</b>	Task that defines the project name and the default target. It is an arbitrary name.
<b>Target</b>	<p>The <i>tasks</i> that must be performed to satisfy a goal. For example, compiling source code into class files might be one target, and packaging the class files into a JAR file might be another target.</p> <p>Targets can depend on other targets. For example, the class files must be up-to-date before you can create the JAR file. Ant can resolve these dependencies.</p>
<b>Task</b>	A single step that must be performed to satisfy a target. Tasks are implemented as Java classes that are invoked by Ant, passing parameters defined as attributes in the XML. Ant provides a set of standard tasks (core tasks), a

set of optional tasks, and an API, which allows you to write your own tasks.

<b>Property</b>	Variables that can be passed to tasks through task attributes. A <i>property</i> has a name and a value pair. Property values can be set inside a build file, or obtained externally from a properties file or from the command line. A property is referenced by enclosing the property name inside <code>\${}</code> , for example <code>\${basedir}</code> .
<b>Path</b>	A set of directories or files. Paths can be defined once and referred to multiple times, easing the development and maintenance of build files. For example, a Java compilation task can use a path reference to determine the class path to use.

## 24.1.2 Ant tasks

A comprehensive set of built-in tasks is supplied with the Ant distribution. We use the following tasks in our example:

<b>delete</b>	Deletes files and directories
<b>echo</b>	Outputs messages
<b>jar</b>	Creates Java archive files
<b>javac</b>	Compiles Java source
<b>mkdir</b>	Creates directories
<b>tstamp</b>	Sets properties containing date and time information

To learn more about Ant, visit the Ant website at the following address:

<http://ant.apache.org/>

In this chapter, we provide an outline of the features and capabilities of Ant. For complete information, consult the Ant documentation that is included in the Ant distribution at the following address:

<http://ant.apache.org/manual/index.html>

**Important:** Rational Application Developer ships with Apache Ant V1.7.1. However at the time of writing this book, but due to the following Eclipse defect, it uses Apache Ant 1.6.5. You can confirm the version by executing the `runAnt.bat -version` command: [https://bugs.eclipse.org/bugs/show\\_bug.cgi?id=325125](https://bugs.eclipse.org/bugs/show_bug.cgi?id=325125). This defect has been reported via authorized program analysis report (APAR) PM24303.

## 24.2 Ant features in Rational Application Developer

Rational Application Developer includes the following features to aid in the development and use of Ant scripts:

- ▶ Rational Application Developer provides the ability to create and run Ant build files in the workbench and run the build process in the background, similar to other tasks.
- ▶ The Ant editor offers content assist (including Ant-specific templates) with the ability to insert snippets and syntax highlighting.
- ▶ The Ant editor has a format function with which you can format Ant files based on your preferences.
- ▶ The Ant editor offers annotation support.
- ▶ Rational Application Developer provides the ability to add new Ant tasks and types that will be available for build files.
- ▶ The Ant editor offers a Problems view to highlight syntax errors in the Ant files.

In this section, we highlight the following Ant-related features in Rational Application Developer:

- ▶ Content assist
- ▶ Code snippets
- ▶ Formatting an Ant script
- ▶ Defining the format of an Ant script
- ▶ Problems view

### 24.2.1 Preparing for the sample

To demonstrate the concepts of Ant, we provide a simple Java application named `HelloAnt`, which prints a message to the console. We use a simple Java project (`RAD8Ant`) and class (`HelloAnt`) for this example.

To create a new Java project, follow these steps:

1. In the workbench, select **File** → **New** → **Project**.
2. In the New Project window, select **Java** → **Java Project** and click **Next**.
3. When prompted, in the Project name field, enter `RAD8Ant` and click **Finish**.
4. If the current perspective is not the Java perspective when you create the project, click **Yes** when Rational Application Developer prompts you to switch to the Java perspective.

To import the HelloAnt class into the RAD8Ant Java project, follow these steps:

1. Right-click the **RAD8Ant** project and select **New** → **Package**.
2. In the New Package window, for Name, type `itso.rad8.ant.hello` and click **Finish**.
3. Right-click **itso.rad8.ant.hello** and select **Import**.
4. In the Import window, select **General** → **File System** and click **Next**.
5. In the Import: File system window (Figure 24-1), for the From directory, click **Browse** and select `c:\7835code\ant\`. Select **HelloAnt.java** and click **Finish**.

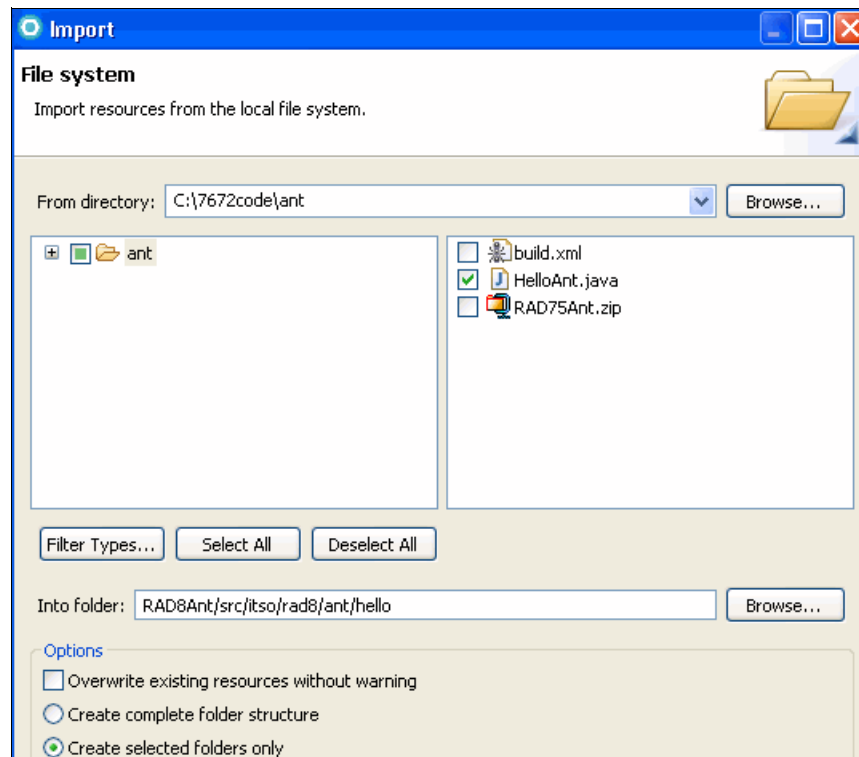


Figure 24-1 Importing a Java class

## 24.2.2 Creating a build file

To create the simple build file, follow these steps:

1. Right-click the **RAD8Ant** project and select **New File**.
2. In the New File window (Figure 24-2 on page 1284), for the File name, type `build.xml` and click **Finish**.

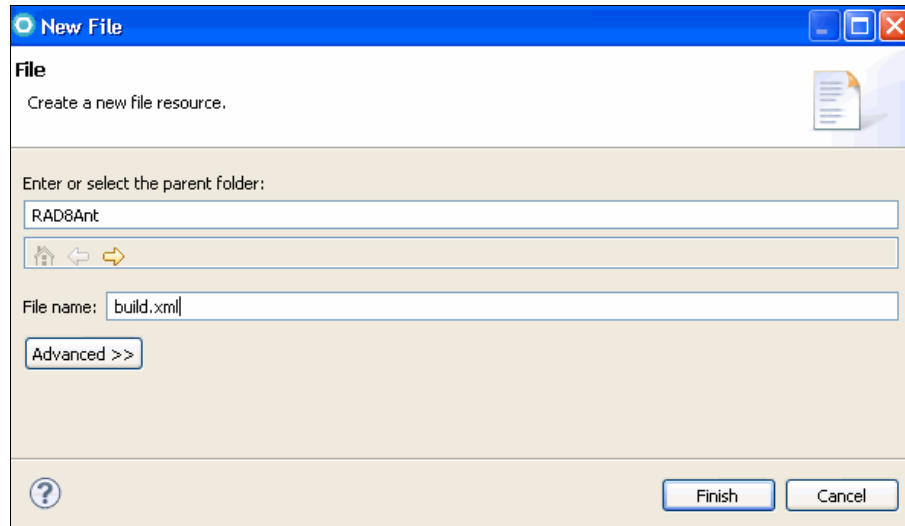


Figure 24-2 Creating the build.xml file

**Linking to external files:** Rational Application Developer can link to external files on the file system. When you click **Advanced** in the New File window, you can specify the location on the file system to which the new file is linked.

3. Double-click the **build.xml** file to open it in the Ant editor. Copy and paste the text from `c:\7835code\ant\build.txt` to the `build.xml` file.

We now take you through the various sections of this file and explain each section.

### 24.2.3 Project definition

The `<project>` tag in the `build.xml` file defines the project name and the default target. The project name is an arbitrary name; it is not related to any project name in your Rational Application Developer workspace.

```
<project name="HelloAnt" default="dist" basedir=".">
```

The project tag also sets the working directory for the Ant script. All references to directories throughout the script file are based on this directory. A dot (.) means to use the current directory, which, in Rational Application Developer, is the directory where the `build.xml` file resides.



## 24.2.4 Global properties

Properties that will be referenced throughout the whole script file can be placed at the beginning of the Ant script. Here, we define the `build.compiler` property that specifies the compiler for the `javac` command to use. We define it to use the Eclipse compiler.

We also define the names for the source directory, the build directory, and the distribute directory. The *source directory* is where the Java source files reside. The *build directory* is where the class files end up, and the *distribute directory* is where the resulting JAR file is placed:

- ▶ We define the source property as ".", which means that it is the same directory as the base directory that is specified in the project definition.
- ▶ The build and distribute directories will be created as the `c:\temp\build` and `c:\temp\RAD8Ant` directories.

Properties can be set as shown in the following example, but Ant can also read properties from standard Java properties files or use parameters that are passed as arguments on the command line:

```
<!-- set global properties for this build -->
<property name="build.compiler"
 value="org.eclipse.jdt.core.JDTCompilerAdapter"/>
<property name="source" value="."/>
<property name="build" value="c:\temp\build"/>
<property name="distribute" value="c:\temp\RAD8Ant"/>
<property name="outFile" value="helloant"/>
```

## 24.2.5 Building targets

The build file contains four build targets:

- ▶ `init`
- ▶ `compile`
- ▶ `dist`
- ▶ `clean`

### Initialization target (init)

As shown in the following example, the first target that we describe is the `init` target. All other targets (except `clean`) in the build file depend on this target. In the `init` target, we execute the `tstamp` task to set up properties that include time stamp information. These properties are then available throughout the whole build. We also create a build directory that is defined by the `build` property.

```

<target name="init">
 <!-- Create the time stamp -->
 <tstamp/>
 <!-- Create the build directory structure used by compile -->
 <mkdir dir="${build}"/>
</target>

```

### Compilation target (compile)

The compile target compiles the Java source files in the source directory and places the resulting class files in the build directory:

```

<target name="compile" depends="init">
 <!-- Compile the java code from ${source} into ${build} -->
 <javac srcdir="${source}" destdir="${build}"/>
</target>

```

With this definition, if the compiled code in the build directory is up-to-date (each class file has a time stamp that is later than the corresponding Java file in the source directory), the source is not recompiled.

### Distribution target (dist)

The dist target creates a JAR file that contains the compiled class files from the build directory and places it in the lib directory under the dist directory. Because the distribution target depends on the compile target, the compile target must have executed successfully before the distribution target is run.

```

<target name="dist" depends="compile">
 <!-- Create the distribution directory -->
 <mkdir dir="${distribute}/lib"/>

 <!-- Put everything in ${build} into the output JAR file -->
 <!-- We add a time stamp to the filename as well -->
 <jar jarfile="${distribute}/lib/${outFile}-${DSTAMP}.jar"
 basedir="${build}">
 <manifest>
 <attribute name="Main-Class"
 value="itso.rad8.ant.hello.HelloAnt"/>
 </manifest>
 </jar>
</target>

```

### Cleanup target (clean)

The last of our standard targets is the clean target, which is shown in the following example. This target removes the build and distribute directories, which means that a full recompile is always performed if this target has been executed.

```

<target name="clean">
 <!-- Delete the ${build} and ${distribute} directory trees -->
 <delete dir="${build}"/>
 <delete dir="${distribute}"/>
</target>

```

The build.xml file does not call for this target to be executed. It has to be explicitly specified when running Ant.

## 24.2.6 Content assist

To access the content assist feature in the Ant editor, follow these steps:

1. Open the **build.xml** file in an editor (if it is not already open).
2. Place the cursor in the file, enter `<pro` and then press `Ctrl+Spacebar`.

The content assist window (Figure 24-3) opens, in which you can use the up and down arrow keys to select the tag that you want.

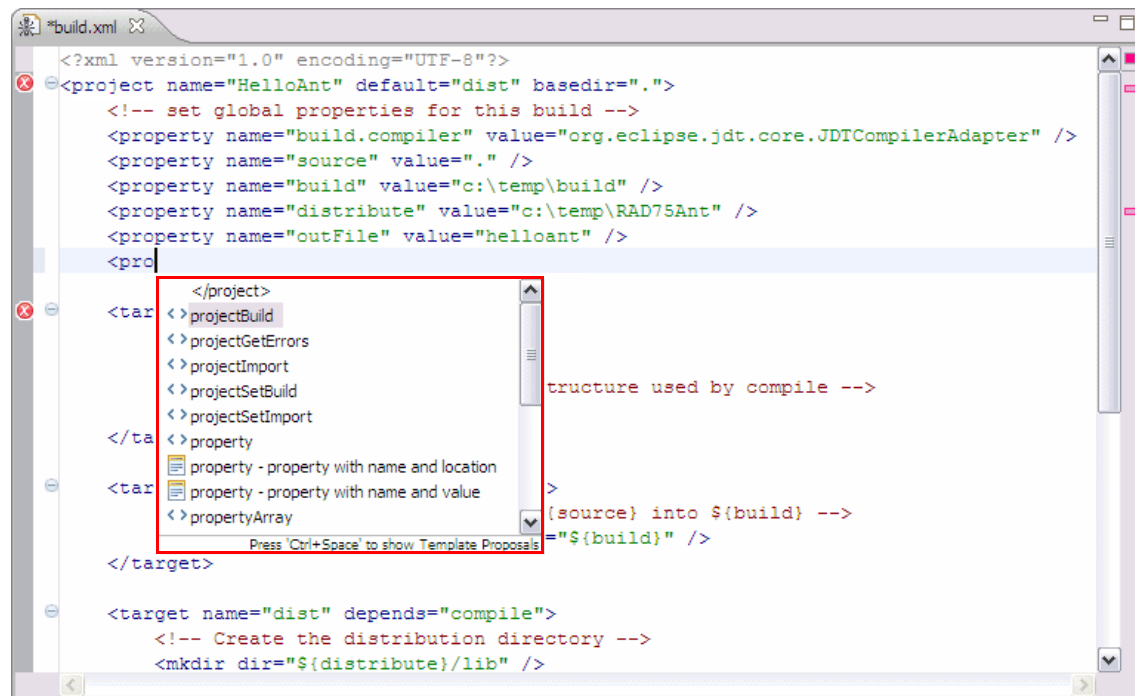


Figure 24-3 Content assist in the Ant editor

## 24.2.7 Code snippets

Rational Application Developer provides the ability to create code snippets that contain commonly used code to be inserted into files so that you do not have to type the code each time.

### Creating code snippets

To create code snippets, follow these steps:

1. Open the Snippets view:
  - a. Select **Window** → **Show View** → **Other**.
  - b. In the Show View window, expand the **General** folder, select the **Snippets** view and click **OK**.
2. Right-click the **Snippets** view and select **Customize** (Figure 24-4).

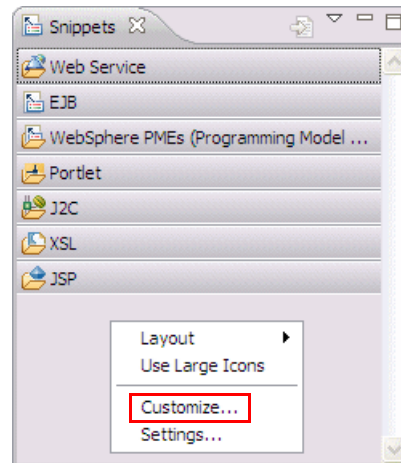


Figure 24-4 Customizing snippets

3. In the Customize Palette window, complete these steps:
  - a. Select **New** → **New Category**.
  - b. In the New Customize Palette window (Figure 24-5 on page 1289), follow these steps:
    - i. For the Name, enter **Ant**.
    - ii. For the Description, enter **Ant Snippets**.
    - iii. For the Show/Hide Drawer, select **Custom**.
    - iv. Under Custom, click **Browse**. For Content Type Selection, select **Ant Buildfile** and click **OK**. You return to the Customize Palette window.

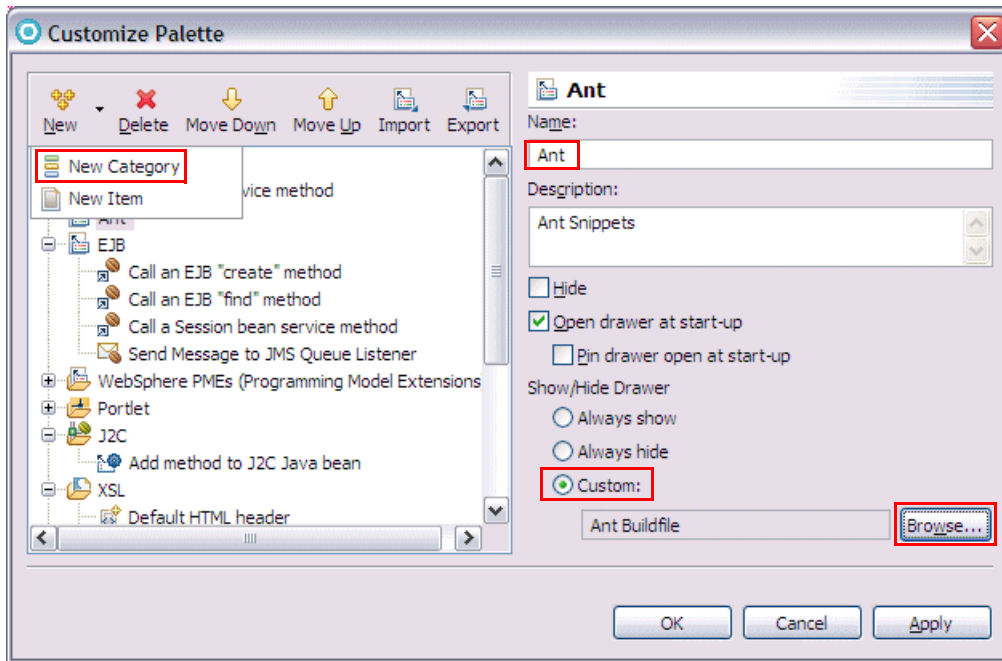


Figure 24-5 New Customize Palette window

- c. Select **New** → **New Item**.
- d. In the Unnamed Template window, enter the following items:
  - i. For the Name, enter Comment Tag.
  - ii. In the variables section, click **New**.
  - iii. For the Variable Name, enter comment.
  - iv. For the Template Pattern, enter `<!-- ${comment} -->`.
- e. In the Customize Palette window, click **OK**.

The Ant category with the Comment Tag entry is added to the Snippets view.

### Using code snippets

After you create a code snippet, you can use it in any Ant build file. To use a code snippet, follow these steps:

1. Open the **build.xml** file.
2. Add an empty line under the `<project>` tag, place the cursor there, and double-click the **Comment Tag** in the Snippets view.

3. In the Insert Template: Comment Tag window (Figure 24-6), in the Variables table, for the value of the comment variable, type `This is a comment`. Click **Insert**.



Figure 24-6 Insert Template: Comment Tag window

The comment line is inserted:

```
<project name="HelloAnt" default="dist" basedir=".">
 <!-- This is a comment -->
 <!-- set global properties for this build -->
```

4. Save the file.

## 24.2.8 Formatting an Ant script

Rational Application Developer offers the ability to format Ant scripts in the Ant editor. To format the Ant script, follow these steps:

1. Open the **build.xml** file.
2. Right-click the editor and select **Format**, or press **Ctrl+Shift+F**.

## 24.2.9 Defining the format of an Ant script

To define the format of an Ant script, follow these steps:

1. Select **Window** → **Preferences**.
2. In the Preferences window, select and expand **Ant**.

In the Ant Preferences window (Figure 24-7), you can specify the console colors.

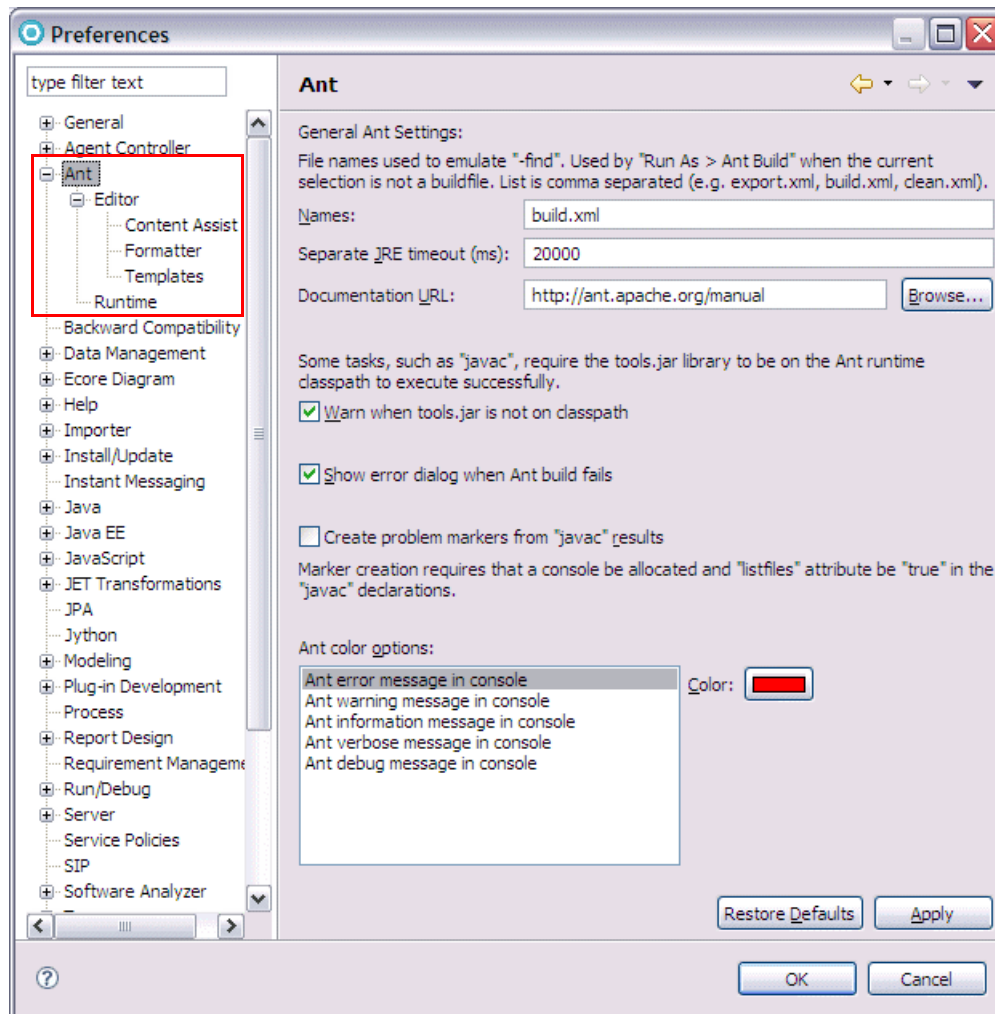


Figure 24-7 Ant preferences

3. Expand the **Ant** folder and select **Editor**.
  - Select the **Appearance** tab to change the layout preferences of your Ant file.
  - Select the **Syntax** tab to change the syntax highlighting preferences with a preview of the results. Then on the Problems tab, you can define how to handle certain problems.

- Select the **Problems** tab to change the severity levels for the build file problems.
  - Select the **Folding** tab to enable folding when opening a new editor, and specify which region types to fold.
4. Expand **Editor**.
- In the Content Assist window, you can define the content assist preferences.
  - In the Formatter window, you can define the preferences for the formatting tool for the Ant files.
  - In the Templates window, you can create, edit, delete, import, and export templates for Ant files.
5. Select **Runtime**.
- In this window, you can define your preferences, such as class path, tasks, types, and properties.

## 24.2.10 Problems view

Rational Application Developer shows the problems of the build file in the Problems view (Figure 24-8).

The screenshot shows the Problems view in Rational Application Developer. The view title is "Problems" and it contains 2 errors, 0 warnings, and 0 others. The table below lists the errors:

Description	Resource	Path	Locat...	Type
Errors (2 items)				
Default target Total does not exist in th	build.xml	/RAD8Ant	line 2	Ant Buildfile ...
The element type "target" must be tern	build.xml	/RAD8Ant	line 97	Ant Buildfile ...

Figure 24-8 Problems view displaying Ant errors



The editor marks an error by placing a *red X* to the left of the line with the problem and a line marker in the file to the right of the window (Figure 24-9).



Figure 24-9 Problems marked in the Ant editor

## 24.3 New Ant features in Rational Application Developer

In this section, we describe the new features that have been added to Rational Application Developer and that are supported by WebSphere Application Server v8.0 Beta.

### 24.3.1 SCA Ant task

The `scaArchiveExport` task exports the Service Component Architecture (SCA) archive file from SCA projects (Table 24-1 on page 1294). This task is similar to accessing the option **Export** → **Service Component Architecture** → **SCA Archive File** for an SCA project. This task was first introduced in Rational Application Developer V7.5.5.

Table 24-1 SCA task parameters

Attribute	Description	Required
archiveLocation	This attribute shows the absolute path of the SCA Archive file to export.	Yes
overwrite	Check to overwrite if the file exists.	False (default)
compressContents	Check to compress the contents of the archive file.	False (default)
isComposites	Check if the file is a composite.	False (default)
fileset	Specify the composite files to be included in the SCA archive.	False (default)

Example 24-1 shows a small sample showing the export of an SCA archive JAR file.

Example 24-1 Exporting an SCA archive JAR

---

```

<scaArchiveExport overwrite="true"
archiveLocation="${basedir}/exportedSCAArchive.jar"
compressContents="true"
isComposites="true">
<fileset dir="${importLocation}" casesensitive="yes">
 <filename name="Hello_World/*.composite"/>
</fileset>
</scaArchiveExport>

```

---

## 24.3.2 OSGi Ant tasks

You can use the following Ant tasks with OSGi application development tools:

- ▶ **osgiApplicationExport**  
Use this task to export an OSGi application.
- ▶ **osgiApplicationImport**  
Use this task to import an OSGi application.
- ▶ **osgiBundleExport**  
Use this task to export an OSGi Bundle.
- ▶ **osgiBundleImport**  
Use this task to import an OSGi Bundle.
- ▶ **osgiCompositeBundleExport**  
Use this task to export an OSGi CompositeBundle.

▶ **osgiCompositeBundleImport**

Use this task to import an OSGi CompositeBundle.

▶ **osgiConvertProject**

This task converts an existing project into an OSGi Bundle project.

Consult the following resources for further information about OSGi Ant tasks:

- ▶ To learn more about OSGi, see Chapter 15, “Developing Open Services Gateway initiative (OSGi) applications” on page 837.
- ▶ For more information about the OSGi Ant task attributes, see this website:  
<http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/com.ibm.ant.tasks.doc/topics/ph-antsca.html>
- ▶ For information about developing OSGi applications, see *Getting Started with the Feature Pack for OSGi Applications and JPA 2.0*, SG24-7911.

### 24.3.3 Other new Ant tasks

The following Ant tasks were first introduced in Rational Application Developer V7.5.5.1.

#### **prepForDeploy task**

This task is equivalent in functionality to the *Prepare for Deployment* currently provided in the product user interface. Table 24-2 shows the attributes for the prepForDeploy task.

Table 24-2 *prepForDeploy* task parameters

Attributes	Description	Required
projectName	The name of the project for which deployment code needs to be generated	Yes
failOnError	Fails on error when set to true	No

Example 24-2 shows an example of using this task to generate deployment code for the MyWebProject project.

Example 24-2 *Example showing the Ant task*

```
<prepForDeploy projectName="MyWebProject" failOnError="false"/>
```

## XML Catalog task

The `xmlCatalog` task provides equivalent functionality to selecting **Window** → **Preferences** → **XML** → **XML Catalog**. For information about the XML Catalog functionality in the product, see this website:

<http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/org.eclipse.wst.xmleditor.doc.user/topics/txmlcat.html>

Example 24-3 creates a local reference to the URI `http://test.sample.org/dtds/my.dtd`. The possible values for the `keyType` attribute are 'public', 'system', and 'uri'. If desired, you can also specify an alternative web address via the `webURL` attribute.

*Example 24-3 XMLcatalog task1*

---

```
<xmlCatalog keyType="public" uri="/path/to/my.dtd"
key="http://test.sample.org/dtds/my.dtd" />
```

---

Example 24-4 appends an existing XML catalog to the XML catalog.

*Example 24-4 xmlcatalog task2*

---

```
<xmlCatalog catalogLocation="/path/to/next_catalog.xml" />
```

---

## 24.4 Building a Java EE application

In this section, we demonstrate how to build a Java Platform, Enterprise Edition (Java EE) application from existing Java EE-related projects.

This section is organized in the following manner:

- ▶ Java EE application deployment packaging
- ▶ Preparing for the sample
- ▶ Creating the build script
- ▶ Running the Ant Java EE application build

## 24.4.1 Java EE application deployment packaging

EAR, WAR, and EJB JAR files contain several deployment descriptors that control how to deploy the artifacts of the application onto an application server. These deployment descriptors are mostly XML files and are standardized within the Java EE specification.

While working in Rational Application Developer, part of the information in the deployment descriptor is stored in XML files. The deployment descriptor files also contain information in a format that is convenient for interactive testing and debugging. Therefore, it is quick and easy to test Java EE applications in the integrated WebSphere Application Server with Rational Application Developer.

The actual EAR that is being tested, and its supporting WAR, EJB, and client application JARs, is not created as a stand-alone file. Instead, a special EAR file is used that points to the build contents of the various Java EE projects. Because these individual projects can be anywhere on the development machine, absolute path references are used.

When an enterprise application project is exported, a true stand-alone EAR file is created, including all the module WAR, EJB JAR, Java Persistence API (JPA), and Java utility JAR files that it contains. Therefore, during the export operation, all absolute paths are changed into self-contained relative references within that EAR, and the internally optimized deployment descriptor information is merged and changed into a standard format. To create a Java EE-compliant WAR or EAR, we must use the export function in Rational Application Developer.

## 24.4.2 Preparing for the sample

To demonstrate how to build a Java 2 Platform Enterprise Edition (J2EE) application using Ant, we use the Java EE applications that were developed in Chapter 12, “Developing Enterprise JavaBeans (EJB) applications” on page 577.

To import the RAD8EJB.zip file that contains the sample code into Rational Application Developer, follow these steps:

1. Open the Java EE perspective.
2. Select **File** → **Import**.
3. Expand the **General** folder, select **Existing Projects into Workspace** and click **Next**.
4. In the Import Projects window, click **Browse** next to the compressed file, navigate to the **c:\7835code\ejb**, and select the **RAD8EJB.zip** file. Click **Open**.

5. Click **Select All** to select all of the projects and then click **Finish**.

After importing, you have the following projects in your workspace:

- ▶ RAD8EJB
- ▶ RAD8EJBEAR
- ▶ RAD8EJBTestWeb
- ▶ RAD8JPA

### 24.4.3 Creating the build script

To build the RAD8EJBEAR enterprise application, we created an Ant build script (`build.xml`) that uses the Java EE Ant tasks that are provided by Rational Application Developer.

To add the Ant build script to the project, follow these steps:

1. In the Enterprise Explorer view, expand **RAD8EJBEAR** and select **META-INF**.
2. Select **File** → **New** → **Other**.
3. In the New File window, select **General** → **File** and click **Next**.
4. In the File name field, type `build.xml` and click **Finish**.
5. Double-click the **build.xml** file to open it in the editor. Copy and paste the text from the `c:\7835code\ant\j2ee\build.txt` file to the `build.xml` file.
6. Modify the value for the `work.dir` property to match your desired working directory (for example, `c:/temp/RAD8AntEE`), as highlighted in Example 24-5.

*Example 24-5 Java EE Ant build.xml script*

---

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="ITSO RAD Pro Guide Ant" default="Total" basedir=".">

 <!-- Set global properties -->
 <property name="work.dir" value="c:/temp/RAD8AntEE" />
 <property name="dist" value="${work.dir}/dist" />
 <property name="project.ear" value="RAD8EJBEAR" />
 <property name="project.ejb" value="RAD8EJB" />
 <property name="project.war" value="RAD8EJBTestWeb" />
 <property name="type" value="incremental" />
 <property name="debug" value="true" />
 <property name="source" value="true" />
 <property name="meta" value="false" />
 <property name="noValidate" value="false" />
```

---

The `build.xml` script includes the following Ant targets, which correspond to common Java EE application builds:

<b>deployEjb</b>	Generates the deploy code for all EJB in the project
<b>buildEjb</b>	Builds the EJB project (compiles resources within the project)
<b>buildWar</b>	Builds the web project (compiles resources within the project)
<b>buildEar</b>	Builds the Enterprise Application project (compiles resources within the project)
<b>exportEjb</b>	Exports the EJB project to a JAR file
<b>exportWar</b>	Exports the web project to a WAR file
<b>exportEar</b>	Exports the Enterprise Application project to an EAR file
<b>buildAll</b>	Invokes the <code>buildEjb</code> , <code>buildWar</code> , and <code>buildEar</code> targets
<b>exportAll</b>	Invokes the <code>exportEjb</code> , <code>exportWar</code> , and <code>exportEar</code> targets to create the <code>RAD8EJBEAR.ear</code> file that is used for deployment

The following targets are additional:

<b>init</b>	Initializes and creates a directory
<b>info</b>	Prints the properties
<b>Total</b>	Invokes <code>buildAll</code> and <code>exportAll</code>
<b>clean</b>	Deletes the output files

## EJB specification level

If you have enterprise beans at the 1.1, 2.0, or 2.1 specification-level, you have to generate deployment code for the enterprise beans. The **`ejbDeploy`** command that is run under the `deployEjb` target generates deployment code for these artifacts.

When you install WebSphere Application Server V6.1 with Feature Pack for EJB 3.0 or WebSphere Application Server V7.0, you can use the EJB 3.0 specification at run time. For the EJB 3.0 specification level, you no longer have to generate the EJB deployment code. The **`ejbdeploy`** command does not generate deployment code for artifacts at the Java EE 5 specification level.

The following list describes the general behavior of the **`ejbdeploy`** command when issued with the presence of Java EE 5 artifacts:

- ▶ It tolerates EAR 5.0 files and EJB 3.0 JAR files.
- ▶ It tolerates EAR files with J2EE 1.4 deployment descriptors that contain EJB 3.0 JAR files. Deployment code is generated only for EJB at the 1.1, 2.0, or

2.1 specification level. However, deployment code is not generated for EJB beans at the 3.0 specification level.

- ▶ If the `-complianceLevel` option for the **ejbdeploy** command is not specified, the default `-complianceLevel "5.0"` setting is for the Java Developer Kit (JDK) 5.0 in these cases:
  - An EAR or JAR file that contains Java EE 5 or EJB 3.0 deployment descriptor files
  - An EAR file without any deployment descriptor files
- ▶ For all other cases, the `-complianceLevel "1.4"` setting defaults to Java Developer Kit 1.4.
- ▶ For JDK 1.6, the `-complianceLevel "1.6"` settings are applicable.

If you are generating deployment code for J2EE 1.4 EAR or JAR files that contain source code files that use the new language features in Java Developer Kit 5.0, you must specify the `-complianceLevel "5.0"` parameter when running the **ejbdeploy** command.

In the global properties for this build script, we define several useful variables, such as the project names and the target directory. We also define many properties that we pass on to the Rational Application Developer Ant tasks. With these properties, we can control whether we want the build process to perform a full or incremental build, whether to include debug statements in the generated class files, and whether to include the metadata information for Rational Application Developer when exporting the project.

When starting this Ant script, we can also override these properties by specifying other values in the arguments field, so that we can perform separate builds with the same script.

#### 24.4.4 Running the Ant Java EE application build

When starting the `build.xml` script, you can select which targets to run and the execution order.

To run the Ant `build.xml` to build the Java EE application, follow these steps:

1. Right-click **build.xml** (in RAD8EJBEAR/META-INF) and select **Run As** → **Ant Build**.



2. In the Edit configuration and launch window, complete these steps:
  - a. Select the **Main** tab:
    - To build the Java EE EAR file with debug, source files, and metadata, enter the following values in the Arguments text area:  
`-DDebug=true -Dsource=true -Dmeta=true`
    - To build the Java EE EAR for production deployment (without debug support, source code, and metadata), enter the following value in the Arguments text area:  
`-Dtype=full`
  - b. Select the **Targets** tab. Ensure that **Total** is selected (default).
  - c. Select the **JRE** tab. Select **Run in the same JRE as the workspace**.
  - d. Click **Apply** and then click **Run**.
3. Verify in the `c:\temp\RAD8AntEE\dist` output directory that the `RAD8EJBEAR.ear`, `RAD8EJB.jar`, and `RAD8EJBTestWeb.war` files were created.

The Console view shows the operations that were performed and their results.

## 24.5 Running Ant outside of Rational Application Developer

To automate the build process even further, you might want to run Ant outside of Rational Application Developer by running Ant in *headless mode*.

### 24.5.1 Preparing for the headless build

Rational Application Developer includes a `runAnt.bat` file, which you can use to invoke Ant in headless mode, and passes the parameters that you specify. You must customize this option for your environment.

The `runAnt.bat` file that is included with Rational Application Developer is in the `<rad_home>\bin` directory.

To create a headless Ant build script for a Java EE project, follow these steps:

1. Copy the `runAnt.bat` file to a new file called `itsRunAnt.bat`.
2. Modify the `WORKSPACE` value in the `itsRunAnt.bat` so that it points to your current workspace (Example 24-6 on page 1302):

```
set WORKSPACE=C:\workspaces\Test\ANT
```

*Example 24-6 Snippet of the itsoRunAnt.bat (modified runAnt.bat)*

---

```
.....
set JAVAEXE="E:\Program Files\IBM\SDP\jdk\jre\bin\java.exe"
.....
set INSTALL_DIRECTORY="E:\Program Files\IBM\SDP"
.....
set LAUNCHER_JAR="E:\Program
Files\IBM\SDPShared\plugins\org.eclipse.equinox.launcher_1.1.0.v2010
0507.jar"
REM #####
REM ##### you must edit the "WORKSPACE" setting below #####
REM #####
REM ***** The location of your workspace *****
set WORKSPACE=C:\workspaces\Test\ANT

:workspace
if not $%WORKSPACE%==$ goto check
.....
```

---

## 24.5.2 Running the headless Ant build script

**Important:** Prior to running Ant in headless mode, you must close Rational Application Developer. If you do not close Rational Application Developer, you receive build errors when you attempt to run an Ant build in headless mode.

To run the `itsoRunAnt.bat` command file, follow these steps:

1. Ensure that you have closed Rational Application Developer.
2. Open a Microsoft Windows command prompt.
3. Navigate to the location of the `itsoRunAnt.bat` file.
4. Run the command file by entering the following command:

```
itsoRunAnt -buildfile
c:\workspaces\Test\ANT\EJBEBEAR\META-INF\build.xml clean Total
-DDebug=true -Dsource=true -Dmeta=true
```

The `-buildfile` parameter specifies the fully qualified path of the `build.xml` script file. We can pass the targets to run as parameters to `itsoRunAnt`, and we can pass Java environment variables by using the `-D` switch.

In this example, we run the `clean` and `Total` targets. We include the debug, Java source, and metadata files in the resulting EAR file.

The following build output files are produced and are in the `c:\temp\RAD8AntEE\dist` directory:

- ▶ `RAD8EJB.jar`
- ▶ `RAD8EJBEAR.ear`
- ▶ `RAD8EJBTestWeb.war`

**Notes:** We include `itsoRunAnt.bat` and `output.txt` files in the `c:\7835code\ant\j2ee` directory. The `output.txt` file contains the output from the headless Ant script for review purposes.

Verify that the installation directory (see Example 24-6 on page 1302, the installation directory is bold) points to the correct directory on your system.

## 24.6 Using the Rational Application Developer Build Utility

Rational Application Developer V7.5 introduced a new feature, called *build utility*, that can be installed stand-alone on a build server running on Windows, Linux, or z/OS. The build utility has a smaller footprint than Rational Application Developer, because it does not contain any user interface code. The inputs to the build utility are the projects that are developed in Rational Application Developer, and the outputs are JAR, WAR, and EAR files.

### 24.6.1 Overview of the build utility

**More information:** For information about installing the build utility, see “Installing Rational Application Developer Build Utility” on page 1840. The Rational Application Developer Build Utility offers support for IBM Installation Manager on the Microsoft Windows and Linux platforms.

In the following example, we assume that the build utility was installed on Microsoft Windows in the `C:\IBM\BuildUtility` folder.

The build utility includes a `runAnt.bat` file that can be used to invoke Ant in headless mode in the same way that Rational Application Developer invokes Ant in headless mode. The `runAnt.bat` file is in the `C:\IBM\BuildUtility\eclipse\bin` directory.

When you invoke a build on a build server, typically you use Ant to create a workspace that contains the projects before you build them. In the following

example, we modify the Ant build script so that first it imports the projects into a new workspace and then it builds them.

## 24.6.2 Example of using the build utility

We use the RAD8EJBWebEAR enterprise application for the build utility example. This application is similar to the RAD8EJB EAR application that we used for the Ant headless build.

Extract the C:\7835codesolution\ejb\RAD8EJBWeb.zip file into a C:\sources directory on the source server.

### Creating the build file (BUbuild.xml)

To create the build file, we copy the build file from the headless Ant example and modify it for the RAD8EJBWebEAR enterprise application:

1. Copy the RAD8EJB\META-INF\build.xml file to the C:\sources\BUbuild.xml file.
2. Modify the BUbuild.xml file, as highlighted in Example 24-7. We made the following changes to the original build.xml file:
  - The project names differ.
  - The projects are imported into the workspace before the build.
  - A full build was done instead of an incremental build.

#### *Example 24-7 BUbuild.xml for the build utility*

---

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="ITSO RAD Pro Guide Ant Build Utility" default="Total"
 basedir=".">
```

```

 <!-- Set global properties -->
 <property name="work.dir" value="c:/temp/RAD8BU" />
 <property name="dist" value="${work.dir}/dist" />
 <property name="project.ear" value="RAD8EJBWebEAR" />
 <property name="project.ejb" value="RAD8EJB" />
 <property name="project.war" value="RAD8EJBWeb" />
 <property name="project.jpa" value="RAD8JPA" />
 <property name="type" value="full" />
 <property name="debug" value="true" />
 <property name="source" value="true" />
 <property name="meta" value="false" />
 <property name="noValidate" value="false" />
```

```
<target name="init">...
```

```
<target name="info">
```

```

 <!-- Displays the properties for this run -->
 <echo message="debug=${debug}" />
 <echo message="type=${type}" />
 <echo message="source=${source}" />
 <echo message="meta=${meta}" />
 <echo message="noValidate=${noValidate}" />
 <echo message="Output directory=${dist}" />
 <echo message="project.ear=${project.ear}" />
 <echo message="project.ejb=${project.ejb}" />
 <echo message="project.war=${project.war}" />
 <echo message="project.jpa=${project.jpa}" />
</target>

<target name="importJPA">
 <projectImport projectName="${project.jpa}" />
 <eclipse.refreshLocal resource="${project.jpa}" />
</target>
<target name="importEJB">
 <projectImport projectName="${project.ejb}" />
 <eclipse.refreshLocal resource="${project.ejb}" />
</target>
<target name="importWAR">
 <projectImport projectName="${project.war}" />
 <eclipse.refreshLocal resource="${project.war}" />
</target>
<target name="importEAR">
 <projectImport projectName="${project.ear}" />
 <eclipse.refreshLocal resource="${project.ear}" />
</target>
<target name="importAll"
 depends="importJPA,importEJB,importWAR,importEAR">
 <!-- Import all projects and exports all files -->
 <echo message="Import All projects" />
</target>

<target name="deployEjb">...
<target name="buildEjb" depends="deployEjb">...
<target name="buildJPA">
 <!-- Builds the JPA project -->
 <projectBuild projectName="${project.jpa}" BuildType="${type}"
 DebugCompilation="${debug}" />
</target>
<target name="buildWar">...
<target name="buildEar">...
<target name="exportEjb" depends="init">...
<target name="exportWar" depends="init">...
<target name="exportEar" depends="init">...
<target name="buildAll"
depends="buildJPA,buildEjb,buildWar,buildEar">

```

```

 <!-- Builds all projects -->
 <echo message="Built all projects" />
 </target>

 <target name="exportAll" depends="exportEjb,exportWar,exportEar">...
 <target name="Total" depends="importAll,buildAll,exportAll">...
 <target name="clean">...
</project>

```

---

The projects are imported into the Eclipse workspace from the workspace directory, using targets, such as the following example:

```

<target name="importEJB">
 <projectImport projectName="{project.ejb}"/>
 <eclipse.refreshLocal resource="{project.ejb}" />
</target>

```

Because we do not specify the `projectLocation` attribute for the `projectImport` task, it is assumed that the projects to import are in the workspace directory. The `projectImport` task does not make a physical copy of the projects. It only imports a reference to the projects.

## Creating the command file for execution

To run the build utility, create a batch command file:

1. Create a command file as `C:\sources\itsoBUBuild.bat` with the content that is shown in Example 24-8.

*Example 24-8 Contents of the ITSOBUBuild.bat file*

---

```

@echo on
setlocal
set WORKSPACE=C:\sources

C:\IBM\BuildUtility\eclipse\bin\runant.bat
 -buildfile C:\sources\BUbuild.xml clean Total

```

---

2. Execute the build using the following commands:

```

cd C:\sources
itsoBUBuild.bat >BUoutput.txt

```

3. Review the `BUoutput.txt` file. You see the following lines toward the end:

```

[ejbExport] EJBExport completed to c:/temp/RAD8BU/dist/RAD8EJB.jar
[warExport] WARExport completed to
c:/temp/RAD8BU/dist/RAD8EJBWeb.war

```

```
[earExport] EARExport completed to
c:/temp/RAD8BU/dist/RAD8EJBWebEAR.ear
```

4. Verify the files that are created in the c:/temp/RAD8BU/dist folder.

**Files:** We included the BUbuild.xml, itsBUBuild.bat, and BUoutput.txt files in the C:\7835code\ant\buildutility folder.

## 24.7 More information about Ant

For more information about Ant, see the following resources:

- ▶ Apache Ant home page  
<http://ant.apache.org/>
- ▶ *Automatically generate project builds using Ant* white paper  
<http://www.ibm.com/developerworks/library/ar-auototask/>







## Deploying enterprise applications

The meaning of the term *deployment* differs depending on the context. In this chapter, we define the concepts of application deployment. Then we provide a working example for packaging and deploying the ITSO Bank enterprise application to a stand-alone IBM WebSphere Application Server v8.0 Beta.

The application deployment concepts and procedures that we describe apply to WebSphere Application Server V8.0 Base, Express, and Network Deployment editions. In IBM Rational Application Developer, the configuration of the integrated WebSphere Application Server v8.0 Beta for deployment, testing, and administration is the same as for a separately installed WebSphere Application Server v8.0 Beta. The base server is the same across all the WebSphere Application Server Base, Express, and Network Deployment editions.

The chapter is organized into the following sections:

- ▶ Introduction to application deployment
- ▶ Preparing for the EJB application deployment
- ▶ Packaging the application for deployment
- ▶ Manual deployment of enterprise applications
- ▶ Automated deployment using Jython-based wsadmin scripting
- ▶ More information

The sample code for this chapter is in the 7835code\jython folder.

## 25.1 Introduction to application deployment

Deployment is a critical part of the Java Platform, Enterprise Edition (Java EE) application development cycle. Having a solid understanding of the deployment components, architecture, and process is essential for the successful deployment of the application.

In this section, we review the following concepts of the Java EE and WebSphere deployment architecture:

- ▶ Common deployment considerations
- ▶ Java EE application components and deployment modules
- ▶ Preparing for the EJB application deployment
- ▶ WebSphere deployment architecture
- ▶ Java and WebSphere class loader

**More information:** You can find further information about the IBM WebSphere Application Server deployment in the following sources:

- ▶ *WebSphere Application Server V6.1: Planning and Design*, SG24-7305
- ▶ *WebSphere Application Server V6.1: Systems Management and Configuration*, SG24-7304
- ▶ *WebSphere Application Server V6: Scalability and Performance*, SG24-6392
- ▶ IBM WebSphere Application Server V8 Beta Information Center:  
<http://publib.boulder.ibm.com/infocenter/wasinfo/beta/index.jsp>

### 25.1.1 Common deployment considerations

The following factors most often affect the deployment of a Java EE application:

- ▶ **Deployment architecture:** How can you create, assemble, and deploy an application properly if you do not understand the deployment architecture?
- ▶ **Infrastructure:** What are the hardware and software constraints for the application?
- ▶ **Security:** What security will be imposed on the application and what is the current security architecture?
- ▶ **Application requirements:** Do the application requirements imply a distributed architecture?
- ▶ **Performance:** How many users are using the system (frequency, duration, and concurrency)?

## 25.1.2 Java EE application components and deployment modules

Within the Java EE application development life cycle, the application components are created, assembled, and deployed. In this section, we explore the application component types, deployment modules, and packaging formats to gain a better understanding of what is being packaged (assembled) for deployment.

### Application component types

In Java EE 5 or later, the following application component types are supported by the runtime environment:

- ▶ Application clients: Run in the Application client container
- ▶ Applets: Run in a browser (or a stand-alone applet container)
- ▶ Web applications (servlets, JavaServer Pages (JSP), and HTML pages): Run in the Web container
- ▶ EJB: Run in the EJB container

### Deployment modules

The Java EE deployment components are packaged for deployment as modules:

- ▶ Web module
- ▶ EJB module
- ▶ Resource adapter module
- ▶ Application client module

### Packaging formats

Each module is packaged on a specific JAR file format:

- ▶ Web modules in web archive (WAR) files
- ▶ EJB and application client modules in JAR files
- ▶ Resource adapter modules in resource adapter archive (RAR) files

You can use enterprise archive (EAR) files to package EJB modules, resource adapter modules, application client modules, and web modules.

## 25.1.3 Deployment descriptors

In Java 2 Platform, Enterprise Edition (J2EE) 1.4 and earlier, information describing a J2EE application and how to deploy it into a J2EE container was stored in XML files called *deployment descriptors*. An EAR file normally contained multiple deployment descriptors, depending on the modules it

contains. Figure 25-1 shows a schematic overview of a J2EE EAR file. The various deployment descriptors are designated with DD after their name.

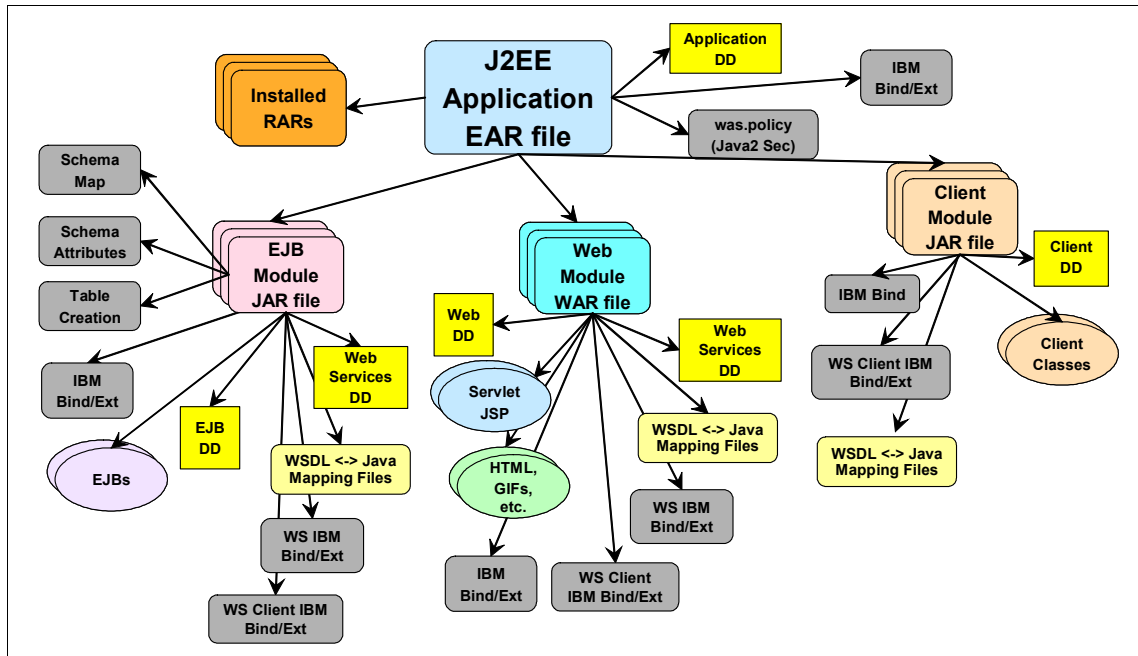


Figure 25-1 J2EE EAR file structure

The deployment descriptor of the EAR file is stored in the META-INF directory in the root of the enterprise application and is called `application.xml`. This file contains information about the modules that make up the application.

The following deployment descriptors describe the modules as indicated and are in the folders that are listed:

- ▶ The `web.xml` file for web modules is stored in the WEB-INF folder.
- ▶ The `ejb-jar.xml` file for EJB modules is stored in the META-INF folder.
- ▶ The `ra.xml` file for resource adapter modules is stored in the META-INF folder.
- ▶ The `application-client.xml` file for application client modules is stored in the META-INF folder.

These files describe the contents of a module and allow the Java EE container to configure servlet mappings, Java Naming and Directory Interface (JNDI) names, and so forth.

Class path information specifies which other modules and utility JARs are needed for a particular module to run. This information is stored in the

manifest.mf file, which is also in the META-INF (or WEB-INF) directory of the modules.

## Deployment descriptors in Java EE 6

With Java EE 6, deployment descriptors become optional. An enterprise application can be deployed by using annotations to replace the information that was previously contained in deployment descriptors.

To generate the standard deployment descriptor for a Java EE module, either select **Generate deployment descriptor** in the Create Project window, or right-click an existing project and select **Java EE → Generate Deployment Descriptor Stub**.

Regardless of whether you use standard Java EE deployment descriptors, Rational Application Developer can also generate additional WebSphere-specific information used when deploying applications to WebSphere Application Servers. This supplemental information is stored in XML files, also in the META-INF (or WEB-INF) directory of the respective modules. Examples of information in the IBM-specific files are IBM extensions, such as servlet reloading and EJB access intents. To use these extension files, you must first generate them.

To generate IBM extension files, right-click the project to which you want to add the WebSphere-specific deployment descriptor. Then select **Java EE → Generate WebSphere XXX Deployment Descriptor**, depending on the descriptor that you want to generate. The following deployment descriptors are available:

- ▶ Bindings, which create `ibm-web-bnd.xml`, `ibm-ejb-jar-bnd.xml`, or similar files
- ▶ Extensions, which create `ibm-web-ext.xml`, `ibm-ejb-jar-ext.xml`, or similar files
- ▶ Programming Model Extensions, which create `ibm-web-ext-pme.xml`, `ibm-ejb-jar-ext-pme.xml`, or similar files

## Deployment descriptor editors

Rational Application Developer has easy-to-use editors for working with the deployment descriptors. The information that goes into the files is accessible from one page in the integrated development environment (IDE), eliminating the need to know which information to place into which file. However, if you are interested, you can click the **Source** tab of the Deployment Descriptor editor to see the text version of what is stored in that descriptor.

For example, if you open the EJB deployment descriptor, you have access to settings that are stored across multiple deployment descriptors for the EJB module, including the following files:

- ▶ EJB deployment descriptor: `ejb-jar.xml`
- ▶ Bindings file: `ibm-ejb-jar-bnd.xml`
- ▶ Extensions deployment descriptor: `ibm-ejb-jar-ext.xml`

You can modify the deployment descriptors in Rational Application Developer by double-clicking the file to open the Deployment Descriptor Editor (Figure 25-2).

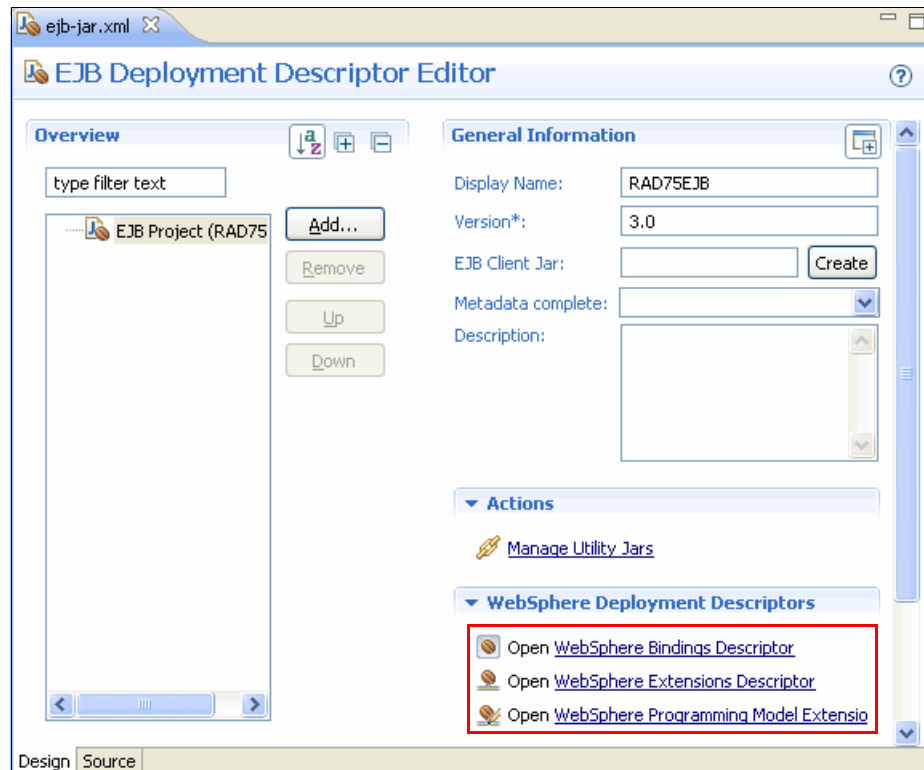


Figure 25-2 Deployment Descriptor editor for an EJB project

While the editor provides the ability to modify the content of the `ejb-jar.xml`, by using the links under Actions, you can open the IBM bindings and extensions that are stored in the WebSphere-specific deployment descriptor files (Figure 25-3 on page 1315). The descriptor files are in the META-INF directory of the module that you are editing. Click the **Source** tab to access and modify the XML source of the deployment descriptor.

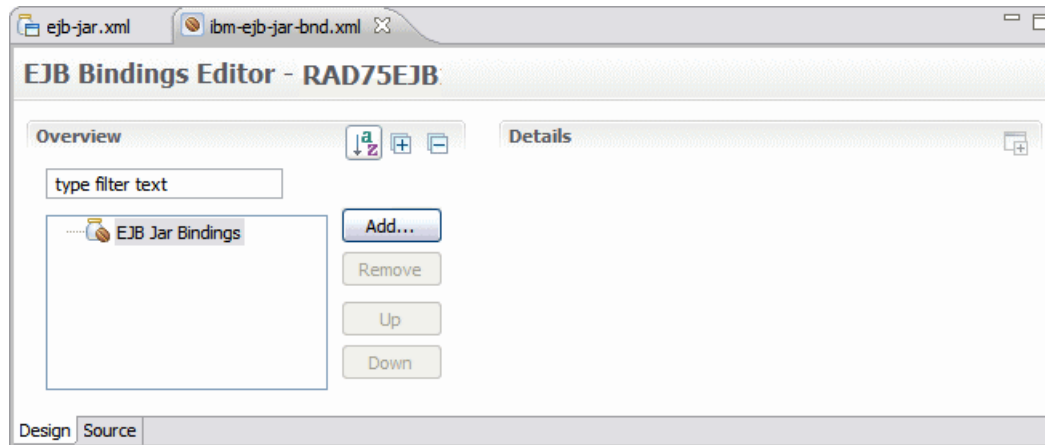


Figure 25-3 Deployment descriptor editor for bindings

## 25.1.4 WebSphere deployment architecture

In this section, we provide an overview of the deployment architecture for IBM WebSphere Application Server.

IBM Rational Application Developer includes support for WebSphere Application Server V6, V6.1, V7, and V8 Beta. You use the WebSphere administrative console to perform administrative functions for the server and applications. For example, you can configure the following items:

- ▶ J2EE Connector architecture (J2C) authentication aliases
- ▶ Data sources
- ▶ Service buses
- ▶ Java Message Service (JMS) queues and connection factories

Because of the loose coupling between Rational Application Developer and WebSphere Application Server, applications can deploy in the following ways:

- ▶ From a Rational Application Developer project to the integrated WebSphere Application Server (not applicable for WebSphere Application Server V6.0)
- ▶ From a Rational Application Developer project to a separate WebSphere Application Server runtime environment
- ▶ Through an EAR file to an integrated WebSphere Application Server (not applicable for WebSphere Application Server V6.0)
- ▶ Through an EAR file to a separate WebSphere Application Server runtime environment

In addition, WebSphere Application Server V8 Beta can obtain an EAR file from external tools and load it into Rational Application Developer. You can publish the application directly from Rational Application Developer to the application server for testing and development purposes. Or, you can use Rational Application Developer to optimize the EAR file, and a new EAR file is saved to deploy to the application server.

The Rational Application Developer functions are included on a trial basis and can be purchased easily through a downloadable license key.

IBM Rational Build Forge and IBM Rational Automation Framework for WebSphere can be used for repeated builds of the application and for deployment to the server run time as an automated task.

**For more information:** For more information about Rational Automation Framework for WebSphere, go to this website:

<http://publib.boulder.ibm.com/infocenter/rafwhelp/v7r1/index.jsp>

For details about how to configure the servers in IBM Rational Application Developer V7, see Chapter 23, “Cloud environment and server configuration” on page 1203.

## WebSphere profiles

WebSphere Application Server V6.0 introduced the concept of WebSphere profiles. WebSphere profiles have been split into two separate components:

- ▶ A set of shared product files, called *runtime files*
- ▶ A set of configuration files, known as *WebSphere profiles*, that are called *configurable files*

A WebSphere profile includes WebSphere Application Server configuration, applications, and properties files that constitute a new application server. Having multiple profiles equates to having multiple WebSphere Application Server instances for use with several applications.

In Rational Application Developer, a developer can configure multiple application servers (WebSphere profiles) for various applications with which the developer might be working. These WebSphere profiles can then be set up as test environments in Rational Application Developer (see Chapter 23, “Cloud environment and server configuration” on page 1203).

## WebSphere enhanced EAR features

WebSphere Application Server v6 also introduced the enhanced EAR feature. The enhanced EAR information, which includes settings for the resources that



are required by the application, is stored in an `ibmconfig` subdirectory of the enterprise application (EAR file) `META-INF` directory.

The enhanced EAR feature provides an extension of the Java EE EAR with additional configuration information for resources that are typically required by Java EE applications. This information is optional, but it can simplify the deployment of applications to WebSphere Application Server for selected scenarios.

You can use the Enhanced EAR editor to edit several WebSphere Application Server-specific configurations, such as Java Database Connectivity (JDBC) providers, data sources, class loader policies, substitution variables, shared libraries, virtual hosts, and authentication settings. You can change the configuration settings within the editor and publish them with the EAR at the time of the deployment.

The advantage of the tool is that it makes the testing process simpler and repeatable, because the configurations can be saved to files and then shared within a team's repository. The Enhanced EAR editor cannot configure all runtime settings, but it is convenient to configure the most common settings.

The disadvantage of the tool is that the configurations are attached to the EAR and are not available server-wide or system-wide. In the WebSphere administrative console, you can navigate to the enhanced EAR deployment information. (Select **Applications** → **Application Types** → **WebSphere enterprise applications**, select the application, and click the **Application scoped resources** link). However, you cannot modify the settings. You can only edit settings that belong to the cluster, node, and server contexts.

When you change a configuration using the Enhanced EAR editor, these changes are made within the application context. The deployer can still make changes to the EAR file using the IBM Rational Application Developer Assembly and Deploy Features for WebSphere 7.0, but it still requires a separate tool. Furthermore, in most cases, these settings depend on the node in which the application server is installed. Therefore, it might not make sense to configure them at the application context for deployment to production.

Table 25-1 on page 1318 lists the supported resources that the Enhanced EAR editor provides and the scope in which they are created.

Table 25-1 Supported enhanced EAR resources and their scope

Scope	Resources
Application	JDBC providers, data sources, substitution variables, and class loader policies
Server	Shared libraries
Cell	Java Authentication and Authorization Service (JAAS) authentication aliases and virtual hosts

To open the Enhanced EAR, right-click an enterprise application and select **Java EE** → **Open WebSphere Application Server Deployment** (Figure 25-4).

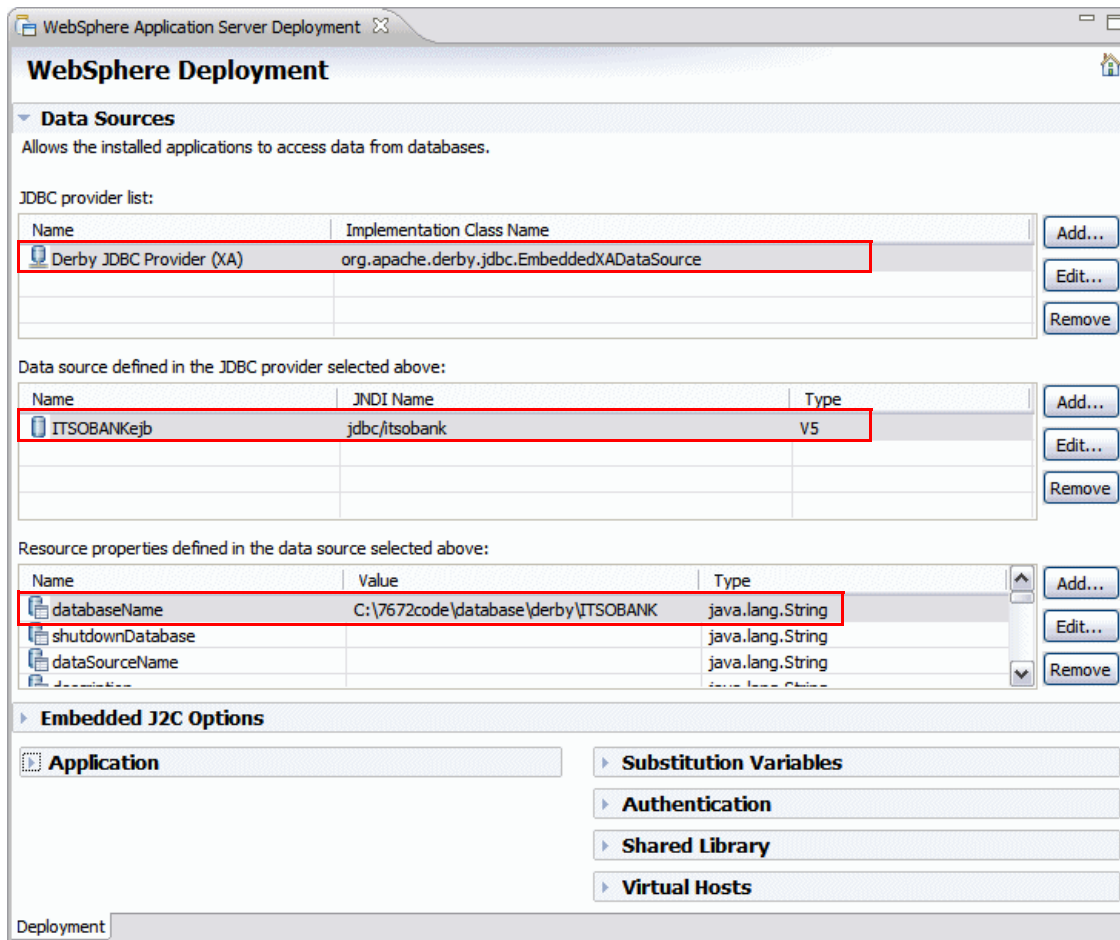


Figure 25-4 Enhanced EAR editor: Deployment Descriptor editor

Figure 25-4 on page 1318 shows the JDBC provider (Derby JDBC Provider (XA)) and a configured data source (ITSOBANKejb) with its resource properties (databaseName). Select a JDBC provider to see the list of data sources. We created this example of defining a data source by using the Enhanced EAR editor was created in 23.8.1, “Creating a data source in the Enhanced EAR editor” on page 1262.

Figure 25-5 shows the contents of the resources.xml file that is part of the enhanced EAR information that is stored in the `ibmconfig/cells/defaultCell1/applications/defaultApp/deployments/defaultApp` directory. The `ibmconfig` directory contains the familiar directories for a WebSphere cell configuration. The XML editor of Rational Application Developer shows the configuration contents for the Derby JDBC Provider (XA) and the data source.

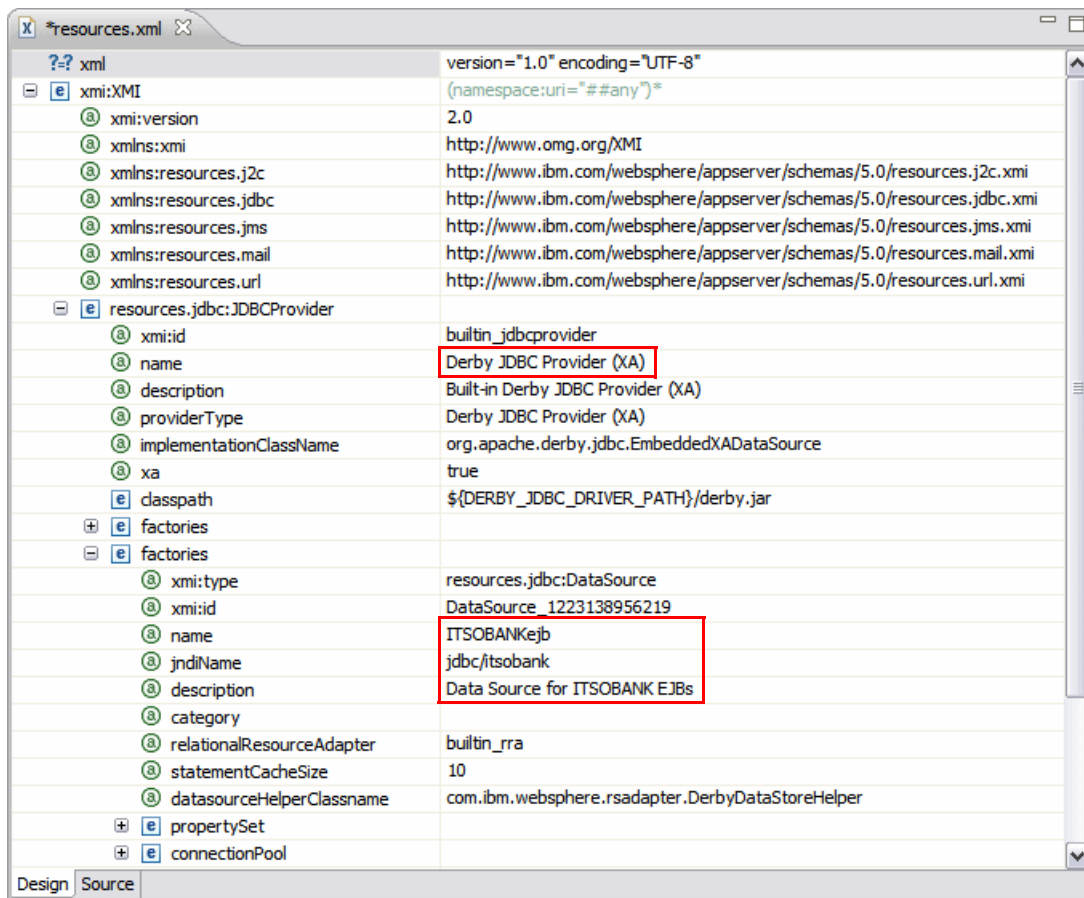


Figure 25-5 Enhanced EAR: Contents of the resources.xml file

**More information:** For more information and an example of using the WebSphere enhanced EAR, see the following sources:

- ▶ “Packaging applications” chapter in *WebSphere Application Server V6.1: Systems Management and Configuration*, SG24-7304
- ▶ IBM WebSphere Application Server V7.0 Beta Information Center  
<http://publib.boulder.ibm.com/infocenter/wasinfo/beta/index.jsp>

## WebSphere Rapid Deployment

WebSphere Rapid Deployment is a collection of tools and technologies that was introduced in IBM WebSphere Application Server V6.1 to make application development and deployment easier than ever before.

WebSphere Rapid Deployment consists of the following elements:

- ▶ Rapid deployment tools
- ▶ Fine-grained application updates

**More information:** For more information about WebSphere Rapid Deployment, see the following sources:

- ▶ *WebSphere Application Server V6.1: Planning and Design*, SG24-7305
- ▶ *WebSphere Application Server V6.1: Systems Management and Configuration*, SG24-7304
- ▶ IBM WebSphere Application Server Beta V8 Information Center  
<http://publib.boulder.ibm.com/infocenter/wasinfo/beta/index.jsp>

## 25.1.5 Java and WebSphere class loader

*Class loaders* are responsible for loading classes, which can be used by an application. Understanding how Java and WebSphere class loaders work is an important element of WebSphere Application Server configuration that is needed for the application to work properly after deployment. Failure to set up the class loaders properly often results in class loading exceptions, such as `ClassNotFoundException`, when trying to start the application.

### Java class loader

Java class loaders enable the Java virtual machine (JVM) to load classes. Given the name of a class, the class loader locates the definition of this class. Each Java class must be loaded by a class loader.

When the JVM is started, the following class loaders are used:

▶ Bootstrap class loader

The bootstrap class loader is responsible for loading the core Java libraries (that is, `core.jar` and `server.jar`) in the `<JAVA_HOME>/lib` directory. This class loader, which is part of the core JVM, is written in native code.

**Java libraries:** Beginning with JDK 1.4, the core Java libraries in the IBM Java developer kit are no longer packaged in `rt.jar`, which was previously the case (and is still the case for the Sun JDKs), but instead split into multiple JAR files.

▶ Extensions class loader

The extensions class loader is responsible for loading the code in the extensions directories (`<JAVA_HOME>/lib/ext` or any other directory that is specified by the `java.ext.dirs` system property). This class loader is implemented by the `sun.misc.Launcher$ExtClassLoader` class.

▶ System class loader

The system class loader is responsible for loading the code that is found on `java.class.path`, which ultimately maps to the system `CLASSPATH` variable. This class loader is implemented by the `sun.misc.Launcher$AppClassLoader` class.

*Delegation* is a key concept to understand when dealing with class loaders. It states that a custom class loader (a class loader other than the bootstrap, extension, or system class loader) delegates class loading to its parent before trying to load the class itself. The parent class loader can either be another custom class loader or the bootstrap class loader. Another way to look at this concept is that a class loaded by a specific class loader can only reference classes that this class loader or its parents can load, but not its children.

The extensions class loader is the parent for the system class loader. The bootstrap class loader is the parent for the extensions class loader. Figure 25-6 on page 1322 shows the class loader hierarchy.

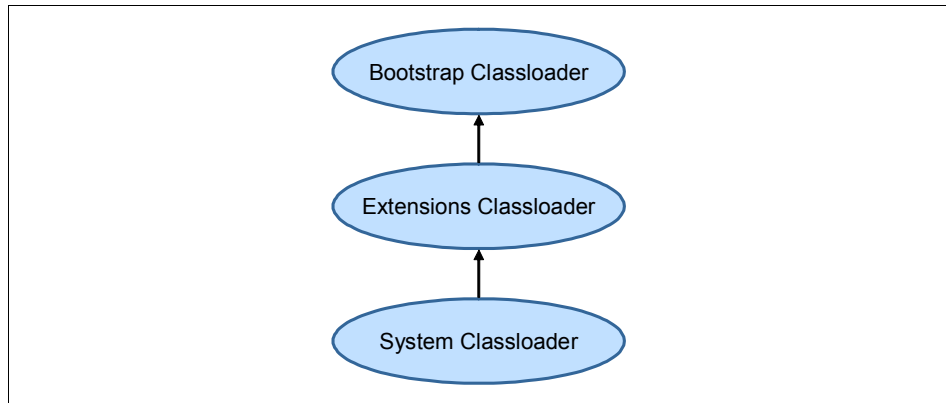


Figure 25-6 Java class loader hierarchy

If the system class loader has to load a class, it first delegates to the extensions class loader, which in turn delegates to the bootstrap class loader. If the parent class loader cannot load the class, the child class loader tries to find the class in its own repository. In this manner, a class loader is only responsible for loading classes that its ancestors cannot load.

## WebSphere class loader

**JVM:** When reading the following material about WebSphere class loaders, remember that each JVM has its own setup of class loaders. Therefore, in a WebSphere environment hosting multiple application servers (JVMs), such as a Network Deployment configuration, the class loaders for the JVMs are completely separated even if they are running on the same physical machine.

WebSphere provides several custom delegated class loaders, as shown in Figure 25-7 on page 1323. The top box represents the Java class loaders (bootstrap, extensions, and system). WebSphere does not load much here, just enough to get itself bootstrapped and initialize the WebSphere extensions class loader.

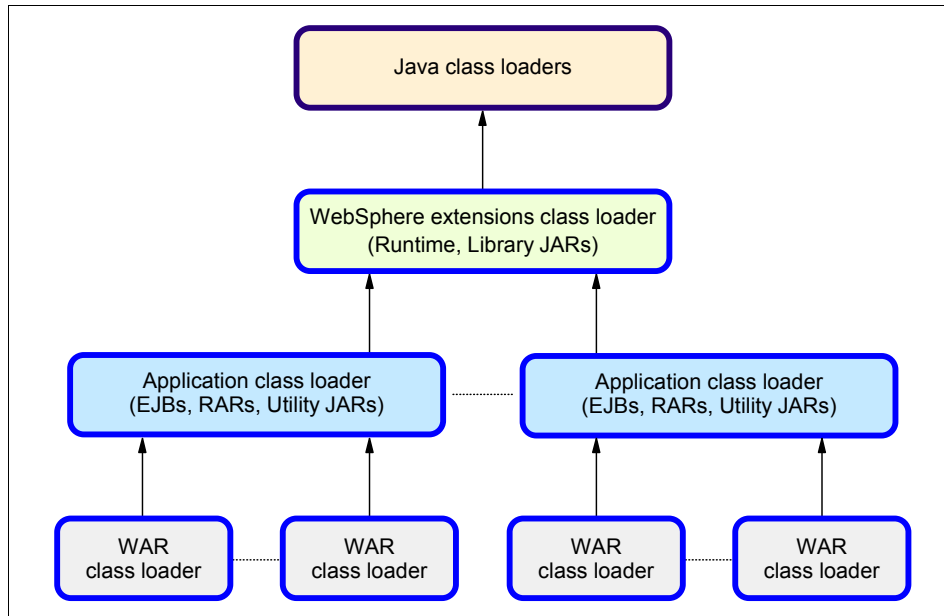


Figure 25-7 WebSphere class loaders hierarchy

The WebSphere extensions class loader is where WebSphere itself is loaded. It uses the following directories to load the required WebSphere classes:

```

<JAVA_HOME>\lib
<WAS_HOME>\classes (Runtime Class Patches directory (RCP))
<WAS_HOME>\lib (Runtime class path directory (RP))
<WAS_HOME>\lib\ext (Runtime Extensions directory (RE))
<WAS_HOME>\installedChannels

```

The WebSphere run time is loaded by the WebSphere extensions class loader based on the `ws.ext.dirs` system property, which is initially derived from the `WS_EXT_DIRS` environment variable set in the `setupCmdLine.bat` file. The following example shows the default value of `ws.ext.dirs`:

```

SET WAS_EXT_DIRS=%JAVA_HOME%\lib;%WAS_HOME%\classes;%WAS_HOME%\lib;
 %WAS_HOME%\installedChannels;%WAS_HOME%\lib\ext;%WAS_HOME%\web\help;
 %ITP_LOC%\plugins\com.ibm.etools.ejbdeploy\runtime

```

The RCP directory is intended to be used for fixes and other authorized program analysis reports (APARs) or problem reports that are applied to the application server run time. These patches override any copies of the same files that are lower in the RP and RE directories. The RP directory contains the core application server runtime files. The bootstrap class loader first finds classes in the RCP

directory and then in the RP directory. The RE directory is used for extensions to the core application server run time.

Each directory that is listed in the `ws.ext.dirs` environment variable is added to the WebSphere extensions class loaders class path. In addition, every JAR file or compressed file in the directory is added to the class path.

You can extend the list of directories and files loaded by the WebSphere extensions class loaders by setting a `ws.ext.dirs` custom property to the JVM settings of an application server.

### ***Application and web module class loaders***

Java EE applications consist of five primary elements:

- ▶ Web modules
- ▶ EJB modules
- ▶ Application client modules
- ▶ Resource adapters (RAR files)
- ▶ Utility JARs

Utility JARs contain code that is used by both EJB and servlets. Utility frameworks, such as `log4j`, are a good example of a utility JAR.

EJB modules, utility JARs, resource adapters files, and shared libraries that are associated with an application are always grouped together into the same class loader. This class loader is called the *application class loader*. Depending on the application class loader policy, this application class loader can be shared by multiple applications (EAR) or be unique for each application (the default).

By default, web modules receive their own class loader (a WAR class loader) to load the contents of the `WEB-INF/classes` and `WEB-INF/lib` directories. The default behavior can be modified by changing the application's WAR class loader policy (the default is `Module`). If the WAR class loader policy is set to `Application`, the web module contents are loaded by the *application class loader* (in addition to the EJB, RARs, utility JARs, and shared libraries). The application class loader is the parent of the WAR class loader.

The application and the web module class loaders are reloadable class loaders. They monitor changes in the application code to automatically reload modified classes. You can alter this behavior at deployment time.

### ***Handling JNI code***

Because of a JVM limitation, code that has to access native code through a Java Native Interface (JNI) must not be placed on a reloadable class path, but on a static class path. This requirement includes shared libraries for which you can define a native class path, or the application server class path. Therefore, if you



have a class loading native code through JNI, this class must not be placed in the WAR or application class loaders, but rather on the WebSphere extensions class loader.

It might make sense to break out the lines of code that actually load the native library into a class of their own and place this class on a static class loader. This way, you can have all the other code on a reloadable class loader.

## 25.2 Preparing for the EJB application deployment

In this section, we explain the required steps to prepare the environment for the deployment sample. We use the ITS0 Bank enterprise application that was developed in Chapter 12, “Developing Enterprise JavaBeans (EJB) applications” on page 577, to demonstrate the deployment process.

This section includes the following tasks:

- ▶ Reviewing the deployment scenarios
- ▶ Installing the prerequisite software
- ▶ Importing the sample application archive files
- ▶ Sample database

### 25.2.1 Reviewing the deployment scenarios

Now that the Rational Application Developer integration with the WebSphere Application Server is managed the same as a stand-alone WebSphere Application Server, the procedure to deploy the ITS0 Bank sample application is nearly identical to that of a stand-alone WebSphere Application Server.

Several configurations are possible in which our sample can be installed, but in this chapter, we deploy the ITS0 Bank application to a separate production IBM WebSphere Application Server V7.0. This scenario uses two nodes: the developer node and the application server node.

### 25.2.2 Installing the prerequisite software

The application deployment sample requires that you have the software that is mentioned in this section installed. Within the example, you can choose between DB2 Universal Database or Derby as your database server. We used Derby for this chapter and our deployment exercises.

The sample for the working example environment consists of the following nodes (see Table 25-2 on page 1326 for product mapping):

► Developer node

The *developer node* is used by the developer to import the sample code and package the application in preparation for deployment.

► Application server node

The *application server node* is used as the target server where the enterprise application will be deployed.

Table 25-2 Product mapping for deployment

Software	Version
<b>Developer node</b>	
Microsoft Windows	Microsoft Windows XP + Service Pack 2 + critical fixes and security patches
IBM Rational Application Developer Integrated IBM WebSphere Application Server	V8 V8
Derby (installed by default)	V10.2
<b>Application server node</b>	
Microsoft Windows	Microsoft Windows XP + Service Pack 2 + critical fixes and security patches
IBM WebSphere Application Server (Base stand-alone)	V8
Derby (installed by default)	V10.2

**More information:** For information about installing the required software for the sample, see Appendix A, “Installing the products” on page 1783.

**DB2 Universal Database or Derby tip:** You can use DB2 or Derby as the database server. Because Derby is installed by default as part of the application server (integrated and stand-alone), we used Derby as the database for this chapter and the deployment exercises.

### 25.2.3 Importing the sample application archive files

In this section, we explain how to import the project archive files into Rational Application Developer. The RAD8EJB.zip file contains the following projects for the ITSO Bank enterprise application that was developed in Chapter 12, “Developing Enterprise JavaBeans (EJB) applications” on page 577:

<b>RAD8JPA</b>	This Java project contains the persistence Java Persistence API (JPA) classes.
<b>RAD8EJB</b>	The EJB project with session beans.
<b>RAD8EJBTestWeb</b>	The web project is used to test the EJB beans.
<b>RAD8EJBEAR</b>	The EAR project for the enterprise application; it includes the previous modules.

A second project archive file, RAD8EJBWeb.zip, contains the web application that uses the EJB:

<b>RAD8EJBWeb</b>	A web project (JSP and servlets) for the front-end application.
<b>RAD8EJBWebEAR</b>	The EAR project for the web application. This EAR also references the RAD8EJB and RAD8JPA projects.

To import the RAD8EJB.zip project interchange file, follow these steps:

1. In the Java EE (or web) perspective, Enterprise Explorer, select **File** → **Import**.
2. Select **Import** → expand **General**, select **Existing Projects into Workspace** from the list of import sources, and then click **Next**.
3. In the Import Projects window, click **Browse** for the compressed (.zip) file, navigate to and select the **RAD8EJB.zip** file in the c:\7835code\jython folder, and click **Open**.
4. Click **Select All** to select all of the projects and then click **Finish**.
5. Repeat this sequence for the RAD8EJBWeb.zip file and select the **RAD8EJBWeb** and **RAD8EJBWebEAR** projects.

### 25.2.4 Sample database

The ITSO Bank application is based on the ITSOBANK database. See “Setting up the ITSOBANK database” on page 1880 for instructions about how to create the ITSOBANK database. You can use either the DB2 or Derby database. For simplicity, we use the built-in Derby database in this chapter.

To make it even simpler, we included the Derby database as part of the file that you downloaded for this chapter. The ITS0BANK folder under the C:\7835code\database\derby folder constitutes the Derby database. Therefore, the complete path or location of the database is C:\7835code\database\derby\ITS0BANK.

You must configure this value for the databaseName resource property for the data source that we define in the server (see 25.4.1, “Configuring the data source in the application server” on page 1331). Therefore, if you configure this location in the data source, you do not have to set up a sample Derby database, because it has already been done for you. Make sure that you test the data source connection.

## 25.3 Packaging the application for deployment

In this section, we explain how to prepare for packaging and how to export the enterprise application from Rational Application Developer to an EAR file, which is deployed on the application server. This section is mainly for users who want to use Rational Application Developer to deploy their application to the production server. For developers who are developing the applications, we recommend that you publish the application directly from Rational Application Developer, which provides much quicker publishing and debugging support in a development environment. For further information and an example of publishing applications directly from Rational Application Developer, see 23.7, “Adding and removing applications to and from a server” on page 1256.

This section includes the following procedures:

- ▶ Removing the enhanced EAR data source
- ▶ Generating the deployment code
- ▶ Exporting the EAR files

### 25.3.1 Removing the enhanced EAR data source

The application deployment descriptor contains enhanced EAR information for the JDBC provider and data source configuration. These settings are useful when running the application within Rational Application Developer.

Because we are deploying the application to a remote application server system and because the enhanced EAR data source configuration overrides the administrative console configuration, you must remove the enhanced EAR data source settings.

To remove the enhanced EAR data source settings, follow these steps:

1. Right-click the **RAD8EJB** project and select **Java EE → Open WebSphere Application Server Deployment**.
2. When the WebSphere Deployment editor opens, select **Derby JDBC Provider (XA)** from the JDBC provider list and click **Remove**.
3. Save the deployment descriptor.

### 25.3.2 Generating the deployment code

Prior to Java EE 5, deployment code had to be generated to deploy the EJB to an application server. Generating deployment code is not necessary for Java EE 5 applications. Therefore, if your EAR file contains only Java EE 5 applications, you do not have to generate deployment code. However, if you have a mix of EJB 2.x and EJB 3.0 in your EAR file, you must generate deployment code for the EJB 2.x projects.

To generate the deployment code, follow these steps:

1. Make sure that you have selected the correct back-end folder in the deployment descriptor of the EJB 2.x projects, either Derby or DB2. Deployment code is generated for the selected mapping (back-end folder).
2. Perform the required deployment for EJB 2.x. You can perform it in the integrated development environment (IDE) now or when you install the application on the server.
3. In the Enterprise Explorer, right-click the EAR project, select **Java EE** and select **Prepare for Deployment**. For a Java EE 5 EAR project, deployment code is only generated for the EJB 2.x modules that it contains.

You can also right-click the EJB 2.x projects, select **Java EE**, and select **Prepare for Deployment** (the web modules do not require deployment).

### 25.3.3 Exporting the EAR files

We have two enterprise applications for the EJB sample in the workspace: RAD8EJB and RAD8EJBWebEAR. We have to export both of these enterprise projects as EAR files.

To export the enterprise applications from Rational Application Developer to EAR files, follow these steps:

1. In the Enterprise Explorer, right-click **RAD8EJB** and select **Export → EAR file**.

- In the EAR Export window (Figure 25-8), enter the destination path (for example, C:\7835code\jython\RAD8EJBEAR.ear). Because we are deploying the application to IBM WebSphere Application Server V7.0, select **Optimize for a specific server runtime** and select **WebSphere Application Server v8.0 Beta**. Click **Finish**.

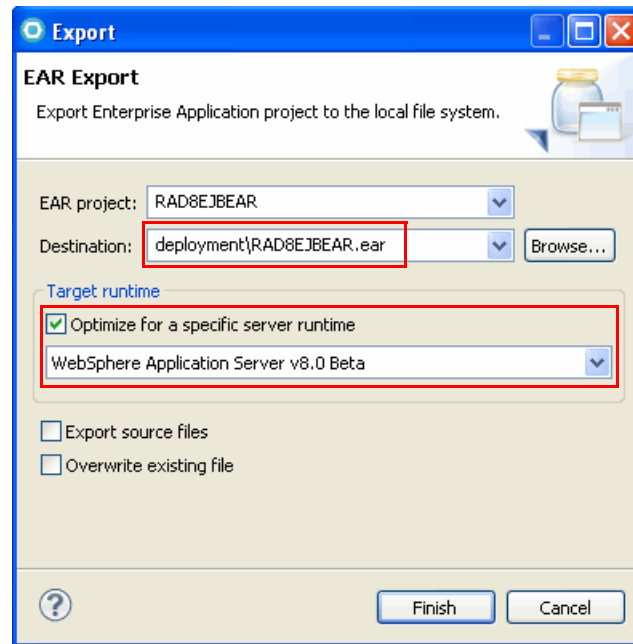


Figure 25-8 Choices for exporting an EAR file

- Repeat the export for the RAD8EJBWebEAR project.

## 25.4 Manual deployment of enterprise applications

Both enterprise applications have been packaged as an EAR file and can be deployed to the application server. In this section, we explain how to configure the target WebSphere Application Server and install the two enterprise applications.

**EAR files:** You can either use the EAR files that we exported in the previous section for deployment to the target application server, or you can use the solution EAR files from the C:\7835code\jython directory of the sample code.

## 25.4.1 Configuring the data source in the application server

You can create the data source for the application server in the following ways, among others:

- ▶ Enhanced EAR

This method is the ideal option for deploying the application to the Integrated Application Server in Rational Application Developer, but because we are deploying to the stand-alone server, we are not using the Enhanced EAR functionality.

- ▶ Scripting using the `wsadmin` command-line interface

For details, see the WebSphere Application Server v8.0 Beta Information Center:

<http://publib.boulder.ibm.com/infocenter/wasinfo/beta/index.jsp>

We also describe this method in the second part of this chapter, which explains the automated deployment using Jython-based `wsadmin` scripting.

- ▶ WebSphere administrative console

For our example, we create and configure the data source for the ITSO Bank application using the administrative console of the target stand-alone WebSphere Application Server V8 Beta.

Configuration of the data source within WebSphere Application Server for the ITSO Bank application sample consists of the following high-level tasks:

- ▶ Starting the application server
- ▶ Starting the administrative console
- ▶ Configuring the JDBC provider
- ▶ Creating the data source

### Starting the application server

Ensure that the application server where you want to deploy the applications is started. You can use any WebSphere Application Server, stand-alone or configured, as a profile in Rational Application Developer.

If the server is not running, start the server in one of the following ways:

- ▶ Use the Microsoft Windows start menu. For example, select **Start** → **Programs** → **IBM WebSphere** → **Application Server v8.0** → **Profiles** → **AppSrv01** → **Start the server**.
- ▶ If you followed the instructions in Chapter 23, “Cloud environment and server configuration” on page 1203, and installed a second WebSphere profile, start that server from the Servers view of Rational Application Developer.

- ▶ Use the **startServer server1** command in the bin folder where the server profile is installed, for example:  

```
C:\Program Files\IBM\WebSphere\AppServer\profiles\default\bin
<RAD_HOME>\runtimes\base_v8\profiles\was80profile1\bin
```
- ▶ If you configured the stand-alone application server as a Microsoft Windows service, start the server by starting its service.

**Tip:** You can also use the WebSphere Application Server V8 Beta test environment to go through the enterprise application installation windows. However, make sure that the RAD8EJBEAR and RAD8EJBWebEAR applications are removed from the server (right-click the server and select **Add and Remove Projects**).

**Verifying that the server has started:** To verify that the server has started properly, you can look for the message `Server server1 open for e-business` in the `SystemOut.log` file in the `<was_profile_root>\logs\server1` directory.

## Starting the administrative console

We use the WebSphere administrative console to define the data source and install the applications. When using a server from Rational Application Developer, you can start the administrative console by right-clicking the server and selecting **Administration** → **Run administrative console**. However, to simulate a real deployment environment, perform the server configuration in an external browser:

1. To start the administrative console, open a web browser (Internet Explorer or Firefox) and enter the following URL:

```
http://<hostname>:<port>/ibm/console
```

In the test environment, we enter the following URL:

```
http://localhost:9060/ibm/console
```

Here, we enter the URL with an alternate profile:

```
http://localhost:9062/ibm/console
```

2. For a stand-alone server, also select **Start** → **Programs** → **IBM WebSphere** → **Application Server v8.0** → **Profiles** → **default** → **Start administrative console**.
3. Click **Login** (without security, so no user ID is required).

Figure 25-9 on page 1333 shows the Welcome page.



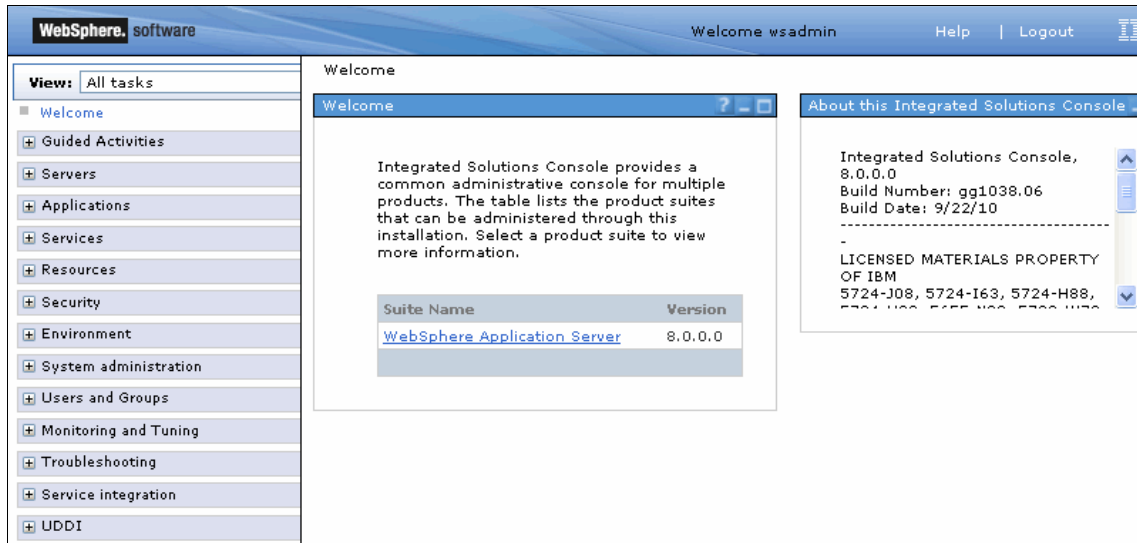


Figure 25-9 Welcome page of the administrative console

## Creating the JDBC driver variable

If you are using Derby, verify that the WebSphere variable for the Derby JDBC driver (DERBY\_JDBC\_DRIVER\_PATH) is defined.

In the left navigation pane, expand **Environment**, and select **WebSphere Variables**. In the right pane, verify that DERBY\_JDBC\_DRIVER\_PATH is defined (Figure 25-10).

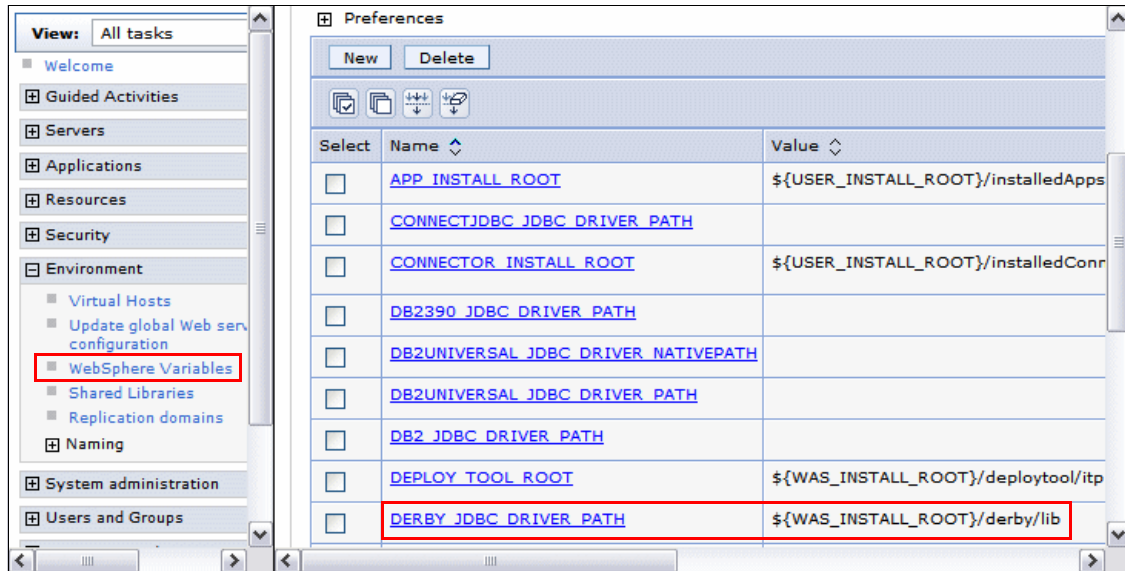


Figure 25-10 WebSphere Variable: DERBY\_JDBC\_DRIVER\_PATH

## Configuring the JDBC provider

We now configure the JDBC provider for the selected database type. The following procedure shows how to configure the JDBC provider for Derby:

1. Select **Resources** → **JDBC** → **JDBC providers**.
2. Select the server scope **Node=<hostname>Node<xx>**, **Server=server1**.
3. Click **New**.
4. In the Create new JDBC provider page (Figure 25-11 on page 1335), follow these steps:
  - a. For the Database type, select **Derby**.
  - b. For the Provider type, select **Derby JDBC Provider**.
  - c. For the Implementation type, select **XA data source**.
  - d. Accept the default name of **Derby JDBC Provider (XA)**.
  - e. Click **Next**.

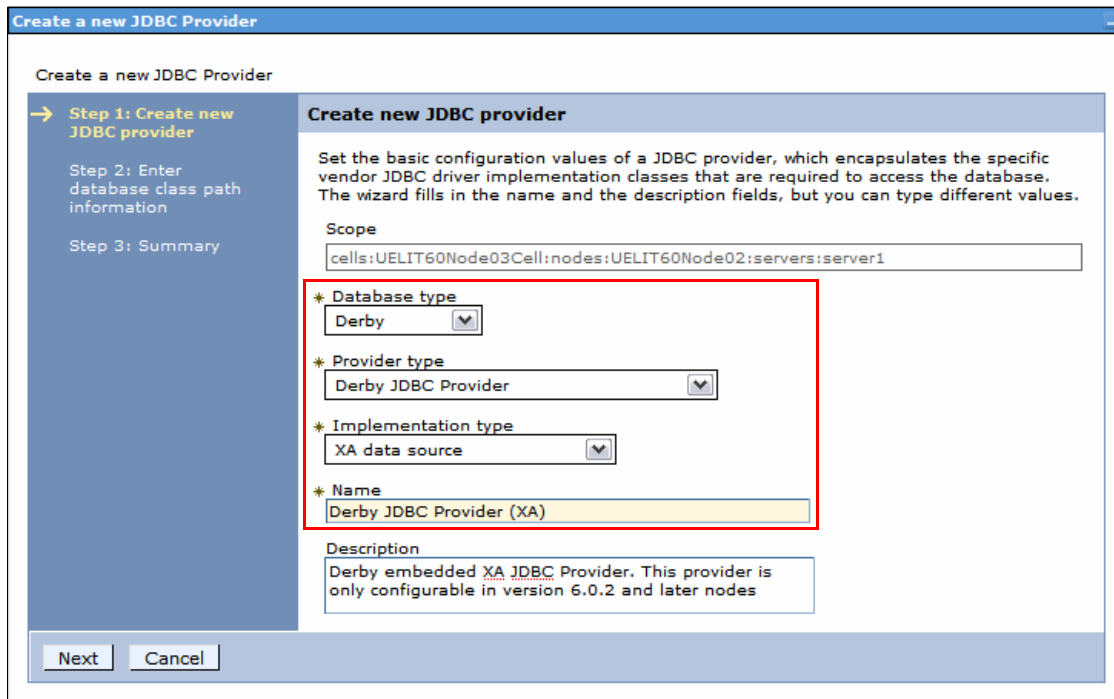


Figure 25-11 Create a JDBC provider

5. For Derby, Step 2: “Enter database class path information” is skipped (the class path is set correctly). In the Step 3: “Summary” page, click **Finish**.
6. Click **Save** to the master configuration.

## Creating the data source

Create the data source for the ITS0BANK database for the selected JDBC provider:

1. Select **Derby JDBC Provider (XA)** for Derby or **DB2 Universal JDBC Driver Provider (XA)** for DB2.
2. Under Additional Properties (right side), click **Data sources**.
3. In the Data source panel, click **New**.
4. Enter the basic configuration for the new data source (Figure 25-12 on page 1336):
  - a. For the Data source name, enter a name. We enter RAD8DS for ITS0BANK.
  - b. For the JNDI name, enter the same JNDI name that was given in the EAR enhanced deployment descriptor (see Figure 25-4 on page 1318). We enter jdbc/itsobank.

If you have already configured a data source with the JNDI name `jdbc/itsobank`, use another name, such as `jdbc/itsobank1`.

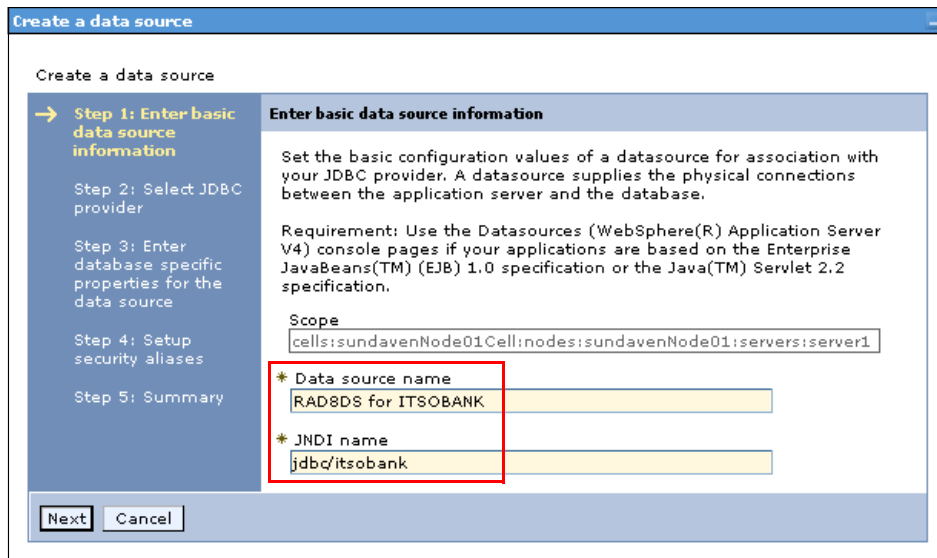


Figure 25-12 Basic configuration for the data source

5. In the “Create a data source: Enter database-specific properties for the data source” panel (Figure 25-13 on page 1337), complete the following steps:
  - a. Enter the database Name. Make sure that you enter the complete path of the database file. For example, in our case, this value is `C:\7835code\database\derby\ITSOBANK`.
  - b. Clear **Use this data source in container managed persistence (CMP)**. This option is for EJB 2.x only.
  - c. Click **Next**.

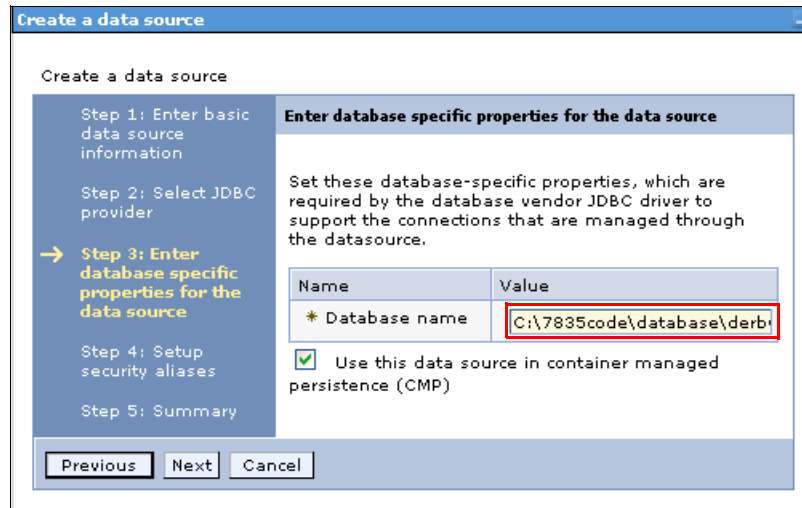


Figure 25-13 Entering the database path or name

6. On the next panel, because no authentication aliases are required for Derby, leave the fields blank. Then click **Next**.
7. In the summary panel, verify the configuration information that you entered for the data source and click **Finish**.
8. Click **Save** to the master configuration.
9. Verify the database connection for the new data source (Figure 25-14 on page 1338):
  - a. Select the data source check box.
  - b. Click **Test connection** and a message indicates success or failure.

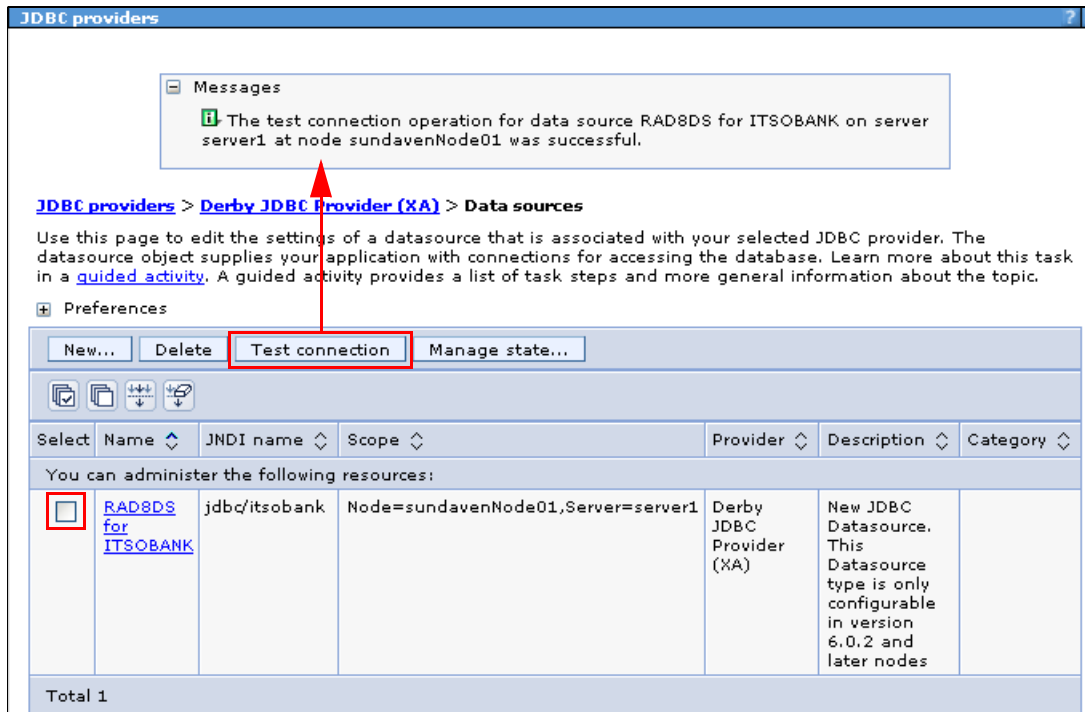


Figure 25-14 Test connection of the data source

**Tip:** If the connection fails, ensure that no connection is active from Rational Application Developer (in the Data perspective, disconnect from ITSOBANK).

## 25.4.2 Installing the enterprise applications

To install the two enterprise applications to the target application server, follow these steps:

1. Copy the RAD8EJBEAR.ear and RAD8EJBWebEAR.ear files from the developer node (where you exported the files from Rational Application Developer) to the application server node, typically, into the installableApps directory:

```
<AppServer_HOME>/installableApps
```

```
<RAD_HOME>/runtimes/base_v8/profiles/<profile>/installableApps. In the WebSphere administrative console, select Applications → New Application.
```

2. Click **New Enterprise Application**.

3. In the “Preparing for the application installation” panel (Figure 25-15), complete the following steps:
  - a. Select **Local file system**.
  - b. For Full path, point to ...installableApps\RAD8EJBEAR.ear.
  - c. Click **Next**.

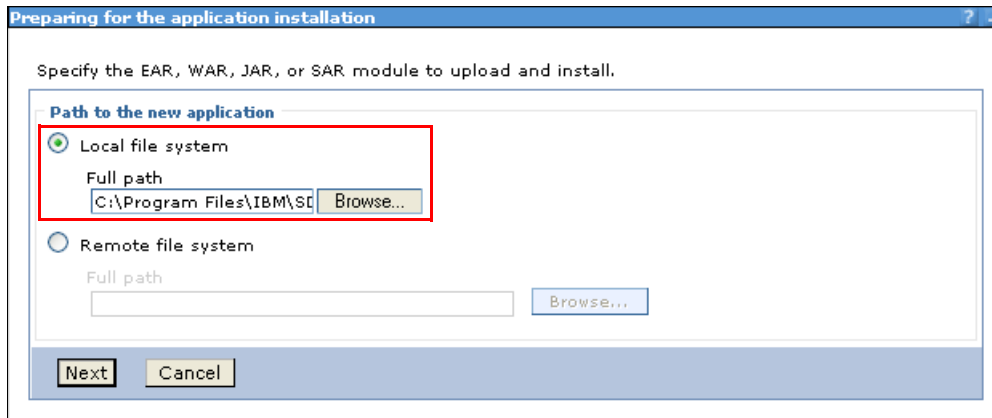


Figure 25-15 Enterprise application installation: Specifying the path to the EAR file

4. Ensure that **Fast Path** is selected and click **Next** to start the installation wizard.
5. In the Select installation options panel, accept the default values. Optional: Select **Precompile JavaServer Pages files** for applications with a web module with JSP. Then click **Next** (Figure 25-16 on page 1340).

**Install New Application**

Specify options for installing enterprise applications and modules.

**→ Step 1: Select installation options**

Step 2 Map modules to servers

Step 3 Metadata for modules

Step 4 Summary

**Select installation options**

Specify the various options that are available for your application.

Precompile JavaServer Pages files Select for web applications

Directory to install application

Distribute application

Use Binary Configuration

Deploy enterprise beans

Application name

Create MBeans for resources

Override class reloading settings for Web and EJB modules

Reload interval in seconds

Deploy Web services

Validate Input off/warn/fail

Process embedded configuration

Figure 25-16 Enterprise application installation: Installation options

6. In the Map modules to servers panel, accept the default values and click **Next**.
7. In the Metadata for modules panel, accept the default values and click **Next**.



- In the Summary page, verify the configuration information for the new enterprise application and click **Finish** to confirm (Figure 25-17).

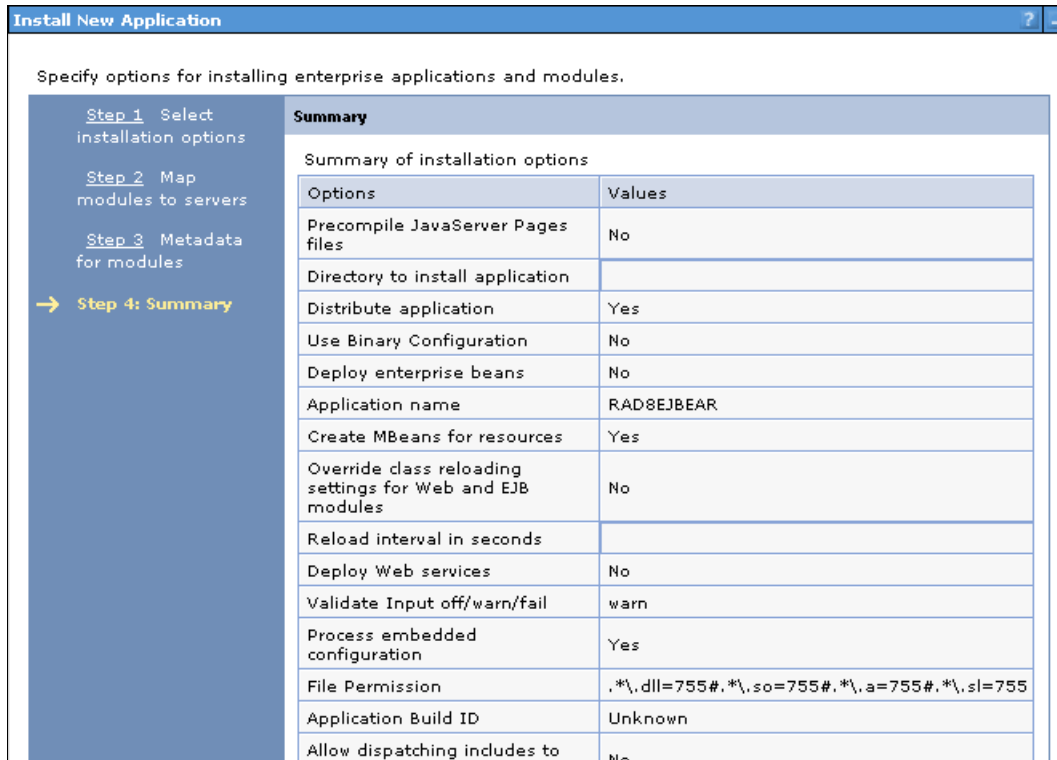


Figure 25-17 Summary page for the Install Application wizard

After several messages, you see that the installation was successful:

Installing...

If there are enterprise beans in the application, the EJB deployment process can take several minutes. Do not save the configuration until the process completes.

Check the SystemOut.log on the deployment manager or server where the application is deployed for specific information about the EJB deployment process as it occurs.

```
ADMA5016I: Installation of RAD8EJBEAR started.
```

```
.....
```

```
ADMA5005I: The application RAD8EJBEAR is configured in the WebSphere Application Server repository.
```

.....

**Application RAD8EJBEAR installed successfully.**

9. Click **Save directly to the master configuration.**
10. Repeat the installation steps to install the RAD8EJBWebEAR.ear enterprise application. In the Select installation options panel, select **Precompile JavaServer Pages files.** The rest of the steps are the same.

### 25.4.3 Starting the enterprise applications

To start the enterprise applications, follow these steps:

1. In the administrative console, select **Applications** → **Application Types** → **WebSphere enterprise applications.**
2. Select the **RAD8EJBEAR** and **RAD8EJBWebEAR** applications and click **Start** (Figure 25-18).

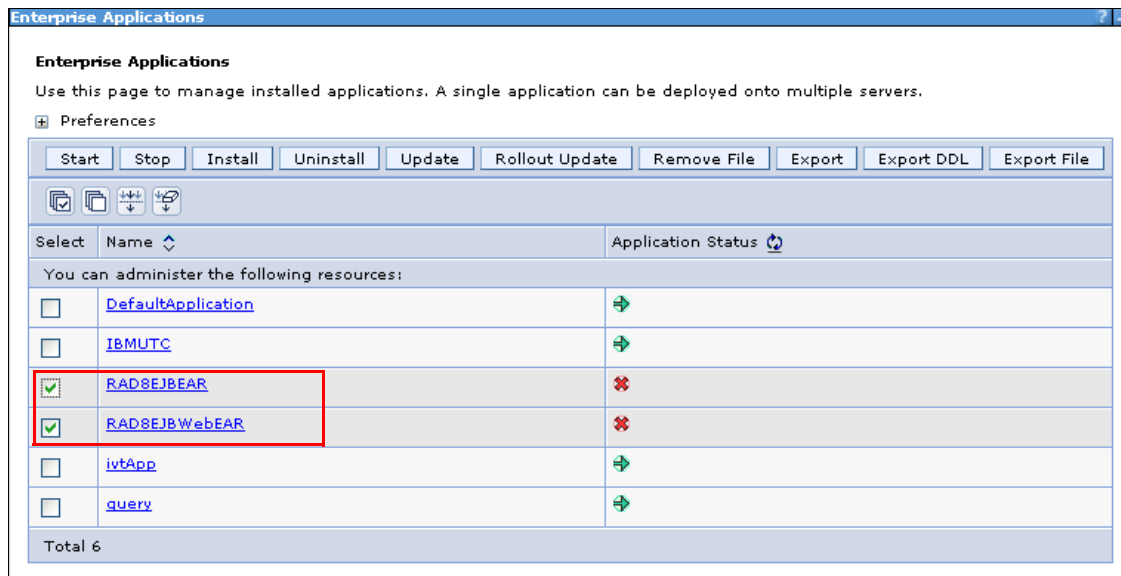


Figure 25-18 Start the deployed enterprise applications

The status for the applications changes to a green arrow, and two messages about the successful start are displayed at the top (Figure 25-19).

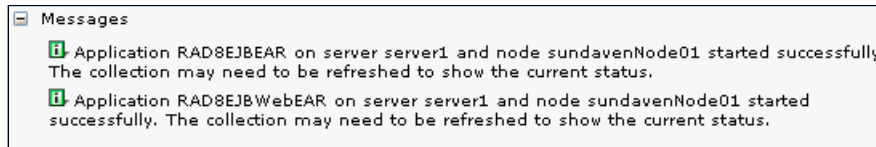


Figure 25-19 Application status messages

## 25.4.4 Verifying the application after manual installation

To verify that the ITSO Bank sample is deployed and working properly, follow these steps:

1. Enter the following URL in a browser to access the ITSO Bank application:

For a stand-alone server, enter

`http://<hostname>:9080/RAD8EJBWeb/`

For a WebSphere profile, enter

`http://localhost:9081/RAD8EJBWeb/`

2. On the ITSO RedBank home page (Figure 25-20), click **RedBank**.



Figure 25-20 ITSO RedBank: Home page

3. In the login page (Figure 25-21), enter a customer ID, for example, 555-55-5555, and click **Submit**.

ITSO RedBank

Rates RedBank Insurance

Please enter your customer ID (SSN):

555-55-5555

Submit

ITSO Home

Figure 25-21 ITSO RedBank: Login page

The accounts page lists the accounts of the customer with the customer's balance in each account (Figure 25-22).

ITSO RedBank

Rates RedBank Insurance

SSN: 555-55-5555

Title: Mr

First name: Brian

Last name: Hailey

Update New Customer Delete Customer

Account Number	Balance
005-555001	65.89
005-555002	72,213.41
005-555003	897.55

Add Account Logout

ITSO Home RedBank

Figure 25-22 ITSO RedBank: Accounts page

We have completed our testing, although you can further experiment with the application by performing these tasks:

1. Update the customer name or title.
2. Click one of the account numbers to get the account maintenance page.
3. Submit banking transactions, such as deposit, withdraw, and transfer.
4. List the transactions.
5. Log out.

### 25.4.5 Uninstalling the application

We also want to show deployment using automation scripts. After testing, we uninstall the enterprise applications, which is necessary if you want to use the same server for the automation scripts.

To uninstall the application, follow these steps:

1. In the WebSphere administrative console, select **Applications** → **Application Types** → **WebSphere enterprise applications**.
2. Select the two **RAD8EJBxxx** applications and click **Stop**.
3. Select the two **RAD8EJBxxx** applications and click **Uninstall**. Click **OK** when prompted to remove the applications.
4. Wait for the “Uninstall successful” messages and then click **Save**.

## 25.5 Automated deployment using Jython-based wsadmin scripting

In this section, we explain the WebSphere scripting client, called *wsadmin*, and the new scripting language that is used in the client, called *Jython*.

WebSphere Application Server’s administration model is based on the Java Management Extensions (JMX) framework. With JMX, you can wrap hardware and software resources in Java and expose them in a distributed environment. WebSphere’s administrative services provide functions that use the JMX interfaces to manipulate the application server configuration, which is stored in an XML-based repository in the server’s file system.

WebSphere Application Server provides the following tools that aid in the administration of its configuration:

- ▶ WebSphere administrative console. A web application.
- ▶ Command-line commands. These executable commands are in the `<was_install_root>/bin` folder and the `<was_profile_root>/bin` folder.
- ▶ **wsadmin** scripting client. A command-line interface (CLI) in which scripts can be used to automate the administration of multiple application servers and nodes.
- ▶ Thin client. A lightweight runtime package that enables you to run the **wsadmin** tool or a stand-alone administrative Java program remotely.

## 25.5.1 Overview of wsadmin

The **wsadmin** tool is a scripting client that has a CLI. It is targeted toward advanced administrators. It provides extra flexibility that is available through the web-based administrative console and helps make the administration much quicker. It is primarily used to automate administrative activities that can consist of several administrative commands and need to be executed repetitively.

The **wsadmin** client uses the Bean Scripting Framework (BSF). The BSF supports a variety of scripting languages. Prior to WebSphere Application Server V6, only one scripting language was supported, Java Tcl (Jacl). WebSphere Application Server now supports two languages: Jacl and Jython (or jpython).

Jython is the strategic direction. New tools in WebSphere Application Server v8.0 Beta and Rational Application Developer are available to help create scripts using Jython. A Jacl-to-Jython migration tool is included with the WebSphere Application Server.

The following **wsadmin** objects are available to use in the scripts:

<b>AdminControl</b>	Use this object to run operational commands.
<b>AdminConfig</b>	Use this object to create or modify WebSphere Application Server configurational elements.
<b>AdminApp</b>	Use this object to administer applications.
<b>AdminTask</b>	Use this object to run administrative commands.
<b>Help</b>	Use this object to obtain general help.

## 25.5.2 Overview of Jython

Jython is an implementation of the Python language. The `wsadmin` tool uses Jython V2.1. The J in Jython represents its Java-like syntax, but Jython also differs significantly from Java or C++ syntax. Although Jython is like Java, the name Jython is a typed, case-sensitive, and object-oriented language. Unlike Java, Jython is an indentation-based language, which means that it does not have any mandatory statement termination characters (such as a semi-colon in Java), and code blocks are specified by indentation.

To begin a code block, you indent in, and to end a block, you indent out. Statements that expect an indentation level end in a colon (:). Functions are declared with the `def` keyword. And, comments start with a number sign (#). Example 25-1 shows a Jython code snippet.

*Example 25-1 Jython snippet*

---

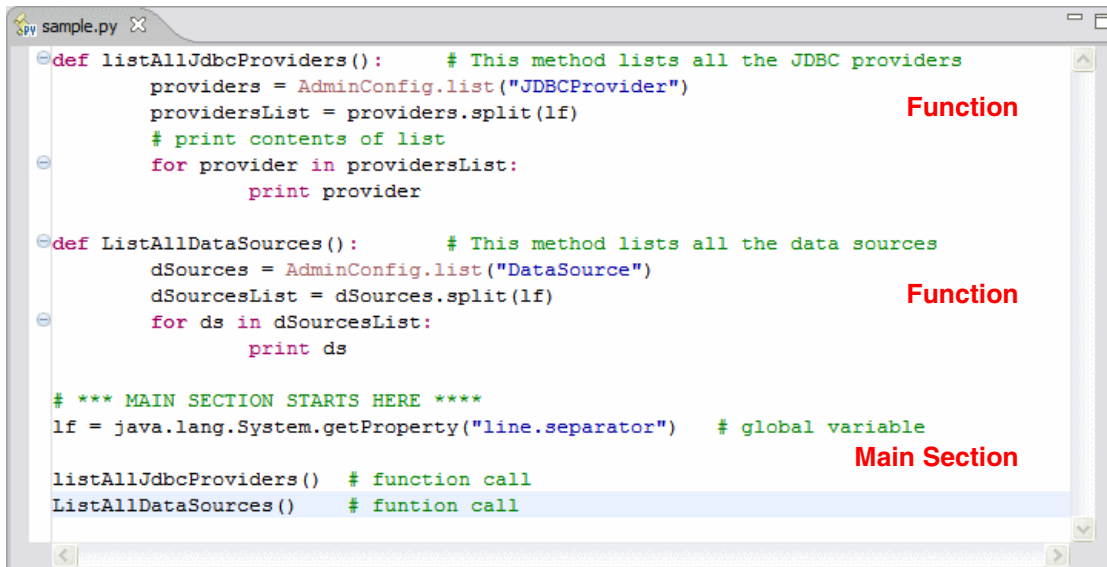
```
def listAllApps # This function lists all application
 apps = AdminApp.list()
 if len(apps) == 0:
 print "Inside if block - No enterprise applications"
 else:
 print "Inside the else block"
 appsList = apps.split(1Sep)
 for app in appsList:
 print app

This is a comment - Main section starts here
1Sep = java.lang.System.getProperty('line.separator')
listAllApps # call the listAllApps function defined above
```

---

### Structure of a Jython script

A Jython script usually consists of method definitions and a main section. The outline view of the Rational Application Developer lists the methods of the script file for easy code navigation. The main section is at the end of the Jython script and consists of the declaration of global variables, which can be used across all the methods. The main section then calls functions (Figure 25-23 on page 1348).



```
def listAllJdbcProviders(): # This method lists all the JDBC providers
 providers = AdminConfig.list("JDBCProvider")
 providersList = providers.split(1f)
 # print contents of list
 for provider in providersList:
 print provider
Function

def ListAllDataSources(): # This method lists all the data sources
 dSources = AdminConfig.list("DataSource")
 dSourcesList = dSources.split(1f)
 for ds in dSourcesList:
 print ds
Function

*** MAIN SECTION STARTS HERE ****
lf = java.lang.System.getProperty("line.separator") # global variable
Main Section

listAllJdbcProviders() # function call
ListAllDataSources() # function call
```

Figure 25-23 Structure of a Jython script

### 25.5.3 Developing a Jython script to deploy the ITSO Bank

In this section, we guide you through a step-by-step process and complete a Jython script that deploys the RAD8EJBWebEAR and RAD8EJB EAR files as enterprise applications to a target WebSphere Application Server V7.0. You must perform the following steps:

1. Create a JDBC provider.
2. Create and configure a data source.
3. Install the ITSO Bank web application.
4. Install the ITSO Bank EJB application.
5. Start both applications.

**More information:** See 23.11, “Developing automation scripts” on page 1275, about Jython.

#### Creating a Jython project and script

In this section, you create a Jython project and a Jython script in Rational Application Developer. The RAD8EJB EAR and RAD8EJBWebEAR.ear files are available in the C:\7835code\deployment folder (where you exported the files), or you can use the solution files from C:\7835code\jython.



To create a Jython project and script, follow these steps:

1. Create a Jython project in Rational Application Developer. If you followed the instructions in Chapter 23, “Cloud environment and server configuration” on page 1203, you already have this project. Otherwise, create the project:
  - a. Click **File** → **New** → **Project**.
  - b. In the New Project wizard, select **Jython** → **Jython Project** and click **Next**.
  - c. For the Project name, type RAD8Jython and click **Finish**.
2. Create a Jython script to deploy the ITS0 Bank application:
  - a. Select **File** → **New** → **Other** → **Jython** → **Jython Script File** and click **Next**.
  - b. In the Parent folder field, specify /RAD8Python, and for the File name, type deployITS0BankApp.py.
  - c. Click **Finish** to create the Jython script file.
3. In the Servers view, decide which server to use for testing. You can use the test environment or the server profile that you defined in Chapter 23, “Cloud environment and server configuration” on page 1203.

## Defining the global variables

Define the global variables:

1. Open the Jython script (if you closed it).
2. Add the following global variables:

```
global AdminConfig
global AdminControl
global AdminApp
global AdminTask
```

3. Define a global variable for the deployment server node name and the deployment server:

```
nodeName = AdminControl.getNode()
srvrInfo=AdminConfig.list('Server')
srvr=AdminConfig.showAttribute(srvrInfo, 'name')
```

4. Define a global variable for the name of the new JDBC provider, the data source, and the database:

```
jdbcProv = "ITS0 Derby JDBC Provider (XA)"
dataSourceName = "RAD8JythonDS"
jndiDS = "jdbc/itsobank2"
dbasename="C:/7835code/database/derby/ITS0BANK"
```

5. Define a global variable to store the name of the new connection factory that is used for EJB container-managed persistence:

```
cfname = "RAD8JythonDS_CF"
```

6. Define global variables to store the path of the EAR files and the enterprise application names:

```
webappEAR = "C:/7835code/jython/RAD8EJBWebEAR.ear"
ejbappEAR = "C:/7835code/jython/RAD8EJBEAR.ear"
webappEARName = "RAD8EJBWebEAR" # display name of the enterprise App
ejbappEARName = "RAD8EJBEAR" # display name of the enterprise App
```

7. Create function calls for the six deployment steps:

```
createProvider() # Step 1 - Create the JDBC
 provider
createDS() # Step 2 - Create the data
 source
installApp(webappEAR, webappEARName) # Step 3 - Install ITS0 Bank
 Web App
installApp(ejbappEAR, ejbappEARName) # Step 4 - Install ITS0 Bank
 EJB App
startApp(ejbappEARName) # Step 5 - Start ITS0 Bank EJB
startApp(webappEARName) # Step 6 - Start ITS0 Bank Web
 App
```

Next we define the six functions.

**Important:** The function code goes before the main section.

## Creating a JDBC provider

Create a function named *createProvider* for this purpose:

1. Define the `createProvider` function, which encapsulates the code for creating a new JDBC provider:

```
def createProvider():
```

2. Enter the following command to check whether the JDBC provider that we want to create already exists. If it exists, we return. Make sure that the rest of the code for this method is indented.

```
prov = AdminControl.completeObjectName("name=" + jdbcProv +
 ",type=JDBCProvider,Server="+srvr + ",node="+nodeName +
 ",*")
if len(prov) > 0:
 return
#endif
```

3. Define local variables for the attributes that are required to create a new JDBC provider:

```
provName=['name',jdbcProv]
impClass=['implementationClassName',
 'org.apache.derby.jdbc.EmbeddedXADataSource']
jdbcAttrs=[]
jdbcAttrs.append(provName)
jdbcAttrs.append(impClass)
```

4. Use a template to create the JDBC provider:

```
tplName = 'Derby JDBC Provider (XA)'
templates =

AdminConfig.listTemplates("JDBCProvider",tplName).split(lineSeparat
or)
tpl = templates[0]
serverId = AdminConfig.getid("/Node:" + nodeName + "/Server:" + srvr
+ "/")
```

5. Create the JDBC provider by using the template, save the configuration, and end:

```
AdminConfig.createUsingTemplate("JDBCProvider",serverId,jdbcAttrs,tm
pl)
AdminConfig.save()
#enddef
```

Example 25-2 shows the complete code for this function.

*Example 25-2 Creating a JDBC provider using the Derby JDBC Provider (XA)*

---

```
def createProvider():
 prov = AdminControl.completeObjectName("name="+jdbcProv + ",
 type=JDBCProvider,Server="+srvr + ",node="+nodeName + ",*")
 if len(prov) > 0:
 return
 #endif

 provName=['name',jdbcProv]
 impClass=['implementationClassName',
 'org.apache.derby.jdbc.EmbeddedXADataSource']
 jdbcAttrs=[]
 jdbcAttrs.append(provName)
 jdbcAttrs.append(impClass)
 tplName = 'Derby JDBC Provider (XA)'
 templates = AdminConfig.listTemplates("JDBCProvider", tplName)
 .split(lineSeparator)
```

```

 tmp1 = templates[0]
 serverId = AdminConfig.getid("/Node:" + nodeName + "/Server:" + srvr
+ "/")

AdminConfig.createUsingTemplate("JDBCProvider",serverId,jdbcAttrs,tmp1)
 AdminConfig.save()
#enddef

```

---

## Creating a data source

To create a data source, follow these steps:

1. Create a function named createDS for this purpose:

```
def createDS():
```

2. Check whether the data source that we want to create already exists. If the data source already exists, we do not create it. Again, make sure that the rest of the code for this method is indented.

```

 dsId = AdminConfig.getid("/JDBCProvider:"+jdbcProv + "/DataSource:"+
 dataSourceName + "/")
 if len(dsId) > 0:
 return
 #endif

```

3. Define local variables for the attributes that are required to create a data source:

```

 dsname=['name',dataSourceName]
 jndiName=['jndiName',jndiDS]
 description=['description','ITSOBank Data Source']
 dsHelperClassname=['datasourceHelperClassname',
 'com.ibm.websphere.rsadapter.DerbyDataStoreHelper']

 dsAttrs=[]
 dsAttrs.append(dsname)
 dsAttrs.append(jndiName)
 dsAttrs.append(description)
 dsAttrs.append(dsHelperClassname)
 provId = AdminConfig.getid("/Node:"+nodeName + "/Server:"+srvr +
 "/JDBCProvider:"+jdbcProv + "/")

```

4. Create the data source and save the configuration:

```

AdminConfig.create('DataSource',provId,dsAttrs)
AdminConfig.save()

```

5. Configure the data source and add resource property set attributes, such as database name, password, description, and login timeout. Because this piece

of code is more than a few lines, place the code into a separate function. Let us define a function call:

```
modifyDS()
```

6. Enable the use of this data source in container-managed persistence. We use a separate method as well.

```
useDSinCMP()
```

Example 25-3 shows the code for this function.

*Example 25-3 Creating a data source*

---

```
def createdS():
 dsId = AdminConfig.getid("/JDBCProvider:"+jdbcProv + "/DataSource:"+
 dataSourceName + "/")
 if len(dsId) > 0:
 return
 #endif
 dsname=['name',dataSourceName]
 jndiName=['jndiName',jndiDS]
 description=['description','ITSOBank Data Source']
 dsHelperClassname=['datasourceHelperClassname',
 'com.ibm.websphere.rsadapter.DerbyDataStoreHelper']
 dsAttrs=[]
 dsAttrs.append(dsname)
 dsAttrs.append(jndiName)
 dsAttrs.append(description)
 dsAttrs.append(dsHelperClassname)
 provId = AdminConfig.getid("/Node:"+nodeName + "/Server:"+srvr +
 "/JDBCProvider:"+jdbcProv + "/")
 AdminConfig.create('DataSource',provId,dsAttrs)
 AdminConfig.save()
 modifyDS() # modify DS to add the properties (databaseName)
 useDSinCMP() # enable this DS in Container Managed Persistence (CMP)
#enddef
```

---

## Modifying the data source with properties

Even though we have already written the code to create the data source, you must still add a resource property set with attributes, such as the database name, password, description, and login timeout. The code in Example 25-4 on page 1354 is the complete code for the `modifyDS` function.

```
def modifyDS():
 dsId = AdminConfig.getid("/JDBCProvider:"+jdbcProv + "/DataSource:"+
 dataSourceName + "/")
 dbnameAttrs = [{"name", "databaseName"}, {"value", dbname},
{"type",
 "java.lang.String"}, {"description", "This is a required
property"}]
 descrAttrs = [{"name", "description"}, {"value", ""}, {"type",
 "java.lang.String"}]
 passwordAttrs = [{"name", "password"}, {"value", ""}, {"type",
 "java.lang.String"}]
 loginTimeOutAttrs = [{"name", "loginTimeout"}, {"value", 0},
{"type",
 "java.lang.Integer"}]

 propset = []
 propset.append(dbnameAttrs)
 propset.append(descrAttrs)
 propset.append(passwordAttrs)
 propset.append(loginTimeOutAttrs)
 pSet = ["propertySet", [{"resourceProperties", propset}]]
 attrs = [pSet]
 AdminConfig.modify(dsId, attrs)
 AdminConfig.save()
#enddef
```

---

### Using the data source in container-managed persistence

Configure this data source for use in the container-managed persistence (CMP) of entity EJB. This task is simple and easy when using the administrative console (Figure 25-24 on page 1355). However, when using a script, this task is a multiple step process. For EJB 3.0 and JPA, you do not have to perform this step.

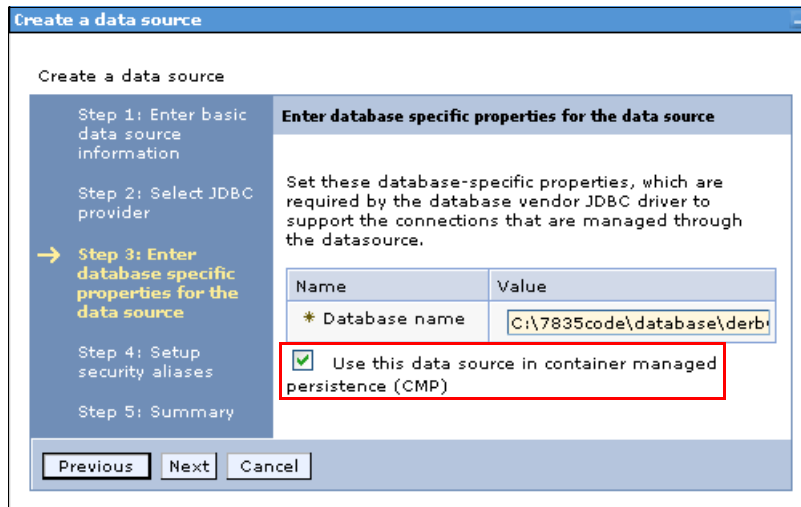


Figure 25-24 Enabling a data source to be used in CMP

The code in Example 25-5 is the complete code for the `useDSinCMP` function.

Example 25-5 Enabling the data source for container-managed persistence

```
def useDSinCMP():
 dsId = AdminConfig.getid("/JDBCProvider:"+jdbcProv +
 "/DataSource:"+dataSourceName + "/")
 rra = AdminConfig.getid("/Node:"+nodeName+"/Server:"+srvr
 +"/J2CResourceAdapter:WebSphere Relational Resource
Adapter/")
 nameAttr = ["name", cname]
 authmechAttr = ["authMechanismPreference", "BASIC_PASSWORD"]
 cmpdsAttr = ["cmpDatasource", dsId]
 attrs = []
 attrs.append(nameAttr)
 attrs.append(authmechAttr)
 attrs.append(cmpdsAttr)
 newcf = AdminConfig.create("CMPConnectionFactory", rra, attrs)
 AdminConfig.save()
 # Modify the CMPConnectionFactory to add the Mapping attributes
 mapAuthAttr = ["authDataAlias", ""]
 mapConfigAliasAttr = ["mappingConfigAlias", ""]
 mapAttrs = []
 mapAttrs.append(mapAuthAttr)
 mapAttrs.append(mapConfigAliasAttr)
 mappingAttr = ["mapping", mapAttrs]
 attrs2 = []
```

```

attrs2.append(mappingAttr)
cfId = AdminConfig.getId("/CMPConnectorFactory:"+cfname+"/")
AdminConfig.modify(cfId, attrs2)
AdminConfig.save()
AdminConfig.modify(dsId, attrs2)
AdminConfig.save()
#enddef

```

---

## Installing the enterprise applications

You have now written the code that is required to create a JDBC provider and a data source for the ITS0 Bank applications. Therefore, you are ready to start installing the EAR files. The best practice is to create a generic method that encapsulates the code to install any EAR file when provided with the location of the EAR file and the display name of the application.

The code in Example 25-6 is the complete code for the `installApp` function.

*Example 25-6 Installing or updating an enterprise application*

---

```

def installApp(appEAR, appName):
 try:
 app = AdminApp.view(appName)
 except:
 options = "-appname " + appName
 AdminApp.install(appEAR, options)
 AdminConfig.save()
 else:
 if app > 1:
 options = "-operation update -contents " + appEAR
 contentType = 'app'
 AdminApp.update(appName, contentType, options)
 AdminConfig.save()
#enddef

```

---

In the main section, we have already defined the variables that contain the complete paths of the EAR files and their display names. Therefore, we can use these variables as parameters and use the `installApp` function to install the RAD8EJBEAR and RAD8EJBWebEAR applications:

```

installApp(webappEAR, webappEARName)
installApp(ejbappEAR, ejbappEARName)

```



## Starting the enterprise applications

After installing the applications, start both enterprise applications. Again, create a generic function that can start any enterprise application when provided with the display name of that application.

The code in Example 25-7 is the complete code for the `startApp` function.

*Example 25-7 Starting an enterprise application*

---

```
def startApp(appName):
 app = AdminControl.completeObjectName
 ("type=Application,name="+appName+",*")
 if len(app) > 1:
 return

 appMgr = AdminControl.queryNames

 ("node="+nodeName+",type=ApplicationManager,process="+srvr+",*")

 AdminControl.invoke(appMgr,'startApplication',appName)
#enddef
```

---

Again, you can use the global variables that contain the display names of both applications and pass them as parameters to the `startApp` function:

```
startApp(ejbappEARName)
startApp(webappEARName)
```

**Tip:** The complete code of the `deployITS0BankApp.py` Jython script is available in the sample code that is located in the file named `C:\7835code\jython\deployITS0BankApp.py`. Import this file into the RAD8Jython project (or copy and paste it from Microsoft Windows Explorer into Rational Application Developer). In addition to the logic that we have discussed, the script has many comments and produces test output to the console (print statements) so that the execution can be followed.

### 25.5.4 Executing the Jython script

We plan to execute the Jython script against the application server that is already configured in Rational Application Developer.

We used a JNDI name of `jdbc/itsobank2` for the new data source, so that we do not have a conflict with an existing data source in the server.

To execute the Jython script, follow these steps:

1. Make sure that the target server is started.
2. Right-click **deployITSOBankApp.py** and select **Run As** → **Administrative Script**.
3. In the Edit Configuration: Edit configuration and launch window (Figure 25-25), follow these steps:
  - a. For the Scripting runtime environment, select **WebSphere Application Server v8.0 Beta**.
  - b. For the WebSphere Profile name, select **AppSrv01**.
  - c. Specify the user ID and password if the server runs with security enabled.
  - d. Click **Run** to execute the Jython script.

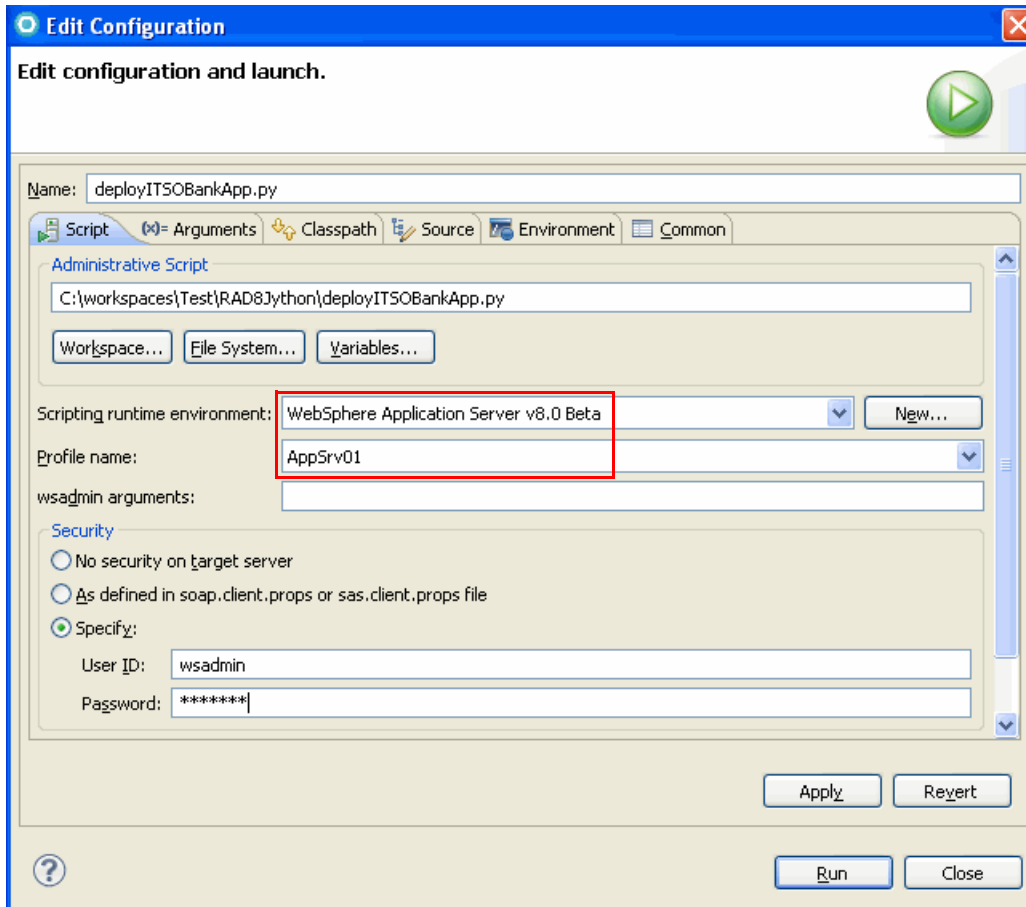


Figure 25-25 Selecting the Scripting runtime environment and the WebSphere Profile name

Example 25-8 shows the result of the execution that is displayed in the Console view.

*Example 25-8 Output of the Jython script*

---

```
WASX7209I: Connected to process "server1" on node UELIT60Node02 using
SOAP connector; The type of process is: UnManagedProcess

***** STEP 1 completed - The ITSO Derby JDBC Provider (XA) has been
created Done creating the DS. Config saved ..
Done modifying the DS. Config Saved ..

***** STEP 2 completed - Done creating the DS. Config saved ..
***** App not found: RAD8EJBWebEAR. Installing it now

ADMA5016I: Installation of RAD8EJBWebEAR started.
ADMA5058I: Application and module versions are validated with versions
of
 deployment targets.
ADMA5005I: The application RAD8EJBWebEAR is configured in the WebSphere
 Application Server repository.
ADMA5053I: The library references for the installed optional package
are
 created.
ADMA5005I: The application RAD8EJBWebEAR is configured in the
WebSphere....
ADMA5001I: The application binaries are saved in C:\<WAS_HOME>\profiles
 \AppSrv02\wstemp\Script1144d386b1d\workspace\cells\<cell>
 \applications\RAD8EJBWebEAR.ear\RAD8EJBWebEAR.ear
ADMA5005I: The application RAD8EJBWebEAR is configured in the WebSphere
...
SECJ0400I: Successfully updated the application RAD8EJBWebEAR with the
 appContextIDForSecurity information.
ADMA5011I: The cleanup of the temp directory for application
RAD8EJBWebEAR is complete.
ADMA5013I: Application RAD8EJBWebEAR installed successfully.

***** Done installing App: RAD8EJBWebEAR. Config saved ..
***** App not found: RAD8EJBWebEAR. Installing it now

ADMA5016I: Installation of RAD8EJBWebEAR started.
.....
ADMA5013I: Application RAD8EJBWebEAR installed successfully.

***** Done installing App: RAD8EJBWebEAR. Config saved ..
```

\*\*\*\*\* The startApplication operation for RAD8EJBEAR completed.

\*\*\*\*\* The startApplication operation for RAD8EJBWebEAR completed.

---

## 25.5.5 Verifying the application after automatic installation

You can find the ITSO Derby JDBC Provider (XA) and the RAD8Jython data source by using the WebSphere administrative console.

To verify that the ITSO Bank sample is deployed and working properly, follow the instructions in 25.4.4, “Verifying the application after manual installation” on page 1343.

The verification concludes this chapter, because we have successfully deployed the enterprise applications, both manually by using the WebSphere administrative console and by using the Jython scripting with the Jython tooling that is provided by Rational Application Developer V7.5.

## 25.5.6 Generating WebSphere admin commands for Jython scripts

You can use the WebSphere administration command assist tool to generate WebSphere administrative **wsadmin** commands that use the Jython scripting language as you interact with the WebSphere administrative console. When you perform server operations in the WebSphere administrative console, the WebSphere Administration command assist tool captures and shows the **wsadmin** commands that are issued. You can transfer the output from the WebSphere Administration Command view directly to a Jython editor, so that you can develop Jython scripts based on actual console actions.

To generate **wsadmin** commands as you interact with the WebSphere administrative console, follow these steps:

1. Enable the command assistance notification option in the WebSphere administrative console:
  - a. Make sure that the server is started.
  - b. Right-click the server **WebSphere Application Server v8.0 Beta at localhost** and select **Administration** → **Run administrative console**.
  - c. Specify the User ID and the Password if the server is secured and click **Log in**.
  - d. In the left pane, expand **Applications** → **Application Types** → **WebSphere enterprise applications**.

- e. Scroll to the right of the Enterprise Applications pane and under the Command Assistance section, click **View administrative scripting command for last action**.
- f. Expand **Preferences** and select **Enable command assistance notifications** (Figure 25-26). Click **Apply** and close the window.

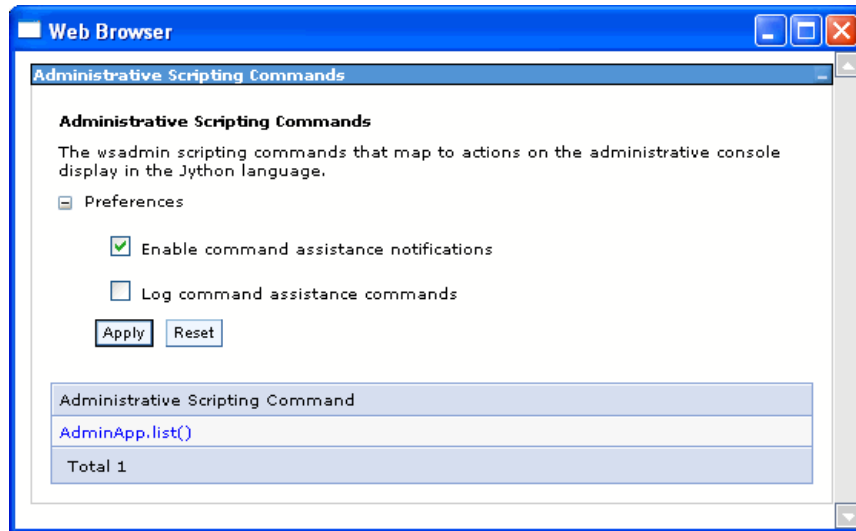


Figure 25-26 Administrative Scripting Commands

2. In the Servers view, right-click the server and select **Administration** → **WebSphere administration command assist**. The WebSphere Administration Command window opens. The view might open in the upper-right section of your window, but you can move it to a better place, such as where the Servers and Console views are.
3. From the Select Server to Monitor list, select **WebSphere Application Server v8.0 Beta at localhost** to ensure that the server is selected.
4. In the WebSphere administrative console, select **Applications** → **Application Types** → **WebSphere enterprise applications** in the left pane. You see that the WebSphere Administration Command view is populated with a `wsadmin` command for Jython.

**Delay:** You might experience a delay of a few seconds before you see the Jython command in the WebSphere Administration Command view.

5. In the left pane of the WebSphere administrative console, select **Resources** → **JDBC** → **JDBC Providers**. Another command is displayed in the WebSphere Administration Command window (Figure 25-27).

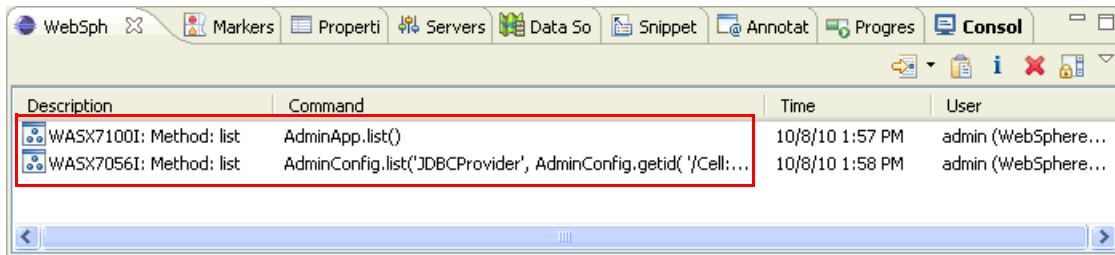


Figure 25-27 WebSphere Administration Command view

6. Create a Jython script file in the RAD8Jython project and name it `CommandAssist.py`.
7. To transfer the **wsadmin** commands that were generated in the WebSphere Administration Command view to the Jython script, follow these steps:
  - a. Make sure that the Jython editor for `CommandAssist.py` is open.
  - b. In the Jython editor, place the cursor at the bottom of the editor window.
  - c. In the WebSphere Administration Command view, press the Shift key and click to select both commands. Right-click the commands and select **Insert**.

In the editor, two commands are added:

```
AdminApp.list()
AdminConfig.list(\
'JDBCProvider', AdminConfig.getid(\
'/Cell:coutinhoNode01Cell/'))
```

8. Add the print method in front of each command so that you have the following steps:

```
print AdminApp.list()
print AdminConfig.list(\
'JDBCProvider', AdminConfig.getid(\
'/Cell:KLCHL2YNode01Cell/'))
```
9. Save the file.
10. Right-click **CommandAssist.py** and select **Run As** → **Administrative Script**.
11. On the Script tab of the WebSphere Administrative Script Launcher, select the following items:
  - a. For the Scripting runtime environment, select **WebSphere Application Server v8.0 Beta**.
  - b. For the WebSphere Profile name, select **AppSrv01**.

- c. In the Security section, if administrative security is enabled, enter the required User ID and Password.
12. Click **Apply** and then click **Run** to execute the script. You see console output similar to the output that is listed in Example 25-9.

*Example 25-9 Console output*

---

```
WASX7209I: Connected to process "server1" on node coutinhoNode01
using SOAP connector; The type of process is: UnManagedProcess
DefaultApplication

IBMUTC
WebServicesEAR
ivtApp

query
"Derby JDBC Provider
(XA)(cells/coutinhoNode01Cell/applications/commsvc.ear/deployments/c
ommsvc|resources.xml#builtin_jdbcprovider)"

"Derby JDBC Provider
(XA)(cells/coutinhoNode01Cell/nodes/coutinhoNode01/servers/server1|r
esources.xml#builtin_jdbcprovider)"

"Derby JDBC Provider
(XA)(cells/coutinhoNode01Cell|resources.xml#builtin_jdbcprovider)"

"Derby JDBC
Provider(cells/coutinhoNode01Cell/nodes/coutinhoNode01/servers/serve
r1|resources.xml#JDBCProvider_1183122153343)"
```

---

The command assist feature is helpful when you are learning Jython. You can use it to create scripts for future use easily.

## 25.5.7 Debugging Jython scripts

With the Jython debugger, you can detect and diagnose errors in Jython scripts that are run on WebSphere Application Server. For an example of debugging the sample `listJDBCProviders` script, see 28.5, “Using the Jython debugger” on page 1492.

## 25.6 More information

For more information about application deployment, see the following resources:

- ▶ Understanding WebSphere Extended Deployment  
<http://www.ibm.com/developerworks/autonomic/library/ac-webxd/>
- ▶ “Learn how to publish an enterprise application with WebSphere Application Server and Application Server Toolkit, V6.1” developerWorks article  
<http://www.ibm.com/developerworks/edu/wes-dw-wes-helloas.html>





# Testing using JUnit

The Rational Application Developer test framework is built on the Eclipse Test and Performance Tools Platform (TPTP), which extends the Eclipse Hyades Tool Project. It contains tracing, profiling, and testing tools. *JUnit* is one of the testing tools and can be used for automated component testing. TPTP also includes profiling capabilities for memory, performance, and other execution-time code analysis. We explore profiling in Chapter 27, “Profiling applications” on page 1419.

In this chapter, we introduce application testing concepts and provide an overview of TPTP and JUnit. We also explain the features of Rational Application Developer for testing. In addition, we include working examples to demonstrate how to create and run component tests using JUnit and demonstrate how to test web applications.

The chapter is organized into the following sections:

- ▶ Introduction to application testing
- ▶ JUnit testing without TPTP
- ▶ Preparing the JUnit sample
- ▶ JUnit testing of JPA entities
- ▶ JUnit testing using TPTP
- ▶ Web application testing
- ▶ Cleaning the workspace

The sample code is available in `c:\7835code\junit`.

## 26.1 Introduction to application testing

Although the focus of this chapter is on component testing, we have included an introduction to testing concepts, such as test phases and environments, to put into context where component testing fits within the development cycle. Next we provide an overview of the TPTP and JUnit testing frameworks. The remainder of the chapter provides a working example of using the features of TPTP and JUnit within Rational Application Developer.

### 26.1.1 Test concepts

Within a typical development project, there are various types of testing performed during the phases of the development cycle. Project requirements that are based on size, complexity, risks, and costs determine the levels of testing to be performed. The focus of this chapter is on component testing and unit testing.

### 26.1.2 Test phases

In this section, we outline and categorize the key test phases.

#### **Unit test**

*Unit tests* are informal tests that are generally executed by the developers of the application code. They are often quite low level in nature and test the behavior of individual software components, such as individual Java classes, servlets, or Enterprise JavaBeans (EJB).

Because unit tests are usually written and performed by the application developer, they tend to be white-box in nature, that is, they are written using knowledge about the implementation details and test-specific code paths. This is not to say that all unit tests have to be written this way. One common practice is to write the unit tests for a component based on the component specification, before developing the component itself. Both approaches are valid, and you might want to use both methods when defining your own unit testing policy.

#### **Component test**

*Component tests* are used to verify particular components of the code before they are integrated into the production code base. Component tests can be performed on the development environment. Within the context of Rational Application Developer, a developer configures a test environment and supporting testing tools, such as JUnit. Using the test environment, you can test customized code including JavaBeans, EJB, and JavaServer Pages (JSP) without having to deploy this code to a runtime system.

## Build verification test

For *build verification tests* (BVTs), members of the development team check their source code into the source control tool and mark the components as part of a build level. The build team is responsible for building the application in a controlled environment based on the source code that is available in the source control system repository.

The build team extracts the source code from the source control system, executes scripts to compile the source code, packages the application, and tests the application build. The test that is run on the application of the build that is produced is called a *build verification test*.

The BVT is a predefined and documented test procedure to ensure that the major elements of the application work properly before accepting the build and making it available to the test team for a function verification test (FVT) or system verification test (SVT).

## Function verification test

*Function verification tests* (FVTs) are used to verify the individual functions of an application. For example, you can verify if the interest was calculated properly within a bank application.

**Rational Function Tester:** Within the Rational product family, the IBM Rational Function Tester is an ideal choice for this type of testing.

## System verification test

*System verification tests* are used to test a group of functions. You use a dedicated test environment with the same system and application software as the target production environment.

For the best results from these tests, you have to find the most similar environment, involve as many components as possible, and verify that all functions work properly in an integrated environment.

**Rational Manual Tester:** Within the Rational product family, the IBM Rational Manual Tester is an ideal choice for this type of testing.

## Performance test

*Performance tests* simulate the volume of traffic that you expect to have for the application, ensure that the system will support this volume of traffic, and determine if the system performance is acceptable.

**Rational Performance Tester:** Within the Rational product family, the IBM Rational Performance Tester is an ideal choice for this type of testing.

### Customer acceptance test

*Customer acceptance test* is a level of testing in which all aspects of an application or system are thoroughly and systematically tested to demonstrate that it meets business and non-functional requirements. The scope of a particular acceptance test is defined in the acceptance test plan.

## 26.1.3 Test environments

When sizing a project, it is important to consider the system requirements for the test environments. We describe several common test environments:

- ▶ Component test environment

This environment is often the development system, and it is the focus of this chapter. In larger projects, development teams must have a dedicated test environment to integrate the components of the team members before putting the code into the application build.

- ▶ Build verification test environment

This test environment is used to test the application that is produced from a controlled build. For example, a controlled build might have source control, build scripts, and packaging scripts for the application. The build verification team runs a subset of tests, often known as *regression tests*, to verify the major functionality of the system that is representative of a wider scale of testing.

- ▶ System test environment

This test environment is used for FVT and SVT to verify the functionality of the application and integrate it with other components. Many test environments can exist with teams of people focused on separate aspects of the system.

- ▶ Staging environment

The staging environment is critical for all sizes of organizations. Prior to deploying the application to production, the staging environment is used to simulate the production environment. This environment can be used to perform customer acceptance tests.

- ▶ Production environment

This environment is the live runtime environment that customers use to access the e-commerce website. In specific cases, customer acceptance testing might be performed on the production environment. Ultimately, the

customers need to test the application. You must have a process to track customer problems and to implement fixes to the application within this environment.

#### **26.1.4 Calibration**

By definition, *calibration* is a set of gradations that shows positions or values. When testing, it is important to establish a baseline for performance and functionality for regression testing. For example, when regression testing, you have to provide a set of tests that have been exercised on previous builds of the application. These tests must be used to test the performance and functionality of the new build. These tests are also important when setting entrance and exit criteria.

#### **26.1.5 Test case execution and recording results**

When trying to determine why a piece of functionality of a component within an application has broken, it is useful to know when the test case last executed successfully. Recording the successes and failures of test cases for a designated application build is essential to having an accountable test organization and a quality application.

#### **26.1.6 Benefits of unit and component testing**

It might seem obvious as to why we want to test our code. Unfortunately, many people do not understand the value of testing. We test our code and applications to find defects in the code and to verify that the changes that we have made to existing code do not break that code. In this section, we highlight the key benefits of unit and component testing.

Perhaps it is more useful to look at the question from the opposite perspective, that is, why developers *do not* perform unit tests. In general, the simple answer is because it is too hard or because nobody forces them to perform unit tests. Writing an effective set of unit tests for a component is not a trivial undertaking. Given the pressure to deliver to which many developers find themselves subjected, the temptation to postpone the creation and execution of unit tests in favor of delivering code fixes or new functionality is often overwhelming.

In practice, this approach usually turns out to be a false economy. Developers rarely deliver bug-free code, and the discovery of code defects and the costs associated with fixing them are pushed further out into the development cycle, which is inefficient. The best time to fix a code defect is immediately after the code has been written and the changes are still fresh in the developer's mind.

Furthermore, a defect that is discovered during a formal testing cycle must be documented, prioritized, and tracked. All of these activities incur cost and might mean that a fix is deferred indefinitely, or at least until it becomes critical.

Based on our experience, we think that encouraging and supporting the development and regular execution of unit test cases ultimately leads to significant improvements in productivity and overall code quality. The creation of unit test cases does not have to be a burden. If done properly, developers can find the intellectual challenge quite stimulating and ultimately satisfying. The thought process involved in creating a test can also highlight shortcomings in a design, which might not otherwise have been identified when the major focus is on implementation.

Take the time to define a unit testing strategy for your own development projects. A simple set of guidelines and a framework that makes it easy to develop and execute tests pays for itself surprisingly quickly.

### **26.1.7 Benefits of testing frameworks**

After you have decided to implement a unit testing strategy for your project, the first hurdles to overcome are the factors that dissuade developers from creating and running unit tests in the first place. A testing framework can help by making it easier to perform these tasks:

- ▶ Write tests
- ▶ Run tests
- ▶ Rerun a test after a change

Tests are easier to write, because a lot of the infrastructure code that you require to support every test is already available. A testing framework also provides a facility that makes it easier to run and rerun tests, perhaps by using a graphical user interface (GUI). The more often a developer runs tests, the sooner the problems can be located and fixed, because the difference is smaller between the code that last passed a unit test and the code that fails the test.

Testing frameworks also provide other benefits:

- ▶ **Consistency:** Every developer uses the same framework. All of your unit tests work in the same way, can be managed in the same way, and report results in the same format.
- ▶ **Maintenance:** If a framework has already been developed and is already in use in a number of projects, you spend less time maintaining your testing code.

- ▶ Ramp-up time: If you select a popular testing framework, you might find that new developers coming into your team are already familiar with the tools and concepts that are involved.
- ▶ Automation: A framework can offer the ability to run tests unattended, perhaps as part of a daily or nightly build.

**Daily build:** A common practice in many development environments is the use of daily builds. These automatic builds usually are initiated in the early hours of the morning by a scheduling tool.

## 26.2 JUnit testing without TPTP

In this section, we explain the fundamentals of JUnit. We also present a working example of how to create and run a JUnit test within Rational Application Developer.

You can find the JUnit home page at the following web address:

<http://www.junit.org/>

### 26.2.1 JUnit fundamentals

A *unit test* is a collection of tests designed to verify the behavior of a single unit. A unit is always the smallest testable part of an application. In object-oriented programming, the smallest unit is always a class.

JUnit tests your class by scenario. You have to create a testing scenario that uses the following elements:

- ▶ Instantiates an object
- ▶ Invokes methods
- ▶ Verifies assertions

**Assertion:** An *assertion* is a statement with which you can test the validity of any assumptions that are made in your code.

### 26.2.2 Test and Performance Tools Platform (TPTP)

Test and Performance Tools Platform (TPTP) provides an open platform that supplies powerful frameworks and services that allow software developers to build unique testing and performance tools.

TPTP addresses the entire test and performance life cycle, from early testing, including test editing and execution, to production application profiling and tracing.

Within the scope of Rational Application Developer, TPTP includes the following types of testing:

- ▶ JUnit testing
- ▶ Performance testing of web/HTTP applications

Although each of these areas of testing has its own unique set of tasks and concepts, two sets of topics are common to all test types:

- ▶ Providing tests with variable data
- ▶ Creating a test deployment

You can obtain more information about TPTP at the following web address:

<http://www.eclipse.org/tptp>

### 26.2.3 New in JUnit 4

JUnit 4 has significant changes from previous releases. It simplifies testing by using the annotation feature, which was introduced in Java 5 (Java Development Kit (JDK) 1.5). One helpful feature is that tests no longer rely on subclassing, reflection, and naming conventions. With JUnit 4, you can mark any method in any class as an executable test case merely by adding the `@Test` annotation in front of the method. Table 26-1 lists the important annotations.

Table 26-1 JUnit 4 annotation overview

Annotation name	Description
<code>@Test</code>	Marks that this method is a test method
<code>@Test(expected=ExceptionClassName.class)</code>	Tests if the method throws the named exception
<code>@Test(timeout=100)</code>	Fails if execution of a method takes longer than 100 milliseconds
<code>@Ignore</code>	Ignores the test method
<code>@BeforeClass</code>	Marks the method that must be executed once before the start of all the tests, for example, to connect to the database
<code>@AfterClass</code>	Marks the method that must be executed once after the execution of all the tests, for example, to disconnect from the database



Annotation name	Description
@Before	Marks the method that must be executed before each test (setUp)
@After	Marks the method that must be executed after each test (tearDown)

An Open Source project is available called *JUnit 4 Extensions*. This project provides the `@Prerequisite(requires="methodName")` annotation, which can be helpful. With this annotation, you can call another method before entering the test. The test is only executed if this method returns *true*. Example 26-1 is an example showing the test case class using these annotations.

For more information about the JUnit 4 Extension project, see the following web address:

<http://www.junitext.org/>

## Test case class

Example 26-1 shows a test case class using JUnit 4.

*Example 26-1 Simple JUnit 4.x test case class*

---

```
package itso.rad8.bank.test.junit.example;

import static org.junit.Assert.assertEquals;

// Imports for the annotations (other imports omitted)
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import org.junitext.Prerequisite;

public class ITSOBankTestExample {

 private Bank bank = null;
 private static final String ACCOUNT_NUMBER = "001-999000777";
 private static final String CUSTOMER_SSN = "111-11-1111";

 @BeforeClass
 public static void runBeforeClass() {}

 @AfterClass
```

```

public static void runAfterClass() {}

@Before
public void runBeforeEveryTest() {
 if (this.bank == null) {
 // Instantiate objects
 this.bank = ITSOBank.getBank();
 }
}

@After
public void runAfterEveryTest() {}

@Prerequisite(requires="isBankAvailable")
@Test
public final void testSearchAccountByAccountNumber() {

 try {
 // Invoke a method

 Account bankAccount = this.bank

 .searchAccountByAccountNumber(ITSOBankTestExample.ACCOUNT_NUMBER);

 // Verify an assertion
 assertEquals(bankAccount.getAccountNumber(),
 ITSOBankTestExample.ACCOUNT_NUMBER);
 } catch (InvalidAccountException e) {
 e.printStackTrace();
 }
}

@Test(expected=InvalidAccountException.class)
public final void testSearchAccountByInvalidAccountNumber()
 throws InvalidAccountException {

 Account bankAccount = this.bank
 .searchAccountByAccountNumber("966-111000999");
}

@Test
public final void testSearchCustomerBySsn() {

 // Invoke a method
 try {

```

```

 Customer bankCustomer = this.bank
 .searchCustomerBySsn(ITSOBankTestExample.CUSTOMER_SSN);

 // Verify an assertion
 assertEquals(bankCustomer.getSsn(),
 ITSOBankTestExample.CUSTOMER_SSN);
 } catch (InvalidCustomerException e) {
 e.printStackTrace();
 }
}

public boolean isBankAvailable() {
 if (this.bank != null) {
 return true;
 } else {
 return false;
 }
}
}
}

```

---

In JUnit, each test is implemented as a Java method that must be declared as `public void` and take no parameters. This method is then invoked from a test runner. In previous JUnit releases, all the test method names had to begin with `test...`, so that the test runner found them automatically and ran them. In JUnit 4, this prefix is no longer required, because we mark the test methods with the `@Test` annotation.

**Important:** The package structure `junit.framework` that was used in JUnit 3.8.1 has been changed to `org.junit` in JUnit 4.x.

### New annotations added:

► **JUnit 4.8** @Categories

From a given set of test classes, the @Categories annotation runs only the classes and methods that are annotated with either the category given with the @IncludeCategory annotation, or a subtype of that category. For now, annotating suites with @Category has no effect. Categories must be annotated on the direct method or class.

► **JUnit 4.7** @Rule

@Rule annotates fields that contain rules. The field must be public, not static, and have a subtype of MethodRule.

A MethodRule is an alteration in how a test method is run and reported. Multiple MethodRules can be applied to a test method. The statement that executes the method is passed to each annotated rule in turn, which can return a substitute or modified statement, which is passed to the next rule, if any. For the method rules, go to this website:

<http://www.junit.org/>

## JUnit assert class

JUnit provides a number of static methods in the org.junit.Assert class that can be used to assert conditions and fail a test if the condition is not met.

Table 26-2 summarizes the provided static methods.

Table 26-2 JUnit assert class: Static methods overview

Method name	Description
assertEquals	Asserts that two objects or primitives are equal. Compares objects using the equals method and compares primitives using the == operator.
assertFalse	Asserts that a boolean condition is false.
assertNotNull	Asserts that an object is not null.
assertNotSame	Asserts that two objects do not refer the same object. Compares objects using the != operator.
assertNull	Asserts that an object is null.
assertSame	Asserts that two objects refer to the same object. Compares objects using the == operator.
assertTrue	Asserts that a boolean condition is true.
fail	Fails the test.

All of these methods include an optional `String` parameter so that the writer of a test can provide a brief explanation of why the test failed. This message is reported along with the failure when the test is executed. You can find the full JUnit 4 application programming interface (API) documentation at the following web address:

<http://junit.org/junit/javadoc/4.5/>

## Test suite class

Test cases can be organized into *test suites*. In JUnit 4.x, the way to build test suites has been completely replaced and no longer uses subclassing, reflection, and naming conventions. Example 26-2 shows how to build a test suite class in JUnit 4.

*Example 26-2 Simple JUnit 4 test suite class*

---

```
package itso.rad8.bank.test.junit;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({ITSOBankTest.class})
public class AllTests {}
```

---

The `AllTests` class is a simple placeholder for the `@RunWith` and `@SuiteClasses` annotations and does not require a static suite method. The `@RunWith` annotation signals the JUnit 4 test runner to use the `org.junit.runners.Suite` class for running the `AllTests` class. With the `@SuiteClasses` annotation, you can define which test classes to include in this suite and in which order. If you add more than one test class, you must use the following syntax:

```
@SuiteClasses({TestClass1.class, TestClass2.class})
```

## 26.3 Preparing the JUnit sample

We use the ITSO Bank application that was created in 7.5, “Developing the ITSO Bank application” on page 240, for the JUnit test working example.

**Important:** Use a new workspace for the JUnit example. If you work in your regular workspace, the Java builder will not function properly any longer for projects with non-Java files in `src/META-INF`, such as projects with the Java Persistence API (JPA). See 26.7, “Cleaning the workspace” on page 1418, for a circumvention.

The JUnit sample is based on the RAD8Java project. Import the completed code for this section from the `c:\7835codesolution\junit\RAD8JUnit.zip` project interchange file.

After you import RAD8JUnit, it is a requirement that you import RAD8JPA and RAD8JPATest for RAD8JUnit to work correctly.

The JPA files are available at `C:\7835codesolution\jpa\RAD8JPA.zip`.

### 26.3.1 Creating a JUnit test case

Rational Application Developer provides wizards to help you build JUnit test cases and test suites. The following step-by-step guide leads you through the example, so that you become familiar with the JUnit tooling within Rational Application Developer:

1. Create a new package called `itso.rad8.bank.test.junit` in the RAD8JUnit project (under `src`).
2. Add the JUnit library to the Java project so that the classes from the JUnit framework can be resolved:
  - a. Right-click the **RAD8JUnit** project and select **Properties**, or press **Alt+Enter**.
  - b. Select **Java Build Path**. Select the **Libraries** tab and click **Add Library**.
  - c. In the Add Library window, select **JUnit** and click **Next**.
  - d. In the JUnit library version field, select **JUnit 4** and click **Finish**.
  - e. Click **OK** to close the Properties window.

#### Creating a JUnit test case

To create a test case for the `transfer` method of the `ITSOBank` class, follow these steps:

1. Right-click the `itso.rad8.bank.impl.ITSOBank` class and select **New** → **JUnit Test Case** (which is only available in the Java perspective) or select **New** → **Other** → **Java** → **JUnit** → **JUnit Test Case**. Alternatively, click the arrow in the  icon in the toolbar and select  **JUnit Test Case**.

2. In the JUnit Test Case window (Figure 26-1 on page 1380), complete the following steps:
  - a. Select **New JUnit 4 test**.
  - b. For the Package, click **Browse** and select `itso.rad8.bank.test.junit`.
  - c. For the Name, accept **ITSOBankTest**.
  - d. Select the **setUp()** and **tearDown()** methods.
  - e. Clear **Generate comments** (default).
  - f. Verify that Class under test is set to `ITSOBank`.
  - g. Click **Next**.

**Stub:** A *stub* is a skeleton method so that you can add the body of the method yourself.

Figure 26-1 on page 1380 shows the JUnit Test Case window.

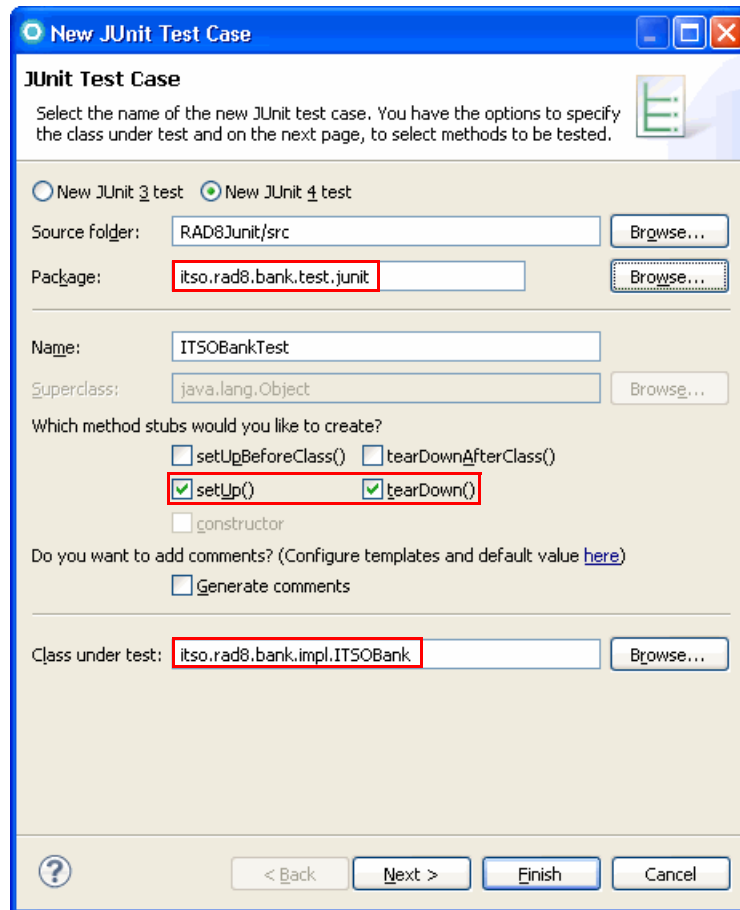


Figure 26-1 JUnit Test Case wizard

3. In the New JUnit Test Case: Test Methods window, select the **transfer** method and click **Create final method stubs** (as shown in Figure 26-2 on page 1381). Then click **Finish**.



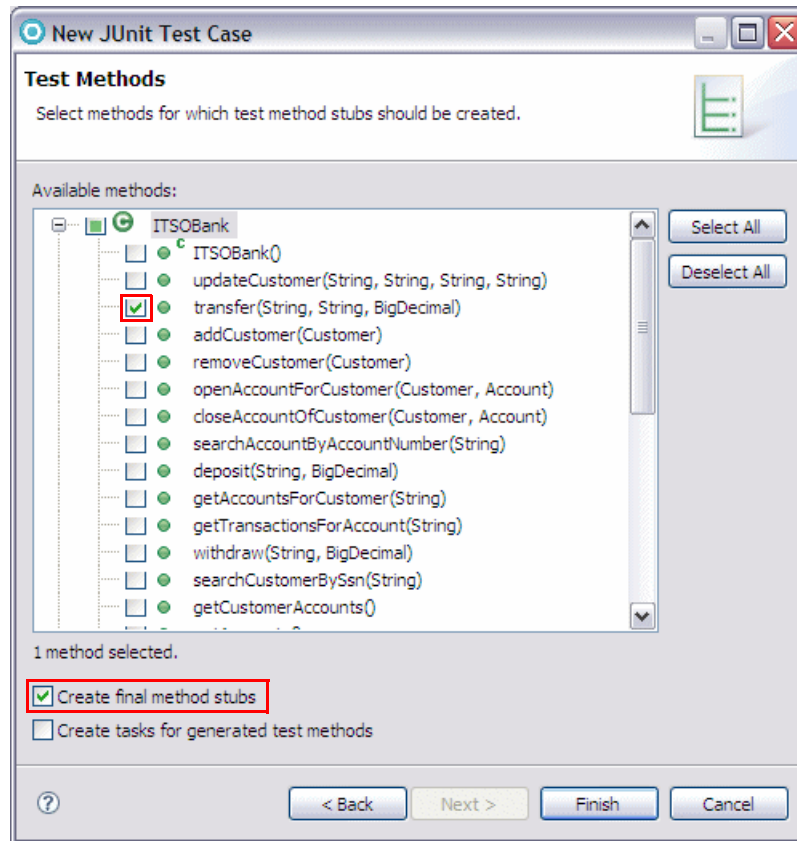


Figure 26-2 Select test methods

The wizard generates the `ITSOBankTest` class and opens the file in the editor.

## Completing the test class

Typically, you run several tests in one test case. To ensure that no side effects occur between test runs, the JUnit framework provides the `setUp` and `tearDown` methods. Every time that the test case is run, `setUp` is called at the start of the run, and `tearDown` is called at the end of the run.

To complete the test class, follow these steps:

1. Add three variables to `ITSOBankTest` class:

```
private Bank bank = null;
private static final String ACCOUNT_NUMBER_1 = "001-999000777";
private static final String ACCOUNT_NUMBER_2 = "002-999000777";
```

Remember that you can add missing imports by selecting **Source** → **Organize Imports**, or by pressing Ctrl+Shift+O.

The bank variable is instantiated in the setUp method before starting each test and is available for use in the test methods.

2. Add the following code to the setUp method:

```
@Before
public void setUp() throws Exception {
 /* Instantiate objects
 * The getBank method returns the initialized (containing
Customers
 * and Accounts) ITSOBank instance.
 */
 if (this.bank == null) {
 this.bank = ITSOBank.getBank();
 }
}
```

3. Keep the generated code of the tearDown method unchanged.

**Flow of the code:** The JUnit framework calls the setUp method before each test method. The ITSOBank class is implemented as a *Singleton* (only one object of this class exists). Therefore, you always get the same ITSOBank object when you call the static getBank method. When the setUp method is called a second time, you get the same ITSOBank instance as in the first call.

For example, if you removed all the customers in the first test method, the ITSOBank object that you get in the second call of the setUp method is empty, because removing a customer from the bank automatically closes all of the customer's accounts. Therefore, it can be useful to call a cleanup service in the tearDown method to reset the ITSOBank instance. This cleanup service is not necessary in our example.

## Completing the test methods

When the ITSOBankTest is generated, the stub of the testTransfer method is added. In this section, we explain how to implement this test method and how to add a second method:

1. Complete the testTransfer method, as shown in Example 26-3 on page 1383.

**Reminder:** Make sure that you import `java.math.BigDecimal` and `org.junit.Assert`.

```
@Test
public final void testTransfer() {
 try {
 BigDecimal account1AmountBeforeTransfer = this.bank
 .searchAccountByAccountNumber(
 ITSOBankTest.ACCOUNT_NUMBER_1).getBalance();
 BigDecimal account2AmountBeforeTransfer = this.bank
 .searchAccountByAccountNumber(
 ITSOBankTest.ACCOUNT_NUMBER_2).getBalance();
 BigDecimal transferAmount = new BigDecimal(20.00D);

 // Invoke a method
 this.bank.transfer(ITSOBankTest.ACCOUNT_NUMBER_1,
 ITSOBankTest.ACCOUNT_NUMBER_2, transferAmount);

 // Verify assertions
 Assert.assertEquals(this.bank.searchAccountByAccountNumber(
 ITSOBankTest.ACCOUNT_NUMBER_1).getBalance().doubleValue(),
 account1AmountBeforeTransfer.subtract(transferAmount)
 .doubleValue(), 0.00D);
 Assert.assertEquals(this.bank.searchAccountByAccountNumber(
 ITSOBankTest.ACCOUNT_NUMBER_2).getBalance().doubleValue(),
 account2AmountBeforeTransfer.add(transferAmount)
 .doubleValue(), 0.00D);
 } catch (ITSOBankException e) {
 e.printStackTrace();
 Assert.fail("Transfer failed: " + e.getMessage());
 }
}
```

---

This test method transfers an amount from one account to another account. After the credit and debit transactions have completed, the method verifies that the balances of the two involved accounts have changed accordingly.

2. If you want to test the method completely, write a test method for each possible outcome of that method. In our example, we add another test method called `testInvalidTransfer`, as shown in Example 26-4 on page 1384. This method also calls the transfer method, but this time, we make a transfer where the debit account does not have enough funds. The method verifies that you receive an `InvalidTransactionException`.

*Example 26-4 ITSOBankTest class: testInvalidTransfer*

---

```
@Test(expected = InvalidTransactionException.class)
public final void testInvalidTransfer() throws Exception {
 BigDecimal transferAmount =
this.bank.searchAccountByAccountNumber(
 ITSOBankTest.ACCOUNT_NUMBER_1).getBalance().multiply(
 new BigDecimal(2.00D));

 // Invoke a method
 this.bank.transfer(ITSOBankTest.ACCOUNT_NUMBER_1,
 ITSOBankTest.ACCOUNT_NUMBER_2, transferAmount);
}
```

---

## 26.3.2 Creating a JUnit test suite

A JUnit test suite is used to run one or more test cases at once. Rational Application Developer has a simple wizard to create a test suite for JUnit 3.8.1 test cases. However, the wizard does not currently support creating JUnit 4 test suites.

**Important:** We found that, with the New JUnit Test Suite wizard, we cannot select between JUnit 3.8.1 and JUnit 4 test suites. We can only use the wizard for JUnit 3.8.1 test suites and therefore can add only JUnit 3.8.1 or earlier JUnit version test cases. This situation is a known Eclipse bug, as explained at the following web address:

[http://bugs.eclipse.org/bugs/show\\_bug.cgi?id=155828](http://bugs.eclipse.org/bugs/show_bug.cgi?id=155828)

We created the JUnit test suite for the example manually. Next we provide a step-by-step guide to create a test suite for JUnit 3.8.1 test cases. This guide is given for completeness and is not intended for use with the example.

To create a JUnit 4 test suite manually, follow these steps:

1. Create a new class called `AllTests` in the same package. The class extends the default superclass `java.lang.Object` and does not require any interfaces or method stubs.
2. Add the import statements and annotations to the `AllTests` class, as shown in Example 26-5. The structure of that class is described in “Test suite class” on page 1377.

*Example 26-5 AllTests class: A JUnit 4.x test suite class*

---

```
package itso.rad8.bank.test.junit;
```

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({ITSOBankTest.class})
public class AllTests {}
```

---


In our example, we only added a single test class. Therefore, a test suite is not required. However, as you add more and more test cases, a test suite quickly becomes a more practical way to manage your unit testing.

To create a JUnit 3.8.1 test suite using the New JUnit Test Suite wizard, follow these steps:

1. Right-click the **JUnit** package and select **New** → **Other** → **Java** → **JUnit** → **JUnit Test Suite**.
2. In the JUnit Test Suite window, complete the following steps:
  - a. For the Name, type `AllTests`.
  - b. For the Test classes to include in the suite, select all of the test classes that you want to include in this suite.
  - c. Click **Finish**.

### 26.3.3 Running the JUnit test case or JUnit test suite

To run a JUnit test case or JUnit test suite, follow these steps:

1. Select the **ITSOBankTest** or **AllTests** class, click the  arrow in the toolbar, and select **Run As** → **JUnit Test**. Alternatively, right-click the class and select **Run As** → **JUnit Test**.

In our example, Rational Application Developer runs the two test methods that are defined in the `ITSOBankTest` class.

2. When the JUnit view opens, move the JUnit view on top of the Console view.

Notice that the two test methods passed the assert verification (Figure 26-3 on page 1386).

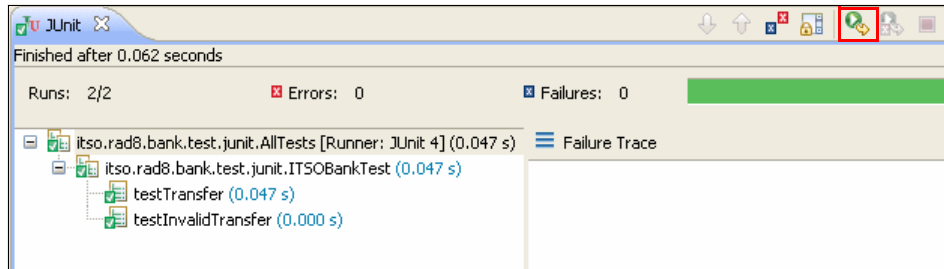



Figure 26-3 JUnit view: Both test methods passed the assert verifications

**Tip:** To run the same test again, click the  button in the JUnit view toolbar, as annotated in Figure 26-3 on page 1386.

## Modifying and running the JUnit test case with assert failures

In our example, we test only for successful execution (although one method throws an exception, which was what we expected). A test is considered *successful* if the test method returns normally. A test *fails* if one of the methods does not pass all assert verifications. An *error* indicates that an unexpected exception is raised by any test, `setUp`, or `tearDown` method. The JUnit view is more interesting when an error or failure occurs.

To modify and run the JUnit test case with assert failures, follow these steps:

1. Modify the process method of the `itso.rad8.bank.model.Credit` class:

```
public BigDecimal process(BigDecimal accountBalance)
 throws InvalidTransactionException {
 if ((this.getAmount() != null)
 && (this.getAmount().compareTo(new BigDecimal(0.00D)) > 0))
 {
 return accountBalance.subtract(this.getAmount());
 } else {
 // rest unchanged
 }
}
```

We change `add` to `subtract`, which is obviously an error in the business logic.

2. Run the `ITS0BankTest` class again as a JUnit Test.

This time, a failure and its trace information are displayed for the `testTransfer` method in the JUnit view (Figure 26-4 on page 1387).

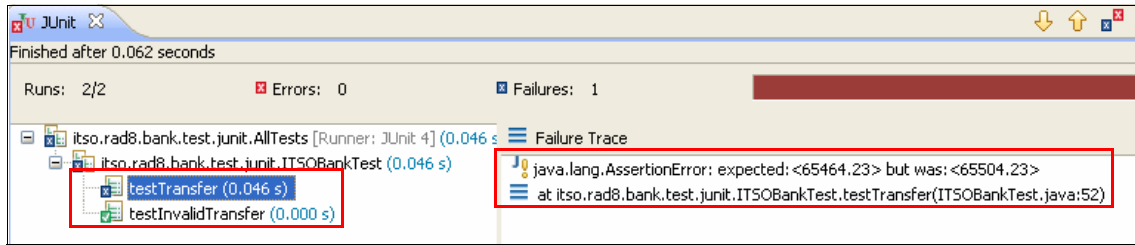


Figure 26-4 JUnit view with failure

Double-clicking the entry in the Failure Trace list takes you to the specified line in the specified Java source file. This line is the line where you set a breakpoint and start debugging the application. For details about how to debug an application, see Chapter 28, “Debugging local and remote applications” on page 1461.

3. Correct the process method of the Credit class, by undoing the change that we made before.

### 26.3.4 Launching individual test methods

In Rational Application Developer, you can select a test method to be launched using the JUnit launch configuration window. Using the previous example of the ITSOBankTest JUnit test, we show how to launch a single test method:

1. Select **RAD8JUnit** → **src** → **itso.rad8.bank.test.junit**.
2. Select the ITSOBankTest Java file. Right-click and select **Run As** → **Run Configurations**. The Run Configurations window opens, as shown in Figure 26-5 on page 1388. Follow these steps:
  - d. Select **JUnit** → **ITSOBankTest** on the left side.
 

If you have not run the ITSOBankTest previously, you can create a Run Configuration by clicking the **New launch configuration** icon, as annotated in Figure 26-5 on page 1388.
  - e. On the right side, select the **Test** tab and select **Run a single test**.
  - f. Click **Browse** and select the **testTransfer** method from the **ITSOBankTest** class.

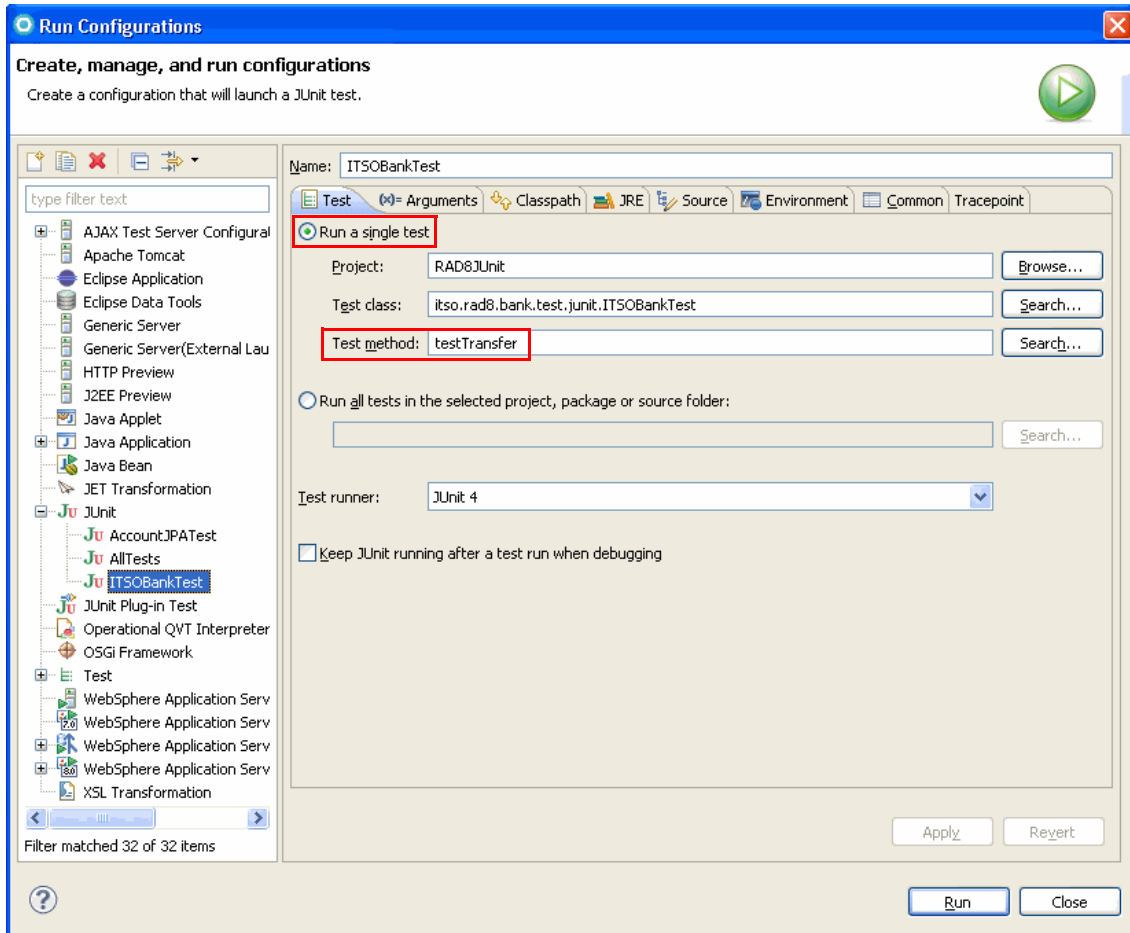


Figure 26-5 Single test method launch configuration

g. Select **Apply** and click **Run**. Figure 26-6 on page 1389 shows the results.

Figure 26-6 on page 1389 shows the selected single test method run. Compare this single test run result to the test run results of the complete file that is shown in Figure 26-3 on page 1386.



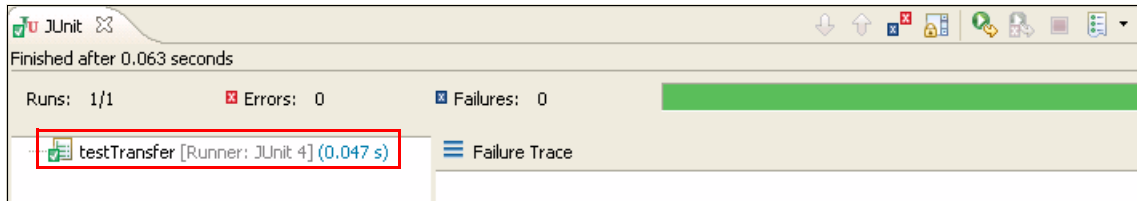


Figure 26-6 JUnit view of single test method results

### 26.3.5 Using the JUnit view

Using the JUnit view, you can view the test run results in a JUnit XML file, as shown in Figure 26-7. For example, you can open a JUnit XML file that is in the project `itso.rad8.bank.test.junit.ITSOBankTest_20101105-180411.xml`, as shown in Figure 26-7.

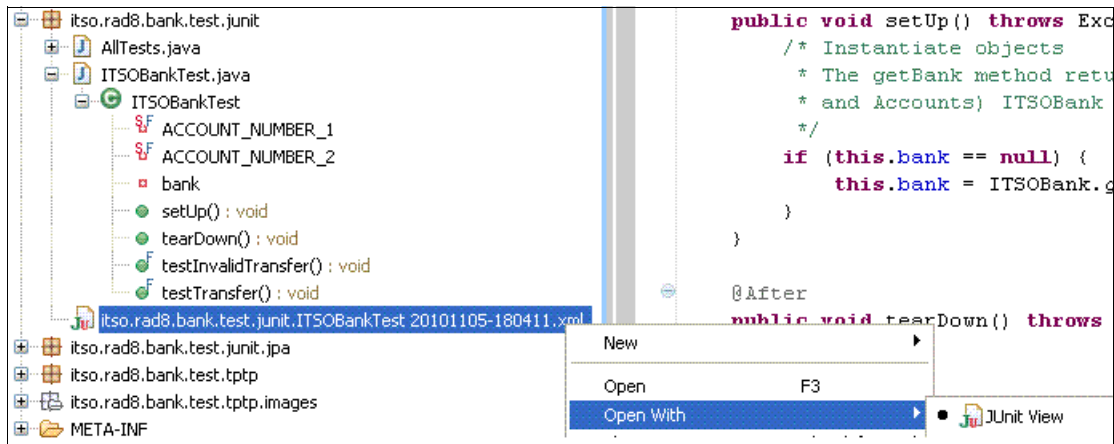


Figure 26-7 JUnit view of a JUnit XML file

You can also import a sample JUnit XML file, such as `TEST-com.ibm.events.cluster.DeployCEIForClusterTests.xml`, and view it in the JUnit view, as shown in Figure 26-8 on page 1390.

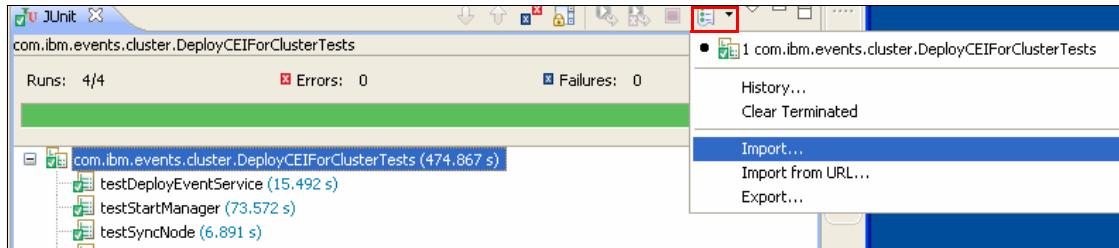


Figure 26-8 JUnit XML import to view in the JUnit view

## 26.4 JUnit testing of JPA entities

According to the JPA specification, JPA entities are not bound to any Java Platform, Enterprise Edition (Java EE) container. JPA entities can run in a Java Platform, Standard Edition (Java SE) environment. Therefore, unit testing of JPA entities is easier to test than it was to test EJB 2.x entity beans.

### 26.4.1 Preparing the JPA unit testing sample

We use the JPA project that was created in 10.2.2, “Create a new JPA project” on page 470, for the JPA unit test working example.

The JPA unit test sample is based on the RAD8JPA project. If you do not have that project in the workspace, import the RAD8JPA project from the `c:\7835codesolution\jpa\RAD8JPA.zip` file.

### 26.4.2 Setting up the ITSOBANK database

The JPA entities are based on the ITSOBANK database. Therefore, we have to define a database connection within Rational Application Developer.

See “Setting up the ITSOBANK database” on page 1880 for instructions to create the ITSOBANK database. For the JPA entities, we can either use the DB2 or Derby database. For simplicity, we use the built-in Derby database in this chapter.

### 26.4.3 Configuring the RAD8JUnit project

The RAD8JUnit project must be configured, so that it can be used to run JPA entity unit tests.

Follow these steps to configure the RAD8JUnit project properly:

1. In the Package Explorer, right-click the **RAD8JUnit** project and select **Properties**.
2. Select **Java Build Path** in the tree and select the **Libraries** tab. Complete these steps:
  - a. Click **Add Variable**, select **ECLIPSE\_HOME**, and click **Extend**. Select **runtimes/base\_v7/derby/lib/derby.jar** and click **OK**.
  - b. Click **Add Variable**, select **ECLIPSE\_HOME**, and click **Extend**. Select **runtimes/base\_v7/runtimes/com.ibm.ws.jpa.thinclient\_7.0.0.jar** and click **OK**.
3. Select the **Projects** tab. Click **Add**, select the **RAD8JPA** project, and click **OK**.
4. Select the **Source** tab. In the Default Output Folder field, type `RAD8JUnit/src` (overwriting `RAD8JUnit/bin`).



**Required change:** This change is necessary so that the `persistence.xml` file, which we create later in the `RAD8JUnit/src/META-INF` folder, is found while executing the test case.

5. Click **OK**.
6. In the Setting Build Path window, click **Yes** to delete the `RAD8JUnit\bin` folder.

The RAD8JUnit project is now properly configured for JPA unit testing.

## 26.4.4 Creating a JUnit test case for a JPA entity

To create a JUnit test case for a JPA entity, follow these steps:

1. Create a new package called `itso.rad8.bank.test.junit.jpa` under **RAD8JUnit/src**.
2. Right-click the **itso.rad8.bank.test.junit.jpa** package and select **New** → **JUnit Test Case** (which is only available in the Java perspective) or select **New** → **Other** → **Java** → **JUnit** → **JUnit Test Case**. Alternatively, click the arrow in the  icon in the toolbar and select  **JUnit Test Case**.
3. In the New JUnit Test Case window, select **New JUnit 4 test**. For the Name, type `AccountJPATest`. Select the **setUp** and **tearDown** methods and click **Finish** (Figure 26-9 on page 1392 shows the dialog window for the JPA entity).

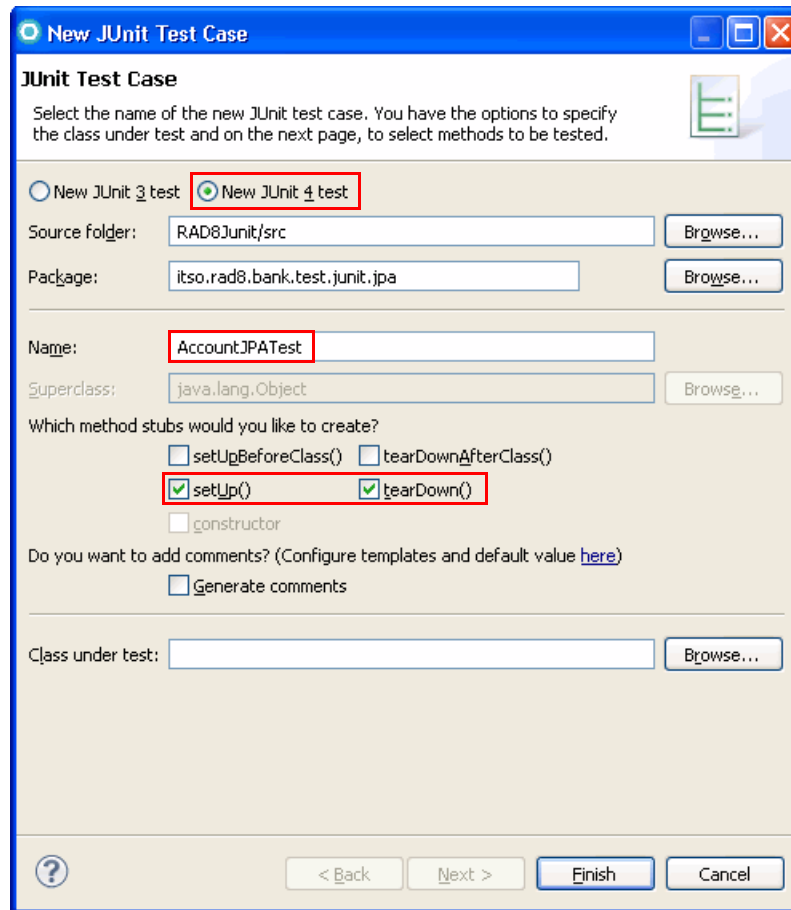


Figure 26-9 JUnit Test Case Wizard for JPA entity

4. Implement the AccountJPATest class, as shown in Example 26-6.

*Example 26-6 JUnit test case for JPA*

---

```

package itso.rad8.bank.test.junit.jpa;

import

public class AccountJPATest {
 EntityManager em;

 @Before
 public void setUp() throws Exception {
 if (em == null) {

```

```

 em = Persistence
 .createEntityManagerFactory("RAD8JPA")
 .createEntityManager();
 }
}

@After
public void tearDown() throws Exception {
 if (em != null) {
 em.close();
 }
}

@Test
public void testLoadAccount() {
 try {
 Account ac = em.find(Account.class, "001-111001");
 assertNotNull(ac);
 } catch (Exception e) {
 fail("Error: Account not found!");
 e.printStackTrace();
 }
}
}

```

---

## 26.4.5 Setting up the persistence.xml file

You must modify the `persistence.xml` file, because the JUnit test runs in Java SE, not on the server. Instead of connecting to the database through a data source, connect to the database directly through a Java Database Connectivity (JDBC) driver. You can modify the `persistence.xml` file in the RAD8JPA project, but it is better to leave that file configured for the data source in the server and to place a new file into the RAD8JUnit project, overwriting the file in the RAD8JPA project. Follow these steps:

1. In the Package Explorer, right-click the **RAD8JUnit** → **src** folder and select **New** → **Folder**. Type `META-INF` as the folder name and click **Finish**.
2. Copy the file `RAD8JPA/src/META-INF/persistence.xml` to `RAD8JUnit/src/META-INF`.
3. Open the **persistence.xml** file (in `RAD8JUnit/src/META-INF`) and change it, as shown in Example 26-7 on page 1394.

Example 26-7 JPA persistence.xml file configured for Derby using OpenJPA

---

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence
 <persistence-unit name="RAD8JPA"
transaction-type="RESOURCE_LOCAL">
 <jta-data-source>jdbc/itsobank</jta-data-source>
 <provider>
 org.apache.openjpa.persistence.PersistenceProviderImpl
 </provider>
 <class>itso.bank.entities.Account</class>
 <class>itso.bank.entities.Customer</class>
 <class>itso.bank.entities.Transaction</class>
 <class>itso.bank.entities.Debit</class>
 <class>itso.bank.entities.Credit</class>
 <properties>
 <property name="openjpa.ConnectionURL"
value="jdbc:derby:C:\7835code\database\derby\ITSOBANK" />
 <property name="openjpa.ConnectionDriverName"
value="org.apache.derby.jdbc.EmbeddedDriver" />
 <property name="openjpa.Log" value="none" />
 </properties>
 </persistence-unit>
</persistence>
```

---

To use the ITSOBANK database in DB2, you use the following properties:

openjpa.ConnectionURL:	jdbc:db2://localhost:50000/ITSOBANK
openjpa.ConnectionDriverName:	com.ibm.db2.jcc.DB2Driver
openjpa.ConnectionUserName:	db2admin (or similar)
openjpa.ConnectionPassword:	<xxxxxxx>

To see the SQL statements that are issued, set the `openjpa.Log` file to the value `SQL=TRACE`.

## 26.4.6 Running the JPA unit test

Now you can run the JPA JUnit test. To run the `AccountJPATest`, follow these steps:

1. Make sure that the `ITSOBANKderby` connection is disconnected. (With Embedded Derby, you can only have one active connection to a database.) You can verify that this connection is disconnected in the Data perspective: Data Source Explorer. If the `ITSOBANKderby` connection is available and active, right-click the connection and select **Disconnect**.

2. In the Package Explorer, right-click the **AccountJPATest** class and select **Run As** → **Run Configurations**. Complete these steps:
  - a. Double-click **JUnit** in the tree. A new JUnit run configuration for AccountJPATest class is created.
  - b. Select the **Arguments** tab (Figure 26-10), and in the VM arguments field, type the value:  
`-javaagent:E:/Progra~1/IBM/SDP/runtimes/base_v7/plugins/com.ibm.ws.jpa.jar`. Make sure to use the directory where Rational Application Developer is installed. Without this agent, the JPA entities are not found.

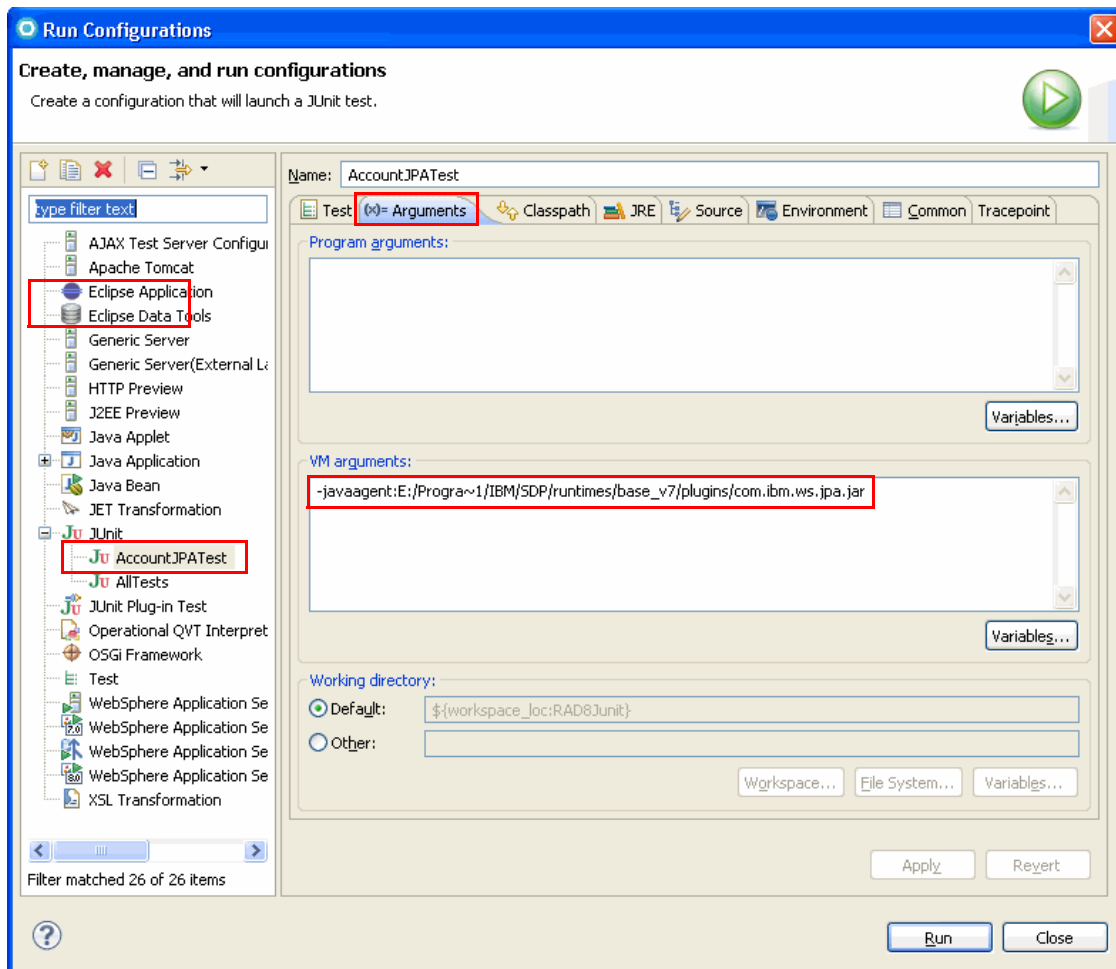


Figure 26-10 Run configuration arguments

- c. Click **Apply** and then click **Run**.

The JUnit view opens, showing that the AccountJPATest test case was successful (Figure 26-11).

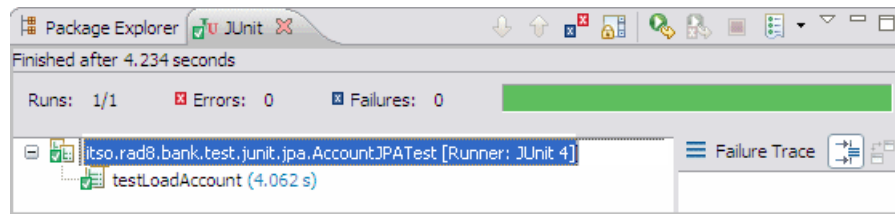


Figure 26-11 JUnit JPA test case was successful

## 26.5 JUnit testing using TPTP

TPTP JUnit test generates an execution history from which a report can be generated. In this section, we discuss the following topics:

- ▶ Creating the TPTP JUnit sample
- ▶ Running the TPTP JUnit test
- ▶ Analyzing the test results

**Note:** TPTP JUnit tests are based on JUnit Version 3.8.1.

### Creating the TPTP JUnit sample

In this section, we continue with the RAD8JUnit project.

**The itso.rad8.bank.test.tptp package:** If you imported the final RAD8JUnit project from the sample code and you want to create this example on your own, delete the package `itso.rad8.bank.test.tptp`.

### Creating a new package

Add a new package called `itso.rad8.bank.test.tptp` to the RAD8JUnit project.

### Creating a TPTP JUnit test manually

To create a TPTP JUnit test manually, follow these steps:

1. Right-click the `itso.rad8.bank.test.tptp` package and select **New** → **Other** → **Test** → **TPTP JUnit Test**.  
Select **Show All Wizards** (at the bottom) to see the Test category.
2. If the Confirm Enablement window opens, click **OK** to enable the Core Testing Support capability of Rational Application Developer.



3. In the New TPTP JUnit Test window (Figure 26-12), complete the following steps:
  - a. For the Source folder, click **Browse** and select **RAD8JUnit/src**. Click **Yes** when prompted to allow Rational Application Developer to add all of the required libraries to the class path.
  - b. For the Package, enter `itso.rad8.bank.test.tptp`.
  - c. For the Name, enter `ITSOBankNew`.
  - d. For “Select how the test behavior is edited”, select **In the test editor** (default setting) and click **Next**.

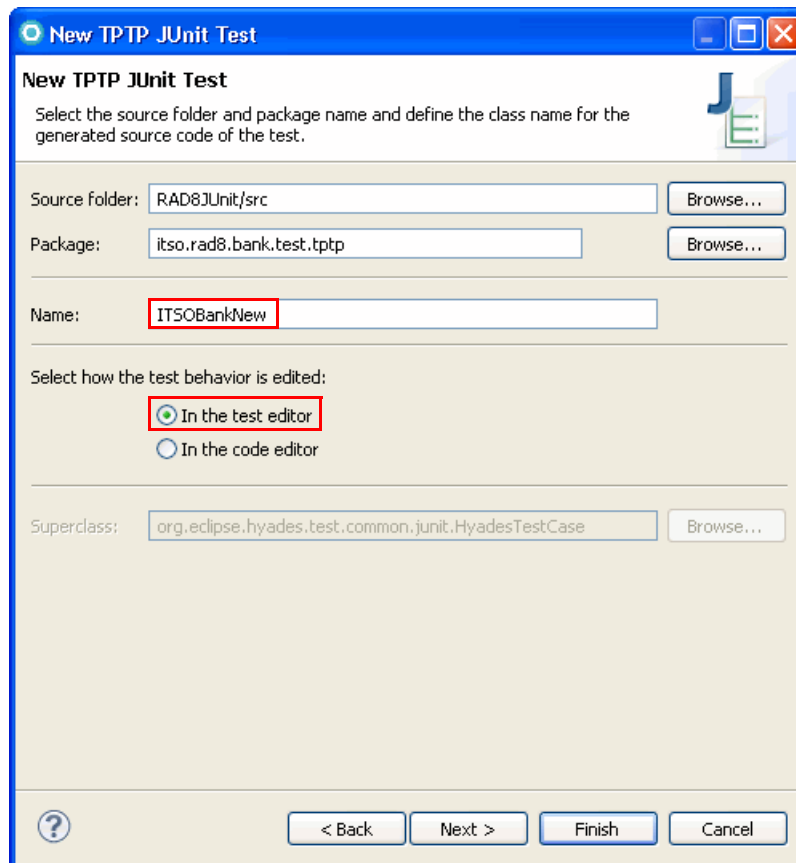


Figure 26-12 New JUnit Test source code window

4. In the JUnit Test Definition window, accept the parent folder (`RAD8JUnit/src/itso/rad8/bank/test/tptp`) and name (`ITSOBankNew`) and click **Finish**.

**Name and location of source code and TPTP Test:** In the previous step, we entered the name and location of the source code. In this step, we enter the name and location of the TPTP Test (the model). By default, they are identical.

The JUnit Test Suite editor opens. You can create and remove methods on a JUnit test, and you can control how those methods are invoked. Three tabs are visible: Overview, Test Methods, and Behavior.

5. In the **Test Methods** tab, click **Add**. In the Name field, enter `testSearchCustomerBySsn`.
6. In the **Behavior** tab, click **Add** and select **Loop**. Select **Loop 1**. Click **Add** again, select **invocation**, and select the **testSearchCustomerBySsn** method. Then click **OK**.

Figure 26-13 on page 1399 shows the Behavior tab after making these changes.

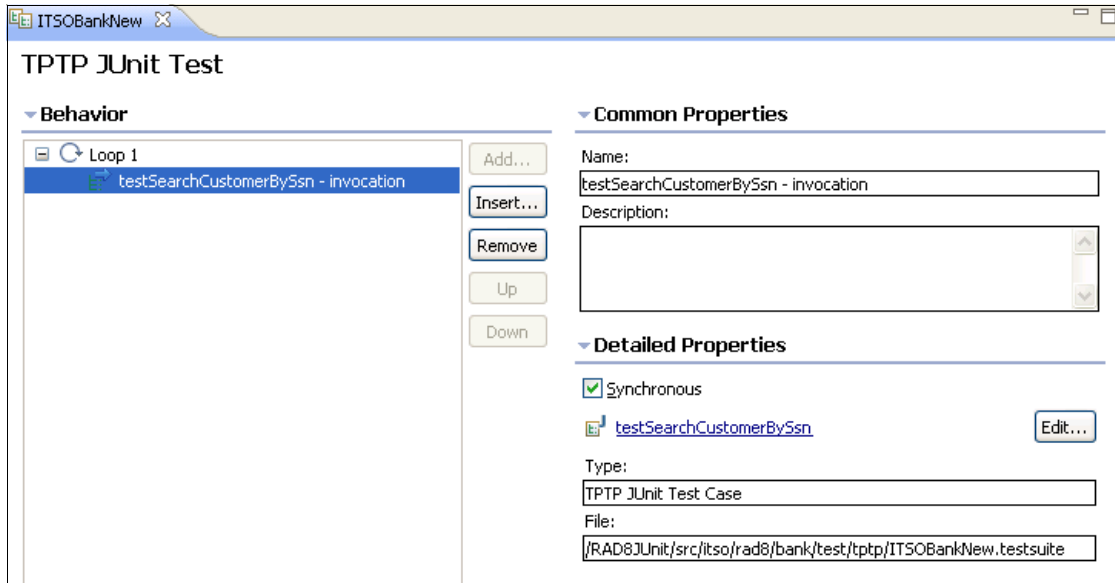


Figure 26-13 TPTP JUnit Test editor (Behavior tab)

**Overview tab:** In the Overview tab, note the following conditions:

- ▶ If “Implement Test Behavior as code” is *selected*, the behavior is purely code-based, that is, the test methods are run exactly as shown in the Test Methods view.
- ▶ If “Implement Test Behavior as code” is *cleared*, the Behavior tab becomes available. Use the behavior feature only for TPTP JUnit tests that have been created manually.

7. Save the TPTP JUnit Test editor.
8. In the JUnit Code Update Preview Options window, select **Never** and click **Always skip to the preview page**. Then click **Finish**.
9. Open the generated **itso.rad8.bank.test.tptp.ITSOBankNew** class in the Java editor and add the highlighted code, as shown in Example 26-8.

*Example 26-8 ITSOBankTest class*

```
package itso.rad8.bank.test.tptp;

import itso.rad8.bank.exception.InvalidCustomerException;
import itso.rad8.bank.ifc.Bank;
import itso.rad8.bank.impl.ITSOBank;
```

```

import itso.rad8.bank.model.Customer;

// Keep other imports unchanged
// All documentation is omitted

public class ITSOBankTest extends HyadesTestCase {

 private Bank bank = null;
 private static final String CUSTOMER_SSN = "111-11-1111";

 public ITSOBankTest(String name) {
 super(name);
 }

 public static Test suite() {
 // Keep method body unchanged
 }

 protected void setUp() throws Exception {
 if (this.bank == null) {
 this.bank = ITSOBank.getBank();
 }
 }

 protected void tearDown() throws Exception {
 }

 public void testSearchCustomerBySsn() throws Exception {
 try {
 Customer bankCustomer = this.bank
 .searchCustomerBySsn(ITSOBankTest.CUSTOMER_SSN);
 assertEquals(bankCustomer.getSsn(),
 ITSOBankTest.CUSTOMER_SSN);
 } catch (InvalidCustomerException e) {
 e.printStackTrace();
 }
 }
}

```

---

## Importing an existing JUnit test case

To create a TPTP JUnit test by importing an existing JUnit test case, follow these steps:

1. Import the `ITSOBank381Test.java` class into the `itso.rad8.bank.test.tptp` package from `C:\7835code\junit\RAD8JUnit`.
2. Right-click the Package Explorer, select **Import** → **Test** → **JUnit tests to TPTP**, and click **Next**.
3. In the Import JUnit tests to TPTP window, select **RAD8JUnit** → **itso.rad8.bank.test.tptp** → **ITSOBank381Test.java** and click **Finish**.

**Import JUnit tests to TPTP window:** In the Import JUnit tests to TPTP window, you cannot select test cases that are JUnit 4.x-based, because TPTP cannot handle JUnit 4.x.

The `ITSOBank381Test.testsuite` is created in the same package.


4. Open the **ITSOBank381Test.testsuite** in the TPTP JUnit Test editor. In the **Test Methods** tab, verify that two test methods are invoked.
5. Save and close the editor.

### 26.5.1 Running the TPTP JUnit test

To run a TPTP JUnit test, follow these steps:

1. Right-click **ITSOBankNew.testsuite** or **ITSOBank381Test.testsuite** and click **Run As** → **Test** to create the run configuration automatically and run the test.
2. Verify the run configurations by selecting **Run** → **Run Configurations**. The configurations are displayed under the *Test* category.

## 26.5.2 Analyzing the test results

When the test run is finished, the execution results (as noted by the  icon) are generated in the RAD8JUnit project (Figure 26-14).

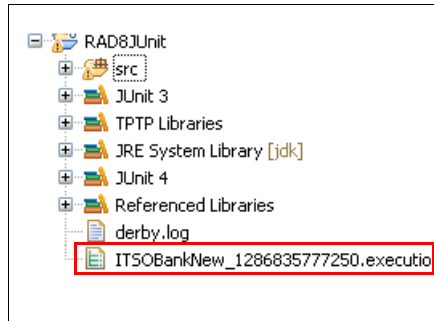


Figure 26-14 Package Explorer view containing test execution results

To analyze the test results, double-click the **ITSOBankNew.execution** result (Figure 26-15 on page 1403).

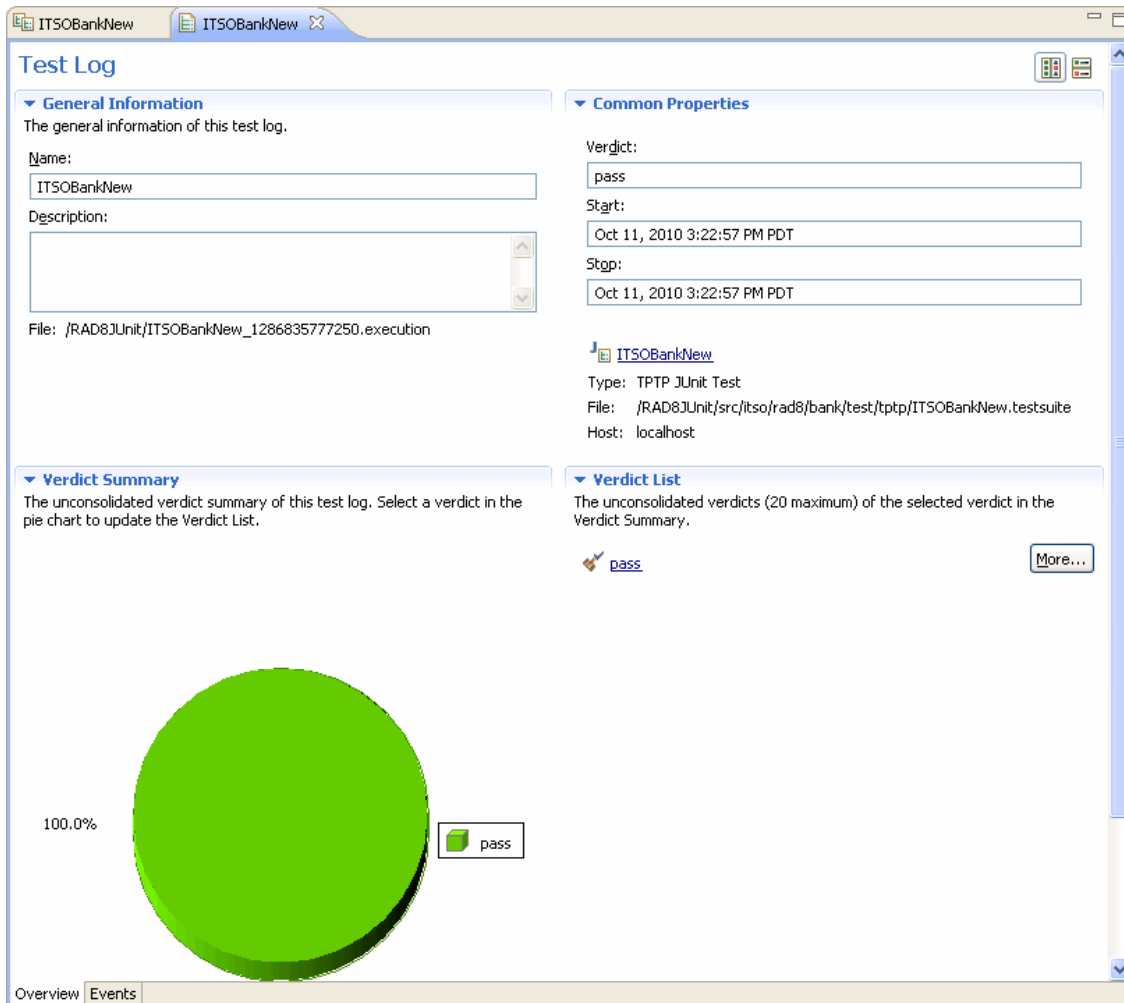


Figure 26-15 TPTP JUnit test execution result

In the Test Log Overview tab, you can see the test verdict (pass) and the start time and stop time of the test run.

The Test Log Events tab lists each step of the test run with additional information.

**Viewing the graphic:** To view the graphic, you must install the Scalable Vector Graphics (SVG) browser plug-in. You can download this viewer at no cost from the Adobe® website:

<http://www.adobe.com/svg/viewer/install/auto/>

## Generating test reports

Based on a test execution results file, you can generate analysis reports:

- ▶ The *Test Pass report* provides a summary of the latest test execution result with a graphical representation of the test success.
- ▶ The *Time Frame Historic report* provides a summary of all test execution results within a certain time frame with a graphical representation of the test success.

To generate a Test Pass report, follow these steps:

1. Open the Test perspective.
  2. Expand **RAD8JUnit** → **src** → **itso** → **rad8** → **bank** → **test** → **tptp**.
  3. In the Test Navigator, right-click **ITSOBankTest** and select **Report**.
  4. In the New Report window, select **Test Pass Report** and click **Next**.
  5. In the Test Pass Report window, for the Name, enter **ITSOBankNew\_TestPass** (the folder is preselected) and click **Next**.
  6. In the Select a time frame window, enter the start date, end date, start time, and end time and click **Finish**:
    - The end time is set as the current time.
    - For the start time, enter a time before you started testing (by default, this field is set to the beginning of the day).
- A Test Pass report is generated.
7. Right-click **ITSOBankNew\_TestPass** and select **Open With** → **Web Browser**.



The Test Pass report opens in the web browser (Figure 26-16).

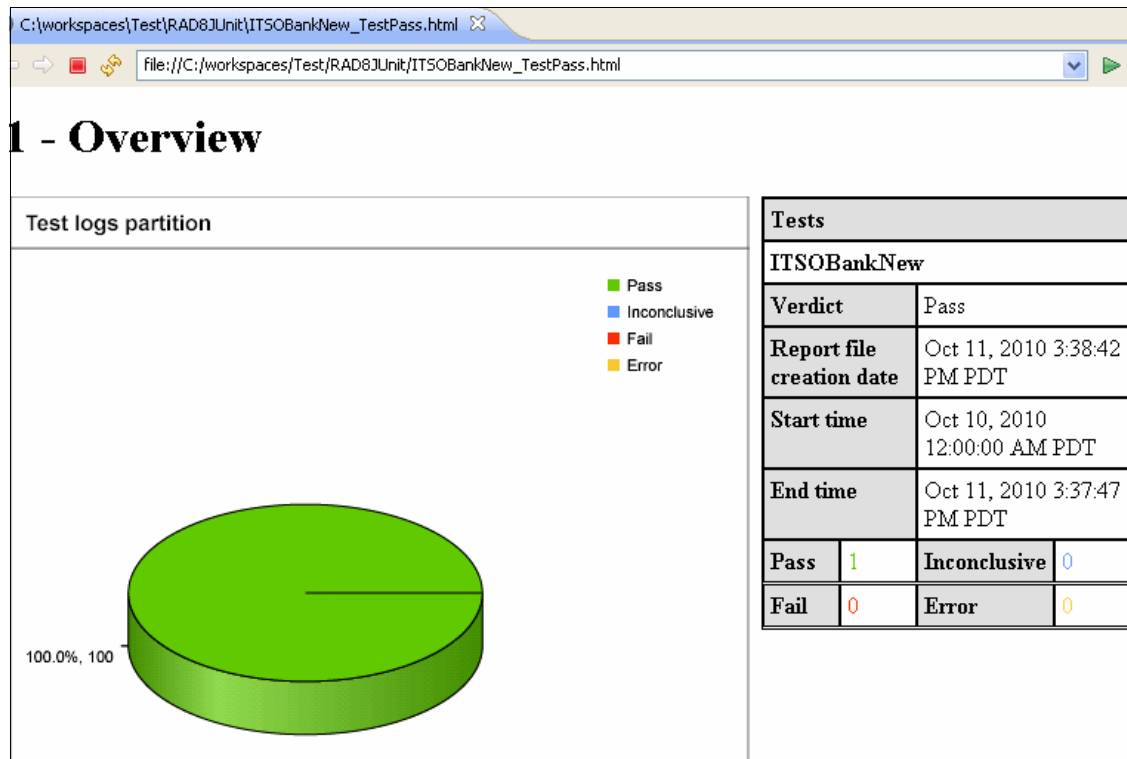


Figure 26-16 Test Pass report

## 26.6 Web application testing

You can also create test cases that run against one of the web projects: RAD8BankBasicWeb, RAD8StrutsWeb, or RAD8EJBWeb. However, when testing anything that runs inside a servlet container, a testing framework, such as *Cactus*, can make the testing much easier.

**Cactus:** Cactus is an open source subproject in the Apache Software Foundation's Jakarta Project. It is a simple framework for unit testing server-side Java code, such as servlets, EJB, tag libraries, and filters.

The objective of Cactus is to lower the cost of writing tests for server-side code. Cactus supports the so-called white box testing of server-side code. It extends and uses JUnit.

You can find more information at the following web address:

<http://jakarta.apache.org/cactus/>

In addition to providing a common framework for test tools and support for JUnit test generation, TPTP enables you to test web applications.

TPTP provides the following web testing tasks:

- ▶ Recording a test: The test creation wizard starts the Hyades proxy recorder, which records your interactions with a browser-based application. When you stop recording, the wizard starts a test generator, which creates a test from the recorded session.
- ▶ Editing a test: You can inspect and modify a test prior to compiling and running it.
- ▶ Generating an executable test: Follow this procedure to generate an executable test. Before a test can be run, the Java source code of the test must be generated and compiled. This process is called *code generation*.
- ▶ Running a test: Run the generated test.
- ▶ Analyzing the test results: At the conclusion of a test run, you see an execution history, including a test verdict, and you can request two graphical reports showing a page response time and a page hit analysis.

## 26.6.1 Preparing for the sample

As a prerequisite to the web application testing sample, you must have the WebSphere Application Server v8.0 Beta test environment installed and running. We use the RAD8BankBas i cWeb application that was created in Chapter 18, “Developing web applications using JavaServer Pages (JSP) and servlets” on page 981, for the web application testing sample.

If you do not have the RAD8BankBas i cWeb application in the workspace, import the c:\7835code\junit\RAD8BankBas i cWeb project into the workspace (select all three projects).

You can also import the completed code for this section from the C:\7835code\junit\RAD8JUnitWebTest.zip project interchange file.

To verify that the web application runs, follow these steps:

1. Switch to the **Web** perspective.
2. Right-click the **RAD8BankBasicWeb** project in the Enterprise Explorer view and select **Run As** → **Run on Server**.
3. Verify that the web browser starts and that the ITSO RedBank welcome page is shown. Close the page.

## Creating a Java project

Create a new Java project called RAD8JUnitWebTest.

### 26.6.2 Recording a test

To create a simple HTTP test, follow these steps:

**Browser cache:** To ensure that your recording accurately captures HTTP traffic, clear the browser cache.

1. Open the **Test** perspective.
2. Right-click **RAD8JUnitWebTest** in the Test Navigator view, select **New** → **Test Element** → **Test From Recording**, and click **Next**.
3. Select **Create Test From New Recording** and click **Next**.
4. In the Select Location for Test Suite window, select the **RAD8JUnitWebTest** project, accept the test file name of **RAD8JUnitWebTest.testsuite**, and click **Finish**.

A progress window opens while your browser starts. Your browser settings are updated, and a local proxy is enabled. If you are using a browser other than Microsoft Internet Explorer, see the online help for detailed instructions to configure the proxy.

**Tip:** You can set the browser that is used for recording by selecting **Window** → **Preferences** → **Test** → **TPTP URL** → **URL Recorder**.

Recording has now started.

5. Start the selected web application by entering the following URL in the browser:

`http://localhost:9080/RAD8BankBasicWeb/`

The port (9080) might differ in your installation.

**Important:** For Internet Explorer 7.0, you must use the IP address to run the web application:

`http://<ip-address>:9080/RAD8BankBasicWeb/`

No recording is produced when using the following URL:

`http://localhost:9080/...`

For Firefox 2.0, you must first configure the network settings. Select **Tools** → **Options** → **Advanced** → **Network**. Click **Settings** and select **Auto-detect proxy settings for the network**.

6. Record a money transfer from a customer's account to another account, and verify that the required transactions have been created:
  - a. On the ITSO RedBank welcome page, select the **redbank** link. For the customer ID (SSN), type 333-33-3333. Then click **Submit**.
  - b. Click account number **003-999000777**.
  - c. On the next page, select **Transfer**. In the Amount field, enter 500. In the To Account field, enter 003-999000888. Then click **Submit**.
  - d. Verify that **List transactions** is selected and click **Submit**. One Debit transaction is listed.
  - e. Click **Account Details** and then click **Customer Details**.
  - f. Click account number **003-999000888**, verify that **List transactions** is selected, and click **Submit**. One Credit transaction is listed.
  - g. Click **Account Details**, click **Customer Details**, and click **Logout**.
7. Close the browser to stop recording or click **Stop Recording** in the toolbar of the Recorder Control view (Figure 26-17 on page 1409).

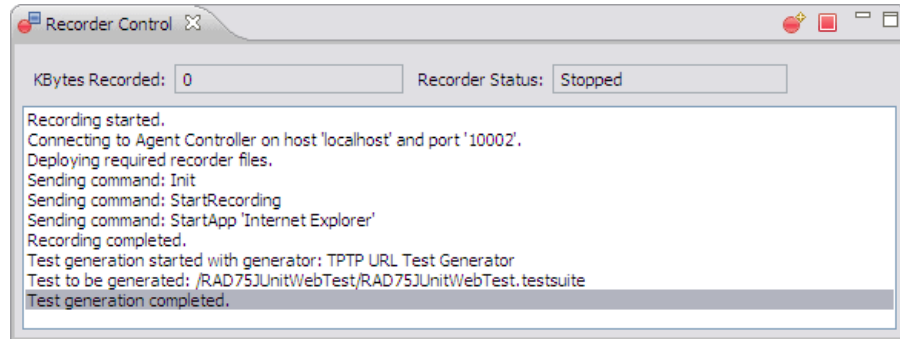


Figure 26-17 Recorder Control view

8. When prompted to Confirm Open Editor, click **Yes**.

After closing the browser, the Recorder Control view shows the messages Recording completed and Test generation completed.

The wizard automatically builds a TPTP URL Test. When the test is successfully generated, the Recorder Control view shows the Test generation completed message.

### 26.6.3 Editing the test

The TPTP URL Test is displayed under the RAD8JUnitWebTest project and is open in the editor. We can inspect and modify it before compiling and running it. The test is not Java code yet, but we can check the requests and modify them:

1. On the Overview tab (Figure 26-18 on page 1410), enter the following data:
  - For the Source Folder, enter /RAD8JUnitWebTest/src.
  - For the Package Name, enter itso.rad8.bank.test.
  - For the Class Name, enter RAD8JUnitWebTest.

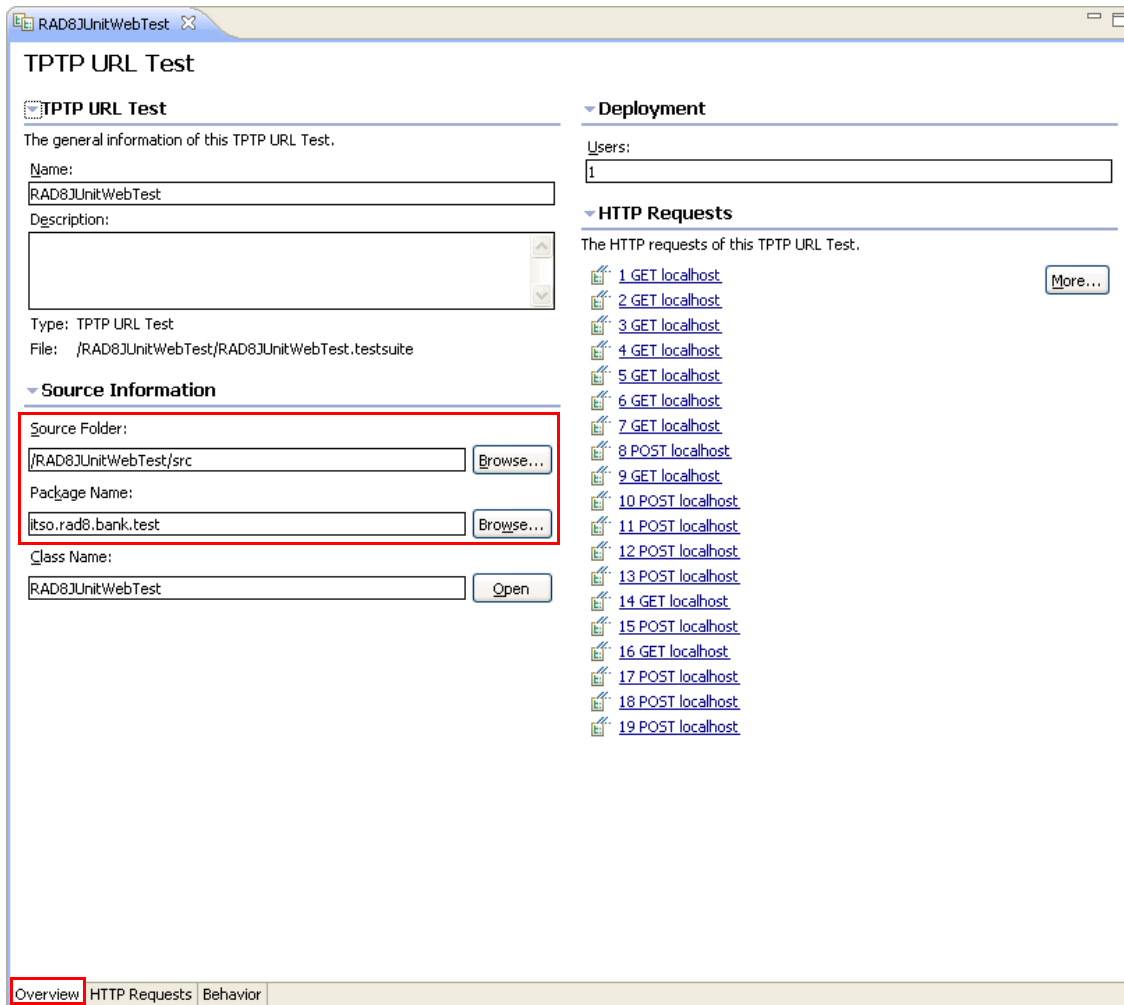


Figure 26-18 TPTP URL Test window: Overview tab

2. Select the **Behavior** tab of the TPTP URL Test editor (Figure 26-19). Change the behavior of the test. For example, we adjust the number of iterations. Save and close the Test editor.

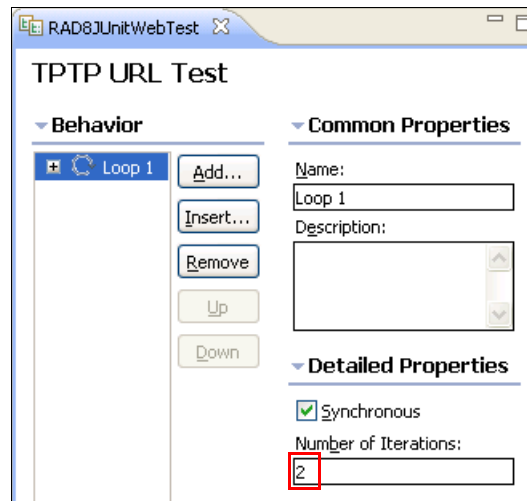


Figure 26-19 TPTP URL Test window: Behavior tab

## 26.6.4 Generating an executable test

Before you can run a test, you must generate and compile the Java source code for the test. This process is called *code generation*. The compiled code is stored in the RAD8JUnitWebTest project. To start the code generation for this project, follow these steps:

1. Right-click **RAD8JUnitWebTest TPTP URL Test** in the RAD8JUnitWebTest project and select **Generate**.
2. In the TPTP URL Test Definition Code Generation window, accept the project and source folder and click **Finish** to start the code generation.
3. To examine the generated Java code, switch to the Java or Web perspective and open the **itso.rad8.bank.test.RAD8JUnitWebTest** class.


## 26.6.5 Running the test


To run the RAD8JUnitWebTest, follow these steps:

1. Switch to the **Test** perspective.
2. Right-click **RAD8JUnitWebTest TPTP URL Test** in the RAD8JUnitWebTest project and select **Run As** → **Test**.

The test executes and creates a result.

## 26.6.6 Analyzing the test results

When the test run is finished, the execution result  is displayed in the Test Navigator view. To analyze the test results, follow these steps:

1. Double-click the execution result  **RAD8JUnitWebTest[<timestamp>]** file in the Test Navigator view. The Test Log Overview tab is displayed (Figure 26-20 on page 1413).

The test log gives the test verdict and the start and stop time of the test run. The verdict can be one of the following possibilities:

<b>fail</b>	One or more requests returned an HTTP code of 400 or greater, or the server was unable to be reached during playback.
<b>pass</b>	No request returned a code of 400 or greater.
<b>inconclusive</b>	The test did not run to completion.
<b>error</b>	The test itself contains an error.



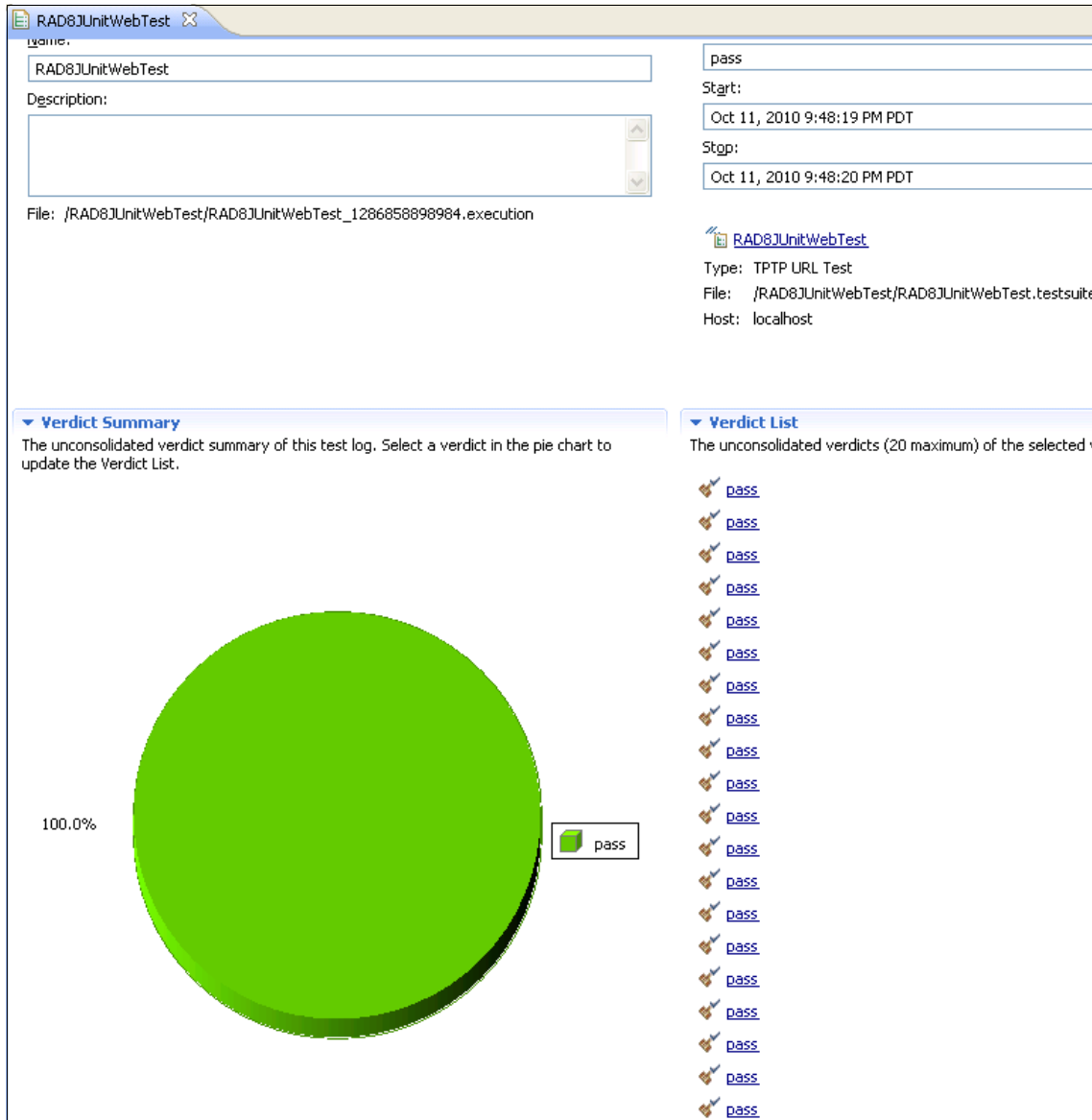


Figure 26-20 Test Log Overview tab

**Error messages:** In applications that use session data, you can have error messages, because the session ID is stored in the generated test case. When you rerun these test cases, a new session ID is created by the server, and it does not match the recorded session ID.

The problem is a known Eclipse issue. You can find discussions about the problem at the following web pages:

- ▶ Bug 128613: HTTP-Header toUpperCase() causes wrong Session ID and wrong webapp path  
[https://bugs.eclipse.org/bugs/show\\_bug.cgi?id=128613](https://bugs.eclipse.org/bugs/show_bug.cgi?id=128613)
- ▶ Bug 139699: HttpResponse class does not expose the complete HTTP response  
[https://bugs.eclipse.org/bugs/show\\_bug.cgi?id=139699](https://bugs.eclipse.org/bugs/show_bug.cgi?id=139699)

2. Click the **Events** tab (26.6.7, “Generating test reports” on page 1416) for detailed information about each HTTP request.

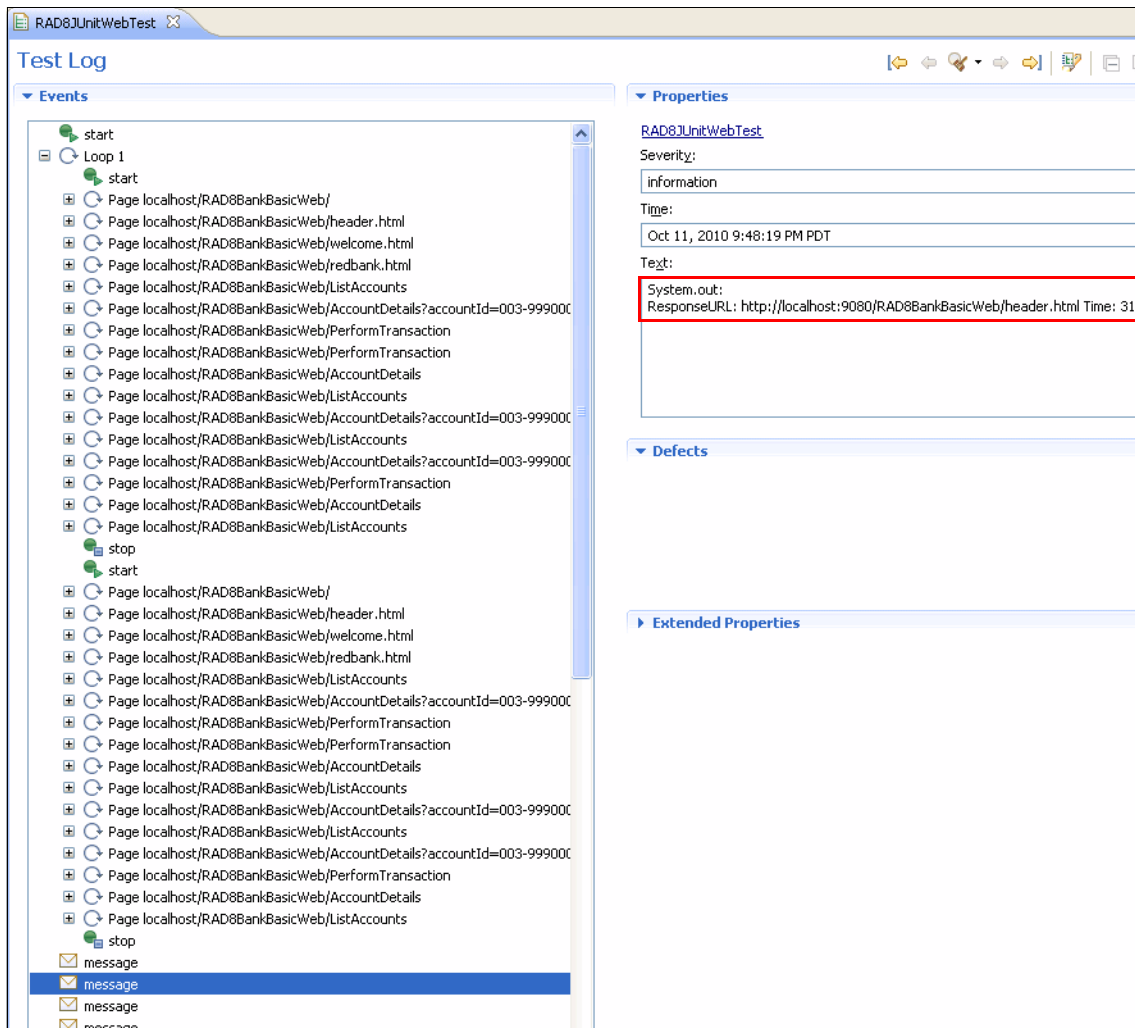


Figure 26-21 Events tab

## 26.6.7 Generating test reports

Based on a test execution results file, you can generate various kinds of analysis reports:


- ▶ The *HTTP Page Response Time report* is a bar graph that shows the seconds that are required to process each page in the test and the average response time for all pages.
- ▶ The *HTTP Page Hit Rate report* is a bar graph that shows the hits per second to each page and the total hit rate for all pages.

You can also generate the Test Pass report in the same way that you generate it for the basic JUnit tests.

**Viewing the reports:** To view the reports, you have to install the Scalable Vector Graphics (SVG) browser plug-in.

### HTTP Page Response Time report

To generate an HTTP Page Response Time report, follow these steps:

1. In the Test Navigator view, right-click the **RAD8JUnitWebTest** test suite  and select **Report**.
2. In the New Report window, select **HTTP Page Response Time** and click **Next**.
3. In the New Report window, accept the parent folder (**RAD8JUnitWebTest**). In the Name field, enter `RAD8JUnitWebTest_HTTPPageResponseTime`. Click **Finish**.
4. If you have multiple test execution results, in the HTTP Report Generator dialog window, select the test execution result for which the report is generated and click **Finish**.

An HTTP Page Response Time report is generated and opens in the browser. Alternatively, you can click the generated file and select **Open With** → **Web Browser** (Figure 26-22 on page 1417).

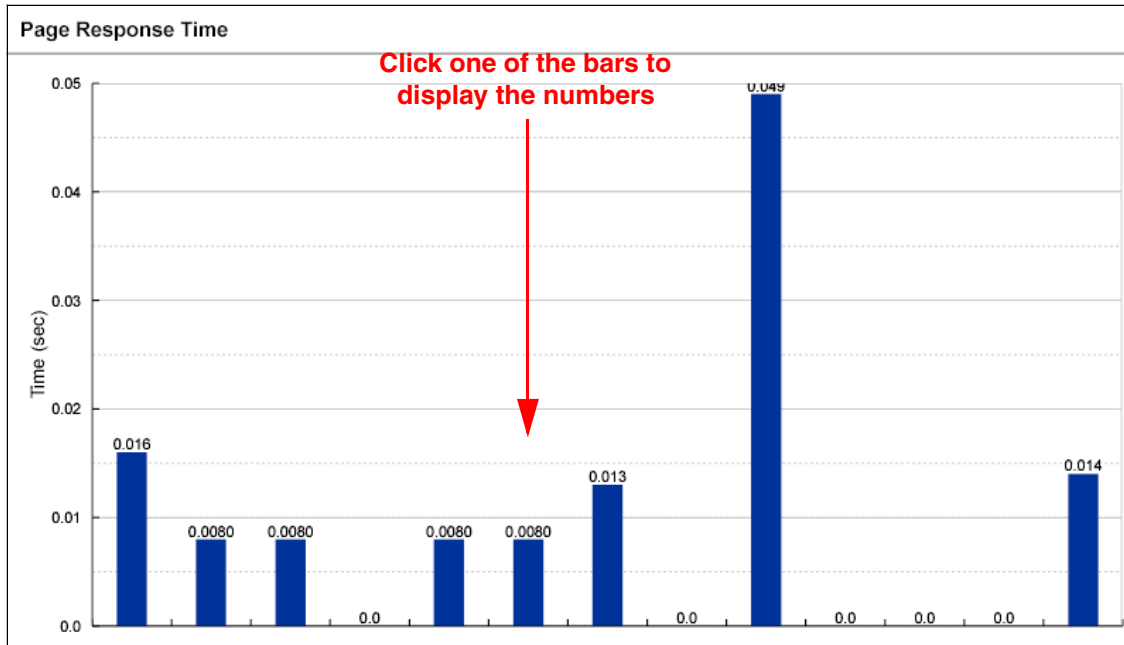



Figure 26-22 HTTP Page Response Time report

## HTTP Page Hit Rate report

To generate a HTTP Page Hit Rate report, follow these steps:

1. In the Test Navigator view, right-click the **RAD8JUnitWebTest** test suite  and select **Report**.
2. In the New Report window, select **HTTP Page Hit Rate** and click **Next**.
3. In the New Report window, accept the parent folder (**RAD8JUnitWebTest**). In the Name field, enter `RAD8JUnitWebTest_HTTPPageHitRate` and click **Finish**.
4. If you have multiple test execution results, in the HTTP Report Generator window, select the test execution result for which the report is generated and click **Finish**.

An HTTP Page Hit Rate report is generated and opens in the browser.

Alternatively, you can click the generated file and select **Open With** → **Web Browser** (Figure 26-23 on page 1418).

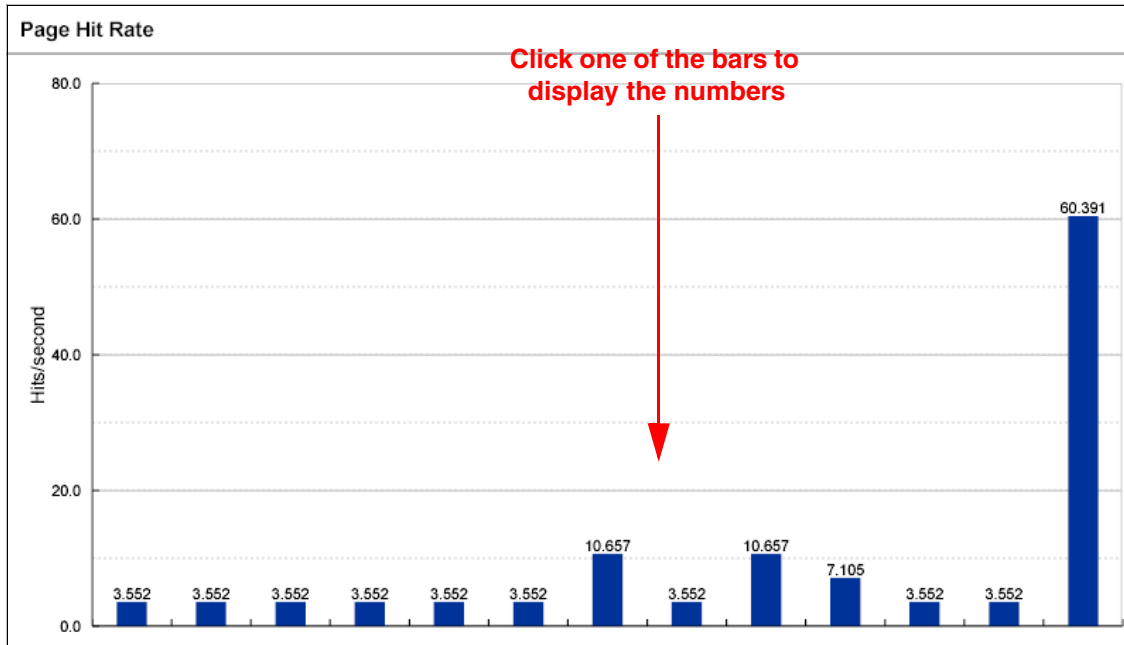


Figure 26-23 HTTP Page Hit Rate report

## 26.7 Cleaning the workspace

When enabling the test perspective or Core Testing Support capability, the Java builder is changed to ignore XML and HTML files. To fix the workspace, select **Window** → **Preferences** → **Java** → **Compiler** → **Building**. Under the Output folder, Filtered resources, remove **\*.xml** and **\*.html** from the text, as shown in this example:

```
.launch,.testsuite,*.deploy,*.location,*.execution,*.datapool,*.artifact,*.testlog,*.xml,*.html,*.svg
```



## Profiling applications

*Profiling* is a technique that developers use to collect runtime data and detect application problems, such as memory leaks, performance bottlenecks, excessive object creation, and exceeding system resource limits during the development phase. In this chapter, we introduce the features, architecture, and process for profiling applications using the profiling features of IBM Rational Application Developer. We also include an example for basic memory analysis, execution-time analysis, and thread/contention analysis.

The chapter is organized into the following sections:

- ▶ Introduction to profiling
- ▶ Preparing for the profiling sample
- ▶ Profiling a Java application
- ▶ Profiling a web application running on the server

## 27.1 Introduction to profiling

Traditionally, performance analysis is performed after an application nears deployment or after it has already been deployed. By using the profiling tools that are included with Rational Application Developer, the developer can move the performance analysis to a much earlier phase in the development cycle. By doing so, the developer has more time for changes to the application that might affect the architecture of the application before the changes become critical production environment issues.

With Rational Application Developer profiling, you can detect the following problems among others:

- ▶ Memory usage problems
- ▶ Performance bottlenecks
- ▶ Excessive object creation

You can use the profiling tools in the following cases to gather data on applications that are running in these situations:

- ▶ Inside an application server, such as WebSphere Application Server
- ▶ As a stand-alone Java application
- ▶ On the same system as Rational Application Developer
- ▶ In multiple Java virtual machines (JVMs)
- ▶ On a remote WebSphere Application Server with the IBM Rational Agent Controller installed. IBM Rational Agent Controller V8.3 can be used for remote profiling on any WebSphere Application Server server running the Java Development Kit (JDK) V1.5 or later.

**IBM Rational Agent Controller:** IBM Rational Agent Controller V8.3 and V8.3.1 are available on the following platforms:

- ▶ Microsoft Windows 32-bit and 64-bit
- ▶ Linux 32-bit and 64-bit
- ▶ AIX® 32-bit and 64-bit
- ▶ z/OS 31-bit and 64-bit
- ▶ Solaris SPARC 32-bit and 64-bit
- ▶ Solaris x86 32-bit and 64-bit
- ▶ Linux for System z 31-bit and 64-bit

IBM Rational Agent Controller V8.3 or later is required to use profiling without manually setting up environment variables.



## 27.1.1 Profiling features

Profiling with Rational Application Developer includes several analysis types. Each analysis type has views with which you can focus on particular problems, such as memory leaks, performance bottlenecks, and excessive object creation while profiling an application.

We describe the following analysis types and their associated views in this section:

- ▶ Basic memory analysis
- ▶ Execution-time analysis
- ▶ Thread/contention analysis

For more details about Code Coverage tooling in IBM Rational Application Developer, you can visit the link:

<http://www.ibm.com/developerworks/rational/library/10/introtocodecoveragegetoolinrationalapplicationdeveloper/?ca=drs->

### Basic memory analysis

*Basic memory analysis* shows statistics about the application heap. It is used to detect memory management problems. Memory analysis can help developers identify memory leaks and excessive object allocation that might cause performance problems. Basic memory analysis has been enhanced with the view that is listed in Table 27-1.

Table 27-1 *Basic memory analysis view*

View name	Description
Object allocations	Shows statistics about the application heap. It provides detailed information, such as the number of classes loaded, the number of instances that are alive, and the memory size that is allocated by every class.

### Execution-time analysis

*Execution-time analysis* is used to detect performance problems by highlighting the most time intensive areas in the code. This type of analysis helps developers identify and remove unused or inefficient coding algorithms. Execution-time analysis has been enhanced with the views that are listed in Table 27-2.

Table 27-2 *Execution-time analysis views*

View name	Description
Execution statistics	Shows statistics about the application execution time.

View name	Description
Call tree	Shows information about method calls during the profiling session in a form that lets you easily identify a hot spot. This view consists of two parts: the execution flow call tree and the call stack view.
Method invocation	Shows a graphical representation of the entire course of a program's execution and provides the ability to navigate through the methods that invoked the selected method.
Method invocation details	Shows statistical data on a selected method.
UML2 trace interactions	Shows the execution flow of an application according to the notation that is defined by the Unified Modeling Language (UML).

## Thread analysis

*Thread analysis* enables an application developer to find threads that otherwise run sooner, or more rapidly, if the resource and thread characteristics of the application are altered. The Thread Analysis view of Rational Application Developer consists of three tabs, as listed in Table 27-3.

Table 27-3 Thread analysis tabs

Tab name	Description
Thread Statistics	A table of statistics for every thread that is launched by the application, both past and present. Listed information includes the thread state; total running, waiting, and blocked times; and the number of blocks and deadlocks per thread.
Monitor Statistics	Provides detailed information about monitor class statistics, including block and wait statistics for individual monitor classes.
Threads Visualizer	Provides a visual representation of all threads that are profiled in the target application, by status.

## Probekit analysis

*Probes* are reusable Java code fragments that you write to collect detailed runtime data about a program's objects, instance variables, arguments, and exceptions. *Probekit* provides a framework on the Eclipse platform to help you create and use probes. One common use of Probekit is to create lightweight profilers that collect only the data in which developers are interested.

A probekit contains one or more probes, and each probe contains one or more probe fragments. You can specify when probes are executed, and on which programs they execute. The *probe fragments* are a set of Java methods that are merged with standard boilerplate code with a new Java class generated and compiled. The functions generated from the probe fragments appear as static methods of the generated probe class.

The probekit engine, which is also called the *byte-code instrumentation* (BCI) engine, is used to apply probe fragments by inserting the calls into the target programs. The insertion process of the call statements into the target methods is referred to as *instrumentation*. The data items that are requested by a probe fragment are passed as arguments (for example, method name and arguments). The benefit of this approach is that the probe can be inserted into a large number of methods with small overhead.

Probe fragments can be executed at the following points:

- ▶ At method entry or exit time
- ▶ At exception handler time
- ▶ Before the original code in the class static initializer
- ▶ Before every line of executable code when source code is available
- ▶ When specific methods are called, which are not inside the called method

Each of the probe fragments can access the following data:

- ▶ Package, class, and method name
- ▶ Method signature
- ▶ This object
- ▶ Arguments
- ▶ Return value

Two major types of probes are available to the user (Table 27-4).

Table 27-4 *Types of probes available with Probekit*

Type of probe	Description
Method probe	This probe can be inserted anywhere within the body of a method with the class or JAR files containing the target methods instrumented by the BCI engine.
Callsite probe	This probe is inserted into the body of the method that calls the target method. The class or JAR files that call the target are instrumented by the BCI engine.

## 27.1.2 Profiling architecture

The profiling architecture that exists in Rational Application Developer is based on the Eclipse Test and Performance Tools Platform (TPTP) project. You can find more detailed information about the Eclipse TPTP project at the following web address:

<http://www.eclipse.org/tptp/>

In the previous TPTP workbench, users required the services of the stand-alone Agent Controller before they were able to use the function in the Profiling and Logging perspective and in the Test perspective. Even when the user tried to profile a Java application locally or to run TPTP tests locally, the Agent Controller had to be installed on the local machine.

The *Integrated Agent Controller* is a new feature in the TPTP workbench, with which users can profile a Java application locally and run a TPTP test locally without requiring the stand-alone Agent Controller on the local machine. Profiling on a remote machine or running a TPTP test on a remote machine still requires the Agent Controller on that remote machine.

This feature is packaged in the TPTP runtime install image; therefore, no separate install step is required. The Integrated Agent Controller does not require any configuration at all. Unlike the Agent Controller, which requires the user to enter information, such as the path for the Java executable, the Integrated Agent Controller determines the required information from the Eclipse workbench during start-up.

TPTP provides the Agent Controller daemon with a process for enabling client applications to start host processes and interact with agents that exist within host processes. Figure 27-1 on page 1425 shows the profiling architecture.

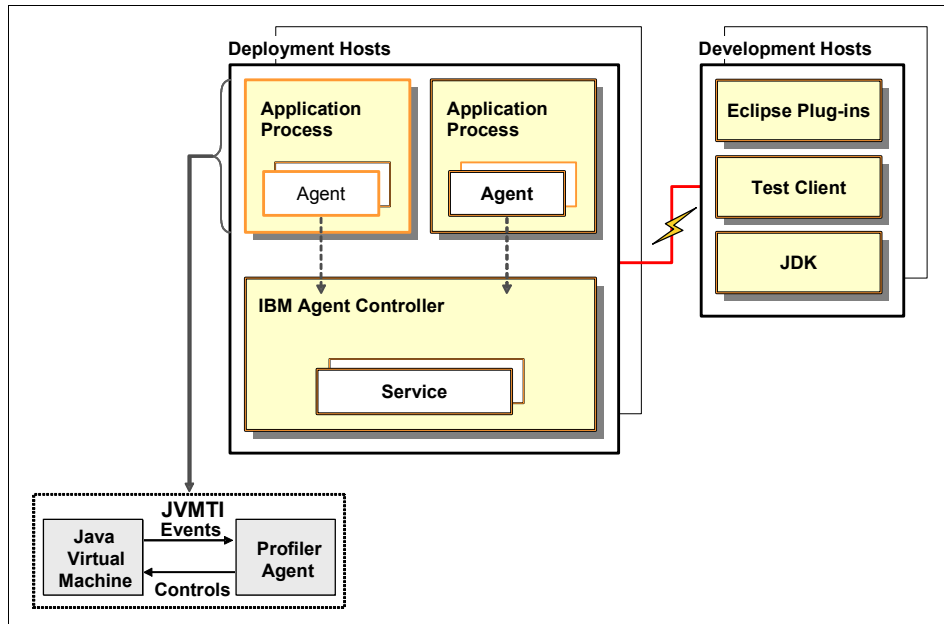


Figure 27-1 Profiling architecture of Rational Application Developer

The profiling architecture consists of the following components:

**Application process** The process that is executing the application that consists of the JVM and the profiling agent.

**Agent** The profiling component installed with the application that provides services to the host process and more importantly provides a portal by which application data can be forwarded to attached clients.

**Test Client** A local or remote application that is the destination of host process data that is externalized by an agent. A single client can be attached to many agents at the same time, but it does not always have to be attached to an agent.

**Agent Controller** A daemon process that resides on each deployment host, providing the mechanism by which client applications can either start new host processes, or attach to agents coexisting within existing host processes. The Agent Controller can only interact with host processes on the same node.

Rational Application Developer comes with an *integrated*

*agent controller*. Therefore, a separate install of the agent controller is not required.

**Deployment hosts** The host to which an application has been deployed and is being monitored for the capture of the profiling agent.

**Development hosts** The host that runs an Eclipse-compatible architecture, such as Rational Application Developer, to receive profiling information and data for analysis.

Each application process that is shown in Figure 27-1 on page 1425 represents a JVM that is executing a Java application that is being profiled. A profile agent is attached to each application to collect the appropriate runtime data for a particular type of profiling analysis. This profiling agent is based on the JVM Tools Interface (JVMTI) architecture. You can obtain more information about the JVMTI specification at the following web address:

<http://download.oracle.com/javase/6/docs/technotes/guides/jvmti/index.html>

The data that is collected by the agent is then sent to the Agent Controller, which then forwards this information to Rational Application Developer for analysis and visualization.

The following types of profiling agents are available in Rational Application Developer:

- ▶ The *Java Profiling Agent* is based on the JVMTI architecture and is shown in Figure 27-1 on page 1425. This agent is used for the collection of both stand-alone Java applications and applications that are running on an application server.
- ▶ The *J2EE Request Profiling Agent* resides in an application server process and collects runtime data for Java 2 Platform, Enterprise Edition (J2EE) applications by intercepting requests to the EJB or Web containers.

**Active instance:** Only one instance of the J2EE Request Profiling agent is active in a process that hosts the WebSphere Application Server.

### 27.1.3 Profiling and Logging perspective

You can access the Profiling and Logging perspective by selecting **Window** → **Open Perspective** → **Other** → **Profiling and Logging** and then by clicking **OK**. If it is not listed, click **Show all**.

If the Profiling and Logging capability is not enabled in the workspace, you are prompted to enable this capability. Click **OK**.

The Profiling and Logging perspective has many supporting views. To see the supporting views, select **Window** → **Show View** → **Other** under Profiling and Logging (Figure 27-2).

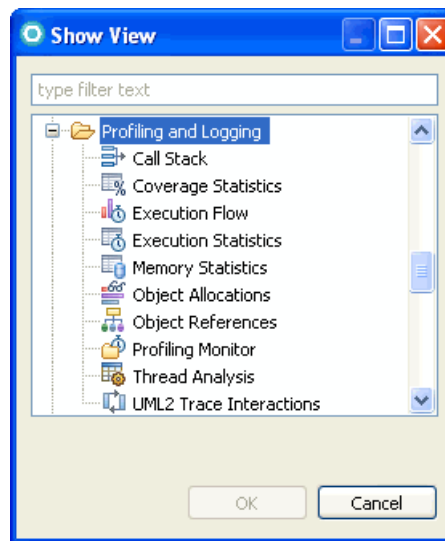


Figure 27-2 Profiling and Logging views

## 27.2 Preparing for the profiling sample

In this section, we explain the tasks that you must complete prior to profiling the sample web application. We use the web application that was developed in Chapter 12, “Developing Enterprise JavaBeans (EJB) applications” on page 577, as our sample application for profiling.

Complete the following tasks in preparation for the profiling sample:

- ▶ Installing the prerequisite software
- ▶ Enabling the Profiling and Logging capability

### 27.2.1 Installing the prerequisite software

The working example requires that you install the following software:

- ▶ IBM Rational Application Developer
- ▶ Integrated Agent Controller

This feature is packaged in the installation image of Rational Application Developer. Therefore, no separate installation step is required.

## 27.2.2 Enabling the Profiling and Logging capability

The Profiling and Logging capability is enabled by default. In order to check if it is enabled, follow these steps:

1. Select **Window** → **Preferences**.
2. In the Preferences window (Figure 27-3), in the left pane, expand **General** → **Capabilities**. In the right pane, click **Advanced**.

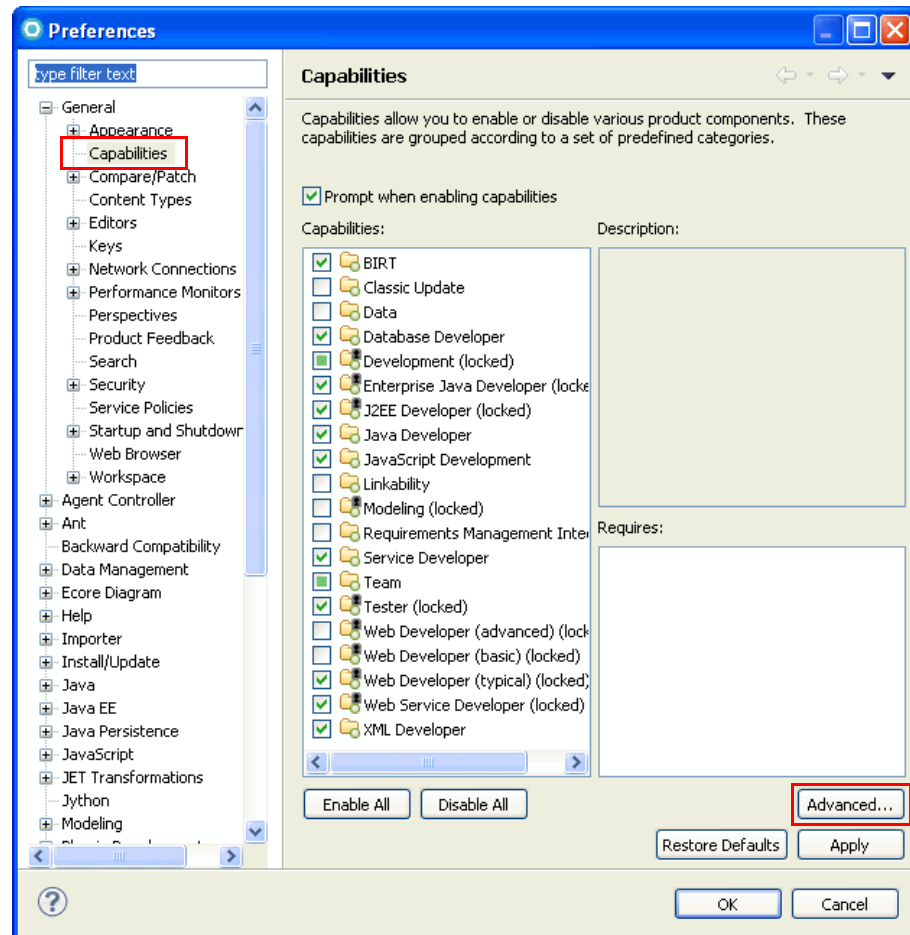


Figure 27-3 Preferences window

3. In the Advanced Capabilities Settings window (Figure 27-4 on page 1429), expand **Tester** and select **Profiling and Logging**. Then click **OK**.



**Probekit:** If you want to use Probekit, select **Probekit** in the Advanced Capabilities Settings window.

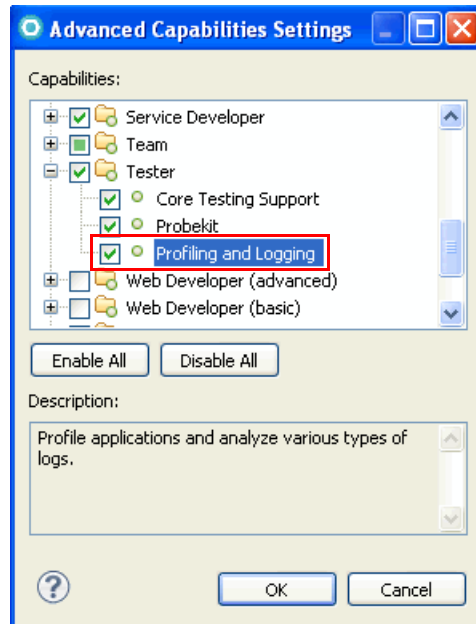


Figure 27-4 Advanced Capabilities Settings window

## 27.3 Profiling a Java application

In this section, we profile a Java application. We import the sample code and run the application in profiling mode.

### 27.3.1 Importing the sample project archive file

**Tip:** If you have the sample Java Persistence API (JPA) application (RAD8JPA and RAD8JPATest projects) already in the workspace, skip this step.

To import the JPA application project archive file, follow these steps:

1. Open the Java EE perspective.
2. Select **File** → **Import**.

3. In the Import window, expand **General** and select **Existing Projects into Workspace**. Click **Next**.
4. In the Import Projects window, click **Browse** and locate the `c:\7835codesolution\jpa\RAD8JPA.zip` file.
5. Select the **RAD8JPA** and **RAD8JPATest** projects and click **Finish**.

Alternatively you can run the Java application (`BankClient`) in the `RAD8Java` project in profiling mode.

**Important:** If you have not already configured the data source settings for the `ITSOBANK` database, follow the steps in “Configuring the data source for the `ITSOBANK`” on page 609, before publishing and running the sample application.

## 27.3.2 Creating a profiling configuration

We use the `EntityTester` class (in `RAD8JPATest`, `itso.bank.entities.test`) as the sample application. See 10.6, “Testing JPA entities” on page 501, for a description of the `EntityTester` class. To create a profiling configuration, follow these steps:

1. Right-click **EntityTester** and select **Profile As** → **Profile Configurations**.
2. In the Profile Configurations window, double-click **Java Application** and an entry named `EntityTester` is added and opened.
3. On the Arguments tab (Figure 27-5 on page 1431), under the Program arguments, type `333-33-3333`. For the VM arguments, enter this information:  
`-javaagent:"<RAD_HOME>/runtimes/  
base_v8_stub/plugins/com.ibm.ws.jpa.jar"` (enter it all on one line).

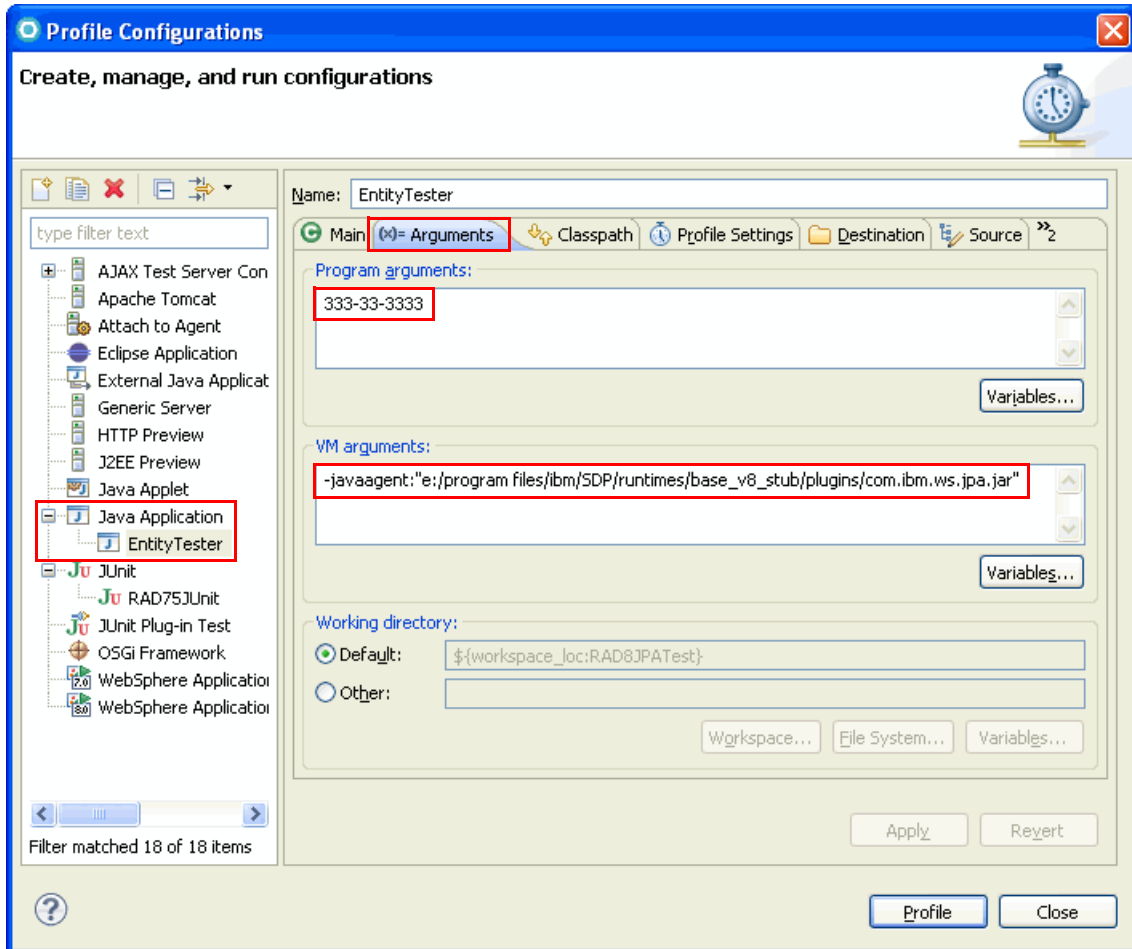


Figure 27-5 Profile Configurations: Arguments

4. On the Profile Settings tab (Figure 27-6), select **Execution Time Analysis**. Click **Edit Options**.

**One analysis type:** You can only select one analysis type. See the Technote at the following web address:

<http://www-01.ibm.com/support/docview.wss?uid=swg21328379>

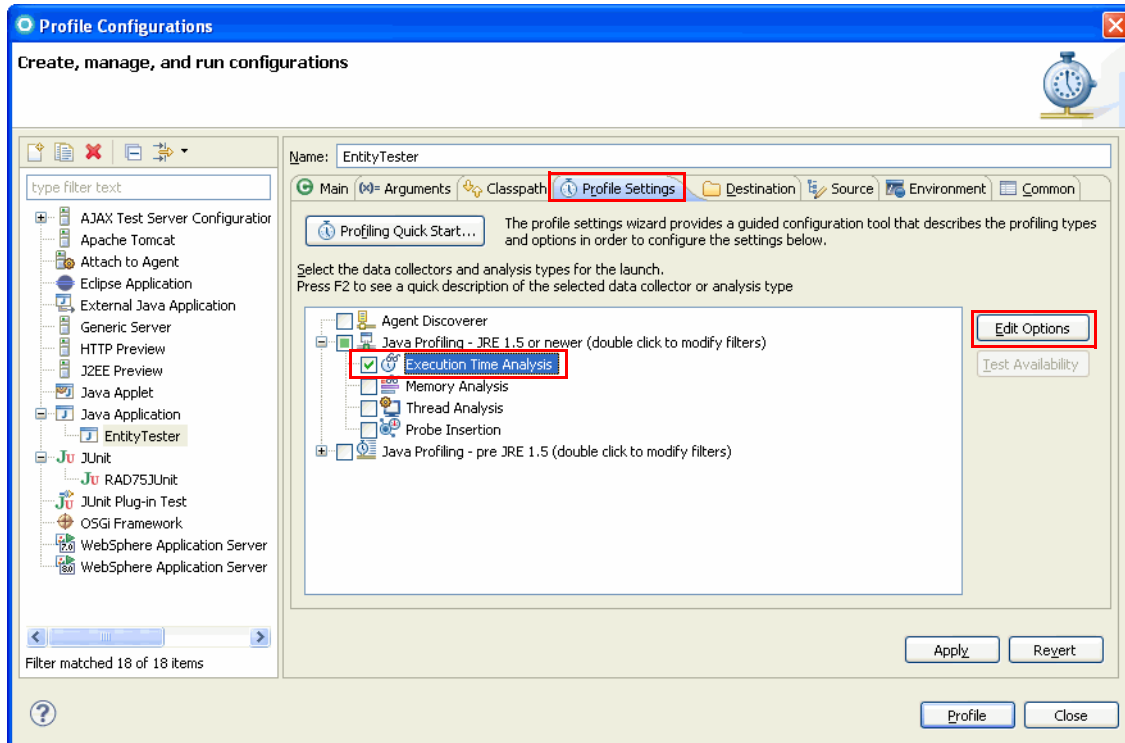


Figure 27-6 Profile settings

**Tip:** Users of Rational Application Developer can now use the Profiling Quick Start button to bring up a wizard that will guide them through selecting the correct profiling options to fit their usage scenarios. Figure 27-7 shows the first step of the Profiling Quick Start Wizard.

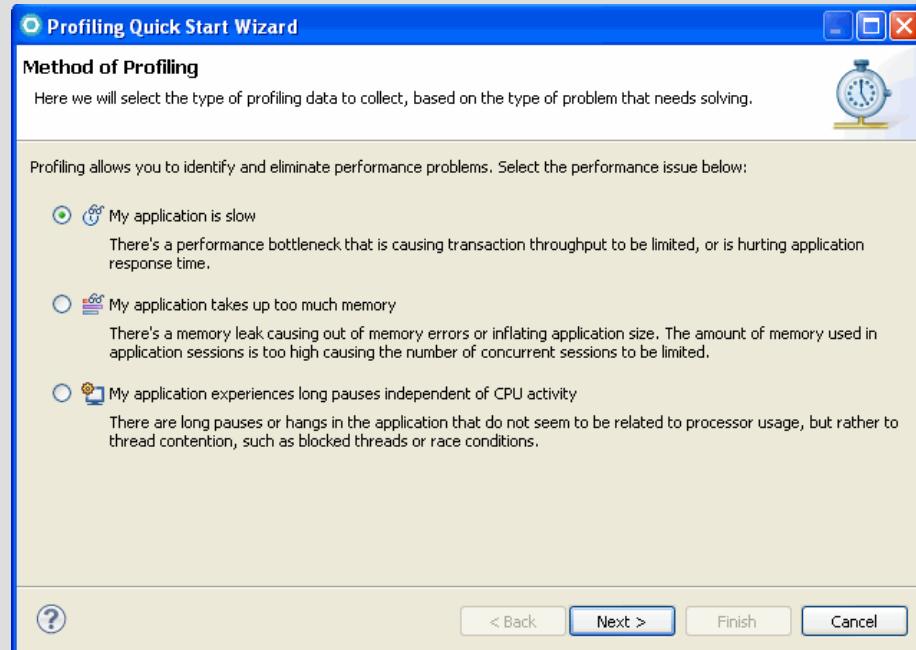


Figure 27-7 Profiling Quick Start Wizard

The three profiling options filter the application problems based on time, memory, and throughput. More options are available to refine each of these options.

5. In the Edit Profiling Options window (Figure 27-8), select **Collect method CPU time information** and click **Finish**.

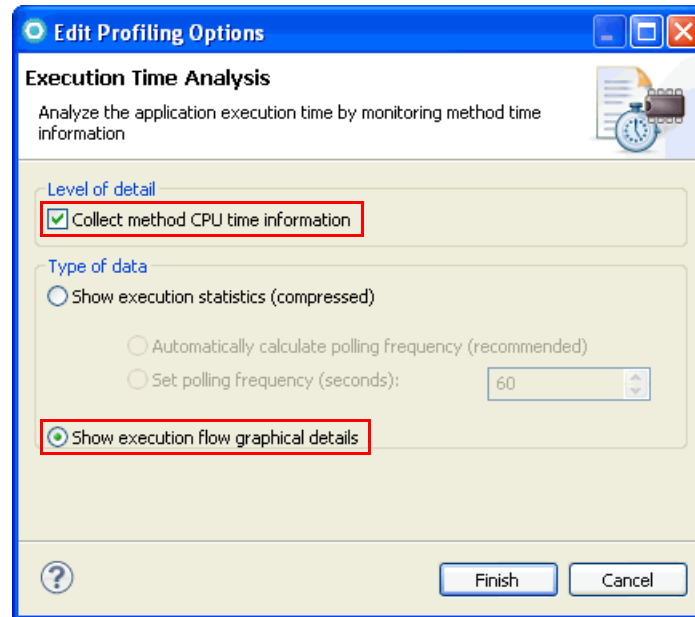


Figure 27-8 Profiling options for Execution-Time Analysis

6. In the Profile Configurations window, click **Apply** to save the configuration.

### 27.3.3 Running the EntityTester application

To run the EntityTester application, follow these steps:

1. In the Profile Configurations window, click **Profile** to run the application.
2. When prompted, click **Yes** to switch to the Profiling and Logging perspective.

In the Profiling Monitor view (Figure 27-9 on page 1435), you can see that execution time is being measured. In the Console view on the right, you can see the program running through its parts and displaying the output.

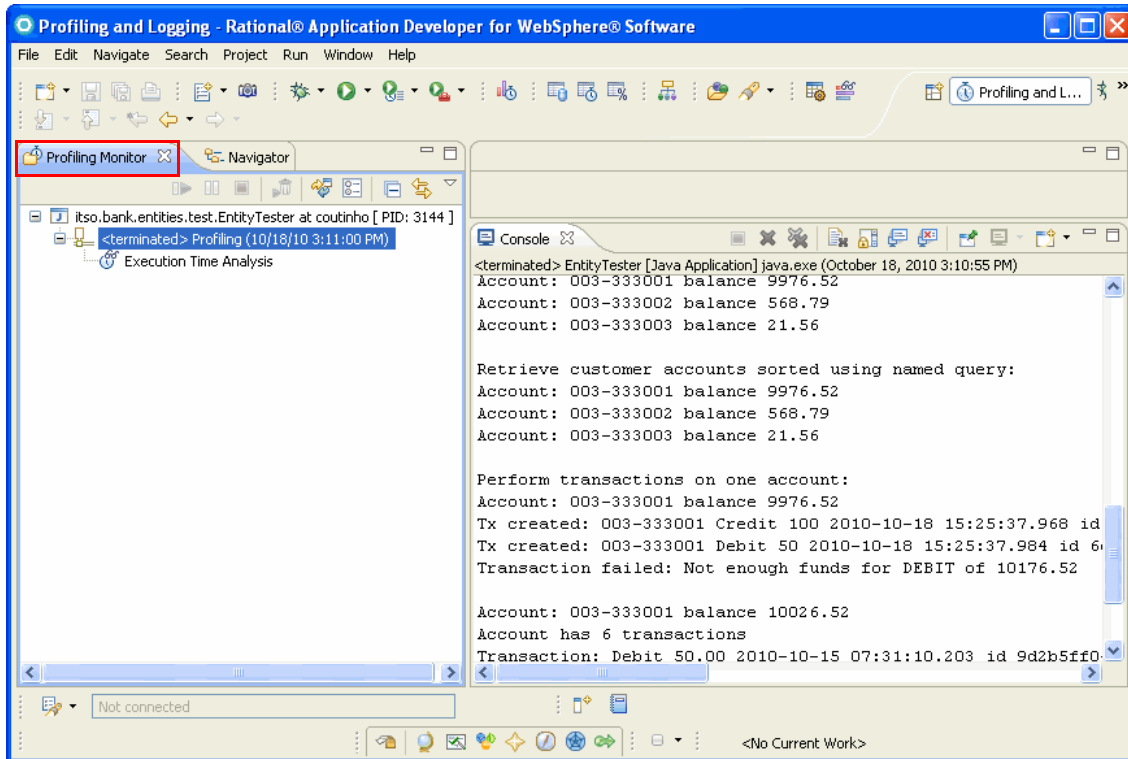


Figure 27-9 EntityTester run in profiling mode

## 27.3.4 Analyzing profiling data

We have now run the sample application for which we want to collect data. In this section, we analyze the collected data for execution statistics.


To display the collected data, right-click the process, select **Open With** → , and specify the desired option:

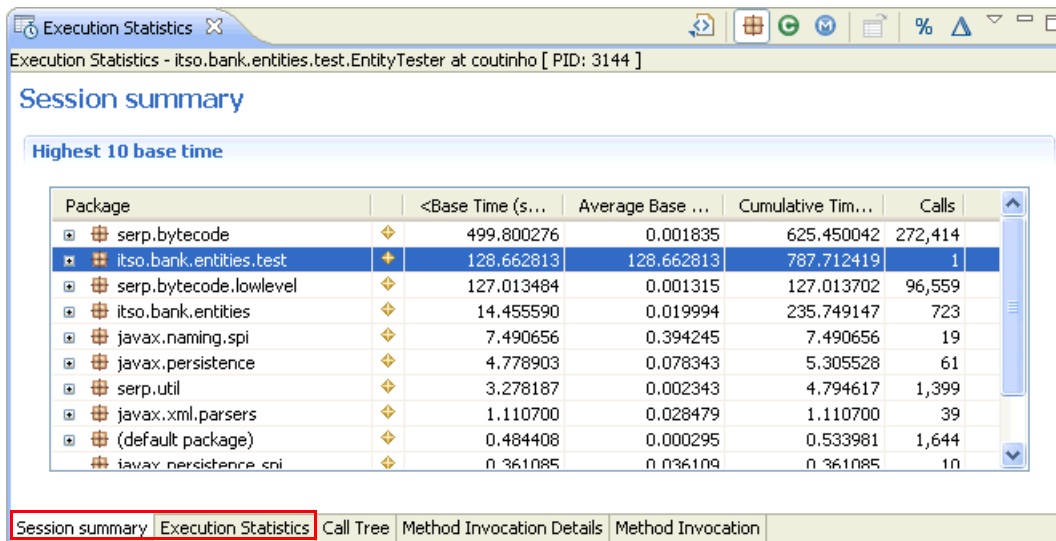
- ▶ Open Thread Analysis
- ▶ Open Object Allocation View
- ▶ Open Execution Statistics

## 27.3.5 Execution statistics

The Execution Statistics view shows statistics about the application execution time. It provides data, such as the number of methods called and the amount of time that is taken to execute every method. Execution statistics are available at the package, class, method, and instance level.

To analyze the execution statistics, follow these steps:

1. In the Profiling Monitor view, double-click **Execution Time Analysis**, click the  icon, or right-click the entry and select **Open With** → **Execution Statistics**. The Session summary tab and the Execution Statistics tab show the same data, but their filters differ.
2. On the **Execution Statistics** tab, set the filter. For example, we chose no filter. You can also choose the Highest 10 base time filter, which is the filter that was selected for the Session summary that is shown in Figure 27-10. You can also choose the Highest 10 cumulative time filter or other filters.




Package	<Base Time (s...	Average Base ...	Cumulative Tim...	Calls
serp.bytecode	499.800276	0.001835	625.450042	272,414
itso.bank.entities.test	128.662813	128.662813	787.712419	1
serp.bytecode.lowlevel	127.013484	0.001315	127.013702	96,559
itso.bank.entities	14.455590	0.019994	235.749147	723
javax.naming.spi	7.490656	0.394245	7.490656	19
javax.persistence	4.778903	0.078343	5.305528	61
serp.util	3.278187	0.002343	4.794617	1,399
javax.xml.parsers	1.110700	0.028479	1.110700	39
(default package)	0.484408	0.000295	0.533981	1,644
javax.persistence.spi	0.361085	0.036109	0.361085	10

Figure 27-10 Execution Statistics: Session summary



- Expand the packages and classes to see the accumulated values per class or per method (Figure 27-11).

Notice the highlighted icons (  ) in Figure 27-11 that are available to switch to package, class, method, and instance views; to open the source; to show data as percentages; and to add delta columns.

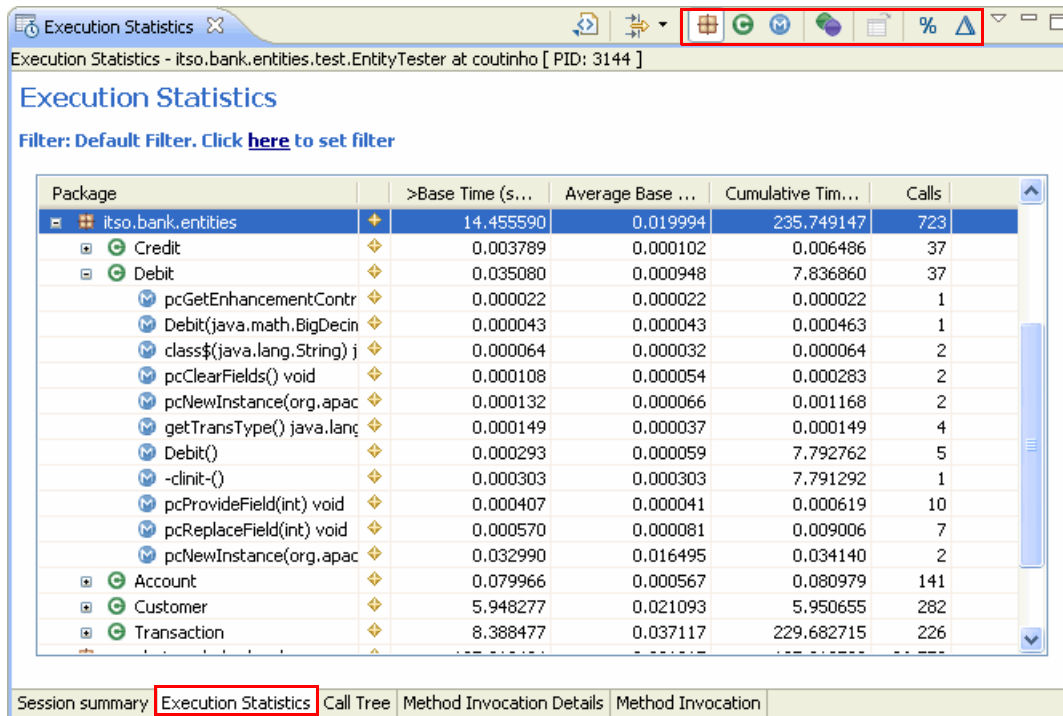


Figure 27-11 Execution Statistics: Expanded

Figure 27-11 shows the following statistics for each object type:

- ▶ *Base Time* refers to the time that is taken to execute the invocation (excluding the time that is spent in the called methods).
- ▶ *Average Base Time* is the base time divided by the number of calls.
- ▶ *Cumulative Time* is the time that is taken to execute the invocation (including the time that is spent in the called method).
- ▶ *Calls* refers to the number of calls that are made to the package, class, or method.

**JPA entity classes:** The JPA entity classes include several generated methods with the pc prefix that are generated for database access.

In the following sections, we describe the additional views that are available at the bottom of the Execution Statistics window (Figure 27-11 on page 1437).

## Call Tree

Select the **Call Tree** tab. Expand **main** → **main** → **processTransaction** → **Credit** to analyze the call tree and the percent of time that is spent in each method of the tree (Figure 27-12).

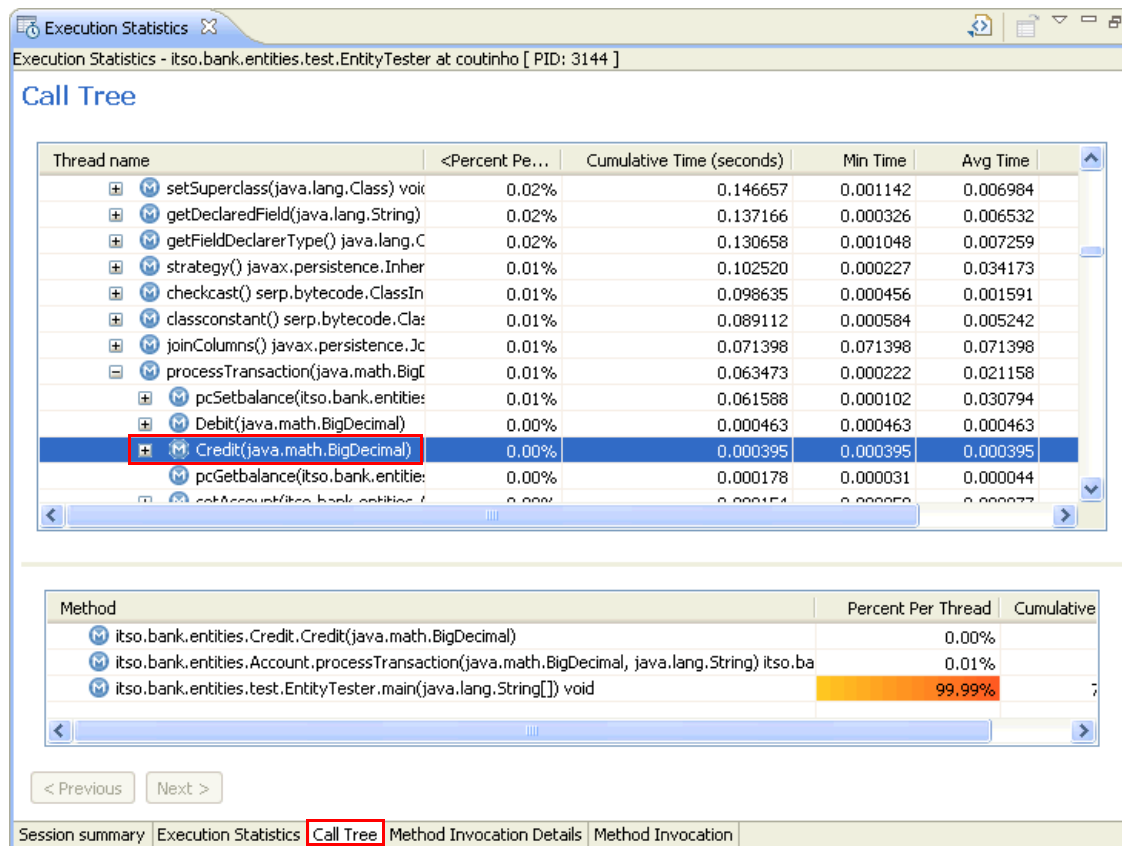


Figure 27-12 Execution Statistics: Call Tree

## Method Invocation Details

In the Execution Statistics tab, find and expand the **Account** class to see the methods of the Account class. Right-click over **processTransaction** and select **Show Method Invocation Details** (Figure 27-13 on page 1439).

The Method Invocation Details tab provides statistical data on a selected method.

The Show Method Invocation Details tab shows the following data for the selected method:

- ▶ *Selected method* (`Account.processTransaction`) shows details, including the number of times that the selected method is called (Calls), class, package, and the time that is taken by this method.
- ▶ *Selected method is invoked by* shows the details of each method that calls the selected method, including the number of calls to the selected method, and the number of times that the selected method is invoked by the caller. In our case, the `main` method invokes the `processTransaction` method.
- ▶ *Selected method invokes* shows the details of each method that is invoked by the selected method, for example, `Credit` and `Debit` constructors, `setAccount` of the `Transaction` class, and internal JPA methods.

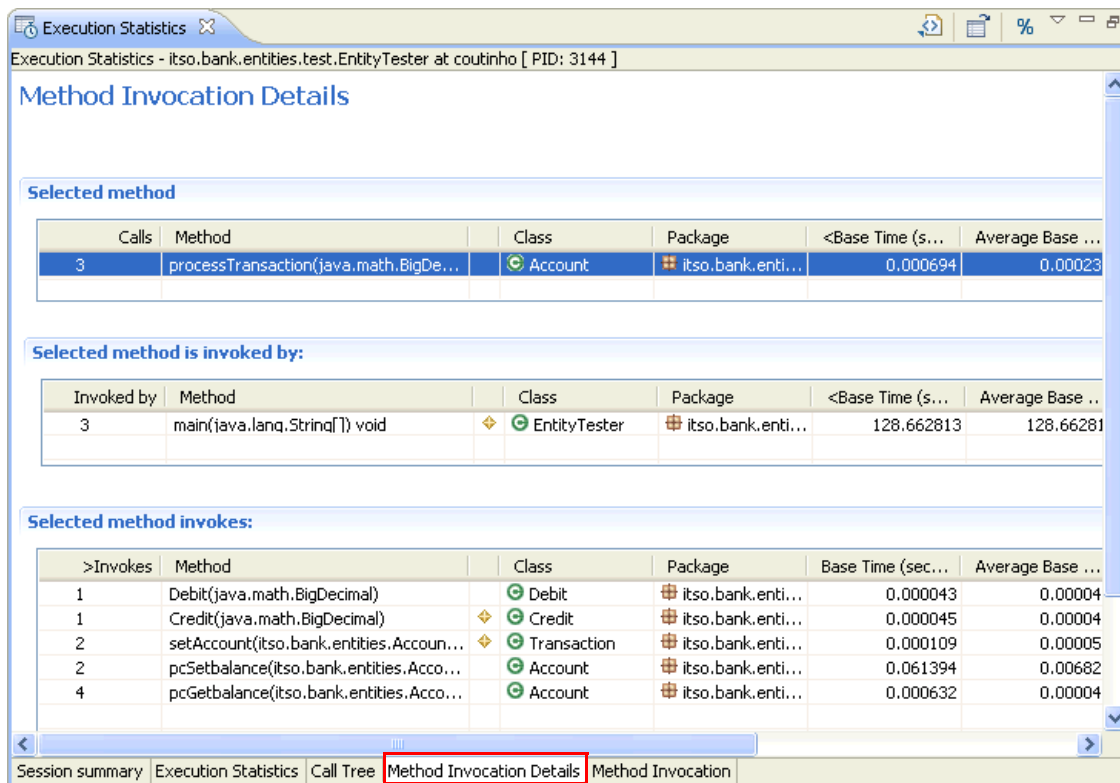


Figure 27-13 Execution Statistics: Method Invocation Details

## Method Invocation

Select the **Method Invocation** tab (Figure 27-14) to see a graphical representation of the calls.

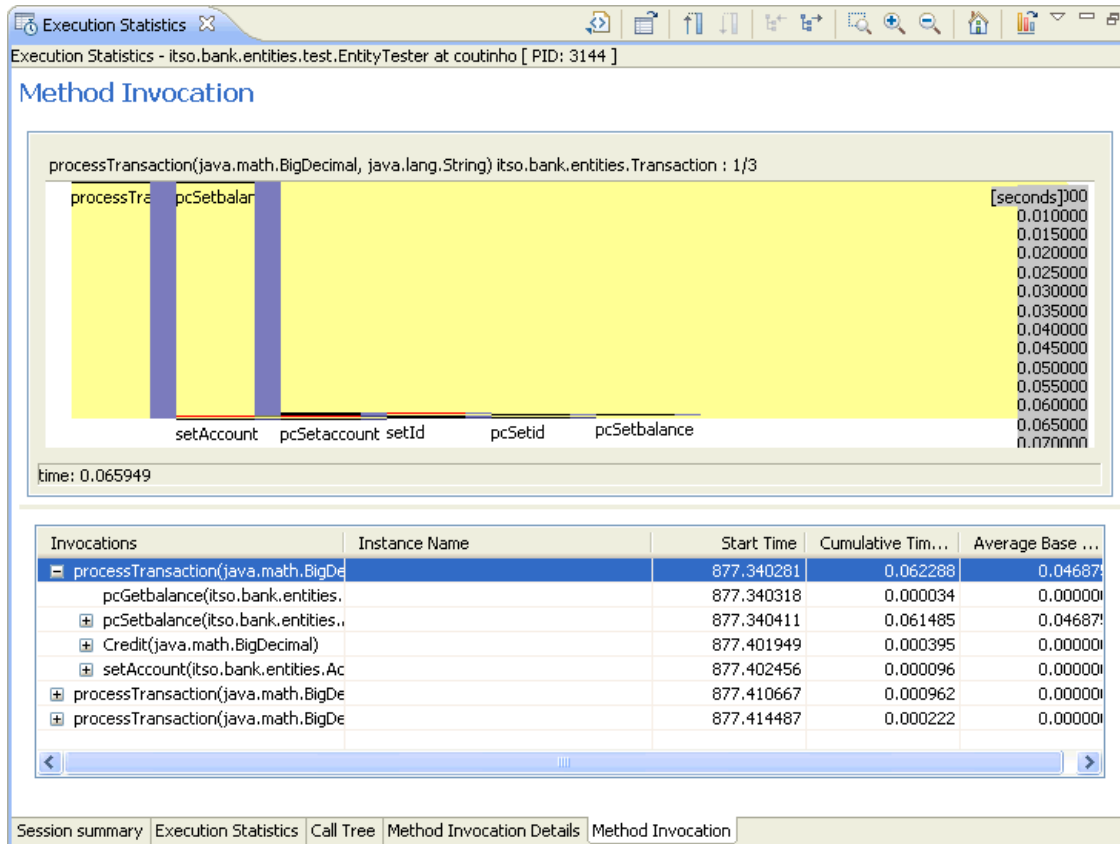



Figure 27-14 Execution Statistics: Method Invocation

### 27.3.6 Execution flow

The Execution Flow view and table both show a representation of the entire program execution. In this view, the threads of the program fit horizontally, and time is scaled so that the entire execution fits vertically. In the table, the threads are grouped in the first column and time is recorded in successive rows.

In the Profiling Monitor view, click the **Open Execution Flow** icon () or select **Open With** → **Execution Flow** (Figure 27-29 on page 1455). The bottom pane shows the action sequence. Expand **main** and select the first **main** method. The top pane shows the execution stripes (Figure 27-15 on page 1441).

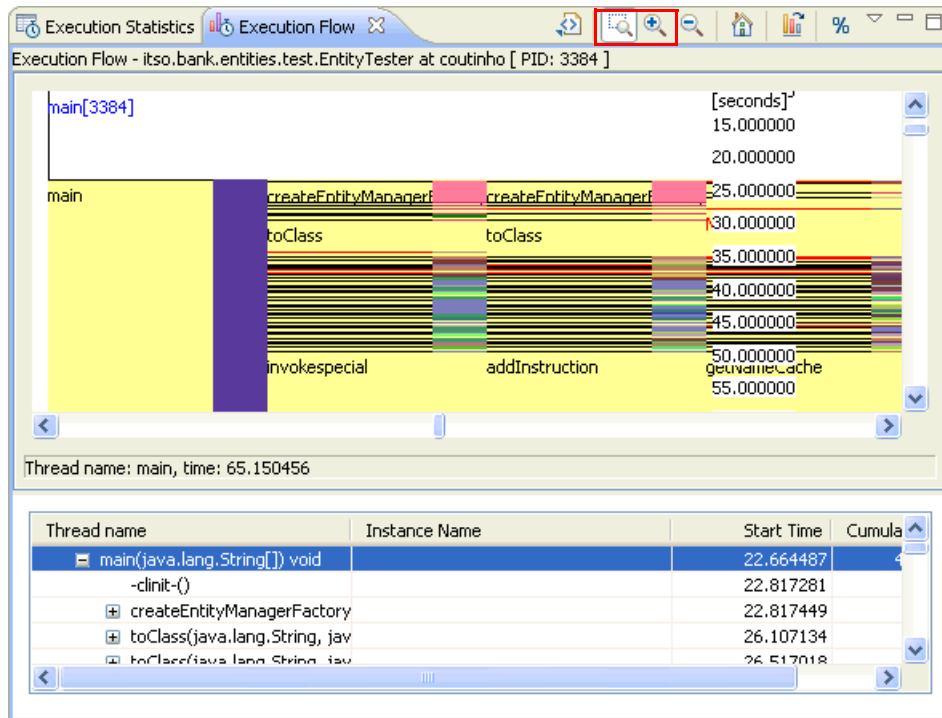


Figure 27-15 Execution Flow

Select the **Zoom In** icon (🔍) and click into the column to see more details.

### 27.3.7 UML sequence diagrams

You can also analyze the graphical details of the execution flow by using the data that is collected with Execution Time Analysis. These graphical details are displayed by using the Unified Modeling Language (UML) sequence diagram notation. The representation of time in these diagrams helps in determining bottlenecks in application performance and network communication. The following types of diagrams are available:

- ▶ The *UML2 Class Interactions* diagram shows the interactions of classes that participate in the execution of an application.
- ▶ The *UML2 Object Interactions* diagram shows the interactions of objects that participate in the execution of an application.
- ▶ The *UML2 Thread Interactions* diagram shows the interactions of methods that run in separate threads, which participate in the execution of an application.

To display UML2 interaction diagrams, in the Profiling Monitor view, right-click the entry and select **Open With** → **UML2 Class Interactions** (Figure 27-16).

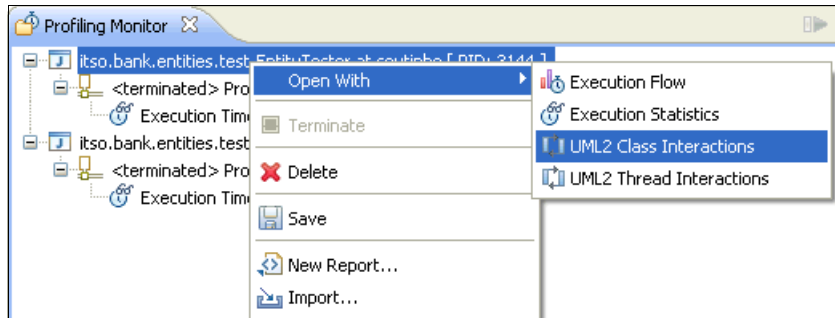


Figure 27-16 Selecting UML2 Class Interactions

Figure 27-17 on page 1443 shows the UML2 Trace Interactions diagram. You have to scroll to find suitable interactions.

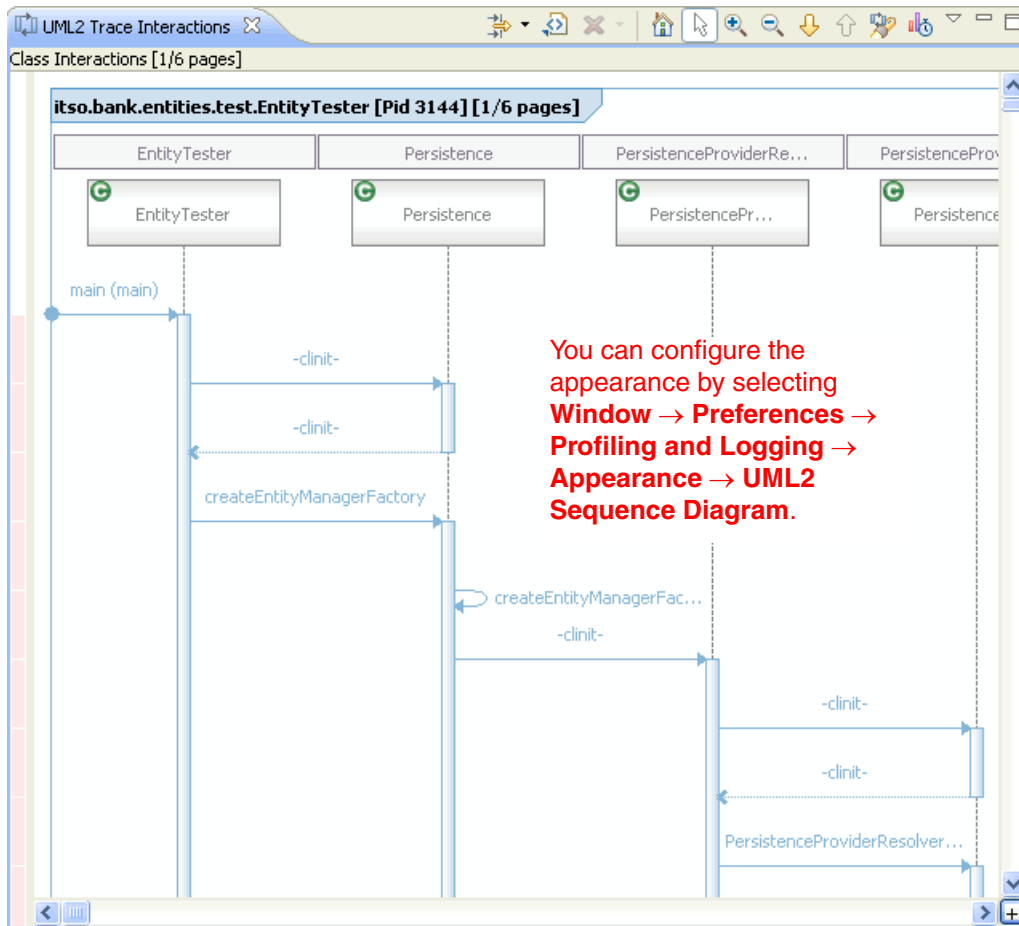


Figure 27-17 UML2 Trace Interactions

The other two UML diagrams are similar: UML2 Object Interactions and UML2 Thread Interactions.

You can drill down into a lifeline with which you can view all the trace interactions within a particular lifeline (right-click the class and select **Drill down into selected lifeline**). This feature helps to trace the root cause of a problem from a host, to a process, to a thread, and eventually to a class or an object.

You can highlight a call stack to view all of the methods' invocations in a call stack by right-clicking a method and selecting **Highlight call stack**.

## 27.3.8 Memory analysis

The Object Allocations view shows statistics about the application heap. It provides detailed information, such as the number of classes loaded, the number of instances that are alive, and the memory size that is allocated by every class. Memory statistics are available at the package, class, and instance level.


To analyze the memory consumption, rerun the application with another profiling option:

1. Select **Run** → **Profile Configurations**.
2. In the Profile Configurations window, select the **EntityTester** (preselected).
3. On the Monitor tab, select **Memory Analysis**.
4. Click **Edit Options** and select **Track object allocation sites**. Click **Finish**.

**Tip:** Another option in memory analysis is to collect heap instance data. It is a new feature in Rational Application Developer that provides the capability to inspect the composition of objects, including obtaining the live object values, member names, and instance sizes during the profiling session. The data that has been collected can be exported and imported as well.

5. Click **Apply** and then click **Profile**.

A new entry opens in the Profiling Monitor view. The Console view shows the application output while the application runs to completion.

6. Double-click **Memory Analysis**, or click the **Open Object Allocations view** icon ()

The Object Allocations view (Figure 27-18 on page 1445) shows live and total instances, active and total size, and average age:

- The *Total Instances* column shows the total number of instances that have been created of the selected package, class, or method.
- The *Live Instances* column shows the number of instances of the selected package, class, or method, where no garbage collection has taken place.
- The *Avg age* column shows the average number of garbage collections that objects of the class have survived.
- The *Total Size* column shows the total size in bytes of the selected package, class, or method, of all instances that were created for it.
- The *Active Size* column shows the total size in bytes of all live instances.



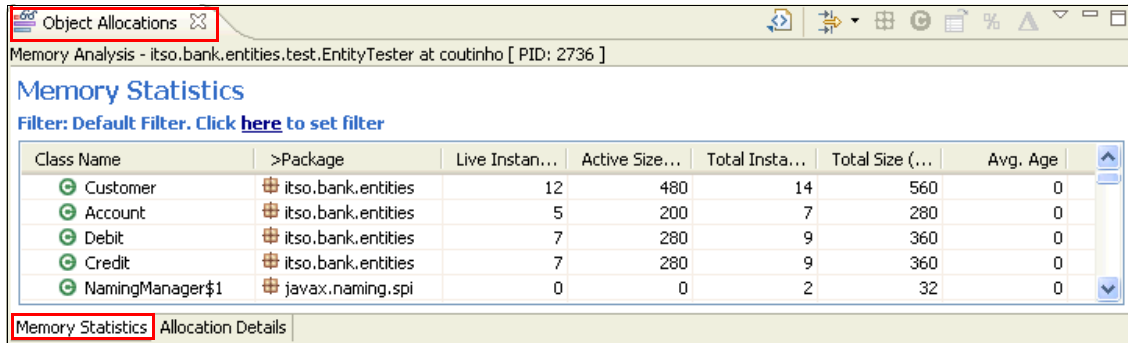


Figure 27-18 Object Allocations view

7. Select a class (**Credit**) and select the **Allocation Details** tab (Figure 27-19).

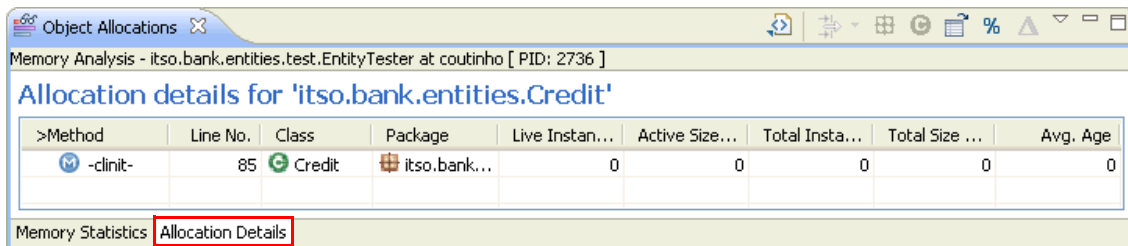


Figure 27-19 Object Allocation Details

### 27.3.9 Thread analysis

To analyze the threads, rerun the application with another profiling option:

1. Select **Run** → **Profile Configurations**.
2. For the EntityTester class, on the Monitor tab, select **Thread Analysis**.
3. Click **Edit Options** and select **Contention analysis**. Click **Finish**.
4. Click **Apply** and then click **Profile**.

A new entry appears in the Profiling Monitor view (Figure 27-20).

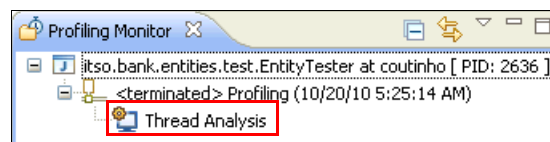


Figure 27-20 Profiling Monitor with three runs

5. Double-click **Thread Analysis**, or click the **Open Thread Analysis view** icon. The Thread Analysis view has the following options:
  - The Thread Statistics tab shows the state, running time, and waiting time of the threads (Figure 27-21).

Thread Name	Class Name	>State	Running Time	Waiting Time
Finalizer thread	java.lang.Thread	Stopped	00:39:487	
main	java.lang.Thread	Stopped	00:33:926	
Thread-2	java.lang.Thread	Stopped	00:33:987	
Attach handler	com.ibm.tools.attach.ja...	Stopped	00:33:956	
Thread-5	java.util.Timer\$TimerImpl	Stopped	00:00:026	00:28:050
derby.rawStoreDae	java.lang.Thread	Stopped	00:00:072	00:25:185
derby.antiGC	java.lang.Thread	Stopped	00:00:006	00:24:010

At the bottom of the window, there are three tabs: **Thread Statistics** (highlighted with a red box), **Monitor Statistics**, and **Threads Visualizer**.

Figure 27-21 Thread Analysis: Thread Statistics

- The Monitor Statistics tab shows the details of a selected thread, including the Java classes that are involved.
- The Thread Visualizer tab shows the threads that were active and the time when they were active (Figure 27-22). We can see two threads for the Derby database and five threads for the application code.

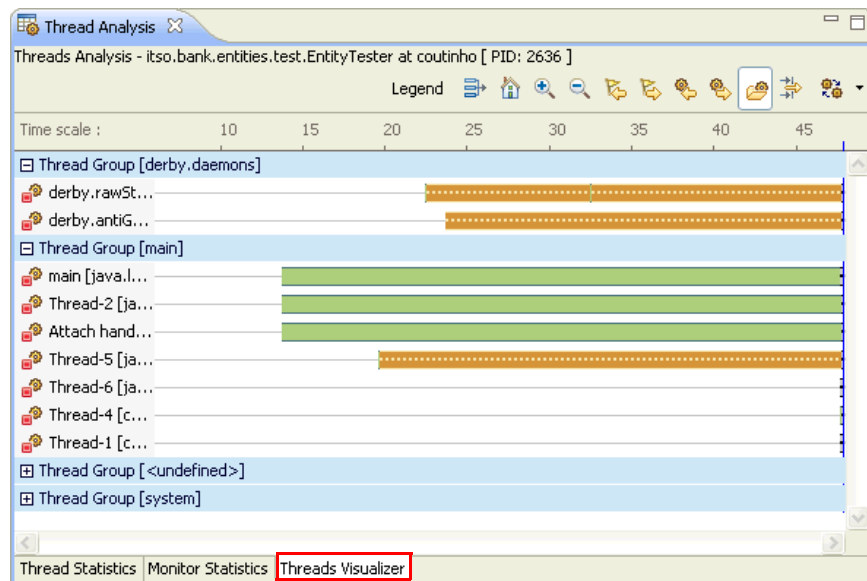


Figure 27-22 Thread Analysis: Threads Visualizer

## 27.3.10 Reports

For several statistics, you can create a report in comma-separated values (CSV), HTML, or XML format by clicking the **New Report** icon ()

## 27.3.11 Cleanup

You can remove measurements by right-clicking an entry in the Profiling Monitor view and selecting **Delete**.

# 27.4 Profiling a web application running on the server

In this section, we run a web application in WebSphere Application Server v8.0 Beta in profiling mode.

## 27.4.1 Importing the sample project archive file

**Tip:** If you have the ITSO RedBank web application (RAD8EJBEAR and dependent projects) already in the workspace, skip this step.

To import the ITSO RedBank web application project, follow these steps:

1. Open the Java EE perspective.
2. Select **File** → **Import**.
3. In the Import window, select **General** → **Existing Projects into Workspace** and click **Next**.
4. In the Import Projects window, click **Browse** and locate the **c:\7835code\** folder. Select the **RAD8JPA**, **RAD8EJB**, **RAD8EJBWeb**, and **RAD8EJBWebEAR** projects, and click **Finish**.

## 27.4.2 Setting up environment variables to profile a server

In specific cases, you must set up environment variables on the target host computer before starting the server in profile mode. The environment variables must be set before you start the server for profiling.

For example, if you connect to a remote WebSphere Application Server by using a version prior to Rational Agent Controller V8.3, you need to manually set up environment variables.

For more information about when to set up environment variables for profiling, refer to this website:

<http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/com.ibm.servertools.doc/topics/rprofilingenvar.html>

### 27.4.3 Publishing and running the sample application

Start WebSphere Application Server in profile mode and then publish the sample application to the server. You can pause the monitoring from the Profiling Monitor view to prevent data from being collected while the application is being published or the server is being configured.

**Important:** If you have not already configured the data source settings for the ITSOBANK database, follow the steps in “Configuring the data source for the ITSOBANK” on page 609, before publishing and running the sample application.

### 27.4.4 Starting the server in profiling mode

To profile the sample application on WebSphere Application Server v8.0 Beta in profiling mode, follow the steps:

1. Right-click over **RAD8EJBWeb** and select **Profile As** → **Profile on server**.
2. On the Profile on Server dialog window, select the **WebSphere Application Server v8.0 Beta at localhost** server and click **Finish**.
3. Another dialog is displayed, allowing you to select the type of profiling data collector that is going to be used in the profiling session. Select **Java Profiling** and click **Next**.
4. After the Profile on Server window opens (Figure 27-23 on page 1449), select **Execution Time Analysis**. Click **Edit Options**.

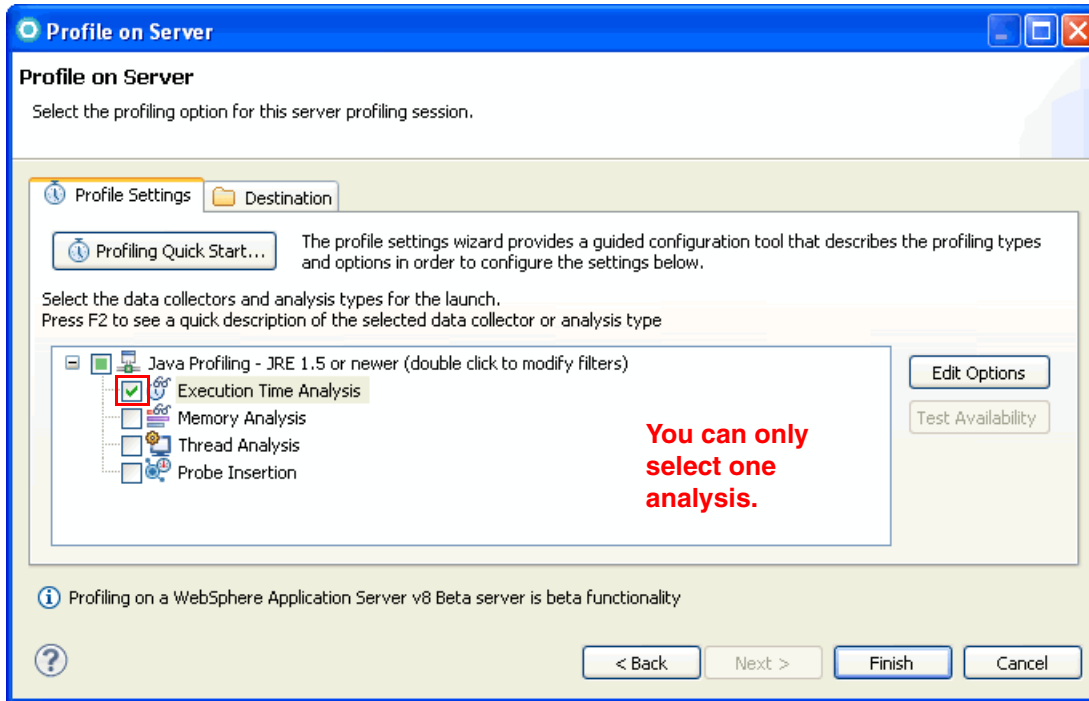


Figure 27-23 Profile on Server window

5. In the Edit Profiling Options window (Figure 27-8 on page 1434), select **Collect method CPU time information** and **Show execution flow graphical details**. Click **Finish**.
6. When prompted, switch to the **Profiling and Logging** perspective.

When the server is started, the Profiling Monitor shows the active monitor (Figure 27-24).

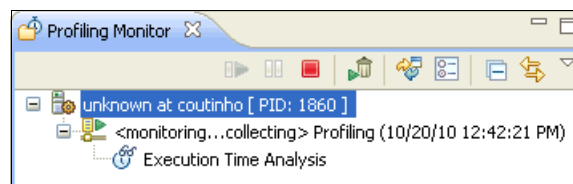


Figure 27-24 Profiling Monitor


## 27.4.5 Running the sample application to collect profiling data

**Published project:** Before you can complete this task, you must have published the project to the server, as explained in 27.4.3, “Publishing and running the sample application” on page 1448.

To collect data, display the accounts of one customer, run a debit and a credit transaction, list the transactions, and update the customer name, perform these steps:

1. In the Java EE perspective, right-click **RAD8EJBWeb** and select **Profile As** → **Profile on Server**. Click **Finish** to publish the application.
2. On the RedBank home page, click **RedBank**. If the heading and footing are not shown, right-click **redbank.jsp** and select **Profile on Server**.
3. On the Login page, in the Customer SSN field, enter 333-33-3333 and click **Submit**.
4. Select the last account.
5. On the Account Details page, select **Withdraw** and withdraw an amount of USD 25.
6. On the Account Details page, select **Deposit** and deposit USD 33.
7. On the Account Details page, select **List Transactions**.
8. On the List Transactions page, click **Account Details**.
9. On the Account Details page, click **Customer Details**.
10. On the Customer page, change the last name and click **Update**.
11. On the Customer page, click **Logout**.


### Pause monitoring

In the Profiling Monitor view, pause the monitoring by selecting the profile agent, and clicking the **Pause Monitoring** icon (). You can also keep the profiling running while you analyze the accumulated data.

## 27.4.6 Statistics views

The statistics views are the same as the statistics views for profiling the Java application.

## 27.4.7 Execution statistics

To analyze the execution statistics, in the Profiling Monitor view, select the <attached> **Profiling** process and click the  icon. Or, select **Open With** → **Execution Statistics** (Figure 27-25).

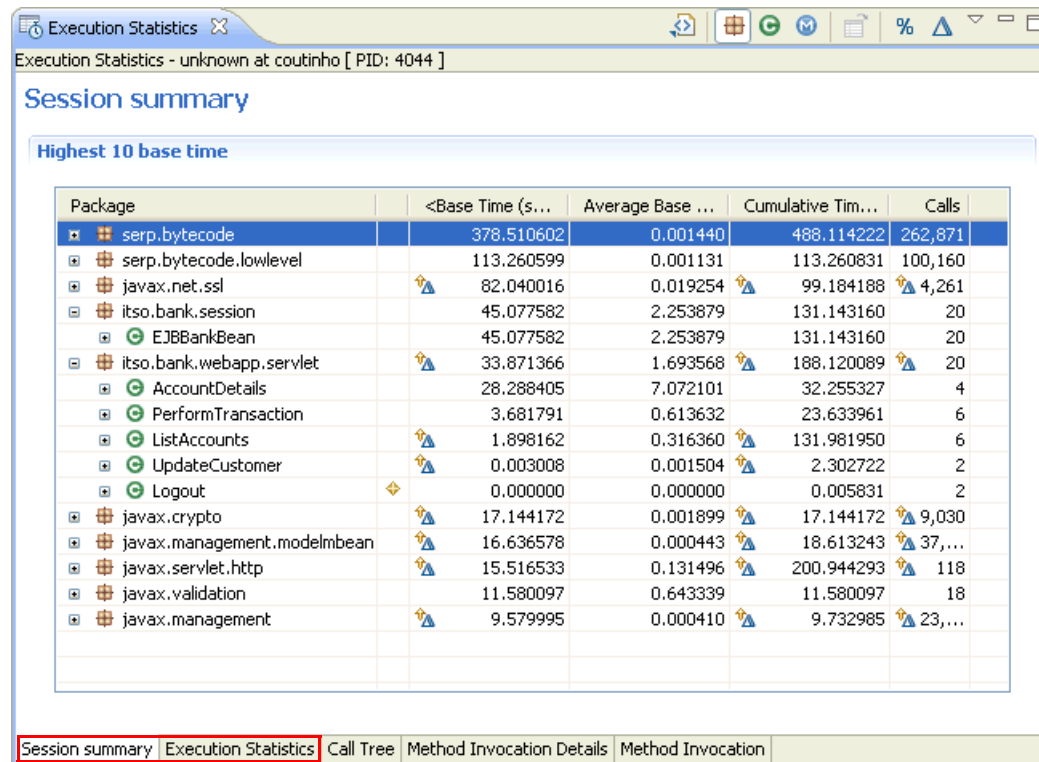


Figure 27-25 Execution Statistics

You can expand a class to see the statistics for each method (Figure 27-26).

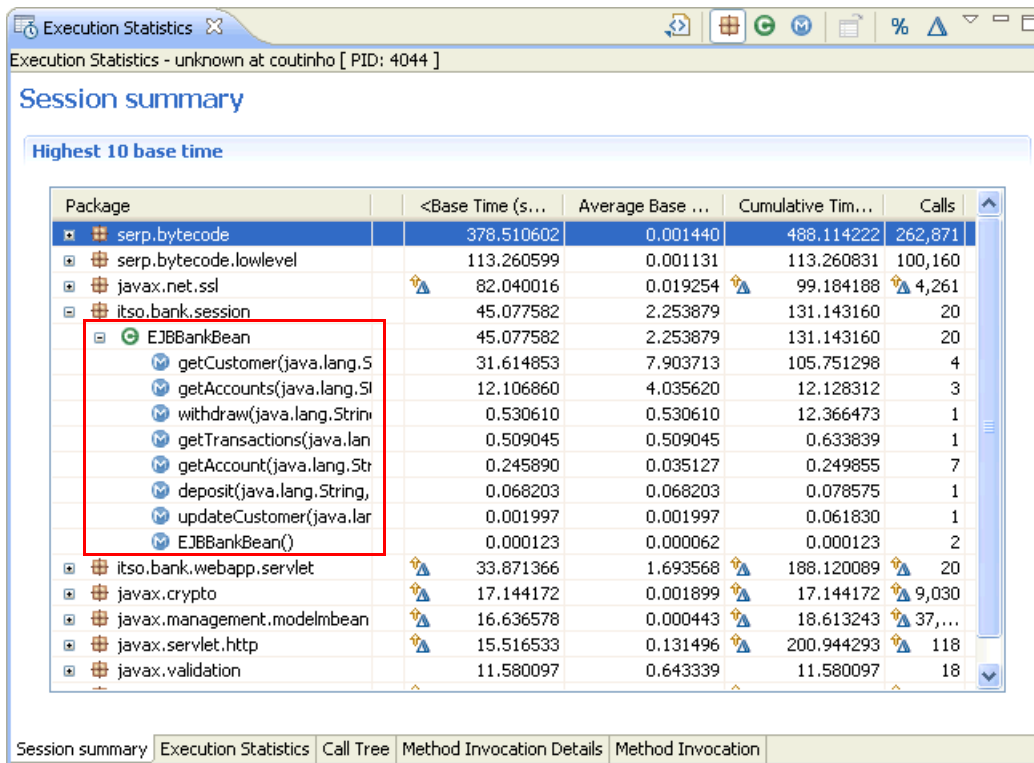



Figure 27-26 Execution Statistics by methods of a class

## Method invocation details

Expand **itso.bank.entities** → **Account** to see the methods of the Account class.

Right-click **processTransaction** and select the **Show Method Invocation Details** icon (.



**Tip:** The Session summary tab only displays the highest 10 base time packages. It is possible that the `itso.bank.entities` package is not shown in the Session summary tab page. To display all packages, change to the **Execution Statistics** tab. Be sure that the configured filters for the Execution Statistics, if any, do not hide the classes for which you are looking. The filter information shows at the top of the tab, as shown in Figure 27-27.

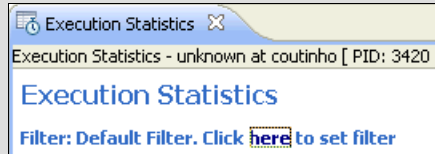


Figure 27-27 Filters in Execution Statistics

The Method Invocation Details view (Figure 27-28 on page 1454) provides statistical data on a selected method. The following data is displayed for the selected method:

- ▶ *Selected method* (`Account.processTransaction`) shows the details, including the number of times that the selected method is called, the class and package information, and the time that is taken by this method.
- ▶ *Selected method invoked by* shows the details of each method that calls the selected method, including the number of calls to the selected method, and the number of times that the selected method is invoked by the caller. In our case, the `deposit` and `withdraw` methods of the `EJBBankBean` class invoke the selected method.
- ▶ *Selected method invokes* shows the details of each method that is invoked by the selected method, for example, `Credit` and `Debit` constructors, `setAccount` of `Transaction` class, and internal JPA methods.

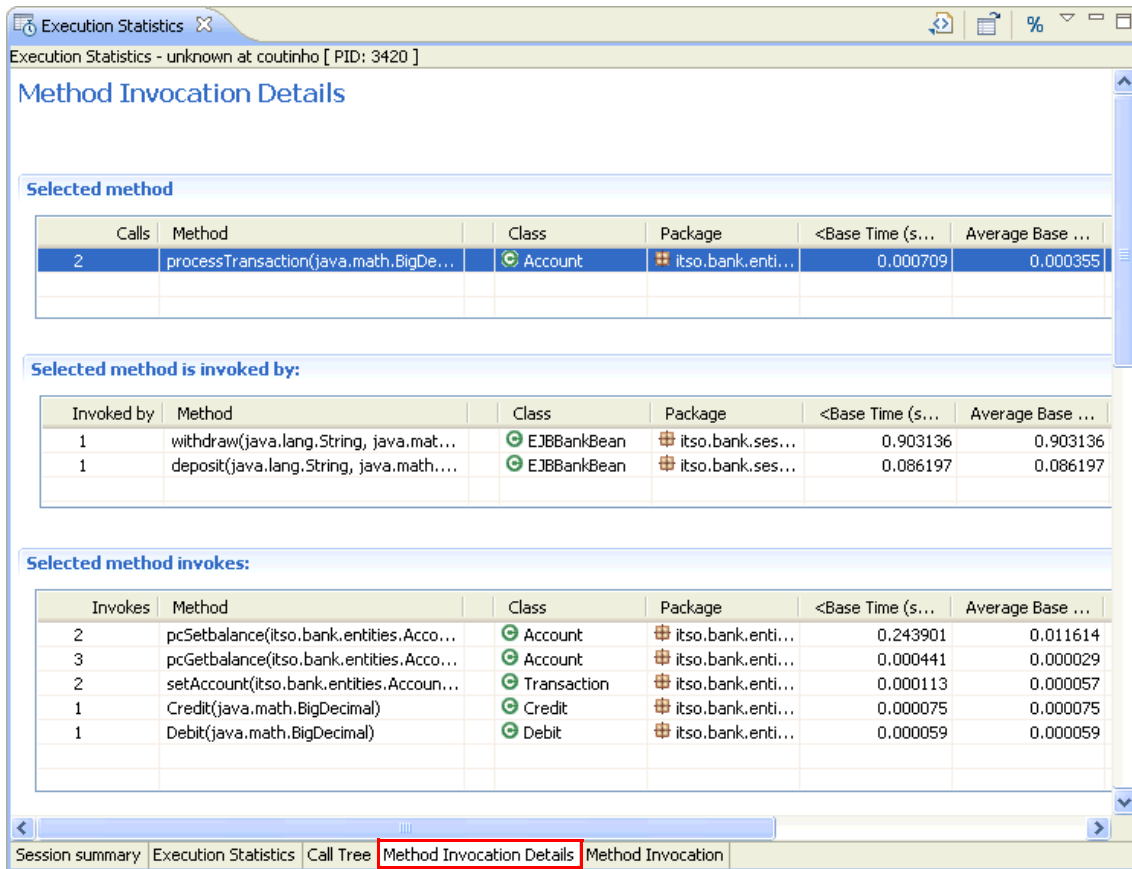



Figure 27-28 Method Invocation Details

## 27.4.8 Execution flow

The Execution Flow view and table both show a representation of the entire program execution. In this view, the threads of the program fit horizontally, and time is scaled so that the entire execution fits vertically. In the table, the threads are grouped in the first column, and the time is recorded in successive rows.

In the Profiling Monitor view, select the **<attached> J2EE Request Profile** process and click the  icon or select **Open With → Execution Flow**.

The Execution Flow (Figure 27-29 on page 1455) shows the following information:

- ▶ The bottom pane shows the action sequence. Expand the Web container and select the first **doPost** method.

- ▶ The top pane shows the execution stripes. Select the **Zoom In** icon (🔍) and click the **Web Container** column, near the start time of the method, until you see the graphic diagram.

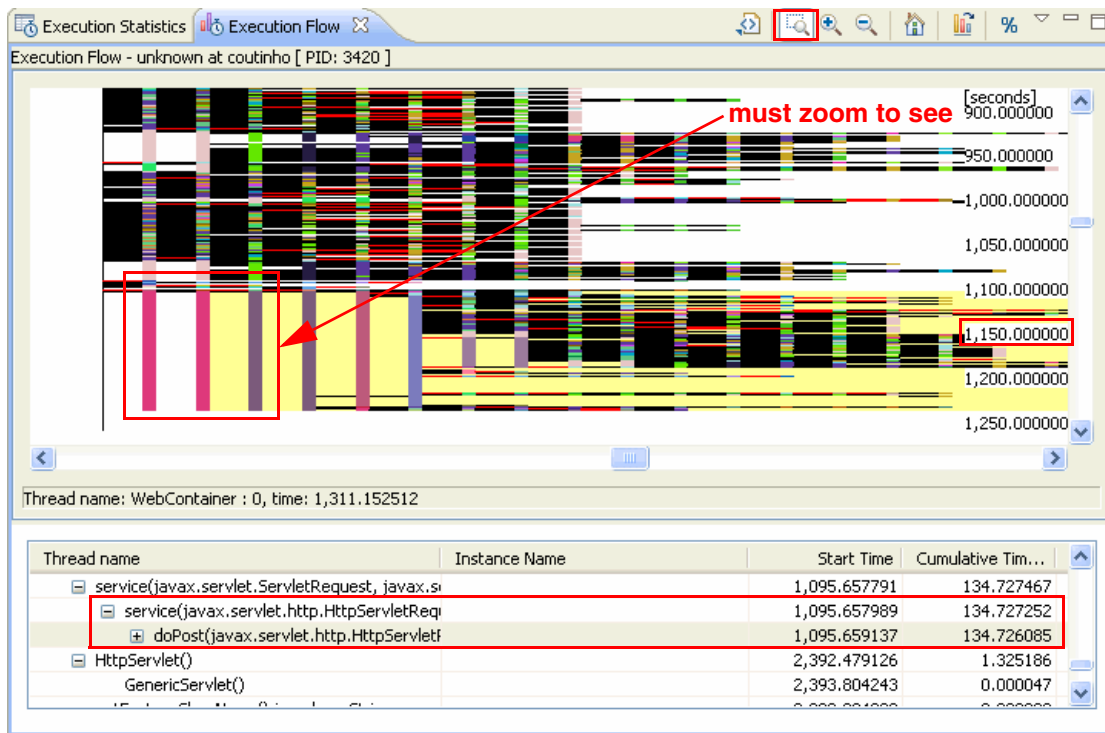


Figure 27-29 Execution Flow

Zoom into the top pane using the **Zoom In** icon to see the methods and their times (Figure 27-30 on page 1456).



Figure 27-30 Zooming in on Execution Flow

## 27.4.9 UML sequence diagrams

To display an UML2 interaction diagram, in the Profiling Monitor view, right-click the **<attached> J2EE Request Profiler** process and select **Open With** → **UML2 Class interactions** (Figure 27-31 on page 1457). You have to scroll to find suitable interactions.



**Tip:** The UML2 Class Interaction diagram contains all classes that are running on the server. It is easier to find the classes that you want to profile by using a filter. In the UML diagram in Figure 27-31 on page 1457, a filter was applied to only show certain classes. In order to create this filter, follow the steps:

1. Click the filter icon and click **Manage filters**.
2. In the Filters dialog window, click **New**.
3. Go to the **Advanced** tab and add the classes that you want to have filtered, as shown in Figure 27-32.

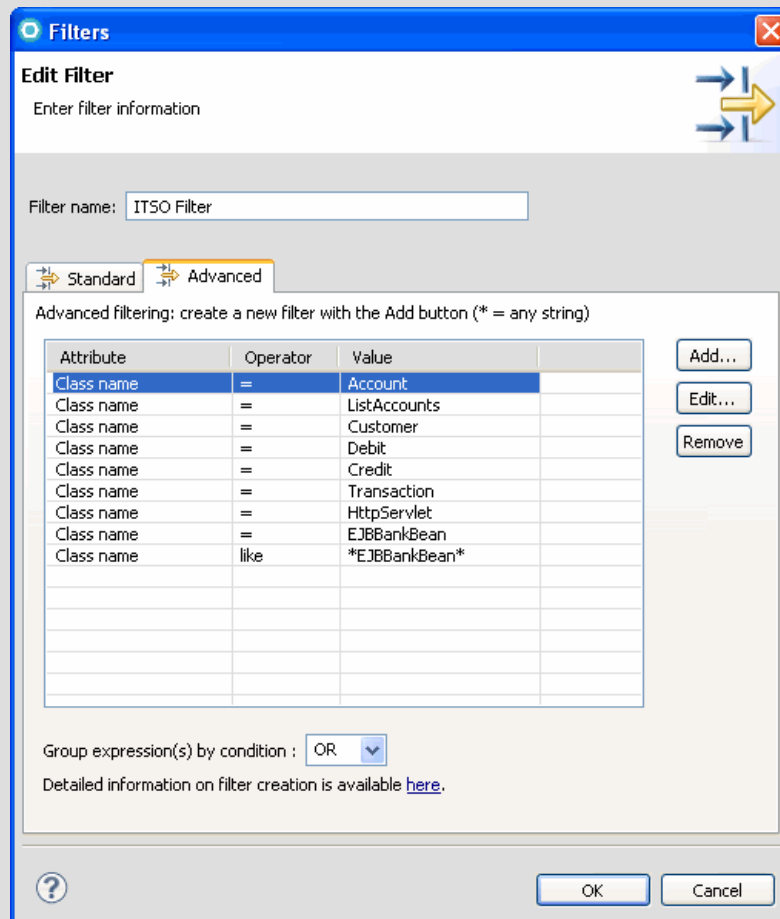


Figure 27-32 Advanced filtering configuration


4. Click **OK**. The UML2 Class Interaction diagram is filtered to show only the classes that you have selected in the filter.

You can drill down to a lifeline in which you can view all the trace interactions within a particular lifeline (right-click the class and select **Drill down into selected lifeline**). This feature helps to trace the root cause of a problem from a host, to a process, to a thread, and eventually to a class or an object.


You can highlight a call stack to view all the methods' invocations in a call stack. Right-click a method and select **Highlight call stack**.

### 27.4.10 Refreshing the views and resetting data

You can keep the profiling running while you analyze the views. Occasionally, you might want to refresh the views with the latest data, or reset the data:

1. Click the **Refresh Views** icon () to refresh the views.
2. Right-click the profiling process and select **Reset Data** to start a new collection of data and subsequent analysis.

### 27.4.11 Ending the profiling session

There are a number of methods to end the profiling session: detach the agent, restart the server in regular mode, or stop the server. To detach the agent, right-click the monitor and select **Detach from Agent**. This option ends profiling without stopping the entire server process. To terminate the entire server process, click the **Terminate** icon (). To remove the profiling agent, right-click the agent and select **Delete**. You are prompted if you want to delete the data in the file system. The connection to the server is removed.

### 27.4.12 Profile on server: Memory and thread analysis

To profile the application on the server for memory or thread analysis, stop the server, change the profiling options, and restart the server:

1. Stop the server.
2. Start the server in profiling mode.
3. In the Profile on Server window (Figure 27-23 on page 1449), complete the following steps:
  - a. Either select **Memory Analysis** and click **Edit Options**. Or, select **Thread Analysis** and click **Edit Options**.
  - b. In the Profile on server window, click **Finish**.
4. In the Confirm Perspective switch window, click **Yes**.

The Profiling and Logging perspective appears, and the agent is displayed in the Profiling Monitor view.

### **Running the sample application**

In the Java EE perspective, right-click **redbank.jsp** and select **Profile As** → **Profile on Server**. You do not have to republish the application. Run the sample sequence of operations, as explained in 27.4.5, “Running the sample application to collect profiling data” on page 1450.

### **Displaying memory and thread analysis**

Open the appropriate views from the Profiling Monitor view to perform the analysis.

## **27.5 More information**

For more information about profiling, see the Rational Application Developer Online Help sections:

- ▶ Select **Developing** → **Profiling, analyzing, and optimizing applications**
- ▶ Select **Developing** → **Testing and publishing on a server** → **Managing servers** → **Starting a server** → **Starting a server in profiling mode**.

For more information about IBM Rational Agent Controller, see these websites:

- ▶ Available releases of IBM Rational Agent Controller  
<http://www.ibm.com/support/docview.wss?uid=swg27013420>
- ▶ Download IBM Rational Agent Controller V8.3.1  
<http://www.ibm.com/support/docview.wss?uid=swg24028323>





## Debugging local and remote applications

Using Rational Application Developer, you can debug a wide range of applications in several languages, running either on local test environments (including local web applications) or on remote servers, such as WebSphere Application Server or WebSphere Portal.

In this chapter, we describe the major debugging features and provide two examples of debugger usage. We introduce the new debug tooling features, such as the ability to transfer Java-based debug sessions between Rational Team Concert team members, in Rational Application Developer. We also provide two examples of debug session transfer.

The chapter is organized into the following sections:

- ▶ Introducing Rational Application Developer new features
- ▶ Reviewing Rational Application Developer debugging tools
- ▶ Debugging a web application on a local server
- ▶ Debugging a web application on a remote server
- ▶ Using the Jython debugger
- ▶ Using the JavaScript debugger

- ▶ Using the debug extension for the Rational Team Concert client (Team Debug)
- ▶ Obtaining more information

## 28.1 Introducing Rational Application Developer new features

The debugging facilities that are available with Rational Application Developer v8.0 Beta are similar to the debugging features that were available in the previous version. The following features are the major new features:

- ▶ Service Component Architecture (SCA) debugger
- ▶ Extensible Stylesheet Language Transformation (XSLT) 2.0 debugger
- ▶ Java tracepoint
- ▶ Client-side JavaScript debugging support via Firebug integration

## 28.2 Reviewing Rational Application Developer debugging tools

In this section, we provide an overview of the major debug tooling features that are included in Rational Application Developer.

We cover the following topics:

- ▶ Supported languages and environments
- ▶ Java debugging features
- ▶ XSLT debugging
- ▶ Stored procedure debugging for DB2 V9
- ▶ Service Component Architecture debugger
- ▶ Java tracepoints
- ▶ Collaborative debugging using Rational Team Concert client

### 28.2.1 Supported languages and environments

Rational Application Developer includes support for debugging the following languages and environments:

- ▶ DB2 stored procedures (either in Java or SQL)
- ▶ Java
- ▶ Client-side JavaScript firebugs extension
- ▶ Jython scripts for WebSphere Application Server administration
- ▶ Mixed language applications (for example, Extensible Stylesheet Language Transformations (XSLT) called from Java)
- ▶ SQLJ

- ▶ WebSphere Application Server (servlets, JavaServer Pages (JSP), Enterprise JavaBeans (EJB), and web services)
- ▶ WebSphere Portal (portlets)
- ▶ XSL Transformation (XSLT) V1.0 and V2.0
- ▶ Service Component Architecture

You can debug applications in all these languages and environments within Rational Application Developer. These languages use a similar process of setting breakpoints, running the application in debug mode, and within the Debug perspective, stepping through the code to track variables and logic to find and fix problems.

Furthermore, the interface for debugging within the Debug perspective is consistent across all these languages and environments.

## 28.2.2 Java debugging features

In this section, we provide a brief description of the major debugging features that are available within Rational Application Developer for Java applications and explain how they typically might be used. Although this description focuses primarily on the tools available for Java, most of the features are available when debugging other languages.

### Views within the Debug perspective

When you run an application in debug mode and reach a breakpoint, you are prompted to switch to the Debug perspective. Use the Debug perspective. Although you can debug in any perspective, the Debug perspective includes the most helpful views for debugging.

By default, when debugging Java, the Debug perspective has the following views:

<b>Breakpoints view</b>	Shows all breakpoints in the current workspace and provides a facility to activate and deactivate them, remove them, change their properties, and to import or export a set of them to other developers.
<b>Console view</b>	Shows the output to <code>System.out</code> .
<b>Debug view</b>	Shows a list of all active threads and a stack trace of the thread that is currently being debugged.
<b>Display view</b>	Allows the user to execute any Java command or evaluate an expression in the context of the current stack frame.
<b>Error Log</b>	Shows all errors and warnings that are generated by plug-ins running in the workspace.

<b>Expressions view</b>	During debugging, the user has the option to inspect or display the value of expressions from the code or even evaluate new expressions. The Expressions view contains a list of expressions and values that the user has evaluated and then selected to track.
<b>Outline view</b>	Contains a list of variables and methods for the code listing that is shown in the Display view.
<b>Servers view</b>	Is useful if the user wants to start or stop test servers while debugging.
<b>Source view</b>	Shows the file of the source code that is being debugged, highlighting the current line that is being executed.
<b>Tasks view</b>	Shows any outstanding source code errors, warnings, or informational messages for the current workspace.
<b>Variables view</b>	Given the selected source code file that is shown in the Debug view, this view shows all the variables that are available to that class and their values. The Variables view is, by default, structured into columns. The use of columns can be toggled by using the <b>Layout</b> → <b>Show Columns</b> menu in the Variables view. Also, step-by-step debugging variables that change values are highlighted in a separate color.

Figure 28-1 on page 1466 shows an application that is stopped at a breakpoint in the Debug perspective.

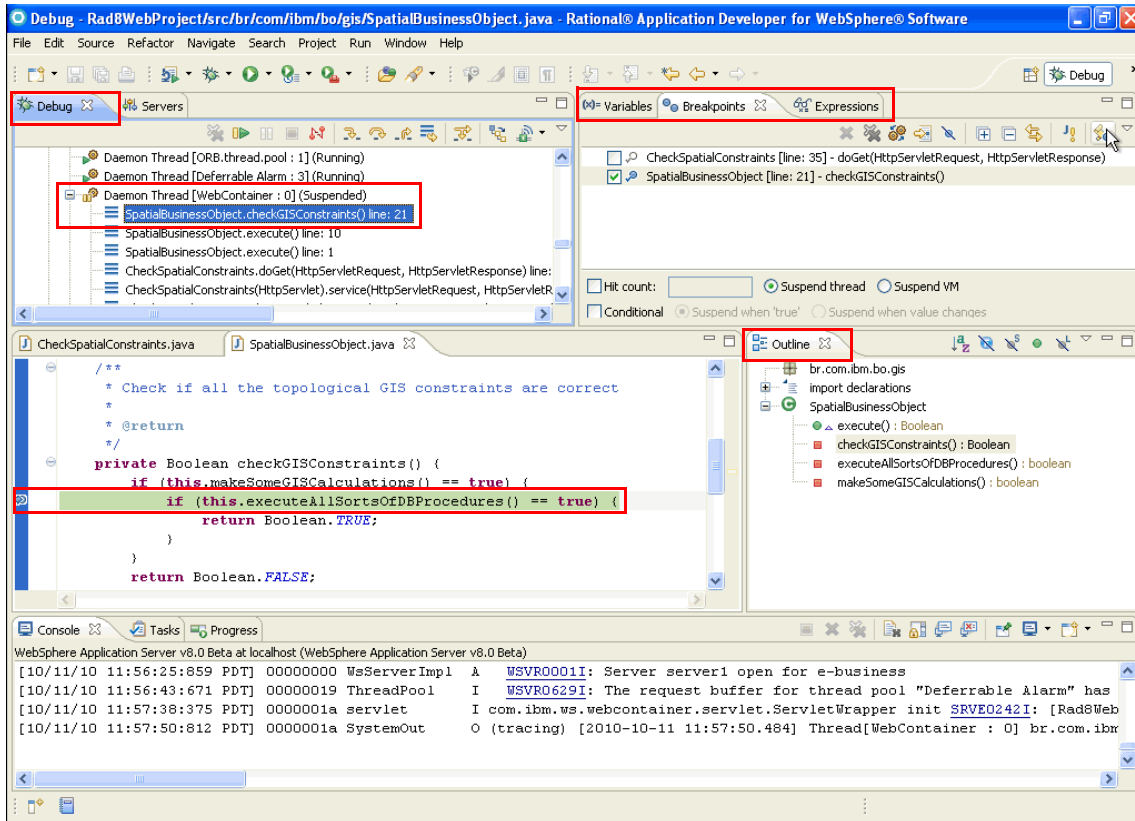
















Figure 28-1 Typical application running in the Debug perspective

## Debug functions

From the Debug view, you can use the functions that are available from the icon bar to control the execution of the application. The following icons are available:

-  **Resume (F8)**      Runs the application to the next breakpoint.
-  **Suspend**              Suspends a running thread.
-  **Terminate**            Terminates a process.
-  **Disconnect**          Disconnects from the target when debugging remotely.
-  **Remove All Terminated Launches**  
Removes terminated executions from the Debug view.
-  **Step Into (F5)**      Steps into the highlighted statement.
-  **Step Over (F6)**     Steps over the highlighted statement.
-  **Step Return (F7)** Steps out of the current method.

-  **Drop to Frame** Provides the facility to reverse back to the calling method in the current stack frame.
-  **Use Step Filters/Step Debug** (Shift+F5)  
Enables or disables the filtering for the *Step Debug* functions.
-  **Step-By-Step Mode**  
Toggles the step-by-step debug feature when it is enabled in the Run or Debug preferences.
-  **Show Qualified Names** (from the drop-down menu)  
Toggles to show the full package name.
-  or  **Debug UI demon**  
Provides a drop-down list for controlling the debug user interface (UI) daemon's listening state and port number.

## The Show Running Threads filter

When debugging, often many extra threads are shown in the Debug view that are not useful for finding a fault in an application under development. This situation is especially true when debugging web applications, where the application server starts several threads that are unlikely to be the cause of an application problem. To show only the threads that are suspended, open the context menu of the debug target that is being debugged in the Debug view, and toggle the **Show Running Threads** filter.


Using these buttons, menus, and the information that is shown in the various views that are available in the Debug perspective, you can debug most problems.

## Enabling and disabling Step Filter in the Debug view

The Debug view's toolbar contains a Use Step Filters command (icon). This command enables you to filter Java classes that do not have to be stepped into while debugging. For example, usually it is unnecessary for programmers to step into code from the classes within the Sun and IBM Java libraries when they are debugging step-by-step. By default, these classes are in the step filter list. You have the capability to add any class or package to this list.

To add a new Java package to the Step Filter feature in the Debug view, follow these steps:

1. Select **Window** → **Preferences**.
2. Expand **Run/Debug** → **Java and Mixed Language Debug** → **Step Filters**.
3. Click **Add Filter**.
4. Enter the new package or class that you want to filter out and click **OK**.


You can toggle the Step Filter or Step Debug feature on and off by clicking **Step Filter** () in the Debug view.

Prior to Version 7.0, the `java.*` and `javax.*` packages were not visible from the Step Filters preferences and were always filtered. Now these packages are displayed in the step filter list. You have the capability to deselect them. Therefore, when debugging, it is possible to step into classes from this package. The default setting is to filter these classes.

## Drop-to-Frame feature

With the Drop-to-Frame feature, you can go back to the calling method. This feature is available when debugging Java applications and web applications running on WebSphere Application Server. This feature is useful when you want to retest a block of code using other values.

For example, if a developer wants to test a method with the minimum and maximum permitted values for a given parameter, a breakpoint can be added at the end of the method. The Drop-to-Frame feature can be used to back up the control of the application to the start of the method, change the parameters, and run it again.

When running an application in the Debug perspective, the Debug view shows the stack frame (Figure 28-2). With the Drop-to-Frame feature, you can back up your application's execution to previous points in the call stack by selecting the desired method level from within the Debug view and then clicking **Drop To Frame** (). Clicking this icon moves the control of the application to the top of the method that is selected in the Debug view.

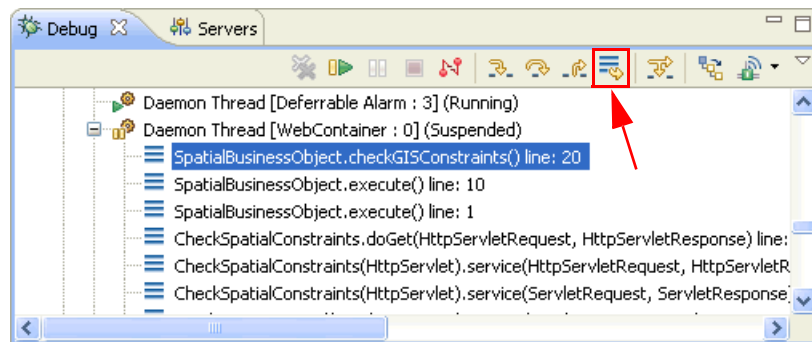


Figure 28-2 Drop-to-Frame icon in the Debug view



### 28.2.3 XSLT debugging

Extensible Stylesheet Language Transformation (XSLT) is a language for transforming XML documents into XHTML or other XML documents. It is a declarative language that is expressed in XML. It provides mechanisms to match tags from the source document to templates (similar to methods), store values in variables, perform simple looping or code branching, and invoke other templates to build up the result document.

Rational Application Developer supports debugging XSLT 2.0 in addition to XSLT 1.0. XSLT 2.0 adds new, powerful features to the languages:

- ▶ Support for XML schema data types
- ▶ Elimination of Result Tree Fragments (RTFs)
- ▶ Support for node grouping
- ▶ Aggregation functions
- ▶ For loops
- ▶ Multiple output documents
- ▶ XHTML output

The Rational Application Developer debugger includes support for the following XSLT 2.0-specific functionality:

- ▶ Supporting schema validation in the transformation
- ▶ Displaying the XPath 2.0 schema type of a variable
- ▶ Displaying sequences and their children
- ▶ Stepping into stylesheet functions
- ▶ Viewing the complete group information for `xsl:for-each-group` in the XSLT Context view

XSLT files can use templates that are stored in other files. XSLT files can become complicated. Rational Application Developer features a full-featured XSL transformation debugger. You can debug XSLT files using a set of tools that is similar to the Java tools.

To start a debugging session for an XSLT file and its source XML file, select both files on the Enterprise Explorer, right-click, and select **Debug As** → **XSL Transformation**.

For example, if you have read Chapter 8, “Developing XML applications” on page 331, select **Accounts.xml**, select **Accounts.xsl**, right-click, and select **Debug As** → **XSL Transformation**.

This action steps into the first line of the XSL file, and from that point, debugging can continue. From the Debug Launch Configuration window (from the context window, use **Debug As** → **Debug**), you can configure the transformation-related files, parameters, processor, and other settings to guide the XSLT debugging.



<b>Source view</b>	Shows the XSL file on the left side and the input XML file on the right side. The line that is executing for each file is highlighted.
<b>Expressions view</b>	Shows the value of XSL expressions, including XPath expressions in the current context.
<b>Variables view</b>	Shows currently visible XSLT variables.
<b>XSLT Context view</b>	Shows the current context of the XML input for the selected stack frame.
<b>XSL Transformation Output view</b>	Shows the serialized output of the transformation as it is produced.

It is also possible to debug an XSLT transformation that is called from a Java application. The easiest way to debug an XSLT transformation that is called from a Java application is to add a breakpoint in the Java code before the XSL transformation is called (typically from the `javax.com.transform.Transformer.transform` method). Then use the **Debug** → **Java And Mixed Language Application** option for the Java class from the context menu. When the debugger stops at the breakpoint, click the **Step Into** icon, and the debugger moves to debugging the XSL file and stops at the first line in the transformation.

It is also possible to debug Java code that is called from the XSL file. To debug Java code that is called from the XSL file, you must use the launch configuration window and ensure that the Class path tab includes the projects that contain the Java code to be debugged. When stepping through the XSLT debugger, it is possible to step into the call to the Java method.

## 28.2.4 Stored procedure debugging for DB2 V9

With stored procedure debugging, you can debug Java and DB2 stored procedures that are running on a local or remote DB2 server. The Debug Launch configuration editor provides fields to specify a stored procedure on a DB2 database to debug, arguments to pass to the procedure, and the associated source code. If the database server is configured correctly, the debugger starts, and the debugging can continue. Rational Application Developer has a detailed Help chapter on this feature:

<http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.debug.spd.doc/topics/cbsovrv.html>

## 28.2.5 Service Component Architecture debugger

Rational Application Developer provides the Service Component Architecture (SCA) debugger that allows you to analyze and diagnose errors in SCA applications. It allows you to step into SCA services via the SCA binding and the web services binding, use step-by-step mode to stop at every service, and examine SCA context variables.

The SCA debugger provides the same usability from general Java Platform, Enterprise Edition (Java EE) application debugging. The SCA debugger allows you to set breakpoints at the service entry point, launch the debug session on a WebSphere Application Server instance, and invoke the service from the Web Services Explorer.

For more information about the SCA debugger, consult this document:

<http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.debug.sca.doc/topics/cbcovrv.html>

## 28.2.6 Java tracepoints

The *Tracepoints* feature, which is new in Rational Application Developer, allows you to trace information at the breakpoints that you have set. When tracing is enabled for a breakpoint, it outputs the breakpoint information every time that the breakpoint is reached.

To enable a tracepoint in a breakpoint open the Breakpoints view, and right-click over any existing breakpoint. Select **Trace** → **Enable Tracing**. The tracing point is active in that breakpoint now, and a tracing string is appended to the breakpoint entry, as shown in Figure 28-4.



Figure 28-4 Tracepoint enabled

Every time that the particular breakpoint is reached, the tracepoint prints the breakpoint information in logs, as shown in Example 28-1

*Example 28-1 Tracepoint information shown in log file*

---

```
[10/12/10 11:00:17:734 PDT] 0000001e SystemOut 0 (tracing) [2010-10-12
11:00:17.625] Thread[WebContainer : 0] itso.bank.session.EJBBankBean
getCustomers (EJBBankBean.java:109)
```

---

You can customize the information that was logged by right-clicking the breakpoint entry and selecting **Trace** → **Edit Trace**.

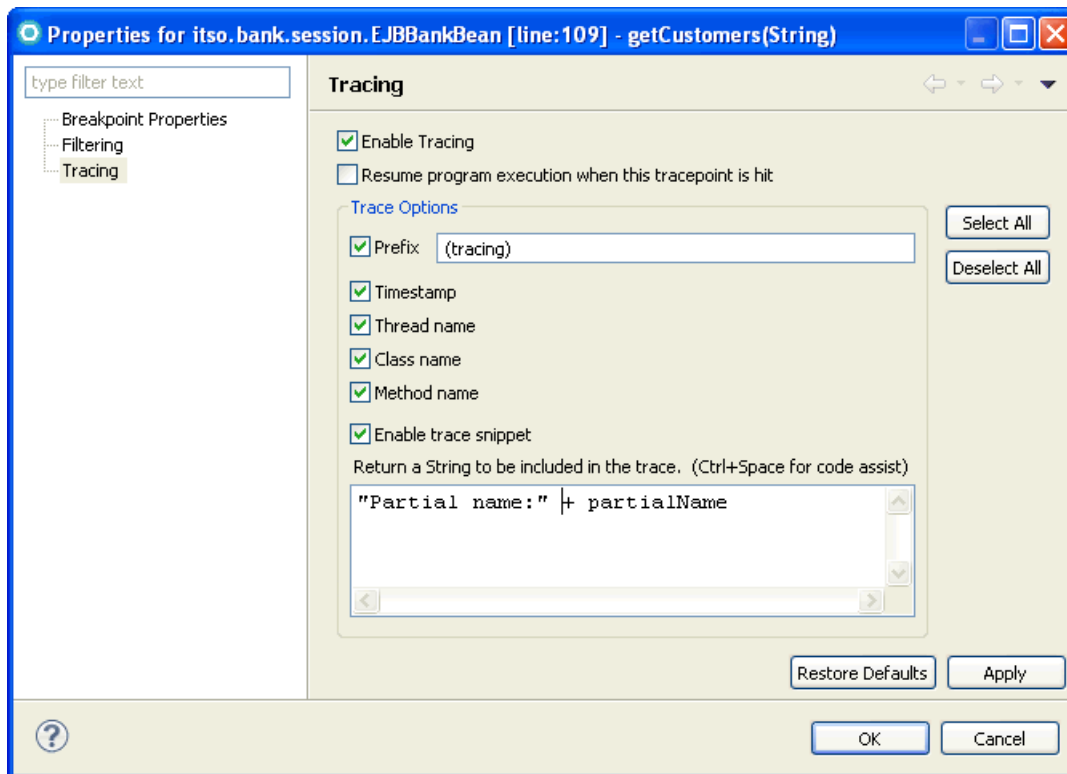


Figure 28-5 Tracepoint configuration

The dialog window that is shown in Figure 28-5 allows you to add or remove runtime information, such as the thread name or class name. Also, this dialog window allows you to create an expression, which contains variables from the breakpoint scope, to add to the tracing output.

For Java code breakpoints that are running in WebSphere Application Server, you can configure whether the tracing output is sent to the console or a file. To configure this option, select **Window** → **Preferences**. In the Preferences dialog window, select **Run/Debug** → **Tracepoint** → **WebSphere Application Server**. From that point, you can set where to write the trace output.

## 28.2.7 Collaborative debugging using Rational Team Concert client

The debug extension for Rational Team Concert Client (Team Debug) allows you to execute Java-based debug collaborative sessions between Rational Team

Concert team members. It is possible to send the session to a selected user or to park it in a team repository for later retrieval. It is even possible to share the session by dragging it to a chat window, provided that the messaging service is enabled.

You can extend Rational Application Developer to support the Team Debug by installing the Rational Team Concert Client as a shell share with Rational Application Developer and selecting the Collaborative debug extensions for the Rational Team Concert Client feature.

For a detailed description of using the Team Debug that is provided by Rational Team Concert Client, see 28.8, “Using the debug extension for the Rational Team Concert client (Team Debug)” on page 1516.

## 28.3 Debugging a web application on a local server

In this section, we take you through a web application scenario in which a sample application is run on the local Rational Application Developer test server. Then we use debugging facilities to step through the code and watch the behavior of the application.

The debug example includes the following tasks to demonstrate the debug tooling:

- ▶ Importing the sample application
- ▶ Running the sample application in debug mode
- ▶ Setting breakpoints in a Java class
- ▶ Watching variables
- ▶ Evaluating and watching expressions
- ▶ Working with breakpoints
- ▶ Setting breakpoints in JSP
- ▶ Debugging JSP

### 28.3.1 Importing the sample application

In the following task, we show how to set up your workspace for the sample application. We use the ITS0 RedBank web application sample that we developed in Chapter 12, “Developing Enterprise JavaBeans (EJB) applications” on page 577 to demonstrate the debug facilities.

If you already have the necessary projects (RAD8EJBWebEAR, RAD8EJBWeb, RAD8EJB, and RAD8JPA) in the workspace, you can skip this step.

To import the ITSO RedBank EJB web application, follow these steps:

1. In the Web perspective, select **File** → **Import** → **Other** → **Project Interchange** and click **Next**.
2. In the Import Projects window, click **Browse** to locate the **C:\7835codesolution\ejb\RAD8EJBWeb.zip** file.
3. Select the projects that are not in the workspace and click **Finish**.

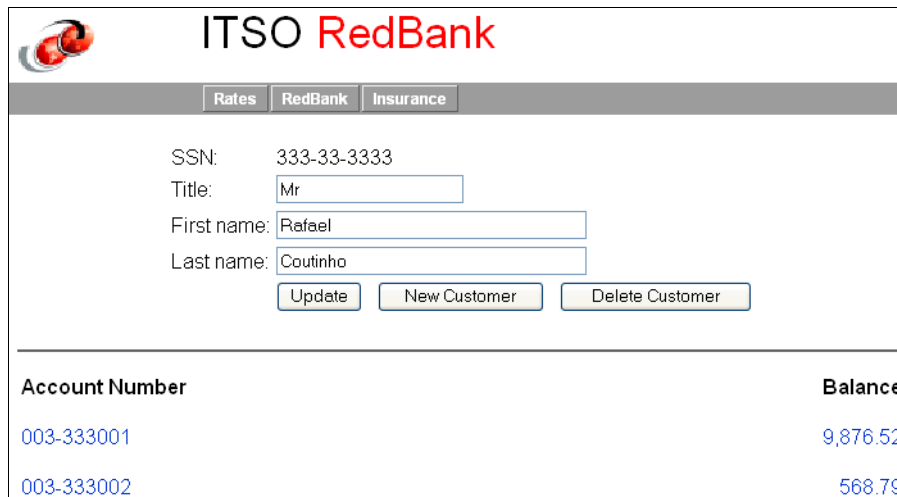
To run this application, you have to configure a database as explained in “Setting up the ITSOBANK database” on page 1880 and “Configuring the data source in WebSphere Application Server” on page 1882.

### 28.3.2 Running the sample application in debug mode

To verify that the sample application was imported properly, run the sample web application on the WebSphere Application Server v8.0 Beta test server in debug mode:

1. In the Enterprise Explorer, expand **RAD8EJBWeb** → **WebContent**.
2. Right-click **redbank.jsp** and select **Debug As** → **Debug on Server**.
3. If the Server Selection window opens, select **Choose an existing server** and select **WebSphere Application Server v8.0 Beta at localhost**. Then click **Finish** to start the server, publish the application to the server, and open a browser that shows the ITSO RedBank page.
4. If the server is already running in normal (non-debug) mode, when prompted to switch mode, click **OK**. The server restarts in debug mode.
5. When the page is displayed, in the Customer SSN field, enter 333-33-3333. Then click **Submit**.

The list of accounts for that customer shows (Figure 28-6). If you can see these results, the application is working as expected in debug mode.



The screenshot displays the ITSO RedBank application interface. At the top left is the ITSO RedBank logo. Below the logo is a navigation bar with three tabs: "Rates", "RedBank", and "Insurance". The "RedBank" tab is selected. The main content area shows customer details for a customer with SSN 333-33-3333. The title is "Mr", first name is "Rafael", and last name is "Coutinho". Below the form are three buttons: "Update", "New Customer", and "Delete Customer". At the bottom, there is a table with two columns: "Account Number" and "Balance".

Account Number	Balance
003-333001	9,876.52
003-333002	568.79

Figure 28-6 Customer details for the RedBank application

### 28.3.3 Setting breakpoints in a Java class

*Breakpoints* are indicators to the debugger to stop execution at that point in the code so that the user can inspect the current state and step through the code. Breakpoints can be set to always trigger when the execution point reaches them or when a certain condition has been met (conditional).

In the ITSO RedBank sample application, before the balance of an account is updated after the withdrawal of funds from an account, the new balance is compared to see if it goes under zero. If the account has adequate funds, the withdrawal is complete. If the account has insufficient funds, an `Exception` is thrown from the `Account` class, and the `showException.jsp` is displayed to the user showing an appropriate message.

In this example, we set a breakpoint where the logic tests that the amount to withdraw does not exceed the amount that exists in the account:

1. In the Enterprise Explorer, select and expand **RAD8JPA** → **src** → **itso.bank.entities** and open **Account.java** in the Java editor.
2. Locate the `processTransaction` method.



**Tip:** You can use the Outline view or expand `Account.java` in the Enterprise Explorer view to find the `processTransaction` method quickly in the source code.

3. Place the cursor in the gray bar (along the left edge of the editor area) on the following line of code in the `processTransaction` method:  
`if (balance.compareTo(amount) < 0)`
4. Double-click to set a breakpoint marker (highlighted in Figure 28-7).

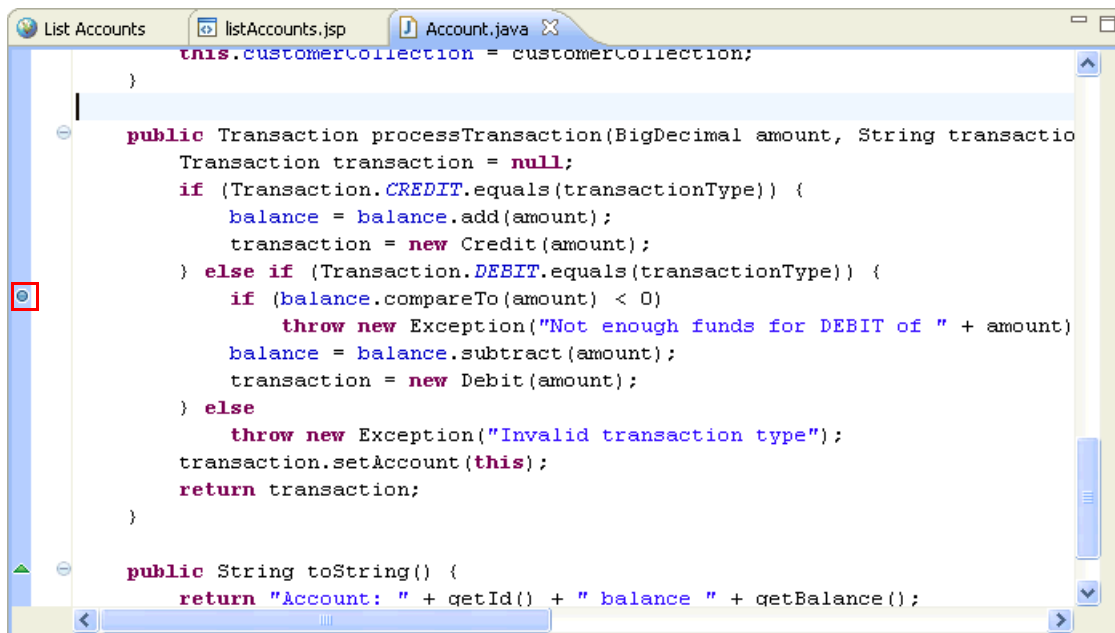


Figure 28-7 Setting a breakpoint in Java

**Enabled and installed breakpoints:** Enabled breakpoints are indicated with a blue circle. Installed breakpoints have an additional check mark overlay. A breakpoint can only be installed when the class in which the breakpoint is located has been loaded by the virtual machine (VM).

5. Right-click the breakpoint and select **Breakpoint Properties**.
6. In the Breakpoint Properties window (Figure 28-8 on page 1478), change the details of the breakpoint.

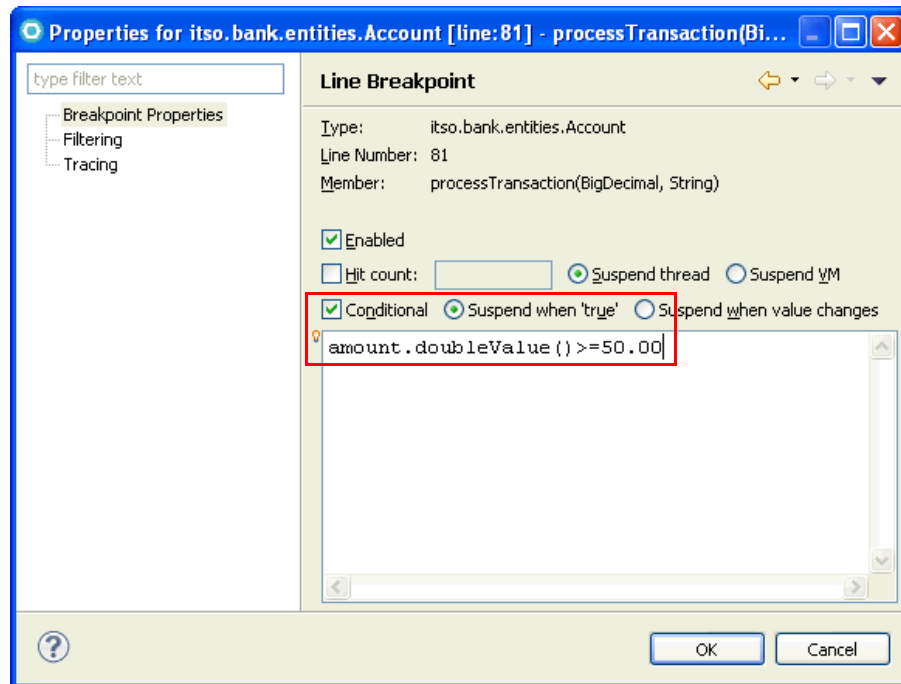


Figure 28-8 Breakpoint properties

Note the following properties in Figure 28-8:

- The *Hit Count* property, if set, causes the breakpoint to be triggered only when the line has been executed as many times as the hit count specified. After it is triggered, the breakpoint is disabled.
- The *Conditional* property allows breakpoints to trigger only when the condition that is specified in the entry field evaluates to true. This condition is a Java expression. You can use code assist (Ctrl+Spacebar) to see the fields and methods that you can use in this expression. When this condition is enabled, the breakpoint is marked with a question mark on the breakpoint, which indicates that it is a conditional breakpoint.

For example, select **Conditional**, enter the expression `amount.doubleValue() >= 50.00`, and select **condition is 'true'**. Now the breakpoint only triggers on transactions of USD 50 or more.

Click **OK** to close the breakpoint properties.

You can now run the application and trigger the breakpoint. *It is not necessary to restart the application server for the new breakpoint to work.*

To trigger the breakpoint, perform the following steps:

1. On the home page, click the **RedBank** tab. For the Customer number, enter 333-33-3333 and click **Submit**.
2. On the List Accounts page, click the first account (003-333001).
3. On the Account Details page, select **Withdraw**, and in the Amount field, enter 50.
4. Click **Submit**. The `processTransaction` method is executed, and the new breakpoint is triggered.

### 28.3.4 Using the Debug perspective

Depending on the preferences that are set in **Windows** → **Preferences** → **Run/Debug** → **Perspectives**, you might be prompted to open the Debug perspective. In most cases, the Debug perspective opens automatically.

If the Debug perspective does not open automatically, select **Window** → **Open Perspective** → **(Other)** → **Debug**.

The Debug perspective shows the source code where the execution stopped at the breakpoint (see Figure 28-1 on page 1466):

- ▶ The Debug view shows the threads and is currently stopped in `Account.processTransaction`.
- ▶ The source code of the `Account` class shows the current line (at the breakpoint).
- ▶ The Outline view shows the current method (`processTransaction`).
- ▶ The Variables view shows the account (`this`), the amount (`BigDecimal`), the transaction type (`Debit`), and the transaction (`null`).
- ▶ The Breakpoints view shows the breakpoint (`Account [line: 81]`).
- ▶ The Console view shows the server console. You might see timeout errors, because we stopped the execution in the middle of an EJB call.

In this perspective, it is possible to step through the code and to watch and edit variables.

## 28.3.5 Watching variables

The Variables view shows the current values of the variables in the selected stack frame (Figure 28-9):

- ▶ When you expand **this**, you can verify the value of `id`. You can also expand the **balance** and see the value (for example, 98765 with a scale 2 = 987.65).
- ▶ When you expand **amount**, although `amount` is of type `BigDecimal`, a string representation of its value is shown in the bottom pane of the window.

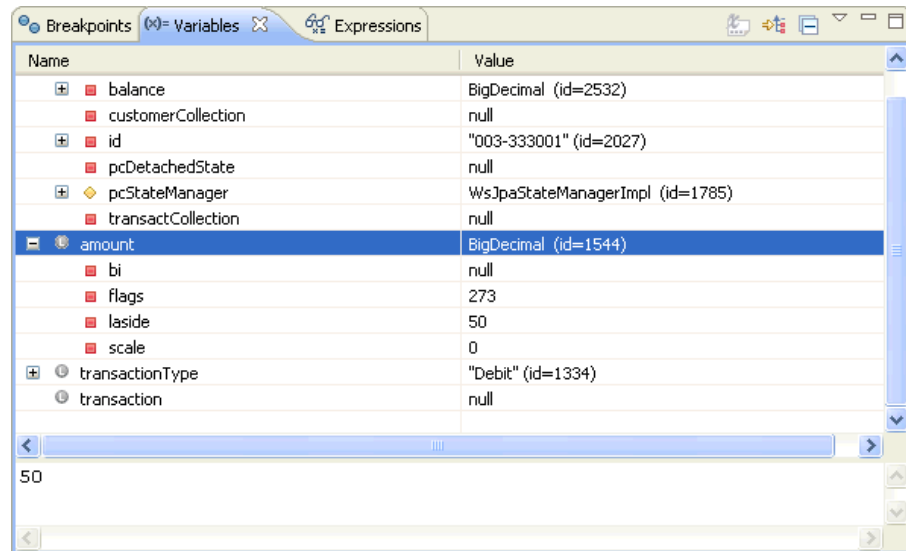



Figure 28-9 Displaying variables

The plus sign (+) next to a variable indicates that it is an object. To display an object's instance variables, click the plus sign.

Follow these steps to see how you can track the state of a variable, while debugging the method:

1. Click **Step Over**  in the Debug view (or press F6) to execute the current statement.
2. Click **Step Over** again, and the `balance` is updated. The color of the `balance` changes in the Variables view.

It is possible to test the code with another value for any of these instance variables. You can change a value by selecting **Change Value** from its context menu. A window opens where you can change the value. For objects, such as `BigDecimal`, you have to use a constructor to set the value.

For example, right-click **balance** and select **Change Value**. In the Change Object Value window, type new `java.math.BigDecimal(900.00)` and click **OK**. If you want to execute again with the new values, use Drop-to-Frame. The debugger will take you back to the beginning of the method processTransaction.

### 28.3.6 Evaluating and watching expressions

When debugging, often it is useful to evaluate an expression that is made up of several variables within the current application context.

To view the value of an expression within the code, perform these steps:

1. Select the expression (for example, `balance.compareTo(amount)` in the breakpoint line), right-click, and select **Inspect**. The result opens in a pop-up window showing the value (Figure 28-10).

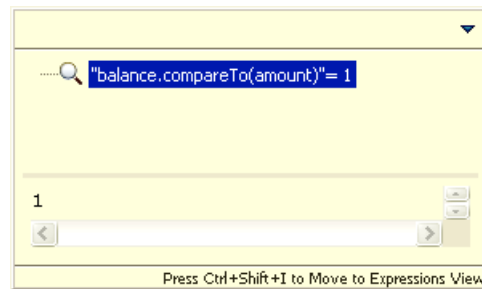


Figure 28-10 Inspect pop-up window

2. To move the results to the Expressions view (Figure 28-11) so that the value of the expression can continue to be monitored, press `Ctrl+Shift+I`.
3. To watch an expression, right-click in the Expressions view and select **Add Watch Expression**. In the Add Watch Expression window, enter an expression, such as `balance.doubleValue()`.

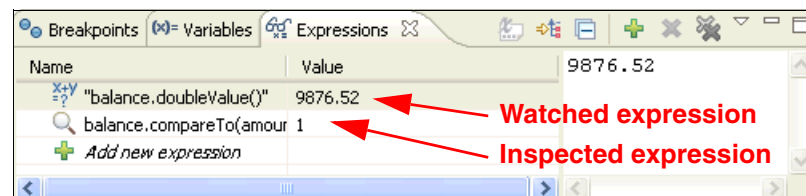




Figure 28-11 Inspecting a variable in the Expressions view

**Watched versus inspected expressions:** The Expressions view contains a list of watched expressions (marked by a  symbol) and a list of inspected expressions marked by a  symbol (Figure 28-11). The difference between these lists is that the value shown for watched expressions changes with the underlying value as the user steps through the code. An inspected expression keeps showing the value it held when it was first inspected.

### 28.3.7 Using the Display view

To evaluate an expression in the context of the currently suspended thread, which does not come from the source code, use the Display view:

1. Set a new breakpoint on the line:  
`transaction.setAccount(this);`
2. Click **Step Return** until you reach that line.
3. From the workbench, select **Windows** → **Show View** → **Debug** → **Display**.
4. In the Display view, type the expression `transaction.getTransTime()` and highlight the expression. Right-click and select **Display** (Figure 28-12).



Figure 28-12 Expression and evaluated result in Display view

**Tip:** When entering an expression in the Display view, you can use code assist (Ctrl+Spacebar).

Each expression is executed, and the result is displayed, as shown in Figure 28-12. This method is a useful way to evaluate Java expressions or even to call other methods during debugging, without having to make changes in your code and recompile.

You can also highlight any expression in the source code, right-click, and select **Watch** (or **Inspect**). The result is shown in the Expressions view.

Select **Remove** from the context menu to remove expressions or variables from the Expressions views. In the Display view, select the text, and delete it.

## 28.3.8 Working with breakpoints

To enable a breakpoint in the code, double-click in the gray area of the left frame (or use the context menu on the left side of the frame) for the line of code for which the breakpoint is required. To remove it, double-click it again. To disable the breakpoint, right-click and select **Disable Breakpoint**.

Alternatively, after you have created the breakpoints, you can enable and disable them from the Breakpoints view (Figure 28-13). If the breakpoint is cleared in the Breakpoints view, it is skipped during execution.

To disable or enable all breakpoints, click the **Skip All Breakpoints** icon, which is highlighted in Figure 28-13. If this option is selected, all breakpoints are skipped during execution.

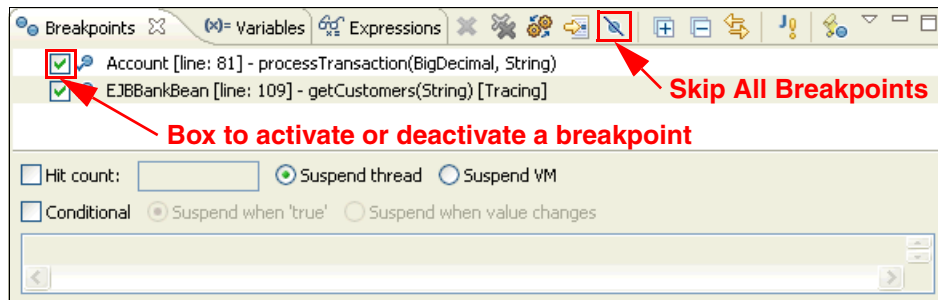




Figure 28-13 Enabling and disabling breakpoints

It is possible to export a set of breakpoints, including the conditions and hit count properties, so that the breakpoints can be shared across a development team. From the context menu of the Breakpoints view, select **Export Breakpoints**, and the breakpoints are saved as a bkpt file to the selected location.

Before continuing with debugging JSP, follow these steps:

1. Click the **Remove All Breakpoints** icon () to remove the breakpoints.
2. Click **Resume** () to continue execution.

You do not see the web page automatically in the Debug perspective. Click the view with the World icon (in the same pane as the source code) to see the resulting web page. Alternatively, switch to the Web perspective.

If you wait too long, the thread is terminated in the server, and you have to restart the application from the `index.jsp`.

**Tip:** Java exception breakpoints are separate from the breakpoints that are triggered when a particular exception is thrown. You can set Java exception breakpoints by selecting **Run** → **Add Java Exception Breakpoint**.

### 28.3.9 Setting breakpoints in JSP

You can also set breakpoints in JSP. Within the source view of a JSP page, you can set breakpoints inside JSP scriptlets, JSP directives, and lines that use JSP tag libraries. You cannot set breakpoints in lines with only HTML code.

In the following example, you set a breakpoint in the `listAccounts.jsp` at the point where the JSP shows a list of accounts for the customer:

1. In the Web perspective, expand **RAD8EJBWeb** → **WebContent** and open **listAccounts.jsp** in the editor. Select the **Source** tab.
2. Set a breakpoint by double-clicking in the gray area next to the desired line of code (Figure 28-14 on page 1485).

The Breakpoint properties are also available for JSP from the context menu. These properties share the same features as Java breakpoints with the exception that content assist is not available in the breakpoint condition field.



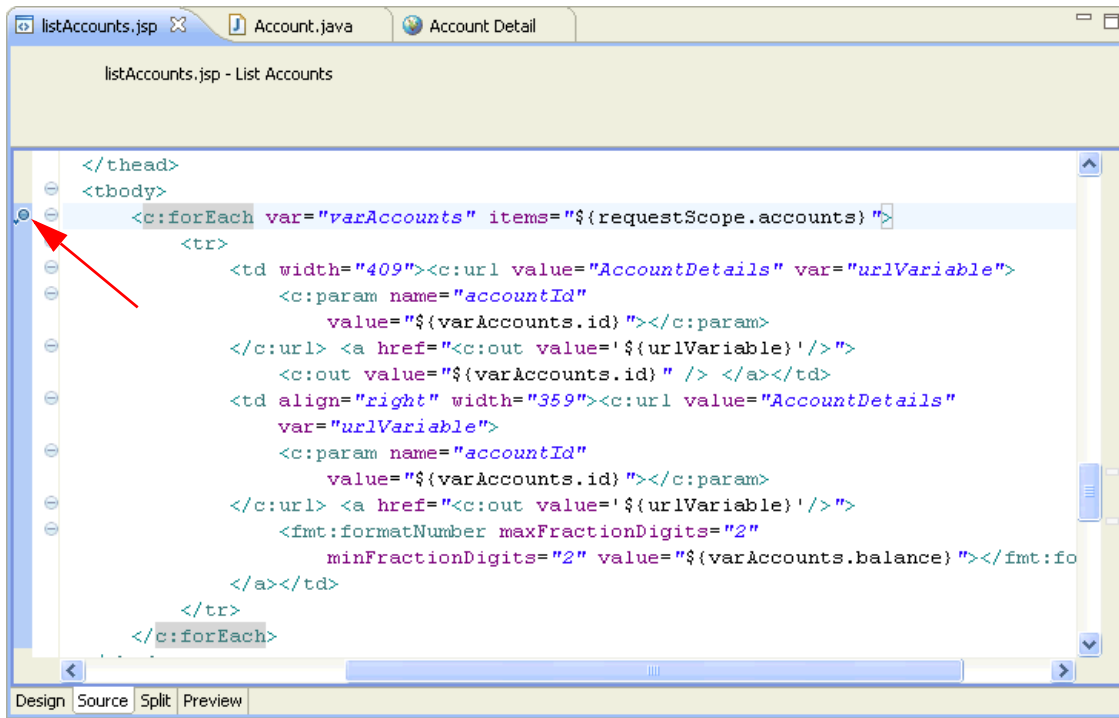


Figure 28-14 Adding a breakpoint to a JSP page

### 28.3.10 Debugging JSP

When a breakpoint is added, it is not necessary to redeploy the web application. To debug a JSP, follow these steps:

1. From the RedBank index page, select the **RedBank** tab.
2. On the RedBank page, for Customer ID, enter 333-33-3333 and click **Submit** to execute the `listAccounts.jsp` file and hit the new breakpoint.
3. In the Confirm Perspective Switch window, click **Yes** to switch to the Debug perspective.

Execution stops at the breakpoint that is set in the `listAccounts.jsp`, because clicking **Submit** in the application attempts to show the accounts by executing this JSP. The thread is suspended in debug, but other threads might still be running (Figure 28-15 on page 1486).

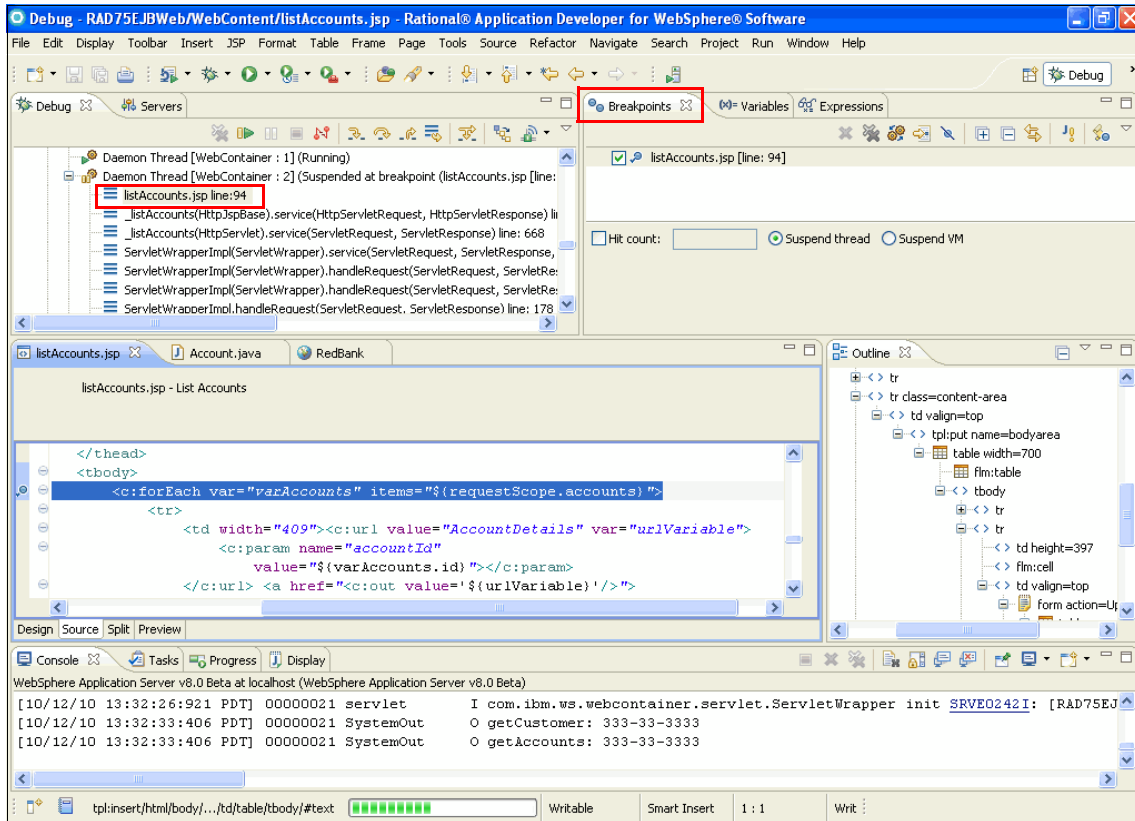


Figure 28-15 Debugging a JSP page

**Two JSP with the same name:** If you have two JSP with the same name in multiple web applications, the wrong JSP source might be displayed. Open the correct JSP to see its source code.

After a breakpoint is hit, you can analyze variables and step through lines of the JSP code. The same functions that are available for Java classes are available for JSP debugging. The difference is that the debugger shows the JSP source code and not the generated Java code.

The JSP variables are shown in the Variables view. The JSP implicit variables are also visible, and it is possible to look at, for example, request parameters or session data (Figure 28-16 on page 1487).

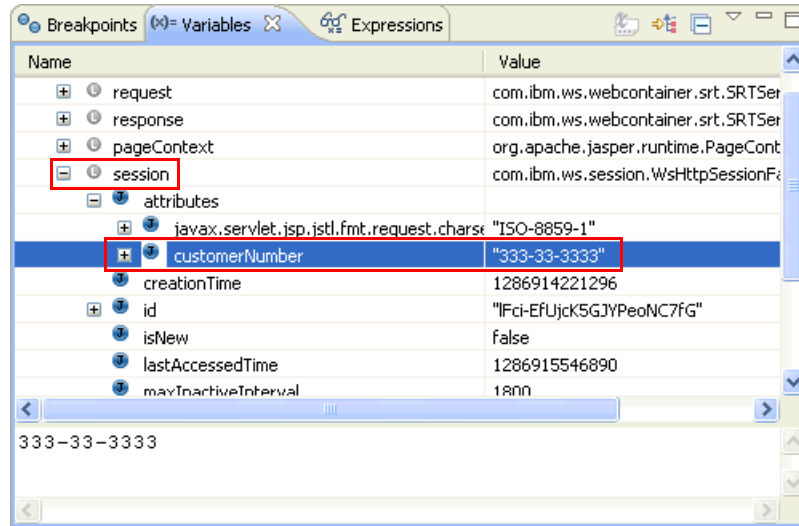


Figure 28-16 JSP implicit variables in the Variables view

Step over the lines within the JSP by pressing the F6 key (Step Over). Notice that the debugger skips any lines with only HTML code.

4. Observe the change in the Variables view. Expand **JSP Implicit Variables** → **pageContext** → **page\_attributes** and select **varAccounts**, which is the variable in the `<c:forEach var="varAccounts" ...>` loop:

Account: 003-333001 balance 9876.52

The customer and the accounts are visible under **JSP Implicit Variables** → **pageContext** → **request\_attributes**.

5. Click **Resume** (🟢) to allow the application to continue with the JSP page generation.
6. Remove the breakpoint.

You have now debugged a local test environment.

## 28.4 Debugging a web application on a remote server

You can connect to and debug a Java web application that has been started in debug mode on a remote application server. When debugging a remote program, the Debug perspective has the same features as when debugging locally. The difference is that the application is on a remote Java virtual machine (JVM), and the debugger must attach to the JVM through a configured debug port. The debugging machine must also map the debug information to its locally stored

copy of the source code. Therefore, it is important that the source code on the debugger machine matches what is deployed on the remote server.

The following example scenario uses the existing IBM WebSphere Application Server v8.0 Beta server as a remote server. To use the existing IBM WebSphere Application Server v8.0 Beta server as a remote server, we delete the current server configuration and launch it externally using the command line. Then the developer node attaches to the application server node and controls it through a debugger.

### 28.4.1 Removing the WebSphere configuration from the workspace

To create the remote WebSphere Application Server, we are going to first delete the current server adapter configuration. For a simple method, follow these steps:

1. Right-click over the **WebSphere Application Server v8.0 Beta at localhost** configuration in the Servers view and click **Delete**.
2. Click **OK** in the confirmation dialog window.

Now the WebSphere Application server is no longer managed by Rational Application Developer.

### 28.4.2 Configuring debug mode to start on a remote WebSphere Application Server V8 Beta

To configure WebSphere Application Server V8.0 Beta to start in debug mode, follow these steps:

1. If the application server is not already running, start it:  

```
<was_home>\bin\startServer.bat server1
```
2. Start the WebSphere administrative console. A secured profile, by default, uses the following URL:

```
https://<hostname>:9043/ibm/console/login.do
```

An unsecured profile, by default, uses the following URL:

```
http://<hostname>:9060/ibm/console/login.do
```

**Tip:** If you do not know the administrative console port number in use in your profile, launch the administrative console from Rational Application Developer first, and inspect the values of the ports. Select **Application servers** → **server1** → **Ports** → **WC\_adminhost\_secure** and select **Application servers** → **server1** → **Ports** → **WC\_adminhost**.

3. In the left frame, expand **Servers** → **Server Types** → **WebSphere Application Servers**.
4. On the Application Servers page, click **server1**.
5. On the Configuration tab, in the Additional Properties section at the lower right, select **Debugging Service** to open the Debugging service configuration page.
6. In the General Properties section of the Configuration tab, follow these steps:
  - a. Select **Enable service at startup** to enable the debugging service when the server starts, as shown in Figure 28-17.

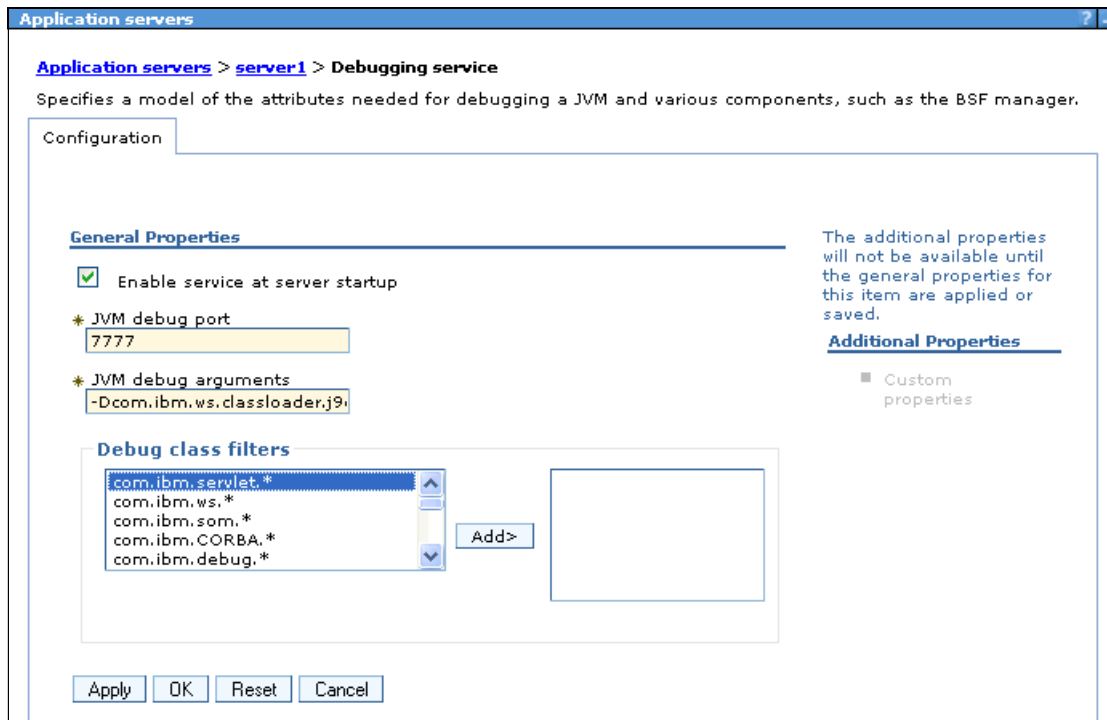


Figure 28-17 Debugging service page in the administrative console

**JVM debug port value:** The value of the JVM debug port is required when connecting to the application server with the debugger. The default value is 7777.

- b. Click **OK** to make the changes to your local configuration.
- c. Save the configuration changes.

- d. Click **Logout**.
7. Restart the application server for the changes to take effect.
8. Verify that the application works properly by navigating to the following URL:  
`http://<hostname/IPAddress>:9080/RAD8EJBWeb/`

### 28.4.3 Attaching to the remote WebSphere Application Server in Rational Application Developer

Assuming that the target server is running in debug mode, complete the following steps to attach to the remote WebSphere Application Server V8 Beta from within Rational Application Developer. The workspace that is used must contain the RAD8EJBWeb project.

1. In the Enterprise Explorer, right-click **RAD8EJBWeb** and select **Debug As** → **Debug Configurations**.
2. Create a new remote debug configuration for WebSphere Application Server V8 Beta. In the Create, manage, and run configurations window (Figure 28-18 on page 1491), follow these steps:
  - a. Double-click **WebSphere Application Server** (or right-click and select **New**).
  - b. Verify the name for this debug configuration (RAD8EJBWeb).
  - c. On the **Connect** tab, make sure that the Project is RAD8EJBWeb. For the IBM WebSphere Server type, select the **WebSphere Application Server v8.0 Beta** server. For the Host name, enter the IP address or target machine name. For the JVM debug port, enter 7777.
  - d. Click **Apply**.

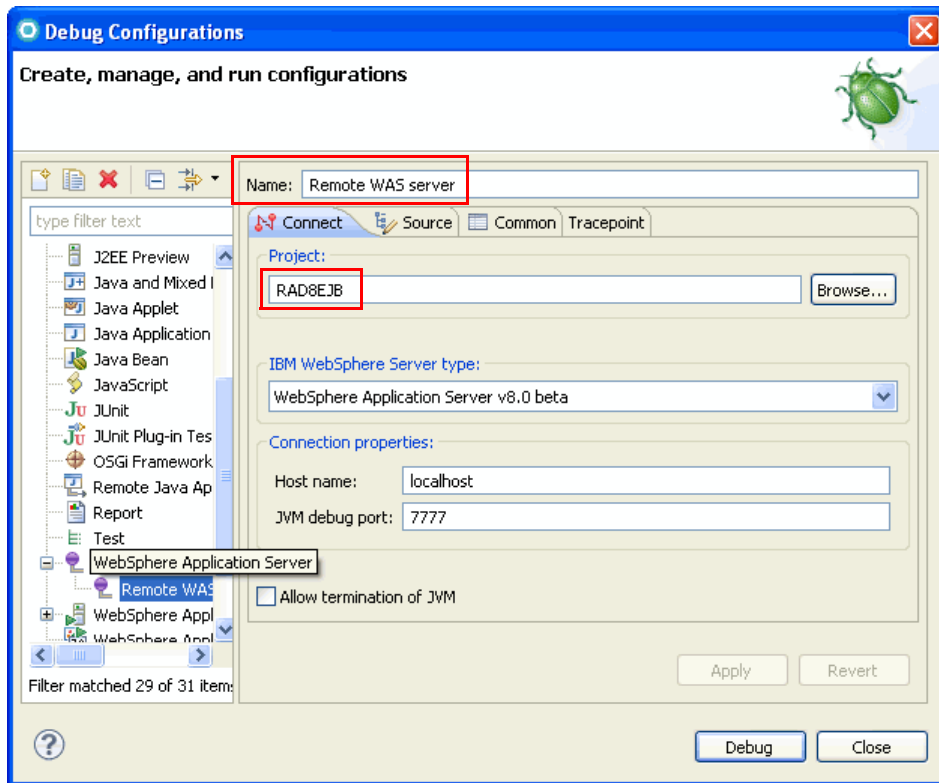


Figure 28-18 Debug configuration for remote debugging

- e. On the **Source** tab, expand the **Default** folder. About halfway down is the RAD8EJBWeb project, so that the debugger knows where the source code is located.
- f. Click the **Common** tab to see the standard options for the debug configuration, including where to save the configuration and where to output the SystemOut file.
- g. Click **Debug** to attach the debugger to the remote server.

The Debug perspective now shows the Remote debugger running in the Debug view, as shown in Figure 28-19 on page 1492. The debugger is waiting for a breakpoint to be triggered. Clicking the Disconnect icon (🔌) stops the debugging.

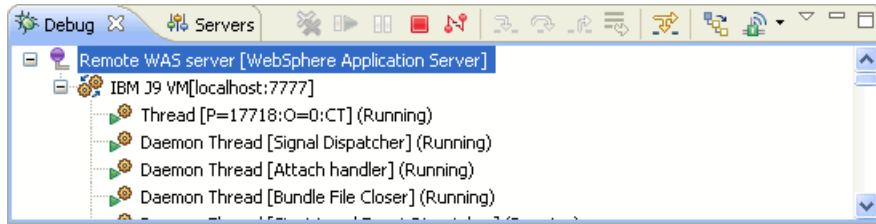


Figure 28-19 Debugging perspective while remote debugging

### Attaching to a local WebSphere instance of Rational Application

**Developer:** When attaching to a local WebSphere instance of Rational Application Developer, you must start the application server in debug mode. Then open the Debug perspective, and disconnect the debug instance that was started by Rational Application Developer automatically when the server was started in debug mode.

Next you can start a remote debug instance using localhost as the host name and port 7777. With these settings, you can attach to the test application server and perform all the usual debugging facilities.

## 28.4.4 Debugging a remote application

From a web browser, navigate to the URL where *hostname* is the IP address or name of the target machine:

```
http://<hostname/IPaddress>:9080/RAD8EJBWeb/
```

You can now debug the remotely running the application in the same way as a locally deployed application:

1. Set breakpoints (Java or JSP).
2. Step through the code.
3. Watch variables.
4. Execute expressions.

To terminate the debugging session, select the remote debugging instance and click **Disconnect** (🔌).

## 28.5 Using the Jython debugger

With the Jython debugger, you can detect and diagnose errors in the Jython script (with the .py or .jy extension) that is used for WebSphere Application



Server administration. You can also control the execution of your code by setting line breakpoints, suspending execution, stepping through your code, and examining the contents of variables. (*Variable values cannot be changed in Jython.*)

## 28.5.1 Considerations for the Jython debugger

The Jython debugger only supports the debugging of scripts that are running on WebSphere Application Server V6.1, V7, and V8 Beta. You can debug a Jython script that has been developed or imported into a Jython project. When you are debugging a Jython script, you can set line breakpoints.

When the workbench is running the script and encounters a breakpoint, the script temporarily stops running. Execution suspends at the breakpoint before the script is executed, at which point you can check the contents of variables. You can then step over (execute) and see the effect that the statement has on the script.

By using the Debug Launch configuration, you can start a debugging session for a given Jython script on either the local test server or a server running on a remote machine. If the target environment is on a remote machine, the host and port numbers must be configured in the `wsadmin` arguments field. See the Rational Application Developer Help for details about this feature.

**Tip:** To debug a Jython script, the server does not have to run in Debug mode.

## 28.5.2 Debugging a sample Jython script

To debug the `listJDBCProviders` script that we describe in 23.11, “Developing automation scripts” on page 1275, follow these steps:

1. Open the **listJDBCProviders.py** Jython script file in the RAD8Jython project.
2. Set a breakpoint in the `showJdbcProviders` function at the following line:  

```
for provider in providerEntryList
```
3. Click **listJDBCProviders.py** and select the following menu from the Rational Application Developer toolbar at the top of the page (not the context menu on the selected file): **Run** → **Debug As** → **Administrative Script**.
4. In the Debug Configurations window (Figure 28-20 on page 1494), complete the following steps:
  - a. For the Name, verify that `listJDBCProviders.py` is entered.

- b. For the Scripting runtime environment, select **WebSphere Application Server v8 Beta**, and for the WebSphere Application Server Profile name, select **AppSrv01**.
- c. Specify a User ID and password if security is enabled.
- d. Click **Apply** and then click **Debug**.

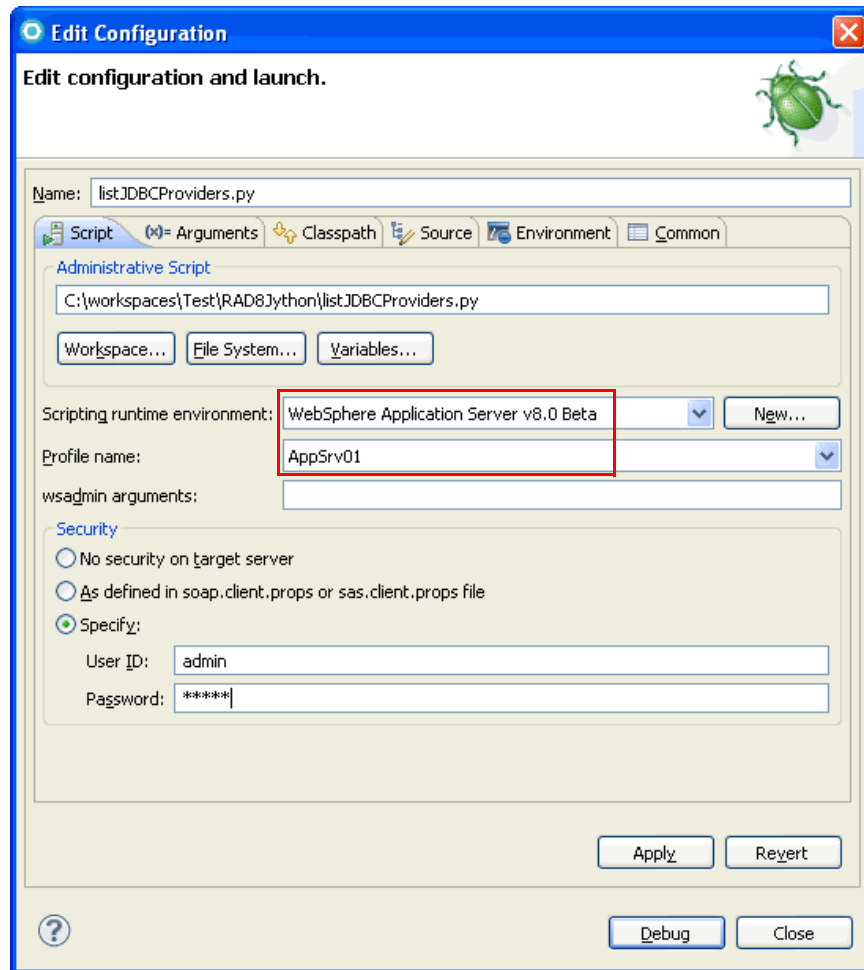


Figure 28-20 Jython debugging configuration

Execution of the script starts and when the breakpoint is encountered execution is suspended.

5. When prompted, switch to the Debug perspective.

The Debug perspective opens and shows the familiar views (Figure 28-21):

- The Debug view shows the thread and is used to step through the code.
- The editor shows the source code and the current position in the code.
- The Variables view shows the Jython variables, which cannot be changed.
- The Breakpoints view shows the breakpoints.
- The Outline view shows the outline of the script.
- The Console shows the output of the script.

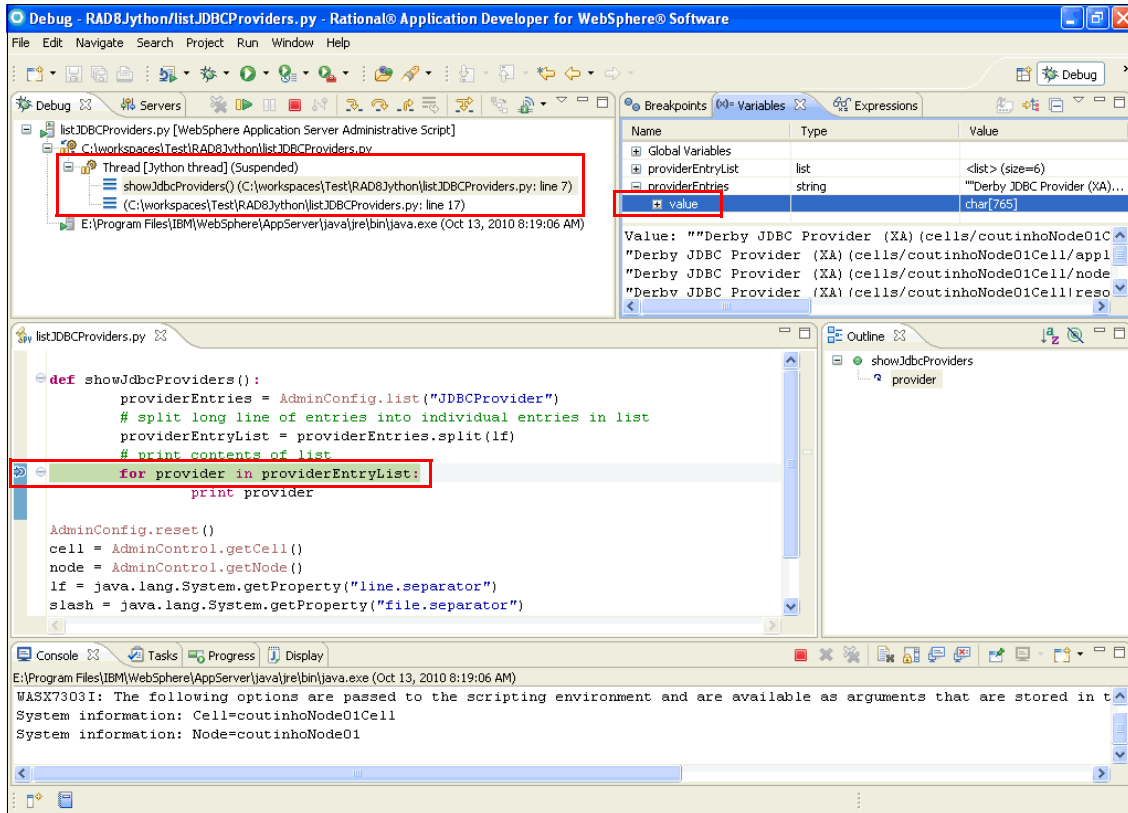


Figure 28-21 Debug perspective when debugging a Jython script

6. Step through the Jython code, and watch the variables.

The Jython debugger is useful when you encounter errors in your Jython scripts. Run the script in debug mode, without having to restart the server.

## 28.6 Using the JavaScript debugger

Rational Application Developer introduces integration with Firebug. *Firebug* is an open source extension for Mozilla Firefox that assists in debugging, editing,

profiling, logging, and monitoring of HTML, Document Object Model (DOM), Cascading Style Sheet (CSS), and JavaScript.

The JavaScript debugger integration provides the following features:

- ▶ The capability to create and remove breakpoints in Rational Application Developer or Firebug and to have the breakpoint changes synchronize with the other automatically.
- ▶ When a breakpoint is reached in Firebug, Rational Application Developer opens the file, either a JavaScript file or HTML style file with a JavaScript region, at the same line that Firebug is currently debugging.
- ▶ When Firebug is stopped at a breakpoint, the Threads view and Variables view of Rational Application Developer are automatically synchronized with Firebug.
- ▶ Errors and warnings that are identified by Firebug are automatically added to the Problems view in Rational Application Developer.
- ▶ Edits to web pages or JavaScript files when pushed to the server, which is done automatically if using the Ajax Test Server, are automatically updated in Firefox and Firebug.

### 28.6.1 Setting the default browser to Firefox

To use the JavaScript debugging tools, Firefox must first be installed. Firefox is needed to set Rational Application Developer to use Firefox with Firebug to debug content that runs in the browser. To configure Rational Application Developer, follow these steps:

1. Click **Window** → **Web Browser**.
2. Select **Firefox with Firebug** (Figure 28-22 on page 1497).

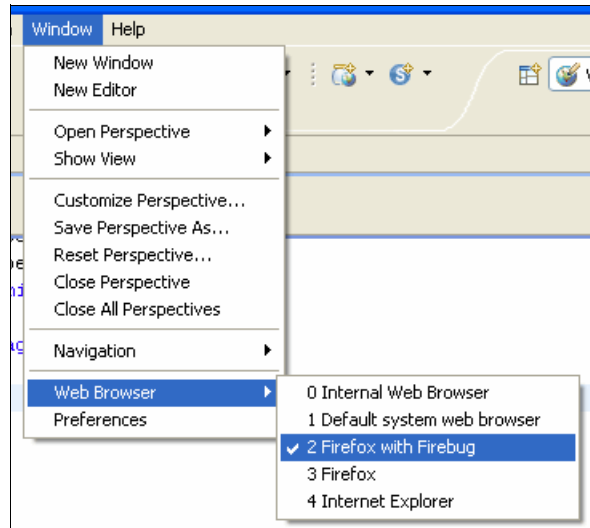


Figure 28-22 Enabling Firefox with Firebug to be the default browser

## 28.6.2 JavaScript debugging

To demonstrate JavaScript debugging, we use a simple sample web application that guesses what number the user enters.

### Importing the sample application

The following task demonstrates how to debug JavaScript. We use the JSDebugWeb web application sample to demonstrate the debug capabilities.

To import the JSDebugWeb web application, follow these steps:

1. In the Web perspective, select **File** → **Import** → **General** → **Existing Project into Workspace** and click **Next**.
2. In the Import Projects window, select **Select archive file** and click **Browse** to locate the **C:\7835code\js\JSDebugWeb.zip** file.
3. Select the **JSDebugWeb** project and click **Finish**.

The sample project is now imported.

### Running the sample Ajax Test Server

After the project is imported, you must run the project on the Ajax Test Server. JavaScript debugging works on several server types, such as WebSphere Application Server, but for this case, we use the Ajax Test Server that comes with Rational Application Developer.

Follow these steps to start the web application:

1. In the Enterprise Explorer, expand the **JSDebugWeb** → **WebContent** directory.
2. Right-click **main.html** and select **Debug** → **Debug on server**.
3. In the servers dialog box, select the Ajax Test Server and click **Finish**.

The application starts and the Firefox web browser opens. If Rational Application Developer is launching Firefox for the first time, Rational Application Developer installs the extra components to Firefox that are needed for enabling the communication between Firefox and Rational Application Developer.

The main.html page is displayed, as shown in Figure 28-23 on page 1471. We have highlighted the icons for the Firefox plug-ins that allow communication with Rational Application Developer (lower-right corner).

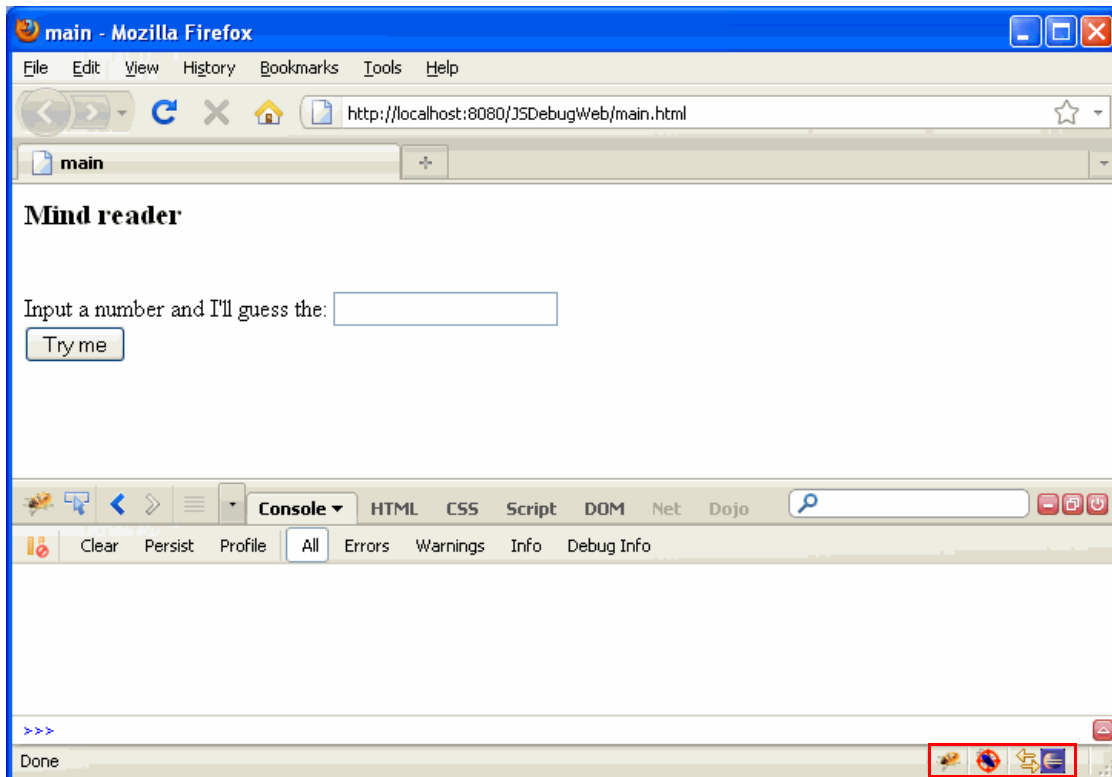


Figure 28-23 Main page in Firefox

Input any number in the text field and click **Try me**. After clicking the button, an error is displayed in Firebug's Console view and automatically synchronized to Rational Application Developer's Problems view (Figure 28-24).

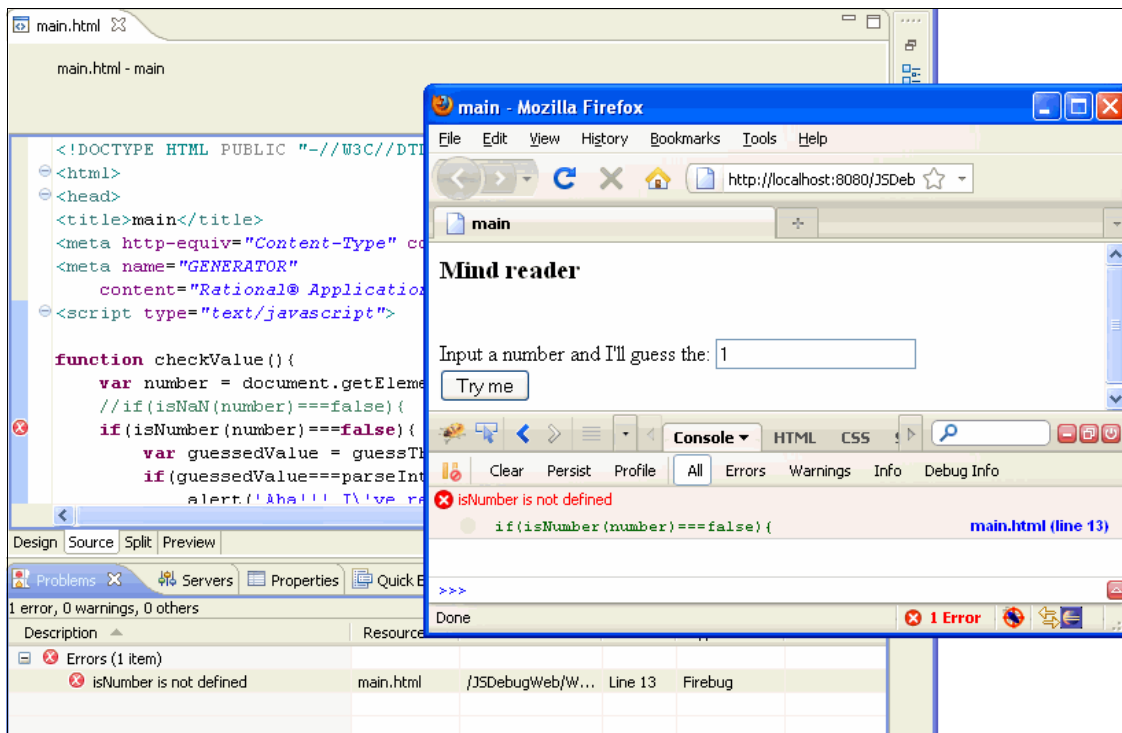


Figure 28-24 Firebug and Rational Application Developer synchronization

Clicking the error in the Problems view of Rational Application Developer opens the editor and highlights the line of JavaScript with the error. We intentionally caused this error by using a call to a non-existent method called `isNumber` instead of the method named `isNaN`. To fix the error, perform the following steps:

1. Go into the editor with the `main.html` file and remove the comment from line 12.
2. Comment line 13.
3. Save `main.html`.
4. Notice, in the Servers view, that the Ajax Test Server automatically synchronizes the latest file changes.
5. Switch to the Firefox window after the server synchronization has finished.

After the server synchronization, Firefox automatically refreshes the page with the latest changes. After the page finishes loading, enter the same number in the text box and click **Try me**. An alert with the resulting message is displayed.

## Debugging the JavaScript sample

You can debug JavaScript code by using breakpoints that are set and modified in either Firebug or Rational Application Developer. The web application must be running on a server.

Perform these steps to start the web application:

1. In the Enterprise Explorer, expand the **JSDebugWeb** → **WebContent** directory.
2. Right-click **main.html** and select **Debug** → **Debug on server**.
3. In the servers dialog box, select **Ajax Test Server** and click **Finish**.

The application starts, and the Firefox web browser opens, displaying the main.html page. Follow these steps to start the breakpoint debugging:

1. In Firebug, select the **Script** tab and select **main.html** from the drop-down menu.
2. In Firebug, click in the margin on line 14 of `main.html` to add a breakpoint.
3. In Firefox, enter a number in the text box and click **Try me**.
4. When the breakpoint is reached, Firebug highlights it.
5. Open Rational Application Developer. You can see that `main.html` is opened and that line 14, which is the line on which Firebug stopped for the breakpoint, is highlighted. See Figure 28-25 on page 1501 for an example of what this action looks like.



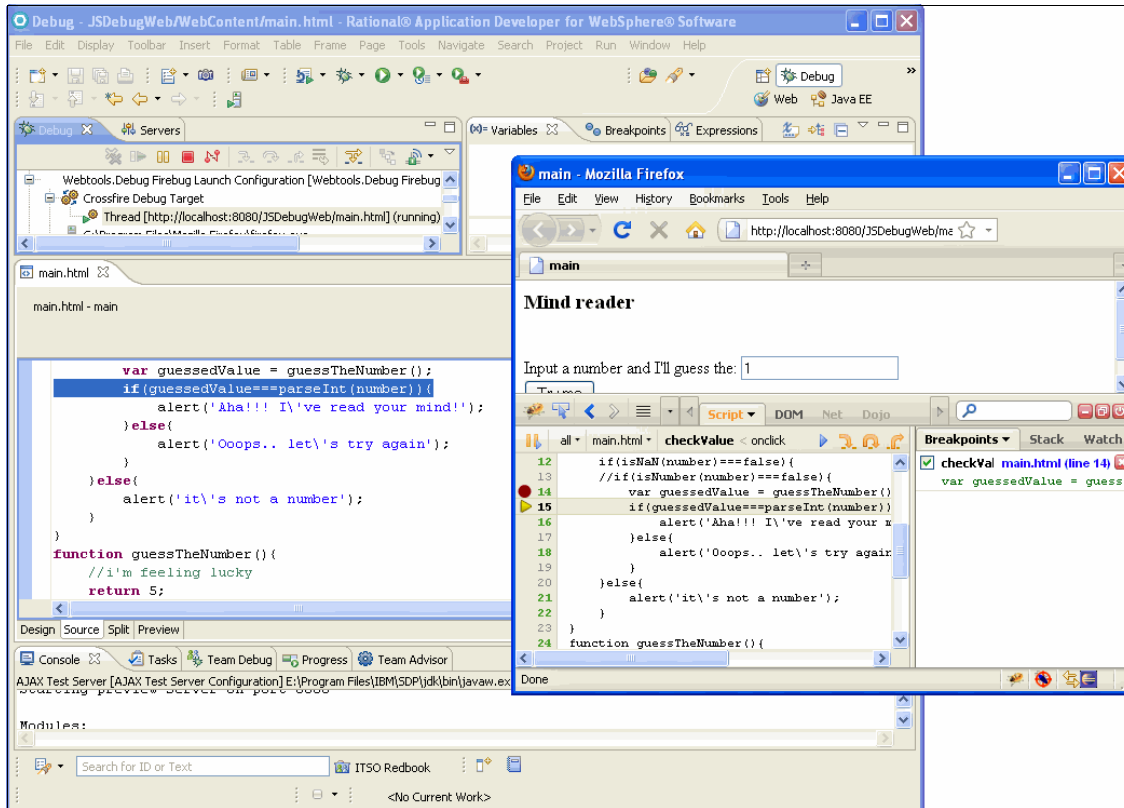


Figure 28-25 Breakpoint synchronization

6. In Rational Application Developer, open the **Debug** perspective.
7. In Rational Application Developer, surface the **Threads** view. The threads that are currently running and paused in Firefox are displayed.
8. In Rational Application Developer, surface the **Variables** view. This view is synchronized with the Watch view of Firebug. This view lists all of the variables that are currently visible when Firebug pauses on a breakpoint.
9. In Rational Application Developer, surface the **Breakpoints** view. This view is synchronized with the Breakpoints view of Firebug.
10. In Rational Application Developer, double-click in the margin of `main.html` on line 14 to remove the breakpoint.
11. Surface **Firefox** with Firebug, and notice that the breakpoint is removed from there as well.
12. In Firebug, click the **Continue** icon to finish loading the page.

For more information about using the new client-side JavaScript debugging, see the Rational Application Developer information center:

<http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.etools.webtoolscore.doc/topics/tdebugfirebug.html>

## 28.7 Using Dojo Debug Extension for Firebug

Debugging Dojo applications presents a unique set of challenges. Fortunately, Rational Application Developer provides additional debugging support for Dojo applications that integrates seamlessly with Firebug. This extension appears as a new Dojo tab under Firebug, and when using the “Firefox with Firebug” browser, is installed automatically.

The Dojo Debug Extension allows you to see important information about your Dojo application, including all widgets, their properties, and event handling using both `dojo.connect()` and the `dojo.subscribe()` APIs. Additionally, this extension allows you to interact with your running application by setting breakpoints and logging messages in key areas of your code, such as when a callback is executed in response to a specific event.

### 28.7.1 Launching the Dojo Debugger

Launching the Dojo Debugger is the same as launching the integrated Firebug extension. The only requirement is that you have a recent version of Firefox installed.

From the Web perspective, choose **Window** → **Web Browser** → **Firefox with Firebug**. After that, any time that you select either **Open With** → **Web Browser** or **Run As** → **Run on Server** on a web page in the Enterprise Explorer, the page opens with Firefox with the Firebug and Dojo Debug extensions automatically installed.

When you launch the Dojo Debugger, you see a Dojo tab appear in Firebug. By default, this extension is disabled and must be enabled to debug Dojo applications. To enable the Dojo Debug extension, select the **Dojo** tab, open the drop-down menu in the tab’s heading, and choose **Enabled** (Figure 28-26 on page 1503).

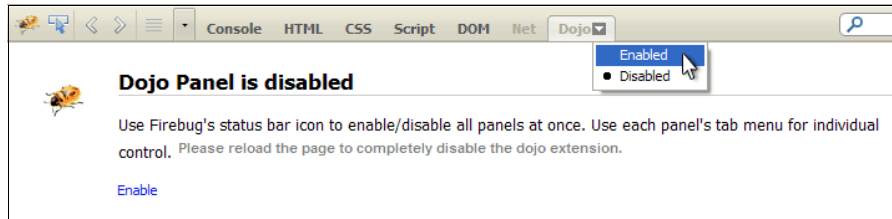


Figure 28-26 The Dojo panel must be enabled to show Dojo-specific information

After enabling the Dojo panel, it is always best to refresh the page to ensure that all the latest information is loaded by the Dojo Debug extension.

The Dojo panel contains three major subviews: *All widgets*, *All connections*, and *All subscriptions*. The following sections examine each of these subviews in detail.

## 28.7.2 Exploring the All widgets view

The *All widgets* view provides a tree-based representation of all the Dojo widgets that are found in the current page. In this simple example page, we see two widgets: a `TextBox` and a `Button` (Figure 28-27).

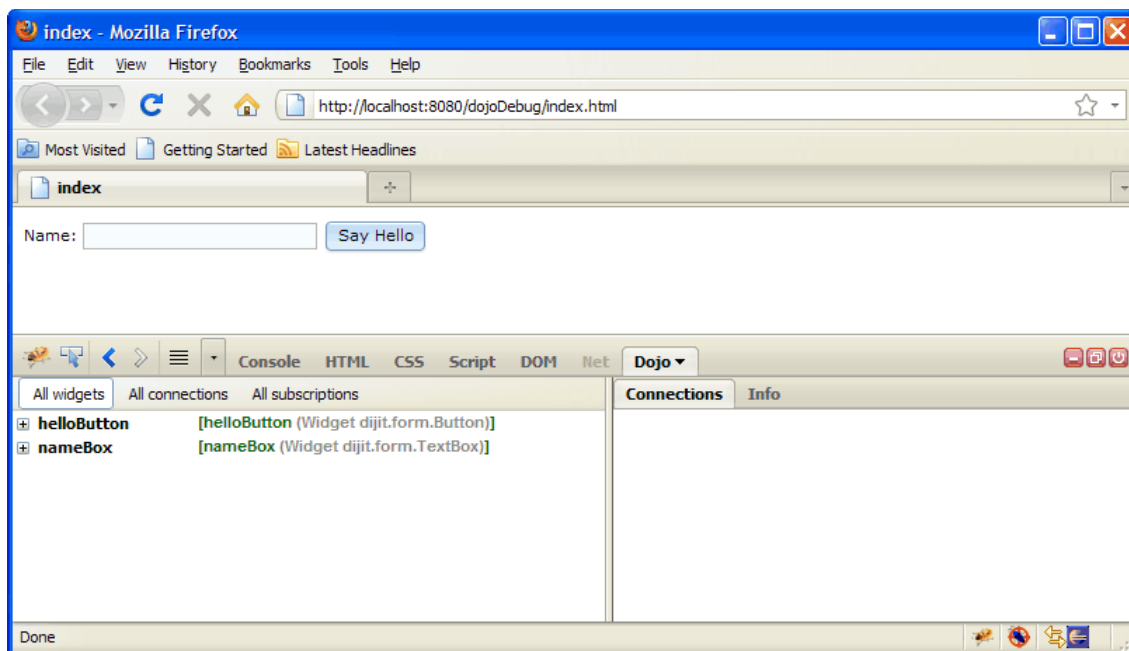


Figure 28-27 The All widgets view shows all the Dojo widgets in the page, including invisible widgets

The All widgets tree contains two columns. The left column contains the widget's ID, and the second column contains both the ID and the declared type for the widget (Figure 28-28). Expanding any widget shows that widget's properties and the current values of those properties.

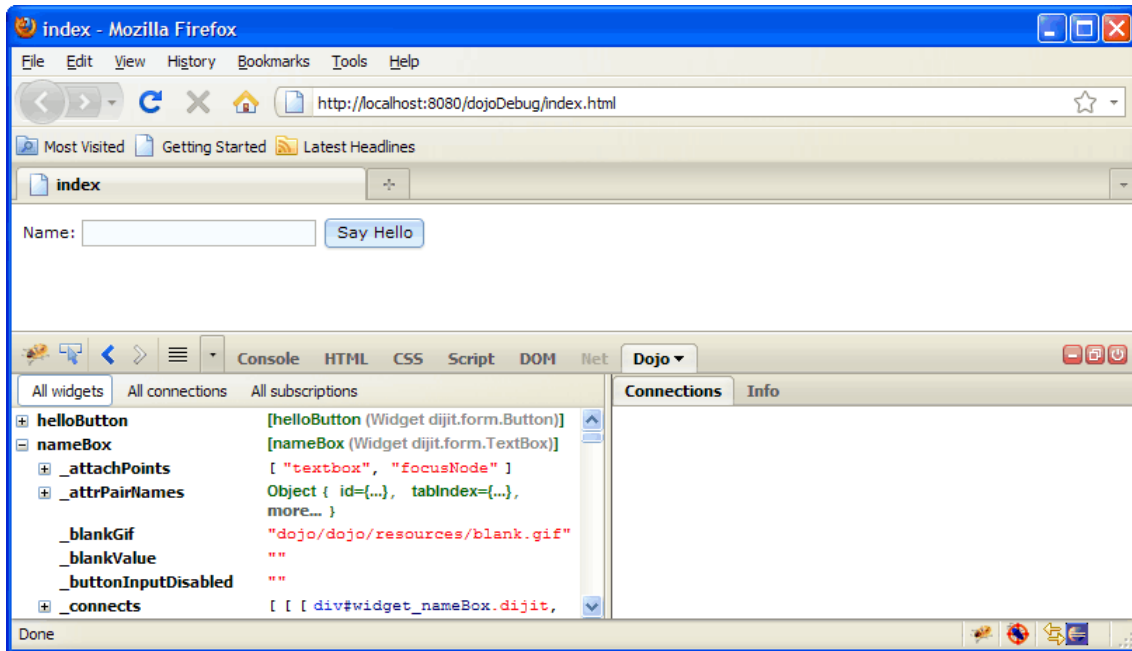


Figure 28-28 Expanding a widget shows its properties and their current values

Hovering your mouse over any widget in the Dojo panel highlights that widget on the page (Figure 28-29 on page 1505).

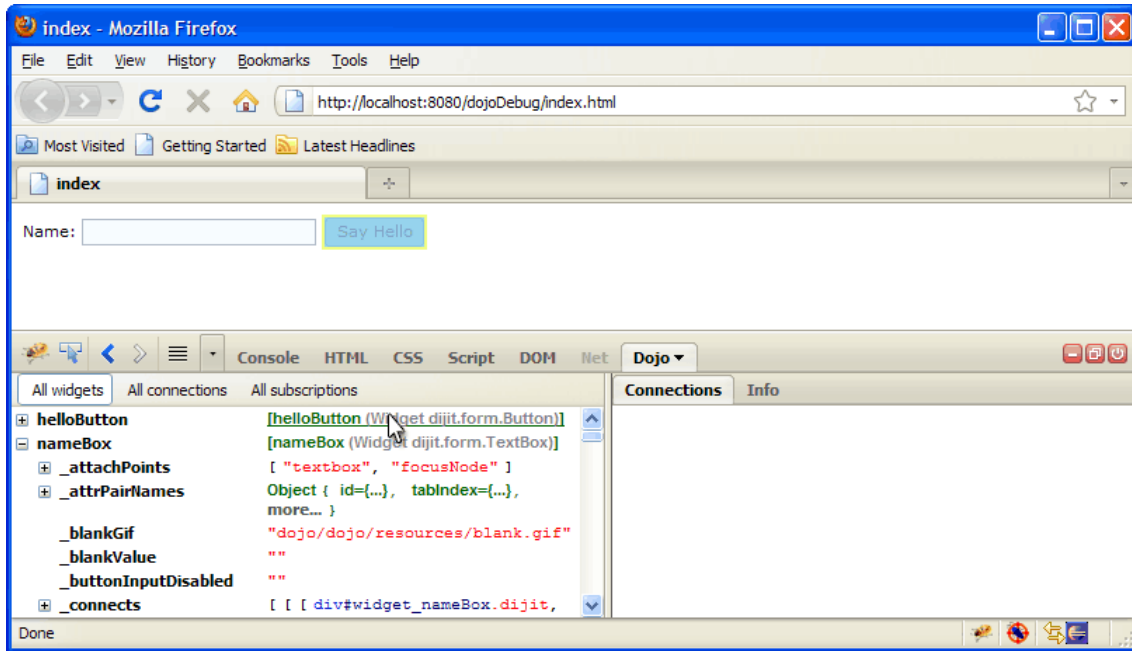


Figure 28-29 Hovering over a widget highlights it on the page

Throughout the Dojo panel, widgets have a variety of helpful context menu actions. In the All widgets view, the context menu actions are *Show Connections*, *Inspect in HTML Tab*, and *Inspect in DOM Tab*. Although this document does not describe each available action, you can examine them by right-clicking to discover all the available actions and how they can help you debug your application.

For now, we look at the Show Connections action (Figure 28-30 on page 1506).

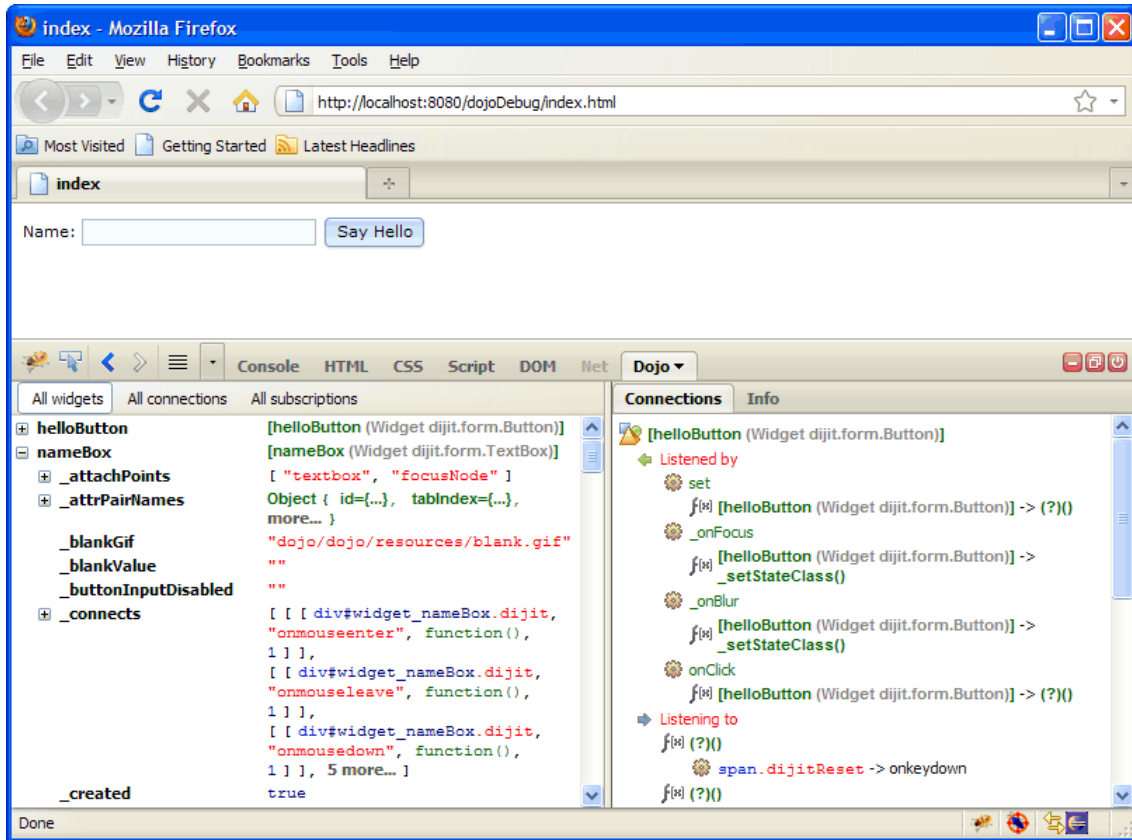


Figure 28-30 The Connections tab shows all of the active connections to and from the widget

We can see that the Connections tab shows a tree of information that is related to the Button widget. The Connections tab shows all callbacks registered using `dojo.connect()` that involve the selected widget.

“Listened by” connections are those callbacks that are registered to listen to events that are generated by the widget. Following the tree deeper, the next level under “Listened by” is the event name against which the callback is registered. Under the event are all the callbacks that are shown as `{object context} -> {callback function}`. For example, you can see that the `onClick` event has a callback registered, which is an anonymous function that will be called with the widget itself as the context. Clicking the callback function in the Connections tab takes us to the Script pane exactly where the function is defined (Figure 28-31 on page 1507).

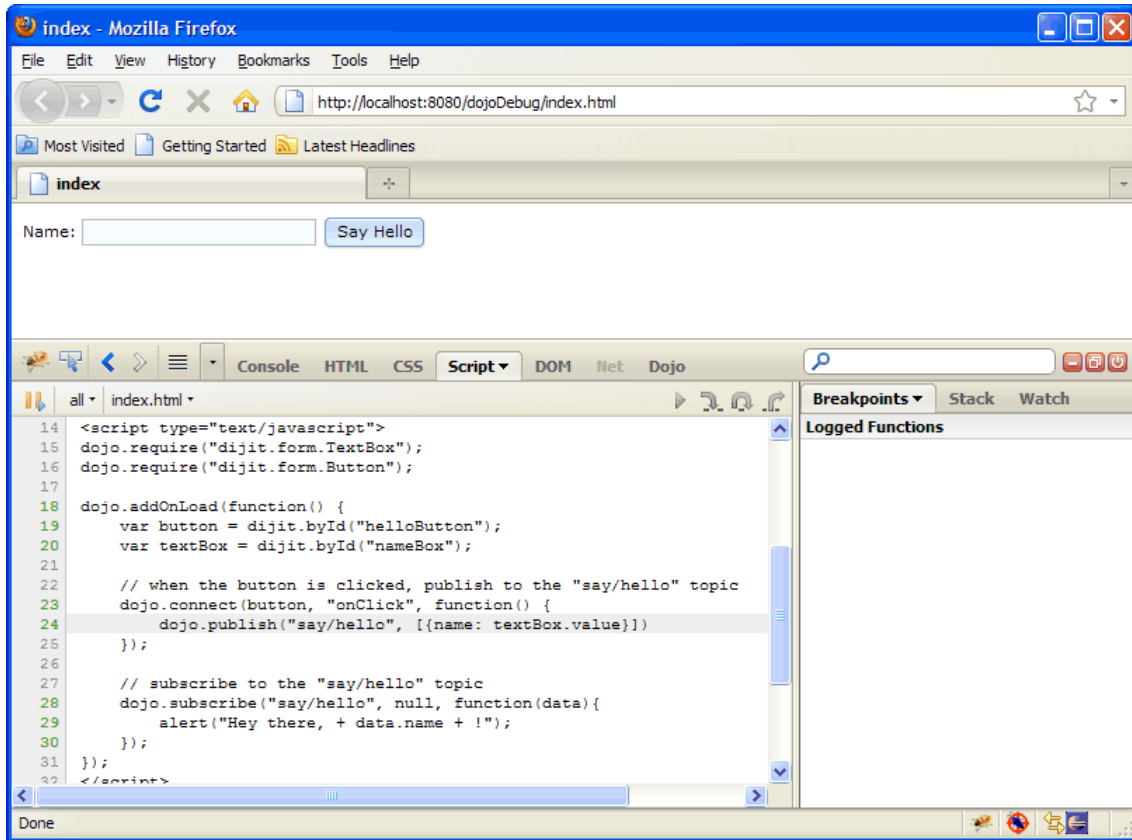


Figure 28-31 Clicking a function navigates to that function's definition in the Script panel

Going back to the Connections tab in the Dojo panel, “Listening to” connections are those connections that the widget has registered to listen to other events. Going deeper into the tree, the next level under “Listening to” is the callback function. Under that level are the object and event to which the callback responds, which are shown as *{object}* -> *{event}*.

In addition to showing useful information about Dojo connections, the Connections tab allows you to set breakpoints on specific pieces of code that are related to each connection. For example, by right-clicking the callback function that is registered against the onClick event of the Button, we see the following available context menu actions for setting breakpoints: *Break on Target*, *Break on Event*, and *Break on connect place*:

- ▶ *Break on Target* sets a breakpoint on the first line of the callback function that is registered to handle this event. This action is particularly handy to ensure that your callback is being called when you expect it to be called. Invoking this

action on our example onClick callback results in the following breakpoint being set (Figure 28-32).

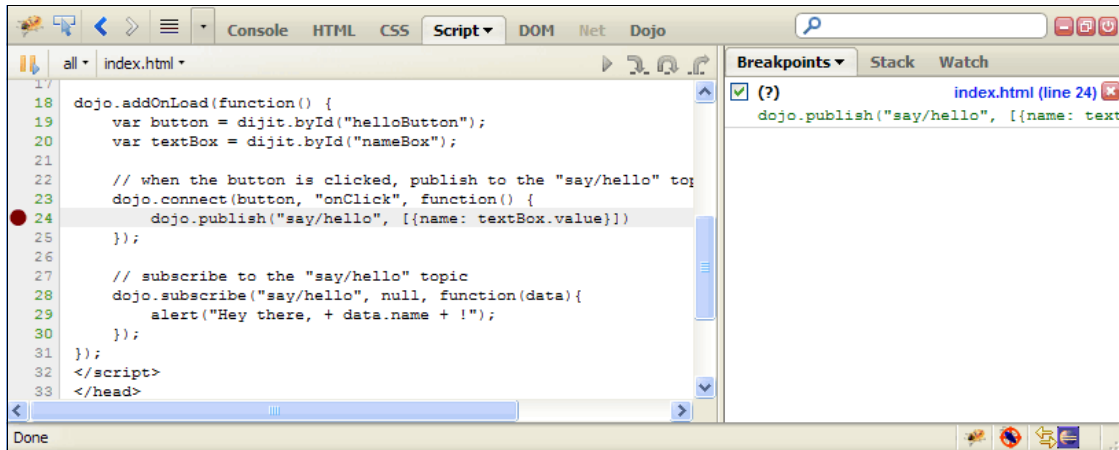


Figure 28-32 The Break on Target breakpoint as it appears in the Script panel

- ▶ *Break on Event* sets a breakpoint inside the function that fires the event. This action is handy if your callback is not being called as expected. Invoking this action on our example onClick callback results in the following breakpoint being set (Figure 28-33).

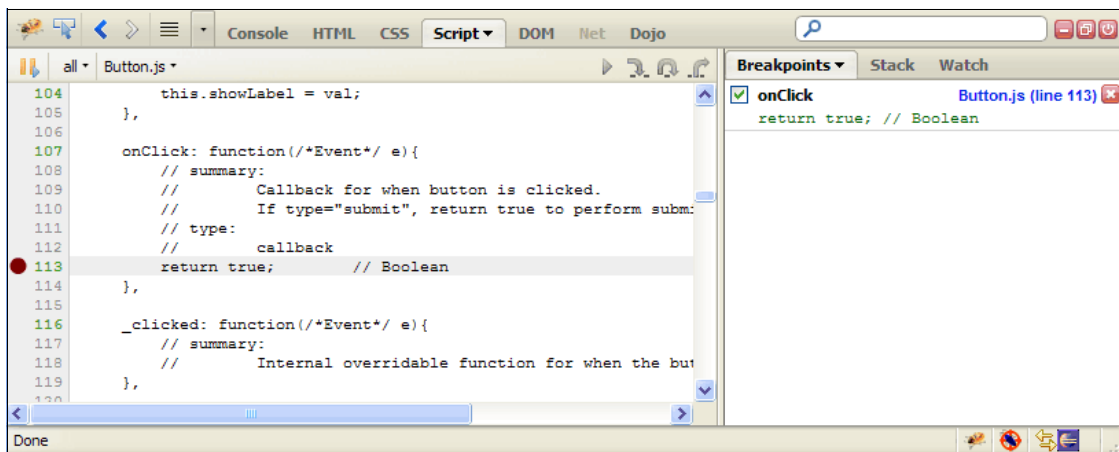


Figure 28-33 The Break on Event breakpoint as it appears in the Script panel

- ▶ *Break on connect place* sets a breakpoint on the dojo.connect() invocation that causes the callback to be registered initially. This action is helpful to ensure that your connection is made at the correct time. Invoking this action



on our example `onClick` callback results in the following breakpoint being set (Figure 28-34).

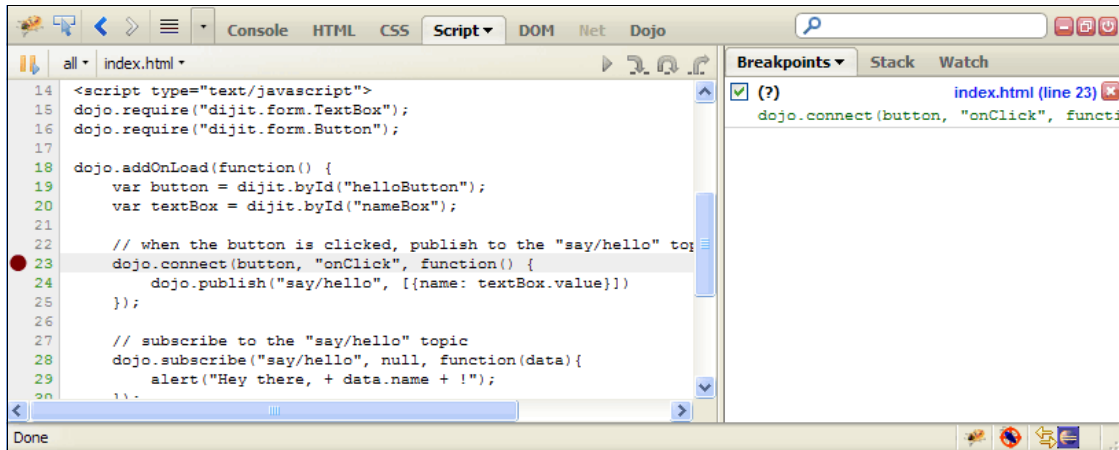


Figure 28-34 The Break on connect place breakpoint as it appears in the Script panel

When used together, the All widgets view, along with the Connections tab in the Dojo Debug Extension can show you valuable information and help you debug through the complex interactions that can occur when using `dojo.connect()` to react to events.

### 28.7.3 Exploring the All connections view

In addition to the All widgets view, the Dojo Debug Extension also provides the *All connections* view, which shows all the active event callbacks in the page in a concise, sortable, and tabular format. Activating the All connections view shows a window that is similar to Figure 28-35 on page 1510.

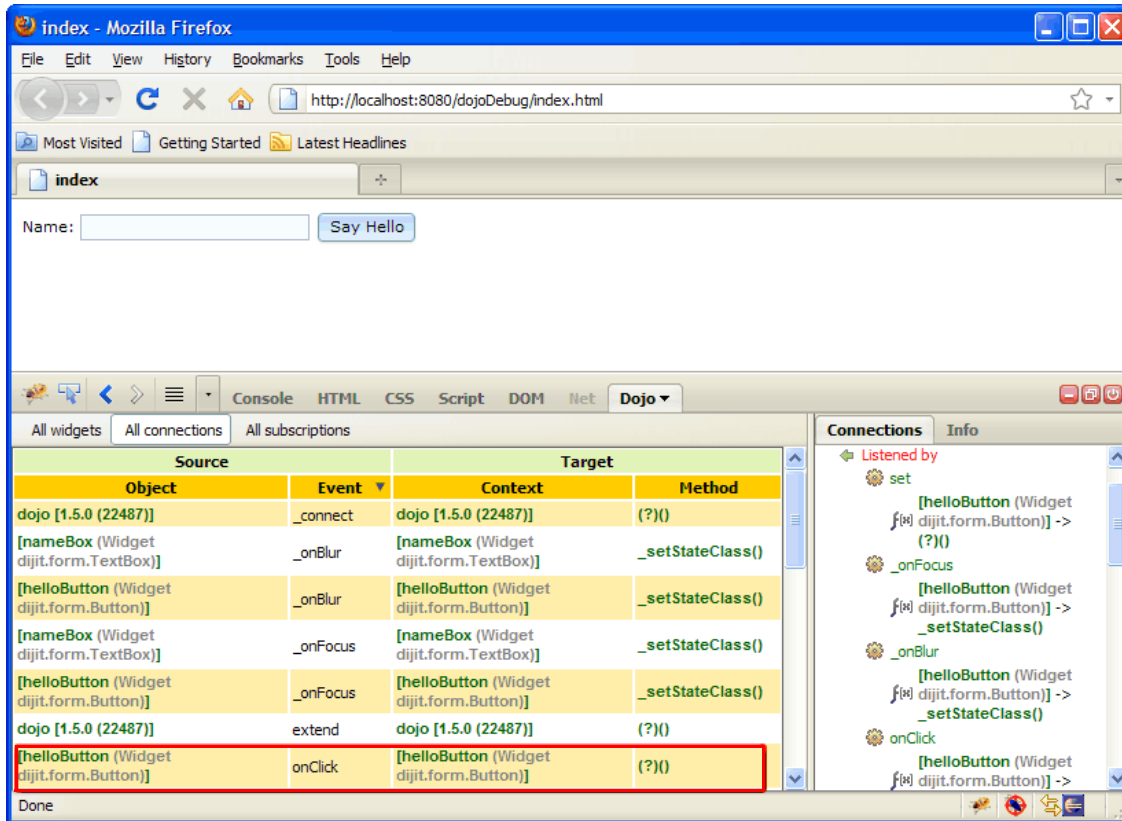


Figure 28-35 The All connections table showing an example connection that is highlighted in red

The table that is shown is first divided into two major headings: Source and Target. The source half of the table lists information about objects that generate events that have callbacks registered to handle. The target half of the table, in turn, lists information about the functions that are registered to handle events that are generated by the source objects. For example, in Figure 28-35, reading left to right, the last connection that is shown in the table (highlighted in red for emphasis) indicates that the Button whose ID is `helloButton` (Object column) has a callback registered against the `onClick` event (Event column). That callback is called in the context of the Button itself (Context column) and is an anonymous function (Method column). This callback is the same example callback that we examined in detail in the previous section.

You can sort each column in the All connections table by clicking the header text. This capability is helpful when trying to answer questions, such as “Are there any callbacks registered against events fired by widget X?” or “Which functions are registered as callbacks on the `onClick` event?”

Notice that hovering over any widget in the All connections table highlights that widget on the page. This behavior is consistent throughout the entire Dojo panel.

Additionally, items in this table have helpful context menu actions that are available. For example, right-clicking the anonymous callback function in the example connection that is mentioned previously presents the following context menu actions (Figure 28-36).

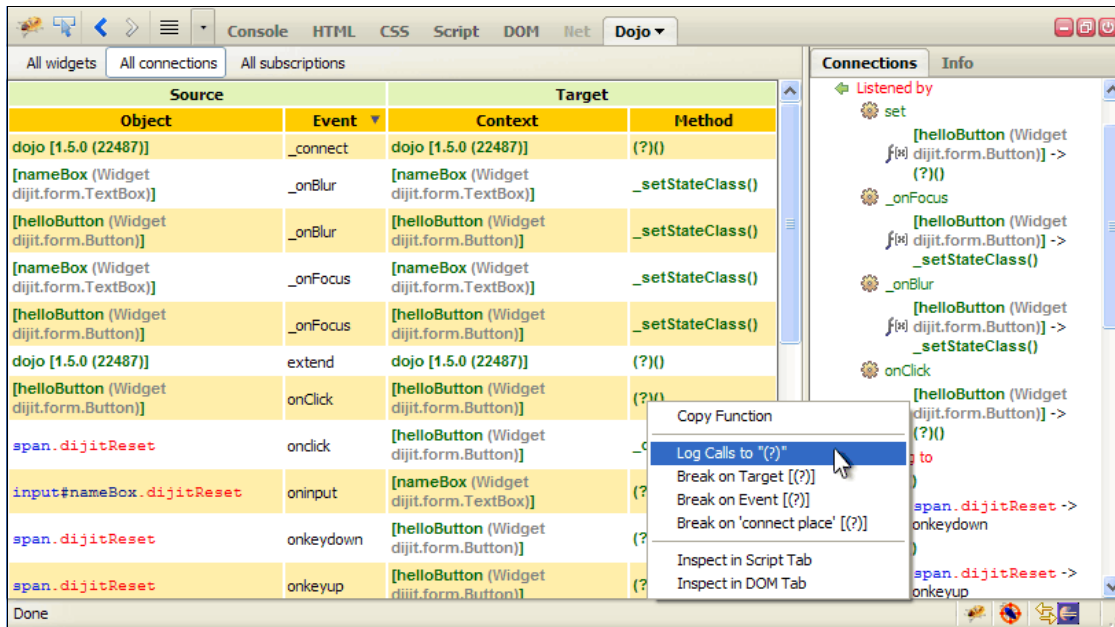


Figure 28-36 A variety of context menu actions are available within the All connections table

Although there are a number of interesting actions available (including the *Break on* actions discussed in the previous section), we see what happens when we select the *Log calls to "(?)"* action. After invoking this action, navigating over to the Script panel shows this anonymous function listed under the Logged Functions section of the Breakpoints tab (Figure 28-37 on page 1512).

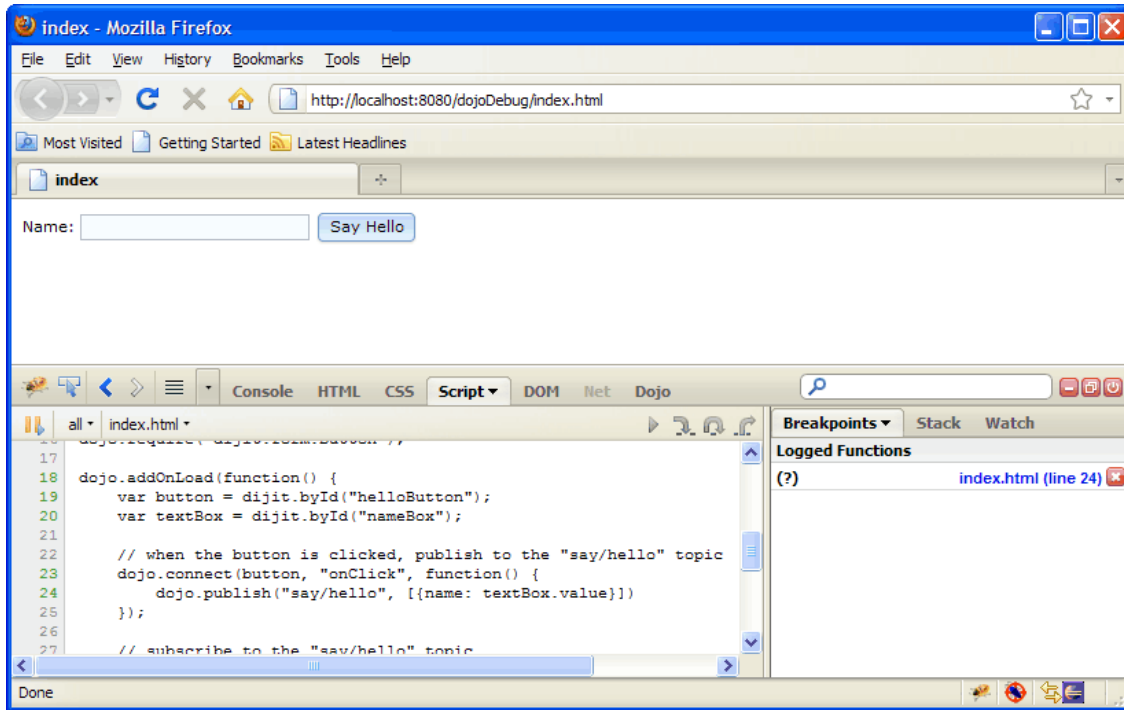


Figure 28-37 Logged functions appear in the Breakpoints tab of the Script panel

If we navigate to the Console panel, a message is logged there each time that the Button is clicked (Figure 28-38 on page 1513).

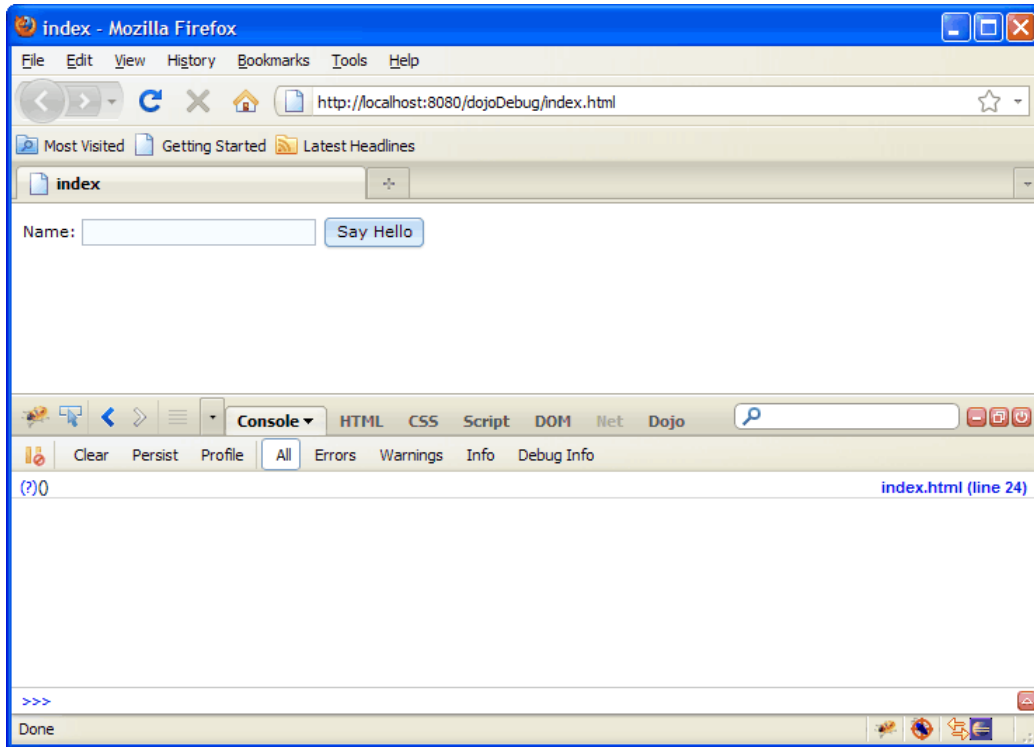


Figure 28-38 A call to the anonymous function is logged in the Console panel

## 28.7.4 Exploring the All Subscriptions view

The third view that the Dojo Debug Extension offers is the *All subscriptions* view, which provides a list of all the topic subscriptions that have been registered using the `dojo.subscribe()` API. See Figure 28-39 on page 1514.

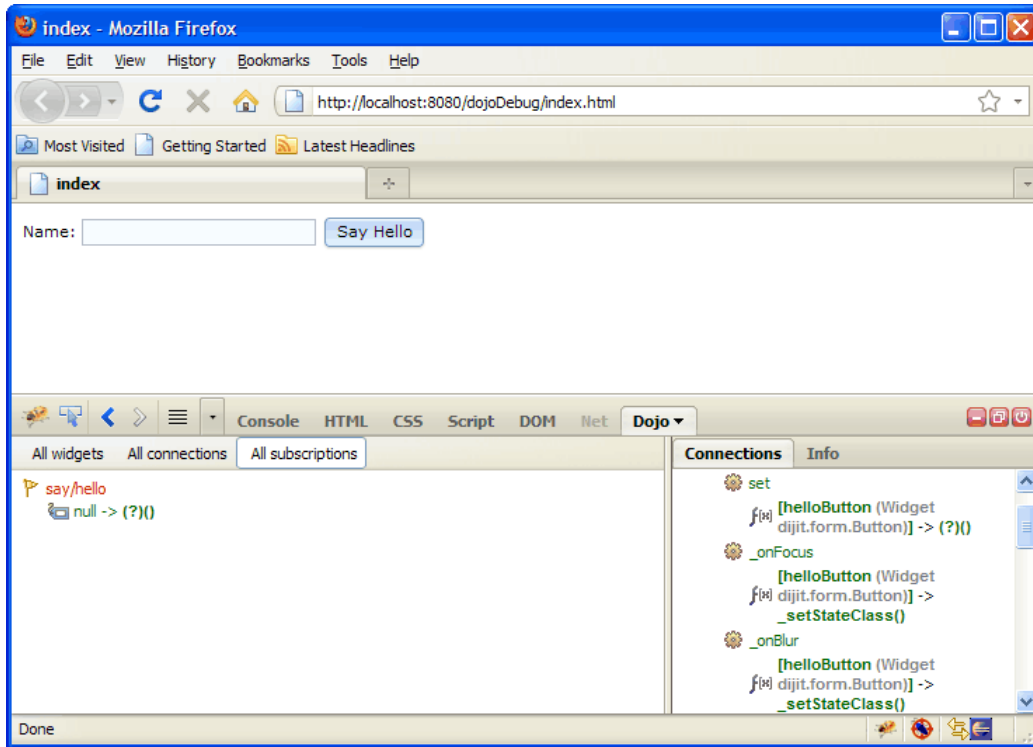


Figure 28-39 The All subscriptions view showing one subscription to the “say/hello” topic

This view provides a tree of information that represents the topics that have active subscriptions. The top-level nodes are the topics themselves. The children of the topic nodes are the callbacks that are registered to fire in response to information being published to that topic using the `dojo.publish()` API. The callback nodes are in the form `{context} -> {callback}`. In this example, our application has one subscription to the “say/hello” topic where the callback is an anonymous function that is called with a null context. Clicking the callback function in the All subscriptions view shows that function’s definition in the Script panel (Figure 28-40 on page 1515).

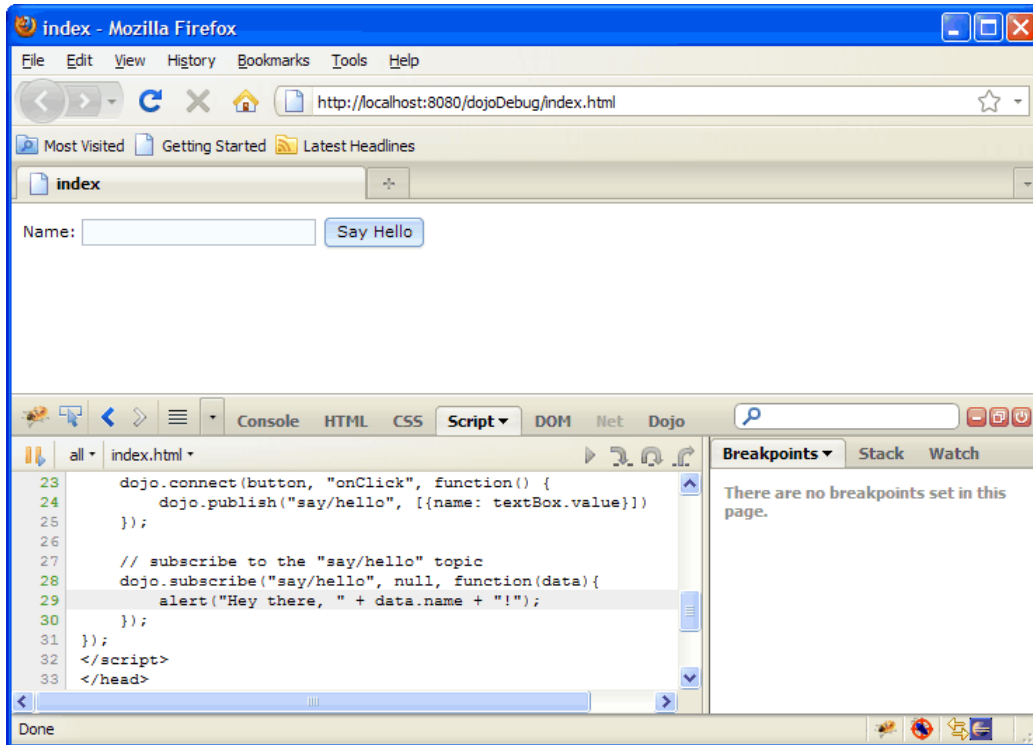


Figure 28-40 The "say/hello" topic subscription callback shown in the Script panel

Additionally, callback functions and context objects have a variety of useful context menu actions to explore.

## 28.7.5 Exploring the Info side panel

The Dojo Debug Extension also provides an Info side panel that displays information about the version of Dojo that is loaded on the page along with all the djConfig loading parameters that are specified (Figure 28-41 on page 1516).

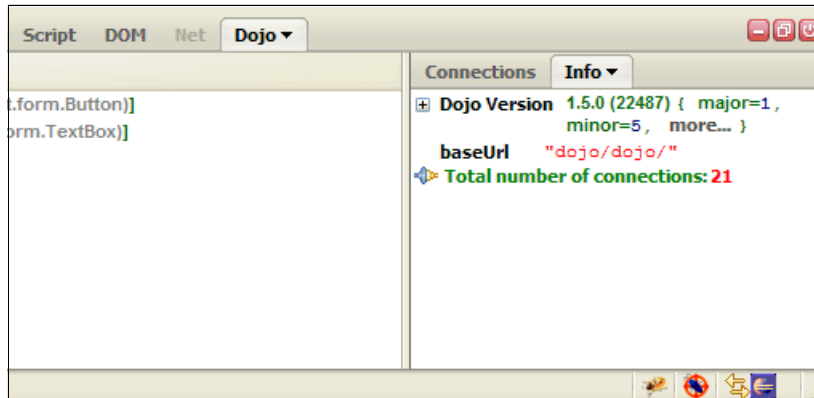


Figure 28-41 The Info side panel

Take special note of the Connection counter in the Info side panel. This counter can be useful to help you find potential lingering, unnecessary connections that can affect performance. As your application runs, you can refresh the connection count by selecting **Refresh** in the Info side panel drop-down menu (Figure 28-42).

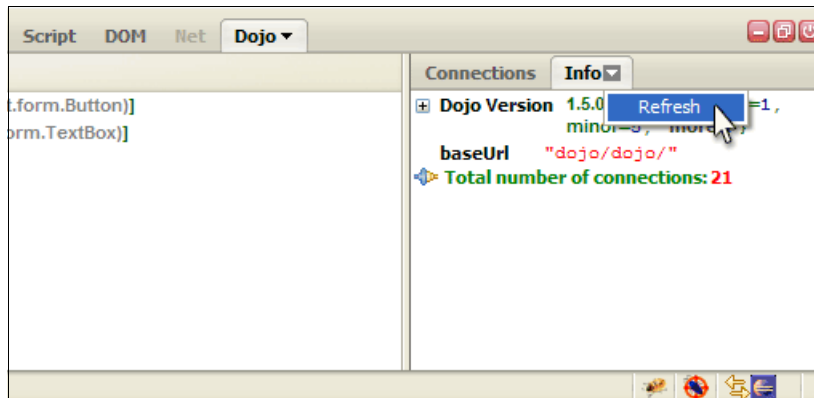


Figure 28-42 Use the Refresh action to update the number of connections

## 28.8 Using the debug extension for the Rational Team Concert client (Team Debug)

You can extend Rational Application Developer by using IBM Installation Manager to support the debug extension for Rational Team Concert.



While debugging complex applications, any given team member might require the expertise of a specialist to understand and correct a specific part of the code. However, recreating a debug session to reproduce a particular scenario can be time-consuming. Rational Application Developer offers the capability to share the complete state of an existing debug session with another user, including any breakpoints that are already set.

If both users are logged on the Rational Team Concert server at the same time, one user can add a debug session to the team repository and then transfer the debug session to the other user (Figure 28-43).

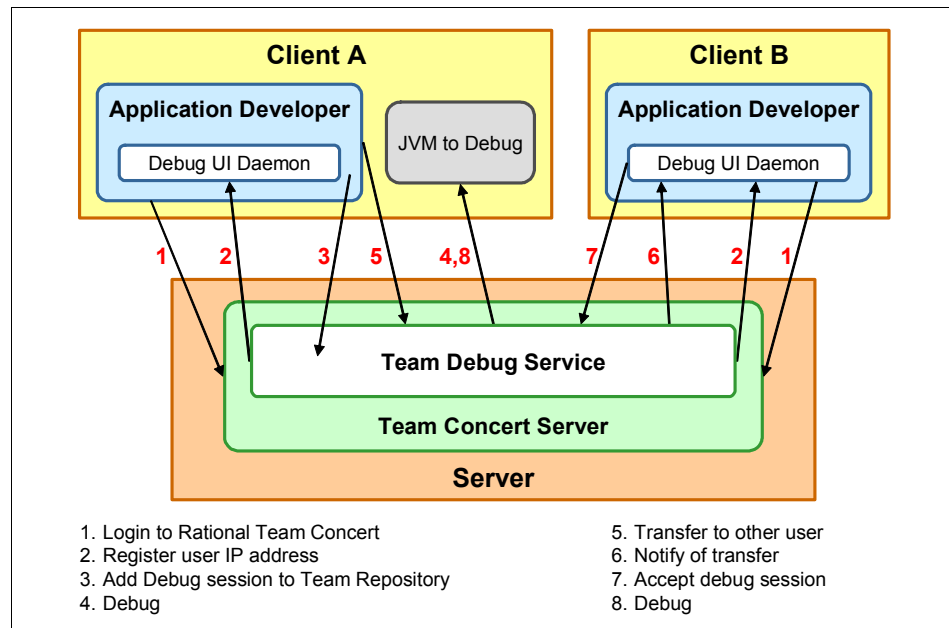


Figure 28-43 Workflow for transferring Java Debug session

## 28.8.1 Supported environments

Team Debug launchers only support Java Version 1.5, and later, as the target Java Runtime Environment (JRE). The supported versions of WebSphere Application Server are V6.0, V6.1, V7.0, and v8.0 Beta. The following types of launch configurations are supported:

- ▶ Debug on Server
- ▶ Eclipse application
- ▶ Java applet
- ▶ Java application
- ▶ JUnit

- ▶ JUnit plug-in test
- ▶ Remote Java application

Although the Team Debug Client supports multiple languages, the Team Debug Server is English only. Therefore, the user can receive error messages from the server in English.

## 28.8.2 Prerequisites

You must satisfy the following prerequisites:

- ▶ This feature uses Rational Team Concert. It requires an installation of additional features on the Rational Team Concert server and on Rational Application Developer (Rational Team Concert client). For installation instructions, see “Installing IBM Rational Team Concert” on page 1824.
- ▶ Two Rational Application Developer users must be logged in to the same Rational Team Concert server, either at the same time in the case of a direct transfer, or potentially at separate times if the debug session is parked on the server.
- ▶ The two users must have imported the same versions of the project to be debugged into the Rational Application Developer workspace. See Chapter 30, “IBM Rational Application Developer integration with Rational Team Concert” on page 1595, for details about how to share a project in Rational Team Concert.
- ▶ In Rational Application Developer, the Debug UI daemon (see 28.2.4, “Stored procedure debugging for DB2 V9” on page 1471) must be active and listening for inbound connections.
- ▶ When the user logs on to the Rational Team Concert server, the user’s IP address and the debug daemon port are registered with the team debug service. This information is necessary to identify the users and send them notifications when they are sent a debug session or when they are requested to transfer their debug session.
- ▶ If users do not receive notifications, check the user IP address and debug daemon settings. The IP address of the client can be determined in the Debug view by clicking the **Debug UI daemon** icon and selecting **Workstation IP**.

After verifying the IP address, debug port, and status of the daemon, try to log out and log in again to Rational Team Concert. This way, the user is registered again with the team debug service with the correct information. If the problem persists, park the debug session and use the Team Debug view to transfer the session.

- ▶ If you test this functionality using two Rational Application Developer instances running on the same machine, remember to change the default value of the Debug UI daemon port (8001) for at least one of the two instances.

### 28.8.3 Sharing a Java application debug session by transferring it to another user

In this example, we show how to share the debug session of a Java application between three users, Rafael, Rodrigo, and Lara. The example scenario is that Rodrigo is debugging a part of the Java Persistence API (JPA) code and asks for help from Rafael. They work on the issue and identify the failing part of the code; however, they do not know how to fix it. They park the debugging session in the server so that they can call Lara, who is the specialist on that part of the code, to continue to debug.

To replace the contents of each user's workspace with a known baseline, perform these steps:

1. In the Team Artifacts view, right-click **My Repository Workspaces**.
2. Select the personal workspace of the user.
3. Right-click **Java Prototype Component**.
4. Select **Replace With** → **Baseline**.
5. Select the **Baseline Imported prototype**.
6. Look at the Pending Changes view.
7. Right-click the **Incoming Changes** and select **Accept**.

To initiate the debug session and transfer it to another user, Lara performs the following steps:

1. Open **EntityTester.java** in the RAD8JPATest project.
2. Place a breakpoint on the first step of the main method:  

```
String customerId = "111-11-1111";
```
3. Right-click **EntityTester.java** and select **Debug As** → **Debug Configurations**.
4. Create a new Java application configuration (Figure 28-44 on page 1520):
  - a. For the Name, enter RAD8 Team Debug Configuration.
  - b. For the Project, enter RAD8JPATest.
  - c. For the Main class, enter `itso.bank.entities.test.EntityTester`.
  - d. Click the **Team** tab (Figure 28-45 on page 1521).

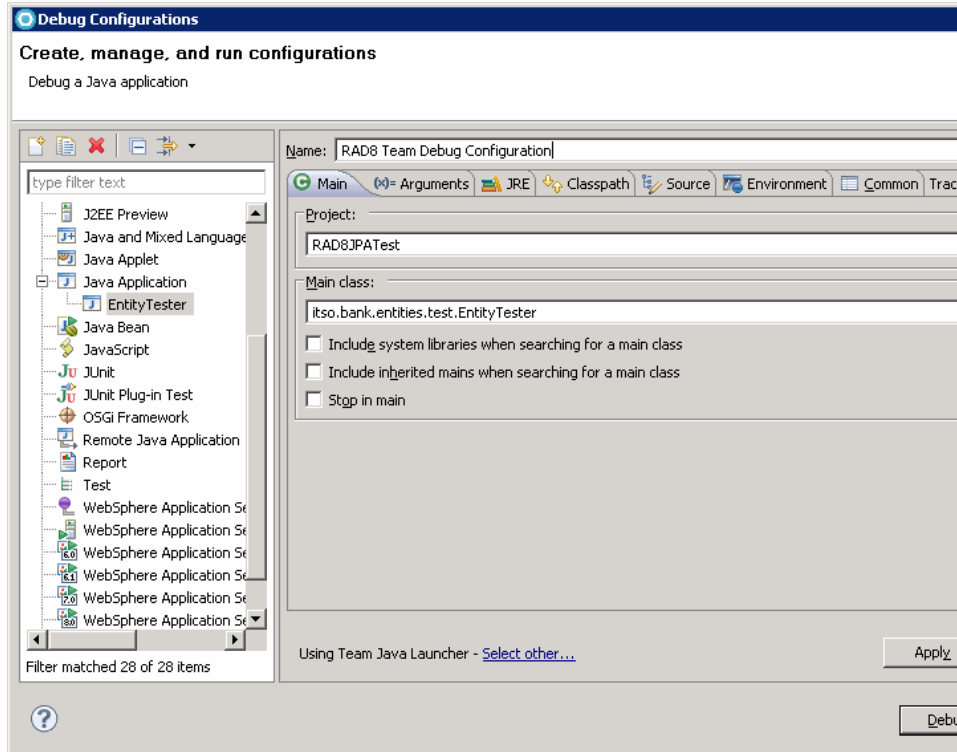


Figure 28-44 Debug configuration for a Java Application

5. On Figure 28-45 on page 1521, Lara performs these steps:
  - a. Select **Add debug session to team repository**.
  - b. For the “Select a team repository to add this debug session to” field, select a name. Our example uses ITS0 RTC REP.
  - c. Click **Debug**.

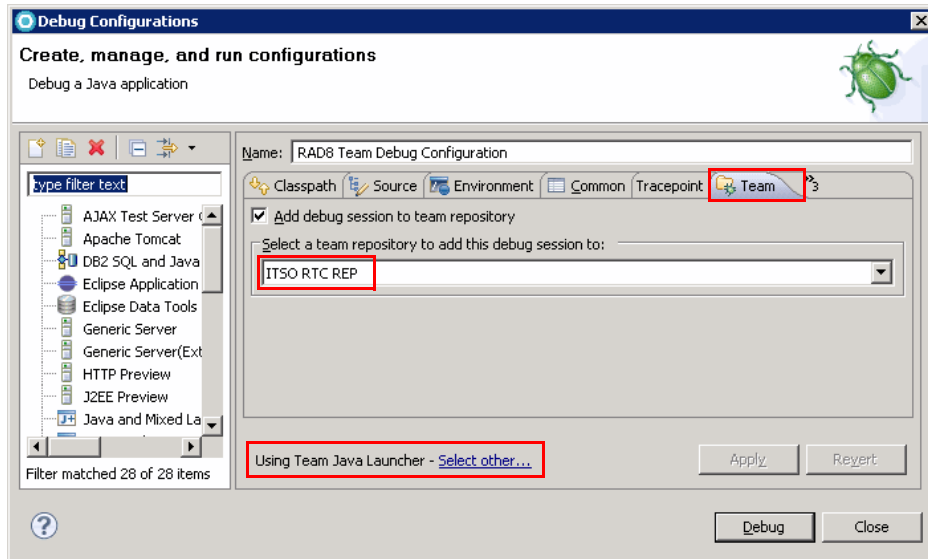


Figure 28-45 Team tab in Debug Configurations window

**Default Java launcher:** The default Java launcher might not be the Team Java Launcher. In order to select it, click the **Select other** link, and in the Preferred Java Launcher dialog, select **Team Java Launcher**.

The Team Java Launcher starts the JVM to debug with the following parameters:

```
-agentlib:jdwp=transport=dt_socket,suspend=y,server=y,address=hostname:
debugPort -Dfile.encoding=<codepage> -classpath <path> <MainClass>
```

You cannot use the Eclipse Java development tools (JDT) Launcher for this purpose, because it starts the JVM to debug without the `server=y` parameter.

Observe the Debug perspective (Figure 28-46 on page 1522). The Debug view shows a line that refers to the virtual machine that is decorated with the following text:

```
[Team] VM [hostname:debugPort]
```

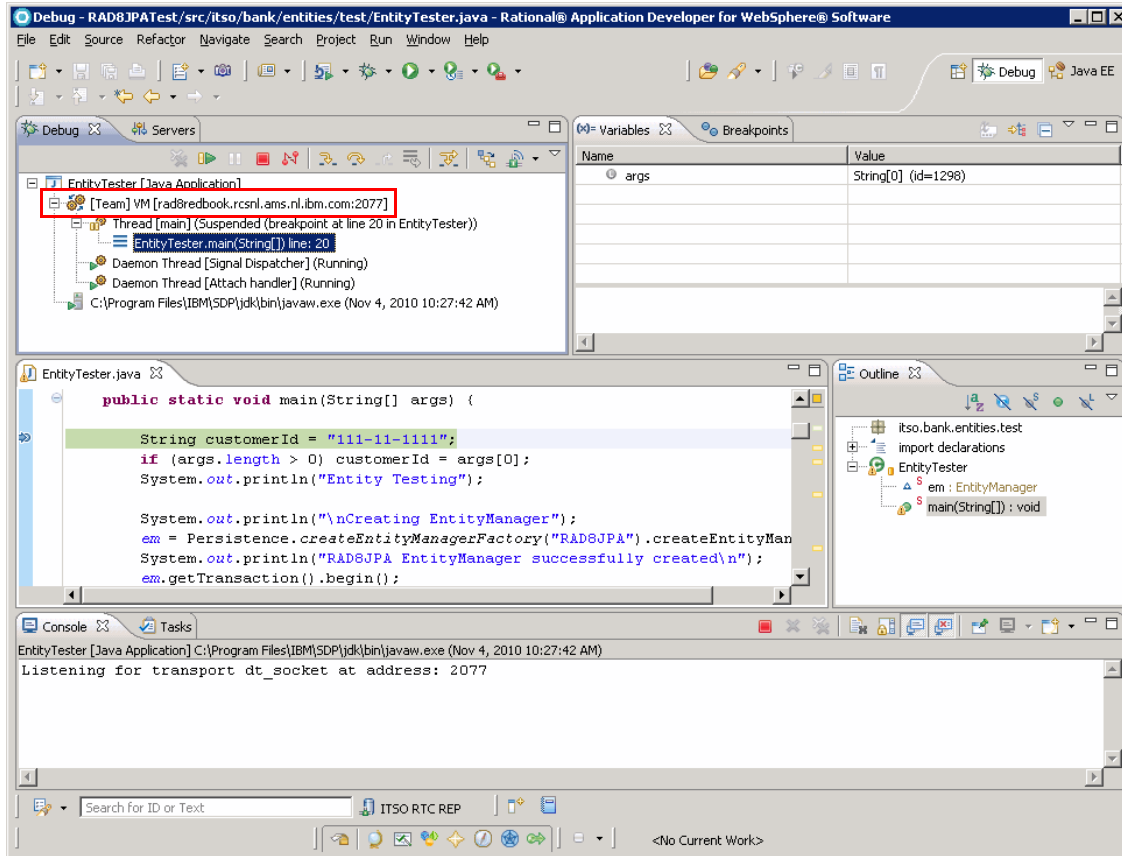


Figure 28-46 Debug perspective during a Team Debug session

To transfer the debug session to Rafael, Rodrigo performs these actions:

1. In the Debug perspective, right-click the line **[Team] VM [hostname:debugPort]** and select **Transfer to User** (Figure 28-47 on page 1523).

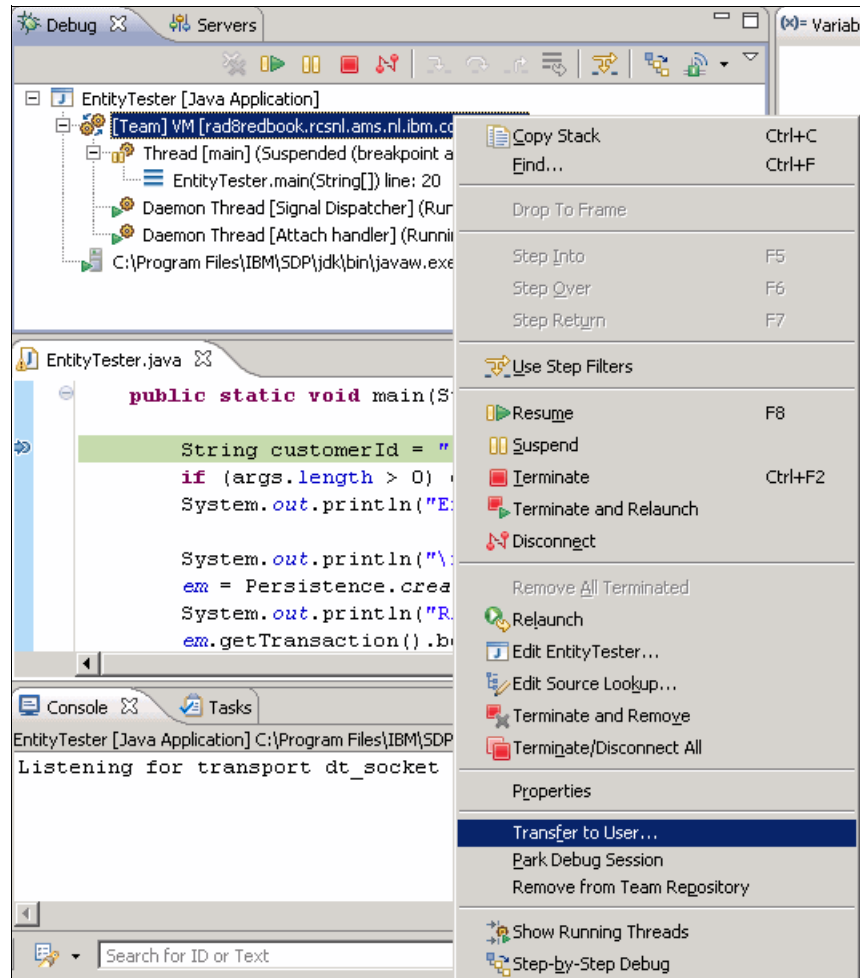


Figure 28-47 Transfer debug session menu

2. Enter the user name or a space followed by the last name to find Rafael (Figure 28-48 on page 1524).

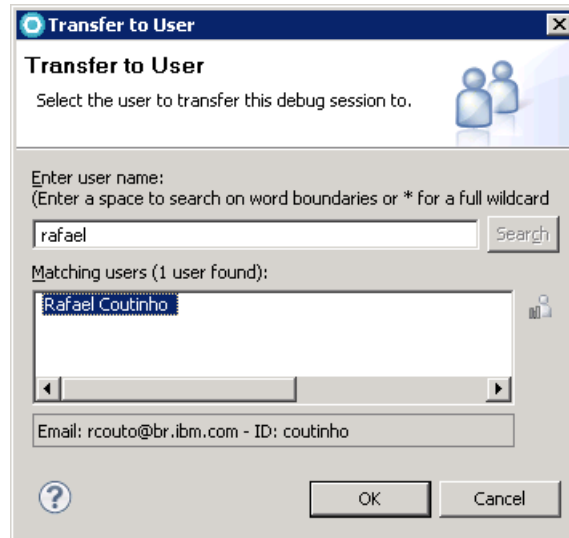


Figure 28-48 Selecting the team member to transfer the debugging session

3. Input any additional information for this transfer request and click **OK** (Figure 28-49).

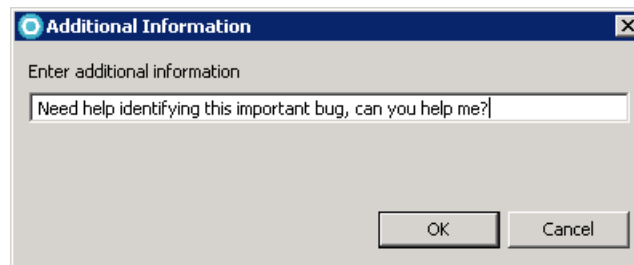
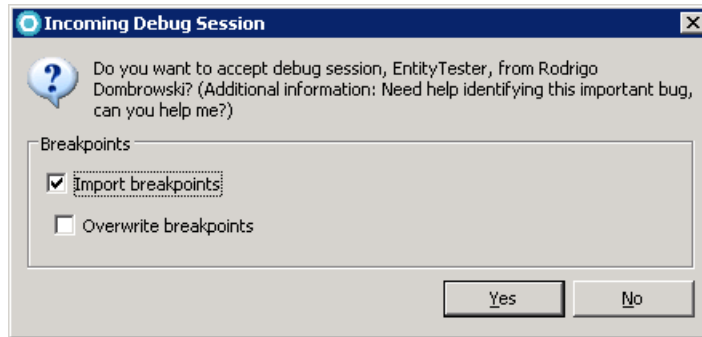


Figure 28-49 Additional information for the transfer request

A dialog box appears on Rafael's window that shows the attempt to transfer the debug session (Figure 28-50 on page 1525).





*Figure 28-50 Invitation to accept an incoming debug session from another user*

The incoming debug session dialog box allows the receiver to import the breakpoints from the sender and to overwrite his own breakpoints. Rafael performs the following steps:

1. Select **Import breakpoints**.
2. Click **Yes**.

**Starting the Java program in debug server mode:** The presence of firewalls between the Rational Team Concert server and Rational Application Developer might prevent this functionality from working. A symptom of this situation is the following message:

```
Unable to add debug session to repository
Reason: Cannot connect to IP_address:debugPort
```

The IP address and port that are shown in the error message are the IP address and port of the JVM to be debugged (typically, on the Rational Team Concert client). The port is randomly chosen by the Team Launcher, which means that you cannot easily open this port on the firewall. One way of fixing the value of the debug port is to start the Java program in debug server mode instead of using the Team Launcher. Note, however, that it is currently not possible to fix the value of the port that is used by the Team Debug Service on the Rational Team Concert server. This function might be considered an enhancement for a future release.

To start the Java program in debug server mode, follow these steps:

1. Create a new Java launch configuration and add it in the VM argument:

```
-agentlib:jdwp=transport=dt_socket,suspend=y,server=y,address=
hostname:debugPort
```

*The host name that is used must be recognizable by the Jazz server. (Do not specify localhost; use the full host name or IP address.)*

2. Start this session in Run mode.
3. Create a Remote Java Application debug configuration and connect to this running JVM for debug.
4. After being connected, add the debug session to the Jazz server.

Rodrigo's debug session is suspended at the same line where he left it, and Rafael can step into the application from the point where Rodrigo transferred it.

If Rafael imported the breakpoints, they are grouped in a Working Set named after Rodrigo's user on the Rational Team Concert server. The debug output and input streams are directed to the console of the user that receives the debug session.

Rafael can decide to transfer the session back to Rodrigo at a later stage in the execution. Alternatively, Rodrigo can also request the session back by performing the following steps:

1. In the Team Artifacts view, expand the **Debug node** under the Project Area.

2. Expand **Search Team Debug Session** → **Started by Me**.
3. Identify the session that was started by Rodrigo and that is currently being debugged by Rafael.
4. Right-click the session and select **Debug** (Figure 28-51).

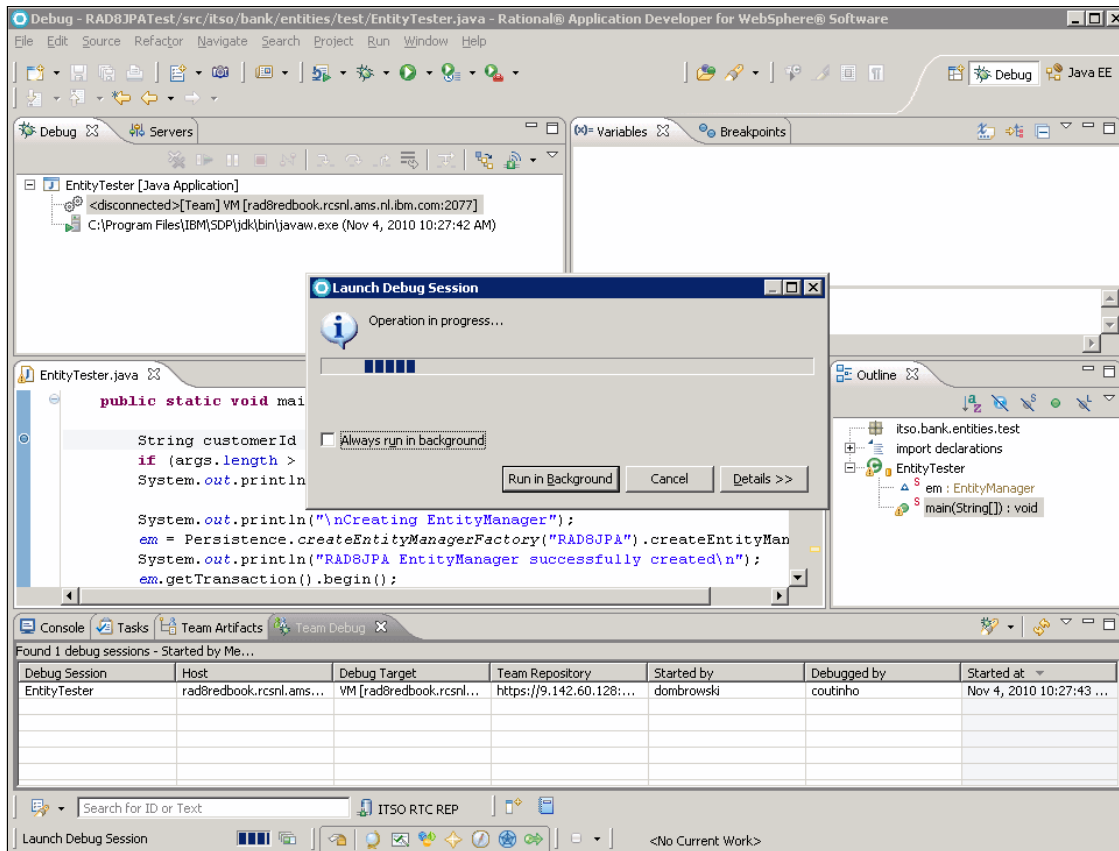


Figure 28-51 Rodrigo requesting the debug session back

Rafael receives a notification that the debug session is requested by Rodrigo (Figure 28-52 on page 1528).

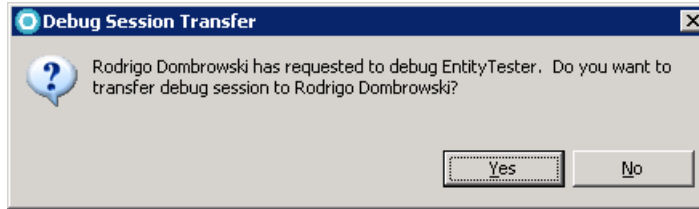


Figure 28-52 Notification

When Rafael accepts, Rodrigo is prompted if he wants to import Rafael's breakpoints. A new debug session is displayed in Rodrigo's Debug view.

Continuing our scenario, we assume that Rodrigo and Rafael were unable to identify the actual problem, but they were able to reduce the problem to certain functions during the debug. They now need the help of a specialist for that part of the code. Lara is the expert on that part of the code, so they need to store that debug session in Rational Team Concert so that Lara can later restart it and work on it.

In order to store that debug session in Rational Team Concert, Rodrigo must park the session in Rational Team Concert. A parked debug session has no owner and can be retrieved by another user at a later time. Anyone can retrieve a parked debug session and gain control of it.

To park the debugging session, open the Debug view, right-click over **[Team] VM [hostname:debugPort]**, and select **Park Debug Session**.

At a later time, Lara can decide to continue Rodrigo's debug session. She performs these actions:

1. In the Team Artifacts view, expand the **Debug node** under the Project Area.
2. In the Search Team Debug Sessions node, select **Parked Debug Sessions**.
3. In the Team Debug view, identify the session that was started by Rodrigo, right-click the session, and select **Debug** (Figure 28-53 on page 1529).

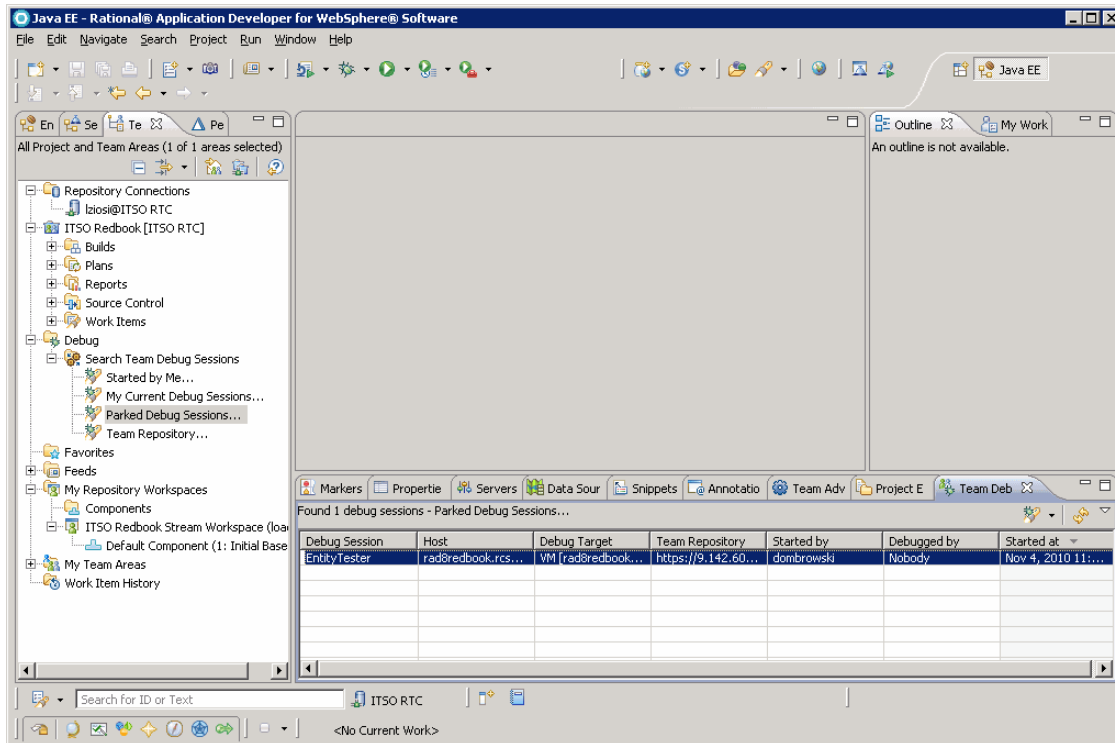


Figure 28-53 Team Debug view showing a parked debug session

4. A dialog prompts whether to import breakpoints. Click **Yes**.

At this point Lara sees, in the Debug perspective, the code that contains the breakpoint. The Variables view shows the contents as determined by the workflow that was previously followed by Rodrigo and Rafael. Lara also sees a copy of the standard output and standard error that is streamed to the console (the same copy that is being streamed to the console of the previous instance of Rational Application Developer). While Lara steps through the code, Rodrigo as the owner of that debug session sees the results in the web browser that is opened on his machine. Lara can then decide to transfer the session to anyone on the team or to park it again on the server.

## 28.9 Obtaining more information

The online help that is provided with Rational Application Developer has detailed information about the following topics:

- ▶ Java development tools debugger
- ▶ Java and mixed language debugger
- ▶ Java 2 Platform, Enterprise Edition (J2EE)/web application debugging
- ▶ Jython debugger
- ▶ Extensible Stylesheet Language Transformation (XSLT) debugger
- ▶ Debug extensions for Rational Team Concert Client
- ▶ DB2 stored procedure debugger
- ▶ Structured Query Language for Java (SQLJ) debugger

See the following websites for Rational Team Concert topics:

- ▶ Installing Rational Debug Extensions for Rational Team Concert Server  
[http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.rad.install.doc/topics/t\\_install\\_teamdebug.html](http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.rad.install.doc/topics/t_install_teamdebug.html)
- ▶ Rational Team Concert Information Center  
<http://publib.boulder.ibm.com/infocenter/rtc/v2r0m0/index.jsp>
- ▶ Video: Team collaboration with Rational Team Concert and Rational Application Developer  
<https://www.ibm.com/developerworks/wikis/download/attachments/140051369/rad75rtccollaborationslimwidescreenpart1.swf?version=1>

IBM developerWorks has several tutorials and articles to explain debugging for various situations, including the following topics:

- ▶ Debugging and Testing Java Applications  
<http://www.ibm.com/developerworks/edu/i-dw-r-radcert2556.html>
- ▶ Getting Started with the New Rational Application Developer XSLT Debugger  
[http://www-128.ibm.com/developerworks/rational/library/05/614\\_debug/](http://www-128.ibm.com/developerworks/rational/library/05/614_debug/)
- ▶ Firebug integration  
[http://www.ibm.com/developerworks/wikis/download/attachments/129008975/crossfire\\_demo2.swf?version=1](http://www.ibm.com/developerworks/wikis/download/attachments/129008975/crossfire_demo2.swf?version=1)



# Part 7

## Management and team development

In this part, we describe the tooling and technologies that are provided by IBM Rational Application Developer for managing and developing applications in a team environment.

This part includes the following chapters:

- ▶ Chapter 29, “Concurrent Versions System (CVS) integration” on page 1533
- ▶ Chapter 30, “IBM Rational Application Developer integration with Rational Team Concert” on page 1595
- ▶ Chapter 31, “IBM Rational ClearCase” on page 1619
- ▶ Chapter 32, “Code Coverage” on page 1697
- ▶ Chapter 33, “Developing Session Initiation Protocol applications” on page 1727

**Sample code for download:** The sample code for all the applications that are developed in this part is available for download at the following address:

<ftp://www.redbooks.ibm.com/redbooks/SG247835>

See Appendix C, “Additional material” on page 1877, for instructions.





## Concurrent Versions System (CVS) integration

In this chapter, we provide an introduction to the widely adopted open source version control system that is known as *Concurrent Versions System* (CVS) and the tools within Rational Application Developer to integrate with it. Through an example scenario showing two developers working on a simulated project, we demonstrate the major features of using CVS within Rational Application Developer.

The chapter is organized into the following sections:

- ▶ Introduction to CVS
- ▶ Configuring the CVS client for Rational Application Developer
- ▶ Configuring CVS in Rational Application Developer
- ▶ Development scenario
- ▶ CVS resource history
- ▶ Comparisons in CVS
- ▶ Annotations in CVS
- ▶ Branches in CVS
- ▶ Working with patches
- ▶ Disconnecting a project
- ▶ Team Synchronizing perspective
- ▶ More information

## 29.1 Introduction to CVS

CVS is a popular open source software configuration management (SCM) system for source code version control. Individual developers or large teams can use it and configure it to run across the web or for any configuration where the users have TCP/IP access to a CVS server. With CVS, users can work on the same file simultaneously without locking, and it provides a facility to merge changes and resolve conflicts when they arise. For these major reasons and the fact that it is available at no cost and relatively easy to install and configure, CVS has become popular both for open source and commercial projects.

CVS is a client/server-based SCM system. The CVS Server software is available from the CVS home page and is released under GNU General Public License. Several options exist for how to run the CVS client software, including the Rational Application Developer functionality, which is described in this chapter.

CVS only implements version control and does not handle other aspects of SCM, such as requirements management, defect tracking, and build management. Other open source projects are available for performing these functions. The IBM Rational suite of products has a large set of tools that perform the same functions in an integrated way.

### 29.1.1 CVS features

CVS offers the following key features (among others):

- ▶ Developers can use multiple client/server protocols over TCP/IP to access the latest code from a variety of clients with access to the CVS server anywhere.
- ▶ All the versions of a file are stored in a single repository file using *forward-delta versioning*, which stores only the differences between sequential versions.
- ▶ Developers are insulated from each other. Every developer works in its own directory, and CVS merges the work in the repository when the developer is ready to commit. *Conflicts* can be resolved as development progresses (using synchronize) and must be resolved before work is committed to the repository.

**Important:** The CVS and Rational Application Developer interpretations of what the term *conflict* means differ slightly:

- ▶ In CVS, a conflict refers to two changes that have been made to the same line or set of lines in the same source code file. In these cases, an automatic merge is not possible, and manual merging is required.
  - ▶ For Rational Application Developer, a conflict means that only a locally modified version exists of a resource for which a more recent revision is available in a branch in the repository. In these cases, an automatic merge might resolve the issue. If changes are made to the same set of lines (which is the case for resolving CVS conflicts), manual merging is required.
- 
- ▶ CVS uses an unreserved checkout approach to version control that helps avoid the artificial conflicts that are common when using an exclusive checkout model.
  - ▶ CVS keeps shared project data in repositories. Each repository has a root directory on the file system.
  - ▶ CVS maintains a history of the source code revisions. Each change is stamped with the time that it was made and the name of the person who made it. Developers must also provide a description of the change. Given this information, CVS can help developers find answers to questions, such as who made the change, when was the change made, why was the change made, and what specifically was changed.

### 29.1.2 CVS support within Rational Application Developer

Rational Application Developer provides a fully integrated CVS client, the major features of which are demonstrated throughout this chapter. At the highest level, two perspectives in Rational Application Developer are important when working with CVS:

- ▶ The *CVS Repository Exploring perspective* provides an interface for creating new links to a repository, browsing the content of the repository, and checking out content (files, folders, and projects) to the workspace.
- ▶ The *Team Synchronizing perspective* provides an interface for synchronizing code on the workspace with code in a repository. This perspective is also used by other version control systems, including IBM Rational Team Concert and IBM Rational ClearCase.

In addition to these two perspectives, numerous menu options, context menu options, and other features are available throughout Rational Application Developer to help users work with a CVS repository.

Rational Application Developer supports the following authentication protocols when establishing a connection to a CVS server:

- ▶ pserver (password server)

This is the simplest but least secure communication method. By using this mechanism, the user name and password are passed to the CVS server using a defined protocol. The disadvantage of this method is that the password is transmitted in clear text over the network, making it vulnerable to network sniffing tools.
- ▶ ext (external)

With this method, the user can specify an external program to connect to the repository. In the Preferences window, you select **Window** → **Preferences** → **Team** → **CVS** → **Ext Connection Method**. Then you can specify the local application to run, the commands to send to the CVS server, and the parameters to pass to the server. Each CVS operation that is performed is then processed through this executable, and passwords are encrypted using the mechanism that the executable uses. Generally, this mechanism requires shell accounts on the server machine so that the clients can use the remote shell applications.
- ▶ extssh (external program using Secure Shell (SSH))

This method is similar to the ext method, but it uses a built-in SSH client that is supplied with Rational Application Developer to perform encryption. In the Preferences window, you select **Window** → **Preferences** → **General** → **Network Connections** → **SSH2**. A set of options is presented for configuring the SSH security, including the SSH application path and private keys.
- ▶ pserverssh2 (password server using Secure Shell)

This method uses the pserver mechanism of storing the user IDs and passwords within the CVS server as shown before, but it uses SSH encryption (as configured in the SSH2 Connection Method Preferences window) to communicate the user and password information to the server.

The pserver option is the easiest to configure and is used in the example in this chapter. For more information about these configurations, see the CVS home page (refer to <http://www.nongnu.org/cvs/>) and Rational Application Developer Help.

## 29.2 Configuring the CVS client for Rational Application Developer

In this section, we explain how to create a client connection within Rational Application Developer to a CVS server. Typically, you perform these activities in Rational Application Developer from a new workspace.

### 29.2.1 CVS Server Installation

The following scenarios assume that CVS Server software has been installed and has two users (`cvsuser1` and `cvsuser2`) configured. Covering the installation of the CVS Server is outside the scope of this book which is focussed on the tooling support available in Rational Application Developer. There are versions of CVS Server software available for Windows, Linux and UNIX some of which are open source and some have a license fee. The CVS home page (<http://www.nongnu.org/cvs/>) is a good starting point for finding links to this software.

### 29.2.2 Configuring the CVS team capabilities

Verify that team capabilities CVS support is enabled by performing these steps:

1. Select **Windows** → **Preferences**.
2. Select and expand **General** and select **Capabilities**.
3. Select the **Team** capability.
4. Click **Advanced**.

5. In the Advanced Capabilities Settings window (Figure 29-1), expand the **Team**, and verify that **CVS Support** is selected. Click **OK**.

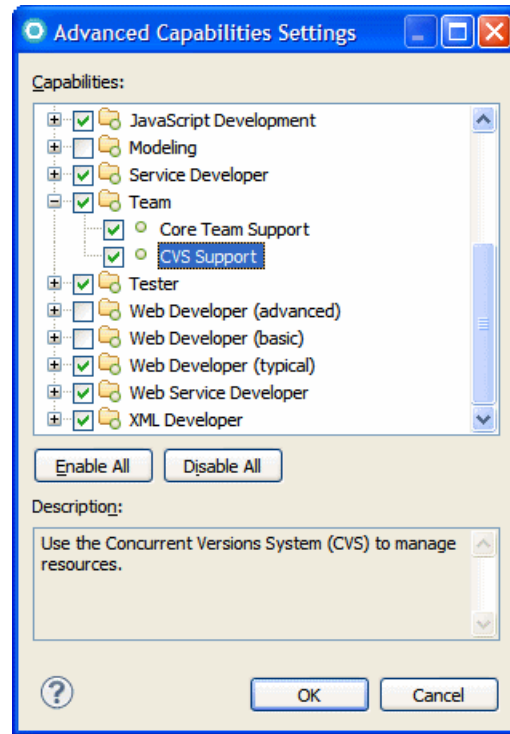


Figure 29-1 Verifying that the Team capability CVS Support is enabled

6. In the Preferences window, click **Apply** and then click **OK**.

### 29.2.3 Accessing the CVS repository

To access a repository that has been configured on a server for users to perform their version management, follow these steps:

1. Select **Windows** → **Open Perspective** → **Other** → **CVS Repository Exploring**. Click **OK**.
2. In the **CVS Repositories** view, right-click and select **New** → **Repository Location**.

3. In the Add CVS Repository window (Figure 29-2), add the parameters for the repository location:
  - a. In the Host and Repository path fields, enter the names that reflect where the CVS repository is located.
  - b. For the User and Password fields, enter the name of the user of the workspace and the password.
  - c. For the Connection type, select **pserver** or whichever connection type is appropriate (refer to 29.1.2, “CVS support within Rational Application Developer” on page 1535).
  - d. Select **Validate connection on finish**.
  - e. Select **Save Password** and click **Finish**.

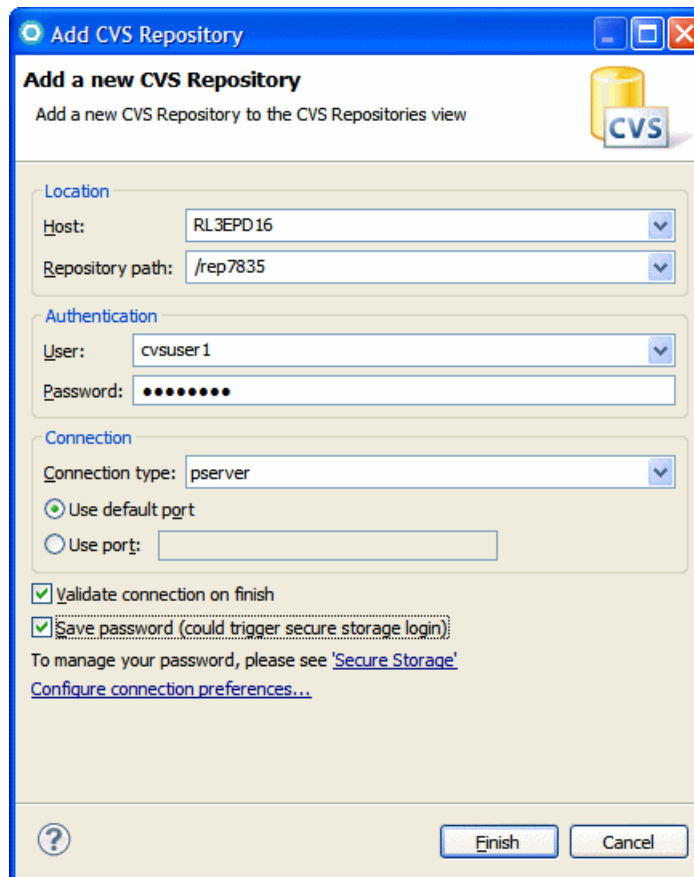


Figure 29-2 Add the CVS repository to the workspace

4. In the Secure Storage window, click **No** unless you want to establish the password recovery feature.

If everything worked correctly, you see a repository location in the CVS Repositories view (Figure 29-3). You can expand the entry to see HEAD, Branches, Versions, and Dates.

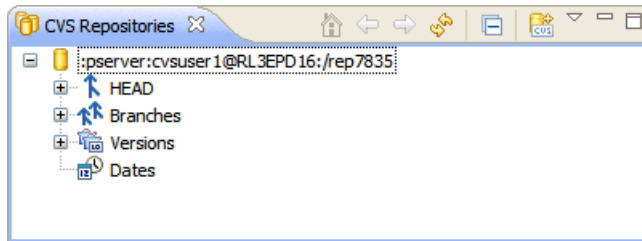


Figure 29-3 CVS Repositories view

## 29.3 Configuring CVS in Rational Application Developer

Within Rational Application Developer, you can set the following CVS-related settings to guide the integration of CVS:

- ▶ Label decorations
- ▶ File content
- ▶ Ignored resources
- ▶ CVS-specific settings
- ▶ CVS keyword substitution

### 29.3.1 Label decorations

*Label decorations* are set to be on for CVS by default, which means that the CVS properties of a particular file are shown on its label or icon. For example, if a file changes from the version in the repository, it has a greater than symbol (>) next to its file name.



To view or change the label decorations, select **Windows** → **Preferences**, expand **General** → **Appearance**, and select **Label Decorations**. By default, the **CVS** labels are selected (Figure 29-4).

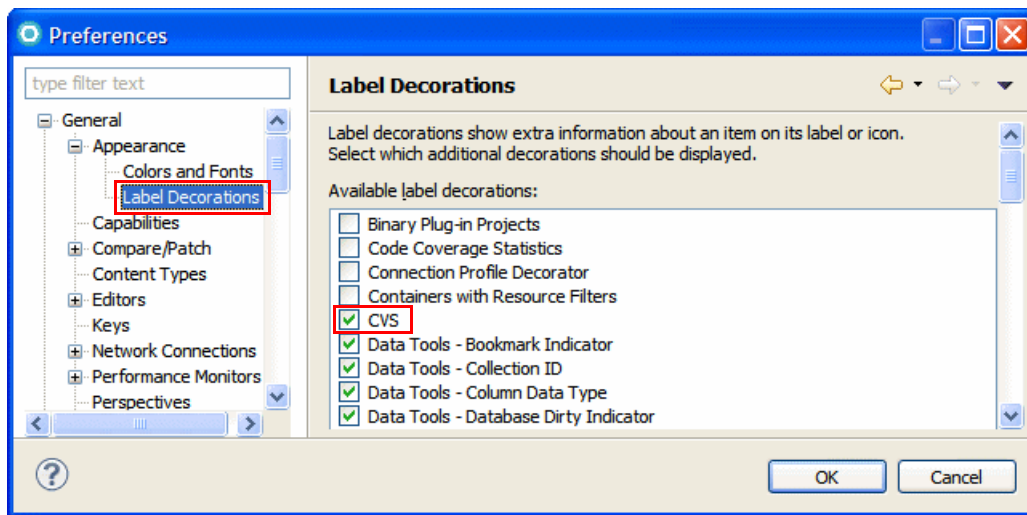


Figure 29-4 CVS Label Decoration preferences

## 29.3.2 File content

You can configure the file content of resources to be stored as either ASCII or binary. When working with a file extension that is not defined in the file content list that is stored in Rational Application Developer, files of this type are saved into the repository as binary, by default. When a resource is stored as a binary, CVS cannot show line-by-line comparisons between versions. However, files that have binary content cannot be stored as ASCII CVS files. After a file is created as one type, the type cannot be changed. Therefore, it is important to ensure that each file content type is configured correctly in Rational Application Developer before adding a new project to the repository.

To verify that a resource in the workspace is stored in the repository correctly, select **Windows** → **Preferences** and expand **Team** → **File Content** (Figure 29-5 on page 1542). Verify that the file extensions that you are using are present and stored in the repository as desired.

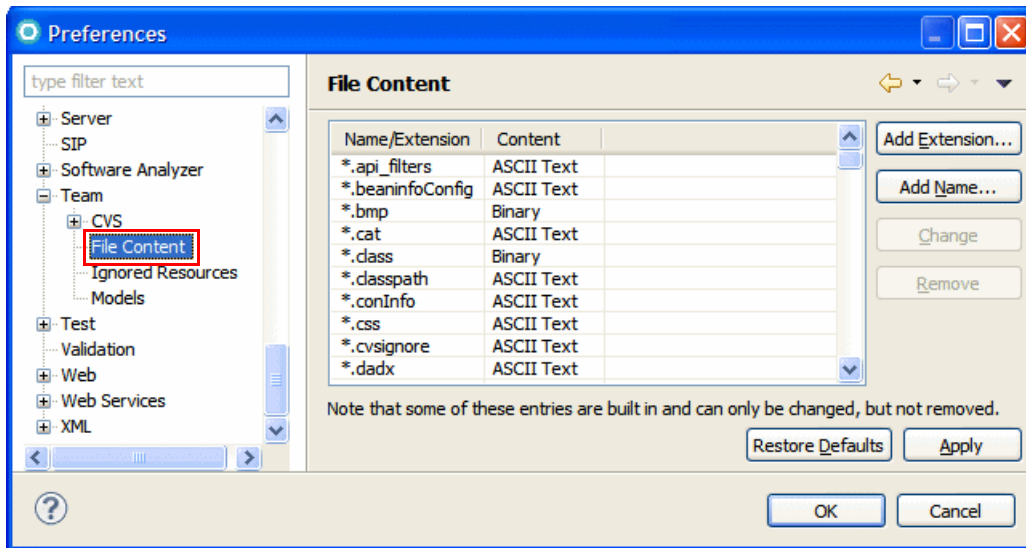


Figure 29-5 Team File Content preferences

If a particular file extension is not in the list, you must add this extension, unless the resource is stored in the default binary format. Rational Application Developer prompts you for the resource type when performing the first check-in (see Chapter 29-12, “Adding new file types at check-in” on page 1554) if it encounters a new type. Or, you can add the new type manually in the Preferences window.

A common file that is often supplied with a source code distribution is a `Makefile.mak` file, which is usually an ASCII file.

To add this file type extension (which is not present in this list), follow these steps:

1. Select **Windows** → **Preferences** and expand **Team** → **File Content**.
2. Click **Add Extension**.
3. Enter the extension name `mak` and click **OK**.
4. Find the extension in the list, and in the Content column, select **ASCII Text**.

**Tip:** You can also change the content by highlighting the extension and clicking **Change**. The setting toggles between ASCII Text and Binary.

5. Click **Apply** and then click **OK**.

### 29.3.3 Ignored resources

Usually, developers do not save resources that are created or changed dynamically through mechanisms, such as the compilation or builds, in the repository. These resources can include class files, executables, and Enterprise JavaBeans (EJB) stubs and implementation codes. Rational Application Developer stores a list of these resources that is ignored when performing CVS operations. To access this list, select **Windows** → **Preferences** and expand **Team** → **Ignored Resources**.

You can add resources to this list by specifying the pattern that will be ignored. The two wildcard characters are an asterisk (\*), which indicates a match of zero or many characters, and a question mark (?), which indicates a match of one character. For example, a pattern of `_EJS*.java` matches any file that begins with `_EJS`, has zero-to-many characters, and ends in `.java`.

In the following example, we explain how to add the filename pattern `*.tmp` to the ignored resources list:

1. Select **Windows** → **Preferences**.
2. Expand **Team** and select **Ignored Resources**.
3. Click **Add Pattern**. Add the pattern `*.tmp` and click **OK**.
4. Ensure that the resource (`*.tmp`) is selected and added to the Ignored Resources list (Figure 29-6 on page 1544). All `*.tmp` resources are now ignored by CVS in Rational Application Developer.

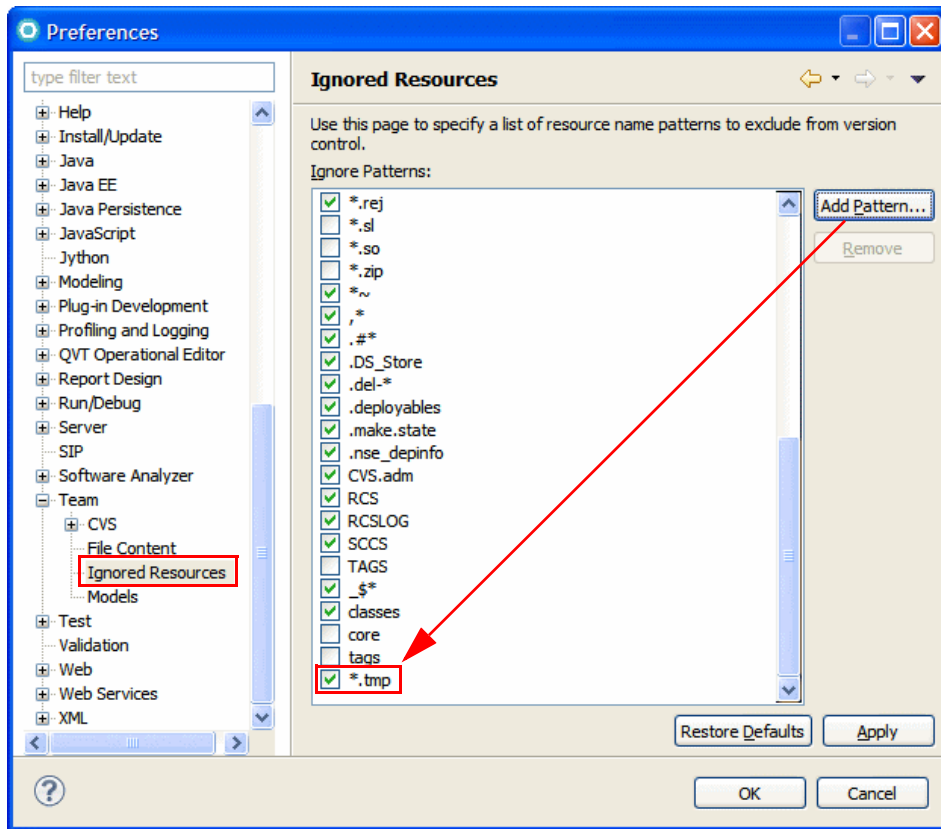


Figure 29-6 Resources that will be ignored when saving to the repository

To remove a pattern from the Ignored Resources list, select it and click **Remove**. To temporarily disable ignoring a file pattern, clear its check box in the list.

Additionally, you can use the following two facilities to exclude a file from version control:

- ▶ The “Resources marked as derived are automatically not checked into the CVS repository by Rational Application Developer” field

This field is set by builders in the Eclipse framework, such as the Java builder. To determine if a resource is derived, right-click the resource and select **Properties**, or look in the Properties view. The Derived field is shown under Info. It is also possible to change the Derived field value in the Properties window.

► Use of a .cvsignore file

This file contains a list of files or directories that must not be placed into the repository. CVS checks this file and does not add to CVS any files that are in this list. You can add a file to the list by right-clicking the file in the Enterprise Explorer view and selecting **Team** → **Add to .cvsignore**.

For more details about the syntax of .cvsignore, see the following web address:

<http://www.cvsnt.org/manual/html/cvsignore.html>

## 29.3.4 CVS-specific settings

The CVS settings in Rational Application Developer are extensive and therefore are difficult to cover here in full. Table 29-1 highlights the more important settings with short descriptions. For a complete description of the remaining settings, see the Rational Application Developer Help system.

Table 29-1 Categories of available CVS settings

Category	Window preferences	Description
General CVS Settings	<b>Team</b> → <b>CVS</b>	Settings for the behavior in communicating with CVS, handling the files and projects received from CVS, and prompting the user for certain activities.
Annotate	<b>Team</b> → <b>CVS</b> → <b>Annotate</b>	Switch on or off the annotating of binary files.
Comment Templates	<b>Team</b> → <b>CVS</b> → <b>Comment Templates</b>	Lets you create, edit, or remove comment templates that can be used while checking in a file.
Console	<b>Team</b> → <b>CVS</b> → <b>Console</b>	Various settings for the CVS console view, including a flag for whether to display CVS commands to the console.
Ext Connection Method	<b>Team</b> → <b>CVS</b> → <b>Ext Connection Method</b>	Settings to identify the SSH external program and associated parameters when using the ext protocol to communicate with the CVS server.
Label Decorations	<b>Team</b> → <b>CVS</b> → <b>Label Decorations</b>	Settings for how to display the CVS state of resources in Rational Application Developer.
Synchronize/Compare	<b>Team</b> → <b>CVS</b> → <b>Synchronize/Compare</b>	Various settings for comparison.

Category	Window preferences	Description
Update/ Merge	<b>Team</b> → <b>CVS</b> → <b>Update/Merge</b>	Settings for guiding the Rational Application Developer process when merging is required during synchronization.
Watch/Edit	<b>Team</b> → <b>CVS</b> → <b>Watch/Edit</b>	Settings for the CVS watch and edit functionality, which allows users to be informed (by email) when a file has been edited or committed by another user.

### 29.3.5 CVS keyword substitution

In addition to storing the history of changes to a given source code file in the CVS repository, you can store meta information (such as the author, date and time, revision, and change comments) in the contents of the file. Typically, a standard header template is configured in Rational Application Developer, which is then applied to each file when CVS operations (usually check-in and checkout) are performed. The template can include a set of CVS keywords inside a heading. These keywords are expanded when a file is checked in or out, and this process is known as *keyword expansion*.

Keyword expansion is an effective mechanism for developers to quickly identify the version of a resource in the repository versus the version of a resource that a user has checked out locally on the user's workspace.

Rational Application Developer, by default, has the keyword substitution set to *ASCII with keyword expansion (-kkv)* under the selection **Windows** → **Preferences** → **Team** → **CVS** and the **File and Folders** tab (Figure 29-7 on page 1547). This setting expands the keyword substitution that is based on the interpretation by CVS and is performed wherever the keywords are located in the file.

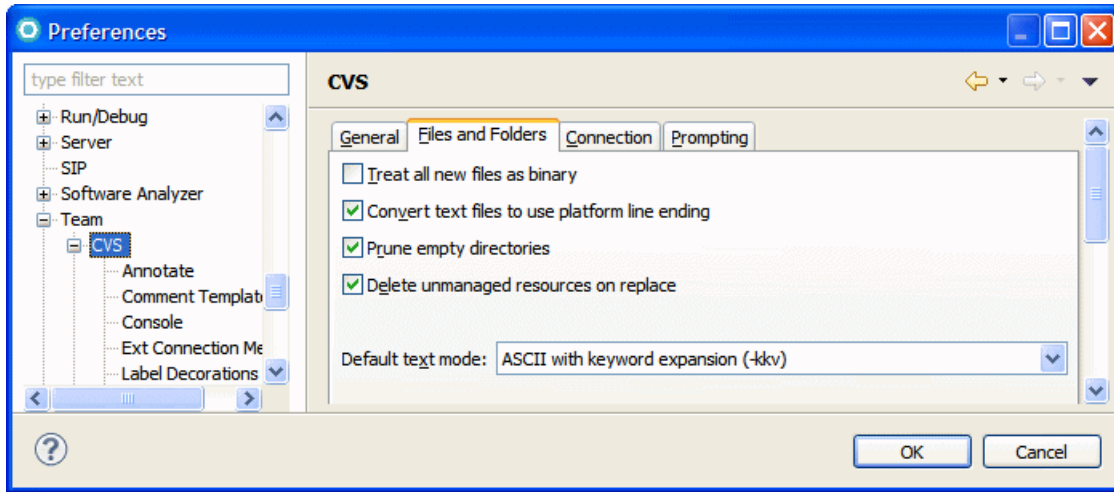


Figure 29-7 CVS keyword expansion setting

Table 29-2 lists several of the available keywords (which are case-sensitive).

Table 29-2 CVS keywords

Keyword	Description
\$\$Author\$	Expands to the name of the author of the change in the file, for example: \$Author: itsodev \$
\$\$Date\$	Expands to the date and time of the change in Coordinated Universal Time (UTC), for example: \$Date: 2008/10/14 18:21:32 \$
\$\$Header\$	Contains the CVS file in repository, revision, date (in UTC), author, state, and locker, for example: \$Header: /rep7835/XMLExample/.project,v 1.1 2008/10/14 18:21:32 itsodev Exp itso \$
\$\$Id\$	Like \$Header\$, except without the full path of the CVS file, for example: \$Id: .project,v 1.1 2008/10/14 18:21:32 itsodev Exp itso \$
\$\$Log\$	The log message of this revision. This log message does not get replaced, but it gets appended to the existing log messages.
\$\$Name\$	Expands to the name of the <i>sticky tag</i> , which is a file that is retrieved by date or revision tags, for example: \$Name: version_1_3 \$
\$Revision\$	Expands to the revision number of the file, for example: \$Revision: 1.1 \$
\$Source\$	Expands to the full path of the RCS file in the repository, for example: \$Source: /rep7835/XMLExample/.project,v \$

To ensure consistency between multiple users working on a team, define a standard header for all Java source files. Example 29-1 shows keywords that are used in Java.

*Example 29-1 CVS keywords that are used in Java*

---

```
/**
 * Class comment goes here.
 *
 * <pre>
 * Date $$Date$$
 * Id $$Id$$
 * </pre>
 * @author $$Author$$
 * @version $$Revision$$
 */
```

---

To ensure consistency across all of the files that are created, each user needs to cut and paste this header into the user's files. Fortunately, Rational Application Developer offers a means to ensure this consistency.

To set up a standard template, follow these steps:

1. Select **Windows** → **Preferences** → **Java** → **Code Style** → **Code Templates**.
2. Expand **Comments** → **Files** and click **Edit**.
3. In the Edit Template window (Figure 29-8 on page 1549), cut and paste or type the comment header that you require. Click **OK** to complete the editing.



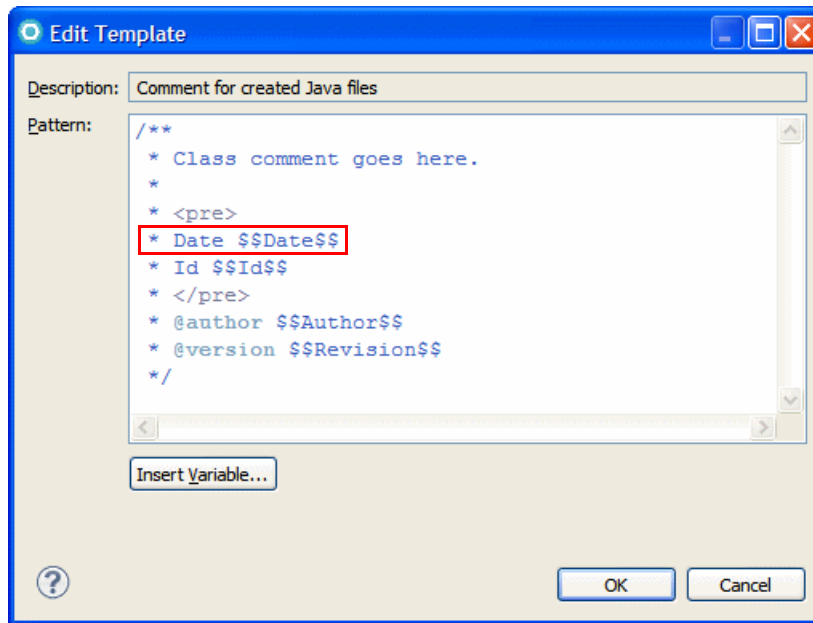


Figure 29-8 Setting up a common code template for Java files

4. Click **Apply** and then click **OK**.

**Double dollar sign:** The double dollar sign (\$\$) is required, because Rational Application Developer treats a single dollar sign (\$) as one of its own variables. The double dollar sign (\$\$) is used as a means of escaping the single dollar sign so that it can be post-processed by CVS.

This task sets up a standard CVS template. The next time that a new class is created, checked in, and then checked out, the header is displayed (Example 29-2).

Example 29-2 Contents of the Java file after the check-in and checkout from CVS

```
/**
 * class comment goes here.
 *
 * <pre>
 * Date $Date: 2010/10/29 21:17:32 $
 * Id $Id: $Id: Example.java,v 1.1 2010/10/29 21:17:32 itsodev Exp itsodev $
 * </pre>
 * @author $Author: itsodev $
```

```
* @version $Revision: 1.1 $
*/
```

---

## 29.4 Development scenario

**Simulated developer systems:** The example in this chapter calls for two simulated developer systems. For demonstration purposes, you can simulate having two systems by having two workspaces on the same machine.

To show you how to work with CVS in Rational Application Developer, we follow a simple but typical development scenario in Table 29-3. Two developers, *cvsuser1* and *cvsuser2*, work together to create a servlet `ServletA` and a view bean `View1`.

Table 29-3 Sample development scenario

Step	Developer 1 ( <i>cvsuser1</i> )	Developer 2 ( <i>cvsuser2</i> )
1	Creates a new dynamic web project, <code>RAD8CVSGuide</code> , and a servlet, <code>ServletA</code> , in it and adds it to the version control and the repository.	N/A
2	<b>2b:</b> Updates the servlet <code>ServletA</code> .	<b>2a:</b> Imports the <code>RAD8CVSGuide</code> CVS module as a workbench project. Creates a view bean <code>View1</code> , adds it to the version control, and synchronizes the project with the repository.
3	<b>3a:</b> Synchronizes the project with the repository to commit the changes to the repository (servlet) and receives changes from the repository (view bean).	<b>3b:</b> Synchronizes the project with the workspace to receive the servlet.
4	<b>4a:</b> Continues changing and updating the servlet. Synchronizes the project with the repository to commit changes to the repository and merges changes. <b>4c:</b> Synchronizes with the repository to pick up the merged servlet.	<b>4a:</b> Begins changes to the servlet in parallel with Developer 1. <b>4b:</b> Synchronizes the project after <i>cvsuser1</i> has committed and must merge code from the workspace and the CVS repository.
5	Assigns a version number to the project.	N/A

Steps 1 through 3 are serial development with no parallel work on the same file being done. In steps 4 and 5, both developers work in parallel, resulting in conflicts. These conflicts are resolved by using the CVS tools in Rational Application Developer.

In the following sections, we perform each of the steps and explain the team actions in detail.

**Important:** Always work in the workspace of the correct user (*cvsusern*).

## 29.4.1 Creating and sharing the project (step 1, cvsuser1)

Rational Application Developer offers a perspective that is specifically designed for viewing the contents of CVS servers, which is called the *CVS Repository Exploring perspective*.

### Adding a CVS repository

For this section, ensure that you have installed and implemented the CVSNT server and completed the steps in 29.2.3, “Accessing the CVS repository” on page 1538.

The CVS Repositories view now contains the repository location (Figure 29-9).

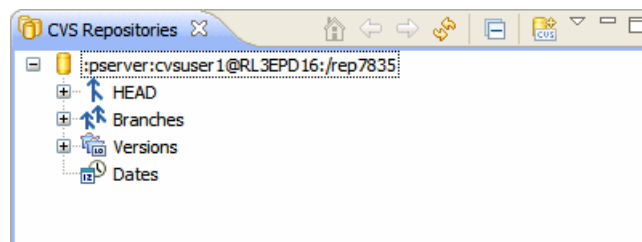


Figure 29-9 CVS Repositories view

Expanding a location in the CVS Repositories view reveals branches and versions. A special branch, which is called HEAD, is shown outside the main Branches folder because of its importance. HEAD is the main integration branch, holding the project’s current development state.

You can use the CVS Repositories view to check out repository resources as projects on the workbench. You can also configure branches and versions, view resource histories, and compare resource versions and revisions.

First, you must create and share a project before full use can be made of the repository.

## Creating a project and servlet

To create a project and a servlet, follow these steps:

1. Switch to the Web perspective and create a new dynamic web project by selecting **File** → **New** → **Dynamic Web Project**.
2. For the project name, type RAD8CVSGuide, keep the default values for all the other fields and click **Finish**.
3. In the Enterprise Explorer, right-click **RAD8CVSGuide** and select **New** → **Servlet**.
4. In the Create Servlet window (Figure 29-10), complete the following steps:
  - a. For the Java package, type `itso.rad8.teamcv.servlet`.
  - b. For the Class name, type `ServletA`.
  - c. Click **Finish**.

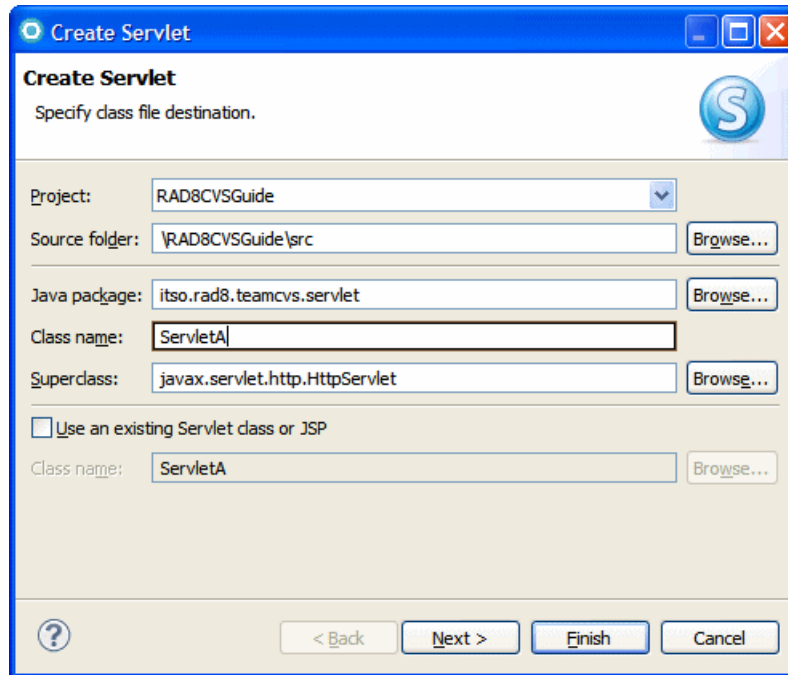


Figure 29-10 Create Servlet wizard

## Adding the project to the repository

To add the web project source code to the repository, follow these steps:

1. In the Enterprise Explorer, right-click **RAD8CVSGuide** and select **Team** → **Share Project**.
2. In the Share Project window, select **CVS** and click **Next**.

3. In the Share Project with CVS Repository window, select **Use existing repository location**, select the repository, and click **Next**.
4. In the Enter Module Name window, select **Use project name as module name** (default) and click **Next**. A status window might open as the resources are added to the repository.
5. After the Share Project Resources window (Figure 29-11) opens, listing the resources to be added, click **Finish**.

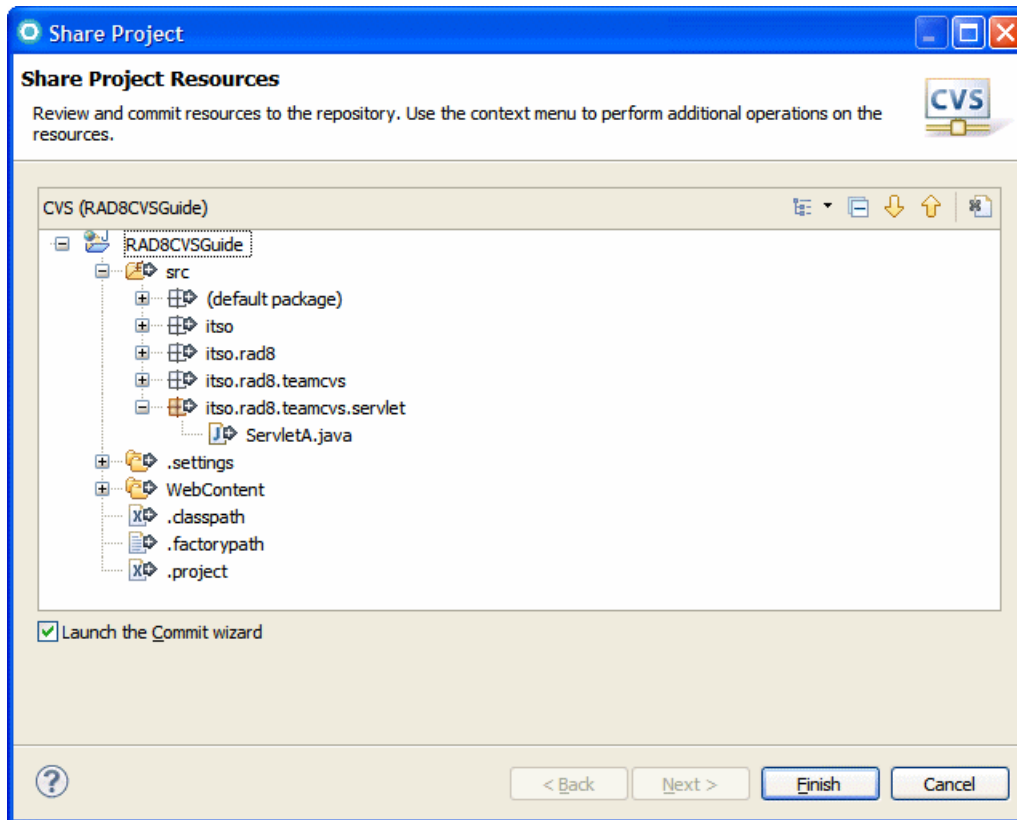


Figure 29-11 Verifying the resources that are added under CVS revision control

A new window (Figure 29-12 on page 1554) opens, indicating that new file types are being added, which are not configured as types in the Rational Application Developer preferences. The files are meta information of the project and must be ASCII Text.

6. Set these file types to **ASCII Text** and click **Next**.

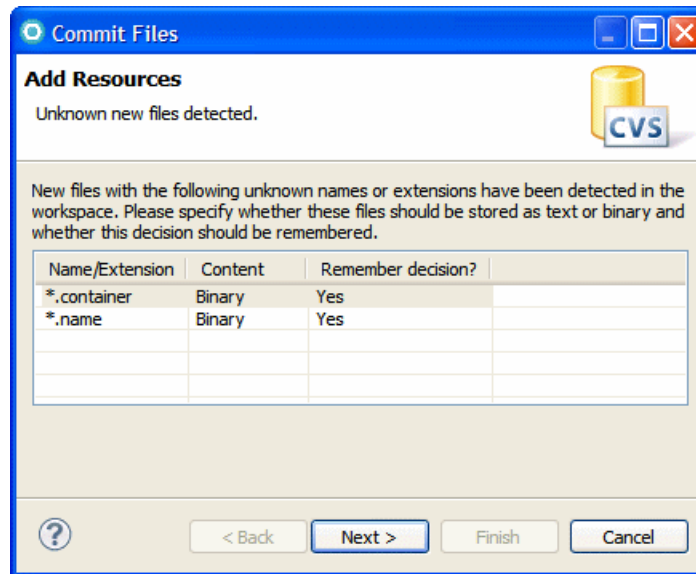


Figure 29-12 Adding new file types at check-in

7. In the Commit Files window, in the Comment field, type Initial version and click **Finish**.

A status window opens, showing the progress as the initial versions of a new project are checked into the repository. Because RAD8CVSGuide is a relatively small project, this process only takes a few seconds.

8. For larger projects for which the initial check-in might take more time, click **Run Background** to continue working in the workspace while the CVS check-in process completes.

After completing this task, the RAD8CVSGuide project is checked into the repository and is available for other developers to use.

## 29.4.2 Adding a shared project to the workspace (step 2a, cvsuser2)

The purpose of any source code repository is for multiple developers to be able to work as a team on the same project. The RAD8CVSGuide project has been created in one developer's workspace and is shared using CVS. Now the second developer needs to attach to the CVS repository and check out a copy.

Complete the following steps from a second developer's workspace:

1. Add the CVS repository location to the workspace using the CVS Repositories view in the CVS Repository Exploring perspective, as explained in "Adding a CVS repository" on page 1551 (Figure 29-13).

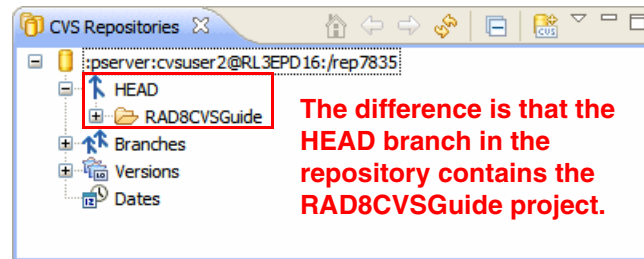


Figure 29-13 CVS Repository view showing the HEAD branch

2. Right-click the **RAD8CVSGuide** module and select **Check Out**. The current project in the HEAD branch is checked out to the workspace.

### Developing the view bean

Now that both developers have exactly the same synchronized HEAD branch of the `RAD8CVSGuide` project on their workspaces, the second developer must create the view bean `View1`:

1. Open the Web perspective.
2. In the Enterprise Explorer, right-click **RAD8CVSGuide** and select **New** → **Class**.
3. In the New Class window (Figure 29-14 on page 1556), complete the following actions:
  - a. For the Package, type `itso.rad8.teamcvs.bean`.
  - b. For the Name, type `View1`.
  - c. Select **Generate comments** and click **Finish**.

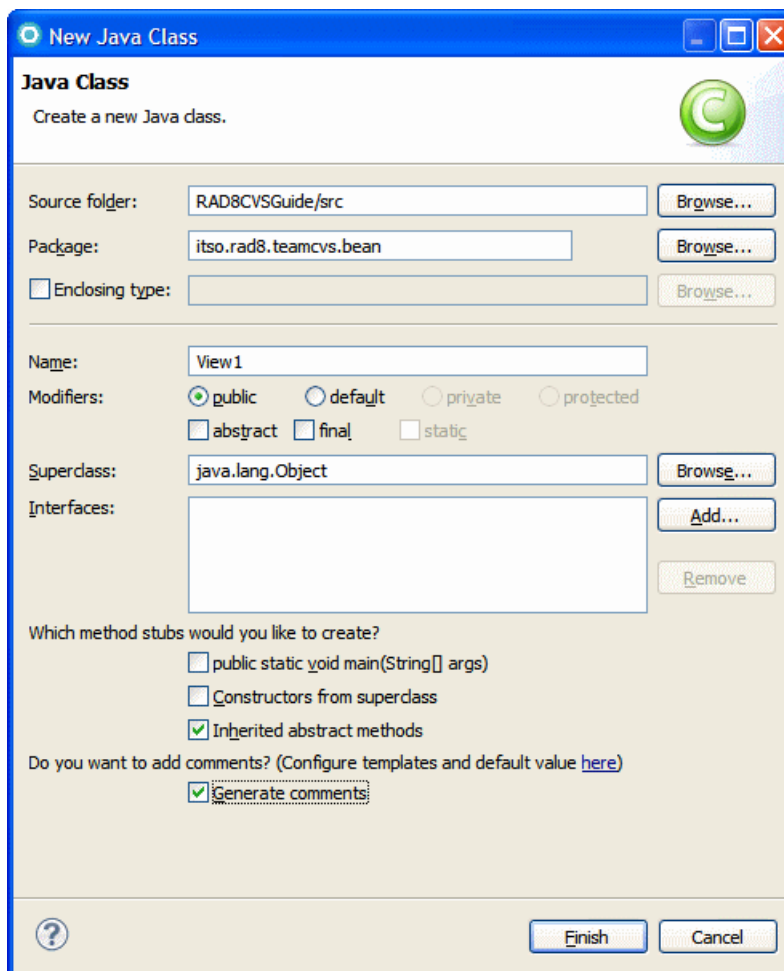


Figure 29-14 Creating the View1 view bean

4. Add the highlighted two private attributes in the View1 class (Example 29-3).

*Example 29-3 Adding two private attributes to the View1 class*

---

```
package itso.rad8.teamcvcs.bean;

public class View1 {
 private int count;
 private String message;
}
```

---



- In the Java Editor, right-click and select **Source** → **Generate Getters and Setters**.
- In the Generate Getters and Setters window (Figure 29-15), click **Select All**. For the Access modifier, verify that **public** is selected. Click **OK**.

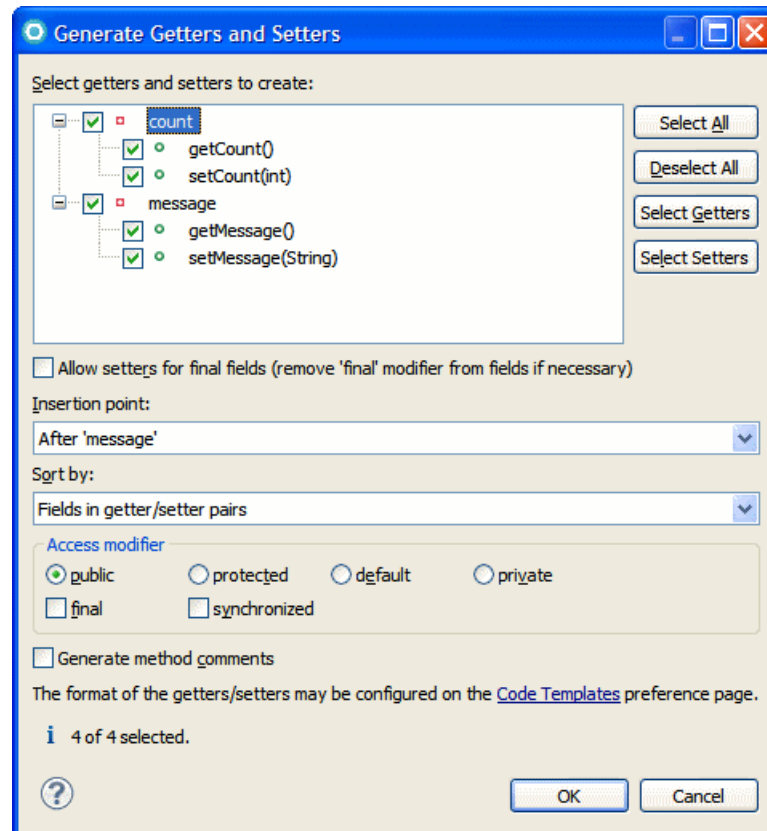


Figure 29-15 Creating setters and getters for class View1

- Save and close the View1 class.

**Tip:** In the Enterprise Explorer view, the greater than sign (>) in front of a resource name means that the particular resource is not synchronized with the repository. The question mark symbol (?) indicates that the file is not in the repository. You can use these visual cues to determine when a project requires synchronization.

## Synchronizing with the repository

To update the repository with these changes, follow these steps:

1. Right-click **RAD8CVSGuide** and select **Team** → **Synchronize with Repository**.
2. When you are prompted to change to the Team Synchronizing perspective, select **Remember my decision** and click **Yes**. The project is compared with the repository, and the differences are displayed in the Synchronize view (Figure 29-16). Along the bottom of the workspace is the count of the number of incoming, outgoing, and conflicting changes.

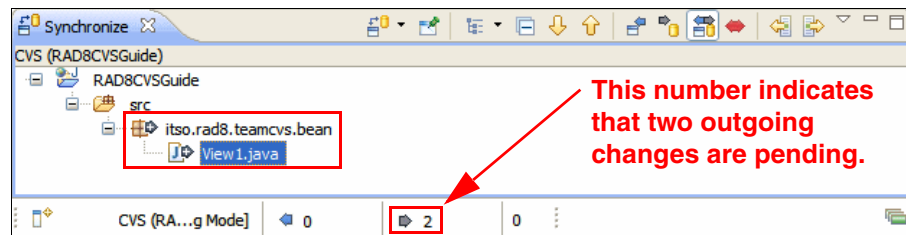


Figure 29-16 Synchronizing RAD8CVSGuide after creating the viewbean View1

In the Synchronize view, you can update resources in the workbench with newer content from the repository (*incoming*), commit resources from the workbench to the repository (*outgoing*), and resolve conflicts that might occur in the process.

The arrow icons with a plus sign (➕) indicate that the files do not exist in the repository. Because the `itso.rad8.teamcvcs.bean` package and the `View1.java` class are new and have not yet been checked in, they show the plus sign.

3. To add these new resources to version control, in the Synchronize view, right-click **RAD8CVSGuide** and select **Commit**.
4. In the Commit Files window (Figure 29-17 on page 1559), for the commit comment, type `View bean itso.rad8.teamcvcs.bean.View1 added`. Check the files that are shown in bottom half of the window to ensure that the changes are as expected. Click **Finish** and the changes are committed to the repository.

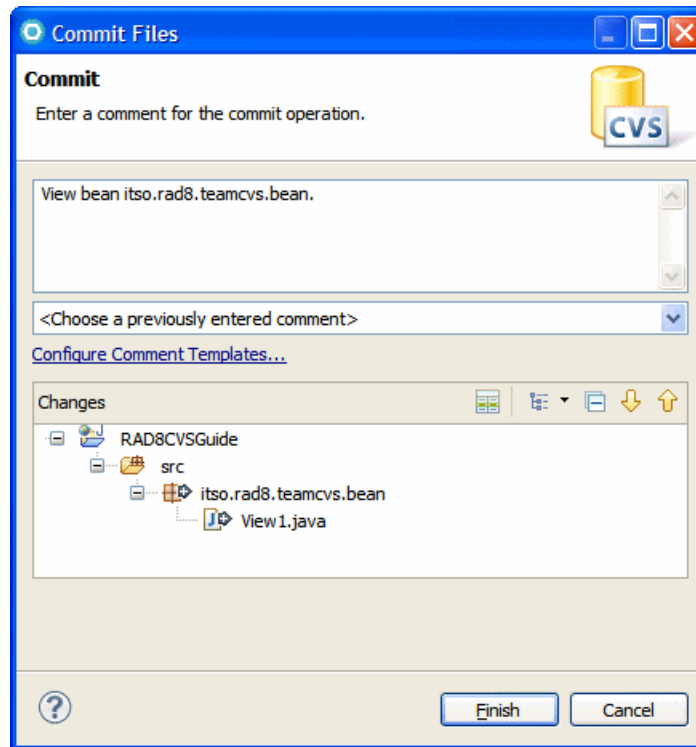


Figure 29-17 Verifying committing resources into the repository

**Commit comments:** The user can specify common text within commit comments, and the user can reuse the same format for all commits. To create commit comments, click **Configure Comment Template** to create a comment and then select it from the drop-down list in the window.

### 29.4.3 Modifying the servlet (step 2b, cvsuser1)

While the actions in “29.4.2, “Adding a shared project to the workspace (step 2a, cvsuser2)” on page 1554” occur, our original user, cvsuser1, develops the servlet further.

In the first workspace that is created for cvsuser1, follow these steps:

1. In the Enterprise Explorer, open **ServletA.java** (in RAD8CVSGuide/Java Resources/src/itso.rad8.teamcvcs.servlet).
2. Create a static attribute called `totalCount` of type `int` and initialized to zero, as highlighted in Example 29-4 on page 1560. Save and close the servlet.

*Example 29-4 A static attribute for ServletA*

```
package itso.rad8.teamcvs.servlet;

.....

public class ServletA extends HttpServlet {
 private static final long serialVersionUID = 1L;
 private static int totalCount = 0;

 public ServletA() {
 super();
 }

}
```

### 29.4.4 Synchronizing with the repository (step 3a, cvsuser1)

User `cvsuser1` now synchronizes with the repository and receives the changes of `cvsuser2` (adding the `View1` bean) and provides the opportunity to check in the changes that have been made to `ServletA`. Follow these steps:

1. In the Enterprise Explorer, right-click the **RAD8CVSGuide** project and select **Team** → **Synchronize with Repository**.
2. On the Confirm Open Perspective dialog window, select **Remember my decision** and click **Yes**.
3. In the Synchronize view, expand the **RAD8CVSGuide** → **src** trees to view the changes (Figure 29-18).

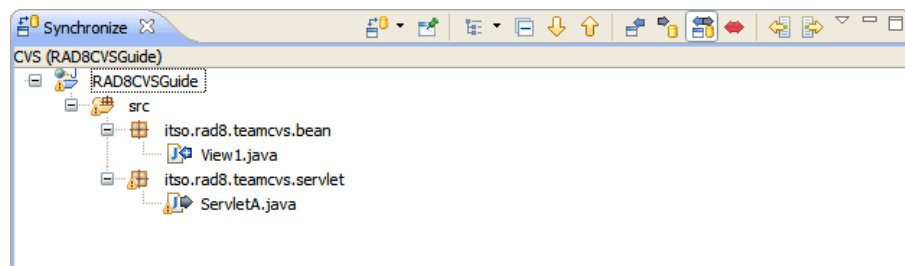




Figure 29-18 User `cvsuser1` merging with CVS repository

**Symbols:** The  symbol that is shown in Figure 29-18 indicates that an existing resource differs from the resource that is in the repository. The  symbol indicates that a new resource is in the repository that does not exist on the local workspace.

4. To obtain updated resources from the CVS repository, right-click the **RAD8CVSGuide** project and select **Update** to bring a copy of the `View1.java` file into this workspace.
5. Verify that the changes do not cause problems with existing resources in the local workspace, by checking the Problems view. In this case, no problems exist. Right-click the **RAD8CVSGuide** project and select **Commit**.
6. In the Commit window (Figure 29-19), add the comment `Static variable totalCount added to ServletA` and click **Finish** to check the changes that have been made to `ServletA` into the repository.

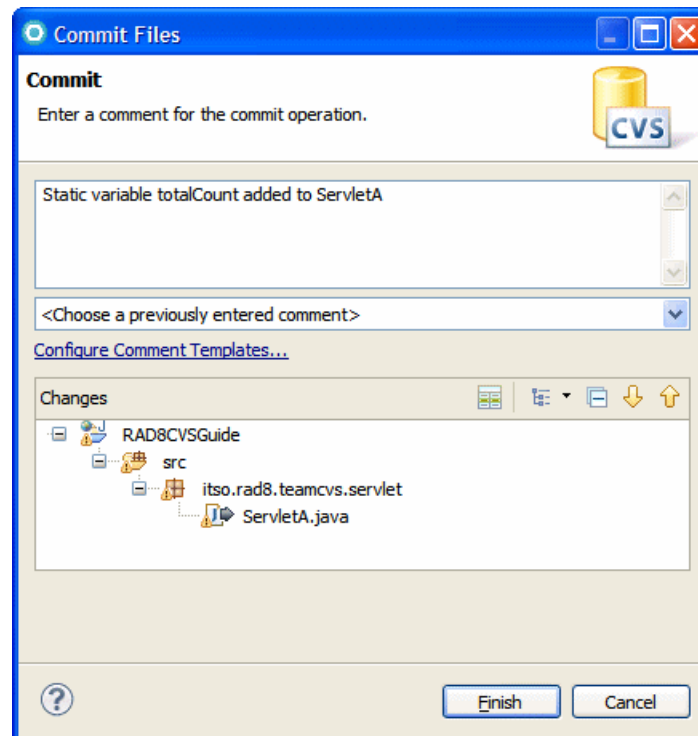


Figure 29-19 Adding a comment for the changes to `ServletA`

The repository now has the latest changes to the code from both developers. The user `cvuser1` is in sync with the repository. However, `cvuser2` has not yet received the changes to `ServletA`.

### 29.4.5 Synchronizing with the repository (step 3b, `cvuser2`)

The second workspace that is used by `cvuser2` must also synchronize (update) with the repository and receive the changes of `cvuser1`. Synchronization brings the changes that have been made to `ServletA` into the workspace and ensures that both workspaces are completely up-to-date.

### 29.4.6 Parallel development (step 4, `cvuser1` and `cvuser2`)

The previous steps highlight development and repository synchronization with two people working on two parts of a project. The steps highlight the need to synchronize between each phase in development before further work is performed.

In the following scenario, we demonstrate how two developers work simultaneously on the same file, starting from the same revision. We explain the sequence of events for each user in the following sections. The time line in Figure 29-20 summarizes the events.

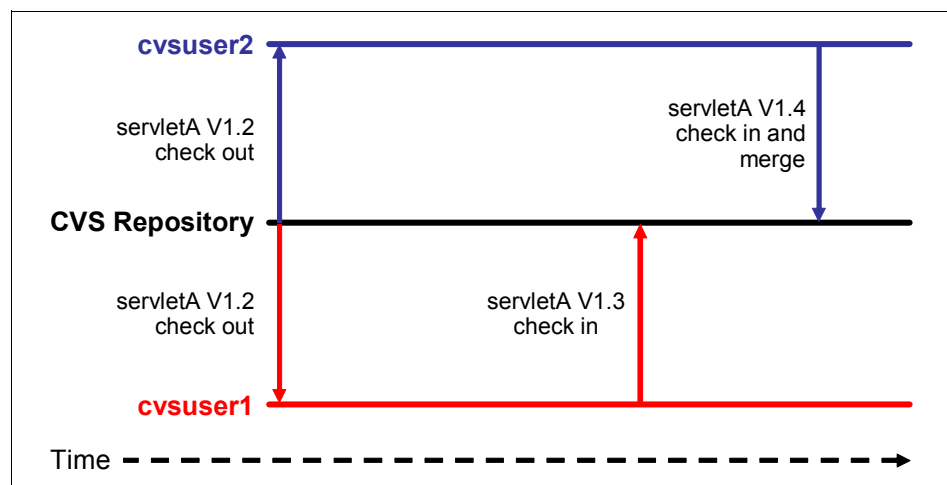


Figure 29-20 Parallel concurrent development by multiple developers

## User cvsuser1 updating and committing changes

In this scenario, user cvsuser1 modifies the doPost method to log information for an attribute. In the following procedure, we demonstrate how to synchronize the source code and commit the changes to CVS:

1. In the Enterprise Explorer, open **ServletA** (in RAD8CVSGuide/Java Resources/src/itso.rad8.teamcvs.servlet).
2. Generate the Getter and Setter method for the static variable totalCount.
3. Navigate to the doPost method by scrolling down the file and adding the code to count the number of post requests received:

```
protected void doPost(HttpServletRequest request,
 HttpServletResponse response) throws ServletException,
 IOException {
 totalCount = totalCount + 1;
 System.out.println("The total number of requests is: " +
 totalCount);
}
```

4. Save and close the file.
5. Synchronize the project with the repository by right-clicking and selecting **Team** → **Synchronize with Repository**.
6. Fully expand the tree in the Synchronize view. The servlet is the only change.
7. Right-click the project and select **Commit**. Add the comment doPost method implemented by cvsuser1 and click **Finish** to commit.

The developer cvsuser1 has now completed the task of adding code into the servlet. The other developers in the team can pick up the changes now.

## User cvsuser2 updating and committing changes

To complete the scenario, the second developer also makes changes to the doPost method of ServletA. The developer makes the changes in the workspace of cvsuser2 and assumes that the workspace was synchronized before this scenario was started.

To make the changes, follow these steps:

1. In Enterprise Explorer, open **ServletA**.
2. Generate the Getter and Setter method for the static variable totalCount.
3. Add the highlighted code that is shown in Example 29-5 on page 1564 to ServletA. An instance variable for the view bean is added and the doPost method gets implemented. Save and close the file.

*Example 29-5 cvsuser2 completing ServletA*

---

```
package itso.rad8.teamcvs.servlet;
.....

public class ServletA extends HttpServlet {
 private static final long serialVersionUID = 1L;
 private static int totalCount = 0;
 private View1 myViewBean;

 protected void doPost(HttpServletRequest request,
 HttpServletResponse response) throws ServletException,
 IOException {
 this.myViewBean = new View1();
 this.myViewBean.setCount(ServletA.getTotalCount());

 switch (ServletA.getTotalCount()) {
 case (0):
 System.out.println("No hits on page");
 break;
 case (1):
 System.out.println("One hit on page");
 break;
 default:
 System.out.println("Hits are greater than one");
 }
 }

}
```

---

4. Synchronize with the repository by right-clicking the **RAD8CVSGuide** project and selecting **Team** → **Synchronize with Repository**.
5. Expand the tree in the Synchronize view to see the changes (Figure 29-21 on page 1565).



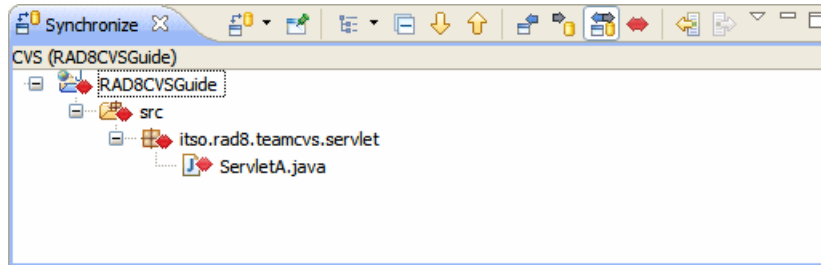




Figure 29-21 Synchronize view showing conflicting changes

**Conflicting changes:** The  symbol indicates that the file has conflicting changes that require merging.

6. Double-click **ServletA.java** to see the changes (Figure 29-22 on page 1566):
  - The left side shows the changes that were made by the current user cvsuser2, and the right side shows the code in the repository (that was checked in by user cvsuser1).
  - You can use the arrow icons at the top () to move from change to change.
  - Black lines between the panes indicate identical blocks.
  - Red lines indicate changes (inserts or conflicts).
  - Red bars on the right pane indicate conflicts.

In this case, merging requires consolidation between the two developers to determine the best solution. In our example, we assume that the changes in the repository (right side) must be placed sequentially before the changes performed by cvsuser2 (left side).

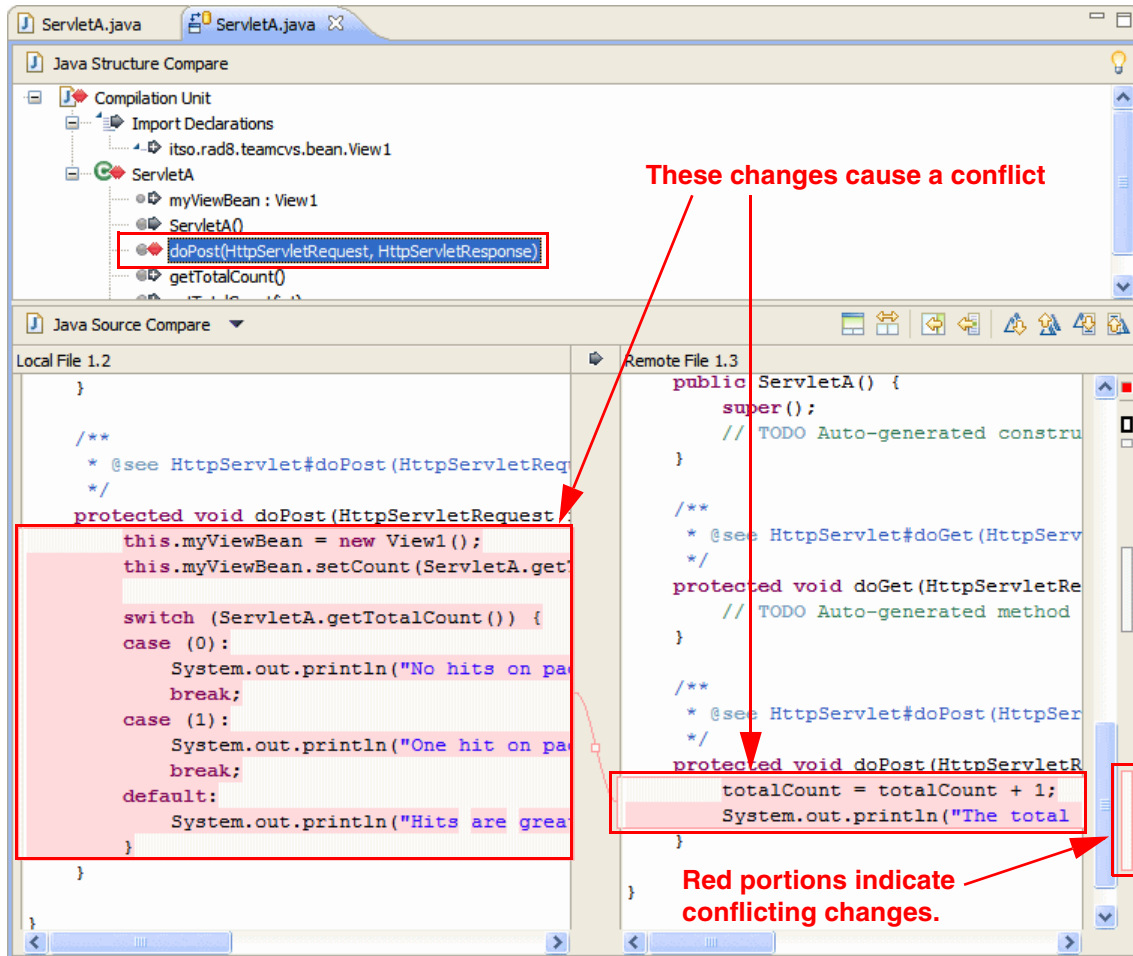



Figure 29-22 The changes between the local and remote repository

7. Double-click the **doPost** method in the Java Structure Compare view. Follow these steps:
  - a. Click the **Copy Current Change from Right to Left** icon () , which places the change of the conflicting section in the right panel to the bottom of the section in the left panel (Figure 29-23 on page 1567).
  - b. In the left pane, highlight the two lines of code that were added and move them to the correct location in the method.

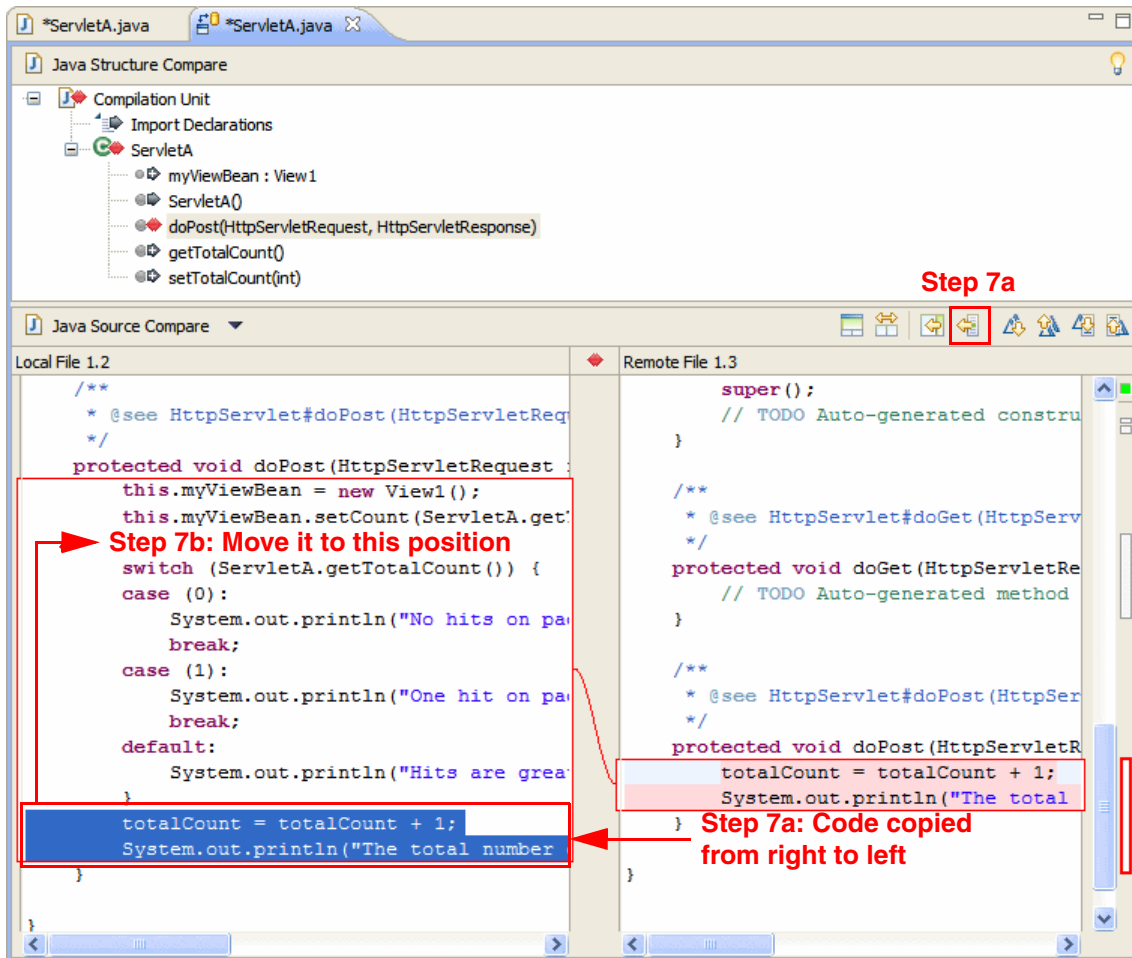


Figure 29-23 Merging changes from right to left

8. Verify that the code is exactly as agreed to by the developers and save the new merged change by selecting **File** → **Save**.
9. Resynchronize the file by selecting **Team** → **Synchronize With Repository**.
10. In the Synchronize view, verify that the changes are correct. Right-click **ServletA.java** and select **Mark as Merged**. Then right-click **ServletA.java** again and select **Commit**.
11. In the Commit window, enter the comment `ServletA changed and doPost method merged with cvsuser1`.

This operation creates a revision of the file, revision 1.4, which contains the merged changes from users cvsuser1 and cvsuser2. This is the case even though both developers originally checked out revision 1.2.

### User cvsuser1 synchronizing

The workspace for cvsuser1 must also be synchronized with the repository at this stage to pick up the merged code of ServletA.

## 29.4.7 Creating a version (step 5, cvsuser1)

Now that the changes for both users are committed and cvsuser1 has synchronized with the repository, we want to create a version to milestone our work. Perform the following steps in the workspace of cvsuser1:

1. Right-click **RAD8CVSGuide** and select **Team** → **Tag as Version**.
2. In the Tag Resources window (Figure 29-24), for the tag version, type `SERVLET_BASELINE` and click **OK**.

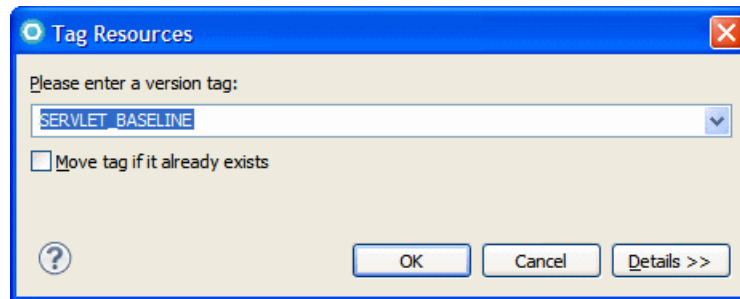


Figure 29-24 Tagging the project as a version

3. Verify that the tag has been performed by switching to the CVS Repositories Exploring perspective and expand **Versions** (Figure 29-25).

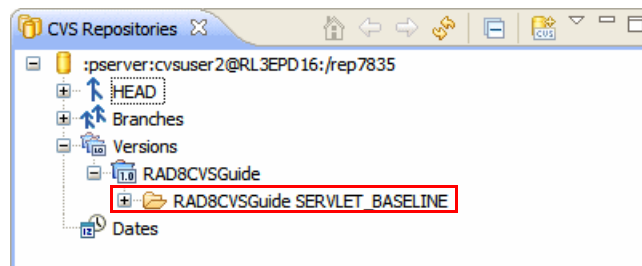


Figure 29-25 CVS Repositories view showing a new project revision

## 29.5 CVS resource history

Within Rational Application Developer, a developer can view the resource history of any file in a shared project in the CVS resource History view. This view shows a list of all the revisions of a resource in the repository. From this view, you can also compare two revisions, revert the existing workspace file to a previous revision, or open an editor to show the contents of a revision.

To understand how this feature works, in the Enterprise Explorer, right-click **ServletA.java** and select **Team** → **Show History**. The History view (Figure 29-26) opens.

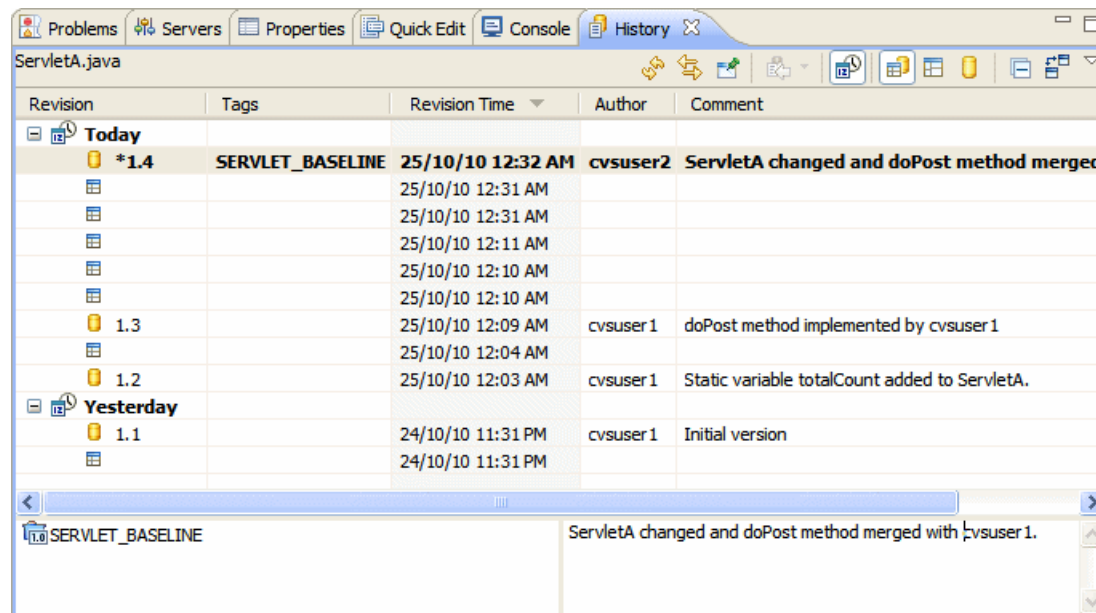


Figure 29-26 CVS History view for ServletA.java

The CVS resource history shows the columns that are listed in Table 29-4.

Table 29-4 CVS resource history terminology

Column	Description
Revision	This column shows the revision number of each version of the file in the repository. An asterisk (*) indicates that this version is the current version in the workspace.
Tags	This column shows any tags that have been associated with the revision.

Column	Description
Revision Time	This column shows the date and time when the revision was created in the repository.
Author	This column shows the name of the user who created and checked in the revision into the repository.
Comment	This column shows the comment (if any) that was supplied for this revision at the time that the revision was committed.

The following icons and features are available at the top of the History view:

**Refresh** 

This icon refreshes the history that is shown for the resource that is currently being shown.

**Link Editor with Selection** 

This icon is a toggle switch that automatically shows the history of the resource currently being shown in the main editor.

**Pin the History View** 

This icon locks the History view into showing only the currently selected resource's history. When this option is toggled on, the "Link Editor with Selection" icon is automatically switched off.

**Group Revisions by date** 

This icon changes the view to show the revisions ordered by date rather than by logical revision number. It is also possible to order the items in the History view by clicking the column headers.

**Local Revisions, Local and Remote Revisions, and Remote Revisions**



These icons provide three options for the type of revisions to show for the selected resource.

**Compare Mode** 

If this toggle is on, double-clicking a line in the history view shows a comparison between the selected repository file and the file in the workspace. If this option is switched off, double-clicking shows the contents of a file revision.

**Filters**

This feature filters the History view by author, date, or text within the check-in comments. This feature is available from the drop-down menu in the History view.

## 29.6 Comparisons in CVS

Often, developers have to view the changes that have been made to a file and in a certain revision. Rational Application Developer provides a mechanism to graphically display two revisions of a file and their differences. Two types of comparisons are possible. Users can compare the version in their workspace with any version in the CVS repository, or any two files in the CVS repository can be compared with each other.

The History view provides these mechanisms. The following scenario illustrates how to compare revisions.

### 29.6.1 Comparing a workspace file with the repository

The user `cvsuser1` has Version 1.4 of the `ServletA` file in the workspace and wants to compare the differences between the current version and Version 1.1. Follow these steps:

1. In the Enterprise Explorer, right-click **ServletA.java** and select **Compare with** → **History**.
2. In the History view, double-click **revision 1.1**.

The Comparison Editor opens (Figure 29-27 on page 1572):

- ▶ The Java Structure Compare (top half) shows an outline view of the changes. This view includes attribute changes and shows the methods that have been changed.
- ▶ The Java Source Compare (bottom two panes) highlights the actual code differences. The left pane shows the revision in the workspace, and the right pane shows revision 1.1 from the repository.

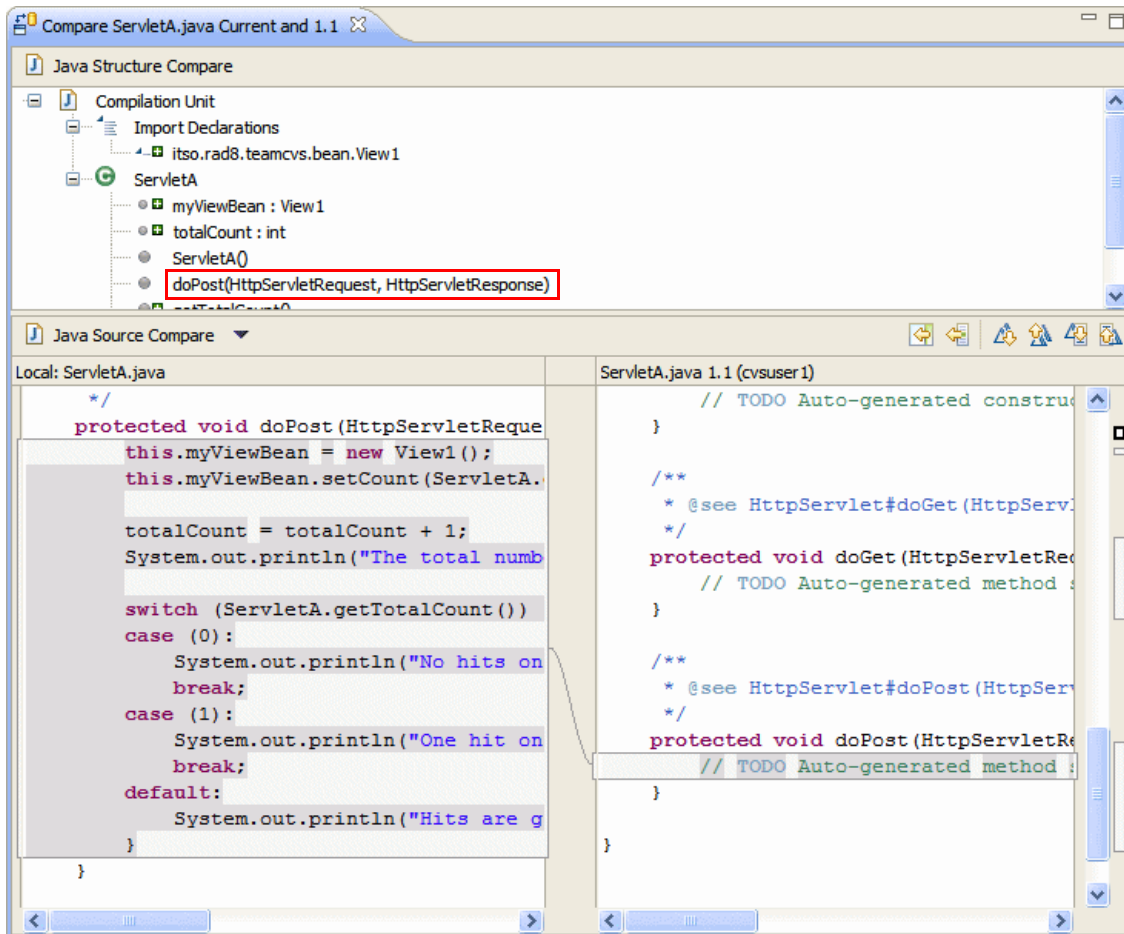


Figure 29-27 Comparison between current ServletA.java and revision 1.1


**Bars in the bottom pane:** The bars in the bottom pane on the right side indicate the parts of the file that differ. By clicking a bar, Rational Application Developer positions the panes to highlight the changes, which can assist you in quickly moving around large files with many changes.

## 29.6.2 Comparing two revisions in the repository

In this case, the developer wants to compare the differences between revision 1.1 and 1.3 in the repository of the ServletA file. However, version 1.4 is in the workspace, and the developer does not want to remove it.



To compare these two files, follow these steps:

1. Open the CVS resource history using the procedure in 29.5, “CVS resource history” on page 1569, which shows the History view in Figure 29-28.
2. Click the  icon to only see remote revisions.
3. Select the row of the first revision to compare, for example, revision 1.1. Then while pressing Ctrl, select the row of the second version, which is 1.3.
4. Right-click, ensuring that the two revisions remain highlighted, and select **Compare With Each Other** (Figure 29-28).

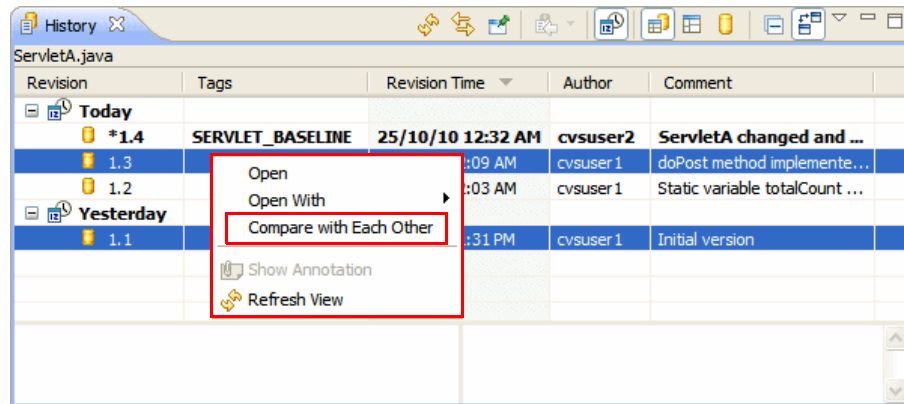


Figure 29-28 Highlighting the two versions to compare

The result is displayed, as shown in Figure 29-29. The higher version is always displayed in the left pane, and the lower version is always displayed in the right pane.

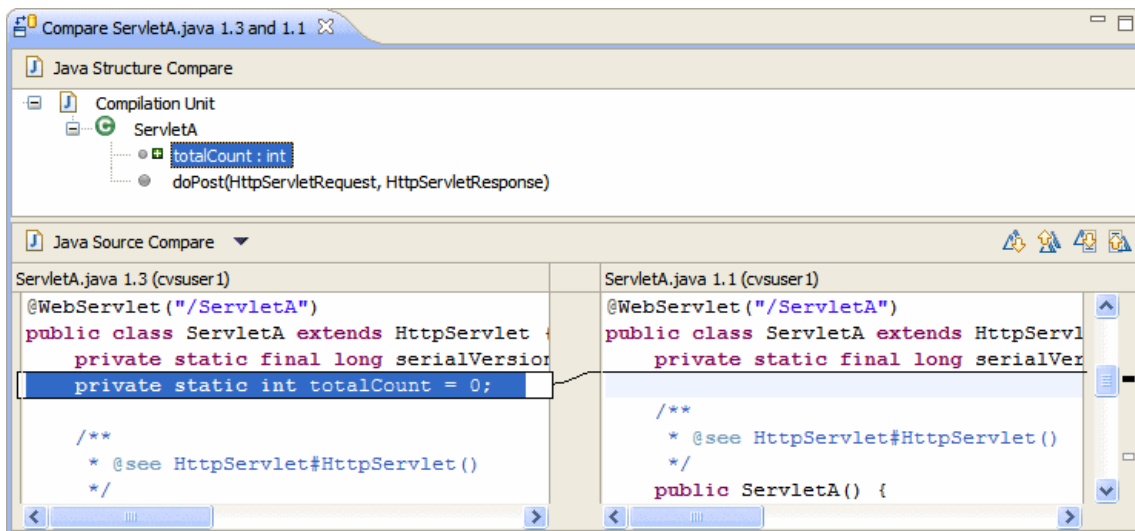


Figure 29-29 Comparisons of two revisions from the repository

## 29.7 Annotations in CVS

With the Annotations view, a user can view all the changes that have been performed on a particular file in a single combination of workspace views. The Annotations view shows which lines were changed in particular revisions, the author that is responsible for the change, when the file was changed, and the change description that was entered at the time. By viewing this information across all revisions of a file and in the same set of connected views, developers quickly can determine the origin of the changes and the explanations behind the changes.

To demonstrate annotations, we go back to our example of looking at ServletA and see what information the annotations feature provides. In the Enterprise Explorer, right-click **ServletA.java** and select **Team** → **Show Annotation**. If the Changing Quick Diff Reference window opens, select **OK**.

Rational Application Developer opens ServletA in a Java Editor, and a colored line is displayed in the left bar of the source code. When you hover over the colored line, a pop-up window shows the CVS revision that was last responsible

for changing that line (Figure 29-30). In the History view, the revision, which is associated with the selected line of source code, is highlighted.

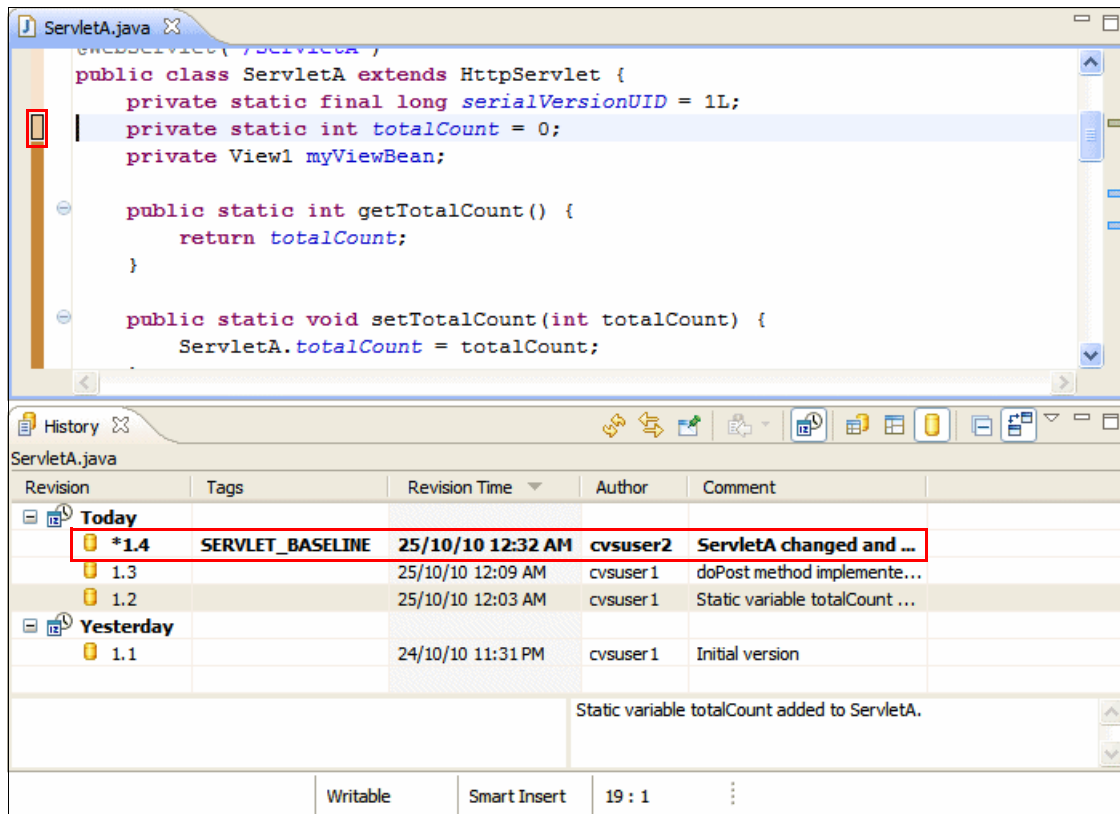


Figure 29-30 CVS Annotation view

## 29.8 Branches in CVS

*Branches* are a source control technique to allow development on more than one baseline in the repository.

In CVS, the HEAD branch refers to the latest or current work that is being performed in a team environment. This technique is only sufficient for a development team that works on one release, which contains all of the latest developments, including major enhancements and bug fixes. In an actual client situation, usually at least two streams are required:

- ▶ One main stream to manage the development
- ▶ A maintenance stream for the version that is currently in production

With two streams, new versions of the production build can be created without the fear of the production build being affected by the changes that are made to the main development stream. Branches are useful in this scenario and where CVS baselines and parallel streams of work need to be created.

At a certain point, the development and maintenance streams have to be merged to provide a new baseline to be a production version. This process ensures that any fixes or enhancements that have been made in the maintenance stream make it into the development stream. This process is known as a *merge*, and the CVS tools within Rational Application Developer provide features to facilitate this process.

Figure 29-31 illustrates the branching technique with two streams.

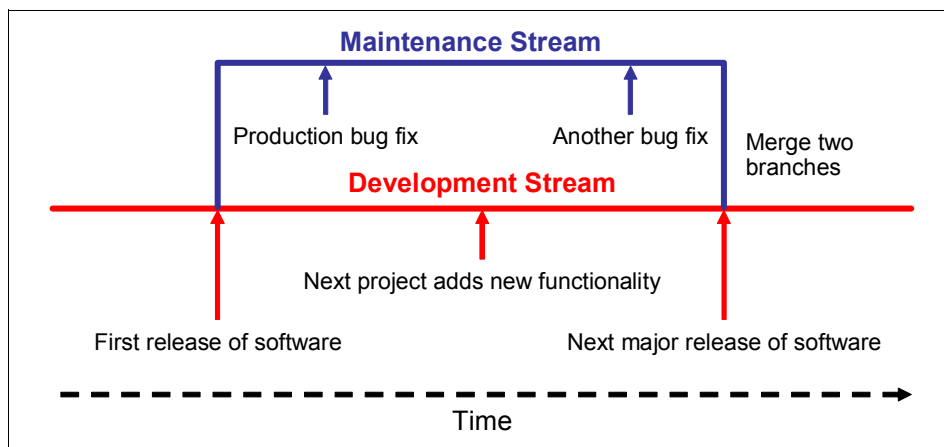


Figure 29-31 Branching with two streams

## 29.8.1 Branching

Creating a branch is useful when you want to maintain multiple streams of the software that is being developed or supported and when the streams are in separate stages of delivery (usually development and production support).

In this scenario, a particular release has been deployed to the production environment, and a new project has started to enhance the application. In addition, the existing production release has to be maintained so that problems that are identified are fixed quickly. *Branching* provides the mechanism to achieve multiple streams of the software, and the following example outlines how to create a branch.

Perform the following steps for the first workspace for cvsuser1:

1. In the Enterprise Explorer view of the Web perspective, right-click **RAD8CVSGuide** and select **Team** → **Tag as Version**. In the tag version field, type `BRANCH_ROOT` and click **OK**.
2. Right-click **RAD8CVSGuide** again and select **Team** → **Branch**.
3. In the Create a new CVS Branch window (Figure 29-32), complete the following actions:
  - a. For the Branch Name, enter `Maintenance`.
  - b. For the Version Name, enter `BRANCH_ROOT`.
  - c. Verify that **Start working on the branch** is selected so that the workspace automatically sets itself up for development on the new branch.
  - d. Click **OK**.

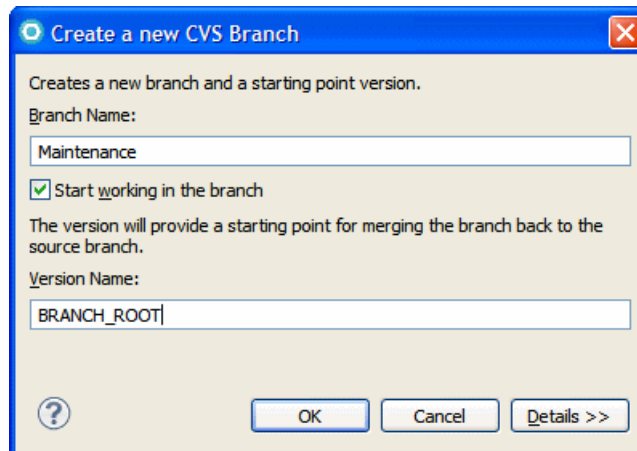


Figure 29-32 Creating a new CVS branch

**Version name:** Remember the version name that you enter here. This name identifies the point at which the branch was created and is required later when the branches are merged.

4. Right-click **RAD8CVSGuide** and select **Properties**.
5. In the Properties that are listed in the left pane, select **CVS**.

In the Enterprise Explorer view, the text Maintenance appears next to the project name. Also, in the CVS properties pane on the right, the Tag name displayed as Maintenance (Branch). This name indicates that the project is now associated with the CVS Maintenance branch, and any changes that are checked in go to that branch (Figure 29-33).

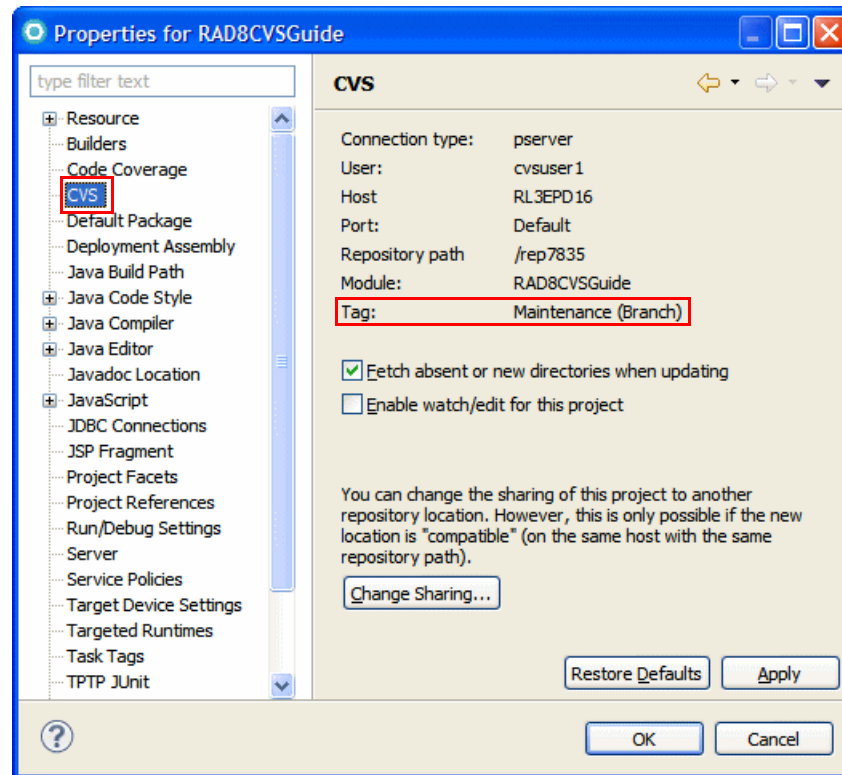


Figure 29-33 Branch information for a project in the local workspace

6. Open the CVS Repository Explorer window. Click **Window** → **Open Perspective** → **Other** and then select **CVS Repository Exploring**.
7. In the CVS Repositories view (Figure 29-34 on page 1579), expand the tree to verify that the branch has been created in the repository.

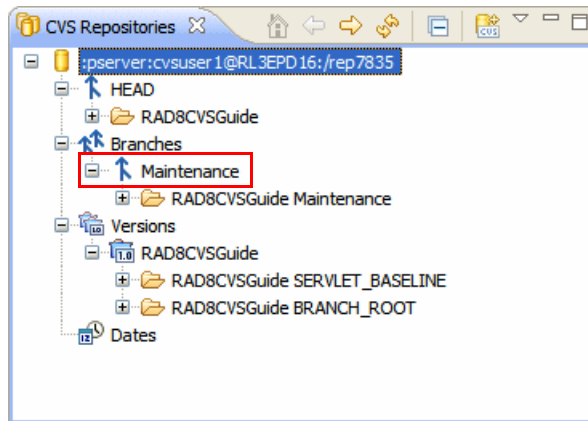


Figure 29-34 List of branches

## Refreshing branching information

The CVS Repositories view does not automatically receive a list of all branches from the server. If a branch has been created on the CVS server, the user of a workspace must perform a refresh to receive the name of the new branch. In our sample case, cvsuser2 must perform a refresh to receive information about the new maintenance branch.

From the cvsuser2 workspace, follow these steps:

1. To refresh the branches in a repository, open the **CVS Repository Exploring** perspective.
2. Select the repository and expand the tree. Select and right-click the **Branches** node and then select **Refresh Branches**.
3. In the Refresh Branches window, click **Select All** and then click **Finish**.

The Maintenance branch is shown now under the Branches folder (Figure 29-35).

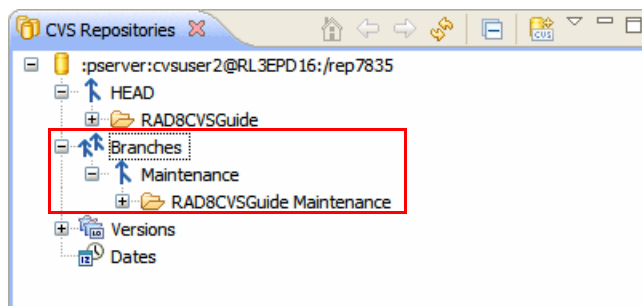


Figure 29-35 Refreshed branch list

## Updating branch code

In this section, we assume that changes are required to be made to `ServletA` and a new view bean (`View2`) must to be created. This scenario demonstrates the merge process with the changes being made in the maintenance branch and then moved into the main branch.

In the workspace of `cvsuser1`, follow these steps:

1. From the Enterprise Explorer, open **ServletA.java**.
2. Navigate to the `doPost` method. At the top, add the following statement:  

```
System.out.println("Added in some code to demonstrate branching");
```
3. Save and close the file.
4. Right-click the **itso.rad8.teamcvs.beans** package and select **New** → **Class**.
5. For the Name of the class, type `View2` and click **Finish**.
6. Right-click **RAD8CVSGuide** and select **Team** → **Synchronize With Repository**.
7. In the Synchronize view, right-click **RAD8CVSGuide** and select **Commit**.
8. In the Commit window, for the Revision comments, type `Branching example` and click **Finish**.
9. In the CVS Repository Explorer perspective (Figure 29-36 on page 1581), expand the tree under the **Maintenance** branch.



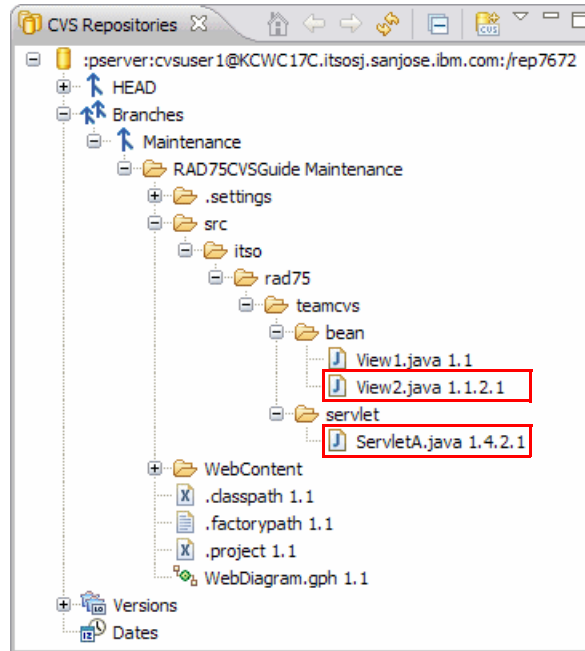


Figure 29-36 Code checked into the branch

**Logical revision:** The logical revision for `View2.java` is 1.1.2.1 and for `ServletA.java` is 1.4.2.1. The extra two numbers in the logical revision are added by CVS when a branch is created. The first two numbers indicate the logical revision where the branch was created. The third number indicates to which branch from the logical revision this version of the file belongs. In this example, it is the second revision if we count the HEAD branch; hence, the number is 2. The fourth number is the logical revision within this branch. In this case, both files are the first revision in the Maintenance branch, and so the last digit is 1.

The changes have now been committed into the Maintenance branch, which now has content that differs from the main branch. These changes are not seen by developers working on the HEAD branch, which is the development stream in our scenario.

## 29.8.2 Merging

Merging the branches occurs when code from one branch must be incorporated into another branch. This merging might be required for several reasons, such as

when a major integration release is about to be released for testing or bug fixes are required from the maintenance branch to resolve certain issues.

Our scenario is that we have completed the development on the main CVS branch. Any production fixes that were made to the Maintenance branch are required in the main branch before a production build can be released.

Merging the two branches requires the following information:

- ▶ The name of the branch or version that contains your changes is required.
- ▶ The version from which the branch was created is required. This name is the version name that you supplied when branching.

In our case, the branch is called *Maintenance*, and the version from which we created the branch was called *BRANCH\_ROOT*.

Merging requires that the target or destination branch is loaded into the workspace before merging in a branch. Because, in our scenario, the changes are merged to HEAD, the HEAD branch must be loaded in the workspace.

Perform the following steps in the `cvsuser1` workspace:

1. In the Web perspective, right-click **RAD8CVSGuide** and select **Replace With** → **Another Branch or Version**.
2. In the window, select **HEAD** and click **OK** to load the latest (HEAD) version of the RAD8CVSGuide project into the workspace.
3. Right-click **RAD8CVSGuide** and select **Team** → **Merge**.
4. In the Merge: Select the merge points window (Figure 29-37 on page 1583), select the start and end points of the merge:
  - a. For the Common base version (start tag) field, click **Browse**, expand **Versions**, select **BRANCH\_ROOT**, and click **OK**.
  - b. For the Branch or version to be merged (end tag) field, click **Browse**, expand **Branches**, select **Maintenance**, and click **OK**.

The Select the Merge Points window now shows the start and end tags of the merge, which will be applied to the version in the workspace.

The “Preview Merge in the synchronize view” option and the “Perform the merge into the local workspace” option provide the facility to select where to perform the merge:

- By previewing it in the Synchronize view, the user can review and make changes to each file as required.
- Performing the merge into the local workspace applies the changes immediately in the workspace based on preferences that were selected in the workspace preferences.

- c. Select **Preview Merge in the synchronize view**.
- d. Clear **Merge non-conflicting changes and only preview conflicts**.
- e. Click **Finish** to start merging.

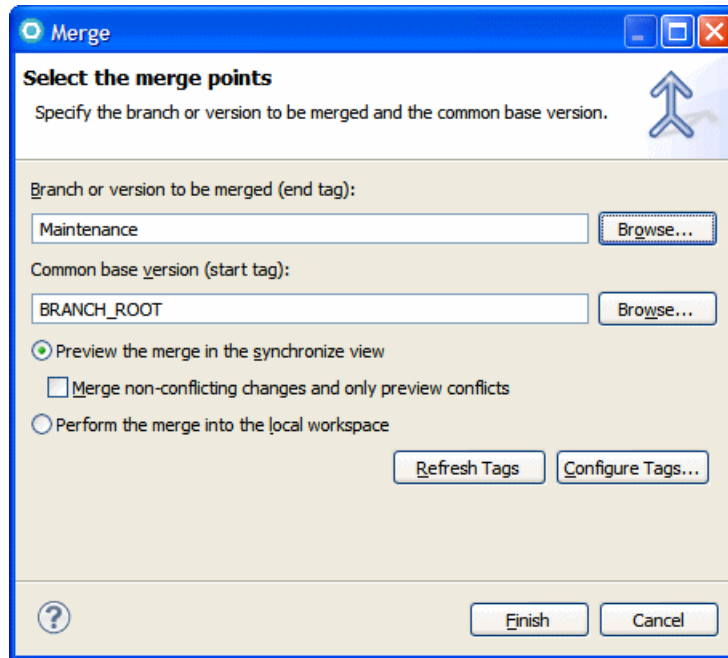


Figure 29-37 Selection of the merge start point

5. Expand the tree in the Synchronize view to display the changes. Verify that there are no conflicts. If there are conflicts, the developer must resolve them. In our case, the merge is simple, and there are no conflicts (Figure 29-38).

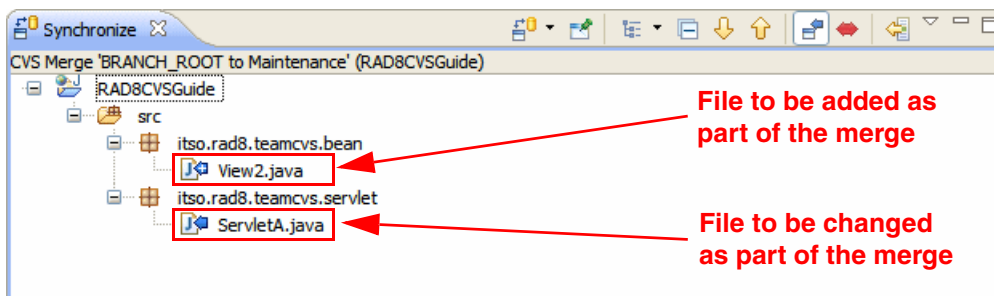


Figure 29-38 Files required to be merged

6. Right-click **RAD8CVSGuide** and select **Merge** to bring the changes from the branch into the main stream. Because there are no conflicts, the merge completes successfully.
7. From the Enterprise Explorer view, right-click **RAD8CVSGuide** and select **Team** → **Synchronize with Repository**.
8. Expand the **Synchronize** view to display the changed files ServletA.java and View2.java (Figure 29-39).

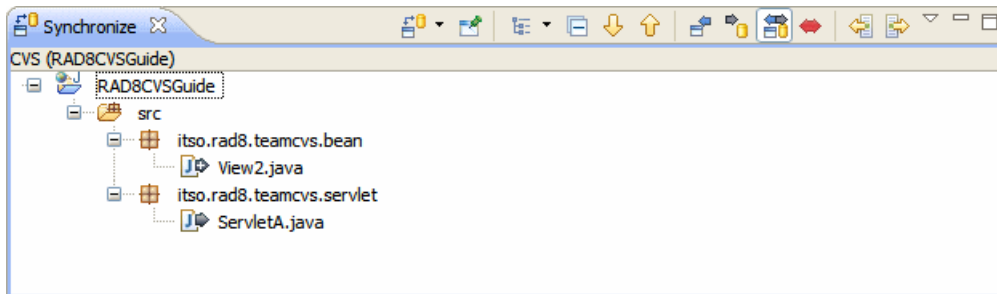


Figure 29-39 CVS updates to HEAD from the merge

This view shows that the file View2.java is a new file to be added to the repository and that the file ServletA.java has been changed. This view is consistent with the changes that were made in the Maintenance branch and that now must be added to the main branch.

9. Right-click the project and select **Commit**.
10. In the Commit window, add the comment Merged changes from Maintenance branch and click **OK**.

The changes from the branch have now been merged into the main development branch.

This scenario, although a simple one, highlights the technique that is required by users to work with branches. In an actual scenario, there might be conflicts, which require resolutions between the developers. Be aware that branching and concurrent development are complex processes and require communication and planning between the two development teams.

Rational Application Developer provides the tools to assist developers when merging. However, equally important are the procedures for handling situations, such as branching and merging code, which must be established among the team early in a project life cycle.

## 29.9 Working with patches

Rational Application Developer enables developers to share work, even when they only have read access to a CVS repository. In this situation, the developer who does not have full access to the repository can create a patch and forward it to another developer who has write access. The patch can be applied to the project, and the changes can be committed.

This configuration is useful when access to the source code repository has to be restricted to a small number of users to prevent uncoordinated changes corrupting the quality of the code. Any number of users can then contribute changes and fixes to the repository using patches. However, they can contribute the changes only through designated code minders who can commit the work and who have the opportunity to review the changes before applying them to the repository.

To contribute the changes, you use the **Team** → **Create Patch** and **Team** → **Apply patch** options from a project context menu. A patch can contain a single file, a project, or any combination of resources on the workspace. The Rational Application Developer online help has a complete description of how to work with CVS patches.

## 29.10 Disconnecting a project

For many reasons, a developer might want to disconnect a project from the current CVS repository. For example, a developer might want to disassociate a project from one repository so that it can be added to another repository. To perform this task, complete the following steps:

1. In the Web perspective, right-click **RAD8CVSGuide** and select **Team** → **Disconnect**.
2. In the Confirm Disconnect from CVS window (Figure 29-40 on page 1586), select **Do not delete the CVS meta information** and click **OK**.

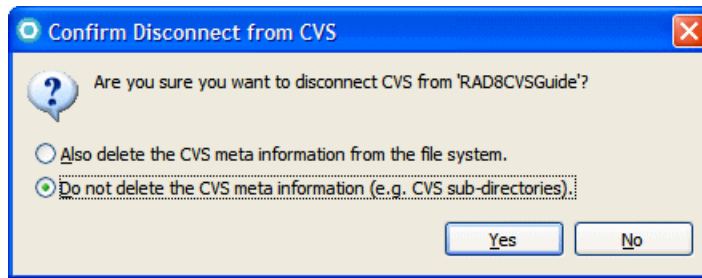


Figure 29-40 Disconnect confirmation

By not deleting the CVS meta information, we can reconnect the project with the CVS repository later more easily. If the meta information is removed, CVS cannot determine with which revision in the repository a particular file is associated.

## Reconnect

You can reconnect a project to the repository by selecting **Team** → **Share Project**. Reconnecting is easier if the CVS meta information was not deleted.

- ▶ If the meta information *was* deleted, the user is prompted to synchronize the code with an existing revision in the repository.
- ▶ If the meta information is still available, follow these steps:
  - a. Select **Team** → **Share Project**.
  - b. In the Share Project window, select **CVS** and click **Next**.
  - c. In the Share Project: Connect Project to Repository window (Figure 29-41 on page 1587), which shows the original CVS repository information, click **Finish** to reconnect.

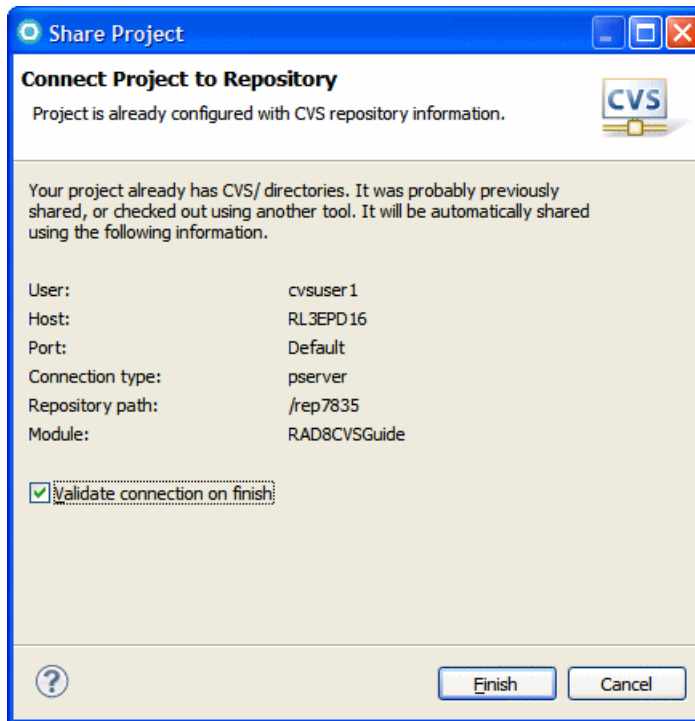


Figure 29-41 Reconnect to repository with original CVS meta information

## 29.11 Team Synchronizing perspective

We have used the Team Synchronizing perspective in Rational Application Developer in the examples in this chapter, but we have not described this perspective in detail yet. This perspective provides to the user with a tool to identify changes in the repository compared with changes on the local workspace and assists in synchronizing the two.

The Team Synchronize perspective provides the following features:

- ▶ A comparison of changes in the workspace (as described in 29.6, “Comparisons in CVS” on page 1571)
- ▶ The commitment of the changes that have been made to the repository (as described in the previous scenarios)
- ▶ The creation of the custom synchronization of a subset of resources in the workspace
- ▶ A scheduled checkout synchronization

## 29.11.1 Custom configuration of resource synchronization

The Synchronize view provides the ability to create custom synchronization sets for the purpose of synchronizing only the identified resources on which a developer might be working. Using this feature, a developer can focus on changes that are part of the scope of work and ensure that the developer is aware of the changes that occur without worrying about changes to other areas.

The developer can make changes to the other areas or someone else might check in changes to these parts, but only the resources in the defined set are synchronized.


This Synchronize view is convenient if the changes being made are localized and if other areas of the code are changing in ways that are not important for the specific scope of work. Problems can also occur with this mode of operation. Developers must be careful that important changes to the non-synchronized parts are not ignored for long periods of time.

**Important:** Custom synchronization is most effective when an application is designed with defined interfaces, where the partitioning of work is clear. However, even in this scenario, use custom synchronization carefully, because it can introduce additional work in the development cycle for final product integration. You must document and enforce procedures to ensure that integration is incorporated as part of the work pattern for this scenario.

The example scenario (again using the RAD8CVSGuide project) demonstrates custom synchronization, through two configurations:

- ▶ Full synchronization of the RAD8CVSGuide project
- ▶ Partial synchronization of the ServletA.java



To perform the example scenario, complete the following steps for the cvsuser1 workspace:

1. Open the **Team Synchronizing** perspective.
2. At the top the Synchronize view, click the **Synchronize** icon  and click **Synchronize** to add a new synchronization definition.
3. In the Synchronize window, select **CVS** and click **Next**.



4. In the Synchronize CVS window, expand the **Java Workspace** tree to view the contents. All resources in the workspace are selected. Accept the defaults and click **Finish**.

If there are no changes, a dialog box opens with the message Synchronizing CVS: No changes found. In the Synchronize view, you see the message No changes in 'CVS (Workspace)'. Click **OK** in the dialog box.

5. To preserve this synchronization, click the **Pin Current Synchronization** icon ().
6. Add a new synchronization by clicking the **Synchronize** icon () at the top of the Synchronize view.
7. In the Synchronize window, select **CVS** and click **Next**.
8. In the Synchronize CVS window, expand the project tree under **JavaSource** to view the contents. Click **Deselect All** to clear all the resources, select only **ServletA.java** (Figure 29-42 on page 1590), and click **Finish**.

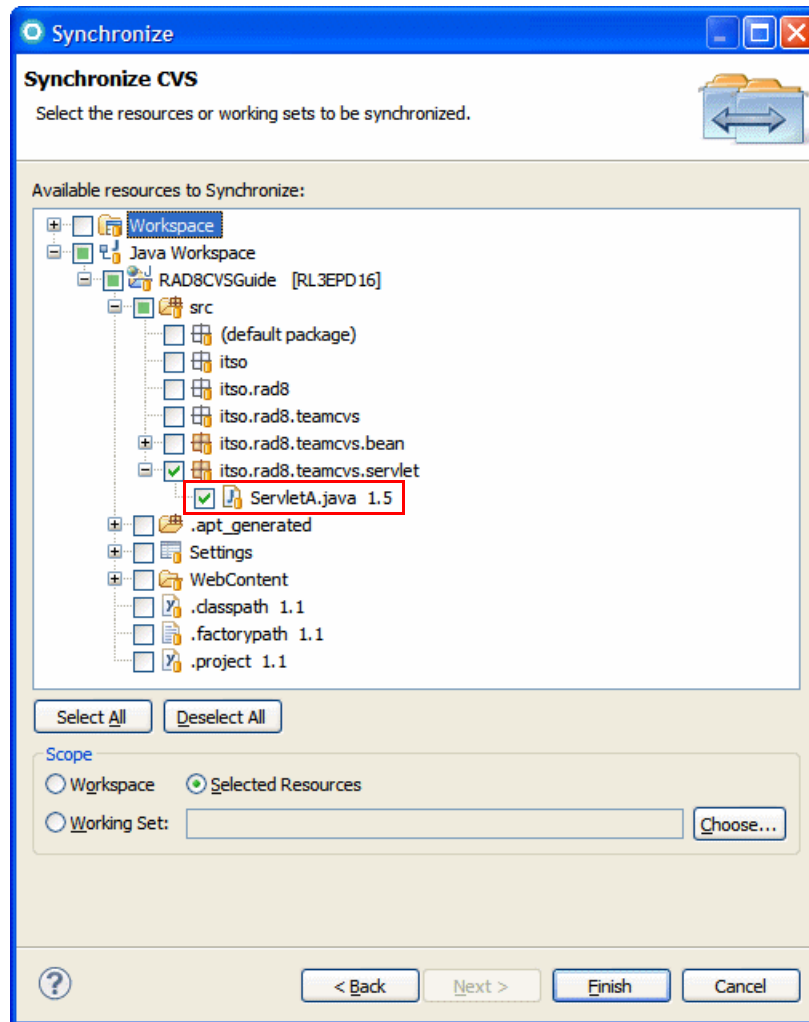


Figure 29-42 Selecting ServletA.java for synchronization

9. If there are no changes, a dialog box opens with the message Synchronizing: No changes found. In the Synchronize view, you see the message No changes in 'CVS (Workspace)'. Click **OK** in the dialog box.
10. To preserve this synchronization, click the **Pin Current Synchronization** icon (📌).
11. At the top of the Synchronize view, click the Synchronize icon (↔️). Now two CVS synchronizations are in the list (Figure 29-43 on page 1591).

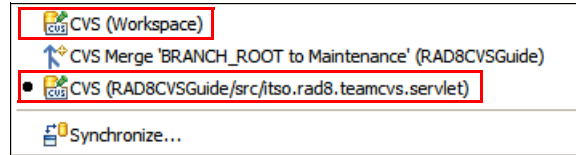


Figure 29-43 List of synchronizations that were created

## 29.11.2 Schedule synchronization

Rational Application Developer can schedule the synchronization of the workspace. This feature follows 29.11.1, “Custom configuration of resource synchronization” on page 1588, in which a user wants to schedule the synchronization that has been defined. *You can only schedule a synchronization for synchronizations that have been pinned.*

To demonstrate this feature, assume that the RAD8CVSGuide project is loaded in the workspace and that a synchronization has been defined for this project and pinned. Then use the following steps to schedule this project for synchronization:

1. In the Synchronize view (Figure 29-44), select **Schedule** from the drop-down list.

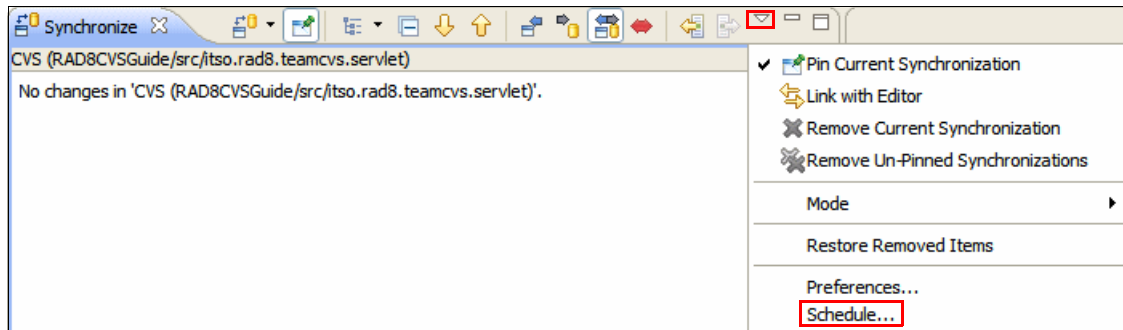


Figure 29-44 Drop-down selection for scheduling synchronization

2. In the Configure Synchronize Schedule - CVS window (Figure 29-45 on page 1592), select **Synchronize automatically** and the time period that you want to synchronize. Click **OK**.

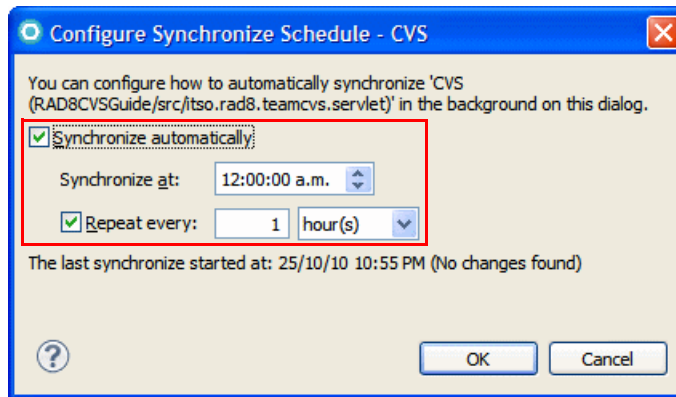


Figure 29-45 Setting the synchronization schedule

3. If the current CVS synchronization is not already pinned, and you are prompted to pin it, click **Yes**.

Assuming that one hour is chosen, the RAD8CVSGuide project is synchronized every hour to ensure that the latest updates are available. This action performs the synchronize operation and shows any available changes in the Synchronize view, where the user can accept or postpone integrating the changes as appropriate.

## 29.12 More information

The Help feature provided with Rational Application Developer has a large section that describes using the Team Synchronizing and the CVS Repository Exploring perspectives and describes all the features that have been covered in this chapter. In addition, the following websites provide further information for the topics in this chapter:

- ▶ CVS home page

This web page is the major source of information for CVS.

<http://www.nongnu.org/cvs/>

- ▶ Eclipse CVS information

The CVS features that are available in Rational Application Developer come from Eclipse 3.4. The following link is the major information page for this project. It contains documentation, downloads, and the source code.

<http://wiki.eclipse.org/ CVS>

- ▶ Tortoise CVS home page

Tortoise is another CVS client that helps users perform CVS operations from Microsoft Windows Explorer. It provides most of the features of Rational Application Developer and is available under the GNU public license. Tortoise is a convenient client to use when CVS operations are required outside Rational Application Developer.

<http://www.tortoise cvs.org>





# IBM Rational Application Developer integration with Rational Team Concert

Rational Application Developer includes integration with Rational Team Concert to provide a collaborative development environment for IBM middleware. You can extend the Rational Application Developer environment with the Rational Team Concert Client to provide embedded access to Rational Team Concert artifacts.

Rational Application Developer also provides an extension to the Rational Team Concert Build System for integrated Code Coverage execution and reporting for builds. Rational Application Developer also provides another extension to the Rational Team Concert Server to add powerful team-based debugging.

This chapter contains the following sections:

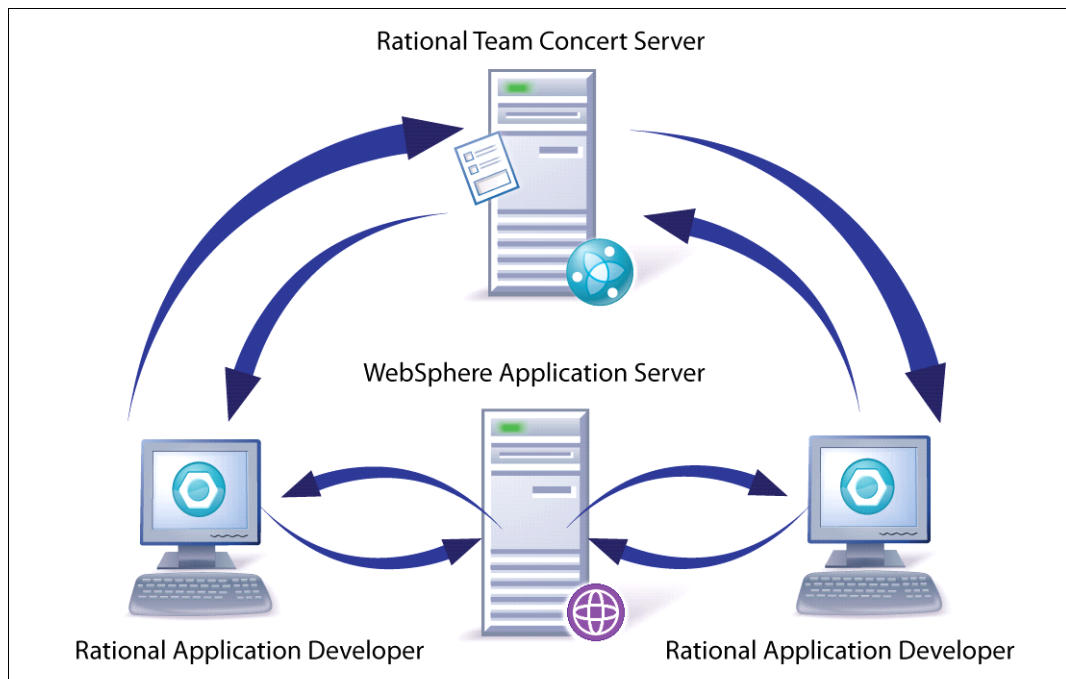
- ▶ System architecture
- ▶ Installing Rational Team Concert Client into the Rational Application Developer workbench
- ▶ Collaborative Code Coverage
- ▶ Collaborative Debug

## 30.1 System architecture

In Figure 30-1, you see two developer machines configured with Rational Application Developer and the Rational Team Concert client extensions. They can be connected to a shared WebSphere Application Server that is used as a test staging server.

Through the Rational Team Concert client extensions, the two developer machines also connect to the Rational Team Concert Server. The server provides the basis for collaboration among developers for Rational Application Developer capabilities, such as Code Coverage and Team Debug, in addition to the capabilities provided by Rational Team Concert, such as milestone plans, work items, and source.

The Rational Team Concert views are also seamlessly integrated into the Rational Application Developer perspectives to allow a rich, integrated workflow for Rational Application Developer developers using Rational Team Concert.



*Figure 30-1 Architecture overview for Rational Application Developer working with Rational Team Concert and WebSphere Application Server*



## 30.2 Installing Rational Team Concert Client into the Rational Application Developer workbench

Installing Rational Team Concert Client into Rational Application Developer gives you a powerful collaborative development environment for your WebSphere applications. This section covers the installation of Rational Team Concert Client V3.0 and V2.0.0.2.

### 30.2.1 Installing Rational Team Concert Client 3.0 into the same workbench as Rational Application Developer

If you want to install Rational Team Concert Client 3.0 into the same workbench as Rational Application Developer V8, you must first download from the <https://jazz.net> website installation files for Rational Team Concert Client 3.0. Perform these steps:

1. From <https://jazz.net>, download Rational Team Concert 3.0, which is a file, such as RTC-Web-Installer-Win-3.0.zip.
2. Extract the contents of the compressed file that you downloaded. For example, extract the contents to C:\RTC30. Ensure that you preserve the directories of the extracted contents.
3. Execute **C:\RTC30\1launchpad.exe**.
4. Choose **Rational Team Concert – Client for Eclipse IDE**.
5. IBM Installation Manager starts. Ensure that **Version 3.0** is selected under the **Rational Team Concert – Client for Eclipse IDE** selection. Click **Next**.
6. If you agree to the terms of all of the license agreements, select **I accept the terms of the license agreements** and then click **Next**.
7. Choose **Use the existing package group** and select the package group in which Rational Application Developer is installed, for example, C:\Program Files\IBM\SDP. Click **Next**.
8. Optional: If you use Sametime® for collaboration, check the **Sametime Integration Update Site**.
9. Click **Next**.
10. Choose your desired help system configuration and click **Next**.
11. Review the summary page and click **Install**.

## 30.2.2 Installing Rational Team Concert Client 2.0.0.2 into the Rational Application Developer workbench

If you want to install Rational Team Concert Client 2.0.0.2 into the same workbench as Rational Application Developer V8, you must download from the <https://jazz.net> website installation files for Rational Team Concert Client 2.0.0.2 that are formatted for *p2 Install*. Perform these steps:

1. From <https://jazz.net>, download a *p2 Install* edition of a version of Client for Eclipse that is compatible with Rational Application Developer V8. For example, download Version 2.0.0.2 interim fix 4.
2. Extract the contents of the compressed file that you downloaded. For example, extract the contents to C:\RTC. Ensure that you preserve the directories of the extracted contents.
3. Start Rational Application Developer.
4. Click **Help** → **Install New Software**.
5. Under **Available Software**, click **Add**. The Add Repository dialog box opens.
6. Beside the Name field, click **Local** and then go to the `rtc-p2-repository` subdirectory of the directory where you previously extracted the compressed file, for example, C:\RTC\rtc-p2-repository. Click **OK**.
7. Follow these steps on the Available Software page:
  - a. In the Name section, select **Rational Team Concert** and optionally one or both global language categories to install language packs.
  - b. In the details section, ensure that **Group items by category** and **Contact all update sites** are selected. Click **Next**.
8. On the Install Details page, click **Next**.
9. On the Review Licenses page, read the text of the license agreements. If you agree to the terms of all of the license agreements, select **I accept the terms of the license agreements** and then click **Finish**.
10. During the installation, a security warning dialog box might open because the Rational Team Concert client bundles and features are not signed. If you see this warning, click **OK** to complete the installation.
11. When you are prompted to restart at the end of the installation, click **Yes**.

When the workbench restarts, the Rational Team Concert Welcome page opens.

## 30.3 Collaborative Code Coverage

The Code Coverage tool is a great way to determine how much of your Java code is being executed during a test run. Through the Rational Code Coverage Extension for Rational Team Concert Build Toolkit from Rational Application Developer, you can create Code Coverage statistics as part of your Rational Team Concert builds. You can publish the statistics and HTML reports as part of the Rational Team Concert build results and view them directly from the Build Results from the Rational Team Concert client in Rational Application Developer or from the Rational Team Concert web page. This function allows project leaders and all members of the project to monitor the coverage results of their JUnit tests from these builds as well as perform comparisons from previous builds to see where coverage might have changed.

The following sections describe how to set up a Rational Team Concert build to achieve this capability.

### 30.3.1 Configuring a build definition

Before Code Coverage statistics can be generated from a Rational Team Concert build, you must configure a Build Definition to support Code Coverage. The instructions provided in this section assume that a new build is being created from the beginning, but you can use the instructions to adapt an existing build easily.

This section also assumes that the Rational Code Coverage Extension for Rational Team Concert Build Toolkit has already been installed. See Appendix A, “Installing the products” on page 1783 for instructions to install this extension. The installation location path is required to complete the configuration. It is also assumed that a Rational Team Concert client has been installed into Rational Application Developer.

Follow these instructions to configure a new build definition for Code Coverage:

1. Switch to the **Work Items** perspective.
2. Create a Repository Connection and connect to a project area.
3. In the Team Artifacts view, expand the node for the project area. Then expand the **Builds** node, followed by the **Build Engines** node.
4. Right-click the **Build Engines** node and select **New Build Engine**. In the General Information section, populate the **ID** and **Project or Team Area** text fields.
5. Right-click the **Builds** node and select **New Build Definition**.

6. The **New Build Definition** window opens. If not already populated, populate the **Project or team area in which to create the new build** text field by selecting your project area. Click **Next**.
7. Supply an ID and Description for your build. In the Available build templates section, select **Ant – Jazz Build Engine**. See Figure 30-2.

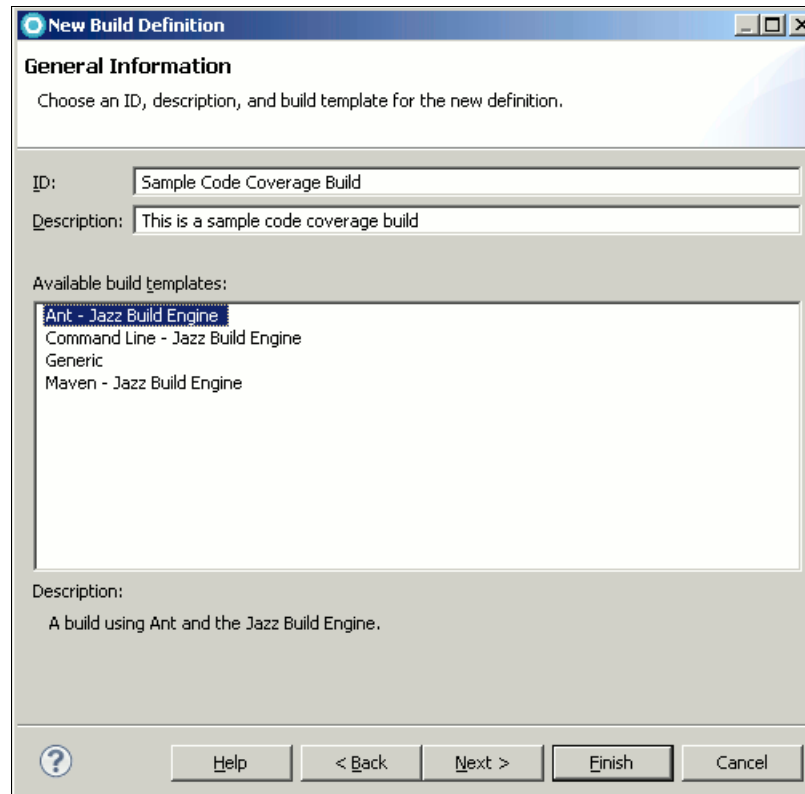


Figure 30-2 Defining a new build definition

8. On the Pre-Build page, select the **Jazz Source Control**.
9. On the Post-Build page, select the item types that will be used for publishing in your script.
10. Click **Finish**.
11. The build editor appears and will be populated based on your selections. In the Supporting Build Engines section on the Overview tab, select the build engine that was created in step 4.
12. Switch to the **Jazz Source Control** tab. In the Build Workspace section, populate the Workspace text field using either **Select** or **Create**. *The*

*workspace chosen here must be accessible by the user account that was used to start the Jazz Build Engine.*

13. In the Load Options section, populate the Load directory text box. The directory that is specified will be used to store code that has been extracted from the jazz source control. The specified path needs to be relative to the Jazz Build Engine execution directory. Optionally, select **Delete directory before loading** to clear the directory every time that the build is executed.
14. Switch to the **Ant** tab. Specify a build.xml script in the **Build file** text field. *This path must be relative to the location of the Jazz Build Engine and it depends on the load directory that was specified in the previous step.*
15. In the Ant Configuration section, select the **Include the Jazz build toolkit tasks on the Ant library path** check box. In the Ant arguments text field, enter `-lib <Code-Coverage-RTC-extension-directory>`, for example, `-lib /opt/IBM/TeamConcertBuild/buildsystem/codecoverage`.
16. Save the new Build Definition.

After the Build Definition has been configured, you need to start a Jazz Build Engine for the defined Build Engine.

### 30.3.2 Creating an Ant build script to generate coverage statistics

The configured Build Definition targets a user-defined `build.xml` script to handle generating statistics. This section contains information about setting up the build file to generate coverage statistics.

#### Setting up the Ant build script

Open your Ant build file and import the `CodeCoverageProperties.xml` file from the Code Coverage Extension directory (see Example 30-1).

*Example 30-1 Generating a baseline and probescript file*

---

```
<import
file="<Code-Coverage-RTC-extension-directory>\CodeCoverageProperties.xml
1"/>
```

---

In Example 30-1, the `<Code-Coverage-RTC-extension-directory>` directory is the installation path of the Code Coverage Extension for Rational Team Concert Build Toolkit.

This file contains several predefined properties that can be reused in your build script. This file is provided as a convenience so that multiple file paths do not have to be defined.

## Generating baseline and probescript files

The probescript and baseline files are requirements of the execution. The probescript file contains a filter set that is used to define the set of classes for which statistics will be collected. The baseline file is used to store additional metadata about the project. Additional details about the probescript and baseline files are available in Chapter 32, “Code Coverage” on page 1697.

The probescript and baseline files must be generated for the particular application before statistics can be generated for the project. The Code Coverage Extension provides an Ant task that can be used by the build.xml script to generate these files. Example 30-2 provides an overview of how to use the Ant task.

*Example 30-2 Generating a baseline and probescript file*

```
<target name="application-analysis">
 <code-coverage-app-analyzer
 projectDir="${basedir}"
 probescript="${probescriptFile}"
 baseline="${baselineFile}"/>
</target>
```

Table 30-1 explains the parameters that are expected as input by the Ant task.

*Table 30-1 Ant task parameters*

Parameter	Description
projectDir	The directory containing the Java project
probescript	The output location for the generated probescript
baseline	Optional: The output location of the baseline project index file

Because the probescript file is required for execution, it must be generated to a location that is easily accessible.

## Running the application and generating statistics

You must execute the application as part of the build to generate coverage statistics. You must configure the Ant task so that data collection will occur when the program is executed. The three Java virtual machine (JVM) arguments that need to be added to the execution call are defined below:

```
▶ <jvmarg
 value=""${llc-jvm-arg-output}${coverageOutputFile}"" />
```

- ▶ `<jvmarg value="&quot;${llc-jvm-arg-engine}&quot;" />`
- ▶ `<jvmarg value="&quot;${llc-jvm-arg-jvmti}${probescriptFile}&quot;" />`

Table 30-2 explains the JVM arguments.

*Table 30-2 JVM arguments*

Property name	Description
llc-jvm-arg-output	This property is automatically defined in the CodeCoverageProperties.xml file and can be reused after the file is imported.
llc-jvm-arg-engine	This property is automatically defined in the CodeCoverageProperties.xml file and can be reused after the file is imported.
llc-jvm-arg-jvmti	This property is automatically defined in the CodeCoverageProperties.xml file and can be reused after the file is imported.
coverageOutputFile	This property is the output location of the coverage statistics file. It must be set as a property by the user before running. This property must specify a file and must have the .coveragedata extension.
probescriptFile	This property is the path to the probescript file that was generated in the previous step. This property is used as input into the instrumentation engine to control data collection.

Example 30-3 provides an overview of how to run Code Coverage using the Java Ant task to execute an application.

*Example 30-3 Generating statistics on a regular Java application*

---

```

<property name="coverageOutputFile" value="stats.coveragedata"/>
<property name="probescriptFile" value="myproject.probescript"/>
<java classname="com.ibm.storeapp.models.Store" fork="true"
newenvironment="true">
 <jvmarg
value=""${llc-jvm-arg-output}${coverageOutputFile}"" />
 <jvmarg value=""${llc-jvm-arg-engine}"" />
 <jvmarg
value=""${llc-jvm-arg-jvmti}${probescriptFile}"" />
</java>

```

---

Example 30-4 provides an overview of how to run Code Coverage using the JUnit Ant task to execute a test.

*Example 30-4 Using the JUnit Ant task to execute a test*

---

```
<property name="coverageOutputFile" value="stats.coveragedata"/>
<property name="probescriptFile" value="myproject.probescript"/>
 <junit showoutput="true" fork="yes" newenvironment="true">
 <jvmarg
value=""${llc-jvm-arg-output}${coverageOutputFile}"" />
 <jvmarg value=""${llc-jvm-arg-engine}"" />
 <jvmarg
value=""${llc-jvm-arg-jvmti}${probescriptFile}"" />

 <formatter type="xml" />
 <test name="com.ibm.storeapp.models.test.TestCustomer"
outfile="TestCustomer" />
 <classpath>
 <pathelement path="${junitJar}" />
 </classpath>
 </junit>
```

---

After the execution completes, a `coveragedata` file is generated. This file contains the coverage statistics for the executed application, and it is posted to the Rational Team Concert server for storage.

## Generating HTML reports

Code Coverage provides another Ant task, which can be used to generate HTML reports for your statistics. Then you can post the HTML reports to the Rational Team Concert server for storage. The reports can be downloaded from the server using a Rational Team Concert client (either the Web-based client or full client in Rational Application Developer).

Generating HTML reports for your build is an optional step. To configure your build to generate HTML reports, follow the steps:

1. Configure the build for BIRT report generation:
  - a. In a web browser, open <http://www.eclipse.org/birt> and select the latest released version. Download the only the Report Engine offering.
  - b. Extract the contents the Report Engine compressed file that you downloaded in the previous step.
  - c. In the Rational Team Concert Eclipse Client, open the Build Definition and change to the **Ant** Configuration tab.



- d. In the Ant Arguments field, add the argument `-lib <BIRT_runtime_install_directory>\ReportEngine\lib`, where `<BIRT_runtime_install_directory>` is the installation directory of the BIRT Report Engine.
2. Generate reports using the code-coverage-report Ant task. Example 30-5 shows the general form of the Ant task.

*Example 30-5 Generating an HTML report*

---

```
<code-coverage-report outputDir="${reportDir}"
 coverageDataFile="${coverageOutputFile}"
 baseLineFiles="${baselineFile}"
 reportFileDirectory="${code-coverage-report-dir}"
 birtHome="${birt-report-engine}" />
```

---

Table 30-3 describes the task parameters.

*Table 30-3 Expected parameters for HTML report generation*

Property	Description
outputDir	[Output] This property is a path to a directory that is used to store the output of the report generation step.
coverageDataFile	[Input] This property is the coveredata statistics file that is generated in the execution step.
coverageDataFile	[Input] This property is the baseline file that is generated for the application in the analysis step.
reportFileDirectory	[Input] This property is the directory containing the report design files. The path to the correct directory is defined in the <code>\${code-coverage-report-dir}</code> property (from the <code>CodeCoverageProperties.xml</code> import file).
birtHome	[Input] This property is the location of the BIRT Report Engine and the location of the ReportEngine directory (for example, <code>D:\Jazz\jazz\buildsystem\codecoverage\extra\birt-runtime-2_5_2\ReportEngine</code> ).

3. Publish the HTML report to the Rational Team Concert server. You can use the following example (Example 30-6 on page 1606). Using the `artifactFilePublisher` Ant task publishes the HTML reports.

*Example 30-6 Compressing and publishing an HTML report*

---

```
<zip destfile="${zippedReportFile}" basedir="${reportDir}"
includes="*" />
<artifactFilePublisher buildResultUUID="${buildResultUUID}"
 repositoryAddress="${repositoryAddress}"
 userId="${userId}" password="${password}"
 verbose="true" filePath="${zippedReportFile}"
 label="The HTML file" />
<buildResultPublisher buildResultUUID="${buildResultUUID}"
 repositoryAddress="${repositoryAddress}" userId="${userId}"
password="${password}" />
```

---

### **Publishing results to the Rational Team Concert server**

In Rational Application Developer, the Code Coverage tool checks for files that are published with specific contribution types. If files with these contribution types are found, they are included in the build result. If no files that match the required contribution type are found, the statistics are absent from the user interface.

The Code Coverage tool checks for files with the following contribution IDs:

- ▶ `com.ibm.rational.llc.build.coverage` is the `coveragedata` file (generated in the execution step). It must be a `.zip` file containing the `coveragedata` file.
- ▶ `com.ibm.rational.llc.build.baseline` is the `baseline` file (generated in the application analysis step). It must be a `.zip` file containing the `baseline` file.

You can configure your Ant build script to publish the Code Coverage results to the Rational Team Concert server with the correct contribution types. Use the `filePublisher` Ant task that is provided by the Rational Team Concert Build Toolkit.

Example 30-7 shows an example usage.

*Example 30-7 Publishing statistics back to the Rational Team Concert server*

---

```
<filePublisher buildResultUUID="${buildResultUUID}"
 repositoryAddress="${repositoryAddress}"
 userId="${userId}"
 password="${password}"
contributionTypeId="com.ibm.rational.llc.build.coverage"
 verbose="true"
 filePath="${zippedCoveragedataFile}"
 label="Coveragedata File"
 failOnError="true" />

<filePublisher buildResultUUID="${buildResultUUID}"
```

```

repositoryAddress="{repositoryAddress}"
userId="{userId}"
password="{password}"
contributionTypeId="com.ibm.rational.llc.build.baseline"
verbose="true"
filePath="{zippedBaselineFile}"
label="Baseline file"
failOnError="true" />

```

---

### 30.3.3 Viewing coverage statistics in Rational Application Developer

You can view Code Coverage statistics that were generated in a Rational Team Concert build environment from within Rational Application Developer. To open the coverage statistics in a build, right-click a build definition and select **Show Build Results**. The Builds view appears (see Figure 30-3).

	Build	Label	Progress	Estimated Comple...	Start Time	Duration
✓	LLC Test Project build	20101029-0931	Completed		October 29, 2010 12:...	46 seconds
✓	LLC Test Project build	20101029-0926	Completed		October 29, 2010 12:...	43 seconds
✗	LLC Test Project build	20101029-0923	Completed		October 29, 2010 12:...	5 seconds
✓	LLC Test Project build	20101029-0918	Completed		October 29, 2010 12:...	48 seconds
✓	LLC Test Project build	20101028-0656	Completed		October 28, 2010 9:5...	48 seconds

Figure 30-3 The Builds view displaying the build results

You can open a build result in the build result viewer by selecting a build, right-clicking, and selecting **Open**. The Build viewer for the selected build appears, and the Overview page is selected. The Coverage section (under the Contribution Summary section) presents the aggregated statistics for the build (see Figure 30-4 on page 1608).

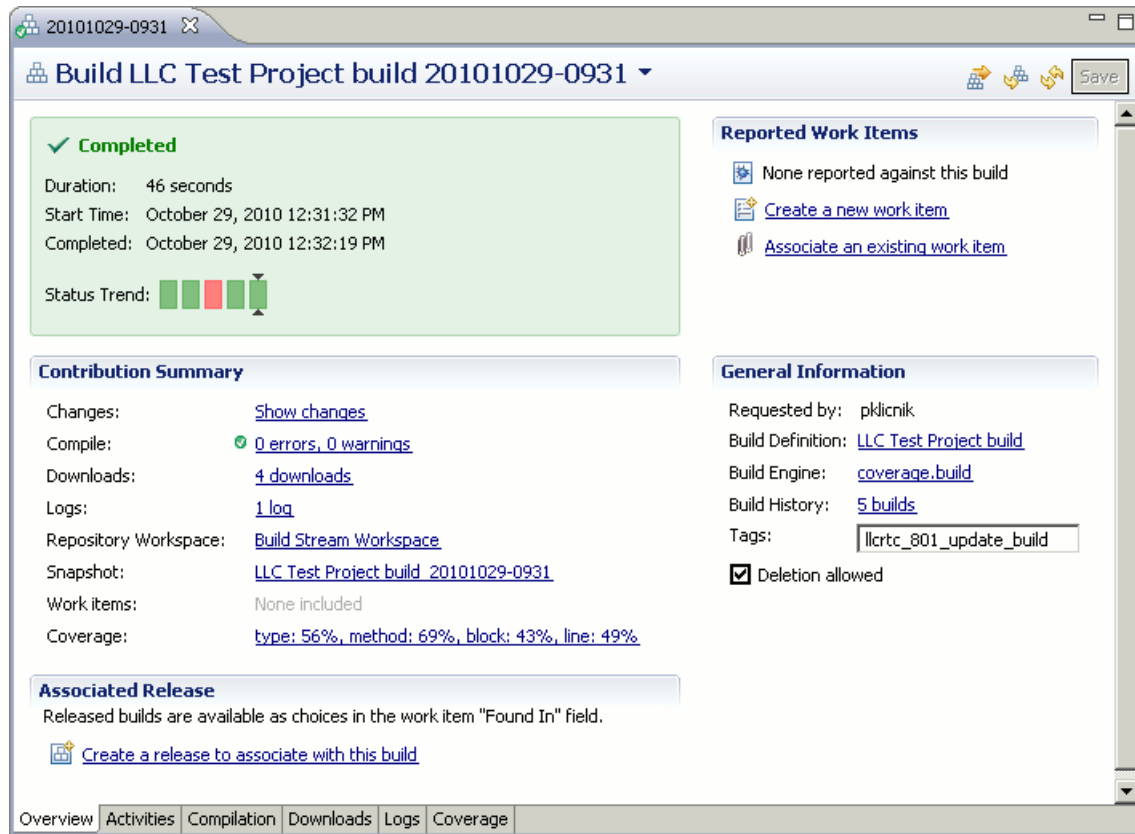


Figure 30-4 Build result viewer with the Overview page selected

To view the statistics in a higher level of detail, you can switch to the Coverage tab (see Figure 30-5 on page 1609). The report on the Coverage tab allows you to view the coverage statistics at all levels of granularity.

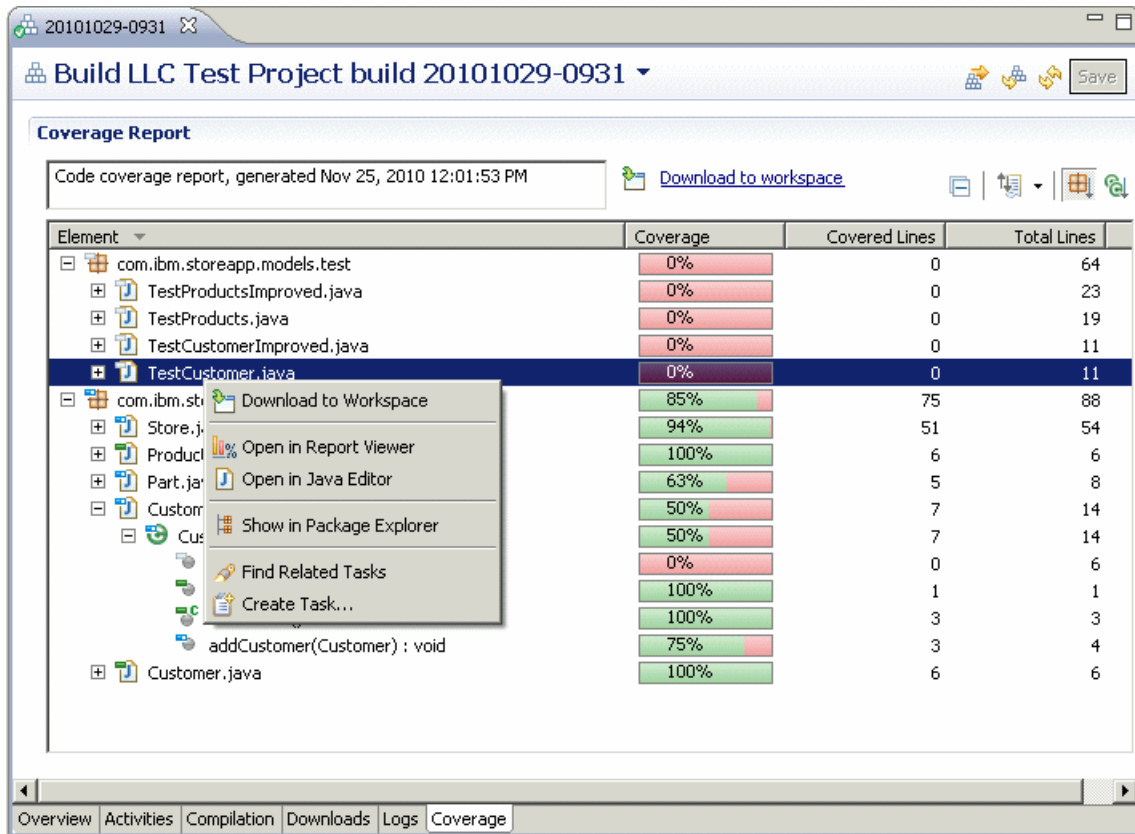


Figure 30-5 Coverage statistics in a coverage build result viewer

After making a selection in the report, you can use several actions from the context menu:

- ▶ **Download to Workspace:** This action triggers the following actions:
  - Decorate the project locally (if available) with the statistics that were generated from the build.
  - Make the statistics available as an imported launch, which allows users to generate a report (using the existing **Run** → **Code Coverage** → **Generate Report UI**) at a later time.
- ▶ **Open in Report Viewer:** This action opens the report outside of the build result view. This view provides the added advantage of being able to view an overview of the statistics.
- ▶ **Open in Java Editor:** This action opens the current selection (class or method) in a Java Editor.

- ▶ **Show in Package Explorer:** This action opens the current selection (package, class, or method) in the Package Explorer of the Java perspective.
- ▶ **Find Related Tasks:** This action attempts to identify existing work items to improve the coverage of the selected Java item.
- ▶ **Create Task:** This action creates a new work item for the selection. Ideally the selection is a package, class, or method that has low coverage. The created coverage task is pre-populated with the appropriate text based on the user's selection (see Figure 30-6).

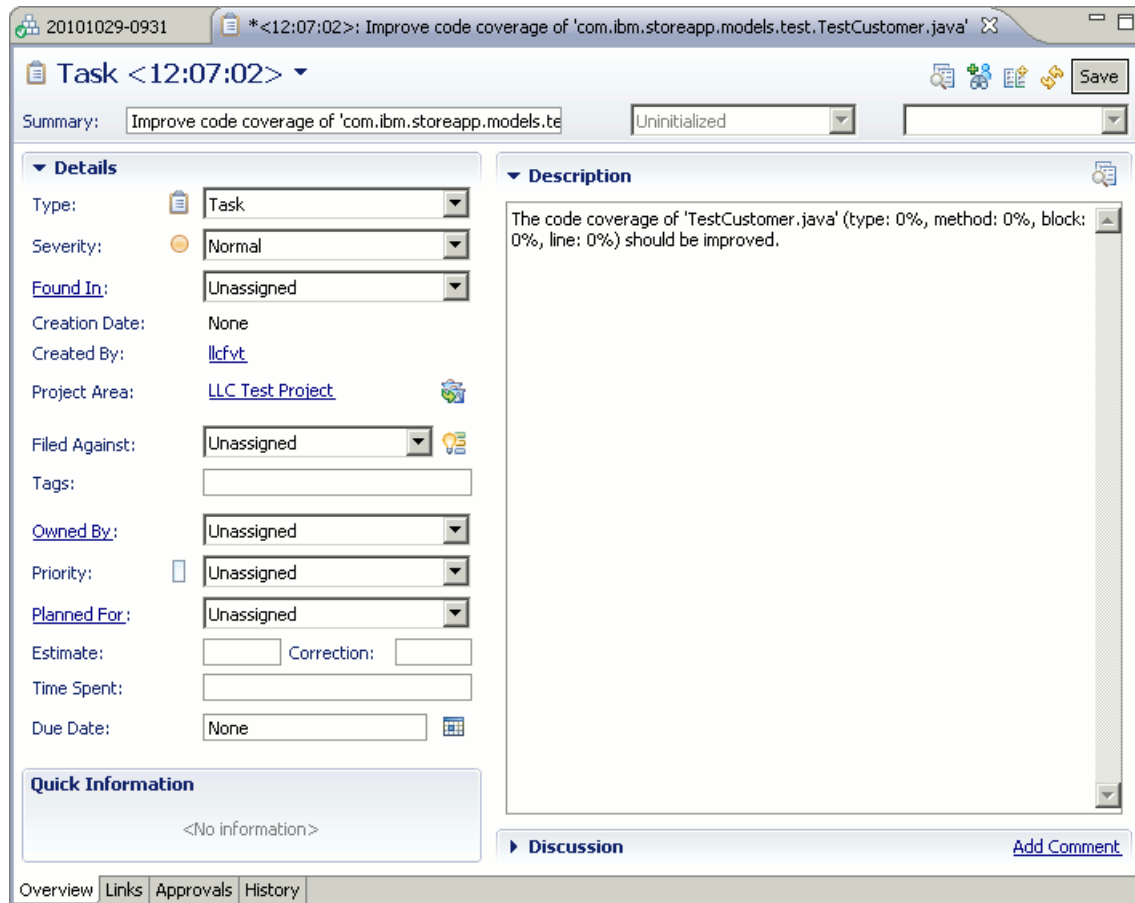


Figure 30-6 A pre-populated work item created that is based on a coverage report selection

## 30.4 Collaborative Debug

Reproducing defects is not always an easy task. Configurations can differ between testers and developers. Even getting the exact sequence of steps recorded can be a challenge. If the application is multithreaded, the timing of the steps or the speed of the machine is a factor. With Team Debug, you have the capability to efficiently and effectively eliminate the time that is wasted in resolving these types of issues.

*Collaborative Debug* allows Java-based debug sessions to be transferred between users of Rational Application Developer with Rational Team Concert. During a debug session, a team member can transfer the session to another team member. The current state and full control of the session is passed on to the receiving team member, who can then inspect the session's threads, stacks, and variables as if that team member had initiated the debugging. Another team member can also continue debugging the execution of the session. This capability speeds up the resolution of issues. Another team member can investigate an issue without having to establish the same environment and try to get the debug session to behave in the same way.

The following sections describe how to install the Collaborative Debug client extensions in Rational Application Developer, configure the Debug extension for Rational Team Concert Server, and park or transfer debug sessions between users.

### 30.4.1 Installing the Collaborative debug extensions for Rational Team Concert Client

In Rational Application Developer, ensure that the Collaborative debug extensions for Rational Team Concert Client software has been installed through the IBM Installation Manager.

Follow these steps to verify and install the extension:

1. Launch the Installation Manager and choose **Modify Packages**.
2. Choose the package group containing Rational Application Developer 8.0.1 and click **Next**.
3. Click **Next** on the Translations page.
4. Check the option for **Collaborative debug extensions for Rational Team Concert Client** if it has not been selected under the **IBM Rational Application Developer for WebSphere Software 8.0.1** group (see Figure 30-7 on page 1612) and click **Next**. If it is already selected, no further steps are required.

5. Review the Summary page and click **Modify**.
6. Close Installation Manager on success.



Figure 30-7 Selection for Collaborative debug extension installation in Installation Manager

## 30.4.2 Installing Rational Debug Extension for IBM Rational Team Concert Server

Rational Application Developer also provides an extension for the Rational Team Concert Server to enable the Collaborative Debug capability. You can install the extension using Installation Manager if you installed Rational Team Concert with Installation Manager.

**Rational Team Concert Server V2.x:** These steps are only necessary for Rational Team Concert Server V2.x because Rational Team Concert Server V3 enables the Collaborative Debug capability by default.

Follow these steps to install the extension on the server:

1. Ensure that the Rational Debug Extension for Rational Team Concert CD or electronic disk image is available on the computer where you installed Rational Team Concert Server.
2. On the computer where you installed Rational Team Concert Server, change to the **InstallerImage\_platform** subdirectory on the Rational Debug Extension for Rational Team Concert Server disk.



3. Enter one of the following commands:
  - a. To install as an Admin: **install**
  - b. To install as a non-Admin: **userinst**
4. On the first page of the Install Packages wizard, ensure that you select **IBM Rational Debug Extension for Rational Team Concert Server**. If IBM Installation Manager, Version 1.2 is not already installed, also select it. Click **Check for Other Versions and Extensions** to install the latest available version of the selected packages. If a newer version is available, it is automatically selected for installation. Click **Next**.
5. On the Licenses page, read the license agreements for the selected packages. On the leftmost side of the License page, click each package version to display its license agreement.
6. If you agree to the terms of all the license agreements, click **I accept the terms of the license agreements**.
7. Click **Next** to continue.
8. Select the package group where you installed Rational Team Concert Server.
9. Click **Install**.
10. When the installation process is complete, a message confirms the success of the process.
11. Close Installation Manager.
12. If Rational Team Concert Server is not using the default Tomcat that comes with the install image, you must update the  
`<install_dir>/server/provision_profiles/teamdebugservice_profile.ini`  
and  
`<install_dir>/server/provision_profiles/teamdebugcommon_profile.ini`  
files:  
  
Under the web address, instead of using a relative path, change it to an absolute path, for example, change  
`url=file:../ext/rad-teamdebug-update-site` to `url=file:C:/Program Files/IBM/RTC/ext/rad-teamdebug-update-site`.

If you did not install Rational Team Concert with Installation Manager, you can install the extension on the server manually. Refer to the Rational Application Developer 8.0.1 Information Center at **Installing and upgrading** → **Installing supporting software** → **Installing Rational Debug Extension for Rational Team Concert Server** → **Installing Rational Debug Extension for Rational Team Concert Server manually**.

### 30.4.3 Using Collaborative Debug

Before beginning a team debug session, each user must have an ID on the same Rational Team Concert Server. For the debug sessions to be identified as Team debug sessions, use the Team Java Launcher for your selected launch configuration type. You can verify that it is being used by displaying the launch configuration dialog window and looking at the bottom of the Main tab (Figure 30-8). To switch it to another launcher, select the hyperlink beside the Launcher type display.

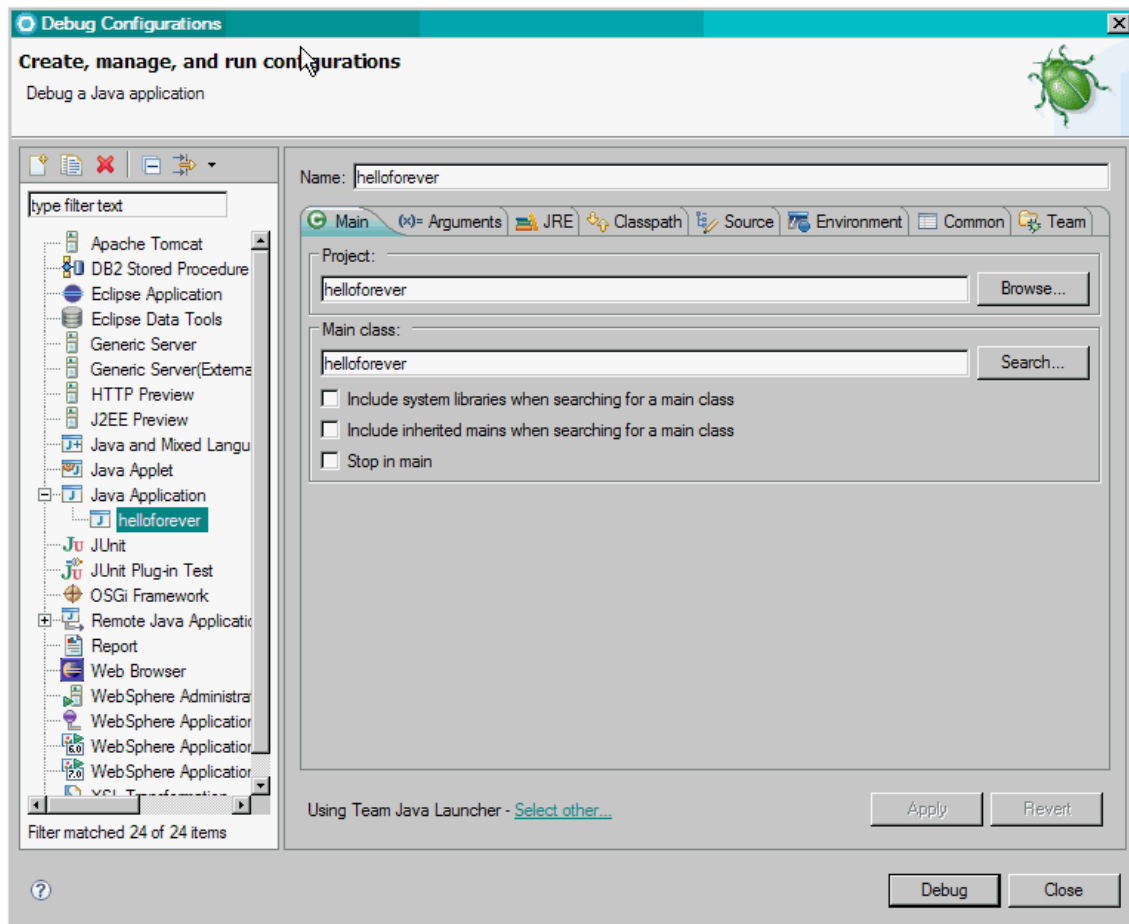


Figure 30-8 Java Application Launch Configuration using Team Java Launcher

To enable the launcher to register the debug session to the appropriate repository, click the **Team** tab and select the desired team repository.

## Transferring the debug session to another online user

The first user can launch the debug session. When the appropriate point in the debug session is reached and transfer to a second user is desired, the first user can right-click the debug target in the Debug view and choose **Transfer to User**. In the dialog box that appears, specify the second user.

## Parking the debug session

The first user can choose to park the debug session, which can be useful for several reasons:

- ▶ The user is disconnected from the remote debug session for a time and wants to park and resume debugging later.
- ▶ The user to whom the session is to be transferred is not online currently.
- ▶ The parked session is useful for the next person who is continuing the debugging, but the user to whom to transfer the session is not known yet. The first user can note the details of where the session is parked in a defect.

The first user can right-click the debug target in the Debug view and choose **Park Debug Session** (see Figure 30-9 on page 1616).

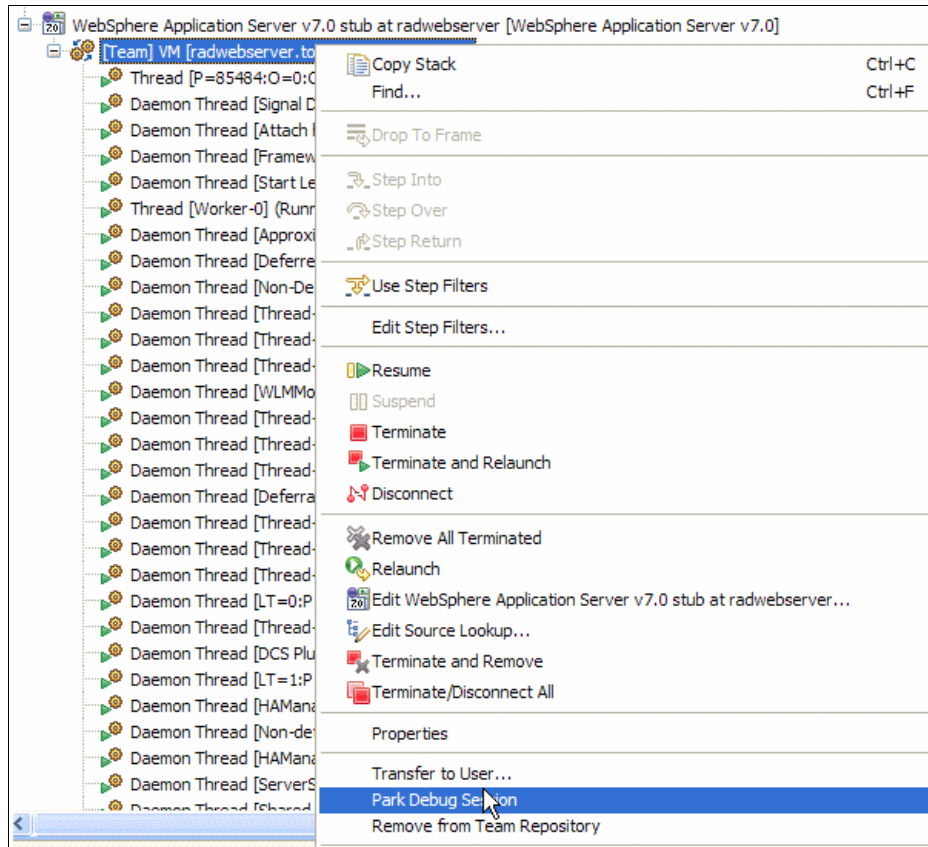


Figure 30-9 Parking a debug session from the debug view

## Taking control of a team debug session

A second user can obtain control of a team debug session in two ways. The first way is via a transfer request when the second user is online. The user currently in control of the debug session can initiate a transfer request directly to the second user. The second user receives a notification that someone wants to transfer a debug session. The second user chooses **Yes** to accept the session.

Another way in which a second user can obtain control of a team debug session is by searching for the debug session using the following steps:

1. Open the **Team Artifacts** view.
2. Ensure that you are logged in to the team repository from which you are searching for the team debug sessions.
3. Expand the **Debug** node.

4. Expand the **Search team debug sessions** node.
5. Select one of the queries that you want to run.
6. To run the query, double-click it or select the Search context menu action (see Figure 30-10).

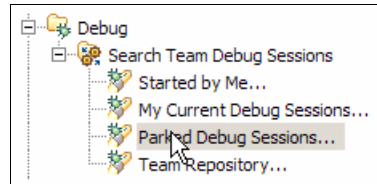


Figure 30-10 Execute query to find debug sessions

7. When the search completes, the **Team Debug** view opens. Select the parked session, right-click, and choose **Debug** (see Figure 30-11).

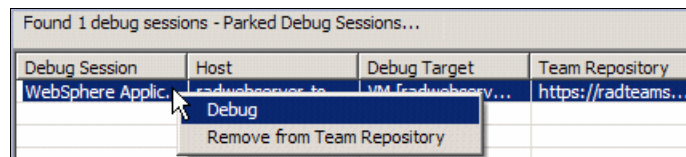


Figure 30-11 Debugging a parked session

## Loading a team debug session

Whether the session is obtained by a transfer or through selection via the Team Debug view, the sequence of loading the debug session into the second user's workspace is the same. The second user is first asked if breakpoints are to be imported from the original user. If **Yes** is selected, the second user can also choose to overwrite the current breakpoints (see Figure 30-12).

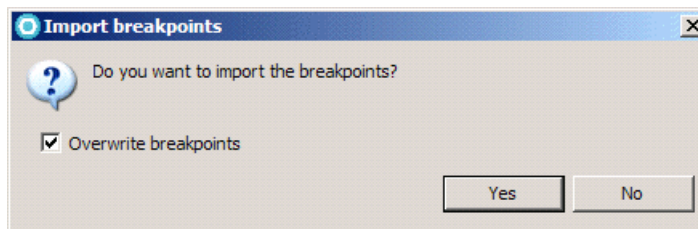


Figure 30-12 Dialog box for importing breakpoints from the previous user's debug session

The Debug view opens, and the second user can see how the debug session was left by the previous user. All information, such as variable values, threads,

and breakpoints are available (see Figure 30-13). The second user can then continue the debugging and can even park or transfer the session back to the first user or other users in the same team.

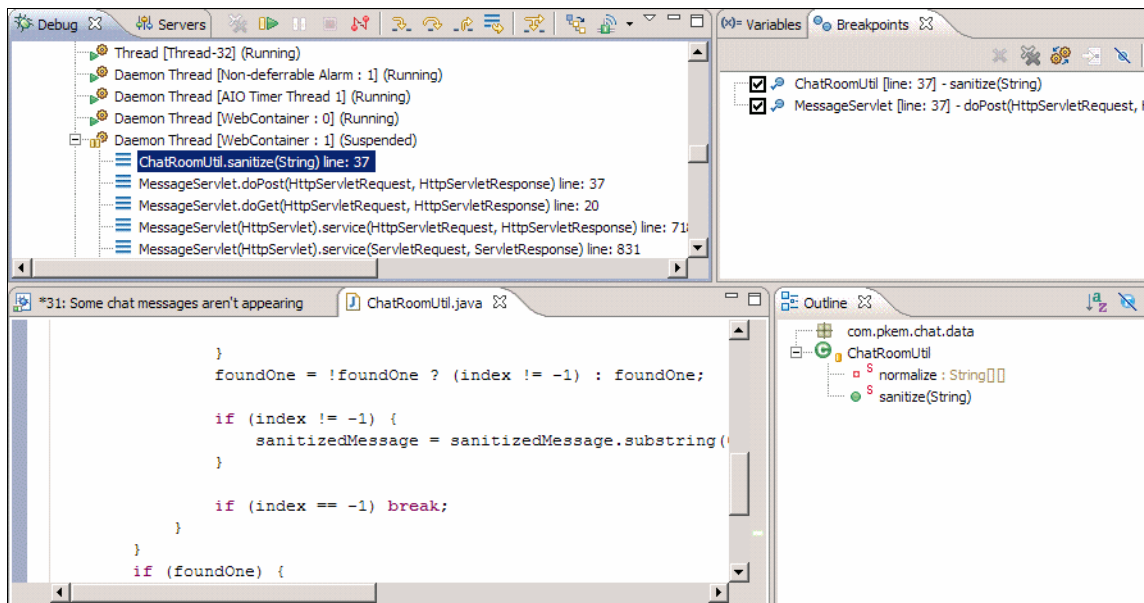


Figure 30-13 Debug perspective for a recently loaded debug session



# IBM Rational ClearCase

This chapter describes the integration of IBM Rational Application Developer and IBM Rational ClearCase, using the ClearCase SCM Adapter that is included in Rational Application Developer and the optionally installable ClearCase Remote Client Extension.

In this chapter, we describe the general team infrastructure of Rational Application Developer, the specific features that are related to ClearCase and the two available integration plug-ins. We introduce the leading practices for managing workspaces in relation to ClearCase views and cover usage of team project sets. We describe two typical workflows: a single branch development scenario in Base ClearCase with dynamic views using the software configuration management (SCM) Adapter and a traditional parallel development Unified Change Management (UCM) scenario with web views using the ClearCase Remote Client.

This chapter contains the following topics:

- ▶ Rational Application Developer team support
- ▶ Integrating Rational Application Developer with ClearCase
- ▶ ClearCase SCM Adapter
- ▶ ClearCase Remote Client
- ▶ ClearCase views and Rational Application Developer workspaces

- ▶ Populating Rational Application Developer workspaces: Using Team Project Set files
- ▶ Working in Base ClearCase with SCM Adapter and dynamic views
- ▶ Working in ClearCase UCM with ClearCase Remote Client



## 31.1 Rational Application Developer team support

Rational Application Developer supports multiple team providers, which allow the users to version artifacts from inside the product. The generic team support infrastructure offers the following preferences, which influence the behavior of the ClearCase integration:

- ▶ **Window** → **Preferences:**
  - **Team**
  - **Team** → **File Content**
  - **Team** → **Ignored Resources**
  - **Team** → **Model**

The generic team support infrastructure offers the following context menu:  
**Team** → **Share Project**.

Finally, you can set individual files and folders to be *Derived*, which excludes them from source control. In the following sections, we describe these options and menus in more detail.

### 31.1.1 Team preferences

You can get detailed information about the general preferences that are found in the team:

<http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/org.eclipse.platform.doc.user/reference/ref-19.htm>

The Preferences options in **Team** → **File Content** indicate which types of files to handle as Binary and which types of files to handle as ASCII Text. ClearCase associates file types to type managers, as described at this website:

[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc\\_proj.doc/c\\_bcc\\_eltypprustypmgr.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc_proj.doc/c_bcc_eltypprustypmgr.htm)

Among the files that are marked as ASCII text, ClearCase creates separate element types, such as *text* or *xml*, depending on the actual file content.

The Preferences options in **Team** → **Ignored Resources** influence the choice of which files are added to source control, although there are other mechanisms for this purpose (see also 31.1.3, “Derived files and folders” on page 1622). If you want to exclude a folder called `target` from being added to source control, you can add the pattern `target` to the Ignored Resources list.

The Preferences options in **Team** → **Model** refer to the support for handling related artifacts as a logical unit in regard to synchronization operations. For

example, you can think of a file move as two separate operations: one file deletion from the original location and one file addition in the target location. With Logical Model support, a source control provider can handle these related operations as a single logical unit, which avoids accidentally recording only the deletion but not the addition. *Logical Model support is present in the ClearCase Remote Client but not in the ClearCase SCM Adapter.*

For more information, click **For Logical Model integrations, perform merges automatically, if possible**. If set, all files in the Logical Model are merged automatically if possible. If manual merges are required, you must resolve each conflict using the merge tool. This option is only available if the ClearCase Remote Client is integrated with a product that supports Logical Models. For more information, see:

[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.crc.help.doc/topics/u\\_prefs\\_compare\\_merge.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.crc.help.doc/topics/u_prefs_compare_merge.htm)

### 31.1.2 Team context menu

The Team context menu is available on all projects, files, and folders in all the explorer views (Navigator, Package Explorer, Project Explorer, and Enterprise Explorer). The menu entry **Share project** is available on projects only. This option is used to share projects in a configuration management tool. In the case of ClearCase, sharing a project that is located inside the workspace physically moves it into the selected ClearCase view. Every source control provider offers additional menu entries inside the Team menu.

### 31.1.3 Derived files and folders

If you want to exclude specific files or folders from source control and it is not feasible to add a naming pattern to the Ignored Resources list, you can use the Derived property, which can be applied in the following way:

1. Right-click a file or folder in the Project Explorer or Enterprise Explorer.
2. Select **Properties** → **Resources** → **Derived**.

If you apply this property to a folder, it applies to all files and folders that are contained in the folder. If you apply this property to a file, it applies to the file only.

Derived resources are typically not kept in a team repository, because they clutter the repository, change regularly, and can be recreated from their source files. It is impractical for team providers to make decisions about which files are derived. The resource application programming interface (API) provides a common mechanism for plug-ins to indicate the derived resources that the

plug-ins create. Rational Application Developer takes advantage of this API and automatically creates certain types of files as *Derived*.

## 31.2 Integrating Rational Application Developer with ClearCase

This product integrates with ClearCase V7.1.1.x and V7.1.2. For more information regarding the supported ClearCase versions, refer to the following information centers:

- ▶ [https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m0/index.jsp?topic=/com.ibm.rational.clearcase.help.ic.doc/helpindex\\_clearcase.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m0/index.jsp?topic=/com.ibm.rational.clearcase.help.ic.doc/helpindex_clearcase.htm)
- ▶ [https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.help.ic.doc/helpindex\\_clearcase.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.help.ic.doc/helpindex_clearcase.htm)

All the examples in this chapter are based on ClearCase V7.1.2. For the installation of ClearCase Server and Client components, refer to “Installing IBM Rational ClearCase” on page 1842.

You can set up the integration using two technologies: ClearCase SCM Adapter and ClearCase Remote Client.

ClearCase SCM Adapter is an Eclipse Plug-in that is delivered by Rational Application Developer. Currently, the SCM Adapter is the only form of integration that supports ClearCase dynamic views. We demonstrate a scenario that uses dynamic views with the SCM Adapter.

The ClearCase Remote Client offers broader functionality than the SCM Adapter, including a ClearCase perspective in Rational Application Developer. In this perspective, it is possible to browse ClearCase web views. With the SCM Adapter, you must browse views using an external tool. ClearCase Remote Client also has integrated tooling for other functions that require launching external executables with the SCM Adapter (Tree view and Find Checkouts). In the next two sections, we describe the major features of these plug-ins.

### 31.2.1 ClearCase terminology

This chapter does not aim to thoroughly introduce ClearCase, but a review of the terminology is necessary to ensure the readability of the following sections.

We use these terms in this chapter:

<b>VOB</b>	A versioned object base (VOB) is the permanent data repository in which you store files, directories, and metadata.
<b>View</b>	A typical view contains a combination of versioned artifacts (versions of VOB elements) and unversioned artifacts (view-private files that do not exist in any VOB).
<b>Dynamic view</b>	A dynamic view provides transparent access to versions of elements in the VOB and to view-private objects. A dynamic view can access any version of an element that is selected by the view's configuration specifications as soon as the version is checked in.
<b>Snapshot view</b>	A snapshot view contains copies of versions of specified VOB elements, along with view-private objects. You must update snapshot views manually to access new versions that have been checked in to the VOB from other views.
<b>Web view</b>	The web view is similar to the snapshot view, but it is accessed from the Rational ClearCase Remote Client.
<b>Storage location</b>	A server storage location is a shared directory on a Rational ClearCase server that is displayed in a list of recommended storage locations when a user creates a VOB or a view from a graphical user interface (GUI). You can create storage locations for VOBs or views during server setup or by using the <b>cleartool mkstgloc</b> command or the Rational ClearCase administrative console.
<b>Config spec</b>	When you create views for your project, you must prepare one or more configuration specifications (config specs). The rules in a view's config spec determine which versions are visible in the view, and for snapshot views, which elements are loaded in the view.
<b>Element</b>	Items that are under ClearCase source control (version control) generally are referred to as <i>elements</i> . An element can be a Java source file, an .xml file, a diagram file, and so on.
<b>View-private</b>	A view-private item is present in a ClearCase view but not in any VOB (it has not been added to source control).
<b>Check in</b>	The Check in action commits resource modifications to the repository (VOB).

<b>Check out</b>	Checking out makes file or directory versions writable in your view.
<b>Branch</b>	ClearCase uses branches to organize the versions of files, directories, and objects that are placed under version control. A <i>branch</i> is an object that specifies a linear sequence of the versions of an element. The entire set of an element's versions is called a <i>version tree</i> . By default, ClearCase provides for every single element in a VOB one principal branch, which is called the <i>main branch</i> . This main branch also can have subbranches.

We introduce the terminology that is related to Unified Change Management (UCM) in 31.8, “Working in ClearCase UCM with ClearCase Remote Client” on page 1666.

## 31.3 ClearCase SCM Adapter

The ClearCase SCM Adapter is not a stand-alone client. It requires the presence of the Rational ClearCase Client components on the same computer, and it offers menus that invoke a number of native tools that are part of the Rational ClearCase Client.

You can obtain more information in the installed product help, which contains a top-level node called Rational ClearCase SCM Adapter.

This node of product help is not available in the Rational Application Developer v8.0 Information Center.

### 31.3.1 Installing ClearCase SCM Adapter

You can install the SCM Adapter by selecting **Rational Lifecycle Integrations** → **Rational ClearCase SCM Adapter** in the IBM Installation Manager (Figure 31-1 on page 1626).

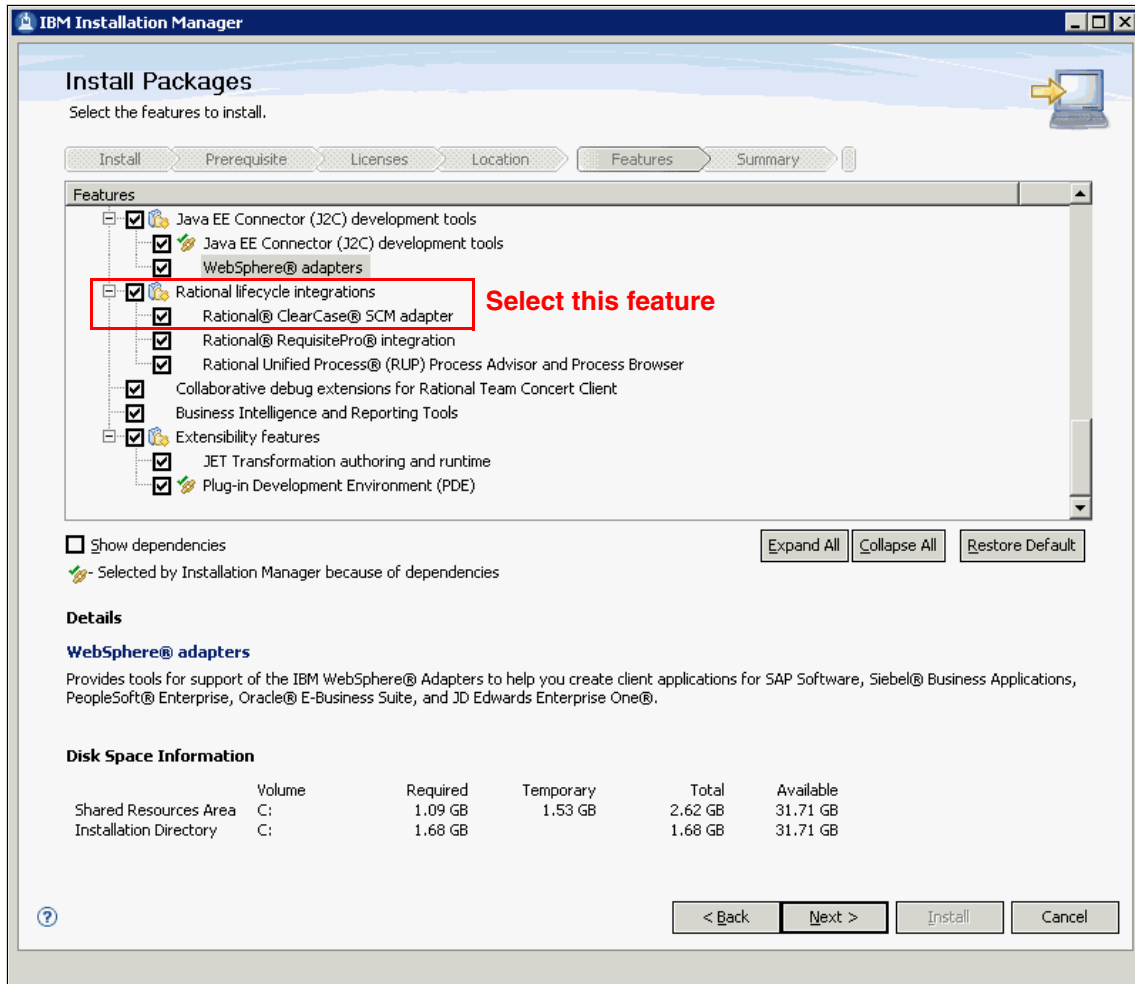


Figure 31-1 Installing Rational ClearCase SCM Adapter in Rational Application Developer

**Important:** Rational Application Developer, because it is a Java application, is case-sensitive. On the Microsoft Windows operating systems, the default installation options for ClearCase multiversion file system (MVFS) are *Not Case-sensitive* (selected) or *Case Preserving* (not selected).

You must change these default options. Either select both of them or clear both of them. Otherwise, ClearCase changes the file names and directory names to lowercase, which makes it impossible for Rational Application Developer to recognize the files and directories.

You can make the selection during the installation, as shown in Figure 31-2.

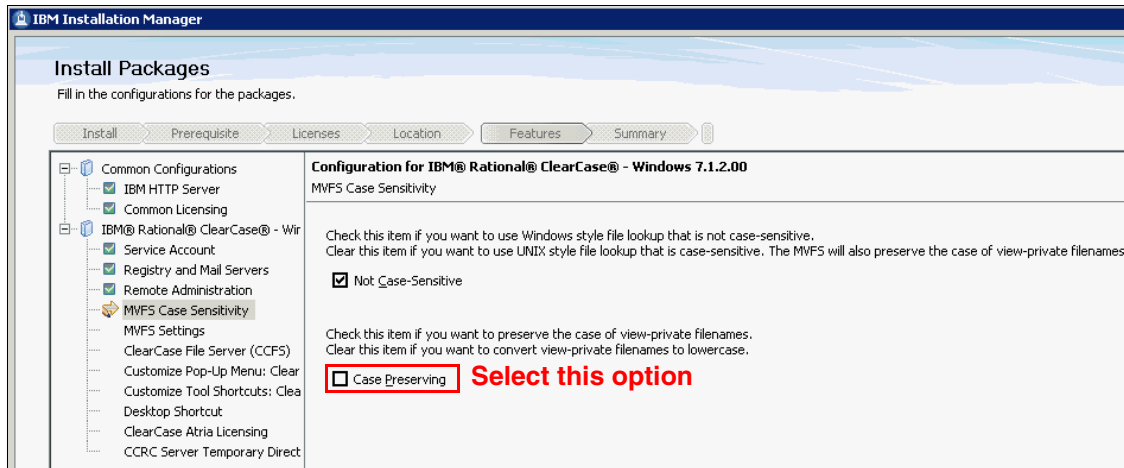


Figure 31-2 Selecting Case Preserving for MVFS during the ClearCase installation

If ClearCase has already been installed, you can change this setting by clicking the **ClearCase** icon in the Microsoft Windows Control panel. Select the **MVFS** tab. Select **Case Preserving**, as shown in Figure 31-3 on page 1628.

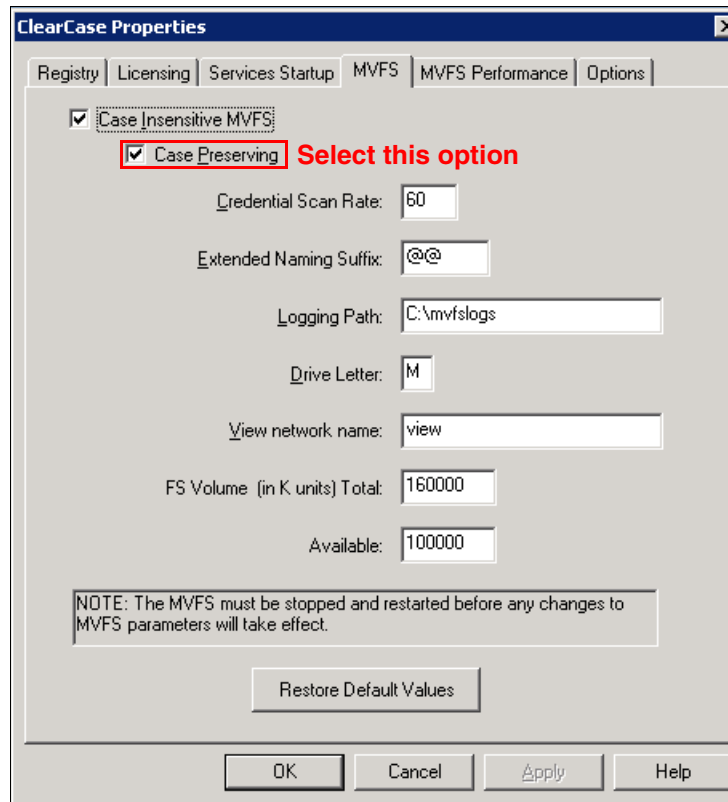


Figure 31-3 Changing MVFS to Case Preserving after the ClearCase installation

### 31.3.2 Connecting to ClearCase with the SCM Adapter

The first step to utilize the SCM Adapter requires enabling the corresponding capability:

1. Click **Windows** → **Preferences**.
2. Click **General** → **Capabilities** → **Team**.
3. Select **Advanced**.
4. Select **ClearCase SCM Adapter**, as shown in Figure 31-4 on page 1629.

After you enable this capability, in the Java Platform, Enterprise Edition (Java EE) perspective, you see a new top-level menu called ClearCase.



**ClearCase menu:** The ClearCase menu is enabled in other common perspectives as well. If for any reason you do not see this menu in the perspective you are currently using, you can add it by performing these actions:

1. Click **Window** → **Customize Perspective**.
2. Select the tab **Commands group availability**.
3. Select **ClearCase**.

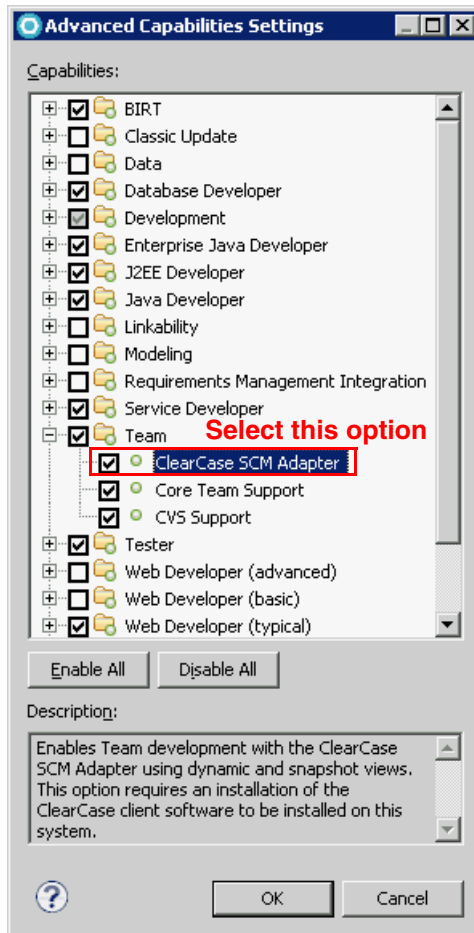


Figure 31-4 Enabling ClearCase SCM Adapter capability

To test that the integration is functional, perform the following steps:

1. Click **ClearCase** → **Connect to Rational ClearCase**, as shown in Figure 31-5.

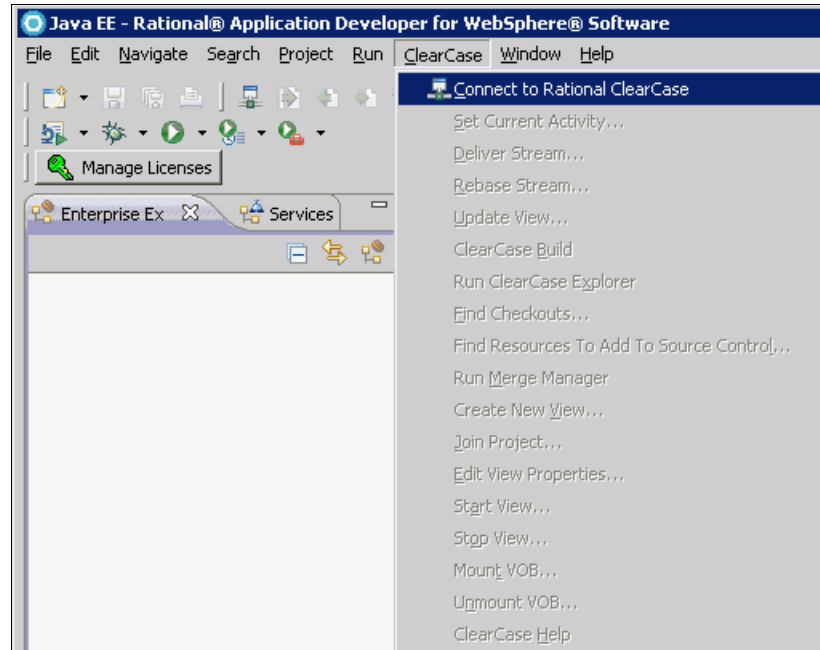


Figure 31-5 SCM Adapter ClearCase menu before connection

2. If the connection is successful, you see icons that become enabled in the toolbar and additional menu entries become accessible, as shown in Figure 31-6 on page 1631.

**Tip:** If the connection is not successful, check if the ClearCase bin directory is added to the PATH environment variable. On the Microsoft Windows operating system, the default location of this directory is:  
C:\Program Files\IBM\RationalSDLC\ClearCase\bin

This directory is required so that the ClearCase SCM Adapter can locate the native code in the ClearCase Client installation. An easy way to check if this directory is on the system PATH consists of typing the following command from the command prompt: **cleartool -ver**. If the **cleartool** command cannot be found, you must update the PATH variable, and then you must restart Rational Application Developer.

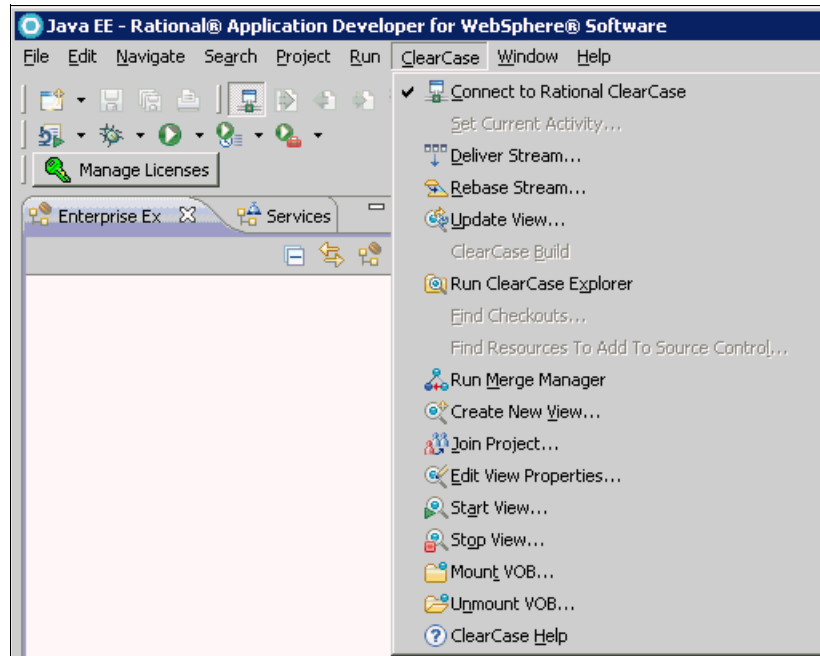







Figure 31-6 SCM Adapter ClearCase menu after a successful connection













To connect automatically to ClearCase on start-up, perform the following steps:

1. Click **Windows** → **Preferences**.
2. Click **Team** → **ClearCase SCM Adapter**.
3. Select **Automatically connect to ClearCase on startup**.

Table 31-1 shows the available ClearCase icons.

Table 31-1 ClearCase SCM Adapter icons

Icon	Name	Notes
	Add to Source Control	Moves the currently selected element into a view and adds it to source control
	ClearCase Build	Starts a software build using <b>omake</b> or <b>clearmake</b>
	Check In	Creates a new version of a file or folder element that you previously checked out
	Check Out	Creates an editable copy of the artifact
	Compare with Previous Version	Compares with the previous version that is stored in the ClearCase VOB

Icon	Name	Notes
	Connect to ClearCase	There is a preference to connect automatically on start-up
	Deliver Stream	UCM only: Delivers changes from the current stream to the integration stream
	Rebase Stream	UCM only: Merges changes from the integration stream to the current stream
	Refresh Status	Refreshes the ClearCase status (checked in/checked out and so on)
	Run ClearCase Explorer	Opens the ClearCase Explorer as an external tool
	Set Current Activity	UCM only: Associates a UCM activity with the current ClearCase view
	Show Properties	Shows the ClearCase properties of the element
	Show Version Tree	Opens the Version Tree as an external tool
	Undo Checkout	Discards the checkout
	Update	Updates the currently selected elements in a snapshot view, based on the configuration specification of the view
	Update View	Synchronizes the selected files and directories in a snapshot view with the contents of the VOB
	Help	Launches the ClearCase help

### 31.3.3 ClearCase SCM Adapter preferences

To set ClearCase SCM Adapter preferences, select **Window** → **Preferences** → **Team** → **ClearCase SCM Adapter**.

The complete description of these preferences is available in the product help. The preferences that are listed in Table 31-2 are important for the proper use of Rational Application Developer (the default values are in **bold**).

Table 31-2 Preferences related to automatic checkout

Preference name	Possible values
When checked in files are edited by an internal, interactive editor	<b>Prompt to Checkout</b> Automatically Checkout Do nothing

Preference name	Possible values
When checked in files are edited by an internal, non-interactive editor	<b>Automatically Checkout</b> Do nothing
When checked in files are saved by an internal editor	<b>Automatically Checkout</b> Do nothing
For all above preferences, if checkout is allowed.	<b>Do not hijack in snapshot views</b>
	Always hijack in snapshot views
	Hijack in snapshot views when disconnected

Rational ClearCase is a *pessimistic source control provider*, which means that it is necessary to check out artifacts before editing them. Typically, checkouts are *reserved*, which means that only one view can check out an artifact at any given time (this characteristic is important only in case multiple views access the same branch or stream). The user can always check out files manually before trying to edit them. However, Rational Application Developer editors need to be able to request checkouts from ClearCase, in case the user forgets to check out the file manually. In many cases, Rational Application Developer needs to edit metadata files that are contained inside the projects, of which the user is not aware (for example, inside the `.settings` folder in faceted projects).

It is therefore important that you do not select the values “Do nothing” values in the preferences that are listed in Table 31-2 on page 1632. For example, if you select “Do nothing” in these preferences, and you try to update a checked in web page using the Page Designer (for example, dropping an icon from the Palette onto the Design pane), Rational Application Developer produces an error dialog window. See Figure 31-7 on page 1634.

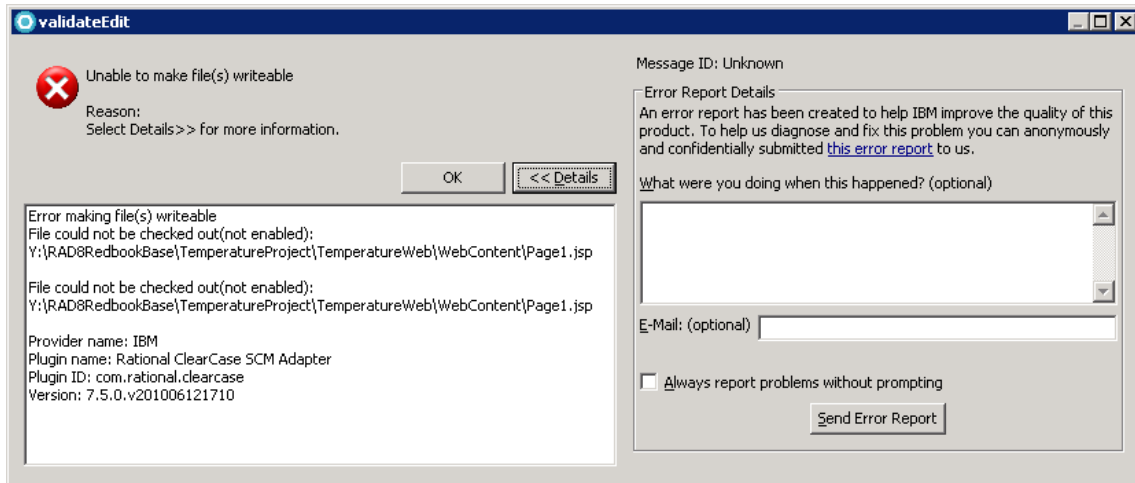


Figure 31-7 Failure to edit a .jsp file due to preferences for automatic checkout

If you use snapshot views, you can choose to hijack resources when a checkout is possible. A *hijacked file* is a modified file that was not previously checked out. After you have hijacked a file, you can take a number of actions to either merge the latest version in the VOB with the contents of the hijacked file or replace the hijacked file with the version in the VOB. The following website describes all of the available options, which are summarized in Table 31-3:

[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.dev.doc/topics/cc\\_dev/about\\_hijacked\\_files.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.dev.doc/topics/cc_dev/about_hijacked_files.htm)

Table 31-3 Miscellaneous SCM Adapter preferences

Preference name	Preference value
When workspace is closed	<b>Prompt to checkin</b> Automatically checkin Do nothing
When new resources are added	<b>Prompt to add to source control</b> Automatically add to source control Do nothing
ClearCase decorations	<b>Enable Icon Decorations</b> Enable Text Decorations Enable Icon and Text Decorations
When a parent directory is automatically checked out to move, rename, or delete an element	<b>Do Nothing</b> Automatically check in parent directory

Preference name	Preference value
Save dirty editors before ClearCase operations	<b>Prompt to save all editors</b> Automatically save all editors Do nothing
Build command	<b>omake -s -f</b> " <b>\${ProjPath}\${ProjName}.mak</b> " clearmake -f " <b>\${ProjPath}\${ProjName}.mak</b> "
Automatically connect to ClearCase at start-up	(default: not selected)
[Windows only indicator] Set default to check out files after adding them to source control	(default: not selected)
Perform Refresh Status operations recursively	(default: selected)
Decorate project names with viewtags	(default: selected)
Request status information on demand only	(default: selected)
[Windows only indicator] Advanced Options	Figure 31-8 on page 1636

In ClearCase, directories are versioned. Every time that a new file must be renamed, added to, or removed from a directory, the directory must be checked out. Before other views can see the change (renamed, added to, or deleted file), the directory must be checked in, to register the file modification, addition, or deletion.

Among the Advanced Options that are shown in Figure 31-8 on page 1636, you can customize the way that checkout and check-in are performed. By default, checkouts are *reserved*, meaning that only one view at a time can check out the same file on the same branch or stream. You can specify what action to take when the file that you intend to check out is already checked out or *reserved* by another view. In this case, you can choose to check it out *unreserved*. After you perform an unreserved checkout, a merge session must occur to reconcile the changes that have been made in parallel by the two users.

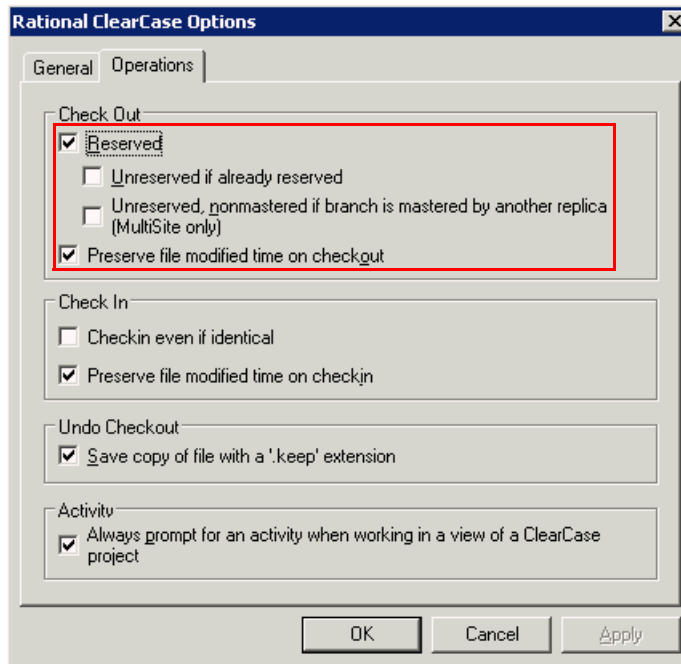


Figure 31-8 Advanced options for ClearCase SCM Adapter

To set the ClearCase SCM Adapter Diff Merge preferences, select **Window** → **Preferences: Team** → **ClearCase SCM Adapter** → **Diff Merge Patterns**.

The file name patterns that you add to this list are compared and merged using the ClearCase Diff Merge tool for that particular file type, instead of the corresponding Eclipse tool.

Unified Modeling Language (UML) diagrams that are created by Rational Application Developer always are compared and merged with a specific Rational Application Developer Compare and Merge tool, in order to protect the referential integrity of these complex artifacts.

### 31.3.4 Clearcase SCM Adapter and dynamic views

A *ClearCase dynamic view* receives automatic updates if newer elements or versions are added to the VOBs that are loaded by the view. This capability is possible due to a particular type of file system called a *multiversion file system* (MVFS).



See this website for more information:

[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc\\_admin.doc/topics/c\\_viewadm\\_dynamic\\_mvfs.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc_admin.doc/topics/c_viewadm_dynamic_mvfs.htm)

Rational Application Developer does not work directly with the files and folders from the file system. It builds an in-memory structure of these artifacts by using the Eclipse Resources plug-in. Therefore, the in-memory resources need to be updated with the changes that the dynamic view makes available.

Follow these steps to activate file system support for ClearCase dynamic views:

1. Enable the ClearCase MVFS Support preference by selecting **Window** → **Preferences: Workbench** → **Refresh workspace automatically**.
2. Select **Window** → **Preferences: Team** → **ClearCase MVFS Support**.
3. Select **Enable ClearCase Dynamic View filesystem support**.
4. Optional: Change the **Refresh polling interval** (the default is 20 seconds).

After you set these options, your workspace shows new elements or a newer version of the existing elements shortly after these elements are added to the VOB, even if adding the elements to the VOB occurs externally to Rational Application Developer. These resource changes can trigger automatic build and automatic publish operations.

## 31.4 ClearCase Remote Client

The ClearCase Remote Client is available in two versions: Stand-alone and Extension. You can download the Extension from the following location:

<http://www-01.ibm.com/support/docview.wss?rs=984&uid=swg24028116>

We describe the steps to install the Extension on Rational Application Developer in “Installing IBM Rational ClearCase Remote Client Extension” on page 1855.

ClearCase Remote Client does not require the installation of any other ClearCase Client on the same computer hosting Rational Application Developer, because it connects to the ClearCase Change Management server (CMS) via HTTP. ClearCase Remote Client uses web views, which are similar to snapshot views, because web views do not propagate changes automatically (dynamic views propagate changes automatically).

After you have installed the ClearCase Remote Client, you see a new perspective called ClearCase Explorer and various preferences and menus.

You can obtain extensive information about developing software with ClearCase Remote Client in the information center:

[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.ccrcl.help.doc/topics/c\\_ccrc\\_dev\\_container.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.ccrcl.help.doc/topics/c_ccrc_dev_container.htm)

### 31.4.1 Connecting to ClearCase with the ClearCase Remote Client

You can connect to ClearCase with the ClearCase Remote Client by clicking the **ClearCase** menu in the toolbar at the top of the Rational Application Developer Java EE perspective and selecting **Connect** (Figure 31-9).

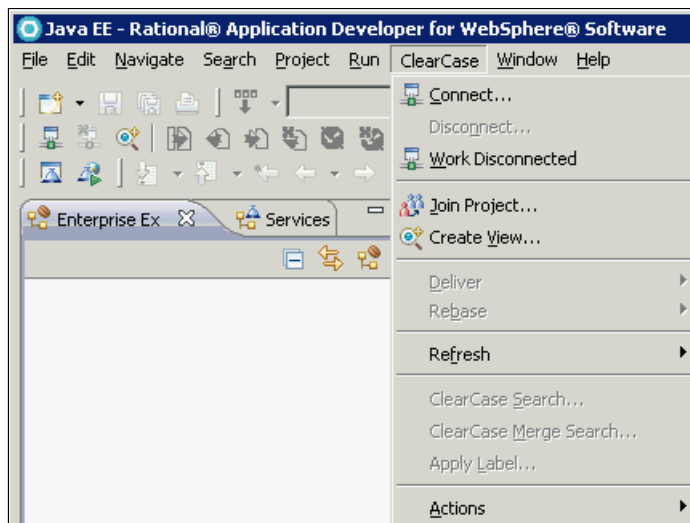


Figure 31-9 Connecting to ClearCase Remote Client

The login window opens (Figure 31-10 on page 1639). You need to enter the Server URL in the form `http://server:port/TeamWeb/services/Team`, as directed by the tooltip. The default port value is 12080.

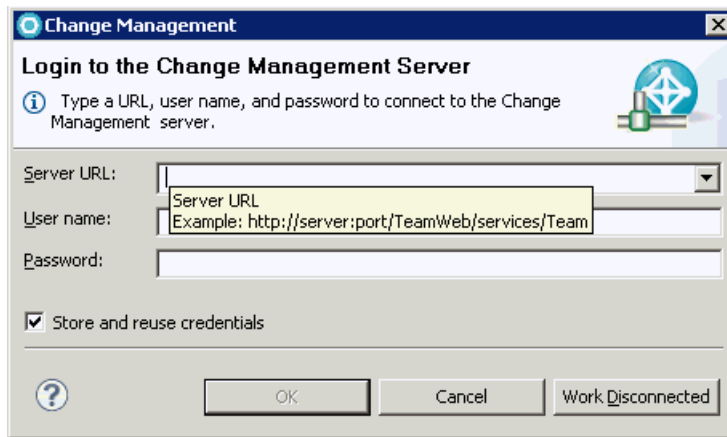


Figure 31-10 Login to the Change Management Server window

### 31.4.2 ClearCase Remote Client preferences

You can modify ClearCase Remote Client Preferences by selecting **Window** → **Preferences** → **Team** → **ClearCase Remote Client**.

In the ClearCase Common Dialog Box preferences, you can specify how the dialog boxes for check-in, checkout, and add to source control behave by default.

For example, you can choose the default actions when attempting to check in a version that is identical to a predecessor:

- ▶ Check in even if the version is identical to its predecessor
- ▶ Undo the checkout if the version is identical to its predecessor
- ▶ Leave the version checked out if it is identical to its predecessor

For checkout, you can choose to show the options:

- ▶ Reserved checkout
- ▶ Prefer reserved, unreserved if necessary
- ▶ Unreserved checkout

You can choose the default behavior:

- ▶ Always reserved
- ▶ Prefer reserved, unreserved if necessary
- ▶ Always unreserved

You can also choose what to do when there are newer versions in the VOB than the version that is currently loaded in your view:

- ▶ Check out the version that is loaded even if it is not the latest
- ▶ Check out the view-selected version if the version loaded is not the latest

You can restrict the checkout in case logical model integrations are used. This capability is important, for instance, in the case of Unified Modeling Language (UML) models that are split in multiple fragments, and Logical Model integration dictates that the entire model with all its fragments is checked out at one time, instead of in selected models or fragments. You can select for Logical Model integrations, to only include the physical files selected for checkout.

You can use the ClearCase Compare/Merge preferences dialog box to specify styles, behaviors, and tool preferences for comparing and merging files.

You can use the ClearCase Pending Changes preferences dialog box to specify how to handle changes in base ClearCase and in UCM.

You can use the ClearCase Rebase and Deliver preferences dialog box to customize the manner in which ClearCase handles rebase and deliver operations. For example, you can select to check in all checkouts before starting the operation, which automatically checks in all checkouts in your relevant activities before the default deliver operation begins.

For a detailed description of all available preferences, see this website:

[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.ccrs.help.doc/topics/u\\_prefs.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.ccrs.help.doc/topics/u_prefs.htm)

### 31.4.3 ClearCase Remote Client menus

ClearCase Remote Client offers a ClearCase menu on the toolbar of Rational Application Developer, with the contents that are shown in Figure 31-11 on page 1641 (the menu looks like this example only after you have successfully connected to ClearCase by selecting the Connect menu entry).

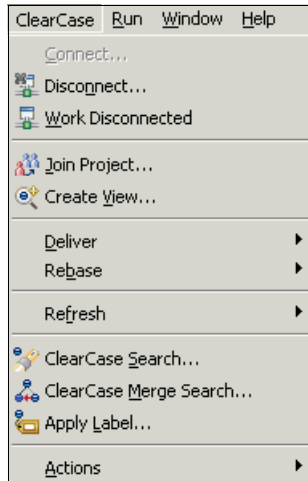


Figure 31-11 ClearCase Remote Client Toolbar menu

This menu offers functionality that is independent of the selection of a specific resource, such as the ability to connect and disconnect from the Change Management Server, the ability to join a UCM project or to create a view, and the ability to deliver and rebase.

ClearCase Remote Client offers a second-level context menu that is available from the Team first-level menu in the Enterprise Explorer, Project Explorer, Package Explorer, and Navigator, as shown in Figure 31-12 on page 1642. Similar functionality is also available as a first-level menu in the ClearCase Navigator view.

This menu is context-sensitive: it enables only those functions that are applicable based on the current ClearCase state of the selected resource. For example, for a resource that is currently checked in, it shows the options to Check Out and Hijack, and Check In and Undo Check Out are disabled.

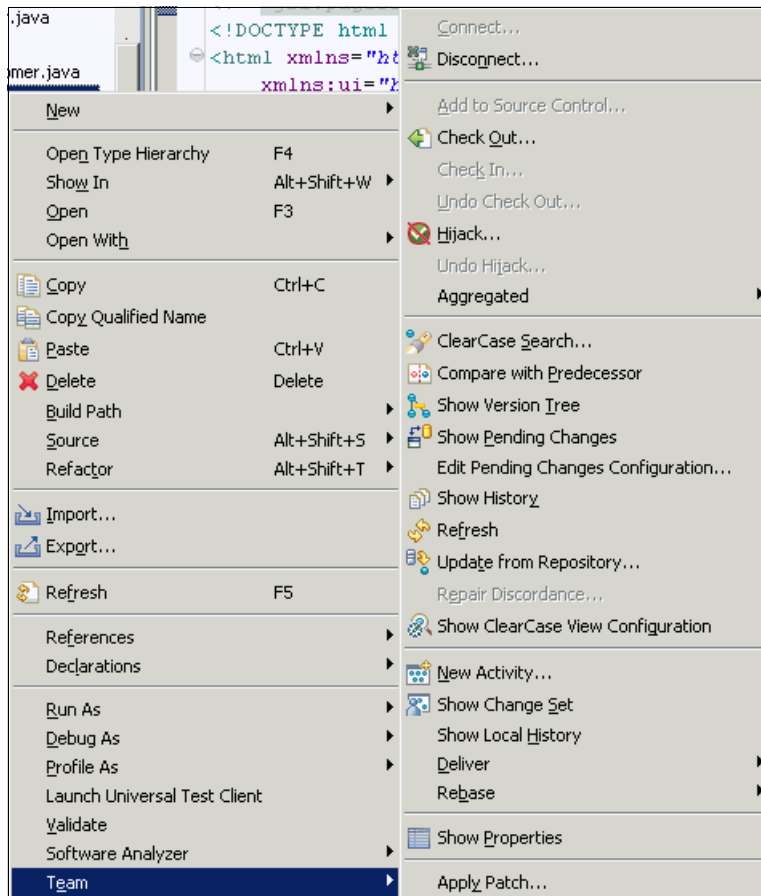


Figure 31-12 ClearCase Remote Client Team context menu

### 31.4.4 ClearCase Explorer perspective

The major distinguishing feature of ClearCase Remote Client compared to the SCM Adapter for a Rational Application Developer user is the addition of a dedicated perspective, which is called ClearCase Explorer. The ClearCase Explorer perspective includes toolbars, views, wizards, and dialog boxes that provide access to ClearCase functionality.

To access the ClearCase Explorer perspective, click **Window** → **Open Perspective** → **Other** and then select **ClearCase Explorer**.








## Toolbars

The following page describes all the available icons in the Toolbars:

[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.ccrs.help.doc/topics/u\\_cctoolbars.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.ccrs.help.doc/topics/u_cctoolbars.htm)

The icons that are shown in Table 31-4 refer to the most common actions that we will perform in the next sections.

Table 31-4 Most used ClearCase Remote Client toolbar icons

Icon	Name	Notes
	Add to source control	Moves the artifact into a location inside a view and creates an element.
	Check out	Makes a reserved or unreserved checkout so that you can write to the resource.
	Check in	Commits changes to the repository by creating a new version of the element in the VOB.
	Join a UCM project and Create UCM views	Selects a UCM project to join. Creates a development stream, a development view and an integration view.
	Import Eclipse project or Project Set into workspace	References the projects or project sets in the current workspace (without copying them into the workspace directory).
	Rebase Stream	Rebases a UCM stream to the parent stream's recommended baseline or to a separate baseline.
	Deliver From Stream	Delivers your work from a UCM development view to an integration stream.

## ClearCase Explorer views

The default configuration of the ClearCase Explorer perspective includes two views: ClearCase Navigator and ClearCase Details.

### ClearCase Navigator

To access the ClearCase Navigator from any perspective, select **Window** → **Show View** → **ClearCase Navigator**. If ClearCase Navigator does not appear in the list, click **Other** → **ClearCase** and then select **ClearCase Navigator**.

The ClearCase Navigator view displays a list of all Rational ClearCase views on the local host as well as a list of ClearCase servers to which you have been

connected previously. You can use the ClearCase Navigator to explore the contents of the objects in each of the lists. You can also use the view to explore the contents of remote Rational ClearCase repositories (VOBs) and select resources to load into a local Rational ClearCase view.

If no Rational ClearCase views appear in the list, use one of the ClearCase view creation wizards to create a view. If your computer already hosts views that were created with the ClearCase web interface, you can use the ClearCase Navigator's **Add Existing Web View** option (**View Menu** → **Add Existing Web View**) to make them visible in the Navigator, as shown in Figure 31-13.

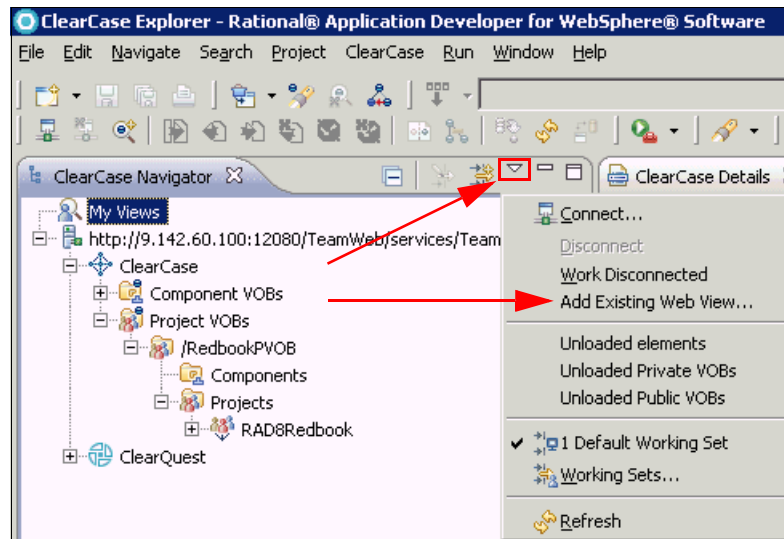
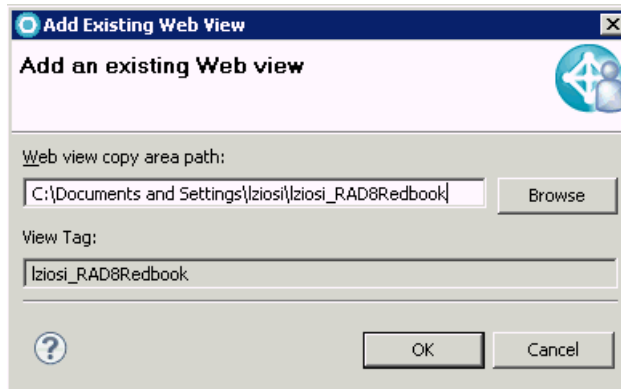


Figure 31-13 Add Existing Web View to ClearCase Navigator

You must browse to the web view copy area path, as shown in Figure 31-14 on page 1645. The *copy area* is the location where files and directories from the ClearCase repository are stored while you work on them. This folder contains the file named `.copyarea.db`. Typically, the file is located under the user's home directory, but you can choose a separate location when you create the view.





*Figure 31-14 Browse to the web view copy area path*

In the ClearCase Navigator and ClearCase Details views, you can right-click a file or folder, and you get access to a context menu that is similar to the context menu that is shown in Figure 31-15 on page 1646.

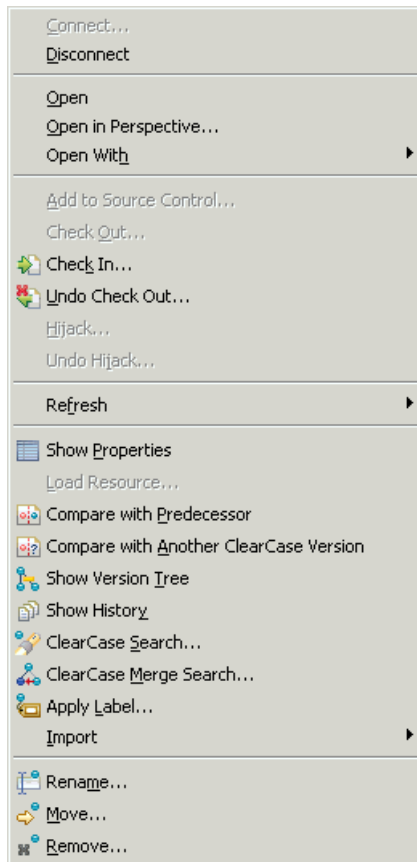


Figure 31-15 ClearCase Navigator and ClearCase Details context menu

This menu only enables those actions that are applicable based on the current ClearCase state of the selected resource. *The Import option allows you to import a project into the workspace.*

### **ClearCase Details**

The details about objects that are selected in the ClearCase Navigator view appear in the ClearCase Details view.

### **ClearCase Pending Changes view**

The Pending Changes view enables you to preview and accept all changes to your view, to resolve conflicts, and to synchronize changes in both the Base ClearCase and UCM environments. The first time that you open the view, you see the contents that are shown in Figure 31-16 on page 1647. You are prompted to configure the view with a dialog box, as shown in Figure 31-17.

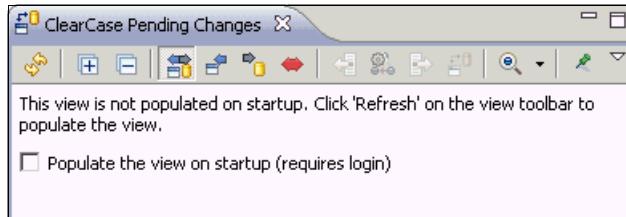


Figure 31-16 ClearCase Pending Changes view

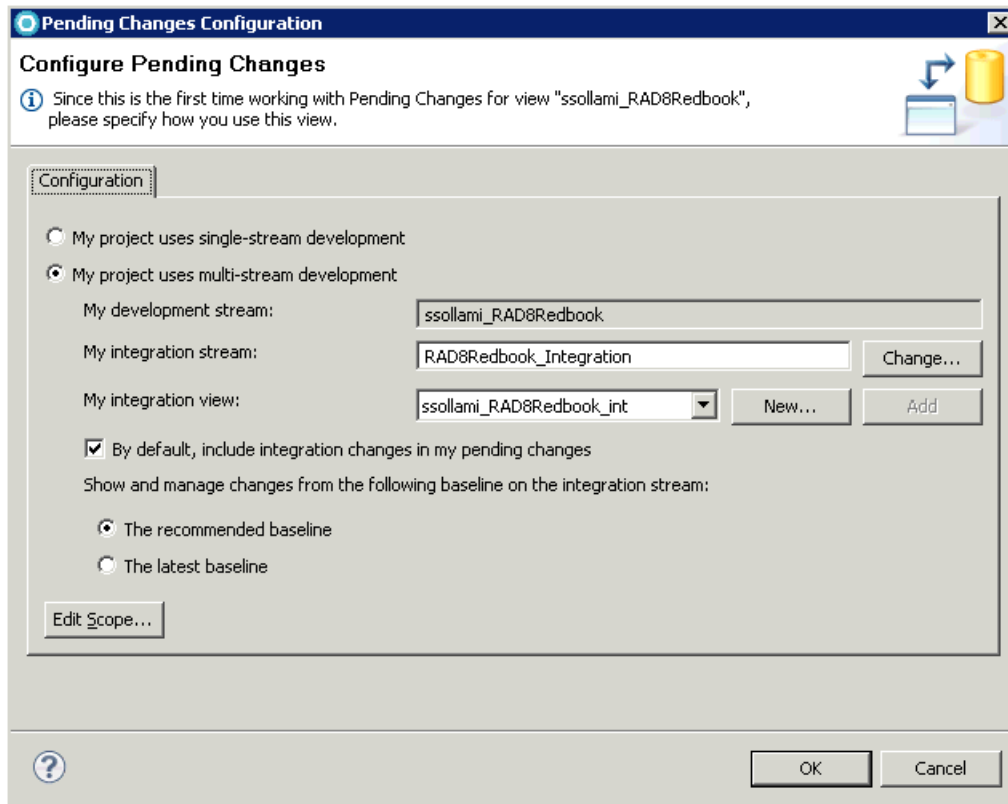


Figure 31-17 Configuring the pending changes view for a UCM multi-stream project

The configuration differs based on whether you use Base ClearCase or UCM and whether you use a single stream or multiple stream development project.

After you complete the configuration, the ClearCase Pending Changes view shows any activities containing changes for the view.

In Figure 31-18, you can see an example of the Pending Changes view with one activity (AddCustomer) that has multiple outgoing changes. You can use options on the Toolbar view to select **Integrate all outgoing changes** or **Synchronize all changes**.

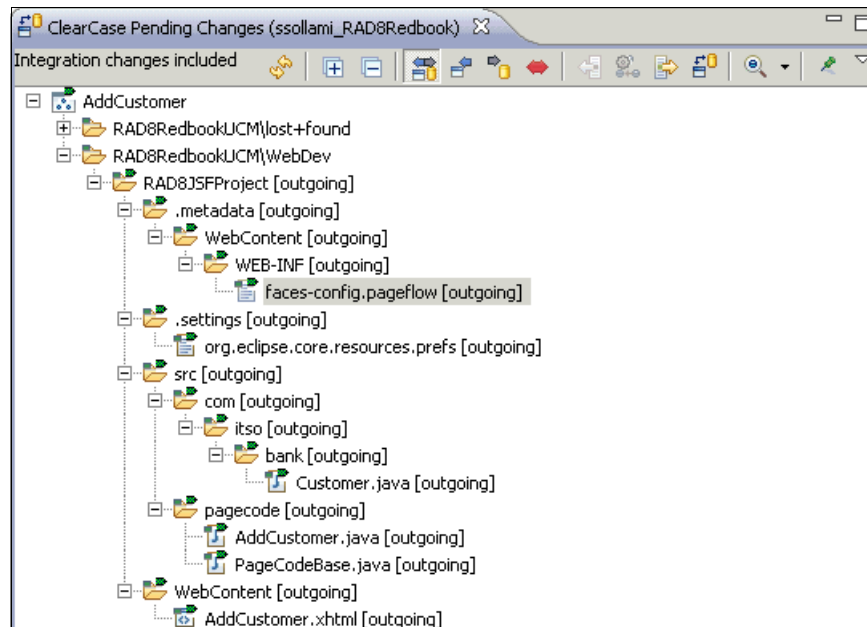


Figure 31-18 Example of activity with pending outgoing changes

### 31.4.5 ClearCase Remote Client decorators

ClearCase Remote Client adds decorators to the icons of files and folders, as seen in the Rational Application Developer Explorer views (Enterprise Explorer, Project Explorer, Package Explorer, Navigator, and ClearCase Navigator). These decorators allow you to know the current ClearCase status of the resource (checked in, checked out, view-private, hijacked, and so on). You can obtain the complete list of decorators at this link:

[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.ccrs.help.doc/topics/r\\_decor.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.ccrs.help.doc/topics/r_decor.htm)

Label decorations can influence the performance of Rational Application Developer. You can enable or disable them by selecting **Window** → **Preferences** → **General** → **Appearance** → **Label Decorations**.

## 31.5 ClearCase views and Rational Application Developer workspaces

For the integration between Rational Application Developer and ClearCase to function properly, you are required to create a separate workspace for each ClearCase view that you use.

Because the contents of a ClearCase view are determined by its configuration specification and load rules, only one version of a resource is valid in any ClearCase view at a time. A ClearCase view selects a consistent set of versions. If you import projects from multiple ClearCase views into a single workspace, the workspace might reference an inconsistent, incompatible combination of resources. Note, however, that this product does not enforce this constraint, it is up to the users to maintain the logical relationship between workspaces and views.

In particular, when using ClearCase UCM, a developer might want to work in both the Development view and the Integration view. It is fundamental that the developer has two separate workspaces that are associated with these two views.

You can create a workspace to associate with a ClearCase view in one of three ways:

- ▶ You can create the ClearCase view. Then click **File** → **Switch Workspace** → **Other** to create a new workspace and switch to it.
- ▶ You can create the workspace first, by starting Rational Application Developer with the additional command-line argument:

```
-data workspace-path
```

In this command, *workspace-path* is the path to the workspace that you want to associate with a ClearCase view that you plan to create.

- ▶ You can verify that the following option is selected:
  - a. Click **General** → **Startup and Shutdown** → **Workspaces**.
  - b. Click **Prompt for workspace at startup**.
  - c. At the next start-up, Rational Application Developer prompts you to select a workspace directory, and you can provide the name of a new, empty directory.

Regardless of the way in which you choose to create the Eclipse workspace, consider creating the workspace in a folder whose name is based on the ClearCase view tag of the view with which the workspace is associated.

If you create new projects in this workspace, perform the following steps:

1. Click **Team** → **Share project**.
2. Select the desired ClearCase plug-in.
3. Select a path inside the view that you associate with this workspace.

This action *moves* the project into the ClearCase view. Repeat this step for each new project that you create in this workspace, moving them all inside the *same* ClearCase view.

If you already have a ClearCase view that contains a number of projects for which you want to create a new workspace, you can populate the workspace in the following way:

1. Click **File** → **Import**.
2. In the Import dialog, select **General** → **Existing Projects into Workspace**. Click **Next**. Click **Select root directory** and browse to the location of the project parent directory in the chosen ClearCase view. Clear the option **Copy Projects into Workspace**.
3. Repeat the same operation for each project root directory in the same ClearCase view (this menu allows you to import multiple projects at a time). The imported projects will be referenced by this workspace, without being copied locally inside the workspace. All changes that are made to the project files will update the files that are stored in the ClearCase view.

*ClearCase Remote Client allows you to import multiple projects at one time from the ClearCase Navigator view (31.4.4, “ClearCase Explorer perspective” on page 1642 and Figure 31-23 on page 1657). To further automate this process, you can use Team Project Sets.*

## 31.6 Populating Rational Application Developer workspaces: Using Team Project Set files

*Team Project Sets* are an Eclipse feature that allows you to automate the creation of new workspaces based on the projects that are contained in a ClearCase VOB. To take advantage of this feature, you must export a Team Project Set file from a workspace where the projects have been shared. You can then share the Team Project File itself in ClearCase for later reuse.

When you want to populate a new workspace, you simply import this Team Project Set file. Importing a Team Project Set file achieves two objectives:

- ▶ It loads the projects from the VOB into the ClearCase view that was selected when importing the Team Project Set file.

- ▶ It loads the projects into the current Rational Application Developer workspace.

Follow these steps to import an Eclipse Team Project Set from Rational ClearCase:

1. Complete these steps if using ClearCase Remote Client:
  - a. In the ClearCase Navigator or ClearCase Details view, select a project set file (.psf) to import.
  - b. From the ClearCase context menu, click **Import** → **Import Project Set into Workspace**.
  - c. Enter or select the project set file (.psf) and click **Finish**.
2. Complete these steps if using the ClearCase SCM Adapter:
  - a. From the menu bar, select **File** → **Import** → **Team Project Set**.
  - b. Enter or select the project set file (.psf) and click **Finish**.
3. If the project set is not in a ClearCase VOB, the ClearCase view dialog box appears.
4. The ClearCase views that are currently running on your system appear in the ClearCase view dialog. Select the view that you want to use to import the project set. *If the ClearCase SCM Adapter is also installed on your machine, your ClearCase SCM Adapter dynamic and snapshot views also are displayed in the ClearCase view dialog window.* If you select a ClearCase SCM Adapter dynamic or snapshot view, any future ClearCase operations that you perform on the project set in Rational Application developer are accomplished through the ClearCase SCM Adapter.
5. Click **OK** to import the project set using the ClearCase view that you selected.

## 31.7 Working in Base ClearCase with SCM Adapter and dynamic views

In this section, we describe a typical scenario using dynamic views in Base ClearCase. You can see the power of dynamic views best when multiple developers work on the same branch and update the same files. This environment leads to the automatic loading of new versions of the files in the dynamic view, triggering the automatic refresh of the resources in the Rational Application Developer workspace (see 31.3.4, “Clearcase SCM Adapter and dynamic views” on page 1636). In case developers attempt to check out the same files at the same time, this scenario also shows how to use reserved and

unreserved checkouts to make changes in parallel, leading to the need to merge the results when the second user tries to check in.

### 31.7.1 Prerequisites

The prerequisites for this scenario are minimal:

1. The ClearCase Administrator must have created one VOB, as described in “Creating a VOB for use in Base ClearCase” on page 1847. We assume the existence of a VOB called RAD8RedbookBase to remind us that this VOB is used in Base ClearCase and does not contain UCM components.
2. In “Creating a dynamic view” on page 1849, we show you how to create a dynamic view called lziosi\_base\_view for a sample developer in our scenario, Lara Ziosi, by using the View Creation Wizard from the Rational ClearCase Explorer.
3. You also can create new views from inside Rational Application Developer by performing the following steps:
  - a. Click **ClearCase** → **Connect to ClearCase**.
  - b. Click **ClearCase** → **Create View**. This menu invokes the same View Creation Wizard as already shown.
4. If the view that is created from inside Rational Application Developer is not visible in the Rational ClearCase Explorer, you can add the View Shortcut by following these steps:
  - a. Right-click the space in the Views tab in Rational ClearCase Explorer.
  - b. Select **Add View Shortcut**, as shown in Figure 31-19 on page 1653.
  - c. For the View type, select **Dynamic**.
  - d. For the View Tag, select **lziosi\_base\_view**.
  - e. For the Page, select **General**.
  - f. For the Drive Letter, select **Y**.



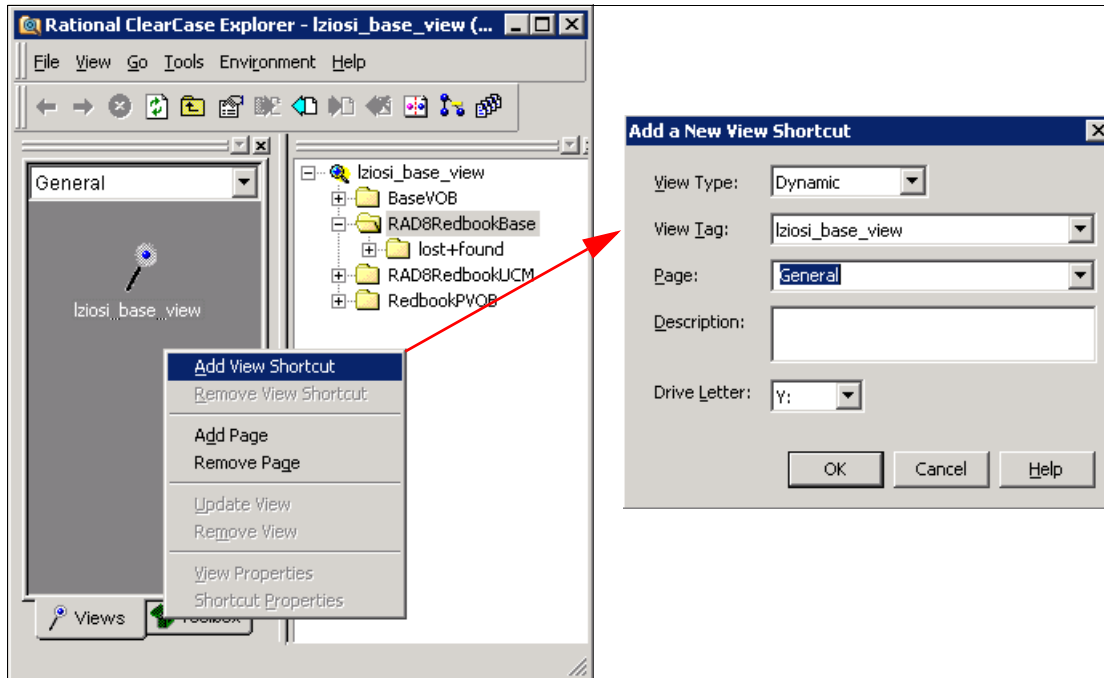


Figure 31-19 How to add a View Shortcut to Rational ClearCase Explorer

## 31.7.2 Project setup

Lara proceeds to populate her workspace and start working in Base ClearCase:

1. Start Rational Application Developer on a new workspace:  
C:\workspaces\lziosi\_base
2. Select **ClearCase** → **Connect to ClearCase**.
3. Import the existing projects into the workspace:
  - a. Click **File** → **Import**.
  - b. Click **General** → **Existing projects into Workspace**.
  - c. Select **Next**.
  - d. Select **Archive File**.
  - e. Browse for **C:\7835code\webservices\RAD8WebServiceStart.zip**.
4. Perform the steps for each project that is now present in the workspace:
  - a. Right-click the project in the Enterprise Explorer.
  - b. Select **Team** → **Share Project** (Figure 31-20 on page 1654).

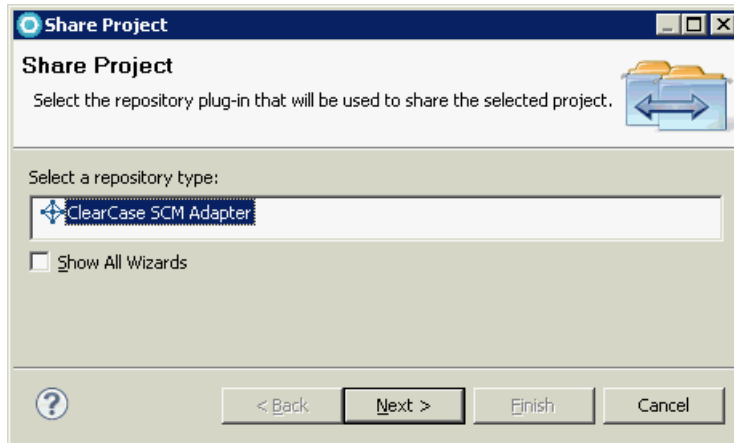


Figure 31-20 Share project dialog window

- c. For the repository type, select **ClearCase SCM Adapter** and select **Next**.
- d. Select **Browse**.
- e. You must browse to a directory inside a ClearCase VOB (Figure 31-21 on page 1655). Complete these tasks:
  - i. You must first navigate to the location of your view. Because we use dynamic views, you can locate them either from the M: drive or from Y:, which is their dedicated drive letter, in this case.
  - ii. After you have selected the view, you must browse to the VOB RAD8RedbookBase (there might be multiple VOBs visible from your view).
  - iii. Inside the VOB, it is best to create at least one subdirectory (RAD8WebServicesProjects) to be the root for all your Rational Application Developer projects. You can use the subdirectory to avoid having to check out the entire VOB root when you need to add or remove projects. Remember that directories are versioned elements in ClearCase, that a directory must be checked out to add new files and folders to it, and that the directory must be checked back in to record the changes. If you keep the VOB root checked out, no one else can add or remove files from it.
  - iv. Select either the Y:\RAD8RedbookBase\RAD8WebServicesprojects folder or the M:\lziosi\_view\_base\RAD8RedbookBase\RAD8WebServicesProjects folder.
  - v. Select **Finish**.

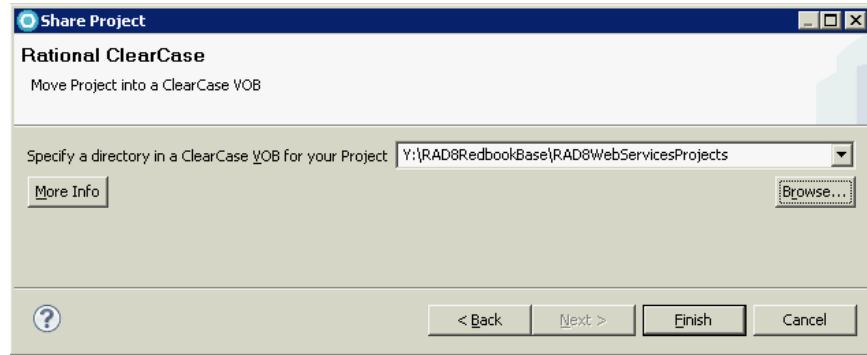


Figure 31-21 Specify a directory in a ClearCase VOB in the Share Project dialog box

- f. You see the dialog Add Elements to Source Control. Perform these steps:
  - i. Review the files to add to ClearCase. Typically, you accept all of the default proposals.
  - ii. Optional: Add a comment.
  - iii. Select **OK**.
5. After you repeat the previous steps for each project, right-click the file:
 

**RAD8WebServiceEJB → ejbModule → itso.rad8.ejb.facade → SimpleBankFacadeBean.java**
6. Select **Team → Check Out**.
7. The Enterprise Explorer view now looks similar to Figure 31-22 on page 1656. Complete these steps:
  - a. The Project folders and files appear with a light blue background indicating that they are checked in. We created this preference by selecting **Window → Preferences: Team → ClearCase SCM Adapter → ClearCase Decorations: Enable Icon Decorations**.
  - b. The Projects are decorated with the `lziosi_base_view` view tag. We created this preference by selecting **Window → Preferences: Team → ClearCase SCM Adapter → Decorate project root names with viewtags**.
  - c. The file `SimpleBankFacadeBean.java` has a light blue background with a green check mark, indicating that it is checked out. We enabled this background by using the same preference that controls the check-in decorator.

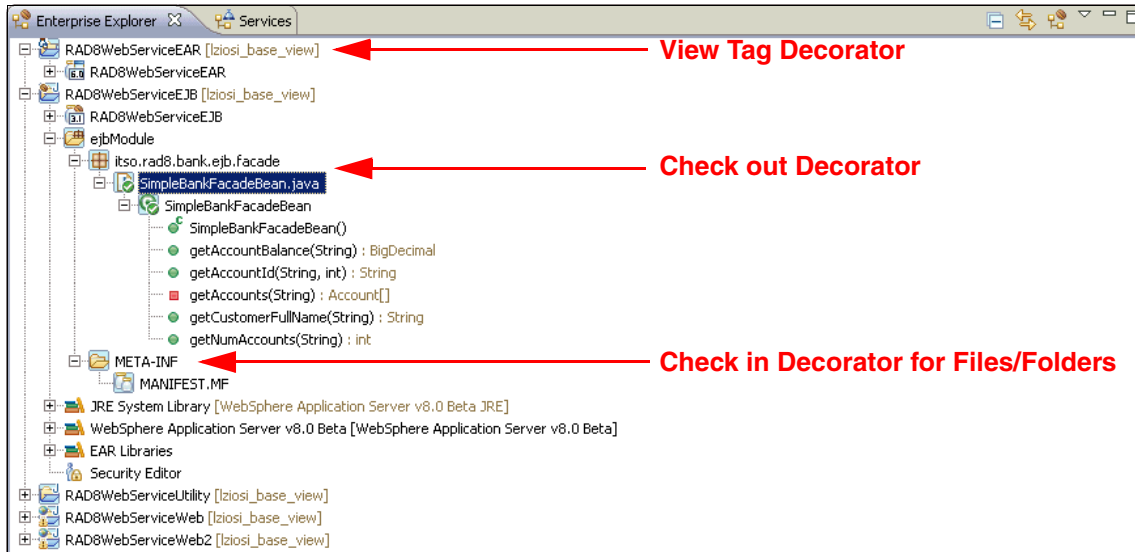


Figure 31-22 Enterprise Explorer view after adding projects to source control with the SCM Adapter

### 31.7.3 Making an unreserved checkout to work on the same file

At this point, our developer in this scenario, Salvatore, also needs to start working on the same projects. Salvatore performs the following actions:

1. Start Rational Application Developer on a new workspace:  
C:\workspaces\ssollami\_base
2. Connect to ClearCase by selecting **ClearCase** → **Connect to ClearCase**.
3. Create a new dynamic view that is called ssollami\_base\_view.
4. Import the projects from the dynamic view (Figure 31-23 on page 1657), for example, by using the drive M:

M:\ssollami\_view\_base\RAD8RedbookBase\RAD8WebServicesProjects

Do not select the option “Copy Projects into the workspace” because the projects need to remain inside the view.

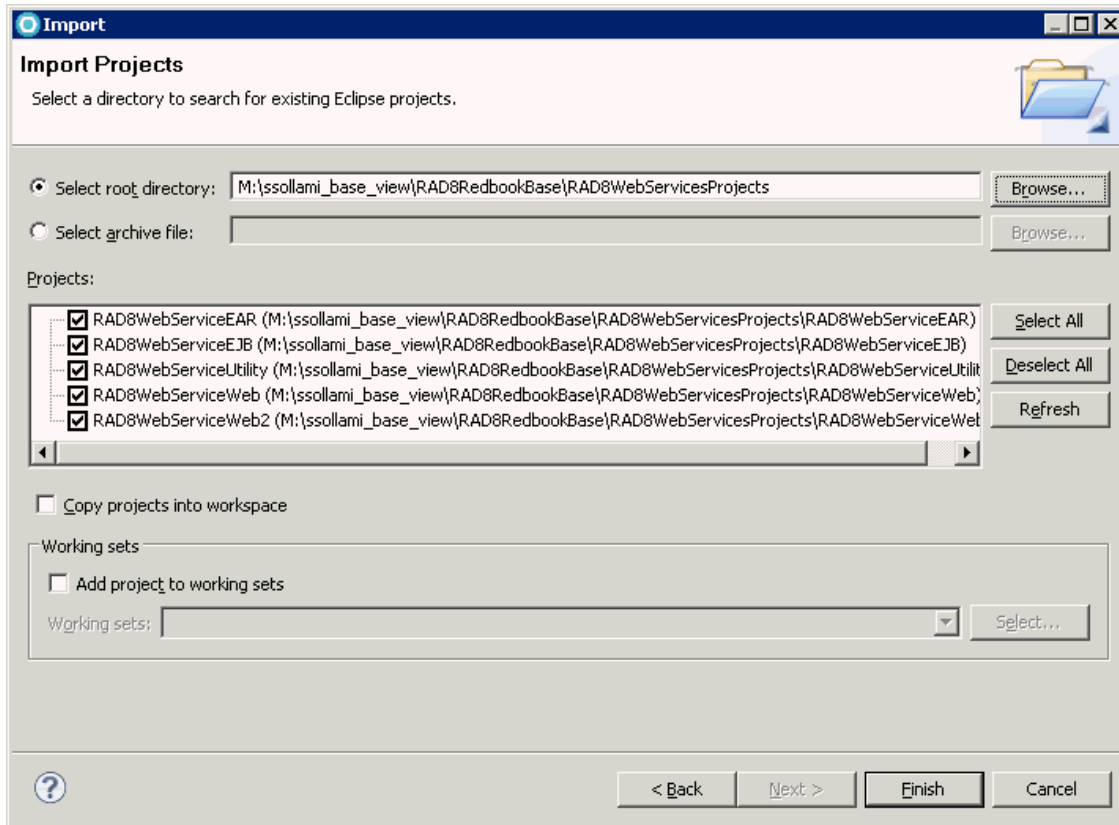


Figure 31-23 Importing projects from the root directory inside the VOB

The view decorator differs, and all of the files appear to be checked in to this view (Figure 31-24 on page 1658).

**ClearCase connection:** If the view decorator does not appear, check whether you have actually connected to ClearCase (**ClearCase** → **Connect to ClearCase**) and make sure that you have selected the preference: **Window** → **Preferences: Team** → **ClearCase SCM Adapter** → **Decorate project root names with viewtags**.

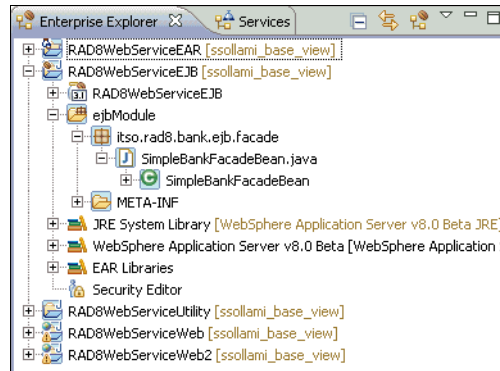


Figure 31-24 Projects after importing them into the new workspace from the new view

Now Salvatore wants to start working on the same file that Lara currently has checked out. Salvatore performs these steps:

1. Right-click **RAD8WebServiceEJB** → **ejbModule** → **itso.rad8.bank.ejb.facade** → **SimpleBankFacadeBean.java**.
2. Select **Team** → **Check Out**.
3. The Confirm Version to Check Out dialog window opens (Figure 31-25 on page 1659) and indicates that the selected version is `\main\1`. However, the branch `\main` version is reserved by the `lziosi_base_view` view. Therefore, the only option is to check out the version `\main\1 unreserved`.
4. Select **Yes**.
5. Examine the results in the Enterprise Explorer. *The decorator for an unreserved checkout is the same as the decorator for a reserved checkout.*

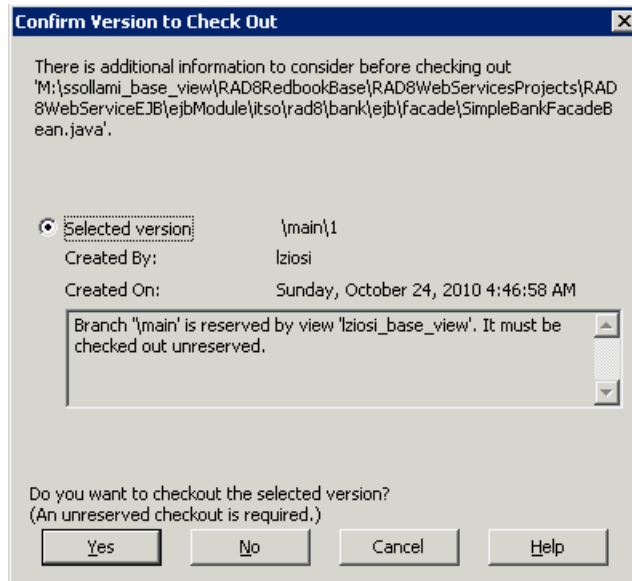


Figure 31-25 Confirm Version to Check Out (for unreserved checkout)

- Right-click the **SimpleBankFacadeBean.java** file in the Enterprise Explorer view and select **Team** → **Show Version Tree**. This action launches an external executable that shows the hierarchy of versions for this element (Figure 31-26). You can see that the element is located on the \main branch and that the current view has an UNRESERVED checkout of version 1 (red icon). The lziosi\_base\_view view has a RESERVED checkout.

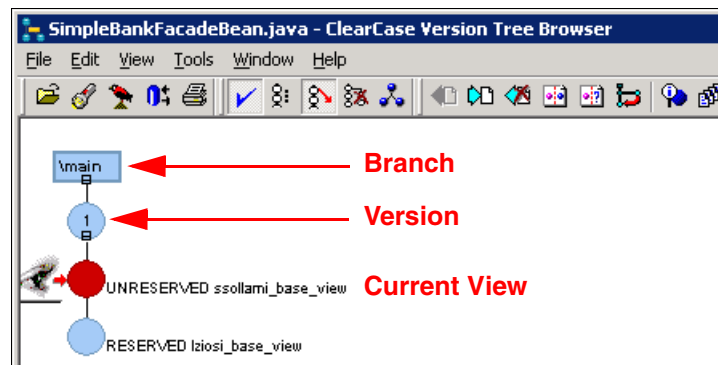


Figure 31-26 ClearCase Version Tree Browser

## 31.7.4 Merging changes

Lara now wants to transform the Enterprise JavaBeans (EJB) into a Java API for XML Web Services (JAX-WS) web service and edits the checked out file `SimpleBankFacadeBean.java`:

1. Add the annotation `@WebService` on the line over the class name. Perform these steps:
  - a. Press `Ctrl+Shift+O` to add the import:

```
import javax.jws.WebService;
```
  - b. Save the file.
  - c. Notice the QuickFix icon on the line with `@WebService` annotation, with the text:

```
This EJB Web Service does not have a router module. Invocation of this web service would fail.
```
2. Right-click the project and select **WebServices** → **Create Router Module (EndPointEnabler)**. Complete these tasks:
  - a. For EJB WebService Binding, select **HTTP**.
  - b. For Project Name, select **RAD8WebServiceEJB\_HTTPRouter**.
  - c. Select **Finish**.

Figure 31-27 shows that the file `org.eclipse.wst.common.component` in the EAR project needs to be checked out (to add the new web module to the EAR).

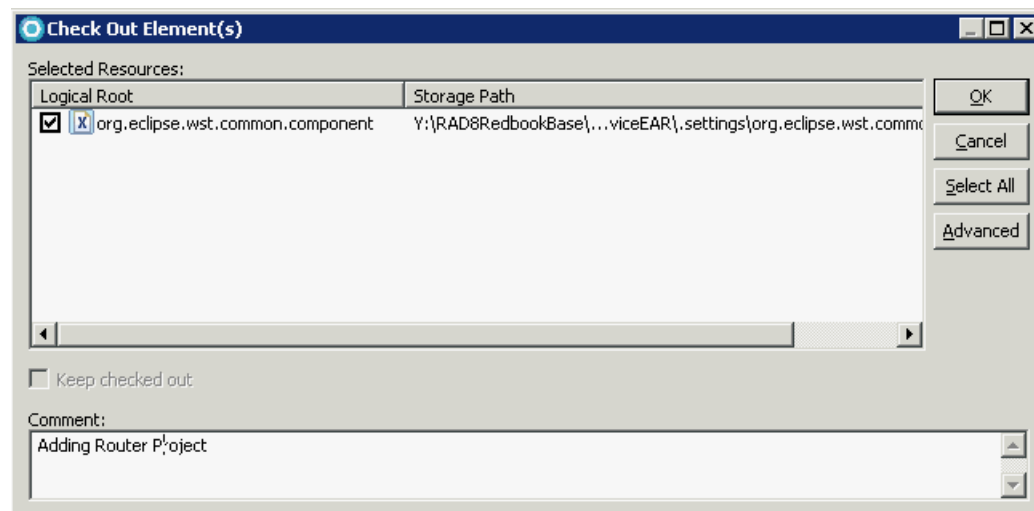


Figure 31-27 Prompt to check out the `org.eclipse.wst.common.component` file



3. Figure 31-28 shows that the .project file in the EAR project needs to be checked out (to add a reference to the new web module to the EAR).
4. At this point, check in all checked out files by right-clicking each file and selecting **Team** → **Check In**. Perform these steps:
  - a. Certain files are hidden in the Enterprise Explorer. You can see them by selecting **Window** → **Show View** → **Other: General** → **Navigator**.

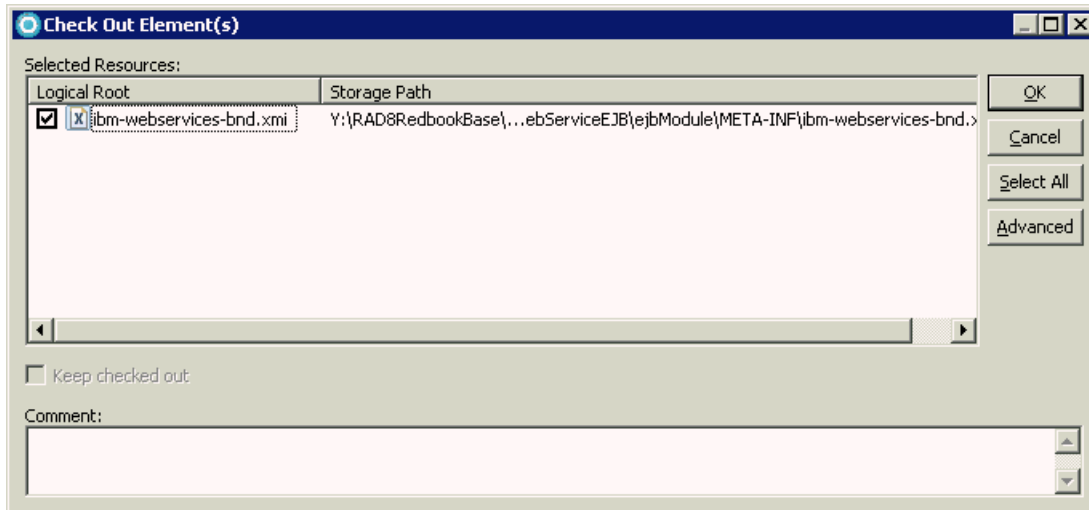


Figure 31-28 Prompt to check out the .project file inside the EAR

- b. Alternatively, you can use **ClearCase** → **Find Checkouts** and select the root directory (RAD8WebServicesProjects) where all of the projects are contained, as shown in Figure 31-29 on page 1662.
  - c. Select **OK**.

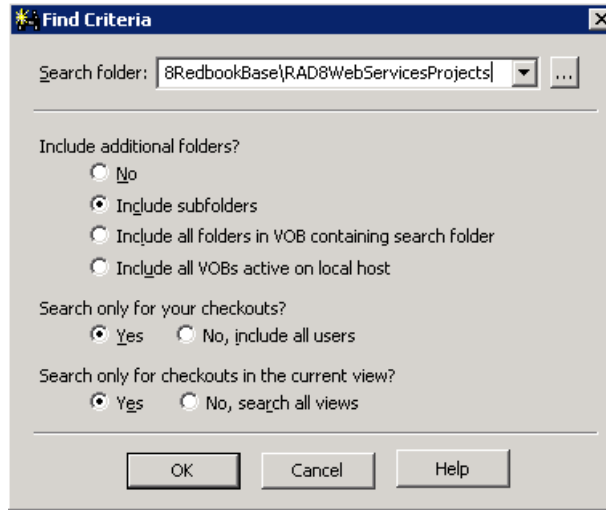


Figure 31-29 Find Criteria dialog box

- d. The Find Checkouts dialog window shows the three files that have been checked out (Figure 31-30).
- e. Select **Edit** → **Select All**.
- f. Select **Tools** → **Check In**.
- g. Verify that the status of the checked out files has changed to checked in the Navigator view. If the status did not change, select **Team** → **Refresh Status**.

Name	User	View	Checked Out	Predecessor
Y:\RAD8RedbookBase\RAD8WebServicesProjects\RAD8WebServiceEAR\project	lziosi	lziosi_base_view	Reserved	{main}1
Y:\RAD8RedbookBase\RAD8WebServicesProjects\RAD8WebServiceEAR\settings\org.eclipse.wst.common.component	lziosi	lziosi_base_view	Reserved	{main}1
Y:\RAD8RedbookBase\RAD8WebServicesProjects\RAD8WebServiceEJB\ejbModule\META-INF\ibm-webservices-bnd.xml	lziosi	lziosi_base_view	Reserved	{main}2

Figure 31-30 The Find Checkouts dialog window allows you to check in multiple files

- h. To finish, right-click the newly created project **RAD8WebServiceEJB\_HTTPRouter**.
- i. Select **Team** → **Share project** and complete the wizard.

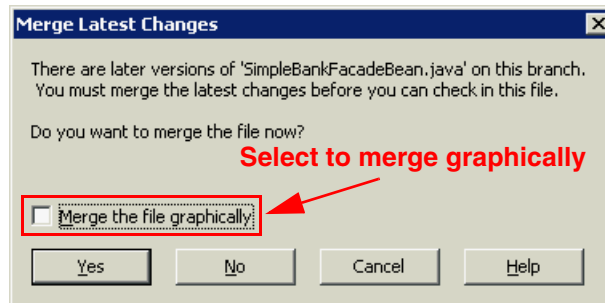
Salvatore, in the meantime, wants to add a new method to the EJB:

1. Add the code that is shown in Example 31-1 on page 1663 into the `SimpleBankFacadeBean.java` class.

*Example 31-1 New method for SimpleBankFacadeBean.java class*

```
public BigDecimal deposit(String accountId, BigDecimal amount)
 throws AccountDoesNotExistException {
 AccountFactory factory = new AccountFactory();
 Account account = factory.findById(accountId);
 account.setBalance(account.getBalance().add(amount));
 BigDecimal result = account.getBalance();
 return result;
}
```

2. Save the file.
3. Select **Team** → **Check in**. This selection results in the Merge Latest Changes dialog box (Figure 31-31), which indicates that there are later versions of SimpleBankFacadeBean.java on this branch. Complete these steps:
  - a. Select **Merge the file graphically**. This option is not required in this case because there are no conflicts and the merge can complete automatically. However, this option is useful to inspect the incoming changes.
  - b. Select **Yes**.



*Figure 31-31 Merge Latest Changes dialog window*

4. You see a Diff Merge Window that is similar to Figure 31-32 on page 1664.

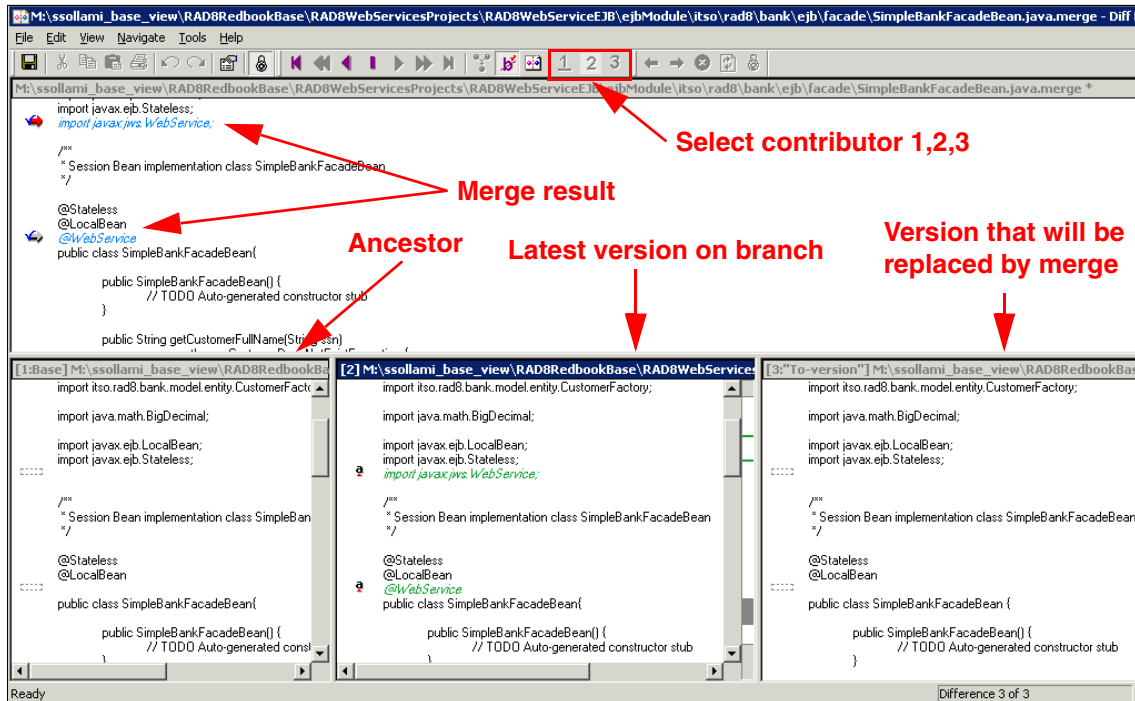


Figure 31-32 Diff Merge Window

5. The top pane in Figure 31-33 on page 1665 shows the merge result.
  - a. The three lower panes show this information (left to right):
 

<b>[1:Base]</b>	The common ancestor
<b>[2]</b>	The later version that was found on the main branch, which was checked in by another view
<b>[3:“To-version”]</b>	The version that the merge result will replace
  - b. Review the changes. If you do not want one of the automated choices, you can choose another contributor by choosing one of Select contributor buttons across the top toolbar (1, 2, or 3). In this case, all the default choices are acceptable.
  - c. Select **Save**.
  - d. Select **Exit**.
  - e. The following message appears: The Merge Completed successfully. Do you want to check the file in now? Select **OK**.
6. Select the file again and select **Team** → **Version Tree**. Follow these steps:
  - a. In the version Tree, select **Tools** → **Options**.

- b. In the Meta Data Filter tab, select **Creation info**. This option results in a display that is similar to Figure 31-33. Note the red arrow that indicates the merge. You can rearrange the icons for better readability.

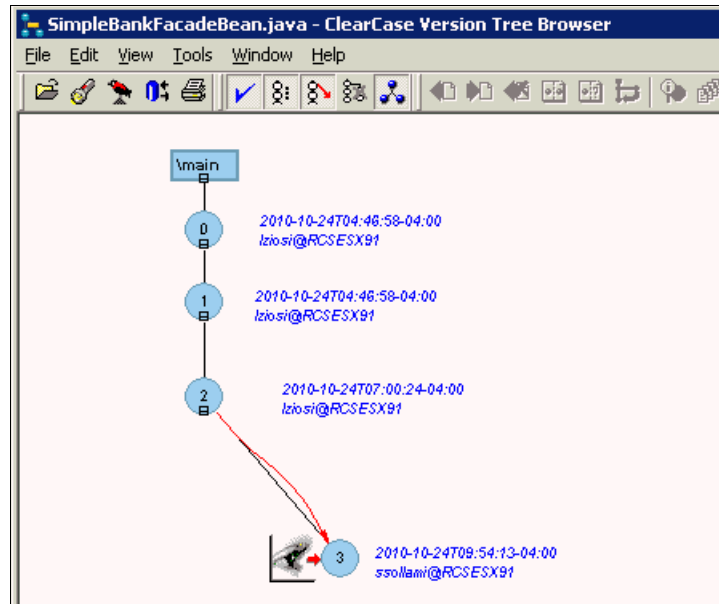


Figure 31-33 Version Tree with merge arrow

The merge completed as expected; however, a validation error appears in the Problems view:

CHKW6045E: The routerModule name RAD8WebServiceEJB\_HTTPRouter.war in ibm-webservices-bnd.xmi references a module that does not exist in the application RAD8WebServiceEAR.

This problem relates to the /RAD8WebServiceEJB/ejbModule/META-INF/ibm-webservices-bnd.xmi file. The dynamic view has automatically brought this file into the EJB project, but the dynamic view cannot bring a new project into the workspace.

To resolve this problem, Salvatore performs these steps:

1. Inspects the dynamic view contents from the Rational ClearCase Explorer.
2. Finds the new project that Lara has added.
3. Imports the new project into the workspace.
4. Selects **Project** → **Clean** → **Build**.
5. No more validation errors are present in the Problems view.

Lara will simply find the new method in her file, without any prompting or error. Lara's dynamic view was notified automatically of the new version of the file, and her workspace was configured to refresh automatically.

We have seen how working with dynamic views simplifies the task of receiving incoming changes, because no user action is required to accept new files or new versions of the files when working on a shared branch. However, we have also seen how the changes that are produced by other users can lead to validation errors and the actions that need to be taken to resolve these errors.

Users, who want to have more control over when they accept changes from the rest of the team working on the same branch, might prefer to use snapshot views.

Additionally, administrators can set up multiple branches to allow for parallel work and establish policies for when to merge these branches back into the main *integration* branch.

A method to streamline the previous approach and isolate the work of each developer consists of setting up a UCM project with traditional parallel strategy.

## 31.8 Working in ClearCase UCM with ClearCase Remote Client

In this section, we describe a typical workflow scenario using Unified Change Management (UCM). Describing UCM is beyond the scope of this book, but we introduce the terminology that a developer must know to be able to work in a UCM project. For more information on UCM, see this website:

[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/topic/com.ibm.rational.clearcase.cc\\_proj.doc/c\\_u\\_ovw\\_ch.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/topic/com.ibm.rational.clearcase.cc_proj.doc/c_u_ovw_ch.htm)

Figure 31-34 on page 1667 displays the logical representation of this workflow.

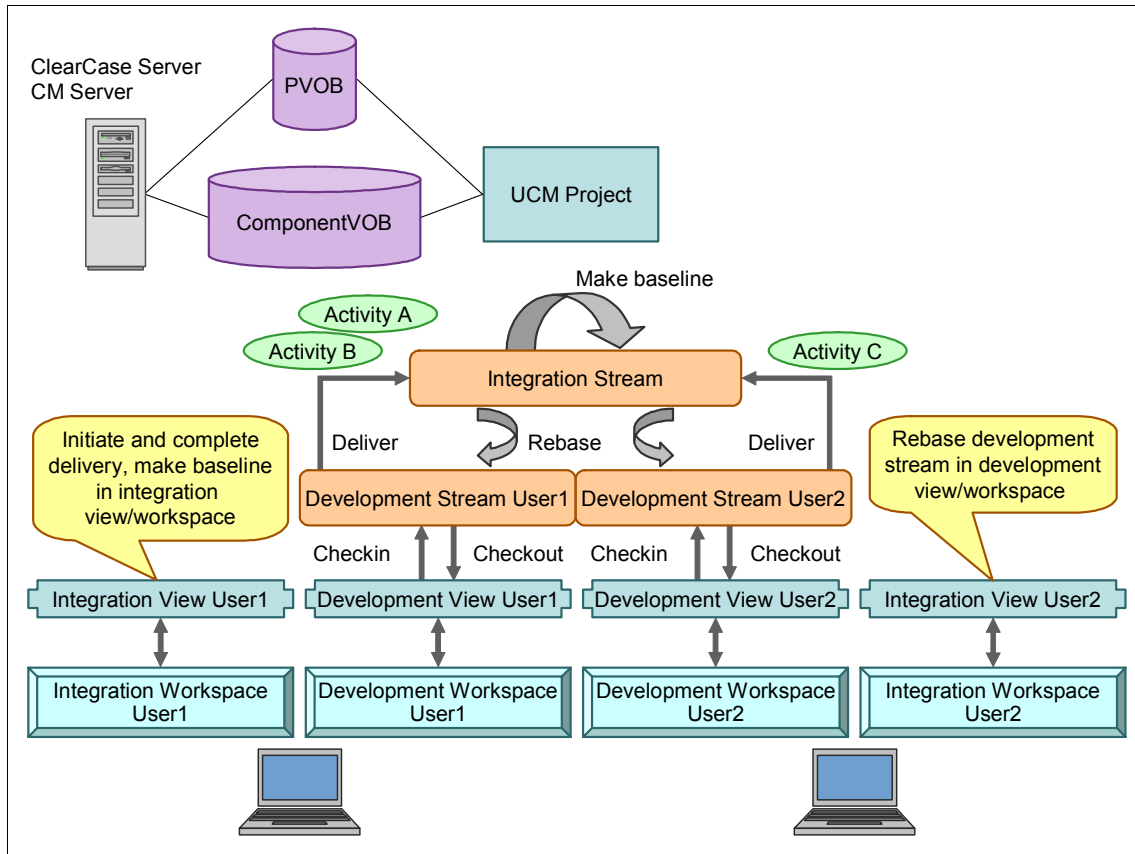


Figure 31-34 UCM scenario

The following descriptions explain the UCM terms that are shown in Figure 31-34:

- Project VOB** A versioned object base (VOB) that stores UCM objects, such as projects, streams, activities, and change sets.
- Component VOB** A VOB that stores one or more components. If there is only one component, the component matches the VOB root directory. If there are multiple components, each component matches a directory inside the VOB.
- Component** A ClearCase object that is used to group a set of related directory and file elements within a UCM project. Typically, the elements that make up a component are developed, integrated, and released together. A project must contain at least one component, and it can contain multiple components. Projects can share components.

<b>Project</b>	A <i>project</i> is the object that contains the configuration necessary information to manage a significant development effort, such as a product release. A project contains one main shared work area and typically multiple private work areas. Private work areas allow developers to work on activities in isolation.
<b>Stream</b>	A <i>stream</i> is an object that maintains a list of activities and baselines and determines which versions of elements appear in your view.
<b>Integration stream</b>	A project contains one integration stream, which records the project baselines and enables access to shared versions of the project elements. The integration stream and a corresponding integration view represent the project main shared work area.
<b>Integration view</b>	View associated with the integration stream.
<b>Development stream</b>	In a typical project, each developer has a private work area, which consists of a development stream and a corresponding development view. The <i>development stream</i> maintains a list of the developer's activities and determines which versions of elements appear in the developer's view.
<b>Development view</b>	This view is associated with the development stream.
<b>Delivery</b>	A ClearCase operation in which developers merge the work from their own development streams to the project's integration stream or to a feature-specific development stream. If required, the delivery operation invokes the Merge Manager to merge versions.
<b>Rebase</b>	<i>Rebase</i> is a ClearCase operation that makes a development work area current with the set of versions represented by a more recent baseline in another stream, usually the project's integration stream or a feature-specific development stream.
<b>Activity</b>	An object that tracks the work that is required to complete a development task. An activity includes a text headline, which describes the task, and a change set, which identifies all versions that developers create or modify while working on the activity.
<b>Baseline</b>	An object that represents a stable configuration for one or more components. A baseline identifies activities and one version of every element that is visible in one or more components.



## 31.8.1 Prerequisites

The development scenario that is described in this section requires the existence of the following artifacts, which are created by the ClearCase administrator:

- ▶ A project VOB (PVOB), which is called RedbookPVOB
- ▶ A VOB, which is called RAD8RedbookUCM, to store components
- ▶ At least one modifiable component, which is called WebDev, in this VOB
- ▶ A UCM project that is called RAD8Redbook
- ▶ An integration stream, which is configured for traditional parallel development, that is called RAD8Redbook\_integration

UCM projects are created on a ClearCase server. You cannot create a UCM project from the ClearCase Remote Client. We describe the setup of all of these artifacts in “Configuring ClearCase for UCM development” on page 1860.

## 31.8.2 Connecting to the ClearCase Change Management Server and joining a UCM project

Each developer must perform these steps to connect to the ClearCase Change Management Server (CMS) and join a UCM project:

1. Configure the Rational Application Workspace for use with the development view.
2. Connect to the ClearCase Change Management Server.
3. Join the project:
  - a. Create a development stream.
  - b. Create a development view.
  - c. Create an integration view.
  - d. Load the VOB in the development view.

We list these required steps in detail:

1. Start Rational Application Developer with a new workspace that is called C:\workspaces\lziosi\_UCM.
2. If the ClearCase Remote Client Capability has been enabled, you see the top-level menu called ClearCase in the Java EE perspective, as shown in Figure 31-35 on page 1670. Complete these steps:
  - a. Select **ClearCase** → **Connect**.

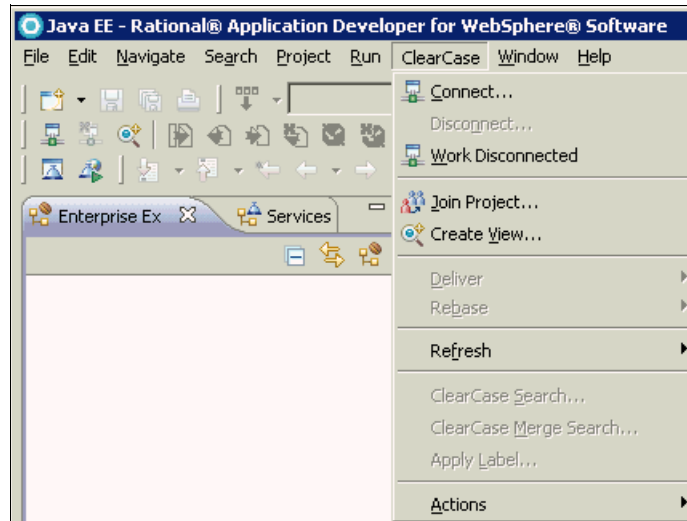


Figure 31-35 Connect to the ClearCase CM server using ClearCase Remote Client

- b. Enter the URL of the ClearCase Change Management Server, which by default is `http://hostname:12080/TeamWeb/services/Team`.
- c. Enter the User name and Password, as shown in Figure 31-36 and click **OK**.

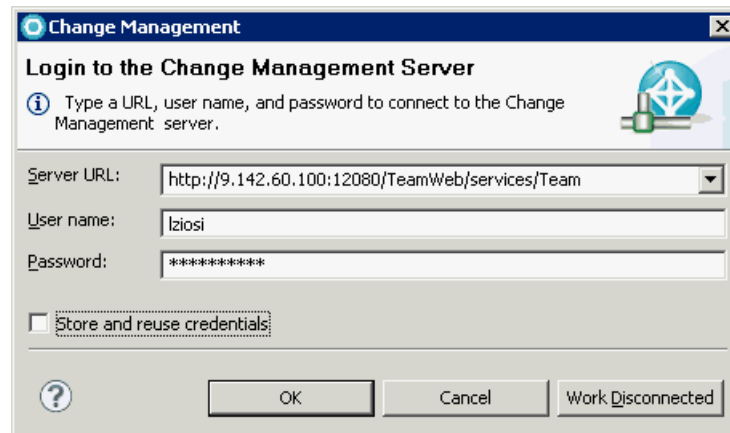


Figure 31-36 Login to the Change Management Server

3. Select **ClearCase** → **Join Project**. You see the existing UCM project, as shown in Figure 31-37. Click **Next**.

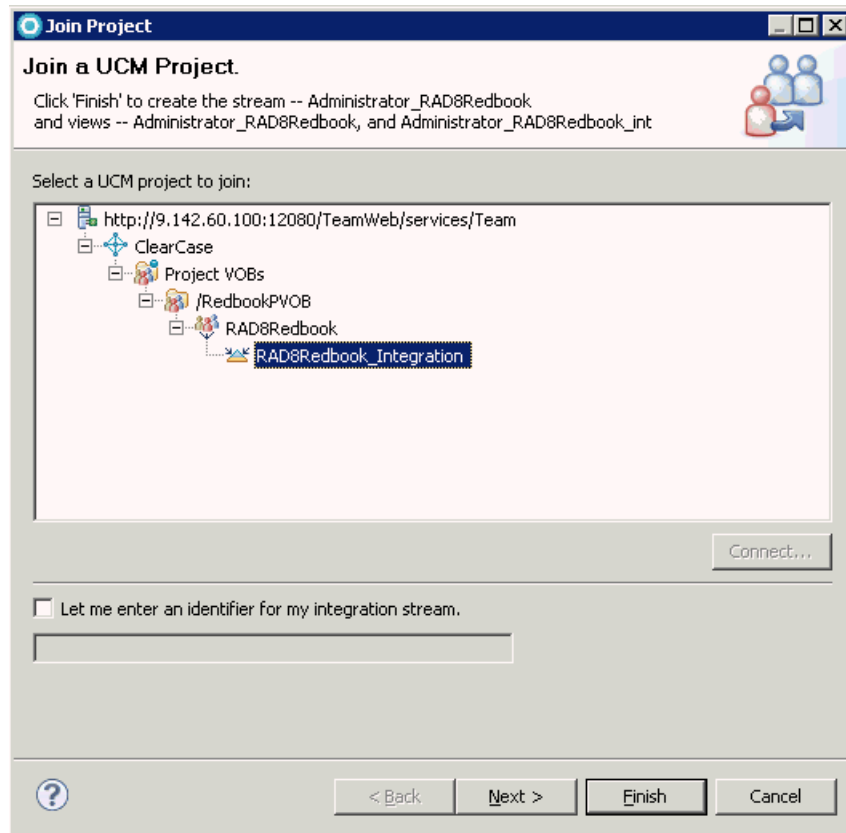


Figure 31-37 Available UCM projects that you can join

4. On the “Set up a ClearCase UCM development environment” page, as shown in Figure 31-38 on page 1672. Complete these steps:
  - a. Click **Setup a Development Stream**.
  - b. For Stream name, type `lziosi_RAD8Redbook`.
  - c. Click **Create a Development View**.
  - d. For the view tag, type `lziosi_RAD8Redbook`.
  - e. Enter a Copy area path name: `C:\Documents and Settings\lziosi\lziosi_RAD8Redbook` (this path represents the local storage for the web view).
  - f. Click **Next**.

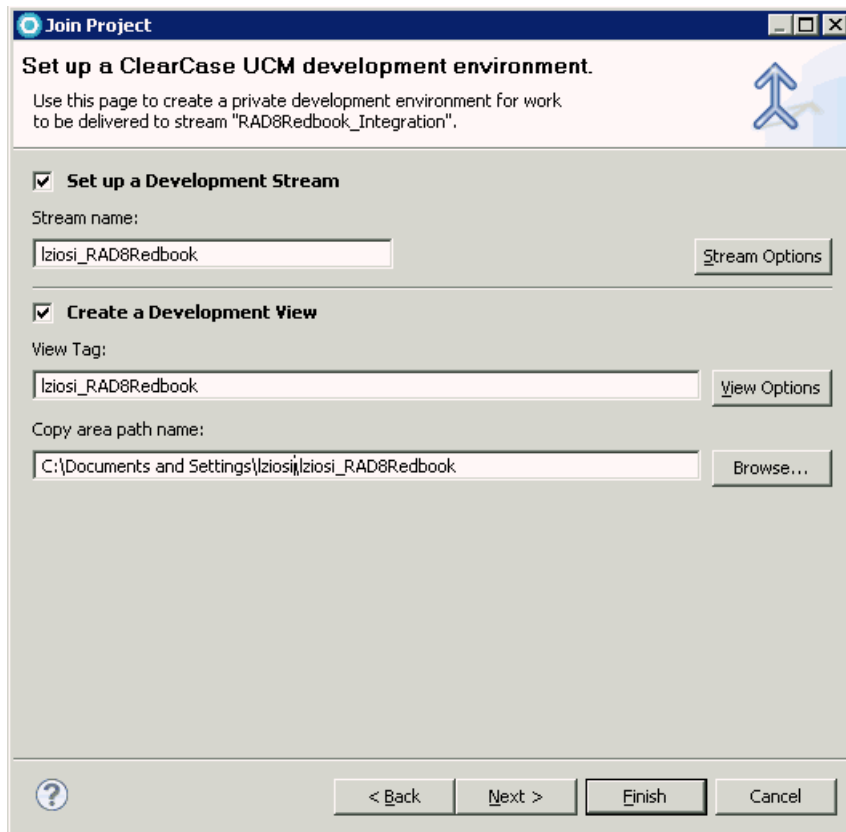


Figure 31-38 Setting up a development stream and development view

5. On the “Create a ClearCase UCM Integration View” page, as shown in Figure 31-39 on page 1673, complete these steps:
  - a. Click **Create an Integration View**.
  - b. For View tag, type lziosi\_RAD8Redbook\_int.
  - c. For Copy area path name, type C:\Documents and Settings\lziosi\lziosi\_RAD8Redbook\_int (this path represents the local storage for the web view).
  - d. Optional: Select **Configure the integration view with the development view, and load both views after configuration**. If this option is selected when joining a UCM project, any operations performed in the Copy Rules tab of the Edit configuration dialog window apply to both the development and integration views. The UCM Custom Rules tab of the Edit configuration dialog window contains two text boxes for entering rules: an upper text box for the development view and a lower text box for the

integration view. We recommend that you select this option. If you do not, you must manually configure the load rules for the integration view at a later time.

- e. Click **Finish**.

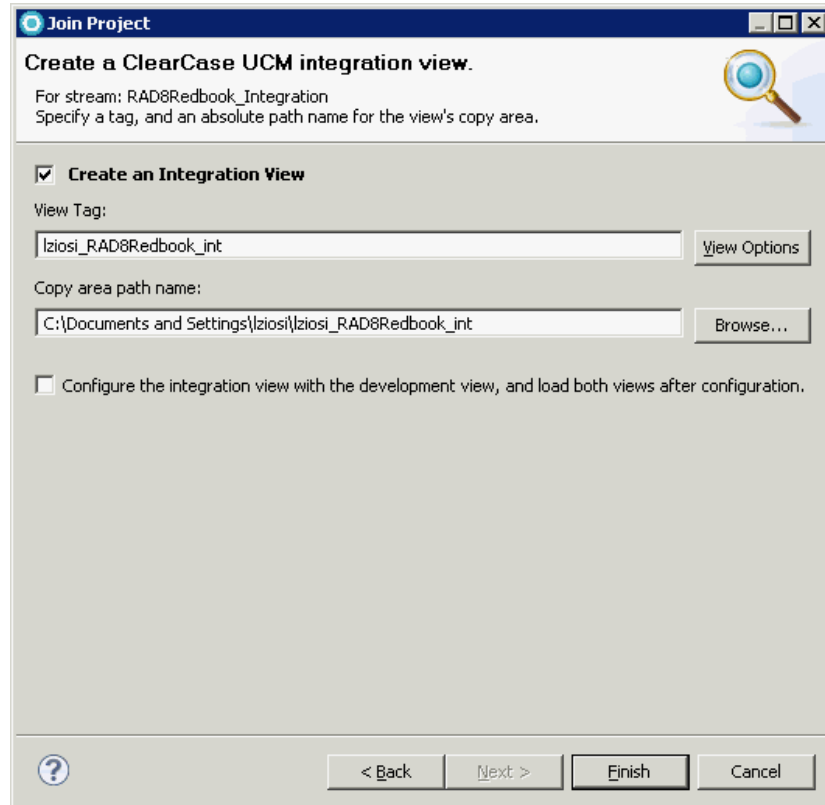


Figure 31-39 Create a ClearCase UCM integration view

6. In the Load Rules tab, select the VOB for the development view to load: **RAD8RedbookUCM**. In this example, it is a private VOB, but typically you work with public VOBs. You can also limit the view to load a component inside the VOB, such as WebDev. See Figure 31-40 on page 1674. Select **OK**.

At this point, you are ready to start working in the development view.

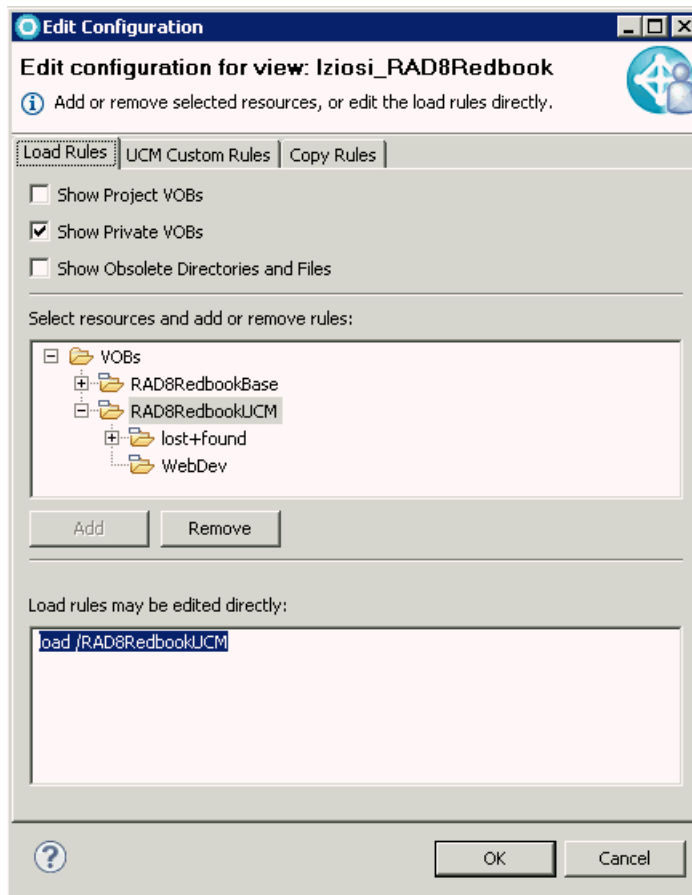


Figure 31-40 Edit configuration for development view: Load Rules tab

### 31.8.3 Initiating work in the development view or stream

Before you start working in the development view or stream, inspect the contents of the ClearCase Explorer perspective (**Window** → **Open Perspective** → **Other** → **ClearCase Explorer**), as shown in Figure 31-41 on page 1675.

In the ClearCase Navigator, you see a node for the development view and a node for the ClearCase Change Management Server, showing the structure of the UCM project under the PVOB.

For any item that you select in the ClearCase Navigator, the ClearCase Details view shows all the items contained within that item. You can right-click an item in the ClearCase Navigator or in the ClearCase Details view and select **Show**



Perform these steps to create and set this activity as the default:

1. Open the ClearCase Explorer perspective.
2. Right-click the **Development view**.
3. Click **New Activity**.
4. Enter the name CreateEJBProject, as shown in Figure 31-42.
5. Click **OK**.

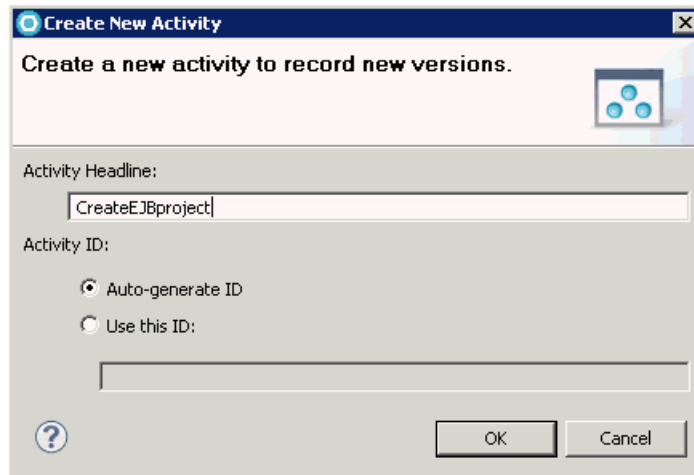


Figure 31-42 Create new Activity

For each project, perform the following steps to add the projects to ClearCase:

1. Right-click the project in the Enterprise Explorer.
2. Select **Team** → **Share project**. Follow these steps:
  - a. For the repository type, select **ClearCase Remote Client**. Click **Next**.
  - b. In the “Move selected Eclipse project into a ClearCase VOB” window, select a ClearCase view to browse the VOBs, as shown in Figure 31-43 on page 1677. Follow these steps:
    - i. For ClearCase view, select **Iziosi\_RAD8Redbook** (development view).
    - ii. Select **Next**.



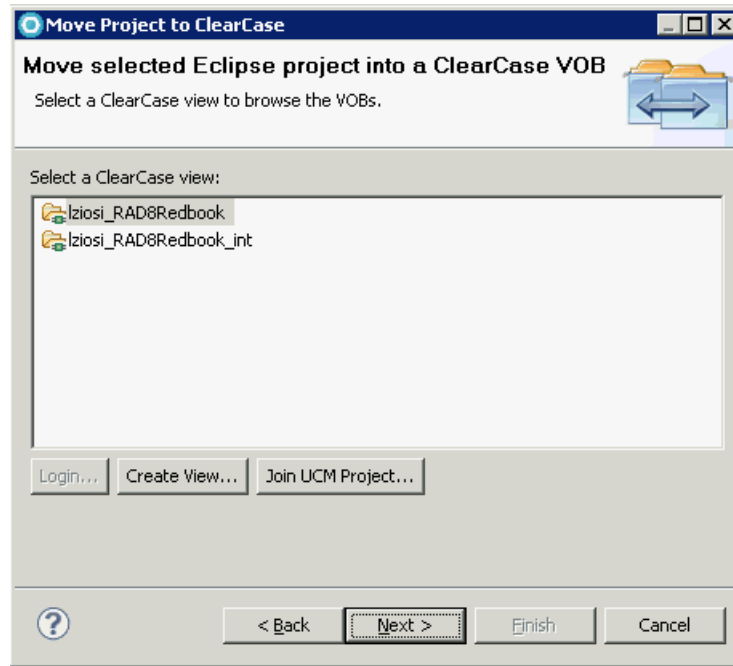


Figure 31-43 Move project into the development view

- c. In the “Move selected Eclipse project into a ClearCase VOB” window, select a folder to hold the project root of the Eclipse project, as shown in Figure 31-44 on page 1678. Complete these steps:
  - i. Click **Move project to a new project folder in the selected directory**. This selection creates a sub-directory with the project's name.
  - ii. Select RAD8RedbookUCN\WebDev.
  - iii. Select **Finish**.

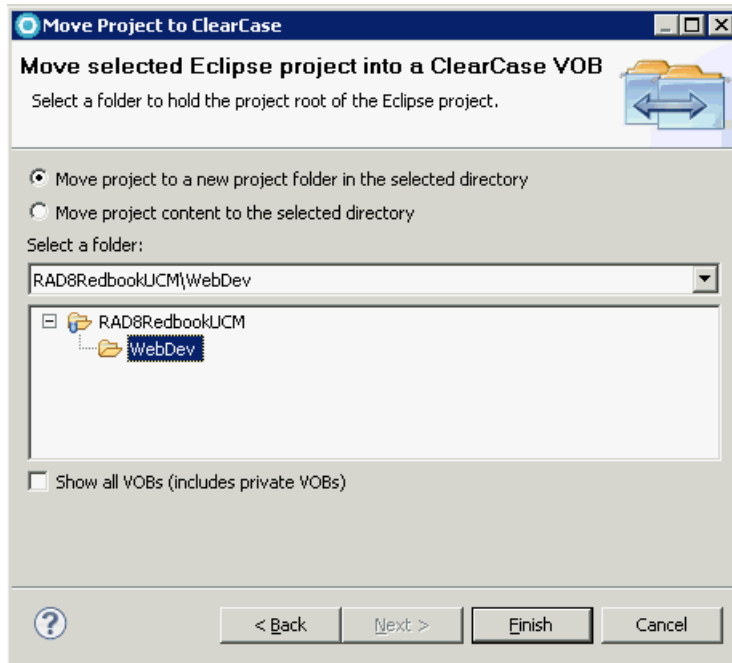


Figure 31-44 Select to move the project into the modifiable component

3. The “Add selected files to source control” window opens. Perform these steps:
  - a. Select **Show Details**, and the dialog window opens, as shown in Figure 31-45 on page 1679.

You can see that `.apt_generated` is not selected, because it is flagged as Derived. This behavior is correct. The activity that was previously added is selected, by default.

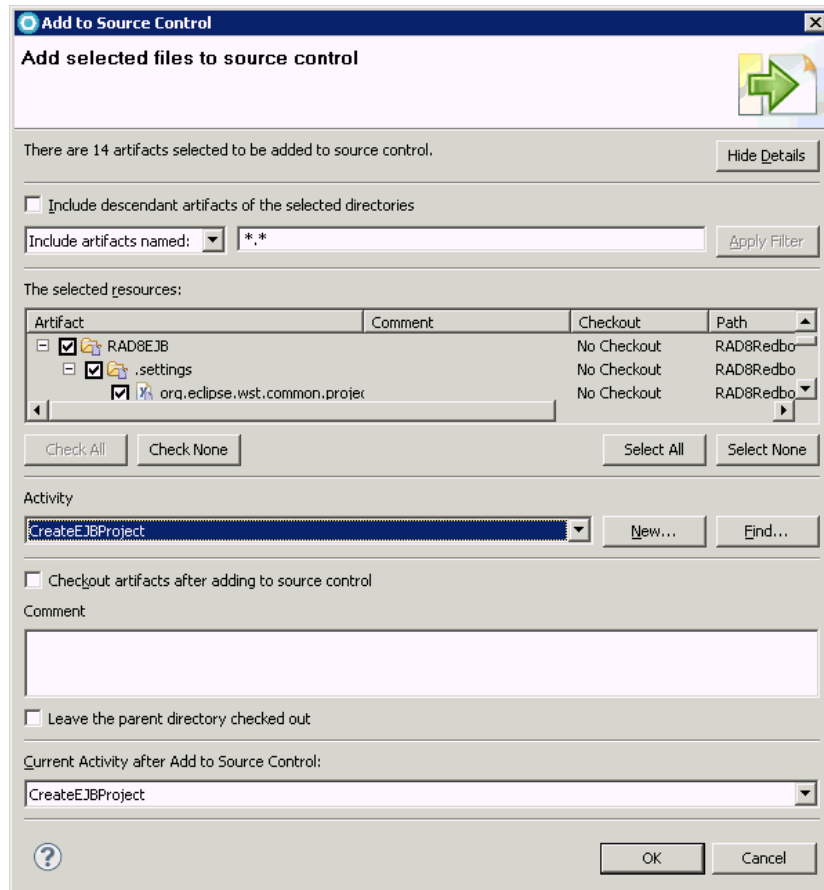


Figure 31-45 Adding the EJB project to Source Control

### 31.8.4 Delivering activities to the integration stream

Now that you have the project created and shared in your development stream, you can deliver the activity CreateEJBProject to the integration stream, so that the projects can become available for other users.

In this particular case, we know that the integration stream does not contain any files or folders yet. In general, however, a user does not know whether the integration stream contains any files or folders.

The general recommended workflow consists of these steps:

1. Update the integration view so that it has the most up-to-date information.

2. Rebase the development stream, which merges changes from the integration stream into the development stream.
3. Resolve any conflicts locally.
4. Deliver the activities.
5. Mark the delivery as complete. This action helps you to avoid files remaining checked out on the integration stream. If you do not mark the files as complete, all other users are prevented from delivering changes to the same files.

In this example, we skip the update and rebase steps, because we know that no changes have been made to the integration stream since when we joined the project.

To deliver the activity, perform the following steps:

1. Select **ClearCase** → **Deliver** → **Advanced Deliver**. Advanced Deliver gives you fine-grained control over which activities you want to deliver. You can also use Deliver if you do not need to verify the available activities.
2. You see a dialog window that lists all of the activities that can be delivered, as shown in Figure 31-46 on page 1681.
3. If you have not yet checked in all of the files that have been modified so far, you can check them in from the Checkouts/Hijacks tab of this dialog window, before you deliver.
4. The Target stream is `RAD8Redbook_Integration` by default.
5. The delivery uses the integration view `lziosi_RAD8Redbook_int`. Files to be reviewed are left checked out in this view if you do not immediately complete the delivery. We recommend that the ClearCase Remote Client users initiate the delivery from the workspace that is associated with the integration view.
6. Select **OK**.

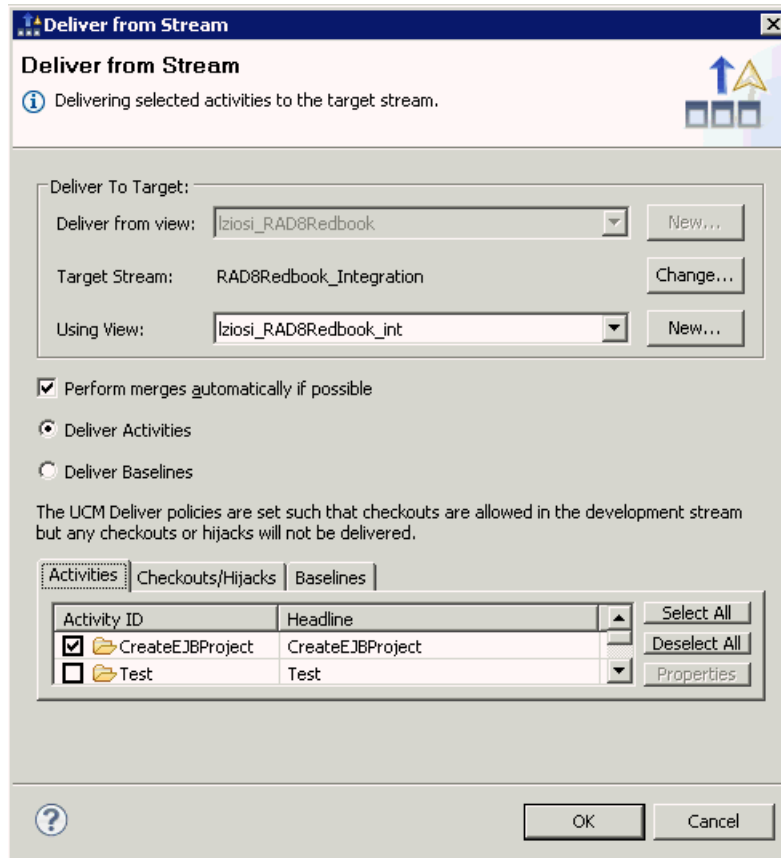


Figure 31-46 Deliver from Stream dialog window

You next see the dialog that is shown in Figure 31-47 on page 1682.

7. Select **Complete the delivery**. You might want to leave the delivery in the current state to check the results in the integration view and workspace. The disadvantage of this approach is that you leave files and folders checked out on the integration stream, which prevents other users from delivering the same files and folders.

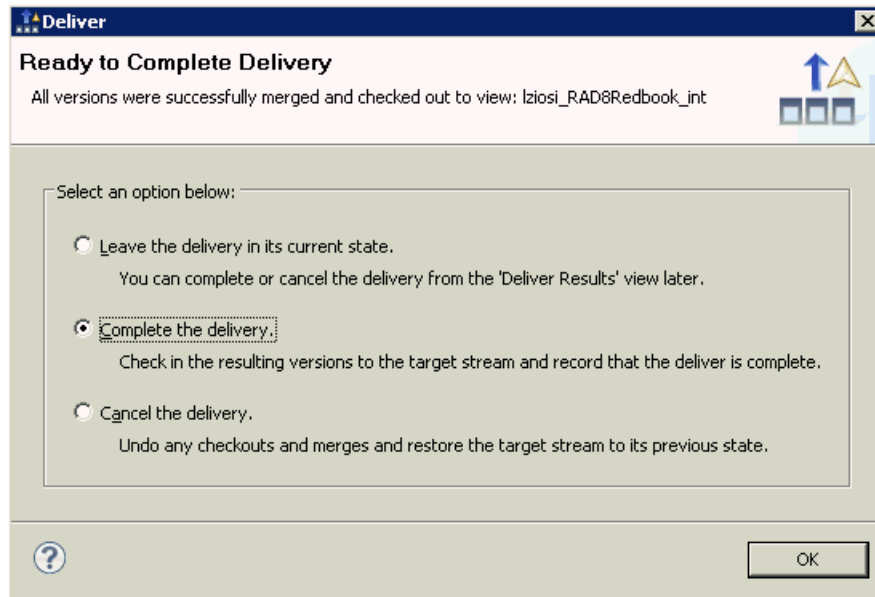


Figure 31-47 Ready to Complete Delivery dialog window

After you complete this delivery, you can select any files in the Project Explorer and select **Team** → **Show Version Tree**. You see a result similar to the window that is shown in Figure 31-48 on page 1683.

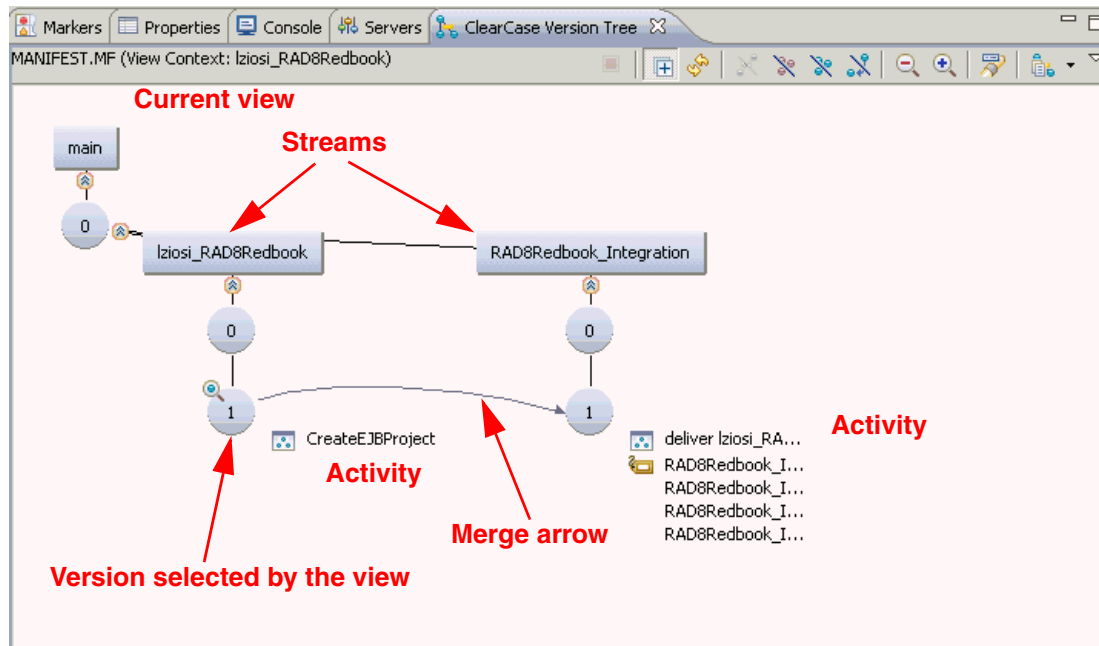


Figure 31-48 ClearCase Version Tree tab showing MANIFEST.MF

Figure 31-48 shows that the current view (lziosi\_RAD8Redbook) has selected version 1 from stream lziosi\_RAD8Redbook (you can tell by the view icon that is placed on this version), which has been merged onto the stream RAD8Redbook\_Integration (you can see the merge arrow). You can also see two activities: the activity that was delivered and the delivery activity.

### 31.8.5 Reviewing the results and creating a new baseline

Perform the following activities, which take place in the integration workspace:

1. Start Rational Application Developer in a new workspace that is logically associated with the integration view, such as C:\workspaces\lziosi\_UCM\_int.
2. Connect to ClearCase by selecting **ClearCase** → **Connect**.
3. Open the ClearCase Explorer perspective by selecting **Window** → **Open Perspective** → **Other** → **ClearCase Explorer**.
4. In the ClearCase Navigator, you see both the development view and the integration view.

5. If you have not yet configured the load rules for the Integration view, perform these steps:
  - a. Right-click the Integration view and select **Show ClearCase View Configuration**.
  - b. Right-click the white space in the ClearCase View Configuration panel and select **Edit Configuration**. Add the rule: load \RAD8RedbookUCM.

Now you can import the projects into the integration workspace to review them:

1. Right-click each project under the integration view.
2. Select **Import** → **Import project into Workspace**.
3. Open the Web perspective and review the projects.

After you are satisfied with the projects, you can create a new baseline so that everyone else will be able to start working on these projects.

A *baseline* is a snapshot of a component at a particular time. It consists of the set of versions selected in the stream at the time that the baseline was made. When a new stream is configured, baselines are used to specify which versions are to be selected in that stream. Baselines are immutable so that a particular configuration can be reproduced as needed and streams that use the same set of baselines are guaranteed to have the same configuration. Therefore, the set of versions included in a baseline cannot be modified. For more information about baselines and their uses, see this website:

[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc\\_proj.doc/c\\_u\\_ovw\\_bl\\_uses.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc_proj.doc/c_u_ovw_bl_uses.htm)

To make the new baseline, perform these steps:

1. In the ClearCase Navigator view, right-click the integration stream **RAD8Redbook\_integration**, as shown in Figure 31-49 on page 1685.
2. Select **Make Baseline**.



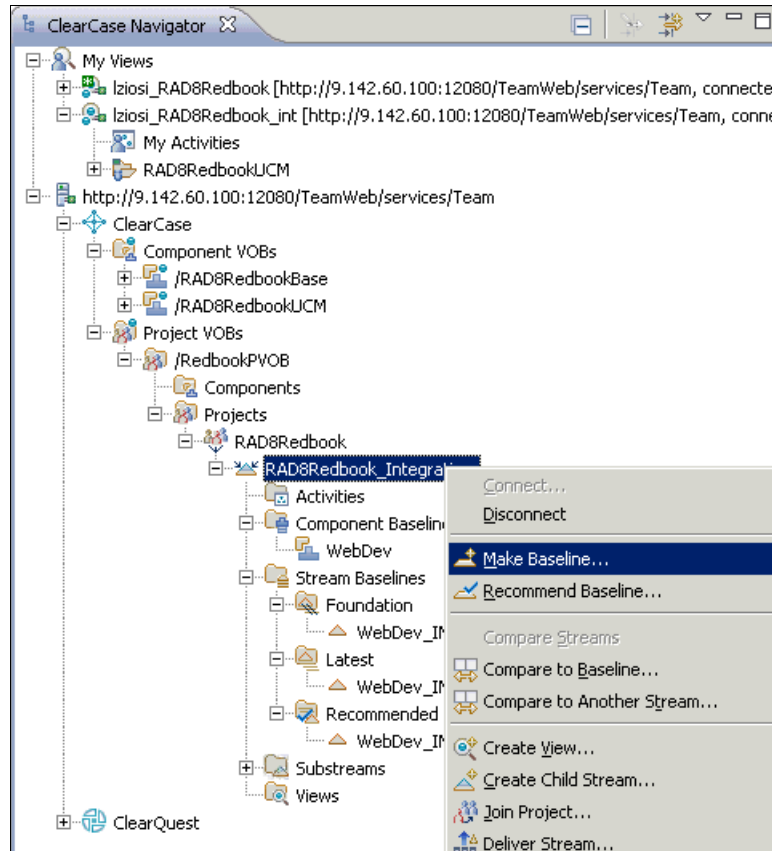


Figure 31-49 Make Baseline

- Review the default name, as shown in Figure 31-50 on page 1686, and change it if required. You can compare this new baseline to the *initial* baseline, and you see which activities and which change sets constitute the delta between these two baselines.

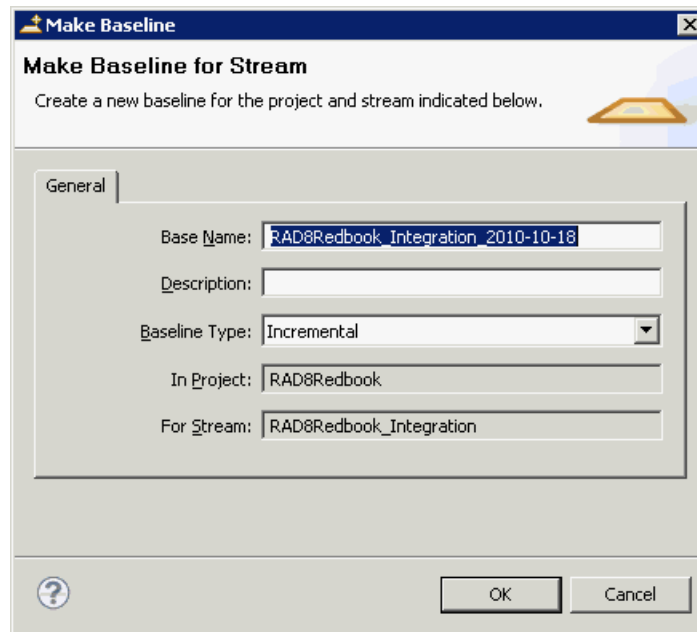


Figure 31-50 Make Baseline for Stream

4. You can now recommend this baseline. *Recommended baselines* are the set of baselines that project team members use to update, or rebase, their development streams. When developers join a UCM project, their development work areas are initialized with the recommended baselines. To recommend the baseline, right-click the integration stream and select **Recommend Baseline**.

Recommending a baseline is typically done by a Project Integrator. In this case, it was possible because this user was also the creator of the PVOB and UCM project. Access requirements, which are documented at this website, exist for all users to be able to recommend a baseline:

[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc\\_proj.doc/c\\_u\\_pol\\_strm\\_bl\\_allw.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc_proj.doc/c_u_pol_strm_bl_allw.htm)

### 31.8.6 A new user joins the project

A new user, Salvatore Sollami, starts working on the same project. Salvatore first launches Rational Application Developer on a new workspace, which is logically associated with his development stream, C:\workspaces\sollami\_UCM.

After connecting to the ClearCase Change Management Server, Salvatore sees no views in the ClearCase Navigator, but he sees the contents of the existing project, as shown in Figure 31-51.

Salvatore now joins the project, following the same steps that are detailed in 31.8.2, “Connecting to the ClearCase Change Management Server and joining a UCM project” on page 1669. Following these steps creates two new views and one new development stream for Salvatore:

- ▶ Development view: sso11ami\_RAD8Redbook
- ▶ Development stream: sso11ami\_RAD8Redbook
- ▶ Integration view: sso11ami\_RAD8Redbook\_int

At this point, from the development view that is shown in the ClearCase Navigator, Salvatore right-clicks the projects and selects **Import** → **Import project into Workspace** to populate his development workspace. It is possible to select multiple projects by holding down Shift or Ctrl.

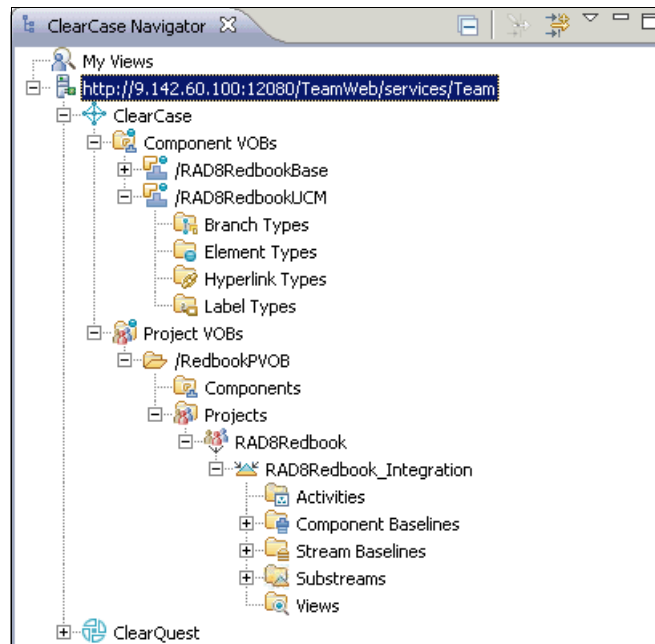


Figure 31-51 ClearCase Navigator contents for a new user

Salvatore starts by setting a new UCM Activity called AddCounterBean:

1. In the ClearCase Navigator view, right-click the development view.
2. Select **New Activity**.
3. For name, type AddCounterBean.

The default activity now appears set in the toolbar, even when you change to the Java EE perspective.

Within this activity, Salvatore adds a new EJB 3.1 Singleton by performing these steps:

1. In the Java EE perspective, select **File** → **New** → **EJB**.
2. For Java package, type `com.ibm.itso`.
3. For Class name, type `CounterBean`.
4. For type, select **Singleton**.
5. For Create business interface, accept to create a **No-interface** bean.
6. Select **Finish**. See Figure 31-52.

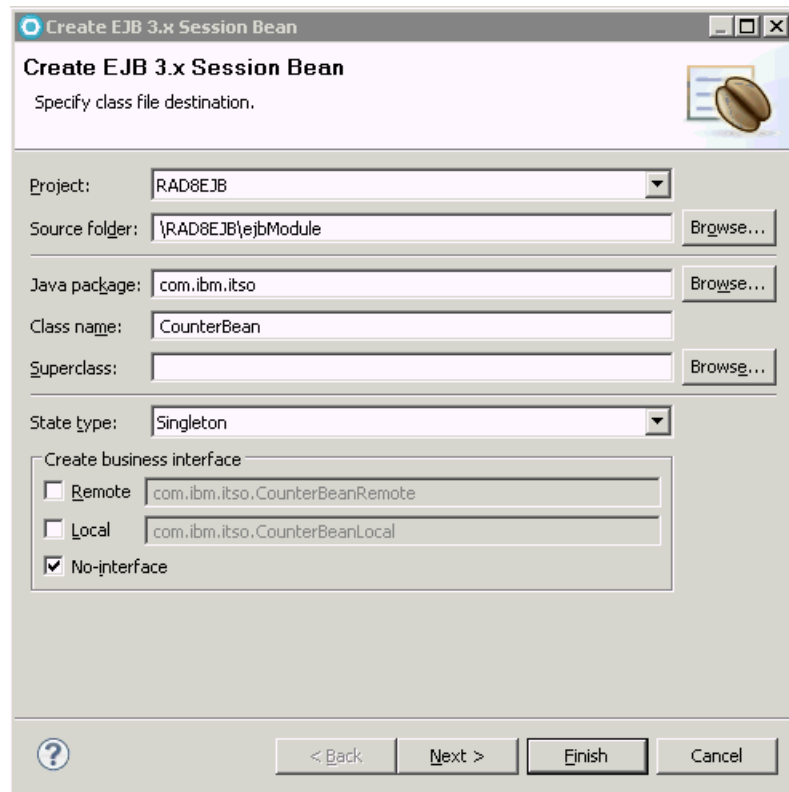


Figure 31-52 Creating a Singleton EJB

ClearCase Remote Client detects all artifacts that Rational Application Developer creates and prompts the user to add the source control. Allow ClearCase to

complete this operation. *The additions are performed in the context of the UCM Activity that has been chosen.*

The following file is added to source control and checked in:

ejbModule\com\ibm\itso\CounterBean.java

The following files are added to source control and checked out:

- ▶ classdiagram.dnx
- ▶ ejbModule\META-INF\ejb-jar.xml

Salvatore implements the EJB with code, as shown in Example 31-2.

*Example 31-2 EJB implementation*

---

```
package com.ibm.itso;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.ejb.LocalBean;
import javax.ejb.Singleton;
import javax.ejb.Startup;

@Singleton
@LocalBean
@Startup
public class CounterBean {

 private int counter;

 public void increment() {
 counter++;
 System.out.println("Singleton, with counter = " + counter);
 }
 @PostConstruct
 public void init() {
 System.out.println("Singleton, initialization");
 };
 @PreDestroy
 public void destroy() {
 System.out.println("Singleton, destruction");
 }
}
```

---

As soon as he starts editing the code, Salvatore is prompted to check it out, as shown in Figure 31-53 on page 1690.

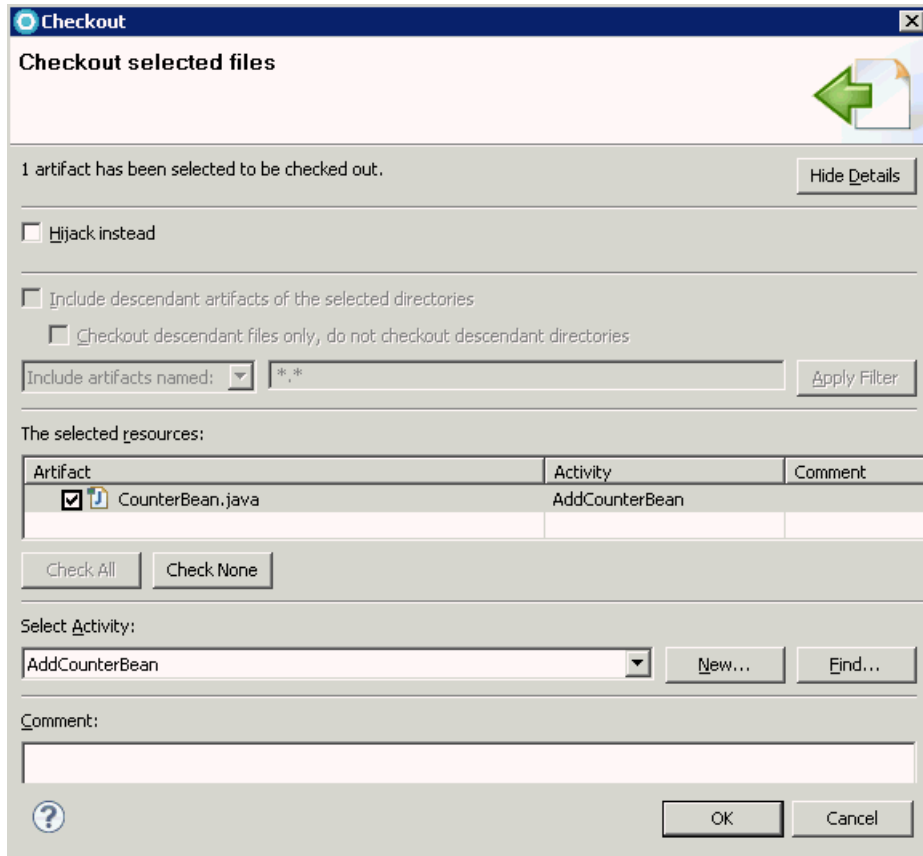


Figure 31-53 Prompt to check out the EJB code

To test the EJB, Salvatore adds a new dynamic web project called RAD8Servlet, which is associated to the same EAR. Before adding this new project to source control, Salvatore creates a new activity called AddCounterServlet.

Salvatore creates a new servlet with code, as shown in Example 31-3, to test the EJB.

*Example 31-3 Servlet to test the CounterBean*

```
package com.ibm.itso.servlets;

import java.io.IOException;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
```

```

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.ibm.itso.CounterBean;

@WebServlet("/CounterServlet")
public class CounterServlet extends HttpServlet {
 private static final long serialVersionUID = 1L;

 public CounterServlet() {
 super();
 }

 @EJB
 CounterBean counterBean;

 protected void doGet(HttpServletRequest request,
 HttpServletResponse response) throws ServletException,
 IOException {
 counterBean.increment();
 }
}

```

---

After testing the servlet on WebSphere Application Server V8 Beta, Salvatore performs these tasks:

1. Opens the ClearCase Explorer.
2. Right-clicks **ssollami\_RAD8Redbook\_int** on the Integration view.
3. Selects **Refresh** → **Update from Repository**.
4. Rebases the development stream and finds that there is no new baseline.
5. Right-clicks **ssollami\_RAD8Redbook** on the Development view.
6. Selects **Deliver** → **Advanced**.
7. Selects to deliver the two activities, AddCounterBean and AddCounterServlet, to the default stream.
8. Checks in any files that are still checked out by selecting the **Checkouts/Hijacks** tab in the Advanced Delivery dialog window.
9. Completes the delivery when prompted.

### 31.8.7 Another user modifies the same project

Lara has the task to add a new EJB to the project, a SchedulerBean. In her development workspace, she creates a new activity: AddSchedulerBean.

Lara adds a new EJB with the code, as shown in Example 31-4, using the New EJB Wizard.

*Example 31-4 SchedulerBean code*

---

```
package com.ibm.itso;

import javax.ejb.LocalBean;
import javax.ejb.Schedule;
import javax.ejb.Singleton;
import javax.ejb.Startup;

@Startup
@Singleton
@LocalBean
public class SchedulerBean {

 public SchedulerBean() {

 }

 @Schedule(second="*", minute="*", hour="*", dayOfWeek="*",
persistent=false)
 public void calledEverySecond(){
 System.out.println("Called every second");
 }

}
```

---

After testing the code on WebSphere Application Server V8 Beta, Lara is ready to deliver this activity. In her change set, Lara must merge the file `classdiagram.dnx` that shows views of both EJBs. For that merge to work correctly, Lara needs to configure Rational Software Architect as the merge provider for files with the extension `*.dnx`. On the preference page, Lara selects **Team** → **ClearCase Remote Client** → **Compare/Merge**, as shown in Figure 31-54 on page 1693 and performs the following steps:

1. Select **Add** near “Override the default tool for the following types”.
2. For Resource type, enter `*.dnx`.



3. Select **Rational Software Architect** from the drop-down list for both Compare Provider and Merge Provider.
4. Select **OK**.

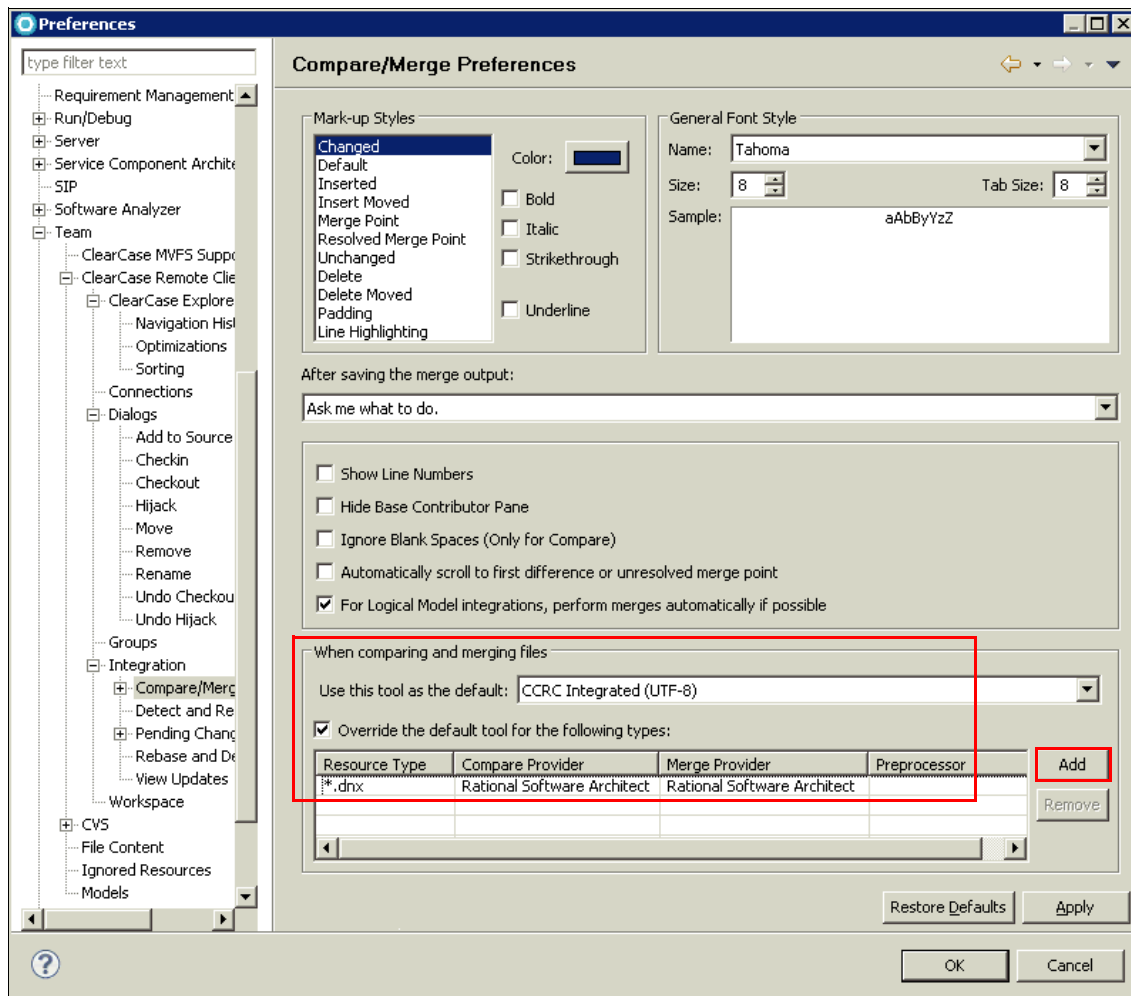


Figure 31-54 Adding Rational Software Architect as the Compare and Merge Provider for .dnx files

Finally, Lara can perform the following steps:

1. Update the integration stream. This step shows Lara that a new project, RAD8Servlet, has been added in the meantime and that a new EJB had been added to the EJB project, as shown in Figure 31-55 on page 1694.

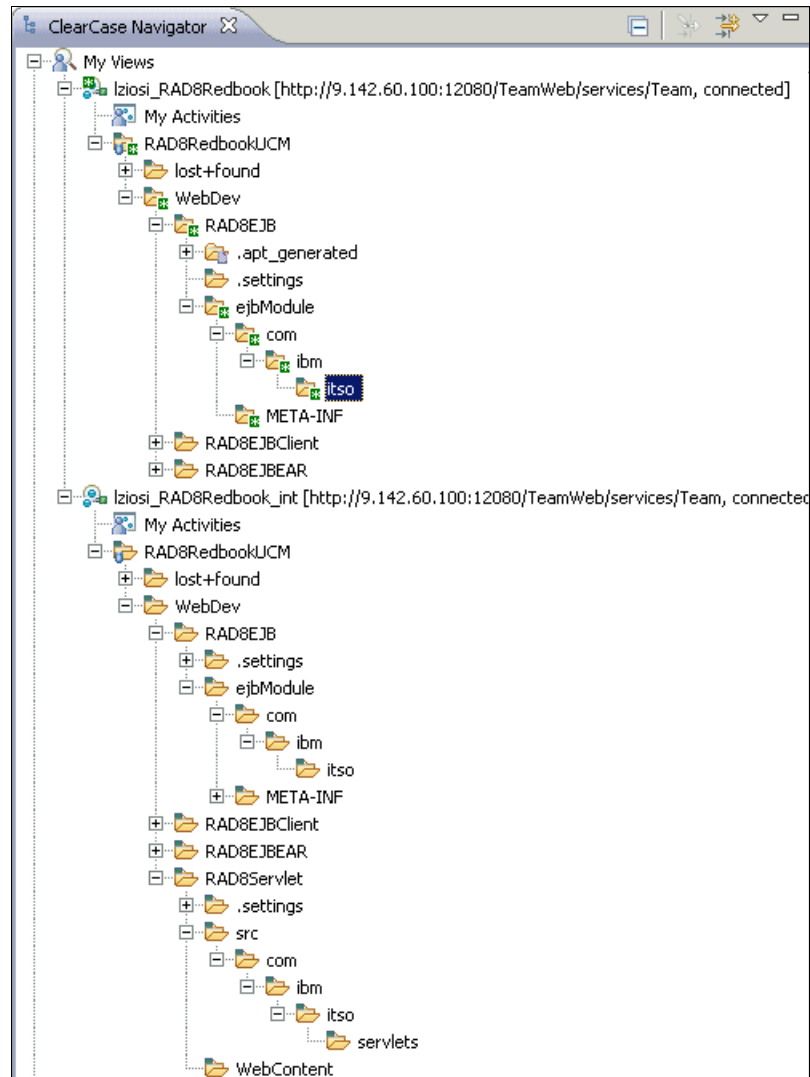


Figure 31-55 RAD8Servlet and ITSO have been added

2. Lara selects the **Development** view and launches **Deliver** → **Advanced Deliver**, as shown as in Figure 31-56 on page 1695.
3. Lara selects the activity to deliver.
4. Lara selects Check in or Undo Checkout (for ejb-jar.xml) for the checked out files.
5. All merges are trivial, including the merger of the diagram, so no user intervention is required.

6. Lara completes the delivery when prompted.
7. At this point, if desired, it is possible to create a new baseline to include all EJBs and the servlet for future use by other team members who rebase their development streams.

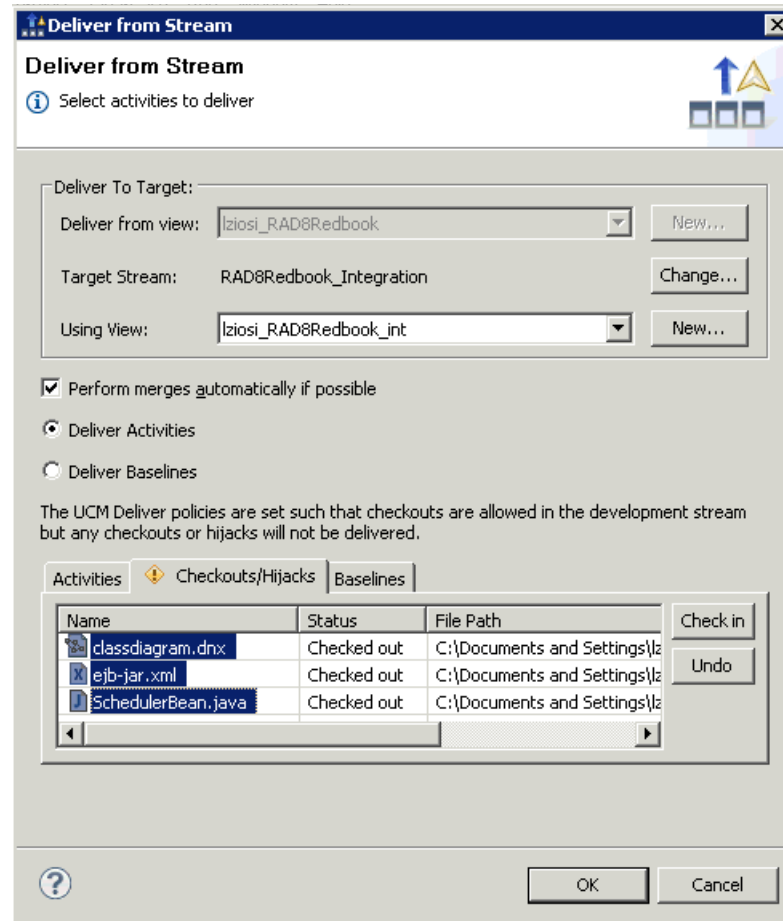


Figure 31-56 Advanced delivery dialog window showing the files that are checked out

In this section, we have explored the integrated functionality that is offered by the ClearCase Remote Client Extension. In particular, we have seen how two developers can work in parallel and in isolation on the same Rational Application Developer projects after joining one UCM project. The developers perform most of their work using their development streams and views. We have seen how they can then deliver their changes to the integration stream and create new baselines for the entire team to use. We have seen how mergers are handled

during *deliver* and we have configured a specific compare and merge provider that is required for properly merging UML diagrams.

## 31.9 More information

For more information about Rational ClearCase and the integration with Rational Application Developer, refer to these resources:

- ▶ The “Rational ClearCase SCM Adapter” chapter of the Rational Application Developer Help
- ▶ Information center for ClearCase 7.1:  
[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m0/index.jsp?topic=/com.ibm.rational.clearcase.help.ic.doc/helpindex\\_clearcase.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m0/index.jsp?topic=/com.ibm.rational.clearcase.help.ic.doc/helpindex_clearcase.htm)
- ▶ Information center for ClearCase 7.1.2:  
[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.help.ic.doc/helpindex\\_clearcase.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.help.ic.doc/helpindex_clearcase.htm)
- ▶ Integration document:  
<http://www-01.ibm.com/support/docview.wss?rs=0&uid=swg27019539>
- ▶ Comparing and Merging UML Models with Rational ClearCase:  
[http://www.ibm.com/developerworks/rational/library/07/0703\\_letkeman/](http://www.ibm.com/developerworks/rational/library/07/0703_letkeman/)  
<http://www.ibm.com/developerworks/rational/library/10/comparing-and-merging-uml-models-in-ibm-rational-software-architect/index.html>



## Code Coverage

Code Coverage is an important aspect of software testing and can be considered fundamental to the overall system testing of a component. The motivation behind coverage tooling is to give developers and testers more insight into the areas of code that are being exercised by a set of test cases. This information is useful to developers and testers who can then use it to devise new test cases so that adequate coverage can be achieved.

This chapter uses an example project, which is included in the sample code archive that accompanies the book. Import the code in `\7835code\codecoverage` into your workspace if you want to follow our steps.

## 32.1 Overview

This section introduces the instrumentation engine that is used by Code Coverage and compares basic blocks to executable units.

### 32.1.1 Instrumentation

To properly analyze the coverage statistics, it is important to understand the technology that is used in the background.

Code Coverage uses an instrumentation engine to manipulate the bytecode of a class and inject custom calls to the coverage data collection engine. Figure 32-1 provides a high-level overview of the process.

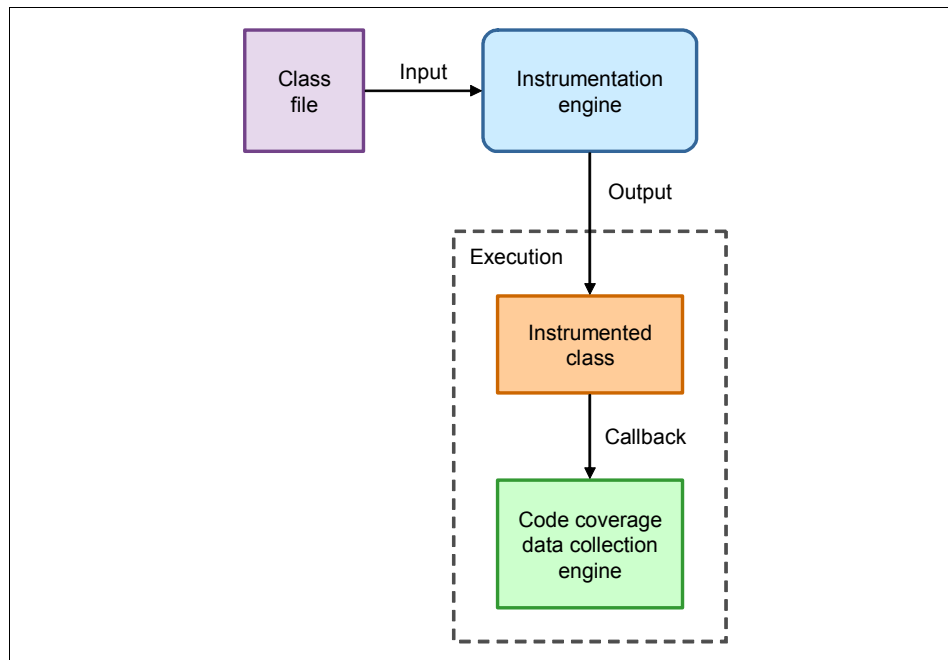


Figure 32-1 Overview of the Code Coverage execution environment

### 32.1.2 Basic blocks versus executable units

The instrumentation engine operates on units of bytecode that are called executable units. The definition of an executable unit differs slightly from the traditional definition of a basic block, but the differences are important to consider when the results are analyzed.

By definition, a basic *block* is a set of instructions that cannot be branched into or out of. The key idea is that when the first instruction runs, all of the subsequent instructions in that block are guaranteed to be executed without interruption. It follows that a basic block can be conceptually considered as a single group or block of instructions. In general, basic blocks end on branch, call, throw, or return statements.

An *executable unit* begins at the start of every basic block and at any instruction that corresponds to a line of source code that differs from the previous instruction. What differentiates an executable unit from a basic block is the condition that triggers the end of the executable unit. For example, the divide instruction is not considered to be the end of an executable unit despite the fact that it can throw an exception.

The instrumentation engine in Code Coverage is used to inject custom code at the start of every executable unit. Consequently, you can customize the Code Coverage feature to report statistics down to the executable unit level of granularity (or *block coverage*). Figure 32-2 provides an overview of how the instrumentation engine modifies the bytecode to support Code Coverage.

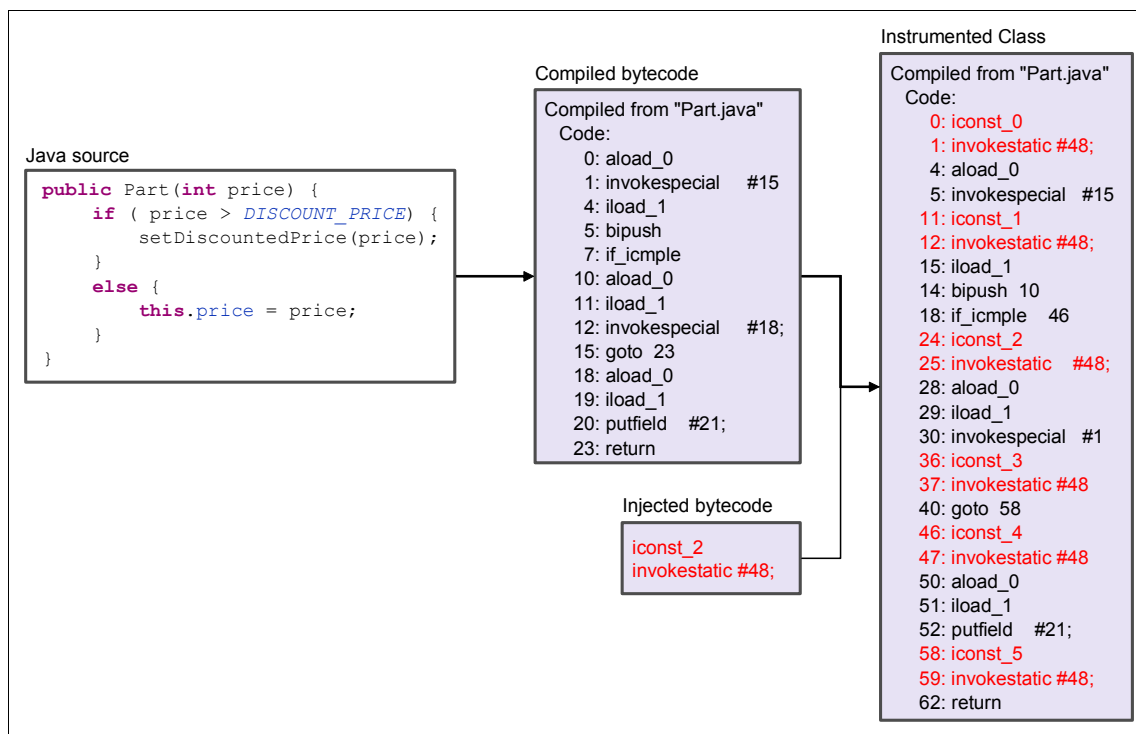


Figure 32-2 Overview of the bytecode instrumentation

## 32.2 Generating coverage statistics in Rational Application Developer

One of the major advantages of Code Coverage is that you can enable it on any Java project in Rational Application Developer by navigating to the **Code Coverage** panel in the project Properties pane, as shown in Figure 32-3.

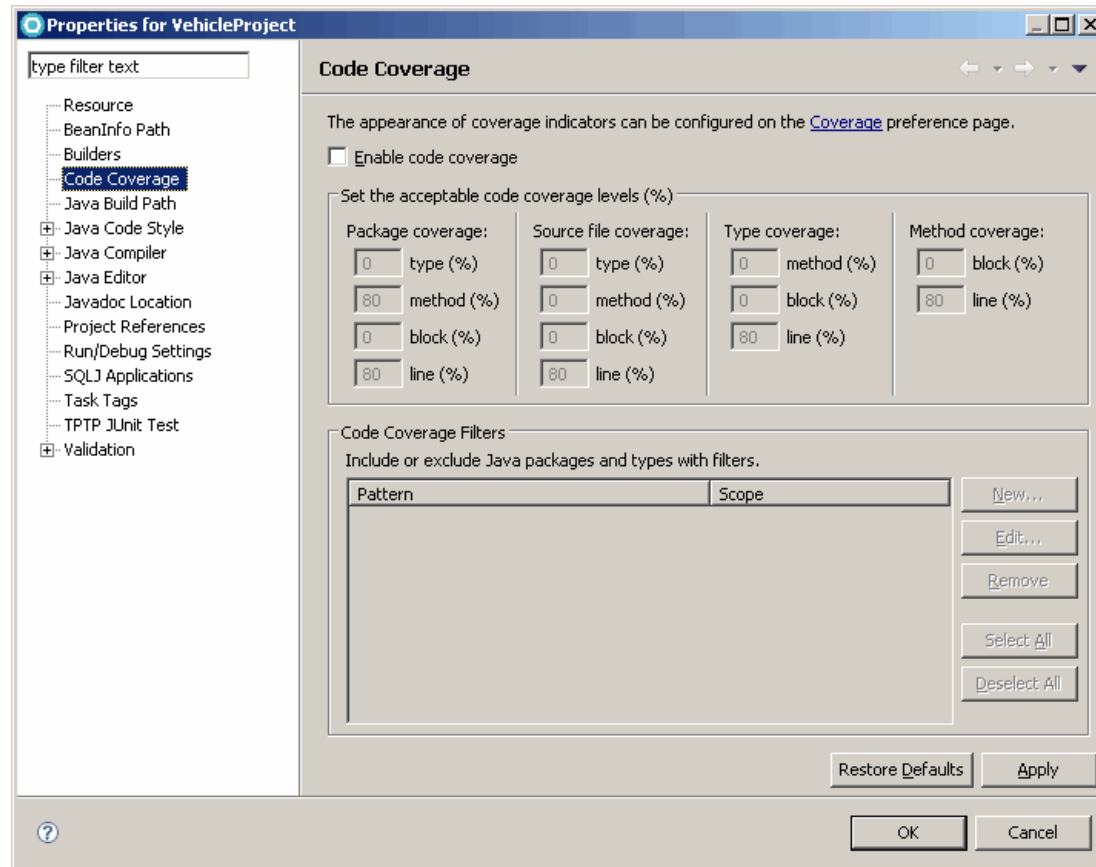


Figure 32-3 Code Coverage panel in the project Properties pane

Select the **Enable code coverage** check box in Figure 32-3 to enable Code Coverage for the project. You can also use this panel to customize acceptable coverage levels for each step of granularity. Code Coverage supports the following levels of granularity:

- ▶ Type coverage: The percentage of types that are covered in a class
- ▶ Method coverage: The percentage of methods that are covered in a class



- ▶ Line coverage: The percentage of lines that are covered in the class file
- ▶ Block coverage: The percentage of blocks that are covered in a class file. A *block* refers to an executable unit.

You can also specify custom filters, which are used to control what gets instrumented in your project. By default, all of the classes in your project are instrumented, but you can create custom filters to exclude target packages or specific types, if you need to restrict the results.

### 32.2.1 Viewing results in the Package Explorer

After you enable Code Coverage on a project, coverage statistics are generated the next time that the application is launched. Statistics are not generated for all types of launch configurations automatically. The following launch types are supported from within Rational Application Developer:

- ▶ Java Applet
- ▶ OSGi Framework
- ▶ JUnit
- ▶ JUnit Plug-in Test
- ▶ Java Application
- ▶ Eclipse Application
- ▶ Standard Widget Toolkit (SWT) Application

The provided sample application is a simple representation of various vehicles (car, van, motorcycle, and transport truck) and the various parts that are associated with each vehicle. Figure 32-4 shows a Unified Modeling Language (UML) diagram outlining the structure of this application.

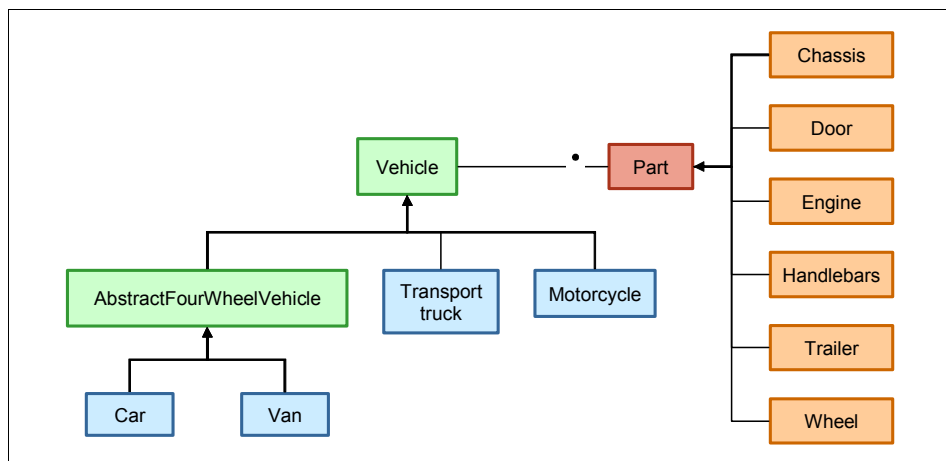


Figure 32-4 UML diagram for the sample application

There are two JUnit tests that are already defined in the project: `TestCar.java` and `Test-CarImproved.java`. These tests target the `Car.java` class. While in the Java perspective in Rational Application Developer, you can start the `Test-Car.java` test by right-clicking **TestCar.java** and selecting **Run As** → **JUnit test**. The results of the JUnit test appear in the JUnit view. The coverage results are integrated into the Rational Application Developer user interface (UI), and you can analyze them by switching back to the Package Explorer view. Figure 32-5 displays a sample result set for the `TestCar.java` test.

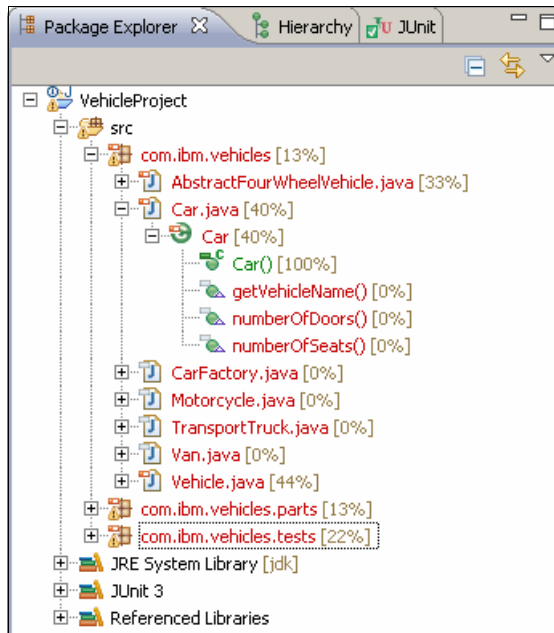


Figure 32-5 Coverage results for `TestCar.java` displayed in the Package Explorer

By default, the UI is annotated with only the line coverage information; however, you can change this annotation in the workbench preferences and optionally choose to include coverage for packages, types, and blocks. The percentage beside each Java item is a breakdown of the line coverage for the last execution. You can drill down into the various Java artifacts (for example, classes, types, and methods) in the Package Explorer to get coverage statistics at a lower level of granularity.

The results are color-coded depending on the success rate: by default, red indicates that the acceptable coverage level has not been met and green indicates that the appropriate coverage level was achieved. Naturally, the goal of the test is to reach an acceptable coverage level on the classes of interest.

Based on the results that are shown in Figure 32-5 on page 1702, the first test was inadequate: the `Car` class (and abstract parents `AbstractFourWheelVehicle` and `Vehicle`) did not reach the appropriate coverage level. Luckily, you have a second attempt to execute: `TestCarImproved.java`. Again, you can execute the test as a normal JUnit, and the results are automatically updated in the Package Explorer (Figure 32-6).

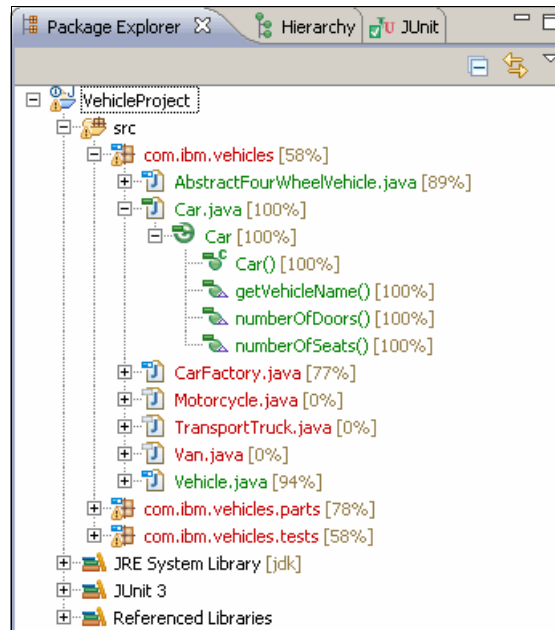


Figure 32-6 Code Coverage results for `TestCarImproved.java` displayed in the Package Explorer

## 32.2.2 Viewing results in the Java Editor

Line coverage results are also displayed and marked in the Java editor, and you can use it to give a precise report of which lines are covered in each class. After coverage statistics have been generated, you can open any class in your project with the Java editor, and the left ruler bar in the editor shows the coverage information. Figure 32-7 on page 1704 displays the results for `Vehicle.java`.

```
Vehicle.java X
public String getDescriptionString() {
 StringBuffer buffer = new StringBuffer();
 buffer.append("Vehicle name: " + getVehicleName() + "\n");
 buffer.append("-----\n");
 buffer.append("Contains parts:\n");
 for(int i = 0; i < parts.size(); i++)
 buffer.append(parts.get(i).getPartName() + "\t" + part

 return buffer.toString();
}

public boolean setTargetSpeed(int speed) {
 if (speed > MAX_SPEED) return false;
 targetSpeed = speed;
 return true;
}

public List<Part> getPartsList() {
 return parts;
}
```

Figure 32-7 Coverage results displayed in the Java editor

The color indicators are the same as the color indicators in the Package Explorer view. That is, by default, a green line was covered and a red line was not covered. There is a slight advantage in viewing the results in the Java Editor because it also indicates the partially covered lines. Partially covered lines can occur when multiple executable units are on a line of source code but only one executable unit has been executed. As an example, look at the first line of code in the `setTargetSpeed(int speed)` method that is shown in Figure 32-7. The first executable unit is the `if` statement, and the second executable unit is the `return` statement. By default, a partial line is colored in yellow.

## 32.3 Generating reports

You can compile the Code Coverage results into reports and view them in Rational Application Developer, or you can save them to the file system for future analysis. You can generate two types of reports: workbench reports (Eclipse-based) and HTML reports. To generate a report, select **Run** → **Code Coverage** → **Generate Report**. Figure 32-8 on page 1705 shows the report generation dialog window.

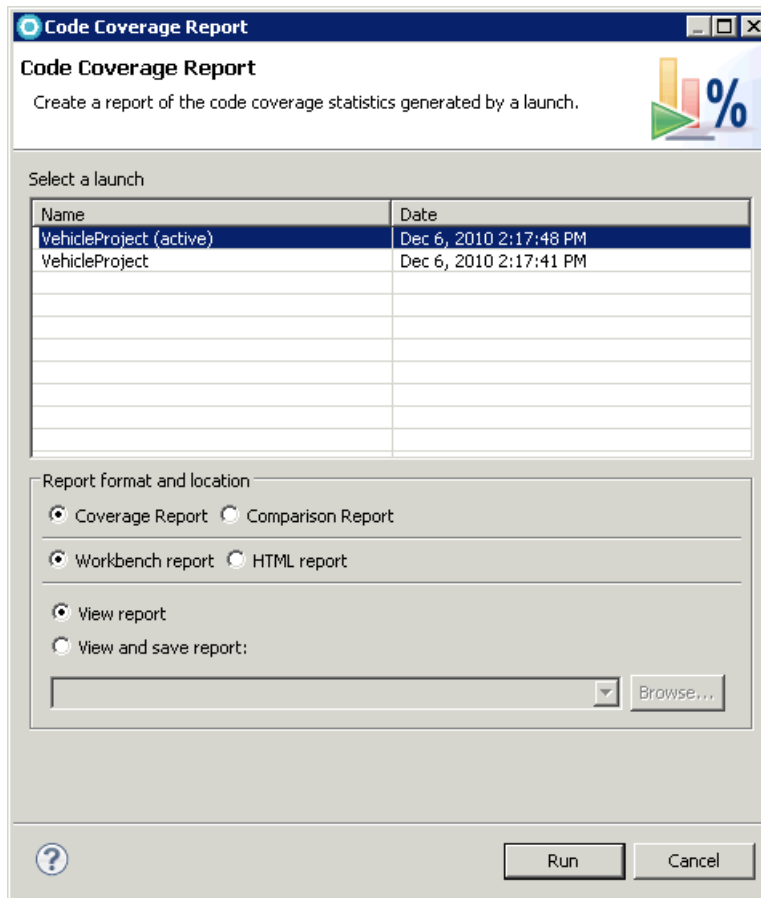


Figure 32-8 Report generation dialog window

You can create and view a report in Rational Application Developer using the **View Report** option on the dialog or save the report to the file system using the **View and save report** option.

### 32.3.1 Workbench reports

The workbench reports provide a consolidated view of all of the coverage statistics for your project and contain coverage information for all of the classes in your project at execution time. Figure 32-9 on page 1706 shows a populated workbench report.

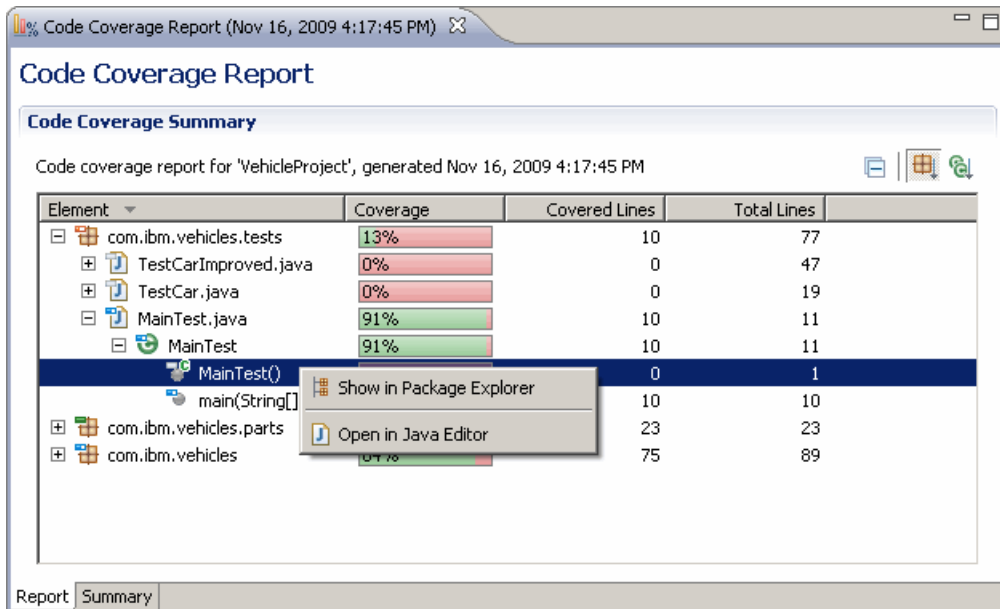


Figure 32-9 Coverage results in an Eclipse-based report

Workbench reports have the added advantage of being integrated in Rational Application Developer, so that you can use them as quick tools to provide insight into the parts of your code that require improved test coverage. As Figure 32-9 shows, the statistics in a workbench report contain coverage information for all levels of granularity: from a package to a method. Right-clicking any of the Java artifacts displays a pop-up menu with two additional actions: Show in Package Explorer and Open in Java Editor. These useful tools can help you identify and investigate the areas of code with low coverage, because the tools highlight the selected area of code by opening it in the appropriate viewer or editor.

### 32.3.2 HTML reports

HTML reports display the same type of information that is provided in the workbench report, but in HTML format. These reports are particularly useful when saved to the file system, because they provide a way for the coverage results to be analyzed independently of Rational Application Developer, shared with team members, or published to a website for viewing.

## 32.4 Generating statistics outside of the workbench

One of the major features of the Code Coverage tool is its ability to generate statistics outside of Rational Application Developer. This capability provides extra flexibility and enables you to customize your environment to take advantage of Code Coverage in your systems. For example, one natural combination is to set up a nightly build environment and generate statistics with JUnit tests on the nightly driver.

You can integrate Code Coverage into your environment by performing the following three steps: instrumentation, execution, and report generation. Code Coverage starting in Rational Application Developer v8.0 and later supports both static and dynamic approaches to instrumentation. Users can decide which instrumentation style to use, depending on their needs.

Both approaches to instrumentation output a baseline file. A baseline file is a concept that is specific to the Code Coverage feature. The *baseline file* contains an index of all of the classes in your project, and it maintains additional metadata about each class. This file is used at the reporting step (32.4.3, “Report generation” on page 1712) to determine which classes in your application were not covered. This step is necessary because the Code Coverage data collection engine is only notified of a class when it is loaded by Java virtual machine (JVM), and so, a list of the classes that were not executed cannot be determined without additional metadata. If the baseline file is not present at reporting time, the classes that were not loaded are absent from the report.

### 32.4.1 Static instrumentation

With the static approach, instrumentation is performed before the code is executed by modifying the bytecode of a class and saving the results back to the file system as a `.class` file. This approach has the advantage that the overhead of instrumentation only has to be dealt with one time. The disadvantage is clear when you consider a class that changes frequently and therefore requires instrumentation for every update.

#### Instrumentation

There are two approaches that you can use to instrument your application. The first approach is to use the instrumentation Ant task that is provided by Code Coverage. Example 32-1 shows an example usage of the instrument task configured to target the sample application in this article.

*Example 32-1 Example usage of instrument Ant tasks on the sample application*

---

```
<target name="instrument">
```

```

 <path id="lib.path">
 <pathelement location="<Rational Application Developer
home>\plugins\com.ibm.rational.llc.engine_<date>"/>
 <pathelement location="<Rational Application Developer
home>\plugins\org.eclipse.hyades.probekit_<date>\os\<platform>\x86\prob
ekit.jar"/>
 </path>
 <taskdef name="instrument"
classname="com.ibm.rational.llc.engine.instrumentation.anttask.Instrume
ntationTask" classpathref="lib.path"/>
 <instrument saveBackups="true"
 baseLineFile="VehicleProjectStatistics.baseline"
 buildPath="VehicleProject"
 outputDir="VehicleProjectInstr"/>
 </target>

```

---

To use this Ant task, you must add the *<Rational Application Developer home>\plugins\org.eclipse.hyades.probekit\_<date>\os\<platform>\x86* path to the systems PATH environment variable.

Table 32-1 outlines a quick overview of the expected parameters.

Table 32-1 Input parameters for instrumentation tasks

Parameter	Description
buildPath	The path to the project on the file system.
outputDir	Optional: The output directory of the instrumented project. If not specified, the classes in the buildPath will be instrumented in place.
baseLineFile	Optional: The output location of the baseline project index file.
saveBackups	Optional: Set to true if the original class files must be backed up before instrumenting.

The second approach is to use the instrument.bat/sh script provided in the *<RAD\_HOME>\plugins\com.ibm.rational.llc.engine\_<date>\scripts\static* directory. Table 32-1 outlines the script input parameters.

## Execution

To execute the instrumented classes, you must configure the Java environment correctly at launch.



The following two specific parameters are necessary for execution:

- ▶ `Dcoverage.out.file=<absolute path to output file>`: The file that is specified by this JVM argument is the output location of the coverage statistics.
- ▶ Add the `<Rational Application Developer HOME>/plugins/com.ibm.rational.llc.engine_<date>/RLC.jar` to the class path. Because the code has been instrumented with callbacks to the Code Coverage data collection engine, the `RLC.jar` file needs to be on the class path at run time.

These parameters can be supplied to a JUnit Ant task. Example 32-2 provides example usage.

*Example 32-2 How to specify the Code Coverage feature arguments in an Ant launch*

---

```
<target name="run">
 <junit showoutput="true" fork="yes">
 <jvmarg value="-Dcoverage.out.file={absolute path to the output
file}"/>
 <classpath>
 <pathelement location="{absolute path to the
 <Rational Application Developer
HOME>\plugins\com.ibm.rational.llc.engine_<date>
 \RLC.jar file}"/>
 <pathelement location="{path to the project classes}"/>
 <pathelement path="{absolute path to the junit.jar}" />
 </classpath>
 <test name="com.ibm.vehicles.tests.TestCar" outfile="TestCar" />
 </junit>
</target>
```

---

## 32.4.2 Dynamic instrumentation

With dynamic instrumentation, the instrumentation step is performed by modifying the bytecode of a class in memory at run time when the classes are loaded. It is easier to work with this approach, because there is no explicit instrumentation step and only the classes that are loaded by the JVM are instrumented. If a large number of classes are being executed, the instrumentation overhead might affect the performance of your application.

### Generating the probescript and baseline files

The probescript file is important in a dynamic instrumentation environment and is a requirement for execution. Code Coverage uses the probescript file at run time

to determine which classes are instrumented and for which classes data is collected.

Code Coverage provides a command-line script and Ant support, which can be used to generate the probescript and baseline files. Example 32-3 shows an example of using the Ant support.

*Example 32-3 Ant task to generate probescript and baseline files for a project*

```
<target name="application-analysis" description="Define the code
coverage application analysis task">
 <path id="lib.path">
 <pathelement location="<Rational Application Developer
home>\plugins\org.eclipse.jdt.core_<date>.jar"/>
 <pathelement location="<Rational Application Developer
home>\plugins\com.ibm.rational.llc.engine_<date>"/>
 <pathelement location="<Rational Application Developer
home>\plugins\org.eclipse.equinox.common_<date>.jar"/></path>
 <taskdef name="code-coverage-app-analyzer"
classname="com.ibm.rational.llc.engine.instrumentation.anttask.Coverage
ApplicationAnalyzerTask" classpathref="lib.path"/>

 <code-coverage-app-analyzer projectDir="VehicleProject\bin"
probescript="VehicleProjectStatistics.probescript"
baseline="VehicleProjectStatistics.baseline"/>
 </target>
```

Table 32-2 shows the parameters that are taken as input.

*Table 32-2 Input parameters for instrumentation tasks*

Parameter	Description
<i>projectDir</i>	The directory containing the Java project
<i>probescript</i>	The output location for the generated probescript file
<i>baseline</i>	Optional: The output location of the baseline project index file

The command-line script to generate probescript and baseline files is in  
*<Rational Application Developer  
home>\plugins\com.ibm.rational.llc.engine\_<date>\scripts\dynamic.*  
Executing the *appinfo.bat/sh* script with the expected parameters generates the  
probescript and baseline files.

Generate the probescript file to an easily accessible location on the file system.  
The probescript file needs to be provided as input for the execution step. *You can*

reuse a probescript file multiple times, but you must regenerate it if classes are added or removed from the project.

We provide a default probescript file `<Rational Application Developer home>/plugins/com.ibm.rational.llc.engine_<date>/scripts/dynamic/default.probescript`. You can use this probescript file to avoid generating a custom probescript, but this default probescript file might not be ideal for all applications. This probescript provides a good default instrumentation filter set, but it might exclude or include extra classes (depending on the structure of the project).

## Execution

To execute the instrumented classes, you must configure the Java environment correctly at launch. The following three specific parameters are necessary for execution:

- ▶ `Dcoverage.out.file=<absolute path to output file>`: The file specified by this JVM argument is the output location of the coverage statistics.
- ▶ Add the `<Rational Application Developer HOME>/plugins/com.ibm.rational.llc.engine_<date>/RLC.jar` to the class path. Because the code has been instrumented with callbacks to the Code Coverage data collection engine, the `RLC.jar` file needs to be on the class path at run time.
- ▶ Add the `-agentpath:<path to JPIBootLoader>=JPIAgent:server=standalone,file=;ProbekitAgent:ext-pk-BCILibraryName=BCIEngProbe,ext-pk-probescript=<path to probescript>` JVM argument. This argument registers the instrumentation engine with the JVM so that Code Coverage can instrument the classes when loaded.

The `<path to probescript>` parameter is the path to the probescript file to be used for execution. This file can be either the default probescript file or the probescript file that was generated in the previous step.

**The `<path to JPIBootLoader>` parameter:** On Windows, the `<path to JPIBootLoader>` parameter is `<Rational Application Developer HOME>\plugins\org.eclipse.tptp.platform.jvmti.runtime_<date>\agent_files\win_ia32\JPIBootLoader`.

On Linux, the `<path to JPIBootLoader>` parameter is `<Rational Application Developer HOME>/plugins/org.eclipse.tptp.platform.jvmti.runtime_<date>/agent_files/linux_ia32/libJPIBootLoader.so`.

Example 32-4 on page 1712 provides an example of an Ant call to run an application and generate coverage statistics.

*Example 32-4 Ant call to run an application and generate statistics*

---

```
<target name="execute">
 <jar basedir="VehicleProject\bin" destfile="VehicleProject.jar" />
 <java classname="com.ibm.vehicles.tests.MainTest" fork="true"
newenvironment="true">
 <jvmarg
value="-Dcoverage.out.file=VehicleProject.coveragedata" />
 <jvmarg value="-Xbootclasspath/a:<Rational Application
Developer home>/plugins/com.ibm.rational.llc.engine_<date>/RLC.jar" />
 <jvmarg value="-agentpath:<path to
JPIBootLoader>=JPIAgent:server=standalone,file=;ProbekitAgent:ext-pk-BC
ILibraryName=BCIEngProbe,ext-pk-probescript=<path to probescript
file>" />
 <classpath>
 <pathelement path="VehicleProject.jar" />
 </classpath>
 </java>
</target>
```

---

### 32.4.3 Report generation

You can generate reports using another Ant task that is provided by the Code Coverage feature. This task uses the reporting functionality that is provided by the Business Intelligence and Reporting Tools (BIRT) Eclipse.org project and requires that you download the latest BIRT Reporting Engine stand-alone offering. Navigate to <http://www.eclipse.org/birt/download>, select the latest release, and download the **Report Engine** offering. After downloading, extract the archive to a location on the file system.

Example 32-5 provides sample usage of the reporting Ant task. As input, it requires the generated coveragedata file, which was generated in the execution step, and optionally the generated baseline file.

*Example 32-5 Usage of the report generation Ant task on the sample application*

---

```
<target name="generate-report">
 <path id="lib.path">
 <pathelement location<Rational Application Developer
home>\plugins\com.ibm.rational.llc.common_<date>.jar"/>
 <pathelement location="<Rational Application Developer
home>\plugins\com.ibm.rational.llc.report_<date>" />
 <pathelement location="<Rational Application Developer
home>\plugins\org.eclipse.equinox.common_<date>.jar"/>
 </path>
 <fileset dir="<BIRT Report Engine home>/lib" includes="*.jar"/>
```

```

</path>
<taskdef name="code-coverage-report"

classname="com.ibm.rational.llc.report.birt.adapters.ant.ReportGenerati
onTask"
 classpathref="lib.path"/>
<code-coverage-report
 outputDir="VechicleProjectReport"
 coverageDataFile="VehicleProjectStatistics.coveragedata"
 baseLineFiles="VehicleProjectStatistics.baseline"/>
</target>

```

---

Table 32-3 shows the parameters for the report generation Ant task.

*Table 32-3 Input parameters for instrumentation tasks*

Parameter	Description
outputDir	The output directory of the generated HTML report.
coverageDataFile	The input coveragedata files. Multiple coveragedata files can be supplied by using the correct path separator character for the platform (a semicolon (;) on Microsoft Windows or a colon (:) on Linux).
baseLineFiles	Optional: The input baseline files. Multiple baseline files can be supplied by using the correct path separator character for the platform (a semicolon (;) on Microsoft Windows or a colon (:) on Linux).

Figure 32-10 on page 1714 shows an example HTML report. Generating HTML reports using the Ant task provides a means by which users can view the statistics that are generated in an Ant environment independently of Rational Application Developer.

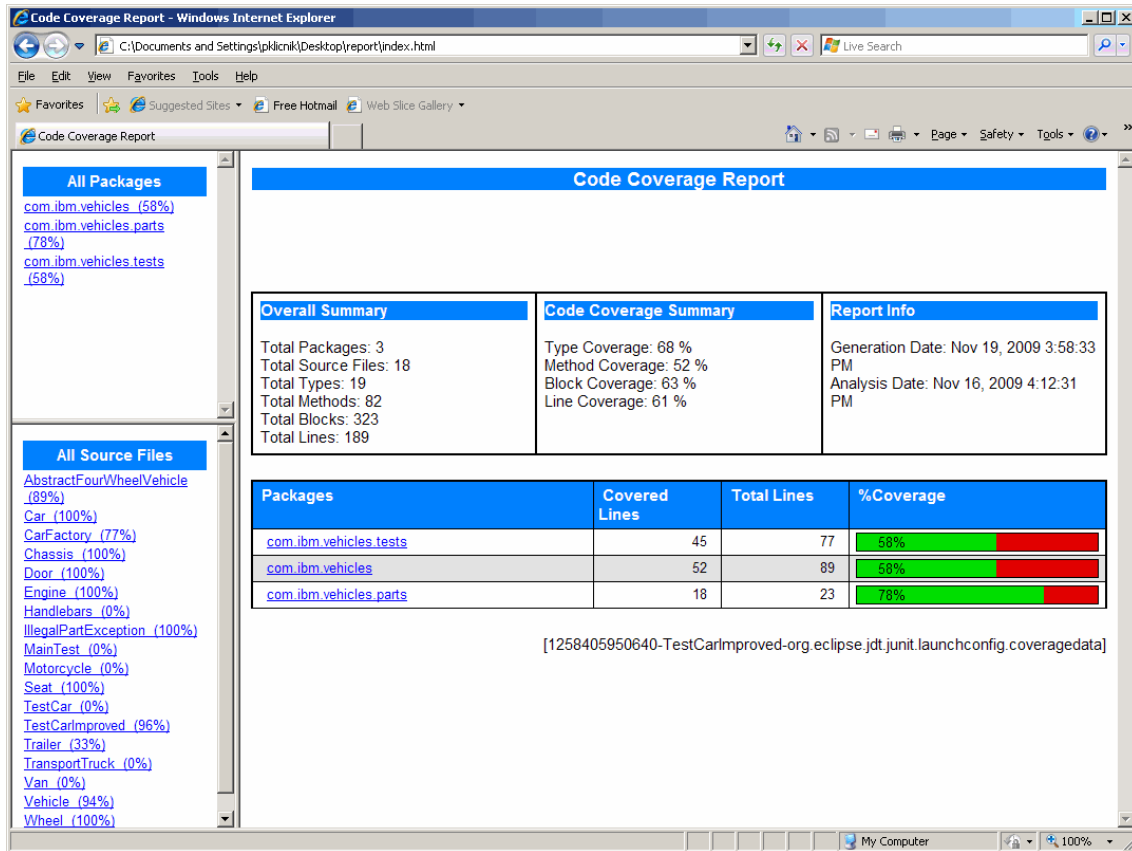


Figure 32-10 Coverage results in an HTML report

## 32.5 Coverage report comparison

You can compare Code Coverage result sets within Rational Application Developer. Comparing coverage results gives a precise value of how much the coverage has changed while your application or tests are updated. You can generate comparison reports either within Rational Application Developer or in an Ant environment.

## 32.5.1 Generating a coverage comparison report in Rational Application Developer

You can generate comparison reports from the report generation dialog window (see Figure 32-11). You can open the Code Coverage Report window by selecting **Run** → **Code Coverage** → **Generate Report**.

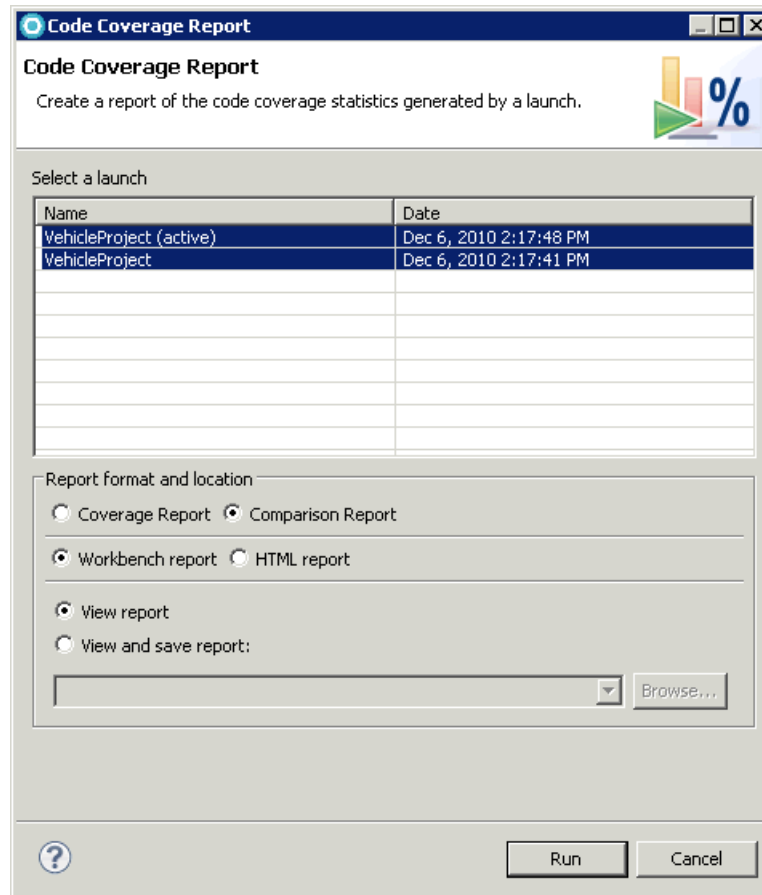


Figure 32-11 Report generation dialog window with comparison options selected

To generate a comparison report, select the **Comparison Report** option from the **Report format and location** section of the report generation dialog window. You must select two launches for comparison. You can create both workbench and HTML reports, and you can save both types of reports to the file system or open them locally.

The comparison algorithm in Code Coverage uses the older report for the baseline value and uses the newer report to calculate the difference between the two reports. Figure 32-12 is an example of a Code Coverage Comparison Report.

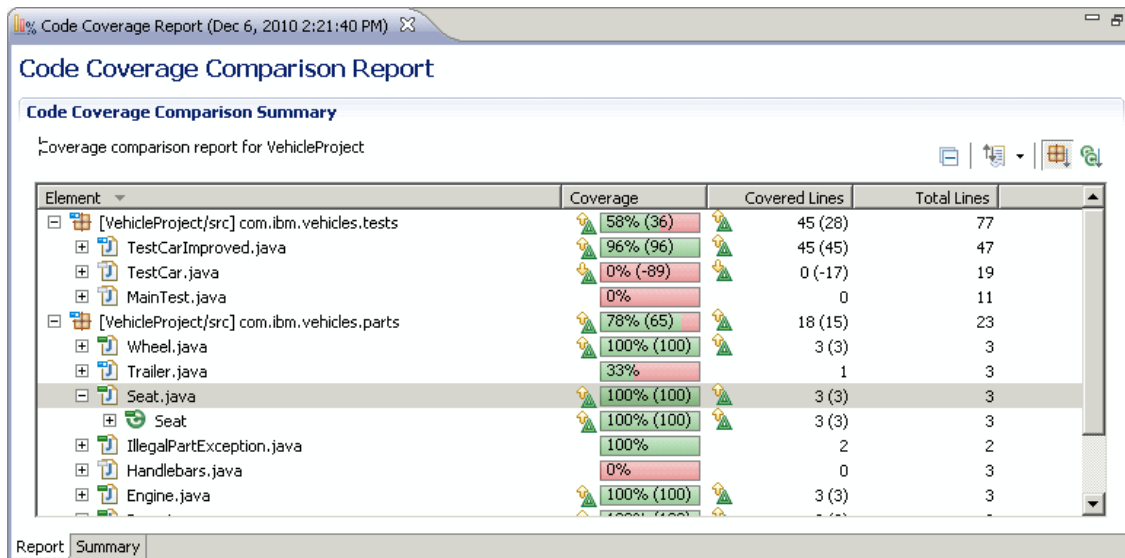


Figure 32-12 Example comparison report

The Code Coverage Comparison Report is an extension of the existing coverage report, and all of the same tools are available. For example, the actions in the context menu of the report still work as designed. However, in a comparison report, all nodes in the tree include information about the change in coverage information (if any). The major value for each node is the value for the oldest report. The value in parentheses represents the change in value when compared against the newer report. An icon represents the type of change (increase, decrease, new item, or removed item).

### 32.5.2 Generating coverage comparison report with Ant

You can generate coverage comparison reports in an Ant environment by using the Ant task that is provided by Code Coverage. This task uses the reporting functionality that is provided by the BIRT Eclipse.org project and thus requires that you download the latest Report Engine stand-alone offering. Navigate to <http://www.eclipse.org/birt/download>, select the latest release, and download the **Report Engine** offering. After downloading, extract the archive to a location on the file system.



Example 32-6 shows how comparison reports can be generated using Ant. This Ant task can only produce comparison HTML reports.

*Example 32-6 Usage of the report generation Ant task on the sample application*

---

```
<target name="define-report-task" description="Define the code coverage
comparison report generation task">
<path id="lib.path">
<pathelement location="<Rational Application Developer
home>\plugins\com.ibm.rational.11c.common_<date>.jar"/>
<pathelement location="<Rational Application Developer
home>\plugins\com.ibm.rational.11c.report_<date>"/>
<pathelement location="<Rational Application Developer
home>\plugins\org.eclipse.equinox.common_<date>.jar"/>
<fileset dir="<BIRT Report Engine home>/lib" includes="*.jar"/>
</path>
<taskdef name="code-coverage-comparison"
classname="com.ibm.rational.11c.report.birt.adapters.ant.ReportComparis
onTask" classpathref="lib.path"/>

<code-coverage-comparison
outputDir="VehicleProjectComparisonReport"
reportFiles="
VehicleProjectStatisticsOld.coveragedata;VehicleProjectStatisticsNew.co
veragedata"
baselineFiles="VehicleProjectBaselineNew.baseline;
VehicleProjectBaselineOld.baseline"/>
</target>
```

---

Example 32-7 on page 1718 shows the parameters for the comparison report generation Ant task.

Example 32-7 Input parameters for instrumentation tasks

Parameter	Description
outputDir	The output directory of the generated HTML comparison report.
reportFiles	The input coveredata files. Two coveredata files must be supplied. Separate the two paths by using the correct path separator character for the platform (a semicolon (;) on Microsoft Windows and a colon (:) on Linux)
baselineFiles	Optional: The input baseline files. Two coveredata files can be supplied. Separate the two paths by using the correct path separator character for the platform (a semicolon (;) on Microsoft Windows and a colon (:) on Linux).

## 32.6 Importing the coverage data statistics file

Code Coverage includes an import wizard, which allows users to import coveredata files into a Rational Application Developer workspace. This feature is particularly useful for users who generate statistics outside of the Rational Application Developer environment, because this feature enables them to import their statistics and analyze the results by using all of the available tooling in Rational Application Developer.

You invoke the import wizard by right-clicking in a Source Explorer view, selecting **Import**, and then selecting the **Code Coverage** → **Code Coverage Data File** option. The import wizard window opens and asks which type of import to perform. Figure 32-13 on page 1719 shows the data import type selection type. Use the “Data File is located on the file system” option to import coveredata files that are generated outside of the Rational Application Developer environment. The “Recent application launched in the workspace” option imports a coveredata file from a recent workspace launch.

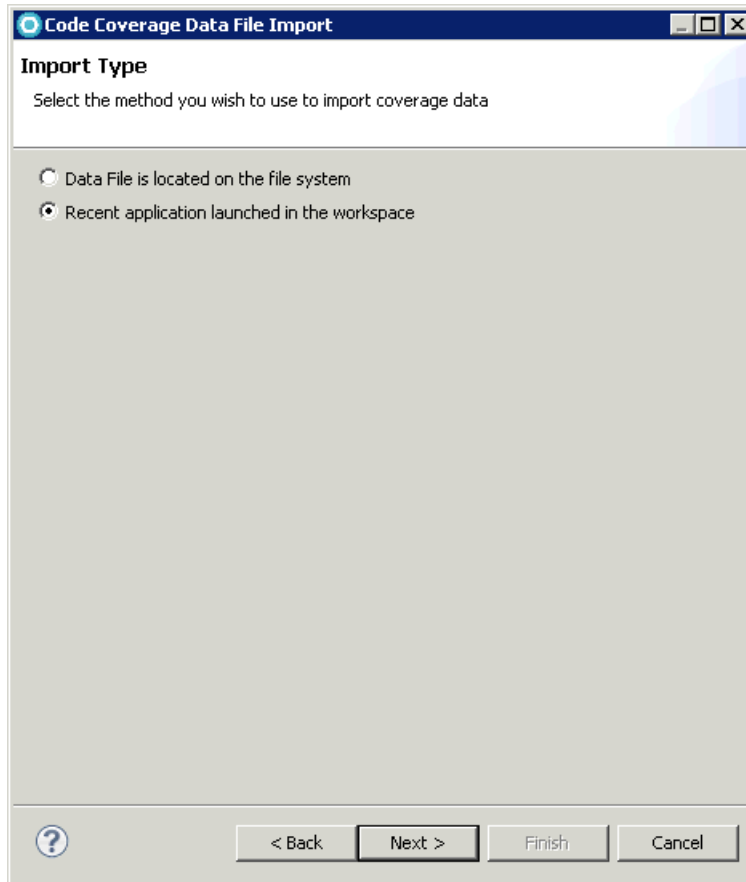


Figure 32-13 Code Coverage data file import wizard selection page

After the coveredata files have been imported into the workspace, you can select the coveredata files in the source view and analyze them using the Code Coverage actions from the context menu. Figure 32-14 on page 1720 shows the Code Coverage actions in the context menu.

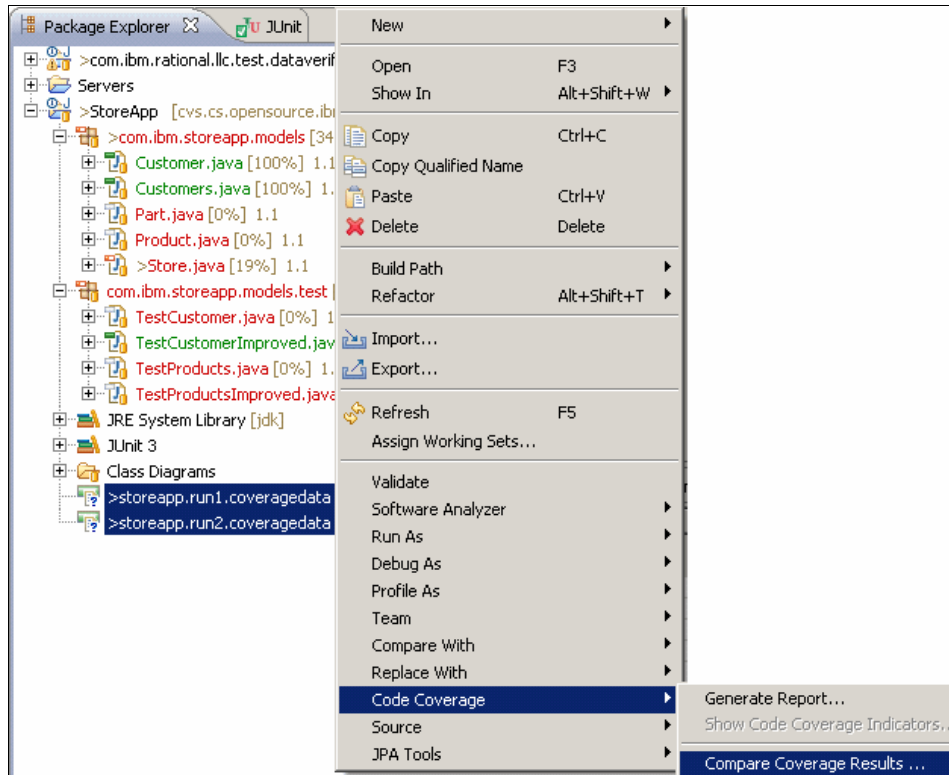


Figure 32-14 Code Coverage actions from the context menu of the coveragedata files

These three actions are available in the context menu of the coveragedata files:

- ▶ Use the “Generate Report” action to generate a coverage report from the selected coveragedata files.
- ▶ Use the “Show Code Coverage Indicators” action to decorate the local Java items with the statistics that are provided by the selected coveragedata file.
- ▶ Use the “Compare Coverage Results” action to generate a comparison report from the two selected coveragedata files. *This action is only enabled when exactly two files are selected.*

## 32.7 Generating statistics for web applications

You can generate Code Coverage statistics for any web application in Rational Application Developer. The tooling provides support for multiple application servers. You can view the generated Code Coverage statistics in near real time.

As the application executes on the server, the statistics are updated to reflect the new changes.

When a Code Coverage-enabled application executes on a server, the server refresh interval controls how often the data is refreshed in the workbench. The default refresh interval is set to 30 seconds, but you can update this interval by modifying the Refresh Interval value on the **Java** → **Code Coverage** → **Server** workbench Preferences page (see Figure 32-15).

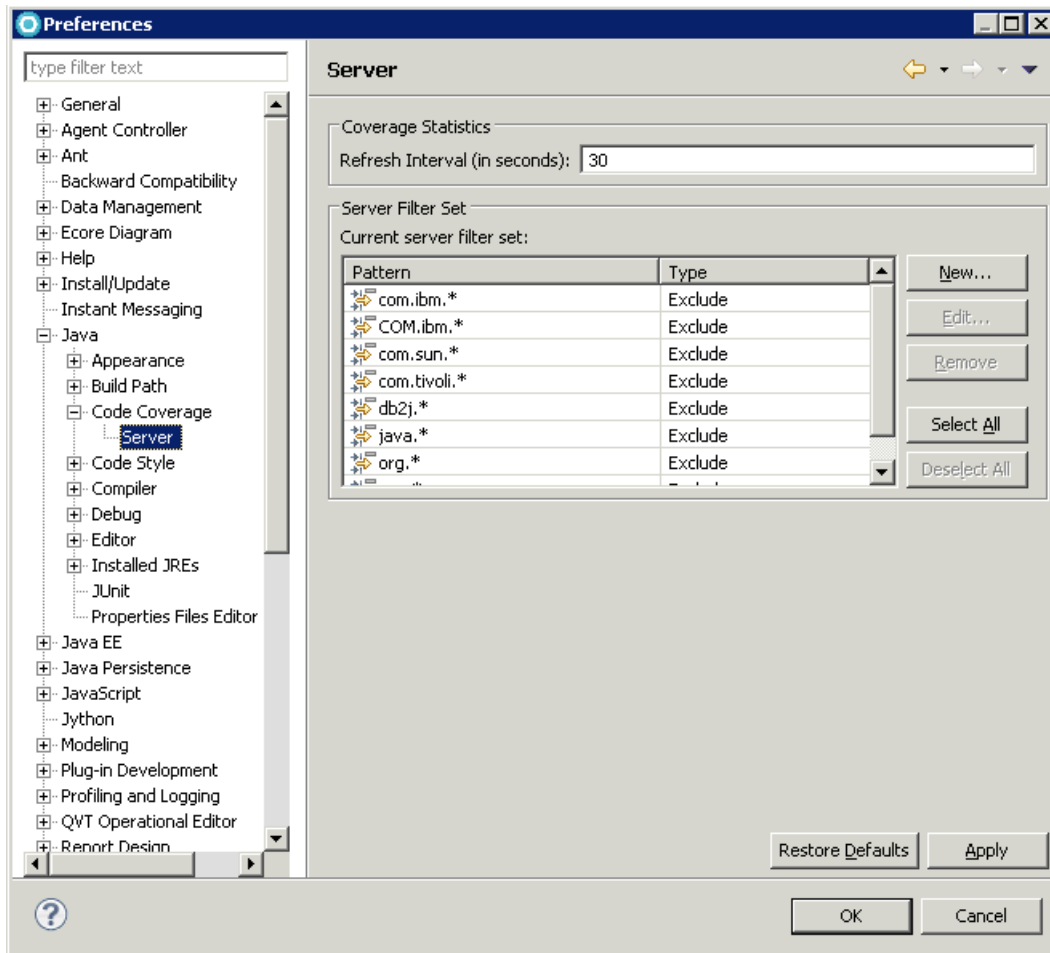


Figure 32-15 The Code Coverage Server Preferences page

You also can refresh the statistics manually by selecting the project folder in the source viewer and selecting the Refresh option from the context menu. Statistics for the project are updated only if the selected project is actively executing on a server.

Also on the Code Coverage Server Preferences page (Figure 32-15 on page 1721), you can use the Server Filter Set option to control the instrumentation of classes on the server. If data for a class that is not contained in your project is being collected, You can add exclude filter rules to this filter set. Likewise, if statistics for classes that are not in your project need to be collected, you can change this filter set to include the appropriate classes.

### 32.7.1 Support for WebSphere Application Server

To generate Code Coverage statistics for web applications running on an instance of WebSphere Application Server, follow these steps:

1. Enable Code Coverage on your web application (see 32.2, “Generating coverage statistics in Rational Application Developer” on page 1700).
2. Add the web application to the correct server. Right-click the server in the Servers view, select **Add and Remove**, and select the web application on the resulting wizard page.
3. Select **Run**, **Debug**, or **Profile** in the Servers view to start the server.

If the server instance was started in Profile mode, an additional wizard appears. You use this wizard to select the correct profiling data collector for the server. To collect Code Coverage statistics, select **Code Coverage Analysis** (see Figure 32-16 on page 1723).

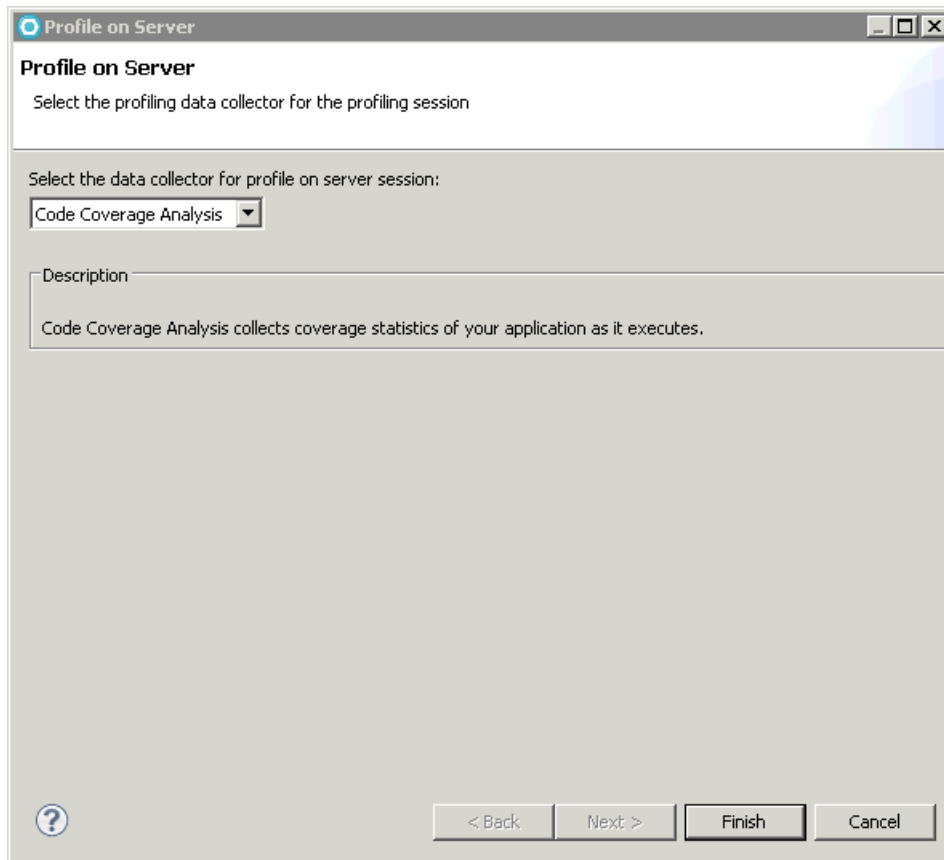


Figure 32-16 Additional Profile on Server wizard page to select the correct data collector

## 32.7.2 Generic application server support

Code Coverage also provides support for all other servers that are supported by Rational Application Developer (Tomcat, JBoss, and so on).

Before Code Coverage statistics can be generated, you must select the **Code Coverage Analysis** profiler on the Profilers page in the workbench Preferences. To open this page, select **Window** → **Preferences**, open the **Server** → **Profilers** page, and ensure that the **Code Coverage Analysis** profiler type is selected (see Figure 32-17 on page 1724).

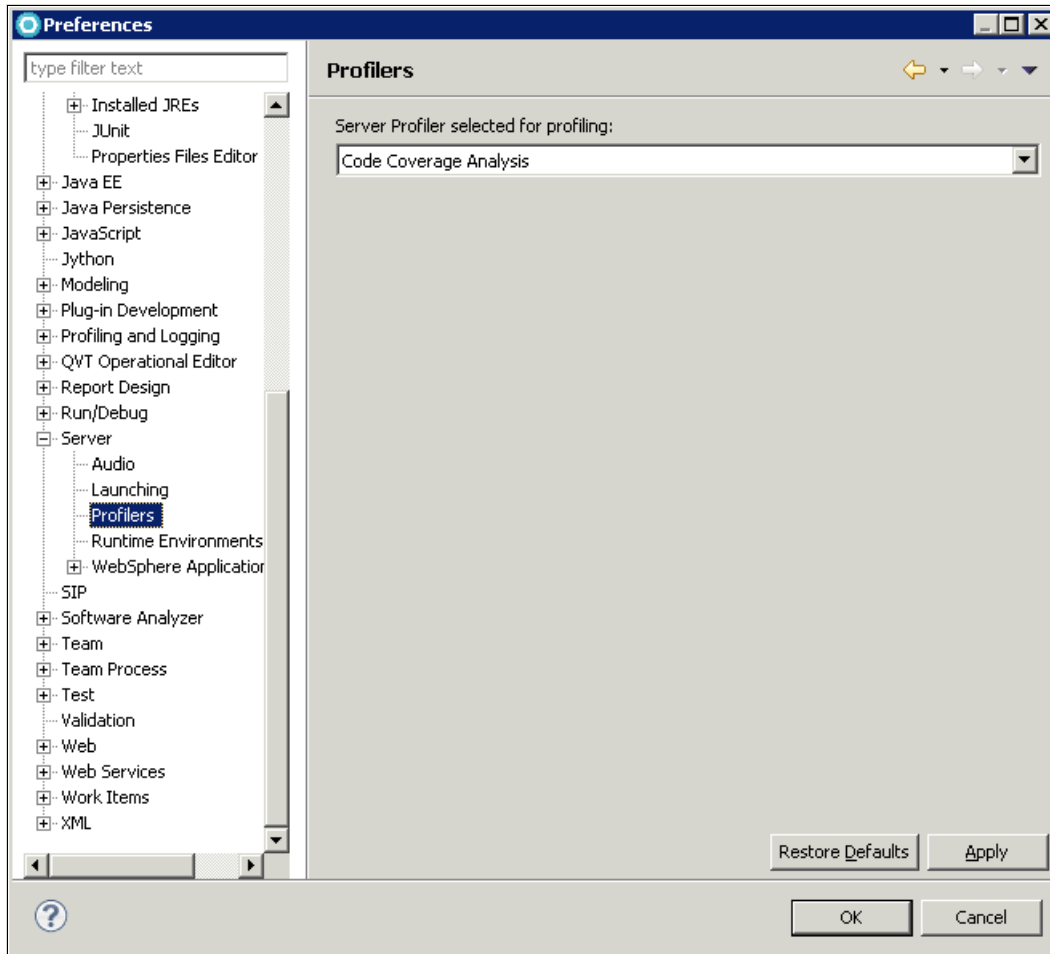


Figure 32-17 Server Profilers selection page for generic servers

To generate Code Coverage statistics for web applications on servers other than WebSphere Application Server, follow these instructions:

1. Enable Code Coverage on your web application (see 32.2, “Generating coverage statistics in Rational Application Developer” on page 1700).
2. Add the web application to the correct server. Right-click the server in the Servers view, select **Add and Remove**, and select the web application on the resulting wizard page.
3. Select **Profile** from the Servers view to start the server.



*Code Coverage statistics can only be generated in Profile mode for servers other than WebSphere Application Server.*

## **32.8 Rational Team Concert integration**

Code Coverage provides integration with the Rational Team Concert Build Toolkit to allow users to generate Code Coverage statistics from within their Rational Team Concert build. Users can then view and analyze the results within Rational Application Developer. See Chapter 30, “IBM Rational Application Developer integration with Rational Team Concert” on page 1595 for additional details.





# Developing Session Initiation Protocol applications

*Session Initiation Protocol (SIP)* is an application-layer protocol that you can use to initiate, modify, or terminate communication and collaborative sessions over Internet Protocol (IP) networks.

This chapter contains the following sections:

- ▶ Introduction to SIP
- ▶ Developing a SIP application
- ▶ Testing the SIP 1.1 application
- ▶ SIP-specific annotations in SIP 1.1 applications
- ▶ More information

## 33.1 Introduction to SIP

SIP sessions consist of various objects:

- ▶ IP telephony calls
- ▶ Multiuser conferences (incorporating voice, video, and data)
- ▶ Instant messaging chats
- ▶ Multiple player online games

You can use SIP to invite participants (persons, automated services, or physical devices) to a scheduled or existing session and to add media to or remove media from a session.

### 33.1.1 SIP 1.1 specification

The SIP 1.1 specification is defined in *Java Specification Request (JSR) 289: SIP Servlet Specification, Version 1.1*. It has the following objectives:

- ▶ Clarify the intentions of *JSR 116: SIP Servlet application programming interface (API), Version 1.0*
- ▶ Standardize many industry practices that have matured around SIP servlet applications.
- ▶ Enable more and better interconnected SIP servlet-based applications, such as applications that incorporate SIP and other Java Platform, Enterprise Edition (Java EE) components (for example, HTTP servlets and Enterprise JavaBeans (EJB)).
- ▶ Provide an application routing mechanism for composing SIP services into application groups.

Both the SIP 1.1 and SIP 1.0 specifications are based on the Java servlet API. SIP servlets are Java-based component applications that typically run in servlet containers on network servers. The SIP servlet and container hide SIP protocol complexities by providing an environment where services cannot violate the protocol or perform restricted operations.

The SIP container performs many functions that simplify creating SIP applications. The servlet container manages the life cycle of a servlet and supports interaction between servlets and SIP clients (User Agents (UA)) by exchanging SIP request and response messages. For instance, a servlet container can perform message queuing, dispatching, and state management.

As with HTTP servlets, SIP servlets extend the base `javax.servlet.GenericServlet` class. The `SipServletRequest` and

SipServletResponse classes are similar to the HttpServletRequest and HttpServletResponse classes. However, there are important differences:

- ▶ SIP applications can perform intelligent request routing, act as proxy requests, and initiate requests.
- ▶ SIP is asynchronous and there can be multiple responses to the same request.

All messages come in through the service method, which calls doRequest for incoming requests or doResponse for incoming responses. Depending on the request method (Table 33-1) or status code (Table 33-2), the call is dispatched.

*Table 33-1 SIP servlet request methods*

<b>Servlet method</b>	<b>SIP request attended</b>
doInvite	INVITE
doAck	ACK
doOptions	OPTIONS
doBye	BYE
doCancel	CANCEL
doRegister	REGISTER
doSubscribe	SUBSCRIBE
doNotify	NOTIFY
doMessage	MESSAGE
doInfo	INFO
doPrack	PRACK
doUpdate	UPDATE
doRefer	REFER
doPublish	PUBLISH

*Table 33-2 SIP servlet response methods*

<b>Servlet methods</b>	<b>SIP responses</b>	<b>Meaning</b>
doProvisionalResponse	1xx	Provisional messages
doSuccessResponse	2xx	Success answers
doRedirectResponse	3xx	Redirection answers

Servlet methods	SIP responses	Meaning
doErrorResponse	4xx	Method failures
	5xx	Server failures
	6xx	Global failures

Table 33-3 lists the servlet classes and interfaces in the SIP Servlet Specification 1.1. These objects provide high-level abstraction of many of the SIP concepts.

*Table 33-3 SIP servlet classes and interfaces*

Class/Interface	Description
SipServlet	The base servlet object, it receives incoming messages through the service method, which calls doRequest or doResponse.
ServletConfig	Used by the servlet container to pass configuration information to a servlet during initialization.
ServletContext	Used by a servlet to communicate with its container.
SipServletMessage	Defines common aspects of SIP requests and responses.
SipServletRequest	Provides high-level access to SIP request messages. Created and passed to the handling servlet when the container processes incoming requests.
SipServletResponse	Provides high-level access to a SIP response message. Instances of SipServletResponse are passed to servlets when the container receives incoming SIP responses.
SipFactory	Factory interface for a variety of servlet API abstractions.
SipAddress	Represents the SIP From and To header.
SipSession	Represents SIP point-to-point relationships and maintains dialog state for UAs.
SipApplicationSession	Represents application instances, acts as a store for application data, and provides access to contained protocol sessions.
Proxy	Represents the operation of proxying a SIP request and provides control over how that proxying is carried out.
SipApplicationRouter	Application router interface.
SipSessionsUtil	Utility class that provides additional support for session management for converged applications.
ConvergedHttpSession	Extension to HttpSession for converged applications.

Class/Interface	Description
B2buaHelper	Utility class with support for Back-to-Back User Agent (B2BUA) applications. B2BUA is a logical entity that receives a request as a User Agent Server and processes the request as a User Agent Client, handling the signaling between both endpoints.
SipURI	Interface that represents SIP and SIPS URIs as defined in Request for Comments (RFC) 2396.
TelURL	Interface that represents telephone numbers.
TimerService	Interface that allows SIP servlet applications to set timers to receive timer expiration notifications.

### 33.1.2 Converged SIP applications

A *converged SIP application* is an application that uses both HTTP Servlet API and Java EE components. Depending on the components used, a converged SIP application can be either of the following combinations:

- ▶ SIP and HTTP converged applications, hosting SIP and HTTP servlets
- ▶ SIP and Java EE converged applications, hosting SIP, HTTP, and Java EE (such as EJB, web services, messaging, and so on) components

Several classes are provided to facilitate the development of converged applications, including these classes:

- ▶ Use the `SipFactory` class to create requests, address objects, or application sessions. You can access it through `ServletContext` or dependency injection (`@Resource`).
- ▶ Use the `SipApplicationSession` class to store application data to correlate a number of protocol sessions. You can create it through `ConvergedHttpSession` (recommended) or `SipFactory`. You also can access it through `SipSessionsUtil` or `SipSession`.
- ▶ Use the `SipSessionsUtil` class to provide a way to access `SipApplicationSession` by ID.

### 33.1.3 SIP 1.1 annotations

With SIP 1.1, you can use annotations for SIP servlet applications, which is a convenient way to develop applications.

You can use the annotations to perform these tasks:

- ▶ Embed data directly into an application instead of using the deployment descriptor
- ▶ Inject resources, such as EJB or SIP utility classes, into an application

The following annotations are the @Sip annotations and properties (see Table 33-4):

- ▶ The @SipApplication annotation maintains the application-level configuration that used to be part of the deployment descriptor. It is a package-level annotation (must be located in a package-info.java file), and all servlets within the package belong to the same application.

**Important:** Only one SIP application can be registered with the container for each .war or .sar archive, regardless of whether you use a deployment descriptor or annotations.

- ▶ The @SipServlet annotation indicates that a class is a SIP servlet.
- ▶ The @SipListener annotation indicates that a class is a SIP listener.
- ▶ The @SipApplicationKey annotation marks the method that associates an incoming request and SipSession with a specific SipApplicationSession. You use this annotation with session key-based targeting.

Table 33-4 Annotation properties

Annotation property	Data definition (DD) replaced element	Description	Default value
<b>@SipApplication</b>			
Name	<app-name>	Define the SIP application name	N/A (It is mandatory to fill this field)
displayName	<display-name>	Displayed name of the application	Application name value
smallIcon	<small-icon>	Path with the location of the small icon	Empty string
largeIcon	<large-icon>	Path with the location of the large icon	Empty string
description	<description>	Explains the application and its function	Empty string



Annotation property	Data definition (DD) replaced element	Description	Default value
distributable	< <i>distributable</i> >	Indicates if the application can function in a distributed environment (true) or not (false)	false (Boolean value)
proxyTimeout	< <i>proxy-timeout</i> >	Default timeout for all proxy operations	3 minutes
sessionTimeout	< <i>session-timeout</i> >	Default timeout (in whole minutes) for all application session operations	3 minutes
mainServlet	< <i>main-servlet</i> >	Indicates the SIP servlet that is designed as the Main Servlet	Empty string
<b>@SipServlet</b>			
Name	< <i>servlet-name</i> >	ID used to reference the servlet in the context	Short name of the annotated class
applicationName	N/A	Application name with which the annotated servlet is associated	Application Name (checks in DD and package annotation)
description	< <i>description</i> >	Declarative data about the servlet	Empty string
displayName	< <i>display-name</i> >	Displayed name of the application	Application name value
smallIcon	< <i>small-icon</i> >	Path with the location of the small icon	Empty string
largeIcon	< <i>large-icon</i> >	Path with the location of the large icon	Empty string
description	< <i>description</i> >	Explains the application and its function	Empty string

### 33.1.4 SIP application packaging

A SIP project has the same directory structure as a web project. The project has a WEB-INF subdirectory that contains these objects:

- ▶ The deployment descriptor files `sip.xml` and `web.xml`

- ▶ Utility classes and jar files in their respective classes and lib directories

A converged SIP application that is created for the purpose of deployment from the integrated development environment (IDE) is, by default, packaged as a WAR in an EAR file.

**Packaging of WARs:** The SIP tooling only supports the packaging of WARs in EARs. The packaging of SARs in EARs is not available even though it is supported in the SIP specification.

You also can import or export converged SIP/HTTP applications as stand-alone SARs or WARs for exchange between developers and deployment directly on the WebSphere Application Server administrative console.

## 33.2 Developing a SIP application

In this section, we describe the wizards and editors in Rational Application Developer that you can use to develop converged SIP applications, including wizards and editors to facilitate the creation and editing of SIP applications. We start with an overview of the tools followed by an example that demonstrates creating, editing, and deploying a SIP 1.1 application.

**Version level:** The IDE supports developing both SIP 1.0 and SIP 1.1 applications. In this chapter, if the version is not specified, assume that the content applies to both SIP versions.

This section describes the following topics:

- ▶ SIP tooling overview
- ▶ Sample application overview
- ▶ Setting up the project
- ▶ Implementing the classes
- ▶ SIP deployment descriptor
- ▶ Preparing for deployment
- ▶ Deploying SIP from Rational Application Developer

## 33.2.1 SIP tooling overview

Rational Application Developer provides wizards and editors that you can use to develop SIP applications easily. Using these tools provides the following benefits:

- ▶ Class path configuration of the SIP project is automatic so that the correct SIP API jar is set.
- ▶ You can generate deployment configuration from the new SIP servlet wizard.
- ▶ You can generate template code when creating new SIP servlets.
- ▶ Content assist support for SIP annotations is available in the Java editor.
- ▶ You can use graphical or source-based editing of the deployment information with the SIP Deployment Descriptor Editor.
- ▶ Automatic or manual merging options for SIP and web deployment descriptors are available to ensure consistent content for the deployment.
- ▶ Validators that identify an incorrect deployment configuration exist.

### Supported run times

Table 33-5 show the supported run times for developing SIP applications in Rational Application Developer.

Table 33-5 Supported run times for SIP development in Rational Application Developer

Run time	SIP 1.1	SIP 1.0
WebSphere Application Server V7.0.0.13 Base	No	Yes
WebSphere Application Server V7.0.0.13 with Communications Enabled Applications (CEA) Feature Pack V1.0.0.7	Yes	Yes
WebSphere Application Server v8.0 Beta Base	Yes	Yes

### New SIP project wizard

To create a new SIP project, perform the following steps:

1. In Rational Application Developer, click **File** → **New** → **Other** → **SIP Project**. Figure 33-1 on page 1736 shows the first page of the wizard when creating a SIP 1.1 project.

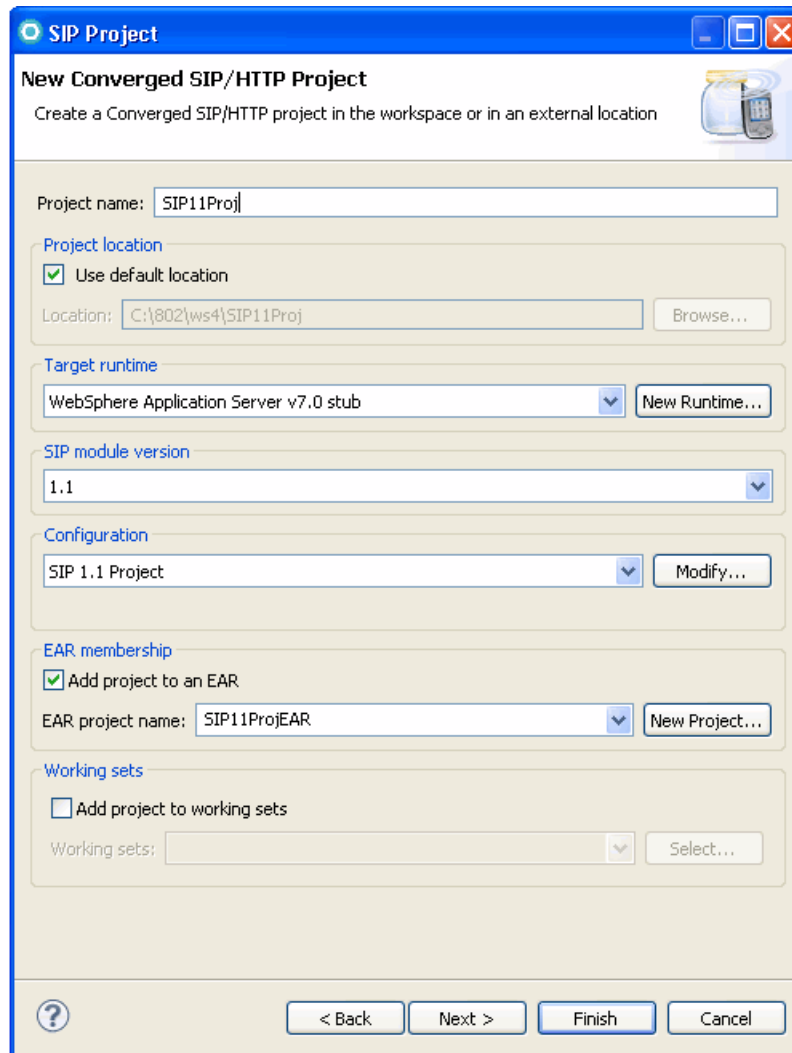


Figure 33-1 New SIP Project wizard

- In the Configuration field, select either **SIP 1.1 Project** or **SIP 1.0 Project**. The configuration sets the facets for the project. Table 33-6 shows the project facets that are set.

Table 33-6 SIP project facets and versions

Facets	SIP 1.1	SIP 1.0
Dynamic Web Module	2.5	2.3

Facets	SIP 1.1	SIP 1.0
Java	1.5	1.5
JavaScript	1.0	1.0
SIP Module	1.1	1.0
WebSphere Web (Coexistence)	7.0	6.1
WebSphere Web (Extended)	7.0	6.1

**Important:** Rational Application Developer does not support the migration of SIP 1.0 projects to SIP 1.1. If you want to migrate, you must create a new SIP 1.1 project, copy over the files, and update their contents.

### SIP servlet wizard

To create a new SIP servlet, run the SIP Servlet wizard by clicking **File** → **New** → **Other** → **SIP Servlet**. The SIP project version that you select determines the pages that are displayed in the wizard. For example, if you select a SIP 1.1 project, the wizard displays SIP 1.1-specific configurations in the second and third pages of the wizard.

Figure 33-2 on page 1738 shows the second page of this wizard, which has options that are specific to creating a SIP 1.1 servlet. The second page of the SIP servlet wizard is used to configure the initialization parameters and servlet selection. When you complete the wizard, the `sip.xml` file is updated with these settings. The red highlights show configurations that are specific to SIP 1.1 that do not appear when creating a new servlet on a SIP 1.0 project.

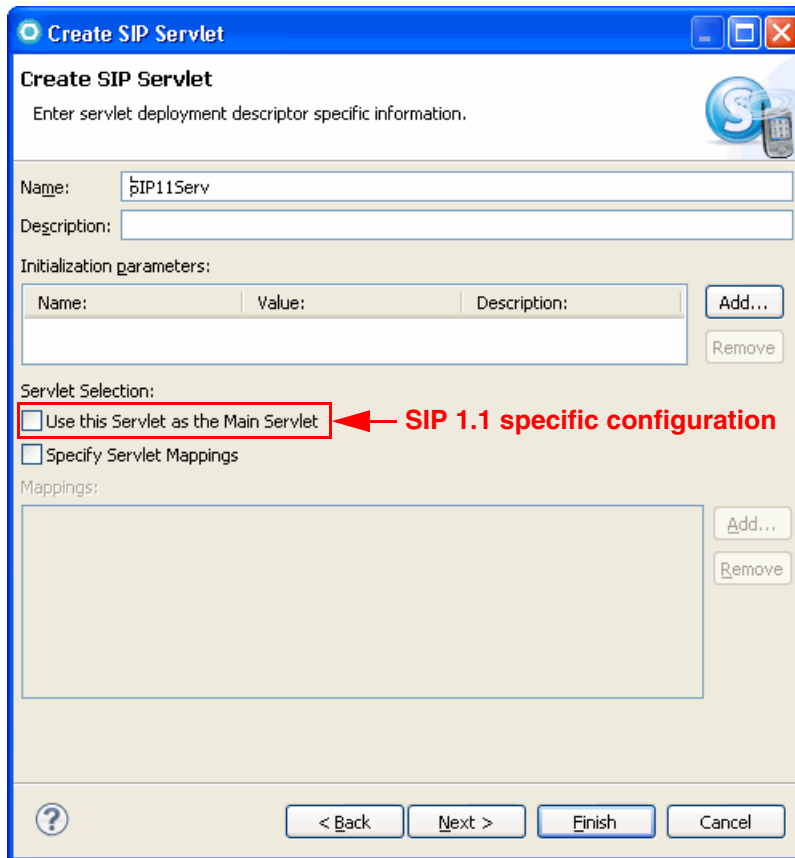


Figure 33-2 Second page of the new SIP project wizard

On page three of the wizard (see Figure 33-3 on page 1739), you select the method stubs to generate in your SIP servlet. The method options that are highlighted in red are specific to SIP 1.1 servlets and are disabled when creating a SIP 1.0-based servlet.

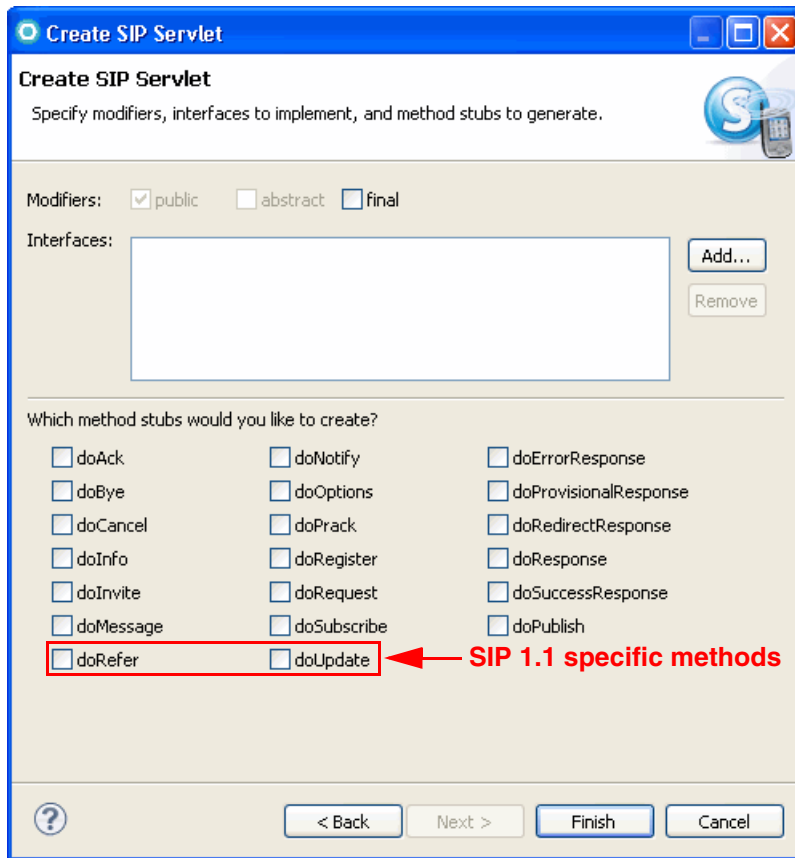


Figure 33-3 Third page of the new SIP project wizard

### SIP 1.1 deployment descriptor editor

You can use the SIP 1.1 deployment descriptor editor to edit the SIP deployment information in either a graphical-based view or an XML source view. To open this editor, double-click either the SIP Deployment Descriptor node or the WEB-INF/sip.xml file (Figure 33-4 on page 1740).

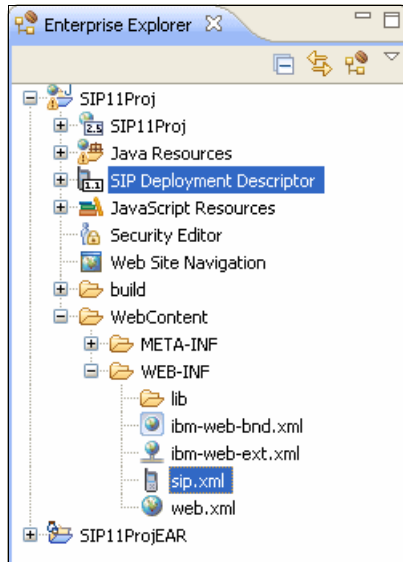


Figure 33-4 SIP 1.1 Deployment Descriptor access points

The layout and functionality of the editor are similar to the other Java EE 5 deployment descriptor editors. Figure 33-5 on page 1741 shows an example of the SIP 1.1 deployment descriptor.



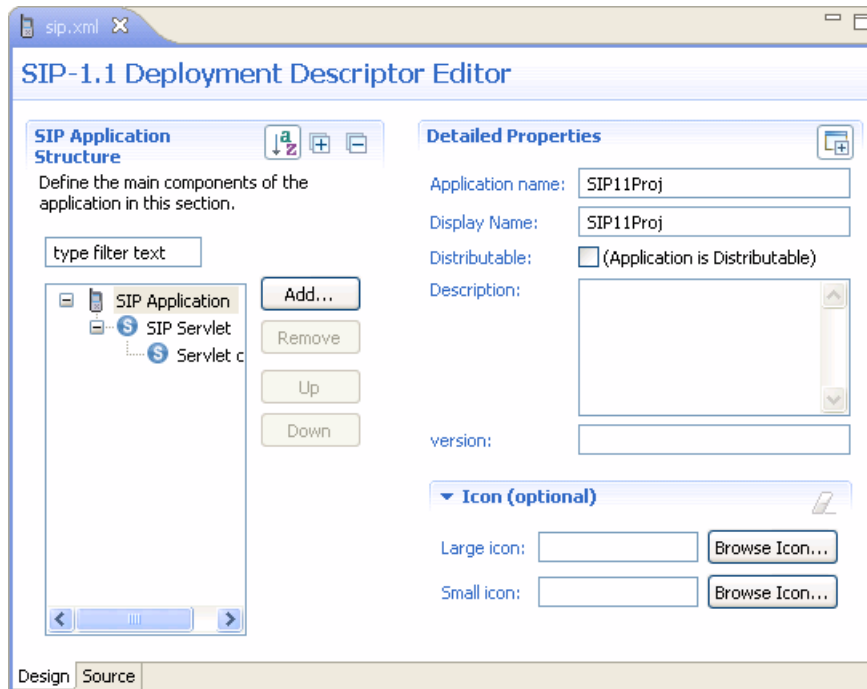


Figure 33-5 SIP 1.1 Deployment Descriptor

## Editing SIP 1.1 annotations

SIP annotation support is available through content assist (Ctrl+Spacebar) when you edit in the Java Editor. Figure 33-6 shows an example of content assist support for the @SipServlet annotation.

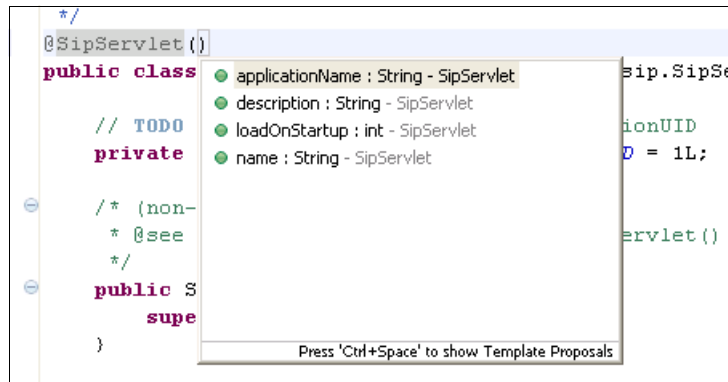


Figure 33-6 Content assist support for SIP annotations

## Merging SIP and web deployment descriptor contents

To deploy a converged SIP application from Rational Application Developer, you must merge the contents of the `sip.xml` file into the `web.xml` file. The contents are merged so that components, such as SIP servlets, are recognized and processed in the context of a web application. By default, every time that you modify the `sip.xml` file, you are prompted to automatically merge the content. To change the merge preferences, click **Windows** → **Preferences** → **SIP** (Figure 33-7).

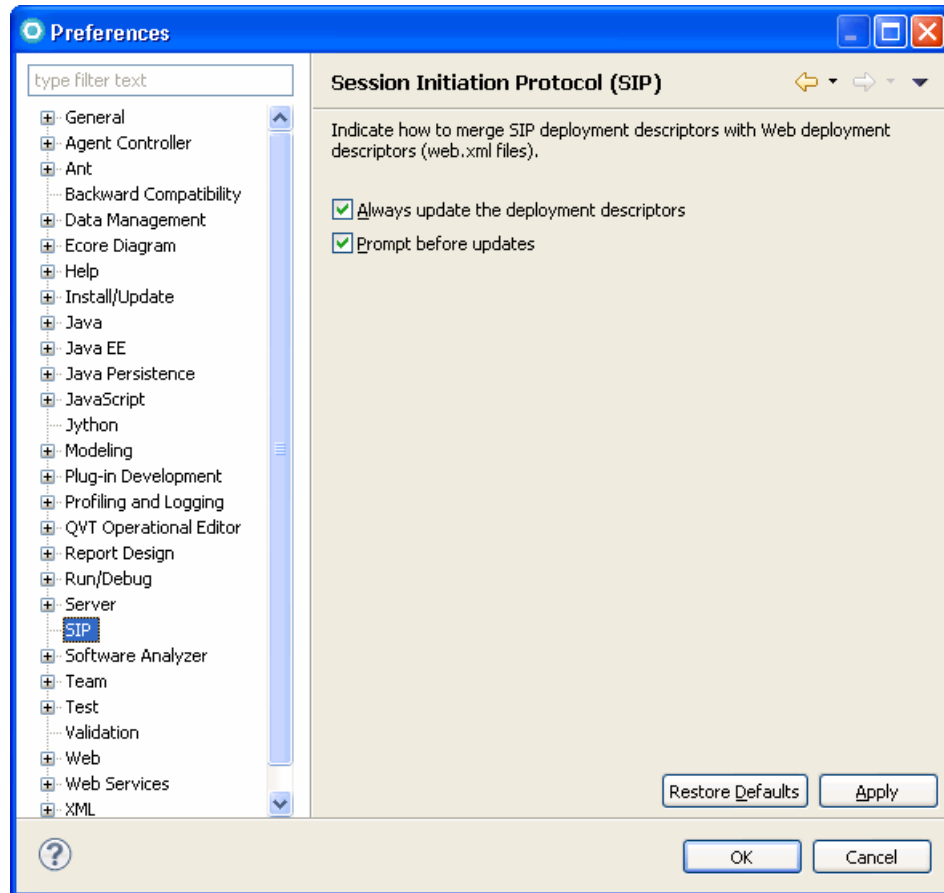


Figure 33-7 SIP merge Preferences page

You can choose to bypass automatic merging of the deployment descriptor if you are actively modifying the `sip.xml` and do not want constant updates to the `web.xml`. In this case, a manual merge option is available in the context menu **SIP Operations** → **update web.xml** (Figure 33-8 on page 1743).

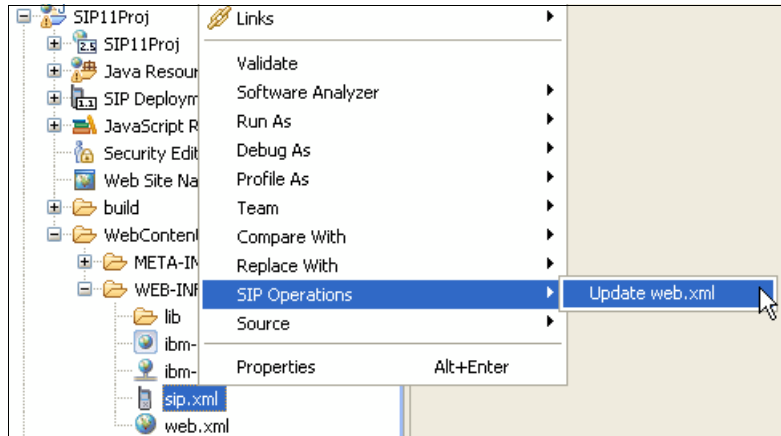


Figure 33-8 Invoking the manual SIP merge operation

## Validation support

SIP applications must constantly check the consistency of deployment information because the deployment configurations can appear in many places: the `sip.xml`, `web.xml`, and for SIP 1.1, annotations. Inconsistent or incorrect deployment information can cause errors when you deploy or run the application.

Validation support checks the following information:

- ▶ Verification of `sip.xml` and `web.xml` according to section 19.2 of the *Java Specification Request (JSR) 289: SIP Servlet Specification, Version 1.1*. This validation is common to SIP 1.0 applications as well and involves verifying the consistent usage between the two deployment descriptors for the following elements:
  - `Distributable`: This setting must be identical if present in both `sip.xml` and `web.xml`.
  - `Context-param`: If parameters are configured in both deployment descriptors, they must have the same values.
  - `Display-name` and `icons`: These values must be identical if they are present in both `sip.xml` and `web.xml`.
- ▶ For SIP 1.1 applications, validation support checks that either a main servlet or servlet mapping is used, but not both.
- ▶ For SIP 1.1 applications, validation support checks that valid servlet names are used in referenced configurations.

To view the list of SIP validators, click **Windows** → **Preferences** → **Validators** (Figure 33-9 on page 1744).

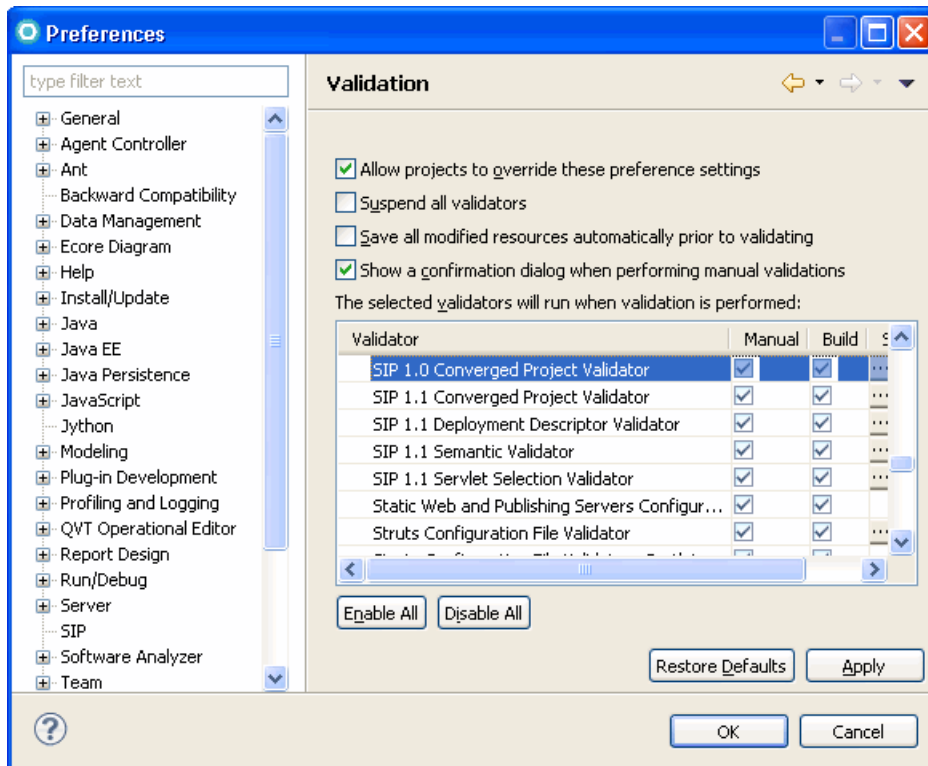


Figure 33-9 SIP validators

These validators are enabled by default but you can disable them for automatic or manual builds.

The *SIP 1.0 and SIP 1.1 Converged Project Validators* verify that the distributable, context-param, display-name, and icon elements are used in their respective SIP project versions. Applications that fail this type of validation have error markers in the Problems view, as shown in Figure 33-10.

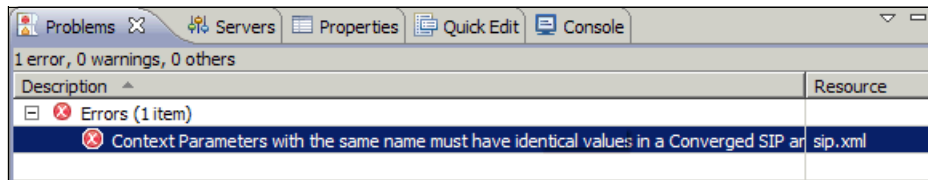


Figure 33-10 Example of a context-param error

The *SIP 1.1 Servlet Selection Validator* checks that only one type of servlet selection is used within a SIP application. Errors are flagged if these conditions occur:

- ▶ Multiple Main Servlets are specified in the sip.xml.
- ▶ A Main Servlet *and* a Servlet Mapping are specified in the sip.xml.
- ▶ A Main Servlet is specified in the @SipApplication annotation *and* in the sip.xml.
- ▶ A Main Servlet is specified in the @SipApplication annotation *and* a Servlet Mapping is configured in the sip.xml.

Figure 33-11 and Figure 33-12 show the error markers in the SIP 1.1 deployment descriptor editor and Problems view.

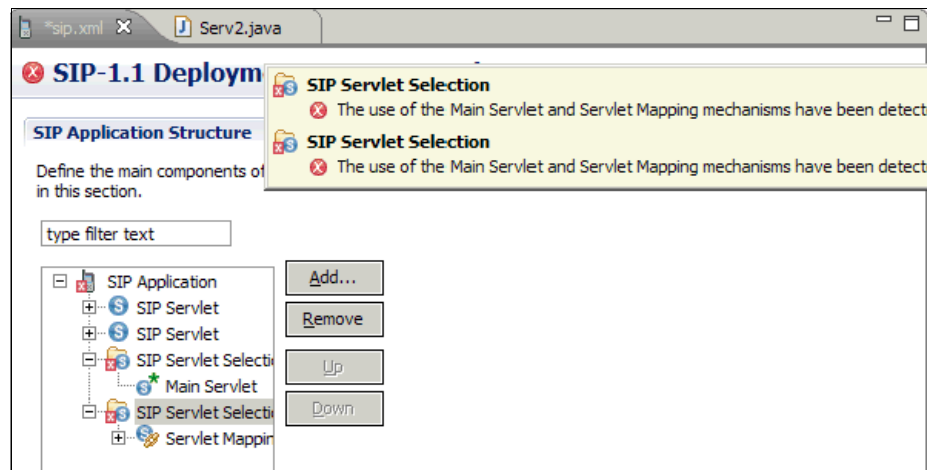


Figure 33-11 Example of multiple servlet selection errors in the SIP 1.1 deployment descriptor editor

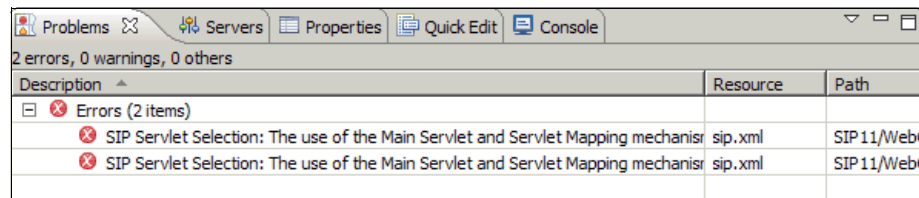


Figure 33-12 Example of multiple servlet selection errors in the Problems view

The *SIP 1.1 Semantic Validator* identifies errors when configurations that reference invalid servlet names are specified in the deployment descriptor.

The following servlet names cause errors if invalid:

- ▶ `<servlet-selection>/<main-servlet>`
- ▶ `<servlet-selection>/<servlet-mapping>/<servlet-name>`
- ▶ `<security-constraint>/<resource-collection>/<servlet-name>`

Figure 33-13 shows an example of an invalid server name error.

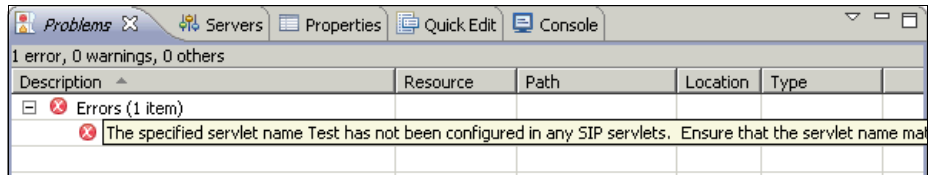


Figure 33-13 Example of an invalid servlet name error

## 33.2.2 Sample application overview

In this section, we create a HTTP/SIP converged application that is targeted for deployment on IBM WebSphere Application Server v8.0 Beta.

This sample demonstrates an implementation of a common feature that is found in public switched telephone networks, which is called the *call forwarding* functionality. This application has two components:

- ▶ HTTP client: The HTTP client is a user interface to input call forwarding information, which is then set in an access control list.
- ▶ SIP servlet: This servlet is responsible for looking up the forwarding address in the access control list (ACL) and forwarding the call to this address.

The sample also shows how to use these functions:

- ▶ Resource injection through the annotation `@Resource` for the `SipFactory` utility class
- ▶ The `MainServlet` selection, which dispatches the request

Figure 33-14 on page 1747 shows the classes in the elements.

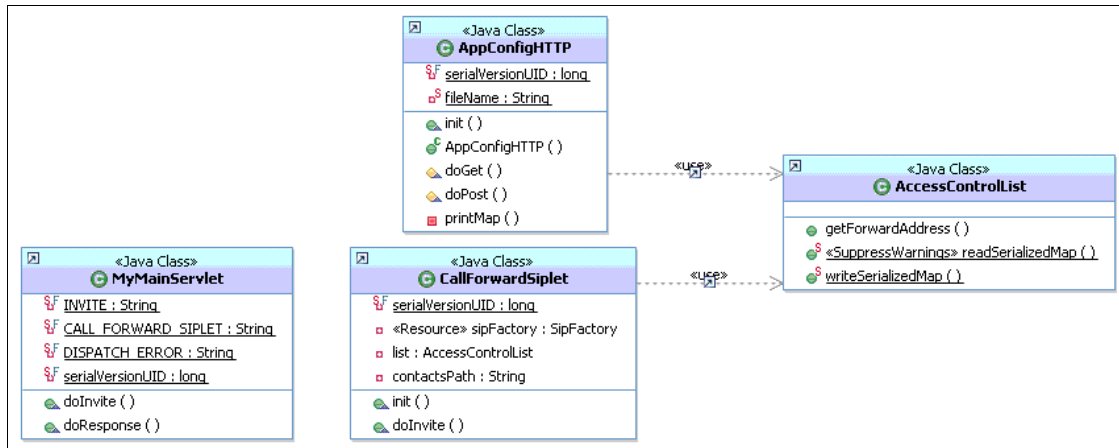


Figure 33-14 Call forwarding sample class diagram

The following elements are in the class diagram that is shown in Figure 33-14:

- ▶ AppConfigHTTP, HttpServlet provides a user interface (UI) to configure the forwarded call address mapping.
- ▶ AccessControlList stores the addresses mapping serializing the Map instance in a file located in the server.
- ▶ MyMainServlet is a SIP servlet that receives the requests and dispatches them to the CallForwardSiplet if it is marked initial and the method is INVITE.
- ▶ CallForwardSiplet is a SIP servlet that looks up the forwarding SIP address in the AccessControlList by using the public SIP address as the key. If the key is found, the request is then redirected to the associated forwarding address.

We deployed the sample by using the following configuration of computers (Figure 33-15 on page 1748).

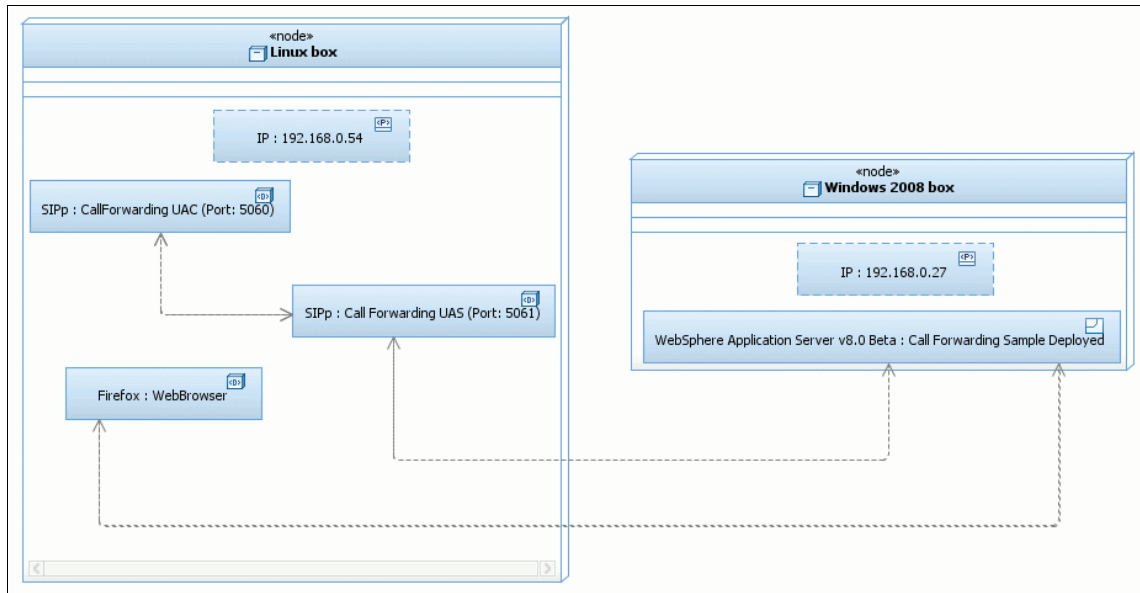


Figure 33-15 Deployment environment

We used these systems:

- ▶ A computer with a Linux operating system and the following software:
  - Firefox web browser (although you can use any browser)
  - SIPp 3.1 installed
  - XML files for the test scenarios:
    - UAC: call\_forwarding\_uac.xml
    - UAS: call\_forwarding\_uas.xml
- ▶ A computer with the Microsoft Windows Server 2008 operating system and WebSphere Application Server v8.0 Beta installed

Figure 33-16 on page 1749 shows the call flow for the CallForwarding sample.



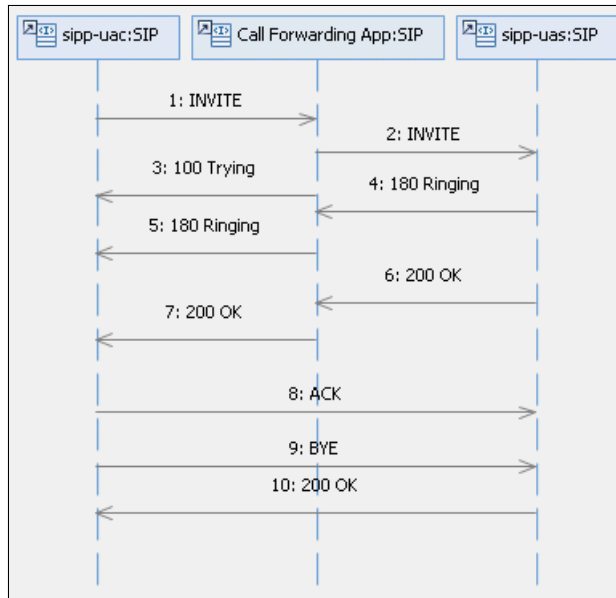


Figure 33-16 Call forwarding sample class diagram

The call flow process consists of these steps:

1. Receives an INVITE request to the UAS public SIP address.
2. Redirects the INVITE request to the forwarding SIP address for the sipp-UAS that is associated with the public SIP address.
3. Sends a Trying (100) response status to sipp-UAC to avoid retransmission for the INVITE request.
4. Receives the Ringing (180) response from sipp-UAS.
5. Resends the Ringing (180) response to the sipp-UAC.
6. Receives the OK (200) response from sipp-UAS.
7. Resends the OK (200) response to the sipp-UAC.
8. At this point, the subsequent requests and responses are between sipp-UAC and sipp-UAS directly. In detail, sipp-UAC sends an ACK request for the OK(200) response to sipp-UAS.
9. At this point, the call takes place:
  - a. sipp-UAC sends a BYE request to the sipp-UAS.
  - b. sipp-UAS sends an OK (200) response for the BYE request to sipp-UAC.

### 33.2.3 Setting up the project

This section describes the following actions:

- ▶ Creating a new SIP 1.1 project
- ▶ Creating the HTTPServlet
- ▶ Creating the MyMainServlet Sip Servlet
- ▶ Creating the CallForwardSiplet Sip Servlet

#### **Creating a new SIP 1.1 project**

Follow these steps:

1. Click **File** → **New** → **Other** → **SIP Project**.
2. Click **Next**.
3. Type the project names (Figure 33-17 on page 1751):
  - a. For Project name, type CallForwardingSample.
  - b. For EAR project name, type CallForwardingSampleEAR.
  - c. Click **Finish**.

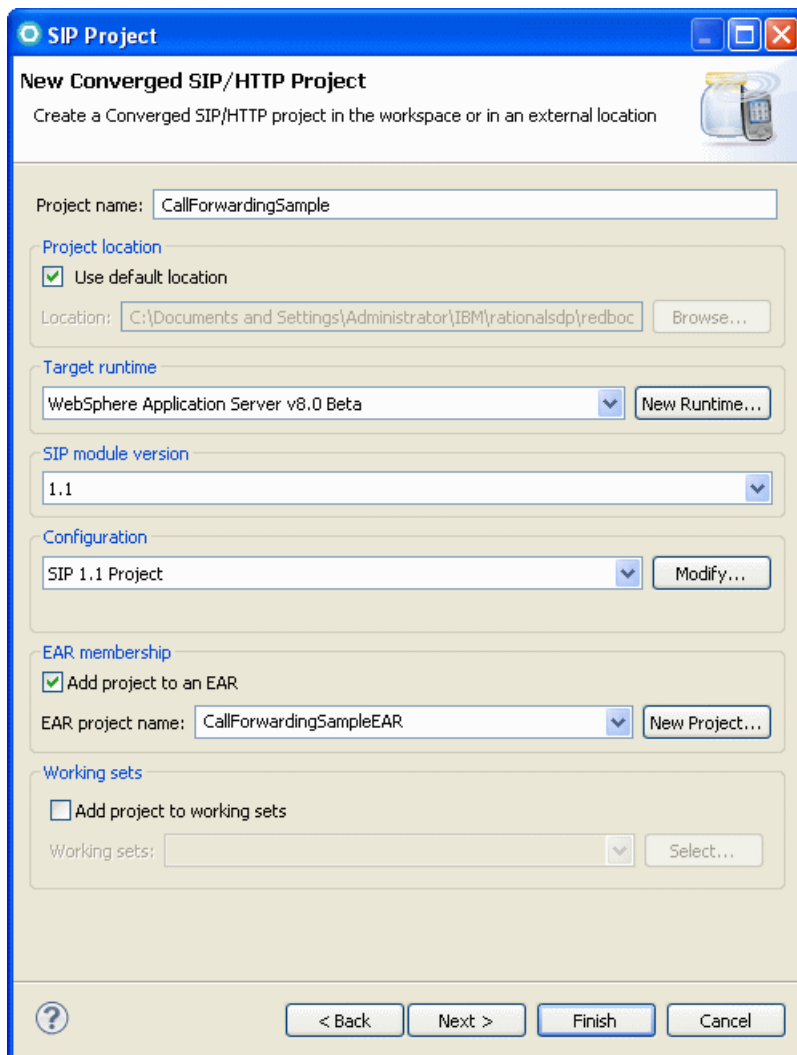


Figure 33-17 SIP 1.1 project creation

## Creating the HTTPServlet

Follow these steps:

1. Click **File** → **New** → **Other** → **Web** → **Servlet**.
2. Click **Next**.
3. Complete the fields in the wizard (Figure 33-18 on page 1752):
  - a. For Java package, type `com.ibm.siptools.samples`.

b. For Class name, type AppConfigHTTP.

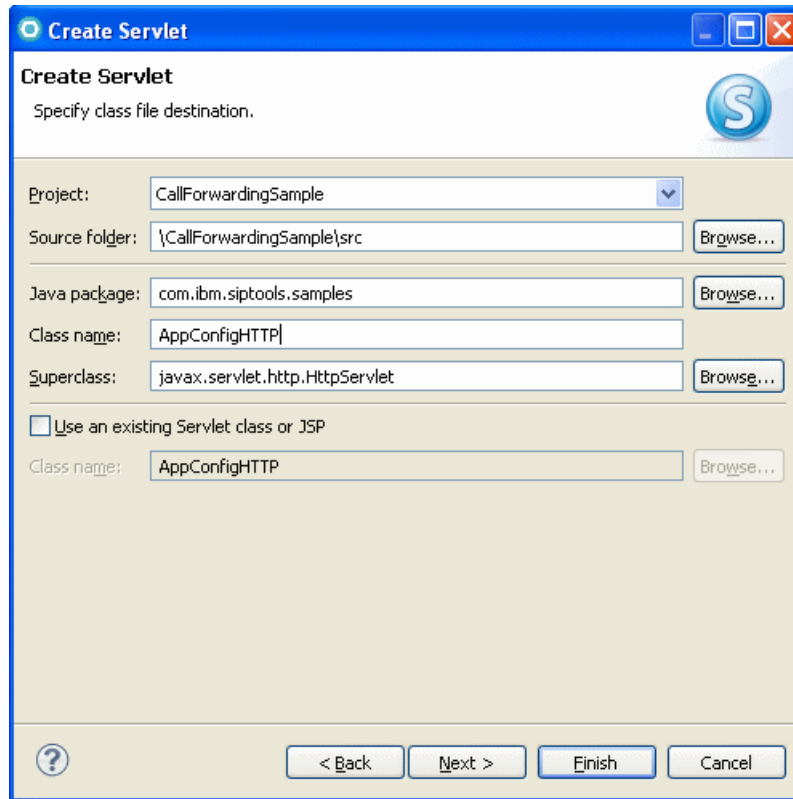


Figure 33-18 AppConfigHTTP servlet creation wizard (page 1)

4. Click **Next**.
5. Click **Next**.
6. Select the required methods (Figure 33-19 on page 1753):
  - **init**
  - **doGet**
  - **doPost**

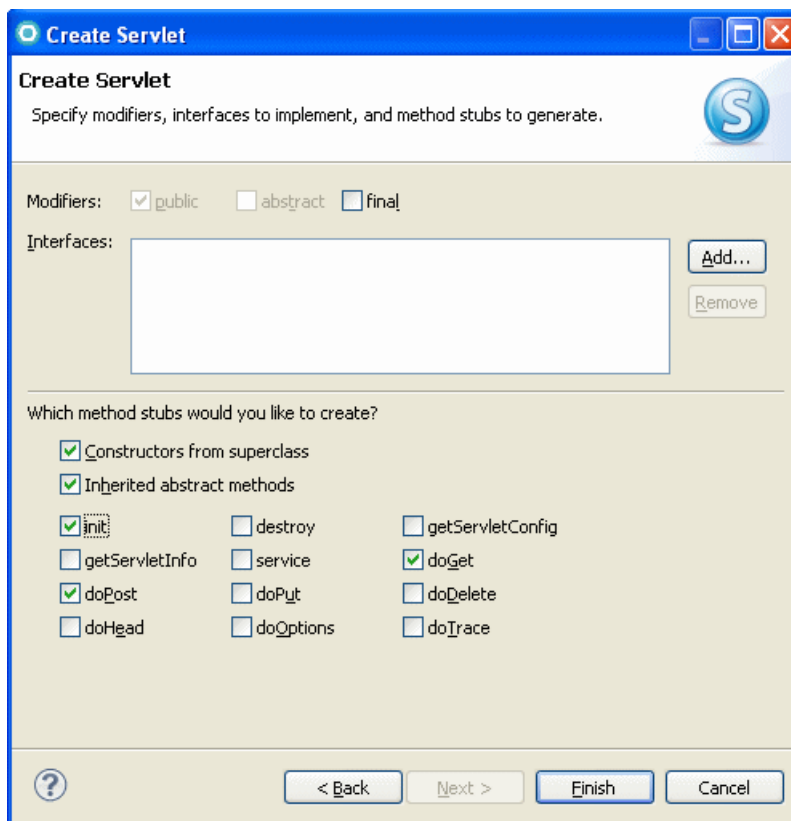


Figure 33-19 AppConfigHTTP servlet creation wizard (page 3)

7. Click **Finish**.

## Creating the MyMainServlet Sip Servlet

Follow these steps:

1. Click **File** → **New** → **Other** → **SIP** → **SIP Servlet**.
2. Click **Next**.
3. Complete the fields in the wizard:
  - a. For Java package, type `com.ibm.siptools.samples`.
  - b. For Class name, type `MyMainServlet`.

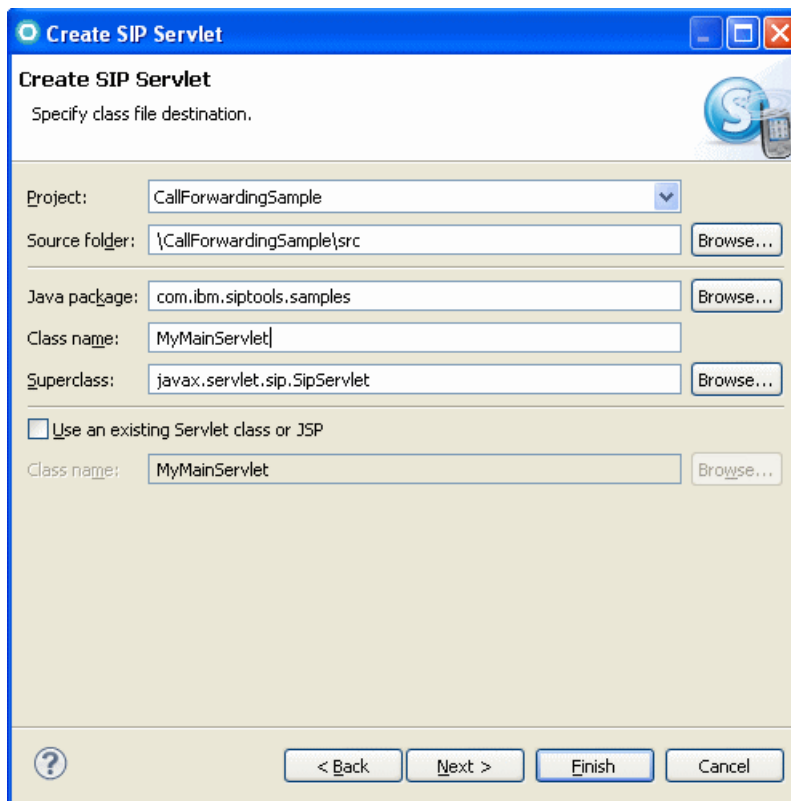


Figure 33-20 Main servlet creation wizard (page 1)

4. Click **Next**.
5. Select **Use this Servlet as the Main Servlet** (Figure 33-21 on page 1755).

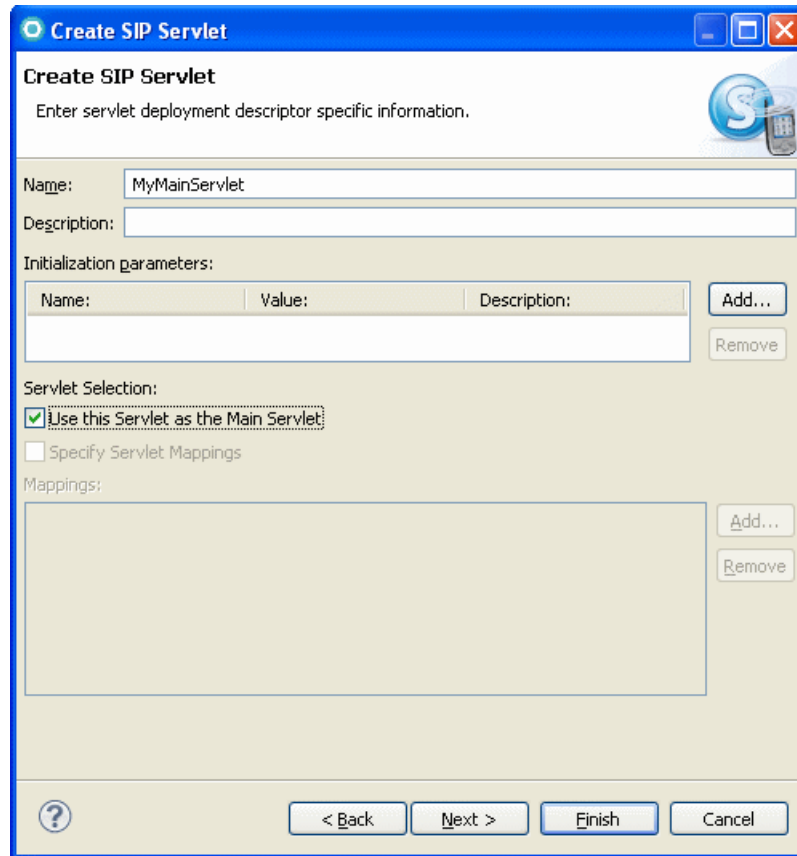


Figure 33-21 Main servlet creation wizard (page 2)

6. Click **Next**.
7. Select the following methods (Figure 33-22 on page 1756):
  - **doRequest**
  - **doResponse**

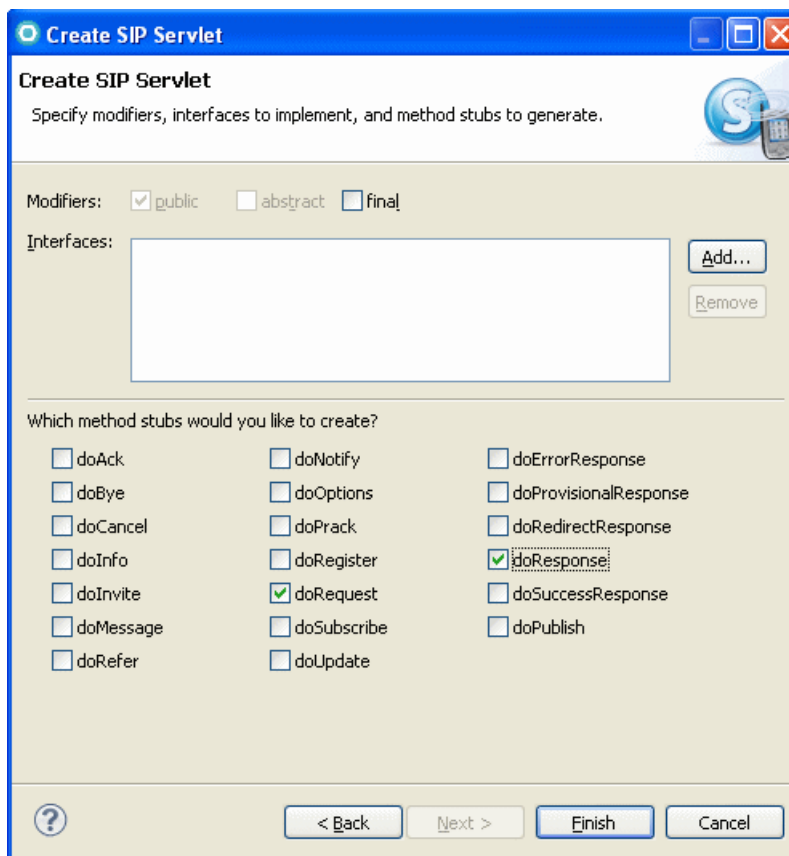


Figure 33-22 Main servlet creation wizard (page 3)

8. Click **Finish**.
9. When you see the Update web deployment descriptor dialog box (Figure 33-23), click **Yes**.

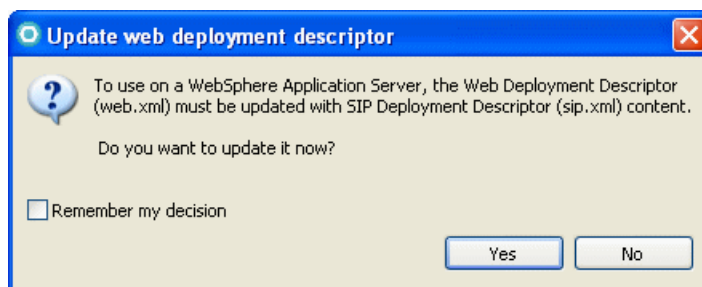


Figure 33-23 Update web deployment descriptor dialog



## Creating the CallForwardSiplet Sip Servlet

Follow these steps:

1. Click **File** → **New** → **Other** → **SIP** → **SIP Servlet**.
2. Click **Next**.
3. Complete the following fields in the wizard (Figure 33-24):
  - a. For Java package, type `com.ibm.siptools.samples`.
  - b. For Class name, type `CallForwardSiplet`.

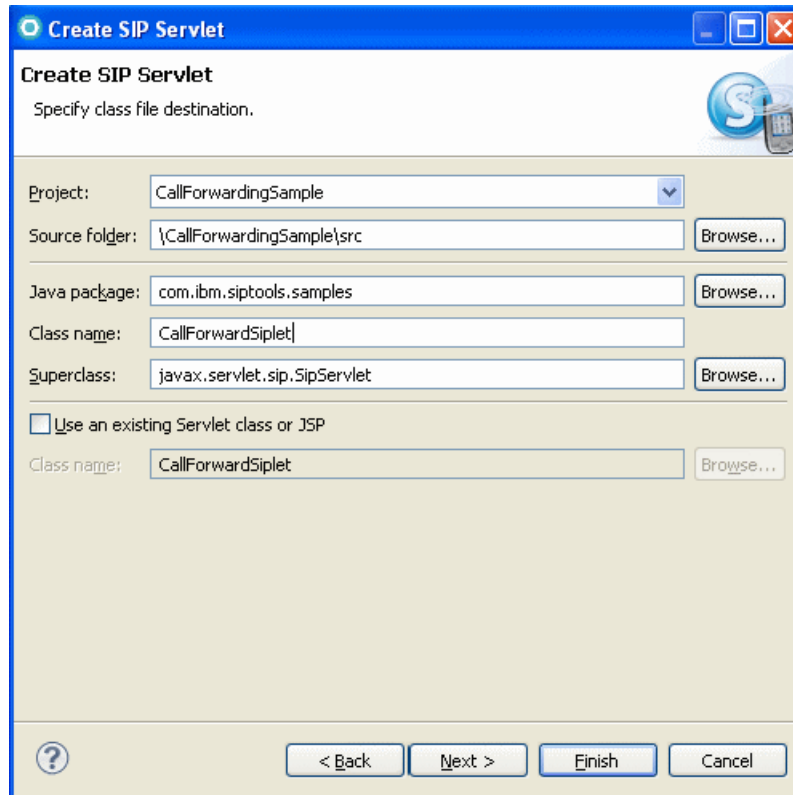


Figure 33-24 Forwarding servlet creation wizard (page 1)

4. Click **Next**.
5. Click **Next**.
6. Select the **doInvite** method (Figure 33-25 on page 1758).

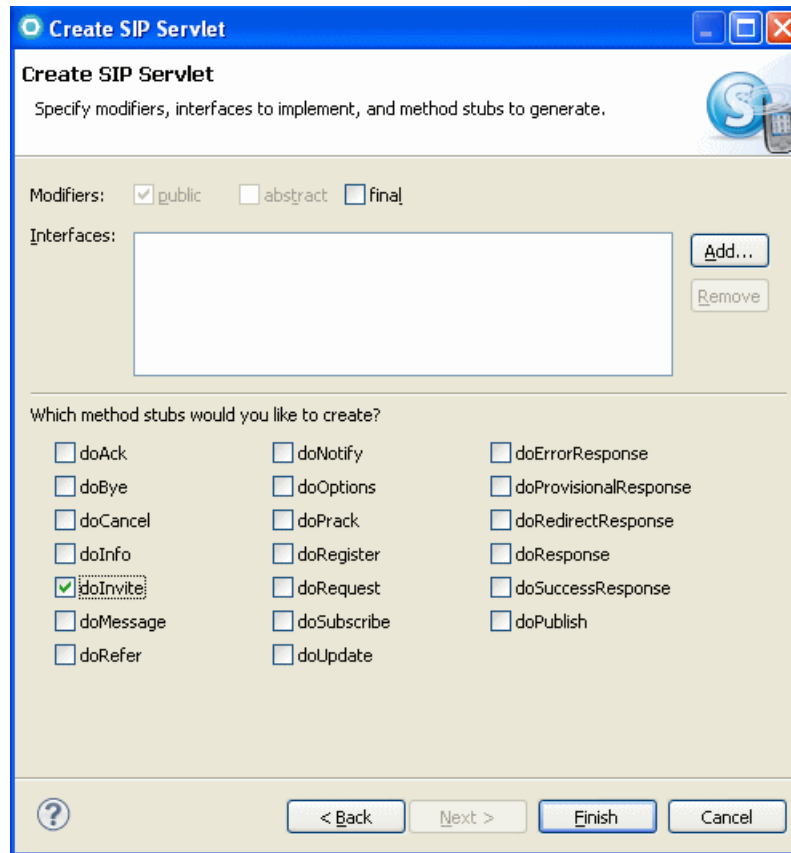


Figure 33-25 Forwarding servlet creation wizard (page 3)

7. Click **Finish**.

**Configuration:** These steps assume that the default configuration for the sip.xml and web.xml merge is set. If it is turned off, it is necessary to call the merge process manually by using **SIP Operations** → **update web.xml** on the sip.xml context menu after adding the last SIP servlet.

### 33.2.4 Implementing the classes

In this section, we describe the following topics:

- ▶ Creating the AccessControlList class
- ▶ Defining the AppConfigHTTP behavior
- ▶ Defining MyMainServlet behavior

- ▶ Defining CallForwardSiplet behavior

## Creating the AccessControlList class

This class serializes and de-serializes the mapping between the public and forwarding addresses in an access control list that is stored in a local file. Follow these steps:

1. Click **File** → **New** → **Other** → **Java** → **Class**.
2. Click **Next**.
3. Set the class data:
  - a. For Java package, type `com.ibm.siptools.samples`.
  - b. For Class name, type `AccessControlList`.
4. Click **Finish**.
5. In the Java Editor, add the contents to the class, as shown in Example 33-1.

*Example 33-1 AccessControlList class method contents*

---

```
public String getForwardAddress(
 String address, InputStream fileStream) {
 System.out.println("AccessControlList-> Address to lookup "
 + address);
 if (fileStream == null){
 System.out.println(
 "AccessControlList->Cannot load contact files");
 return "";
 }
 Object forwardAddress = null;
 Map<String,String> contactMap =
 readSerializedMap(fileStream);
 if (contactMap != null){
 forwardAddress = contactMap.get(address);
 } else {
 System.out.println(
 "AccessControlList->Contact Map loading error");
 }

 if (forwardAddress != null){
 System.out.println(
 "AccessControlList -->Forward address : "
 + (String) forwardAddress);
 } else {
 System.out.println(
 "AccessControlList -->Forward address is null");
 }
}
```

```

 forwardAddress = "";
 }

 return (String)forwardAddress;
}
@SuppressWarnings("unchecked")
public static HashMap<String, String>
 readSerializedMap(InputStream iStream) {
 System.out.println(
 "AccessControlList -->Read serialized file");
 ObjectInputStream oin = null;
 HashMap<String, String> map = null;
 if (iStream != null) {
 try {
 oin = new ObjectInputStream(iStream);
 map = (HashMap<String, String>)oin.readObject();
 oin.close();
 oin = null;
 } catch (FileNotFoundException e) {
 e.printStackTrace();
 } catch (IOException e) {
 e.printStackTrace();
 } catch (ClassNotFoundException e) {
 e.printStackTrace();
 } catch (ClassCastException e) {
 e.printStackTrace();
 } finally {
 if (oin != null) {
 try {
 oin.close();
 } catch (IOException e1) {
 e1.printStackTrace();
 }
 }
 }
 }
 } else {
 System.out.println("File empty");
 }
 return map;
}
public static void writeSerializedMap(
 HashMap<String, String> map, String file) {
 System.out.println(
 "AccessControlList -->Write serialized File");
 ObjectOutputStream oops = null;

```

```

try {
 oops = new ObjectOutputStream(
 new FileOutputStream(file));
 oops.writeObject(map);
 oops.flush();
 oops.close();
 oops = null;
} catch (FileNotFoundException e) {
 e.printStackTrace();
} catch (IOException e) {
 e.printStackTrace();
} finally {
 if (oops != null) {
 try {
 oops.close();
 } catch (IOException e1) {
 e1.printStackTrace();
 }
 }
}
}
}

```

---

6. Press Ctrl+Shift+O. To fix the imports, select these classes:

- **java.io.FileNotFoundException**
- **java.io.InputStream**
- **java.util.Map**

## Defining the AppConfigHTTP behavior

AppConfigHTTP is an HTTP servlet that provides a graphical interface that the user can use to add and remove new forwarding mappings. Follow these steps to define its behavior:

1. Open the **AppConfigHTTP.java** file.
2. In the Java Editor, add the contents to the class, as shown in Example 33-2.

*Example 33-2 AppConfigHTTP class*

---

```

private static final long serialVersionUID = 1L;
private static String fileName;
public void init() throws ServletException {
 fileName = new StringBuilder(
 getServletContext().getRealPath("/")
 .append(File.separator).append("WEB-INF")
 .append(File.separator).append("contacts").toString());
}

```

```

 super.init();
 }
 protected void doGet(
 HttpServletRequest req, HttpServletResponse res)
 throws ServletException, IOException {
 File file = new File(fileName);
 if (file.exists()) {
 HashMap<String, String> map =
 AccessControlList.readSerializedMap(new
FileInputStream(fileName));
 printMap(map, res);
 } else {
 printMap(null, res);
 }
 }
 protected void doPost(
 HttpServletRequest req, HttpServletResponse res)
 throws ServletException, IOException {
 // Get the publicURI and forwardURI
 String publicURI = req.getParameter("publicSIPAddress");
 String forwardURI =
 req.getParameter("forwardingSIPAddress");
 // what action to take
 String clear = req.getParameter("Clear");
 if (clear != null) {
 System.out.println("Clear the file list");
 // delete the file
 File fileObj = new File(fileName);
 if (fileObj.exists()) {
 if (!fileObj.delete()) {
 System.out.println(
 "Cannot clear the file");
 }
 }
 // print content
 printMap(null, res);
 } else {
 HashMap<String, String> map = null;
 File fileObj = new File(fileName);
 if (fileObj.exists()) {
 // read file content
 map = AccessControlList.readSerializedMap(
 new FileInputStream(fileName));
 // insert new entry
 map.put(publicURI, forwardURI);
 }
 }
 }
}

```

```

 // write the map to a file
 AccessControlList.writeSerializedMap(
 map, fileName);
 } else {
 map = new HashMap<String, String>();
 map.put(publicURI, forwardURI);
 // write the map to a file
 AccessControlList.writeSerializedMap(
 map, fileName);
 }
 // print content
 printMap(map, res);
}
}

private void printMap(
 HashMap<String, String> map, HttpServletResponse res)
 throws IOException {
 StringBuilder htmlContents = new StringBuilder()
 .append("<BODY><H1>")
 .append("Call Forwarding Sample configuration
")
 .append("
</H1>")
 .append("<FORM method='post' action='AppConfigHTTP'>")
 .append("<TABLE border='0'><TBODY><TR>")
 .append("<TD>Enter Public SIP address (e.g. sip:bob@someIP.com:5060)</TD><TD>")
 .append("<INPUT type='text' name='publicSIPAddress' size='20'>")
 .append("</TD></TR><TR>")
 .append("<TD>Enter Forwarding SIP address (e.g. sip:bob@otherIP:5060)</TD>")
 .append("<TD><INPUT type='text' name='forwardingSIPAddress' size='20'></TD>")
 .append("</TR><TR>")
 .append("<TD><INPUT type='submit' name='Submit' value='Submit'></TD>")
 .append("<TD><INPUT type='submit' name='Clear' value='Clear'></TD>")
 .append("</TR></TBODY></TABLE></FORM>")
 .append("<HR width='400' align='left'>")
 .append("<P><U>Existing mappings:</U>");
 if (map != null) {
 Iterator<String> iterator = map.keySet().iterator();
 while (iterator.hasNext()) {
 Object key = iterator.next();
 Object value = map.get(key);

```

```

 if (value != null) {
 htmlContents.append("
").append(key).
 append(" <==> ").append(value);
 }
 } else {
 htmlContents.append("
Empty");
 }
 htmlContents.append("</BODY>");
 res.getWriter().write(htmlContents.toString());
}

```

---

In Example 33-2 on page 1761, the following methods perform these tasks:

- The `init()` method initializes the path to use to store the file where the forwarding Map is serialized.
  - The `doGet()` method prints the UI to add or remove forwarding mappings and the already stored mappings, if any stored mappings exist.
  - The `doPost()` method processes the request to add or remove forwarding mappings, printing the page at the end.
  - The `printMap()` method prints the HTML that provides the user interface.
3. Press Ctrl+Shift+O. To fix the imports, select these classes:
- **javax.servlet.Servlet**
  - **java.io.File**
  - **java.util.Iterator**

Figure 33-26 on page 1765 shows the UI that is displayed by the servlet. The following functional elements are part of the UI:

- ▶ Public SIP address field: The requests are directed to this SIP URI.
- ▶ Forwarding SIP address field: This field is the SIP URI to which the requests are forwarded.
- ▶ Submit button: Adds a new mapping.
- ▶ Clear button: Removes all the stored mappings.
- ▶ Existing mapping list: Provides a list of all the stored mappings, if any stored mappings exist.



## Call Forwarding Sample configuration

Enter Public SIP address (e.g. sip:bob@someIP.com:5060)

Enter Forwarding SIP address (e.g. sip:bob@otherIP:5060)

---

Existing mappings:  
Empty

Figure 33-26 HTML page provided by the AppConfigHTTP servlet

### Defining MyMainServlet behavior

This SIP servlet shows how to use a Main servlet to dispatch the requests and responses to the proper SIP servlet.

**Main servlet mechanism:** The Main servlet mechanism to dispatch requests and responses is only present in SIP 1.1. SIP 1.0 projects cannot make use of this feature and must use Servlet Mappings instead.

1. Open the **MyMainServlet.java** file.
2. In the Java Editor, add the contents to the class to the `doRequest` method, as shown in Example 33-3.

Example 33-3 *MyMainServlet.doRequest* method contents

```
protected void doRequest(SipServletRequest request)
 throws ServletException, IOException {
 //Checks the Request, if it is initial and the method is
 //INVITE, then it is redirected to the
 //proper Servlet, in this case CallForwardSiplet
 if (request.getMethod().equalsIgnoreCase("INVITE")
 && request.isInitial()) {
 SipSession session = request.getSession();
 session.setHandler("CallForwardSiplet");
 RequestDispatcher dispatcher = getServletContext().
 getNamedDispatcher("CallForwardSiplet");
 System.out.println("Dispatching request to: "
 + "CallForwardSiplet");
 }
}
```

```

 dispatcher.forward(request, null);
 } else {
 request.createResponse(
 SipServletResponse.SC_SERVER_INTERNAL_ERROR,
 "Servlet dispatching error : main servlet received subsequent
request")
 .send();
 }
}

```

---

After the `SipSession` is received, the `CallForwardSiplet` is set as the handler for this request and further requests or responses that are associated with this call. The request dispatcher is obtained by using the name and the request that were forwarded through it.

3. Press `Ctrl+Shift+O` to fix the imports.

## Defining `CallForwardSiplet` behavior

This SIP servlet redirects the `INVITE` request to the proper SIP URI, according to the stored mapping. When an incoming request is directed to a SIP URI that matches an existent mapping, a separate SIP URI is obtained and used instead. Follow these steps:

1. Open the `CallForwardSiplet.java` file.
2. In the Java Editor, add the contents to the class, as shown in Example 33-4.

*Example 33-4 CallForwardSiplet class contents*

---

```

private static final long serialVersionUID = 1L;
/** i) - sipFactory injected using Annotations*/
@Resource
private SipFactory sipFactory;
/**A simple access control implementation. Could be replaced with
* database lookup or other sophisticated lookup.*/
private AccessControlList list;
/**Path where the contacts list object will be saved*/
private String contactsPath =
 new StringBuilder("/").append(File.separator)
 .append("WEB-INF").append(File.separator)
 .append("contacts").toString();
/**init()- get and save the Factory, instantiate accesscontrol
object*/
public void init() throws ServletException {
 list = new AccessControlList();
 super.init();
}

```

```

/**doInvite() - doInvite message handler. Check if user exists then
send Trying and proxy to address else send Not Found */
public void doInvite(SipServletRequest request)
 throws ServletException, IOException {
 InputStream iStream =
getServletContext().getResourceAsStream(contactsPath);
 // ii) Extract the public address and obtain the forwarding
//address
 String requestTo=request.getTo().getURI().toString();
 String uas = list.getForwardAddress(requestTo, iStream);
 if (uas.equals("")) {
 // Generate the Not Found response
 SipServletResponse notFoundResponse =
 request.createResponse(
 SipServletResponse.SC_NOT_FOUND);
 notFoundResponse.send();
 return;
 }
 System.out.println("UAS :" + uas);
 if (iStream != null) {
 iStream.close();
 }
 // iii) Generate the forwarding SIP URI
 SipURI uri = (SipURI) sipFactory.createURI(uas);

 if (uri.getHost().equals("")) {
 // Generate the Not Found response
 SipServletResponse notFoundResponse = request
 .createResponse(SipServletResponse.SC_NOT_FOUND);
 notFoundResponse.send();
 } else {
 //iv) Create a RequestURI Object and set the data
 SipURI suri = (SipURI)request.getRequestURI().clone();
 // v) check if contact found
 if (uri.getUser() != null) {
 suri.setUser(uri.getUser());
 }
 if (uri.getHost() != null) {
 suri.setHost(uri.getHost());
 }
 if (uri.getTransportParam() != null) {
 suri.setTransportParam(uri.getTransportParam());
 }
 suri.setPort(uri.getPort());
 StringBuilder sb = new StringBuilder("User =").

```

```

 append(uri.getUser());
 append(" Host=").append(uri.getHost());
 append(" Port = ").append(uri.getPort());
System.out.println(sb);
// Create a proxy object
Proxy p = request.getProxy();

//v) Generates the header for the SIP message that
//used by the UAC and UAS continue with the
//communication after the proxy action
SipURI from = (SipURI) request.getFrom().getURI();
request.setHeader("forwarded-to-ip", uri.getHost());
request.setHeader("forwarded-to-port",
 Integer.toString(uri.getPort()));
request.setHeader("forwarded-from-ip", from.getHost());
request.setHeader("forwarded-from-port",
Integer.toString(from.getPort()));

 p.setRecordRoute(false);
 p.setRecurse(true);
 p.setSupervised(false);
 // vi) proxy it now
 p.proxyTo(suri);
 }
}

```

---

3. Press Ctrl+Shift+O. To fix the imports, select these classes:

- **java.io.InputStream**
- **java.io.File**
- **javax.servlet.sip.Proxy**
- **javax.servlet.sip.SipFactory**

Consider these additional notes and clarifications regarding the functionality that is implemented in this SIP servlet:

- ▶ The sipFactory is injected through the @Resource annotation.
- ▶ The public SIP address is obtained from the incoming request, by using the forwarding SIP address that is retrieved from the Map instance.
- ▶ Using the forwarding SIP Address (a String instance), the SIP URI is generated.
- ▶ The destination SIP URI is generated with the data from forwarding the SIP address.

- ▶ Additional information is set to ease the direct communication between the User Agent Client and User Agent Server after the forwarding has finished.
- ▶ The request is then proxied to the forwarding SIP address.

### 33.2.5 SIP deployment descriptor

If you use the SIP servlet wizard to create the `MyMainServlet` and `CallForwardServlet` classes, the related metadata is added to the SIP deployment descriptor (Figure 33-27). If the classes were created by using the generic Class wizard, you can configure the application by using the deployment descriptor.

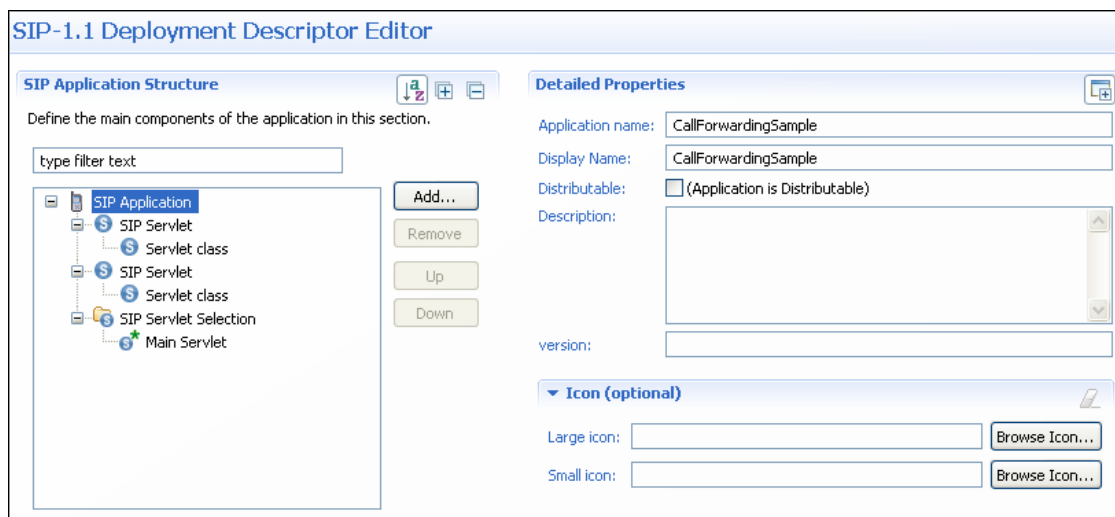


Figure 33-27 SIP deployment descriptor after adding all of the related servlets

After the merge process, the web deployment descriptor looks similar to Figure 33-28 on page 1770.

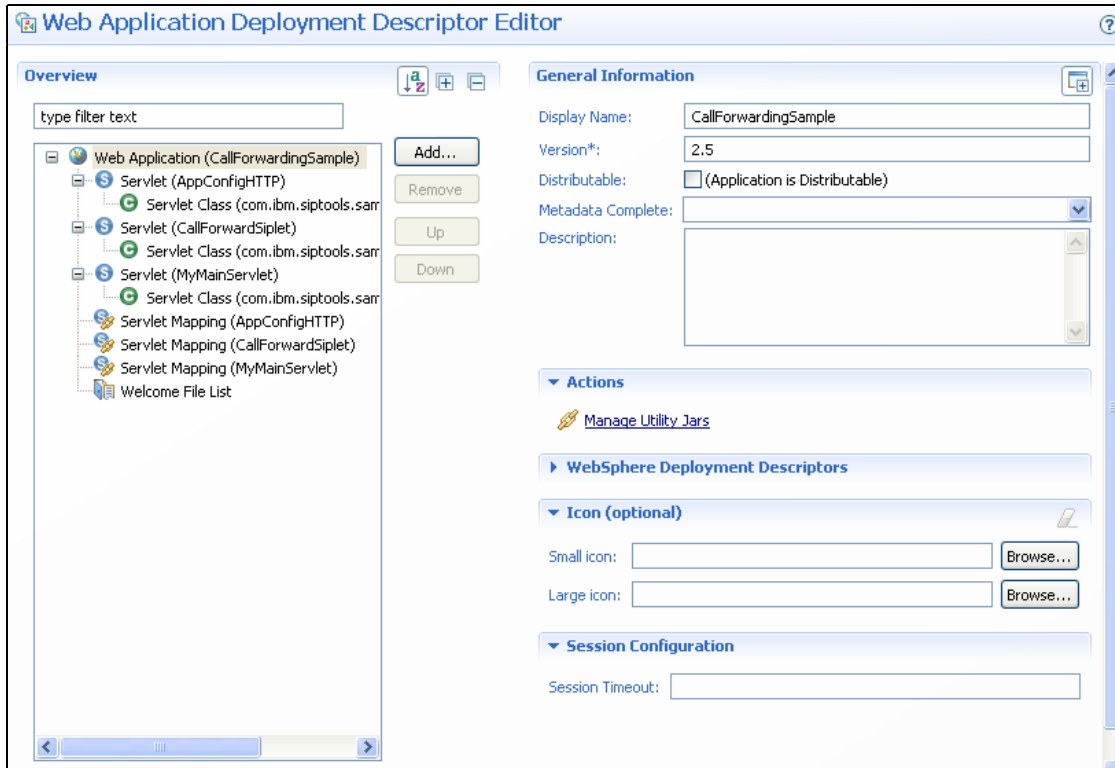


Figure 33-28 Web deployment descriptor after the merge process finished

### 33.2.6 Preparing for deployment

After the files are merged, to deploy the application to Rational Application Developer, you must first verify that the WebSphere Application Server v8.0 Beta profile is configured and working in the workspace.

If the workspace does not have a server profile, create a server profile:

1. Click **File** → **New** → **Other** → **Server** → **Server**.
2. Click **Next**.
3. Select **WebSphere Application Server v8.0 Beta**.
4. For the Server's host name field, type the IP of the remote server or keep localhost if the Server is installed in the same computer (Figure 33-29 on page 1771).

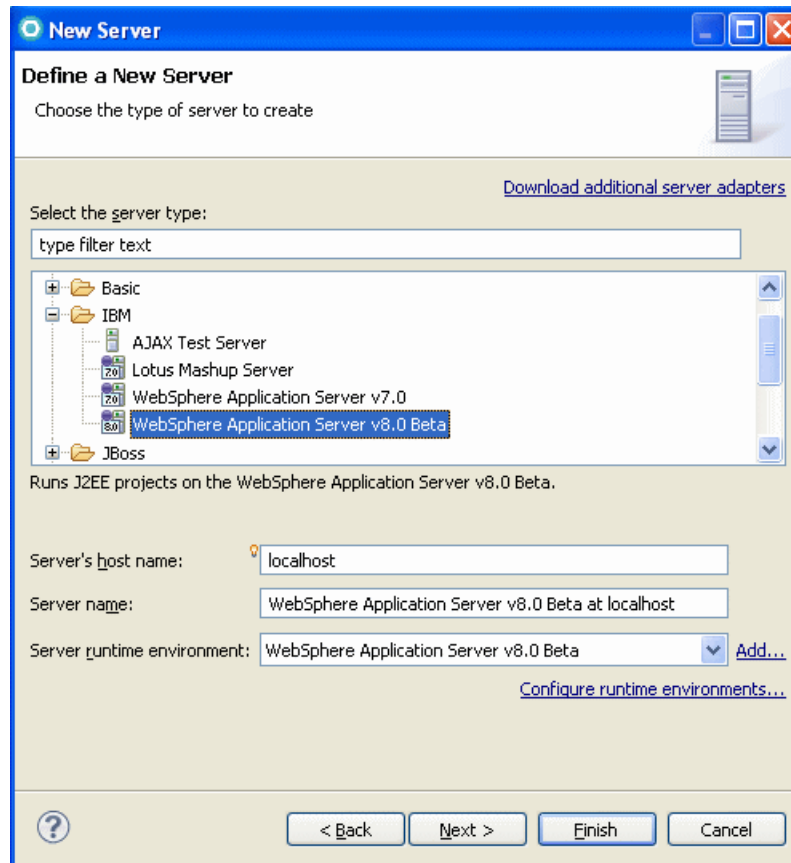


Figure 33-29 New server creation wizard

5. In the case of a remote server, follow these steps:
  - a. Provide the Server connection types and administrative ports (Figure 33-30 on page 1772).

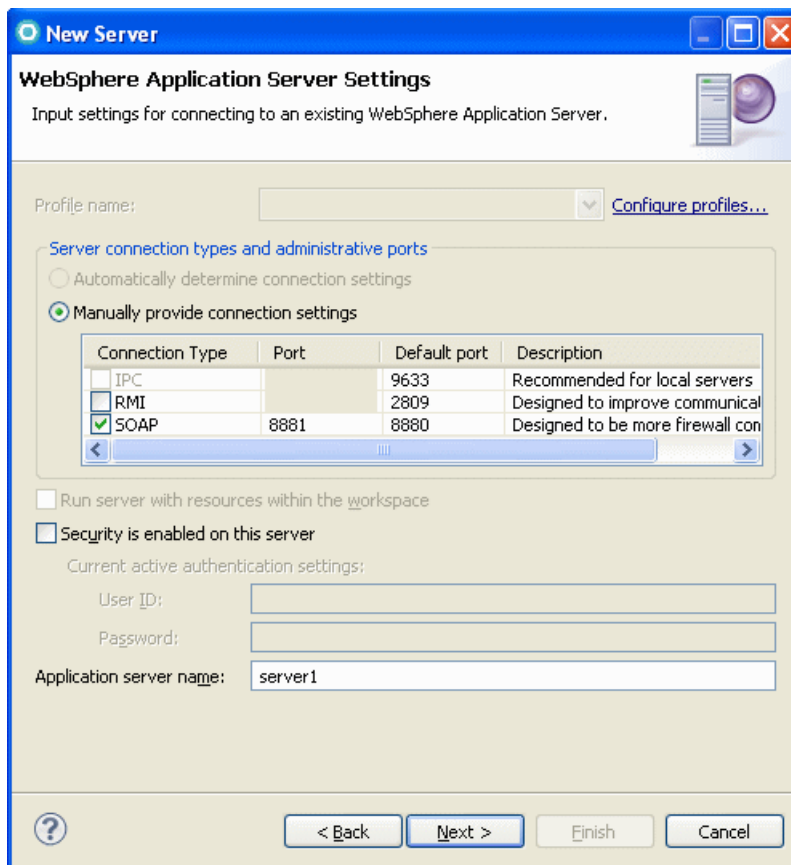


Figure 33-30 New server creation wizard (page 2)

- b. Provide the specific details of the remote server (Figure 33-31 on page 1773).



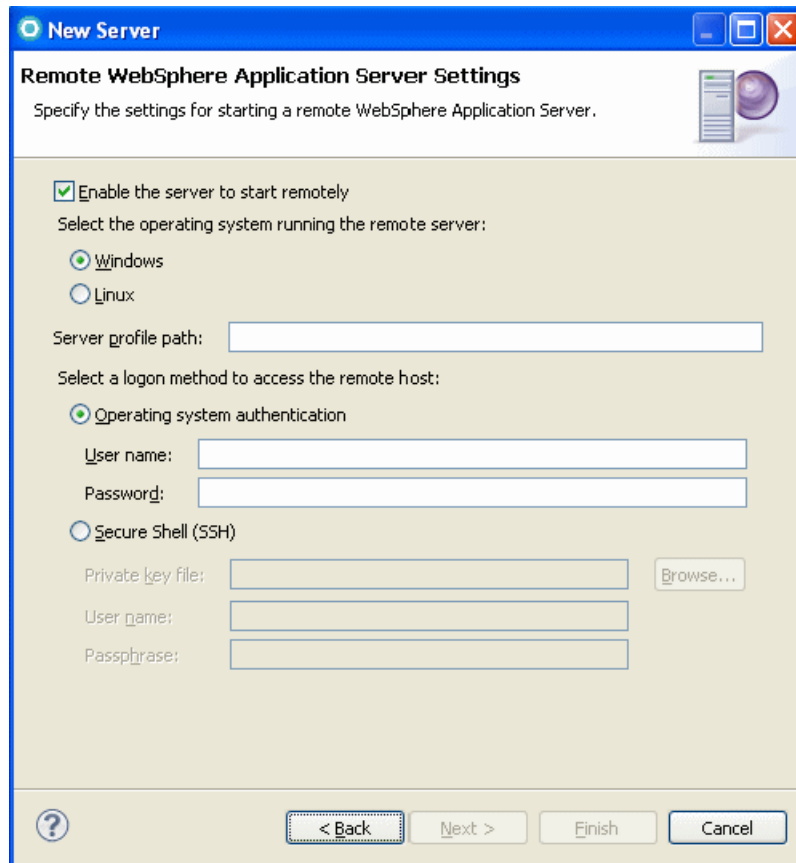


Figure 33-31 New server creation wizard (page 3)

- c. Click **Finish**.
6. Verify that there are no errors in the Problems tab.

### 33.2.7 Deploying SIP from Rational Application Developer

Complete these steps:

1. Open the **Server** tab.
2. Select the server to host the SIP application.
3. Open the context menu on the server and click **Add and Remove**.
4. Select **CallForwardingSampleEAR** (Figure 33-32 on page 1774).

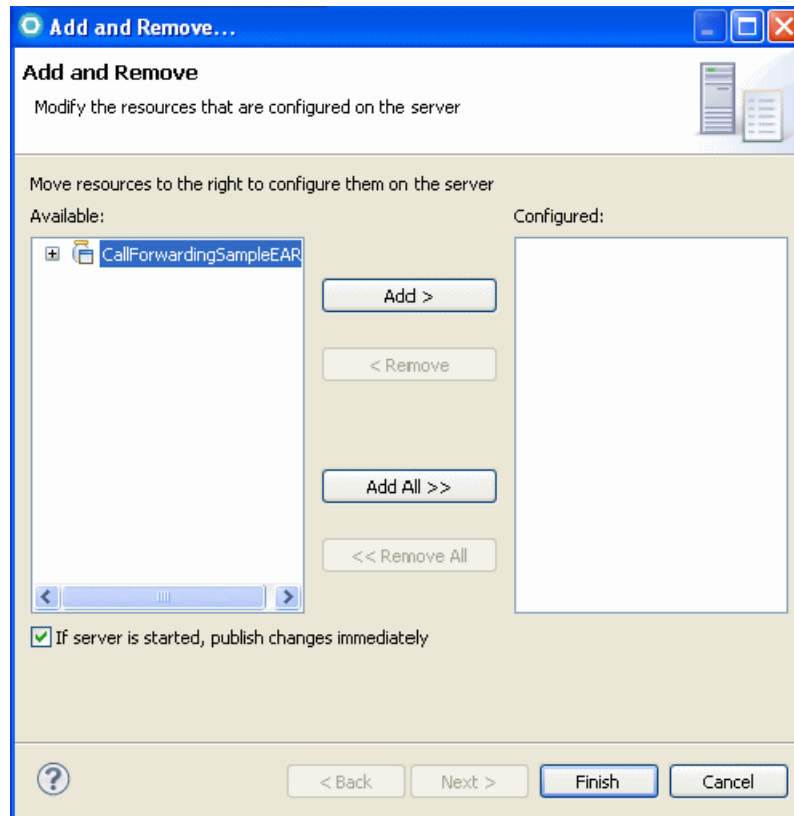


Figure 33-32 Add and Remove wizard

5. Click **Add** and then click **Finish**.
6. Verify that the application was added and started without any problems or errors (Figure 33-33).

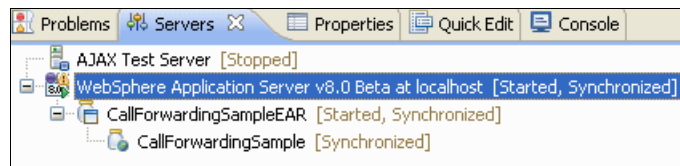


Figure 33-33 Add and Remove wizard

## 33.3 Testing the SIP 1.1 application

To run a SIP application from Rational Application Developer, you must set up the SIP user agents to act as endpoints to generate, receive, or terminate calls. Third-party tools supply these user agents. One tool is called *SIPp*, which is available from this website:

<http://sipp.sourceforge.net/>

We used SIPp to test the CallForwarding sample.

Additional, freeware-based softphones exist that can be used, such as Xlite, sipXphone, and SJPhone. For more information about these tools, refer to *Developing SIP and IP Multimedia Subsystem (IMS) Applications*, SG24-7255.

### 33.3.1 Test environment

We tested the CallForwarding sample in an environment with SIPp running as the SIP user agents. We tested the CallForwarding sample on the Microsoft Windows XP and Ubuntu Linux operating system platforms. We used the following specific SIPp V3.1 packages.

#### Microsoft Windows XP with Service Pack (SP) 3

We downloaded `sipp-win32-2008-07-18.exe` downloaded from this website:

<http://www.sipp.sourceforge.net/snapshots>

**Prerequisites:** Installation of this binary executable might require prerequisite software, such as winpcap, which is available from this website:

<http://www.winpcap.org/install/default.htm>

#### Ubuntu Linux

We downloaded `sip-tester 3.1.r590-1`, which is available from this website:

<http://www.packages.ubuntu.com/lucid/sip-tester>

To set up the user agents, you need the SIPp call flow files, `call_forwarding_uas.xml` and `call_forwarding_uac.xml`, which we supply in Appendix C, “Additional material” on page 1877.

## 33.3.2 Running the application

Follow these steps:

1. Download the `call_forwarding_uas.xml` and `call_forwarding_uac.xml` files from `\7835code\sip` in the Appendix C, “Additional material” on page 1877 and copy them to a directory, such as `c:\temp`.
2. Ensure that the application is published to the server, as described in 33.2.7, “Deploying SIP from Rational Application Developer” on page 1773.
3. Run the web client by expanding the **CallForwardingSample11** project and right-clicking the **AppConfigHTTP.java** file.
4. Click **Run as** → **Run on Server**. The web client starts (Figure 33-34).

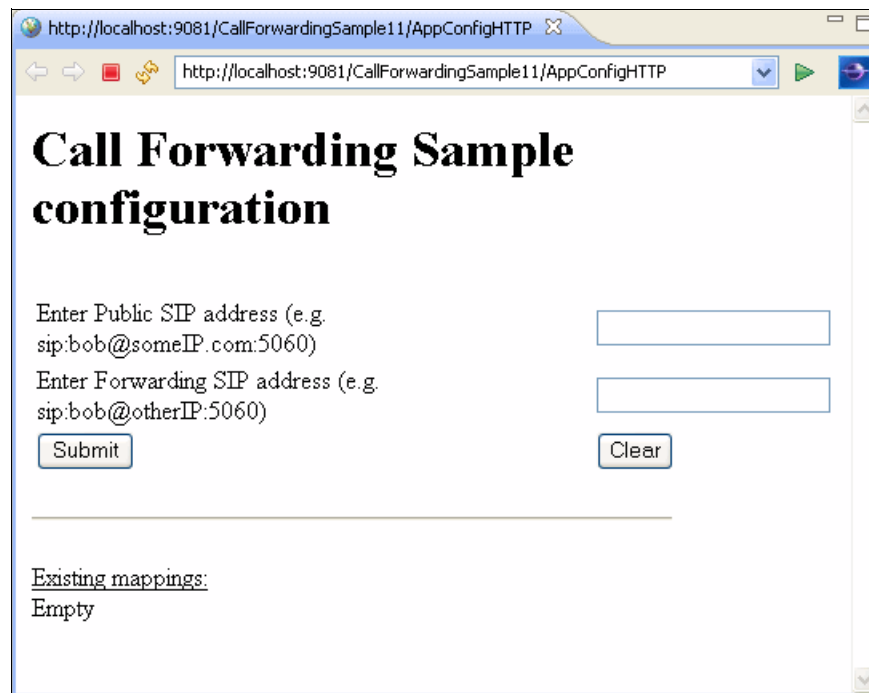


Figure 33-34 CallForwarding web client

5. Set the mapping by entering the public and forwarding address in the web client. The addresses have the following general form:  
`sip:[service]@[ip]:[port]`
6. The parameters have these values:
  - `[service]: sipp-uas`

- [ip]: The public IP is the machine where WebSphere Application Server is installed. The forwarding IP is the machine where the User Agent Server agent is launched.
  - [port]: The public port number is the SIP listener (either secure or unsecure) and the forwarding port number is the port number on which the User Agent Server is listening.
7. After a mapping is specified, it is listed in the Existing mappings section of the web client (Figure 33-35).

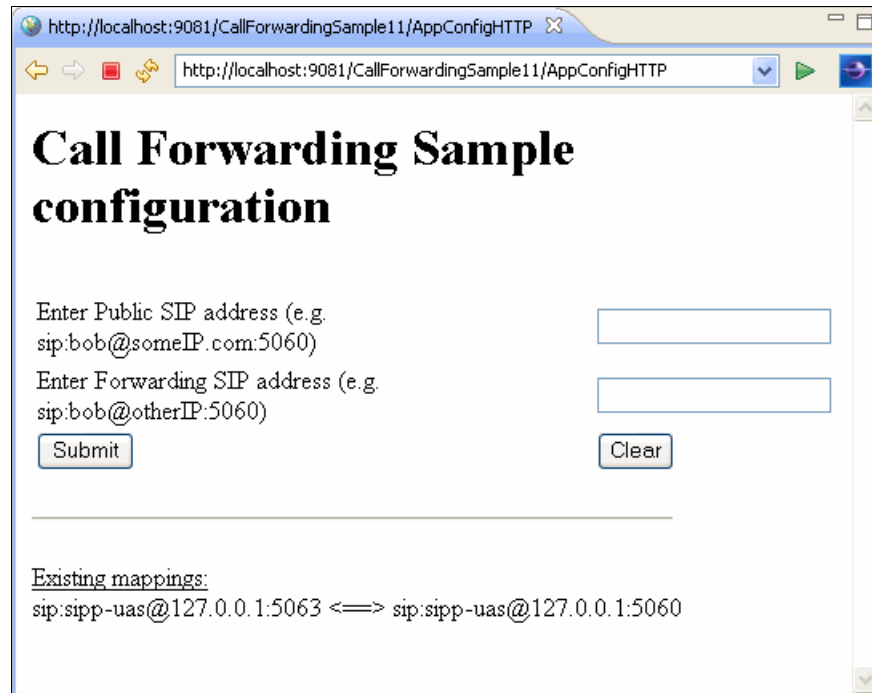


Figure 33-35 Existing mappings in the web client

8. Start the SIP user agents. For example, on Microsoft Windows, click **Start** → **All Programs** → **Sipp\_3.1** → **Start sipp**.
9. In one sipp terminal, enter the command to start the User Agent Server session on port 5060:

```
sipp [UAC_host_ip]:[UAC_port] -s sipp-uac -p 5060 -sf
./call_forwarding_uas.xml -m 2
```

In the preceding command, `-s` indicates the name of the service called `sipp-uac`, `-sf` is the location of the User Agent Server scenario file, and `-m` limits the number of calls to 2.

The User Agent Server session looks similar to Figure 33-36.

```
start sipp - sipp 127.0.0.1:5061 -s sipp-uac -p 5060 -sf ./call_forwarding_uas.xml -m 2
----- Scenario Screen ----- [1-9]: Change Screen -----
Port Total-time Total-calls Transport
5060 35.17 s 0 UDP

0 new calls during 1.016 s period 15 ms scheduler resolution
0 calls Peak was 0 calls, after 0 s
0 Running, 1 Paused, 4 Woken up
0 dead call msg (discarded)
3 open sockets

-----> INVITE Messages Retrans Timeout Unexpected-Msg
0
<----- 180 0 0
<----- 200 0 0
-----> ACK 0 0 0 0

-----> BYE 0 0 0 0
[NOP]
<----- 200 0 0

----- Sipp Server Mode -----
```

Figure 33-36 SIPp CallForwarding UAS session

10. In the second sipp terminal, issue the command to start the User Agent Client session on port 5061:

```
sipp [WAS_host_ip]:[SIP_listener_port] -i [local_ip] -s sipp-uac -p
5061 -d 100000 -l 256 -sf ./call_forwarding_uac.xml -m 2
```

In the preceding command, `-s` is the service called `sipp-uac`, `-sf` is the location of the User Agent Client scenario file, and `-m` limits the number of calls to 2.

**SIP\_listener\_port number:** You can determine the *SIP\_listener\_port number* through the WebSphere Application Server administrative console by navigating to **Application servers** → **server1** → **ports**.

11. After the User Agent Client session starts, the test runs and the results in the SIPp terminals look like Figure 33-37 on page 1779 and Figure 33-38 on page 1779.

```

start sipp -s sipp 127.0.0.1:5061 -s sipp-uac -p 5060 -sf ./call_forwarding_uas.xml -m 2
----- Scenario Screen ----- [1-9]: Change Screen --
Port Total-time Total-calls Transport
5060 16.65 s 2 UDP

Call limit reached (-m 2), 1.015 s period 15 ms scheduler resolution
0 calls Peak was 1 calls, after 4 s
0 Running, 3 Paused, 3 Woken up
0 dead call msg (discarded)
3 open sockets

-----> INVITE Messages Retrans Timeout Unexpected-Msg
-----> 180 2 0 0 0
<----- 200 2 0 0 0
<----- 200 2 0 0 0
-----> ACK 2 0 0 0

-----> BYE 2 0 0 0
[NOP]
<----- 200 2 0 0 0
----- Sipp Server Mode -----

```

Figure 33-37 User Agent Server session results

```

start sipp
----- Scenario Screen ----- [1-9]: Change Screen --
Call-rate(length) Port Total-time Total-calls Remote-host
10.0(100000 ms)/1.000s 5061 0.34 s 2 127.0.0.1:5063(UDP)

Call limit reached (-m 2), 0.000 s period 0 ms scheduler resolution
0 calls (limit 256) Peak was 1 calls, after 0 s
0 Running, 4 Paused, 0 Woken up
0 dead call msg (discarded) 0 out-of-call msg (discarded)
1 open sockets

-----> INVITE Messages Retrans Timeout Unexpected-Msg
100 <-----> 2 0 0 0
180 <-----> 2 0 0 0
200 <-----> 2 0 0 0
[NOP]
ACK -----> 2 0 0 0
BYE -----> 2 0 0 0
200 <-----> 2 0 0 0
----- Test Terminated -----

----- Statistics Screen ----- [1-9]: Change Screen --
Start Time ! 2010-12-03 09:41:45:248 1291387305.248367

```

Figure 33-38 User Agent Client session results

## 33.4 SIP-specific annotations in SIP 1.1 applications

The WebSphere Application Server V7.0 Communications Enabled Applications (CEA) Feature Pack or WebSphere Application Server v8.0 Beta run times cannot process SIP-specific annotations in SIP 1.1 applications deployed from Rational Application Developer. For more information about this situation and the work-around, refer to this web doc (technote):

[http://www-01.ibm.com/support/docview.wss?rs=0&q1=1443583&uid=swg21443583&loc=en\\_US&cs=utf-8&cc=us&lang=en](http://www-01.ibm.com/support/docview.wss?rs=0&q1=1443583&uid=swg21443583&loc=en_US&cs=utf-8&cc=us&lang=en)

## 33.5 More information

For more information about SIP 1.1, see the following resources:

- ▶ *JSR 289: SIP Servlet Specification, Version 1.1*  
<http://jcp.org/en/jsr/summary?id=289>
- ▶ IBM Education Assistant  
[http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/index.jsp?topic=/com.ibm.iea.wasfpcea/plugin\\_types.html](http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/index.jsp?topic=/com.ibm.iea.wasfpcea/plugin_types.html)
- ▶ Rational Application Developer Information Center  
<http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/com.ibm.siptools.doc/topics/developSIP.html>
- ▶ *Developing SIP and IP Multimedia Subsystem (IMS) Applications*, SG24-7255
- ▶ Introduction to the SIP Modeling Toolkit  
[http://www.ibm.com/developerworks/rational/library/07/0807\\_conallen/](http://www.ibm.com/developerworks/rational/library/07/0807_conallen/)



## Appendixes

This part includes the following appendixes:

- ▶ Appendix A, “Installing the products” on page 1783
- ▶ Appendix B, “Performance tips for Rational Application Developer” on page 1871
- ▶ Appendix C, “Additional material” on page 1877





# A

## Installing the products

In this appendix, we highlight the key installation considerations and options, identify components that were installed while writing this book, and provide a general awareness regarding the use of IBM Installation Manager to install IBM Rational Application Developer.

We organized this appendix into the following sections:

- ▶ Download locations
- ▶ Installation Launchpad
- ▶ IBM Installation Manager
- ▶ Installing Rational Application Developer
- ▶ Installing WebSphere Portal V7
- ▶ Installing IBM Rational Team Concert
- ▶ Installing Rational Application Developer Build Utility
- ▶ Installing IBM Rational ClearCase
- ▶ Installing IBM Rational ClearCase Remote Client Extension
- ▶ Configuring ClearCase for UCM development

## Download locations

You can download the software from the following locations:

- ▶ You can download IBM Rational Application Developer V7 from this website:  
<http://www-01.ibm.com/support/docview.wss?uid=swg24027295>
- ▶ You can download the trial from this website:  
<http://www.ibm.com/developerworks/downloads/r/rad/>
- ▶ You can download IBM Rational Application Developer Standard Edition V8 from this website:  
<http://www-01.ibm.com/support/docview.wss?uid=swg24027296>
- ▶ The trial is available from this website:  
<http://www.ibm.com/developerworks/downloads/r/radse/index.html>
- ▶ For other versions of IBM Rational Application Developer, go to this website:  
[http://www-947.ibm.com/support/entry/portal/All\\_download\\_links/Software/Rational/Rational\\_brand\\_support\\_%28general%29](http://www-947.ibm.com/support/entry/portal/All_download_links/Software/Rational/Rational_brand_support_%28general%29)

## Installation Launchpad

You can install Rational Application Developer by using any of the following methods:

- ▶ Installing from the CDs
- ▶ Installing from a downloaded electronic image on your workstation
- ▶ Installing from an electronic image on a shared drive
- ▶ Installing from a repository on an HTTP or HTTPS server

In the following steps, we describe how to install from a downloaded electronic image on your workstation:

1. After you download all the components of Rational Application Developer, extract the files into an installation folder. From that folder, start the Launchpad by executing ***RAD\_SETUP\1launchpad.exe***.
2. On the first page, select the language, for example, **English**, and click **OK**.
3. On the Launchpad that opens (Figure A-1 on page 1785), click **Install IBM Rational Application Developer for WebSphere Software**.



Figure A-1 Launchpad

## IBM Installation Manager

We used IBM Installation Manager to install Rational Application Developer. IBM Installation Manager is a program that helps you install the Rational desktop product packages on your workstation. It also helps you update, modify, and uninstall this package and other packages that you install. A *package* can be a product, a group of components, or a single component that is designed to be installed by Installation Manager.

If you use the Launchpad to install Rational Application Developer, it first installs IBM Installation Manager. Then it configures the repository for the product and installs the product. You can also install IBM Installation Manager separately (or you might have it installed already due to other product installations).

To install Version 1.4.1 of Installation Manager from the Launchpad, select **IBM Installation Manager** → **Version 1.4.1**, as shown in Figure A-2 on page 1786, and click **Next**.

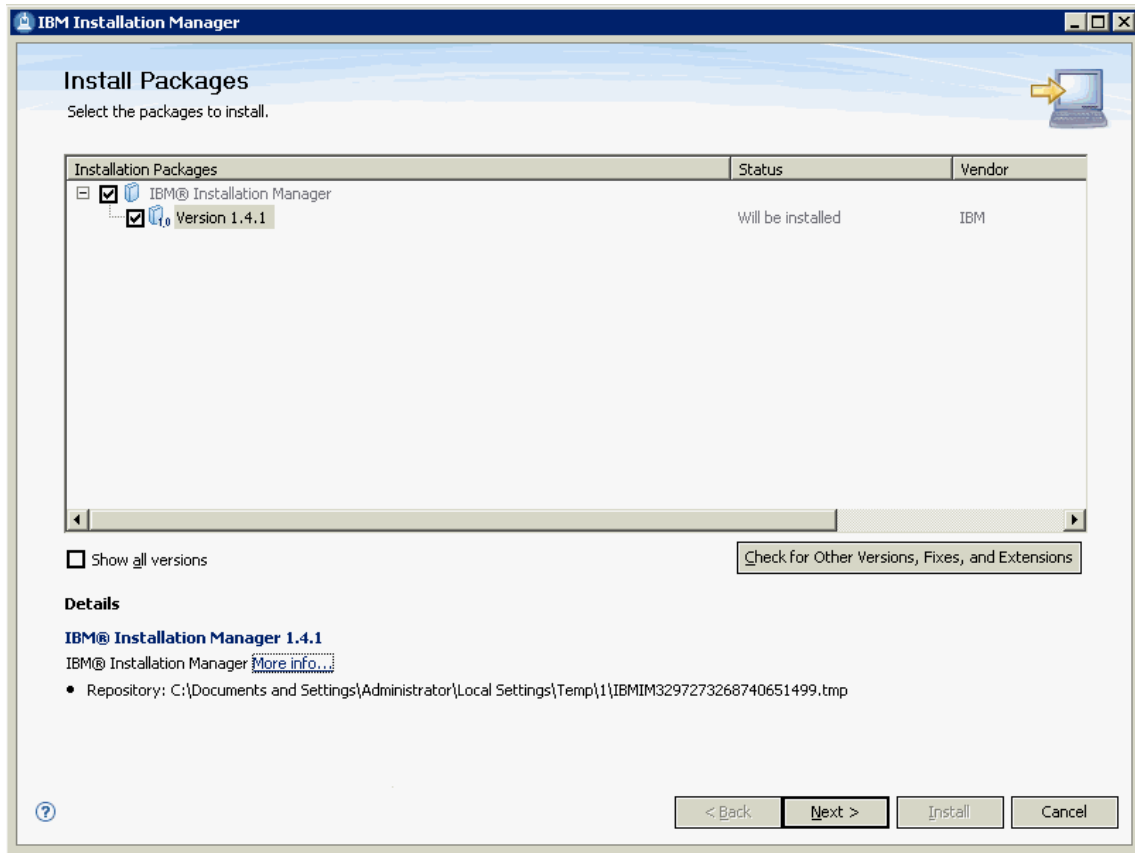


Figure A-2 Installing IBM Installation Manager from the Launchpad

To install Version 1.4.1 of Installation Manager separately, select ***RAD\_SETUP\InstallerImage\_win32\install.exe***.

In all cases, continue the installation with the following steps:

1. Accept the license agreement.
2. Select the **C:\Program Files\IBM\Installation Manager\eclipse** installation directory.
3. Click **Install**, wait for the installation to finish, and click **Restart Installation Manager**.

Six wizards in the Installation Manager make it easy to maintain your package through its life cycle (Figure A-3 on page 1787):

- ▶ The *Install wizard* guides you through the installation process.

- ▶ The *Update wizard* searches for available updates to packages that you have installed.
- ▶ With the *Modify wizard*, you can modify certain elements of a package that you have installed already.
- ▶ The *Manage Licenses wizard* helps you set up the licenses for your packages.
- ▶ With the *Roll Back wizard*, you can revert back to a previous version of a package.
- ▶ The *Uninstall wizard* removes a package from your computer.



Figure A-3 Installation Manager

For more information about IBM Installation Manager 1.4.1, go to this website:  
<http://publib.boulder.ibm.com/infocenter/install/v1r4/index.jsp>

# Installing Rational Application Developer

Perform these steps to install Rational Application Developer:

If you installed Installation Manager separately or had it pre-installed, perform these preliminary steps:

1. Select **File** → **Preferences** → **Repositories**.
2. Add the following repository location:  
`<path to extracted files>\RAD\disk1\disk\diskTag.inf`

Continue with the following steps, which are required for the users of the Launchpad.

If you used the Launchpad, perform these steps:

1. From the Installation Manager, click **Install** to install Rational Application Developer.
2. On the Install Packages page (Figure A-4), select **Rational Application Developer for WebSphere Software Version 8.0.1**. Complete these tasks:
  - a. Optional: Click **Check for Other Versions or Extension** to see if newer versions are available.
  - b. Click **Next**.

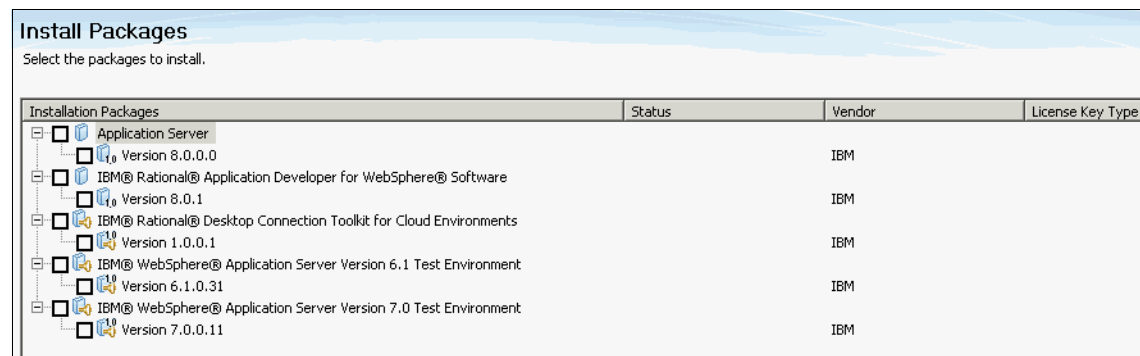


Figure A-4 Install Packages page that is shown when selecting Rational Application Developer

3. On the next page, validate the prerequisites, which include stopping any antivirus programs, and click **Next**.
4. Select **I accept the terms in the license agreements** and then click **Next**.
5. On the “Select a location for the shared resources directory” page (Figure A-5 on page 1789), accept the Shared Resource Directory from **C:\Program Files\IBM\SDPShared**. Depending on whether you have already used



Installation Manager to install software, you might have to select the shared resources directory on the Location page. Type the path in the Shared Resources Directory field or accept the default path. Then click **Next** to continue. The shared resources directory contains resources that can be shared by one or more package groups. You can specify the shared resources directory only at the time that you install Installation Manager. Use your disk with the most available space for the shared resources to help ensure that you have adequate space for the shared resources of future packages. You cannot change the directory location unless you uninstall all packages. Click **Next**.

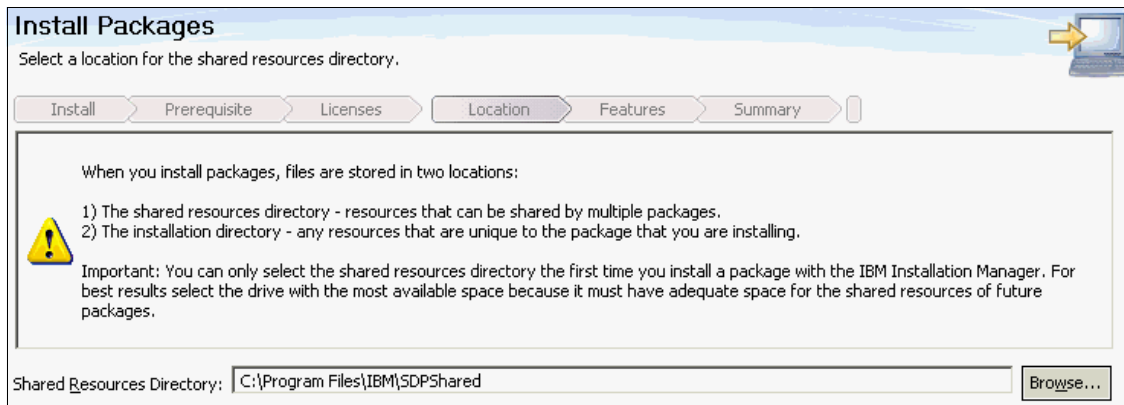


Figure A-5 Shared Resources Directory selection page

6. On the Install Packages page (Figure A-6), select **Create a new package group**. Set the installation directory for the package group to **C:\Program Files\IBM\SDP** and click **Next**.

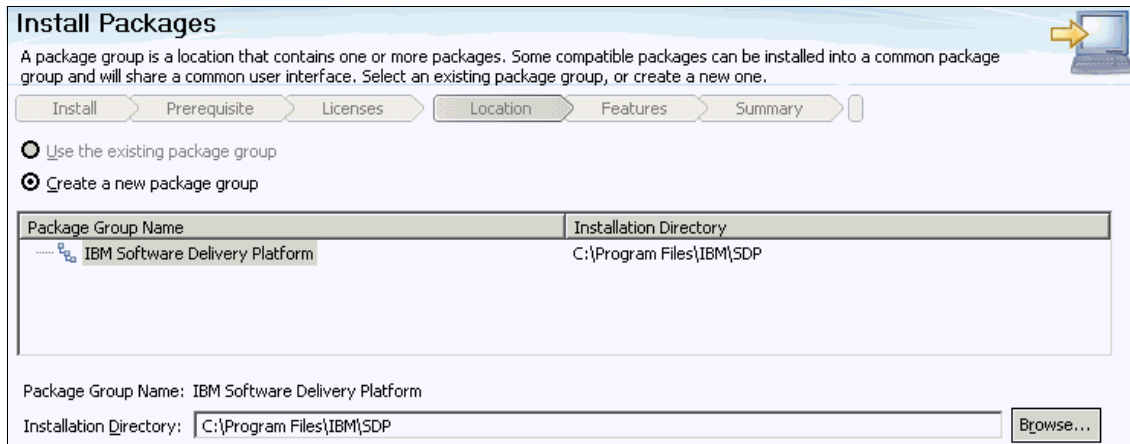


Figure A-6 installation location for the package group

7. On the Extend an existing Eclipse page, leave the default settings. In this example, we do not want to extend an existing Eclipse. Click **Next**.
8. On the “Select the translations you want to install” page, select your desired languages and click **Next**.
9. On the “Select the features you want to install” page, select the package features that you want to install (Figure A-7). We explain the relevant features in each chapter of this book. We highly recommend that you do not select all features because selecting all features increases the size of this installation unnecessarily. At any time, you can return to Installation Manager, select the Modify option, and return to this page, where you can select additional features, provided that you still have the original source files available.



Figure A-7 Available features for installation (page 1)

On Figure A-8 on page 1791, you can see that now the development tools for each server are separate from the required tools when there is no local server installation. If you install WebSphere Application Server V8 Beta locally, you do not need to install the corresponding tool for developing without a local server installation.

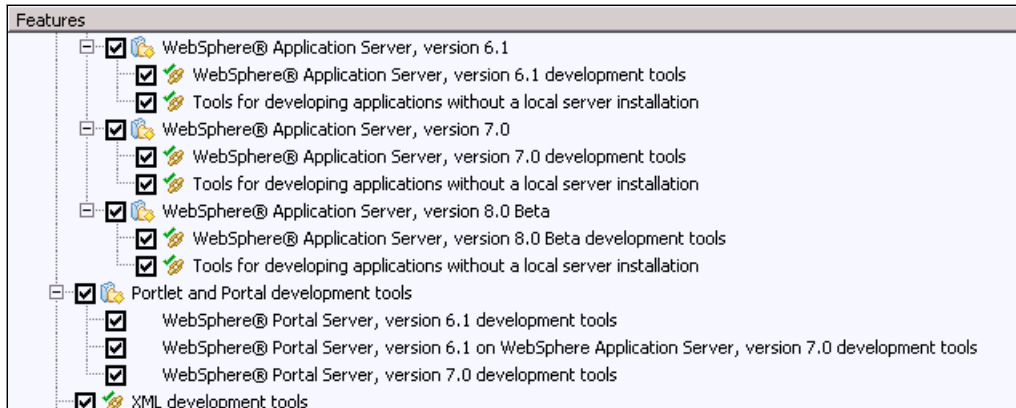


Figure A-8 Available features for installation (page 2)

On Figure A-9, you can see the J2EE Connector architecture (J2C) tools, which are needed only if you connect to enterprise systems. You also can see the life-cycle integrations, where you can find the Rational ClearCase software configuration management (SCM) Adapter.

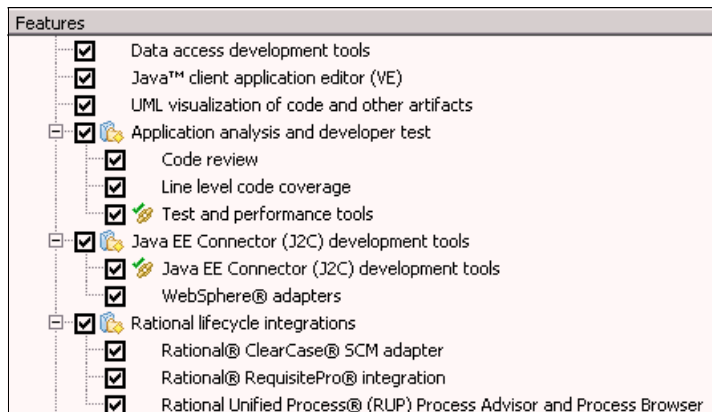


Figure A-9 Available features for installation (page 3)

On Figure A-10 on page 1792, you can see the Collaborative Debug Extension for Rational Team Concert, which allows you to share debug sessions among users.

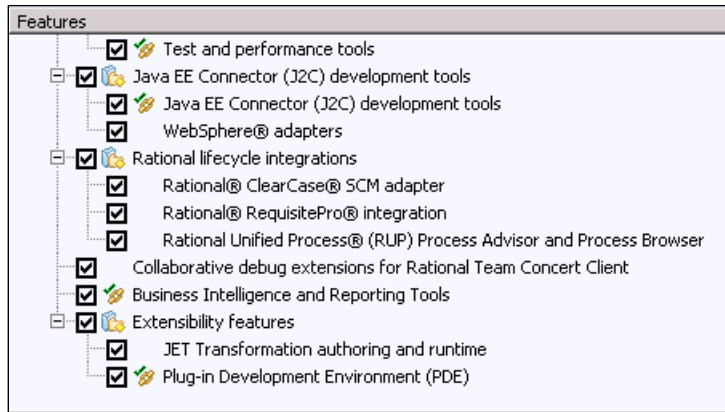


Figure A-10 Available features for installation (page 4)

10. On the Common Configurations page (Figure A-11), select how you want to access the Help system (web, download for local access, or intranet server).

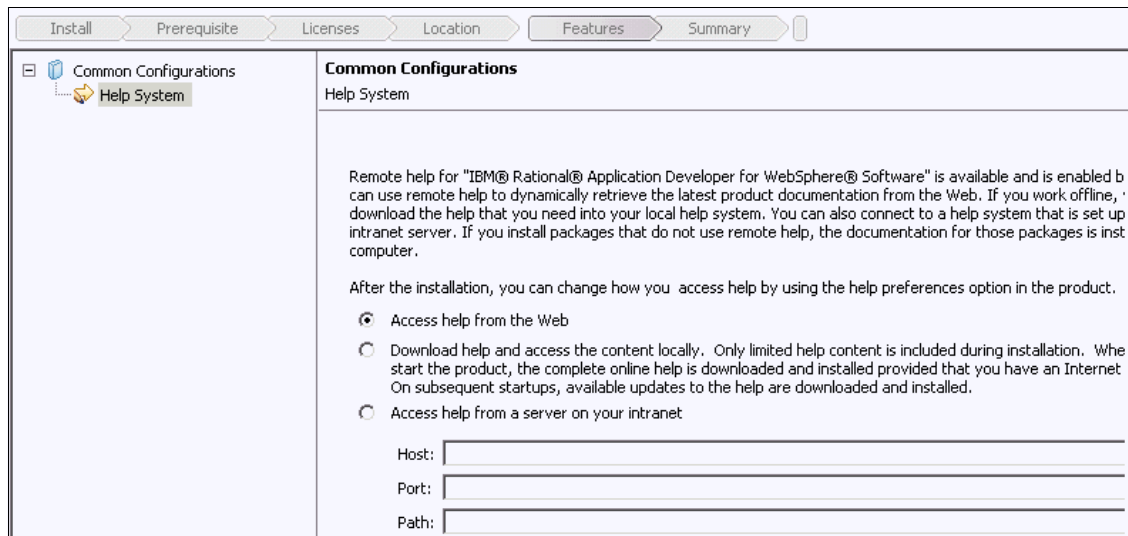


Figure A-11 Configuring the Help system

11. You see a summary page where you can review all parameters. Click **Install**.

12. At the end of the installation, you are prompted to start Rational Application Developer.

## Installing IBM WebSphere Application Server V7

Perform these steps to install IBM WebSphere Application Server.

1. Change to the *RAD\_SETUP* subdirectory of the directory where you extracted the binaries for Rational Application Developer for WebSphere Software.
2. Start the Launchpad program: Run **1aunchpad.exe** (Figure A-1 on page 1785).
3. On the Launchpad dialog box, click **Install IBM Rational Application Developer V8.0**. IBM Installation Manager starts.
4. On the first page of the Install Packages wizard, select **WebSphere Application Server version 7.0 Test Environment** and **Version 7.0.0.13** (Figure A-4 on page 1788) and then click **Next**.
5. To search for updates to the packages, click **Check for Other Versions and Extensions**. Installation Manager searches for updates at the predefined IBM update repository for the product package. It also searches any repository locations that you have set. Click **Next**.
6. On the Licenses page, read the license agreements for the selected packages. On the left side of the License page, click each package version to display its license agreement. If you agree to the terms of all of the license agreements, click **I accept the terms of the license agreements**. Click **Next** to continue.
7. On the Location page, accept **Use existing package group**, which is the default selection of installing the test environment in the same package group as Rational Application Developer. Then click **Next** (Figure A-12 on page 1794).

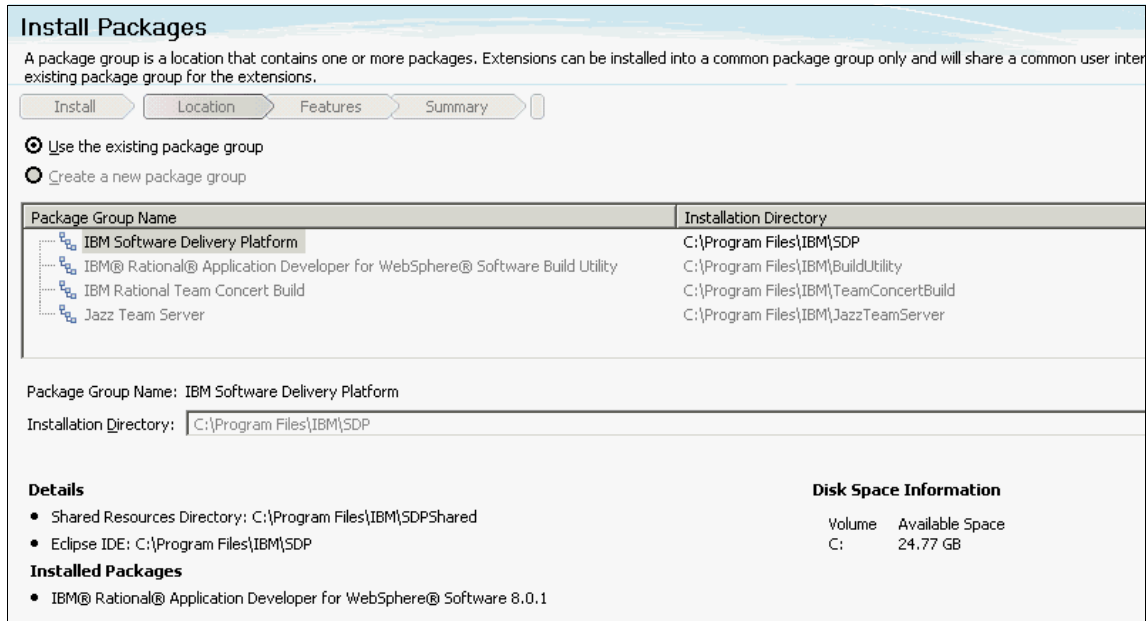


Figure A-12 Installing the test environment in the existing package group

- On the Features Page, select all the desired feature packs (Figure A-13). Select **Next**.

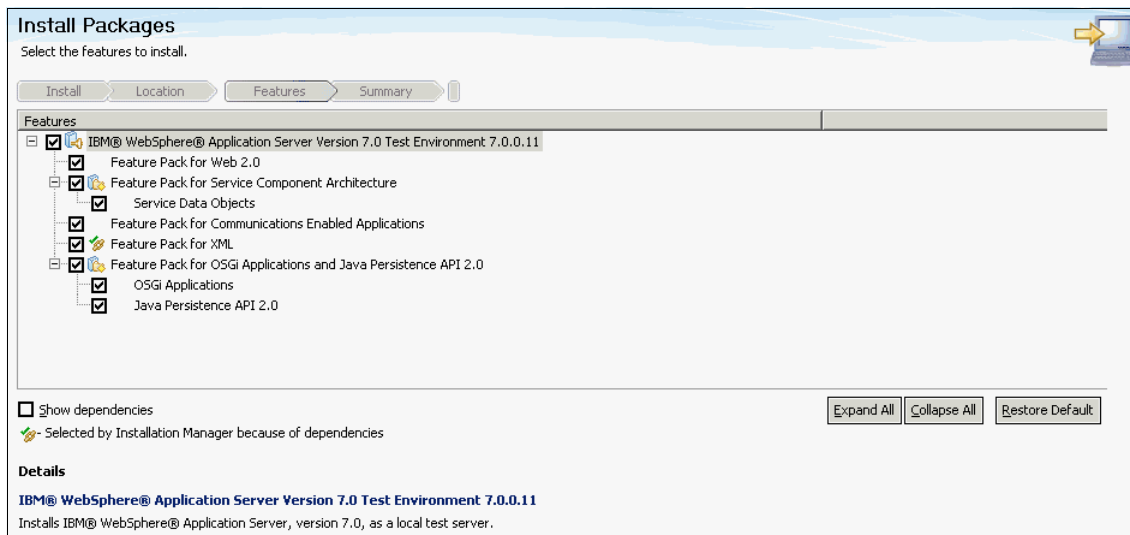


Figure A-13 Selecting WebSphere Application Server 7 feature packs

9. On the configuration page, enter the information to create a profile and select **Next** (Figure A-14).

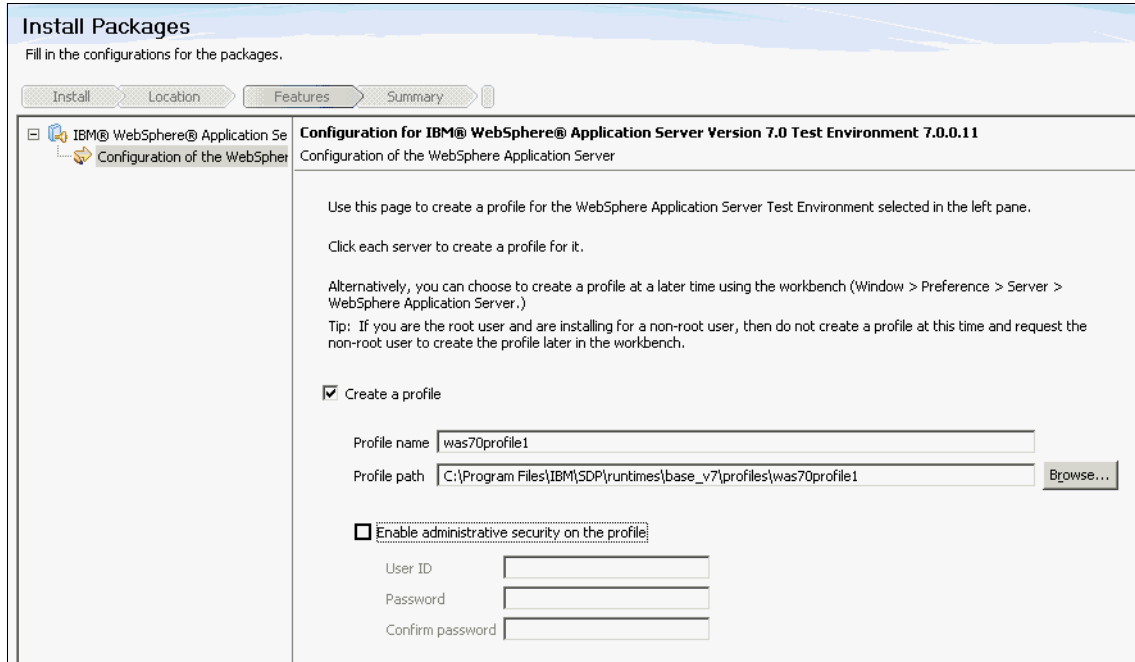


Figure A-14 Creating a profile for WebSphere Application Server 7

10. On the Summary page, review your choices before installing the product package. If you want to change the choices that you made on previous pages, click **Back**, and make your changes. When you are satisfied with your installation choices, click **Install** to install the package. A progress indicator shows the percentage of the installation completed.
11. When the installation process is complete, a message confirms the success of the process.
12. Close Installation Manager.

## Installing WebSphere Application Server V8 Beta

Perform these steps:

1. Change to the `RAD_SETUP` subdirectory of the directory where you extracted the “disks” for Rational Application Developer for WebSphere Software.
2. Start the Launchpad program: Run `launchpad.exe` (Figure A-1 on page 1785).
3. On the Launchpad dialog box, click **Install IBM Rational Application Developer V8.0**. IBM Installation Manager starts.

4. On the first page of the Install Packages wizard, select **Application Server 8.0.0.0** (Figure A-4 on page 1788) and then click **Next**.
5. You can install updates at the same time that you install the base product package. To search for updates to the packages, click **Check for Other Versions and Extensions**. Installation Manager searches for updates at the predefined IBM update repository for the product package. It also searches any repository locations that you have set. Click **Next**.
6. On the Licenses page, read the license agreements for the selected packages. On the left side of the License page, click each package version to display its license agreement. If you agree to the terms of all of the license agreements, click **I accept the terms of the license agreements**. Click **Next** to continue.
7. Depending on whether you have already used Installation Manager to install software, you might have to select the shared resources directory on the Location page. Type the path in the Shared Resources Directory field or accept the default path. Then click **Next** to continue.
8. On the Location page, type the path for the installation directory for the package group and then click **Next**. The name for the package group is created automatically. *You must install WebSphere Application Server V8 Beta in a separate package group* (Figure A-15).

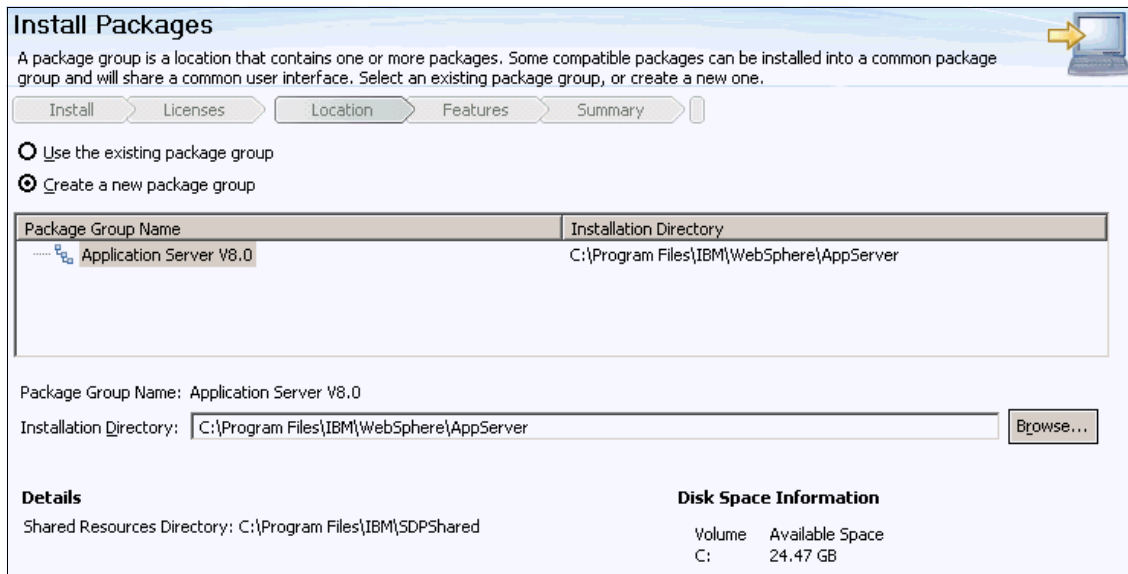


Figure A-15 Installing WebSphere Application Server 8 in a separate package

9. On the features page, select the features that you want to install. Click the feature to see more information about it in the Details section of the page. You



might need the Stand-alone thin clients for certain examples in this book. When you are finished, click **Next** (Figure A-16).

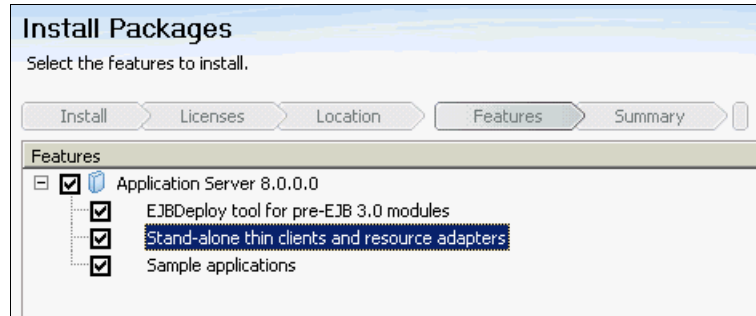


Figure A-16 Features for WebSphere Application Server 8

10. On the summary page, review your choices before installing the product package. If you want to change the choices that you made on previous pages, click **Back**, and make your changes. When you are satisfied with your installation choices, click **Install** to install the package. A progress indicator shows the percentage of the installation that has been completed.
11. When the installation process completes, select **Profile Management Tool to create a profile** (Figure A-17) and then click **Finish**.

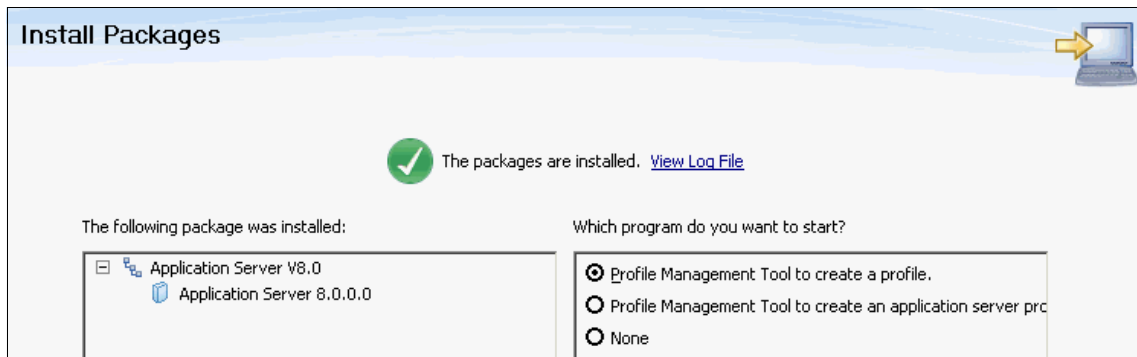


Figure A-17 Launching Profile Management Tool

## Creating a Profile for WebSphere Application Server V8 Beta

Perform these steps:

1. On the Welcome page of WebSphere Customization Tools 8.0, click **Launch Selected Tool** to select the Profile Management Tool (Figure A-18 on page 1798).

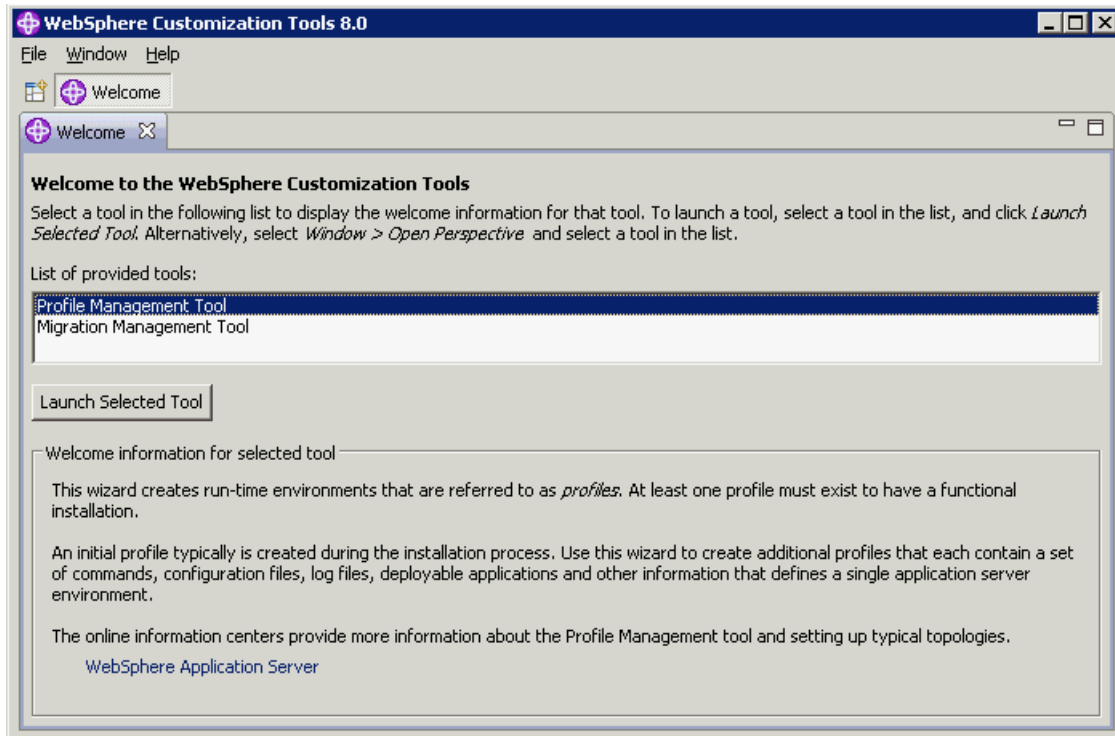


Figure A-18 Profile Management Tool

2. On the Profiles page, click **Create** (Figure A-19).

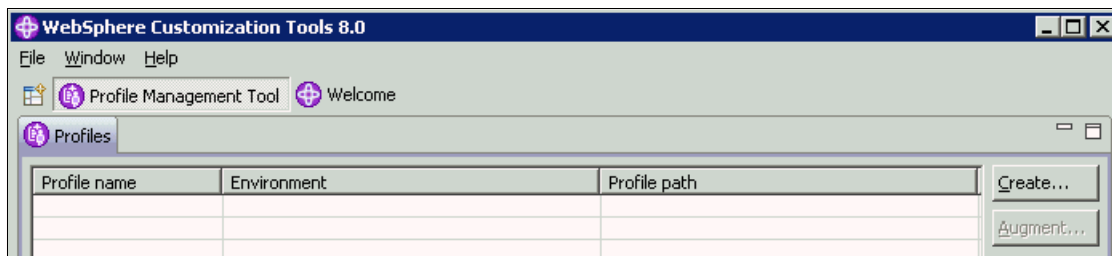


Figure A-19 Creating the profile

3. On the Environment Selection page, select **Application server** and then click **Next** (Figure A-20 on page 1799).

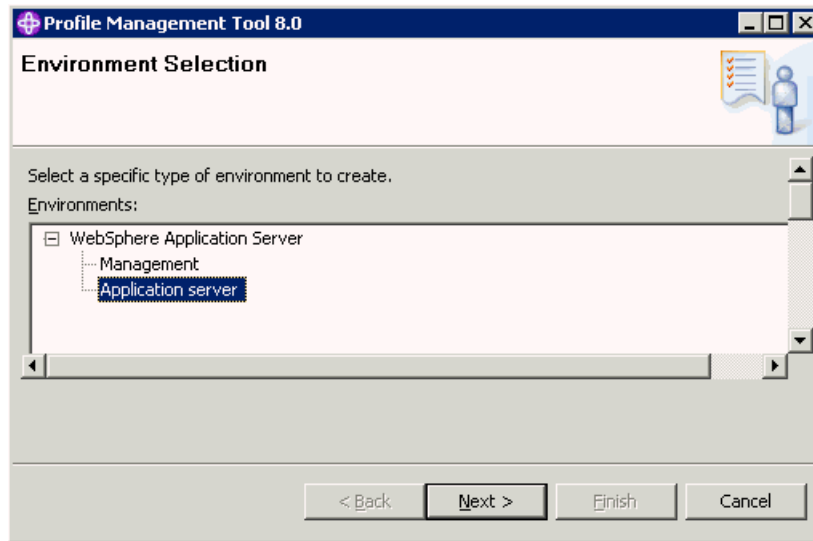


Figure A-20 Creating an Application server profile

4. On the Profile Creation Options page, select **Advanced profile creation** and then click **Next** (Figure A-21 on page 1800).

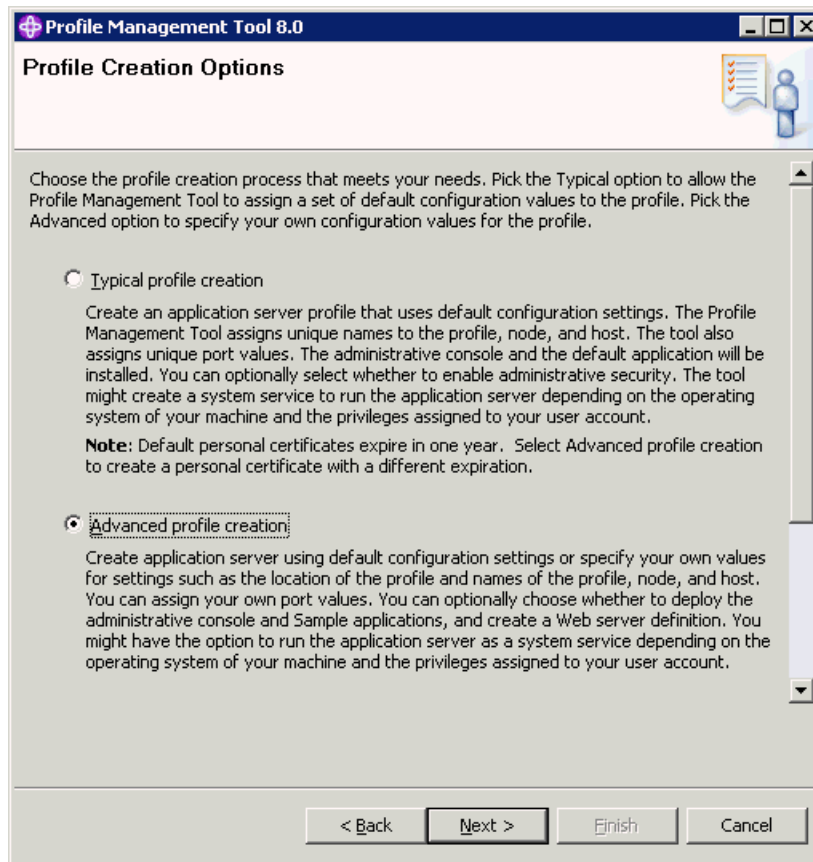


Figure A-21 Selecting Advanced profile creation

5. On the Optional Application Deployment page, select both **Deploy the administrative console** and **Deploy the default application** and then click **Next**.
6. Complete these tasks on the Profile Name and Location page:
  - a. For Server runtime performance tuning setting, select **Development**.
  - b. Type a Profile name and Profile directory or keep the default values.
  - c. Click **Next** (Figure A-22 on page 1801).

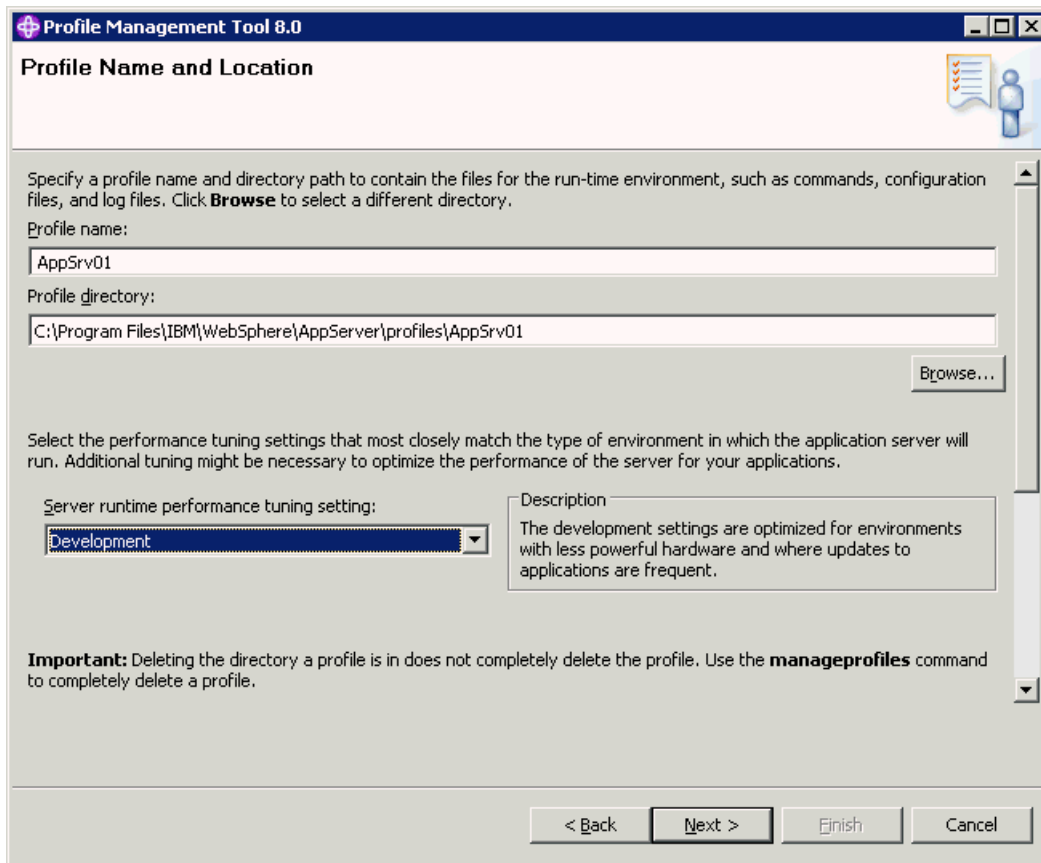


Figure A-22 Selecting Development for the Server runtime performance tuning setting

7. On the Node and Host names page, follow the instructions to enter the Node name, Server name, and Host name. Then click **Next**.
8. To enable security, select **Enable administrative security on the profile**. Enter the user ID and password in the corresponding fields and then click **Next**.
9. On the Security Certificate (part 1) page, select **Create a new default personal certificate** and select **Create a new root signing certificate** and then click **Next**.
10. On the Security Certificate (part 2) page, modify the certificate values or accept the default values.
11. On the Port Values Assignment page, click **Recommended port values**. The tool returns values for the ports that it does not detect are in use (Figure A-23 on page 1802). Click **Next**.

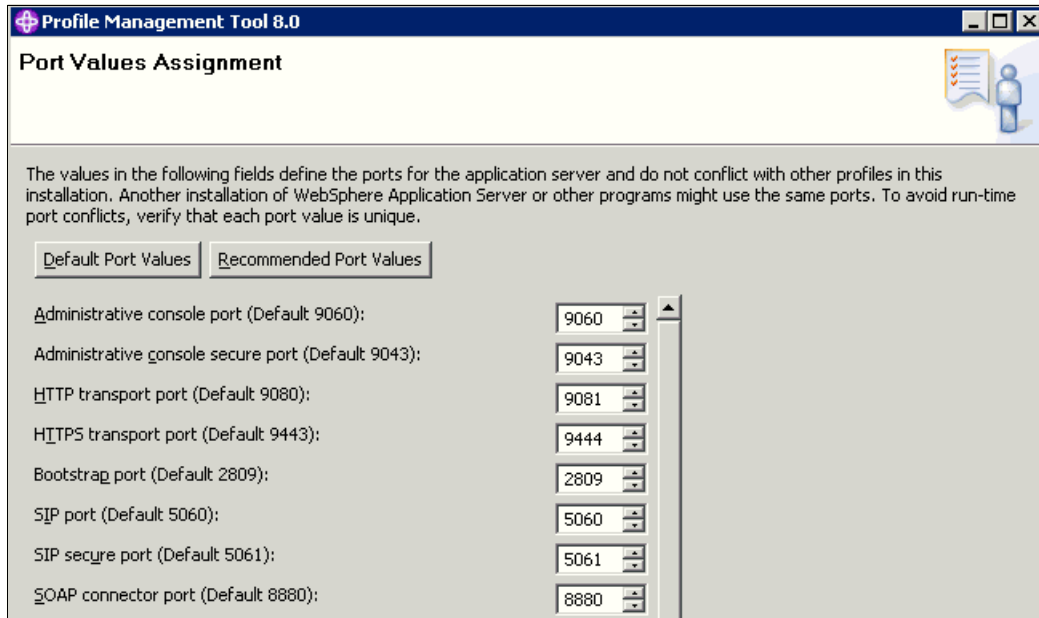


Figure A-23 Recommended port values

12. On the Windows Service Definition page, if you use Microsoft Windows, clear the selection "Run the application server process as a Windows service" (Figure A-24 on page 1803) and then click **Next**.

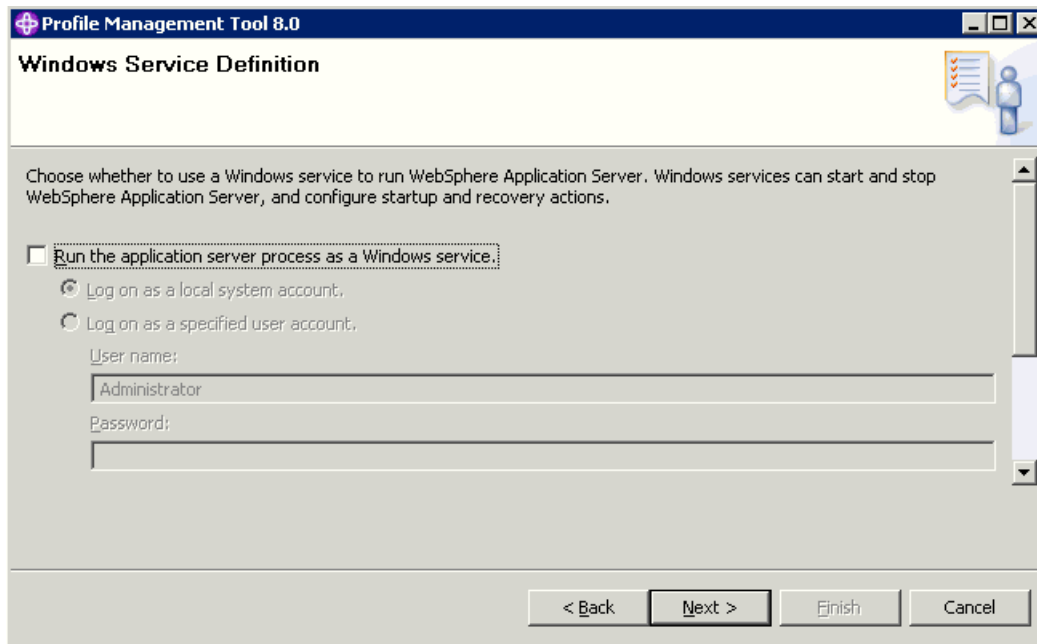


Figure A-24 Clear the option to run as a Windows service

13. Click **Next** on the Web Server Definition page.
14. Review the information on the Profile Creation summary page and then click **Create**.
15. When the profile creation process ends, click **Finish**.

## Installing the license for Rational Application Developer

You have the following options for enabling licensing for Rational Application Developer:

- ▶ Importing a product activation kit
- ▶ Enabling Rational Common Licensing to obtain access to floating license keys

In this section, we show you how to import a product activation kit:

1. Start IBM Installation Manager.
2. In the main window, click **Manage Licenses**.
3. In the Manage Licenses window, select **Application Developer Version 8** and **Import product Activation Kit**. Click **Next**.

4. In the Import Activation Kit window (Figure A-25), browse to the path of the download location for the kit, select the appropriate Java archive (JAR) file, and click **Open**. Click **Next**.

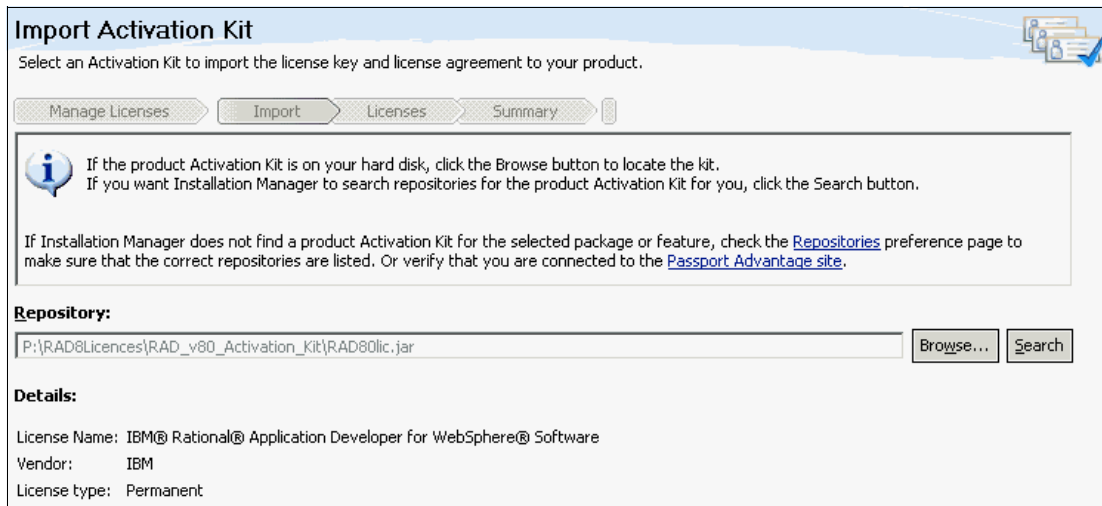


Figure A-25 Import Activation Kit

5. In the Licenses window, select **I accept the terms in the license agreements**. Click **Next**.
6. In the summary window, click **Finish**.

The product activation kit with its permanent license key is imported to Rational Application Developer. The Manage Licenses wizard indicates whether the import is successful.

**Upgrading the Floating License server is mandatory:** If you intend to use floating licenses, you must upgrade to IBM Rational License Key Server V8.1.1, which is described at this website:

<http://www-01.ibm.com/support/docview.wss?uid=swg24027295>

## Updating Rational Application Developer

After you install Rational Application Developer, the Installation Manager provides an interface to update the product. Perform these steps:

1. In the Installation Manager overview window, select **Update Packages**.



2. In the Update Packages window, select **Update all** to update all of the installed products, or select a specific product. Click **Next** to search for the updates to the selected products. Internet access is required unless your repository preferences point to a local update site. Each installed package has the location embedded for its default IBM update repository. For Installation Manager to search the IBM update repository locations for the installed packages, the preference **Search service repositories during installation and updates** on the Repositories preference page must be selected. This preference is selected by default.
3. Click **Update** to download and install the updates. A progress indicator shows the percentage of the installation that is completed.
4. Optional: When the update process completes, a message that confirms the success of the process is displayed near the top of the page. Click **View log file** to open the log file for the current session in a new window. You must close the Installation Log window to continue.
5. Click **Finish** to close the wizard.

## Uninstalling Rational Application Developer

You can uninstall Rational Application Developer interactively through the IBM Installation Manager. Perform these steps:

1. Before the uninstallation of any products, terminate the programs that you installed by using Installation Manager.
2. In the Installation Manager overview window, select **Uninstall**.
3. In the Uninstall Packages window, select the Rational Application Developer product package that you want to uninstall. Click **Next**.
4. In the Summary window, review the list of packages that will be uninstalled and click **Uninstall**.
5. In the Complete window that opens after the uninstallation finishes, click **Finish** to exit the wizard.

## Rational Desktop Connection Toolkit for Cloud Environments

To install Rational Desktop Connection Toolkit for Cloud Environments, complete these tasks:

1. Change to the *RAD\_SETUP* subdirectory of the directory where you extracted the “disks” for Rational Application Developer for WebSphere Software.
2. Start the Launchpad program: Run **1aunchpad.exe** (Figure A-1 on page 1785).

3. In the Launchpad dialog box window, click **Install IBM Rational Application Developer V8.0**. IBM Installation Manager starts.
4. On the first page of the Install Packages wizard, select **IBM Rational Desktop Connection Toolkit for Cloud Environments** and then click **Next**.
5. On the Licenses page, read the license agreement. If you agree to the terms of all of the license agreements, click **I accept the terms in the license agreements** and then click **Next**.
6. On the Features page, select any additional features that you want to install and then click **Next**.
7. On the summary page, review your choices before starting the installation process. If you want to change your selections, click **Back** to return to the previous pages. When you are satisfied with your installation choices, click **Install**.
8. When the installation process completes, click **Finish**.
9. Close Installation Manager.

## Installing WebSphere Portal V7

In this section, we explain how to install WebSphere Portal V7, add it to Rational Application Developer, and configure the portal test environment for performance.

### Installing WebSphere Portal V7

If you have already installed WebSphere Portal V7 on the same machine where you are about to install Rational Application Developer, the installation wizard of Rational Application Developer automatically integrates the installed WebSphere Portal Server as a target run time.

Many clients install the WebSphere Portal server after installing Rational Application Developer. However, as of this release, the Launchpad for Rational Application Developer no longer offers the capability to install WebSphere Portal server. You must therefore download WebSphere Portal Server separately from IBM Passport Advantage®. The information at the following website describes which files you need to download:

<http://www-01.ibm.com/support/docview.wss?uid=swg21446742>

Also, you can find more information about the options for the installation procedure in the Lotus® Wiki for Portal Server 7:

[http://www-10.lotus.com/1dd/portalwiki.nsf/dx/Setting\\_up\\_a\\_standalone\\_server\\_on\\_Windows\\_wp7](http://www-10.lotus.com/1dd/portalwiki.nsf/dx/Setting_up_a_standalone_server_on_Windows_wp7)

Next you must extract all of these compressed files to the *same directory* to avoid the decompression utility creating subfolders that are named after the compressed files, such as CZLM1ML.

Perform these steps:

1. Open a command prompt and execute **install.bat**.
2. You see the installation wizard that is shown in Figure A-26. Click **Next**.

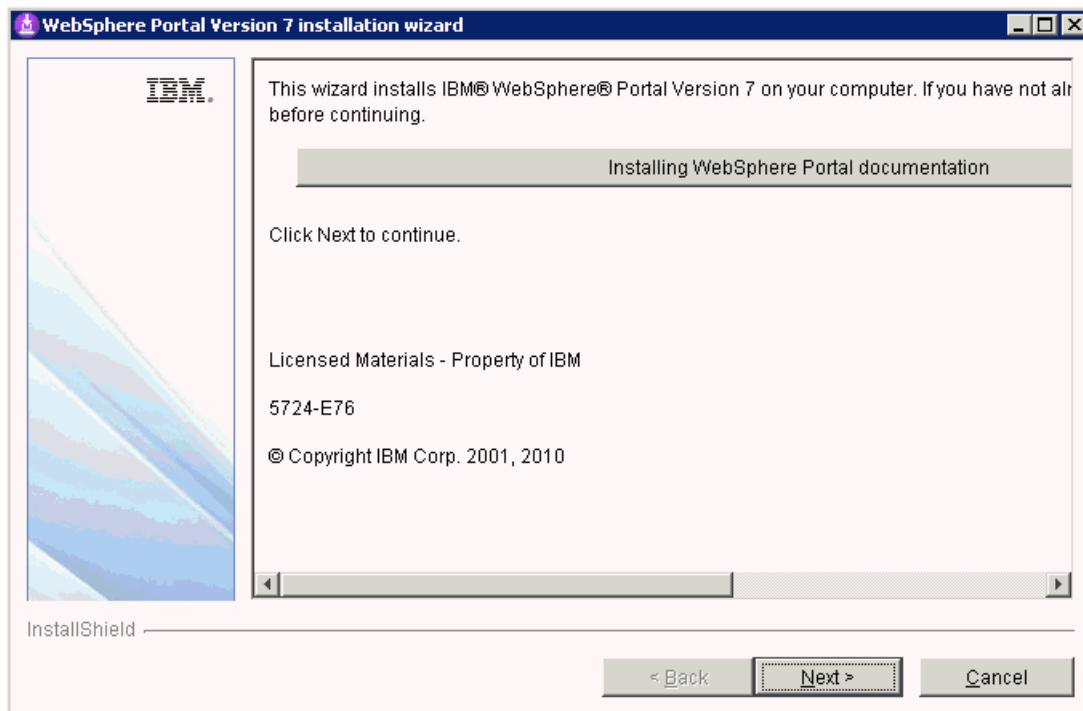


Figure A-26 Installation wizard (install.bat)

3. In the Software license agreement window, if you agree to the terms, click **Next**.
4. For installation type, select **Base** (Figure A-27 on page 1808).

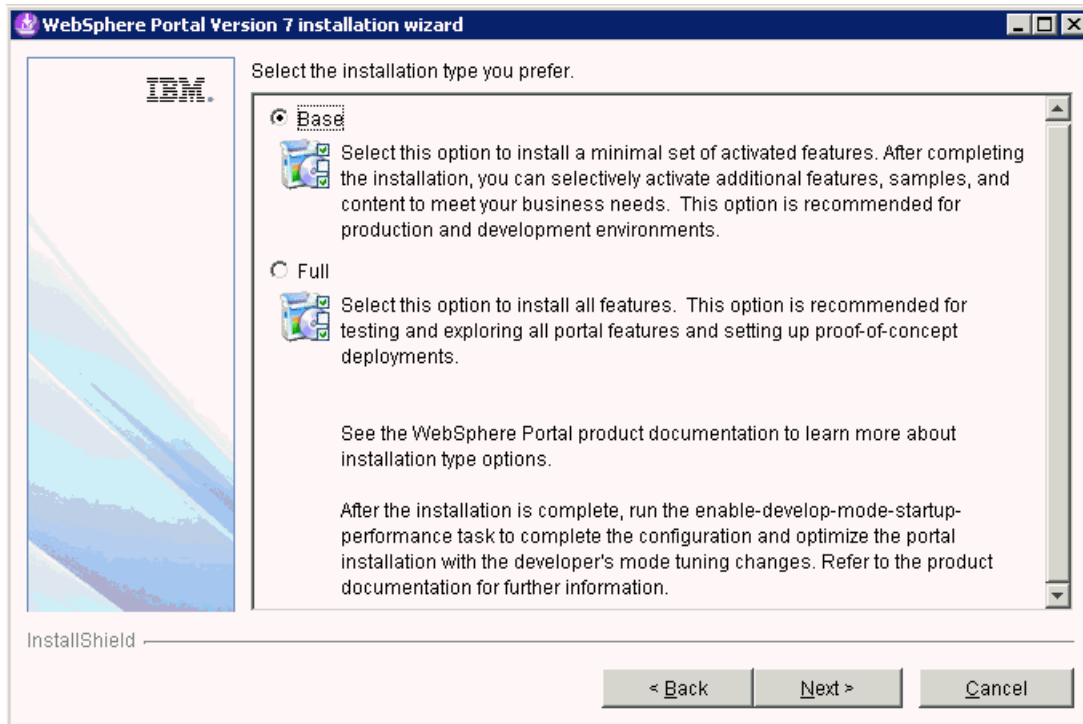


Figure A-27 Selecting the Base installation type

5. In the next window, specify the WebSphere Portal installation directory. Accept the default (**C:\IBM\WebSphere**) and click **Next**. See Figure A-28 on page 1809.

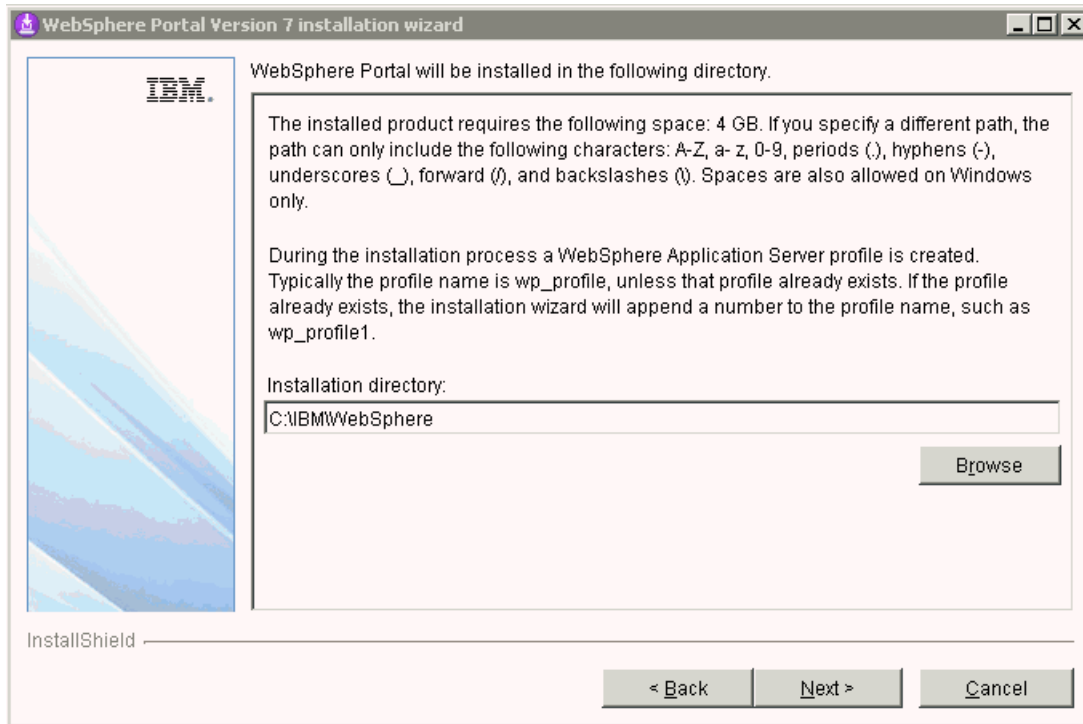


Figure A-28 Selecting the installation directory

6. In the next window, the installation wizard determines the node name based on the host name and detects the fully qualified host name. There are restrictions on the possible values for the host name, as shown in Figure A-29 on page 1810.

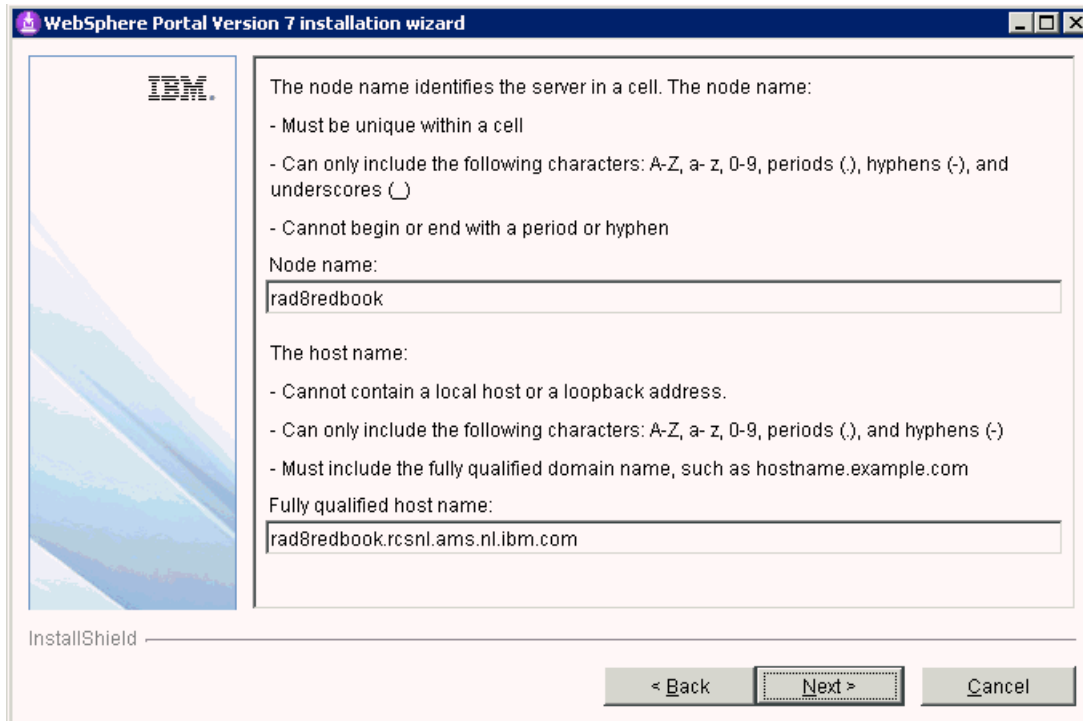


Figure A-29 The installation wizard detects the node name and the fully qualified host name

7. Select **Install on top of an existing instance** and click **Next**.
8. In the next window, enter the administrative user and password, which are the same for WebSphere Portal Server and for the underlying WebSphere Application Server (in this example, we chose `wpsadmin/wpsadmin`), and click **Next**. See Figure A-30 on page 1811.

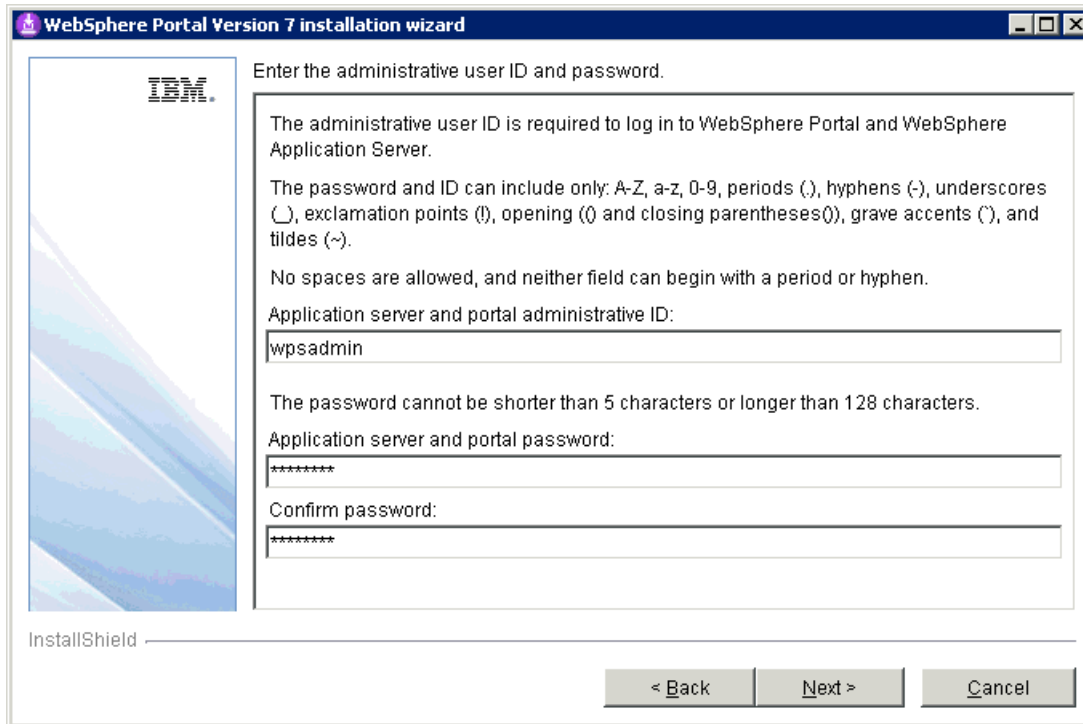


Figure A-30 Entering the administrative user ID and password

9. In the next window, decide whether you want to create Windows services to start and stop WebSphere Portal Server and additional actions (Figure A-31) and click **Next**.

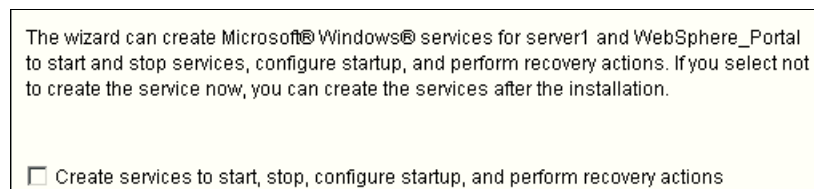


Figure A-31 Decide whether to create Windows services

10. In the next window, you see the summary of the installation configuration (Figure A-32 on page 1812). Read and verify the details. If the details are as you want them, click **Next** to start the installation process. This process can take a long time (up to two hours), depending on the machine resources.

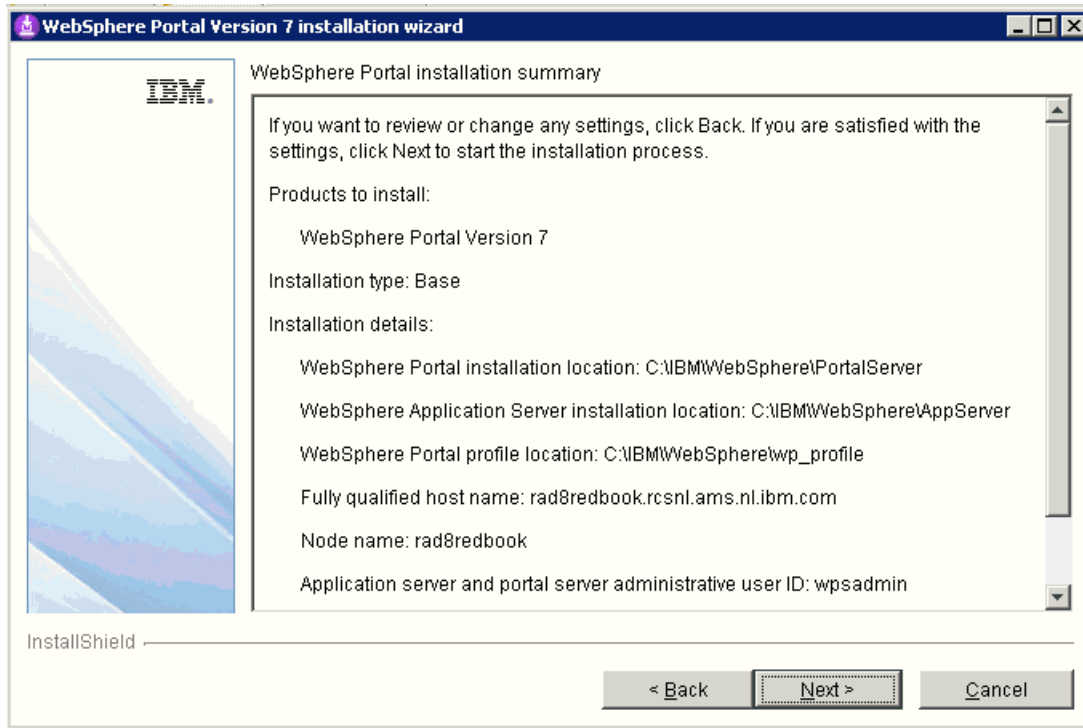


Figure A-32 Summary of the installation configuration

11. In the next window, you see a message stating the location of the installation log file. It is a good idea to document the installation log file location for future reference. In our example, it was C:\Documents and Settings\Administrator\Local settings\Temp\1\wpinstalllog.txt (Figure A-33).

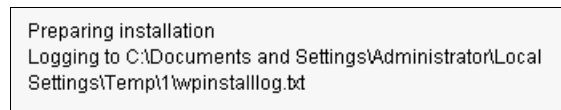


Figure A-33 Location of installation log file

12. If you see a message similar to the following message, you have not downloaded all the required parts. Determine the missing part and download it from Passport Advantage (Figure A-34 on page 1813).



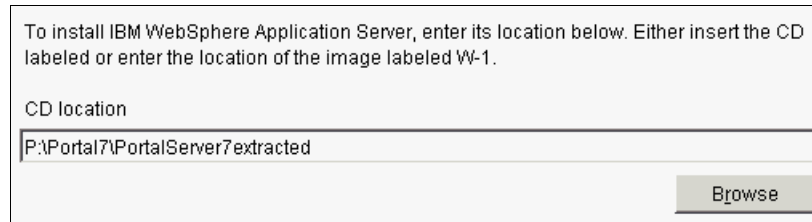


Figure A-34 Message indicating that one part is missing

13. In a successful installation, you see a window, as shown in Figure A-35. If you click **Open**, you see a window where the log streams.

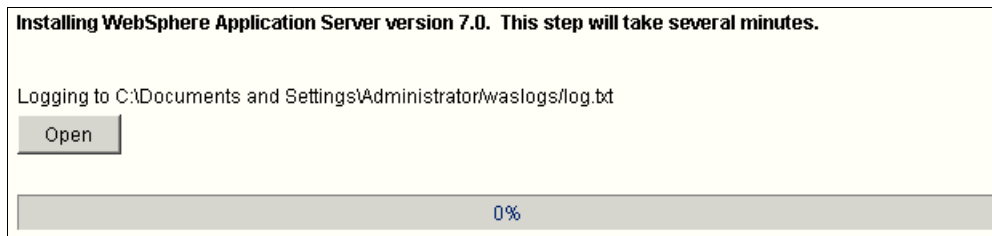


Figure A-35 Progress bar for installation

14. When the wizard completes the product installation and you see the final summary window, click **Finish**.

## Adding WebSphere Portal V7 to Rational Application Developer

To execute the portal and portlet applications, you must create a new server in Rational Application Developer for the installed WebSphere Portal Server as the target run time. You need to create this new server in each new workspace from which you want to access WebSphere Portal Server. However, upon start-up, Rational Application Developer detects the presence of the installed WebSphere Portal Server and preconfigures it in **Windows** → **Preferences, Server** → **WebSphere Application Server**, as shown in Figure A-36 on page 1814.

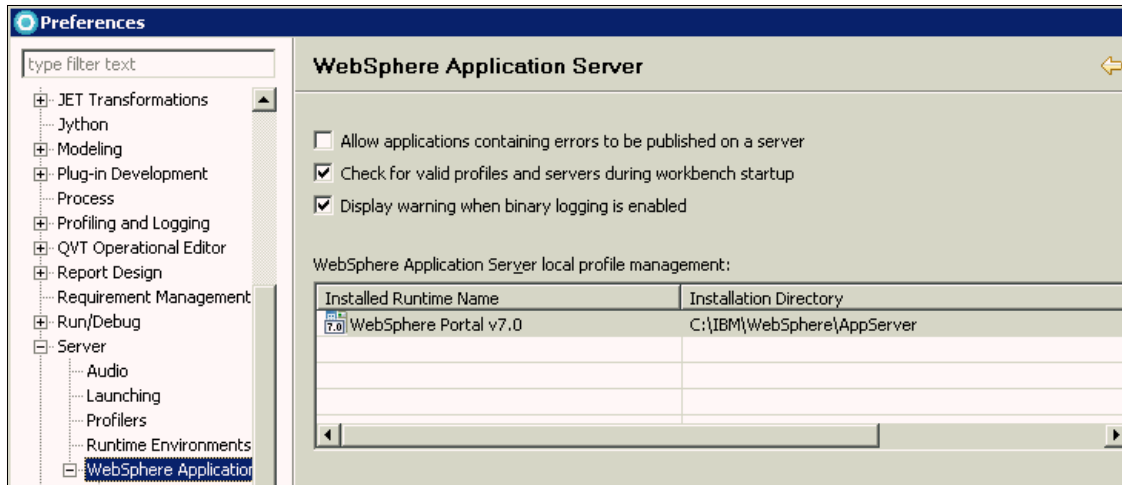


Figure A-36 Detecting the presence of the installed WebSphere Portal Server

Complete these steps to configure a new server:

1. Start Rational Application Developer.
2. In the Servers view window, right-click and select **New** → **Server**.
3. In the New Server window (Figure A-37 on page 1815), for server type, select **WebSphere Portal v7.0 Server** and click **Configure runtime environments** to verify if the installation wizard has configured the newly installed WebSphere Portal Server correctly.

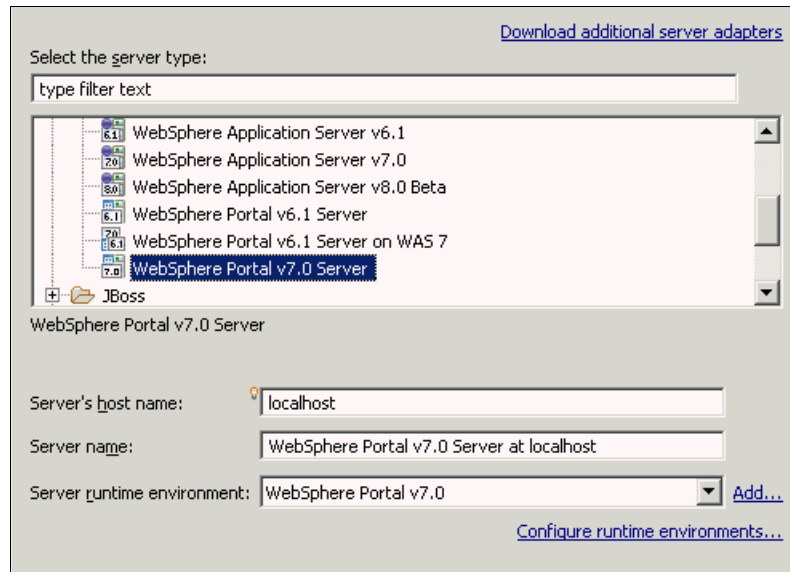


Figure A-37 Defining the new server window

4. In the New Server window, click **Next**.

5. In the WebSphere Settings window (Figure A-38), accept the selection for connection settings, retype the password `wpsadmin`, and click **Next**.

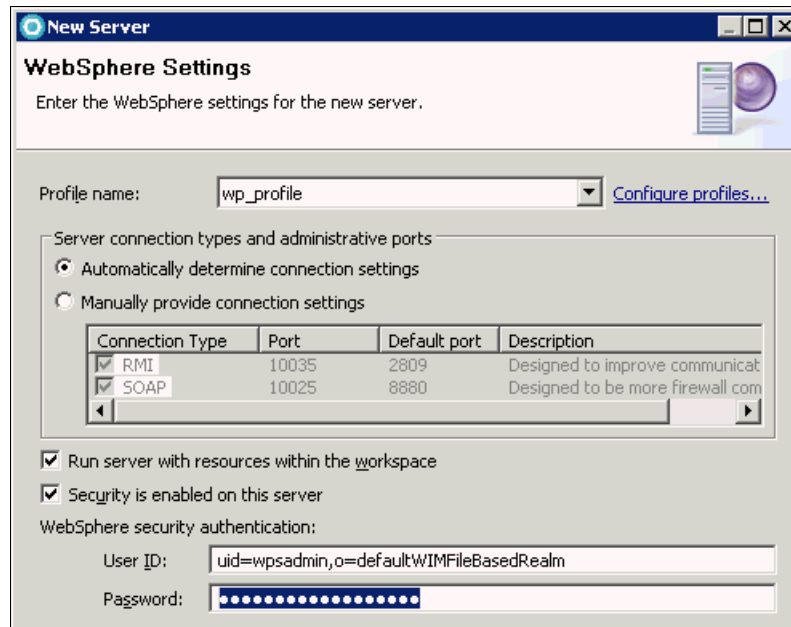


Figure A-38 Entering the password again in the WebSphere settings window

6. In the WebSphere Portal Settings window (Figure A-39 on page 1817), perform these steps:
  - a. Verify the portal settings, such as context root, default home, and personalized home, and the installation location of WebSphere Portal Server.
  - b. Enter the user ID and password for the WebSphere Portal Server administrator, for example, `wpsadmin/wpsadmin` (even if you think that they are already entered).
  - c. Decide if you want to enable the automatic login of a particular user when the WebSphere Portal test environment starts.
  - d. Click **Next**.

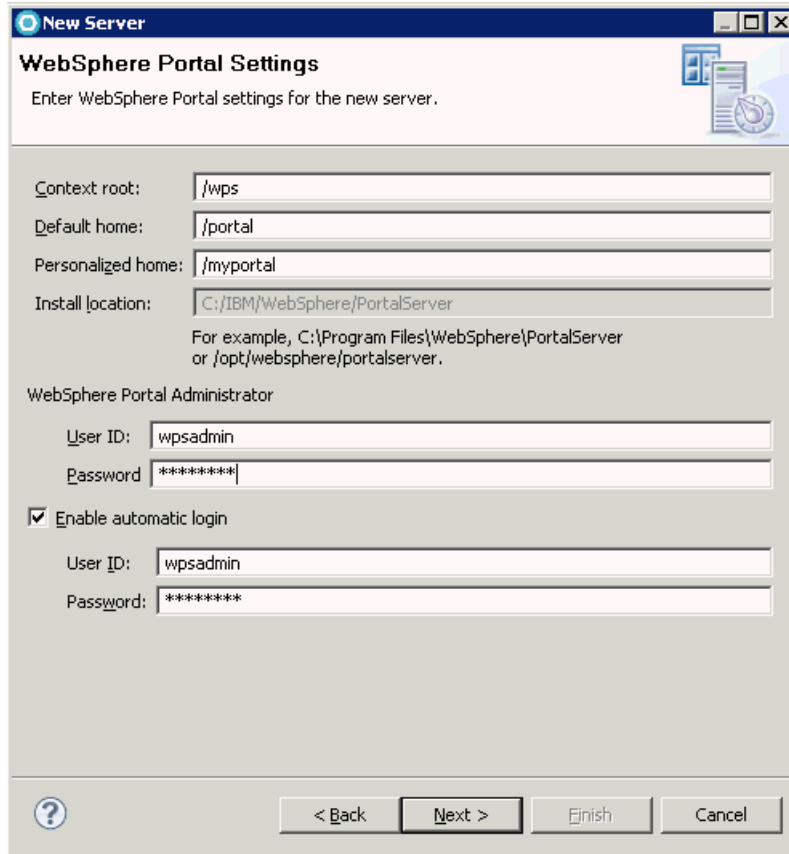


Figure A-39 WebSphere Portal settings

7. To ensure that the correct passwords are used in Rational Application Developer at all times, change the default passwords in the file `wkplc.properties` by changing the default values of the properties `WasPassword` and `PortalAdminPwd`, as described at this website:  
[http://www-10.lotus.com/ldd/portalwiki.nsf/dx/wkplc.properties\\_file\\_reference\\_wp7](http://www-10.lotus.com/ldd/portalwiki.nsf/dx/wkplc.properties_file_reference_wp7)
8. In the Properties Publishing Settings window (Figure A-40 on page 1818), select the default value of **Local Copy** for the transfer method. Click **Next**.

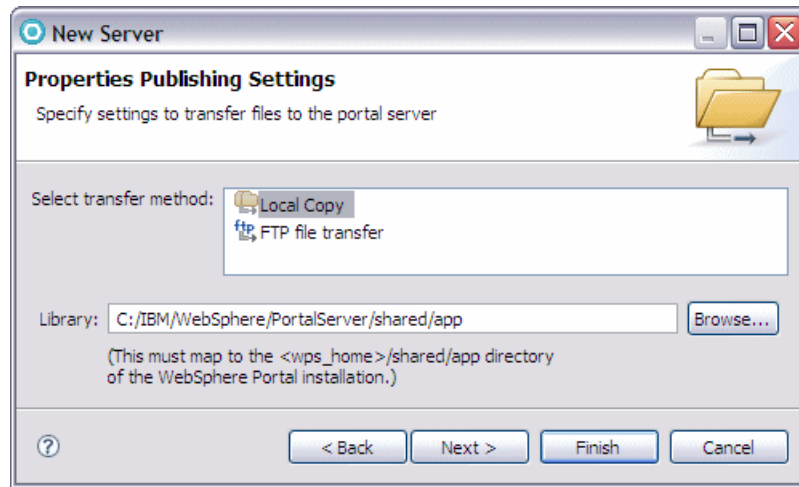


Figure A-40 Properties Publishing Settings

9. In the Add and Remove Projects window, click **Next**. You do not have any portal or portlet projects to add to this server.
10. In the Tasks window, click **Finish** to install a new test server.

## Optimizing the WebSphere Portal Server for development

To optimize the WebSphere Portal Server for development and to improve the start-up performance, run the task that is described in the Rational Application Developer Information Center at the following Web address:

<http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/com.ibm.portal.doc/topics/tenabledevmode.html>

The new context menu *Enable Dev Mode* in the Server view in Rational Application Developer is equivalent to running the **ConfigEngine** command in `<Portal_Profile-HOME>/ConfigEngine:`

```
ConfigEngine.bat enable-develop-mode-startup-performance
```

If you do not see the Enable Dev Mode context menu, enable the following capability:

1. Select **Windows** → **Preferences**.
2. Select **General** → **Capabilities**.
3. Select **Advanced**.
4. Select **Web Developer (advanced)** → **Portal development**.

In the console, you can see that the script ends with entries similar to Example A-1.

*Example: A-1 End of execution of Enable Dev Mode*

---

```
BUILD SUCCESSFUL
Total time: 1 minute 55 seconds
isIseries currently set to: null
uploading registry
CELL: rad8redbook
NODE: rad8redbook
Websphere:_Websphere_Config_Data_Type=Registry,_Websphere_Config_Data_I
d=cells/rad8redbook|registry.xml#Registry_1288357406007,_WEBSHERE_CONF
IG_SESSION=anonymous1288373787851
update-registry-sync-property:
Fri Oct 29 13:36:28 EDT 2010
 [echo] updated RegistrySynchronized in file wkplc.properties with
value: true
Return Value: 0
[wplc-modify-ear-lazy-load-impl] 47 PA_Clients_Manager
Target Server: WebSphere_Portal, MappEnable Old: true, New: false
```

---

For potential issues related to enabling development mode, see the following technote (web doc):

<http://www-01.ibm.com/support/docview.wss?uid=swg21316233>

## Verifying development mode

In Rational Application Developer, follow these steps:

1. Right-click the WebSphere Portal Server and select **Start** (or click the **Start** icon) to start the server. It takes a while to start the WebSphere Portal Server. Wait until it changes its status to Started and its state to Synchronized.
2. Right-click the portal server entry in the Servers view and select **Administration** → **Run administrative console**.
3. In the browser session window that opens with the administrative console of the WebSphere Portal Server, enter the user ID and password of wpsadmin.
4. Navigate to **Server** → **Server Types** → **WebSphere Application Servers** → **WebSphere\_Portal**.
5. Verify that **Run in development mode** is selected (Figure A-41 on page 1820). If you make changes, click **Apply** and then click **Save** to apply the changes to the master configuration.

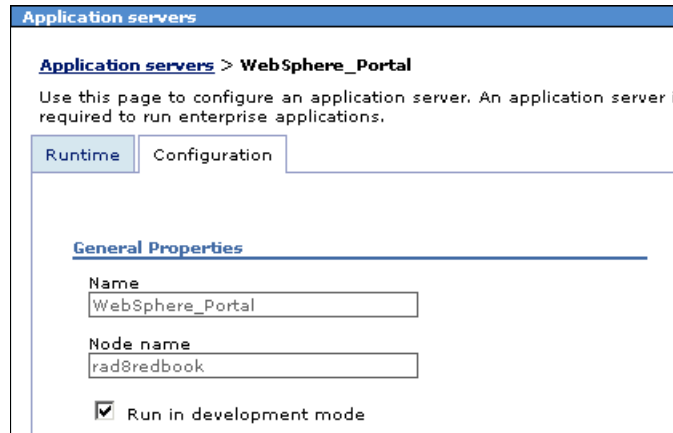


Figure A-41 Verifying that Run in development mode is selected

## Defining remote servers for testing portals

Whether you plan to use the local machine or a remote server for testing and debugging, you can use the WebSphere Portal server node in the New Servers wizard to set up connections to either of the test environments. Perform these steps to create and configure a remote portal server:

1. Switch to the Servers view by selecting **Window** → **Show View** → **Servers**.
2. In the Servers view window, right-click and select **New** → **Server**.
3. Select the appropriate WebSphere Portal server from the server-type list.
4. Provide the host name of the remote WebSphere Portal server. Click **Next**.
5. On the WebSphere Settings page, define the following options:
  - a. Set the RMI Port if you want to use this connection type. Clear the check box if you do not need it.
  - b. Set the SOAP Port if you want to use this connection type. Clear the check box if you do not need it.
  - c. Select **Security is enabled on this server** and specify the administrator ID and password for the Portal server.
6. Define the following settings and click **Next**:
  - a. Define the Context Root. The default is /wps.
  - b. Define the Default Home. The default is /portal.
  - c. Define the Personalized Home. The default is /myportal.



- d. Define the Install location, which is the WebSphere Portal installation root. For example, the directory on the target server might be C:\Program Files\WebSphere\PortalServer; however, if the *<portal home>* directory on the target server has been mapped as a network drive, the path to the *<portal home>* directory might be E:\Portalserver. Consult your systems administrator for the exact path, and specify this information only when you use the deploy action on portal or portlet projects.
  - e. Specify the WebSphere Portal Administrator User ID and password.
  - f. If you select to enable automatic login, provide a user ID and password for a WebSphere Portal user. You must create the user on the Portal Administration page on the remote server before testing or debugging. Edit permission is automatically assigned to the user and the user ID is used as part of the label. To use a single WebSphere Portal server for multiple users, use a separate user ID for each person.
7. Select **Enable the server to start remotely** if you want to start and stop the server. If you select **Enable the server to start remotely**, all the input options on the page are enabled. Define the following options:
- a. Define the server platform as Microsoft Windows or Linux.
  - b. Specify the server profile path, for Microsoft Windows, the path is \\<host name>\<drive>\$\<PathToProfile>. For Linux, the path is /opt/IBM/wp\_profile.
  - c. Select one of the following system logon methods:
    - Operating system logon: Specify the username and the password.
    - Secure shell (SSH): Specify the private key file, user ID, and the passphrase.
- All the input options stay disabled if you do not select “Enable the server to start remotely”.
8. Click **Next** to open the Properties Publishing Settings page and define the following options:
- a. Select the **Local Copy** or **FTP file transfer** option.
  - b. Provide the Library directory, which maps to the shared/app directory of the WebSphere Portal server on the remote system. For example, C:\WebSphere\PortalServer\shared\app might be the directory on the target server. However, if the shared/app directory on the target server has been mapped as a network drive, the path to the shared/app directory might be similar to E:\sharedApp. Or, if you use FTP access and the FTP root for the user that you specify is the C:\WebSphere directory, the path might be similar to PortalServer/shared/app. Consult your Systems Administrator for the exact path.

- c. If you select the FTP option, provide the following information:
  - i. A user ID and password combination that has FTP access to the target system.
  - ii. The connection timeout value if the default of 10000 milliseconds is not adequate.
  - iii. If the target server is beyond a firewall, select **Use PASV** mode or **Use firewall**. You can set additional firewall settings by clicking **Firewall settings**.
9. Click **Next**. On the Add and Remove Projects page, select one or more projects and select **Add** or select **Remove** to associate or disassociate the project with the server. During publishing, all projects that are associated with the publishing server are deployed.
10. Click **Finish**.
11. The server instance for the remote server is defined. Right-click the instance and select **Start**.

## Defining page creation settings

You can define your page creation settings and the page creation aggregation mode by using a server editor. The default mode for a portlet is Server Side Aggregation (SSA), and for an iWidget, the mode is Client Side Aggregation (CSA). The default mode for portlet and iWidget together is CSA mode. Perform these steps to define page creation settings:

1. Create an instance of WebSphere Portal V7.0.
2. Double-click the server instance to open server editor.
3. Select the **Portal** tab and expand the **Page Creation Settings** tab.
4. Select the artifact type and render mode for that artifact.
5. Select **CSA Mode** for client-side aggregation and **SSA Mode** for server-side aggregation.
6. Save the settings and publish the project to the server.

A page is created and the server home page opens after publishing completes. Go to **Administration** → **Manage Pages** and navigate to the page that got created for your portlet. Click **Page Properties** to view the render mode of the page.

## Enabling the debugging service

If you use a remote WebSphere Portal Server without using the Rational tools, and if you intend to use this WebSphere Portal Server for debugging purposes, you might want to enable the debugging service for this server during its start-up process. Perform these steps:

1. Select **Server** → **Server Types** → **WebSphere Application Servers** → **WebSphere\_Portal** → **Debugging Service (under Additional properties)**.
2. Select **Enable service at server startup**.
3. Click **Apply** and then click **Save** to apply changes to the master configuration (Figure A-42).

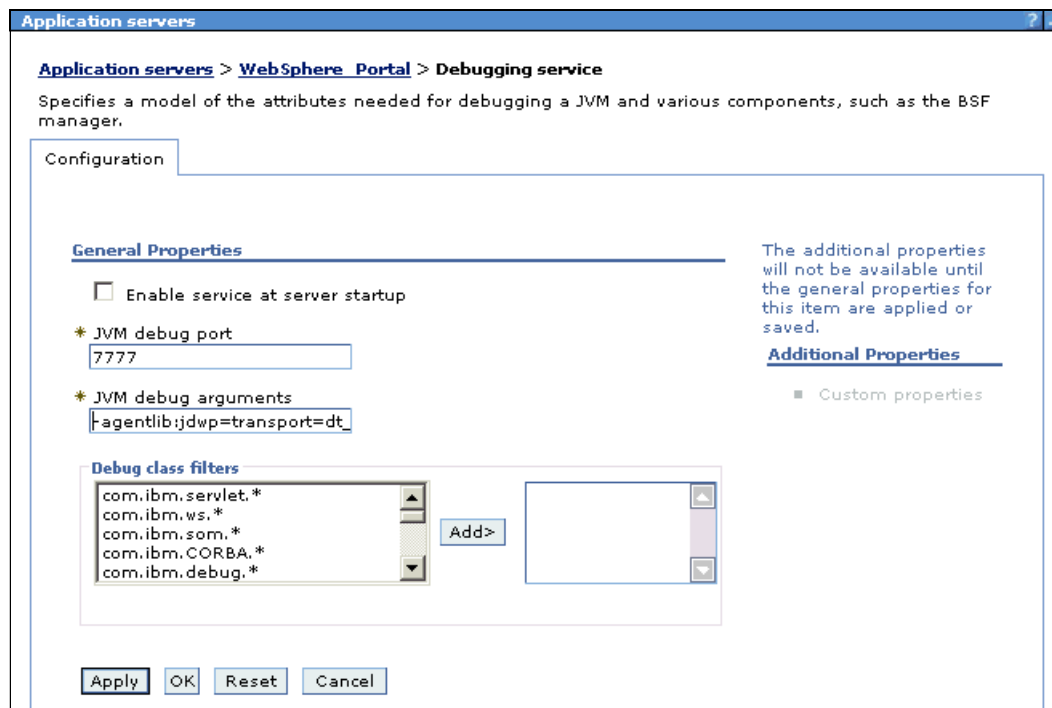


Figure A-42 Enabling Debugging Service on Portal server

## Stopping the server

You have completed the installation and configuration. Now you can stop the server.

In addition to these changes, see 21.5, “More information” on page 1181, for more tips to help you increase the performance of the WebSphere Portal Server during the development mode or to reduce its start-up time.

## Installing IBM Rational Team Concert

You can find the complete instructions for installing any of the three editions of Rational Team Concert in the product help under **Installing and upgrading** → **Installing Rational Team Concert**.

### Installing Rational Team Concert Standard Edition server

In this section, we show how you can get started with the installation of Rational Team Concert Standard Edition on Microsoft Windows 32-bit systems. For other editions and platforms, see the product help. You can obtain IBM Rational Team Concert from the following website:

For this chapter, we installed the following version:

<https://jazz.net/downloads/rational-team-concert/releases/2.0.0.2iFix4>

**No charge for small teams:** For a small team of up to 10 developers, you can download Rational Team Concert Express-C at no charge.

You are required to register to access the download page.

If you want to install Rational Team Concert Standard Edition on a computer that already has IBM Installation Manager installed, perform these steps:

1. Select **Other Download Options** → **Installation Manager** → **Server and Optional Features (Local Install)** to initiate the download of the file called `RTC-Standard-Full-2.0.0.2iFix4-Win32-Local.zip`.
2. After you download the file, extract the archive to a directory on your file system, such as `C:\RTC-Standard-Full-2.0.0.2iFix4`.
3. From a command prompt, launch the `C:\RTC-Standard-Full-2.0.0.2iFix4\launchpad.exe` executable, which is shown in Figure A-43 on page 1825.
4. Select the **Jazz Team Server Standard Edition** link. You can choose to install as Administrator or not. In this example, the installation was performed as Administrator. The Launchpad detects that Installation Manager is already installed and launches Installation Manager.

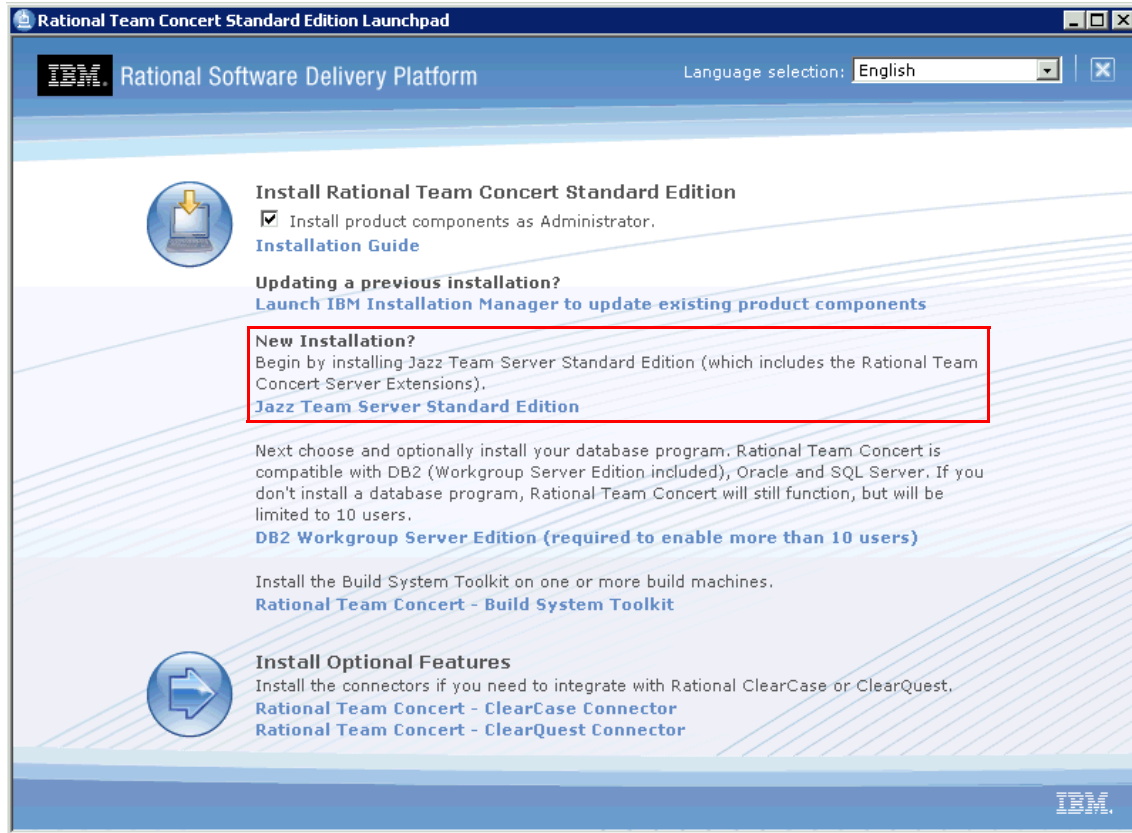


Figure A-43 Rational Team Concert Standard Edition Launchpad

5. Installation Manager shows the package to install (Figure A-44). Select **Next**.

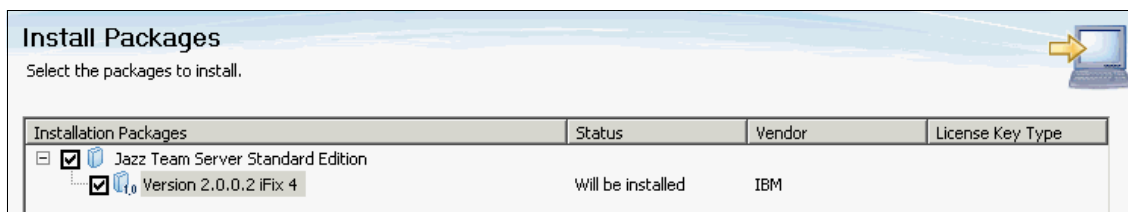


Figure A-44 Selecting the package to install

6. On the next page, you are prompted to accept the License Agreement. If you agree, select **I agree** and **Next**.
7. On the next page, you are prompted to create a New Package Group, which is required. If you have previously installed Rational Application Developer, the two products must be in separate package groups. However, if you have

already installed any other products, the shared resources directory is already set and cannot be changed. If this is the first product that you install, you also are prompted to choose the Shared Resources Directory (Figure A-45). Accept the default and select **Next**.

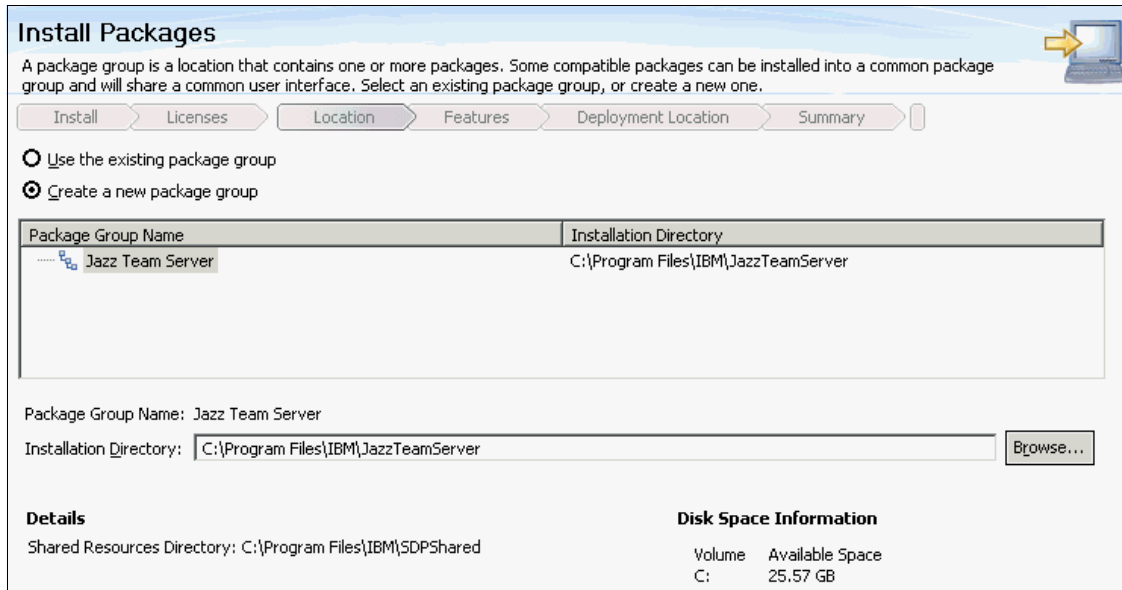


Figure A-45 Creating a new package group

8. On the next page, select the translations to install and click **Next**.
9. On the next page, you can select the features to install. Accept the default, and select **Next** (Figure A-46).

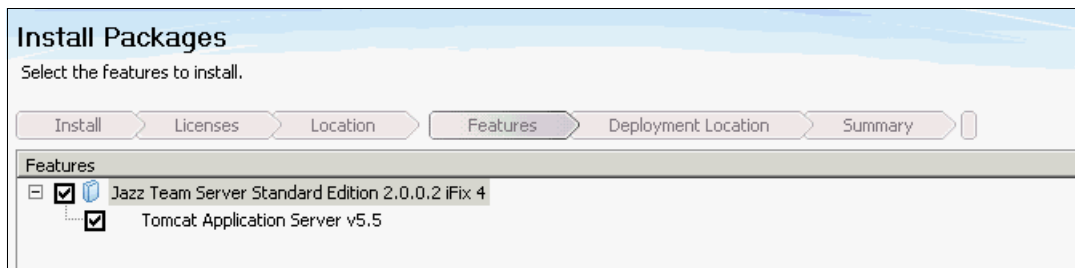


Figure A-46 Selecting the features to install

10. On the next page, review the summary and select **Install**.

11. The installation completes, except to start the Standard Edition Setup Guide, which is an HTML page describing how to set up the server. Select **Finish**, which stops Installation Manager and launches the Standard Edition Setup Guide (Figure A-47).

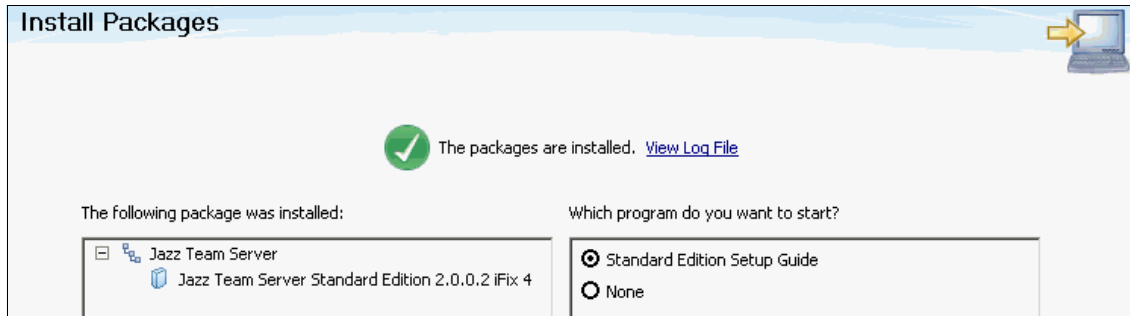


Figure A-47 Launching the Standard Edition Setup Guide

12. Start the provided Apache Tomcat server. For instructions to change the default server ports and the default launch directory, see the product help. Select **All Programs** → **Jazz Team Server** → **Start Jazz Team Server** to start the server.

## Running the setup wizard

To configure the team server, perform these steps:

1. Point a web browser to the following web address:  
`https://localhost:9443/jazz/setup`

2. After entering ADMIN as the default for both the Username and Password, the window that is shown in Figure A-48 opens.

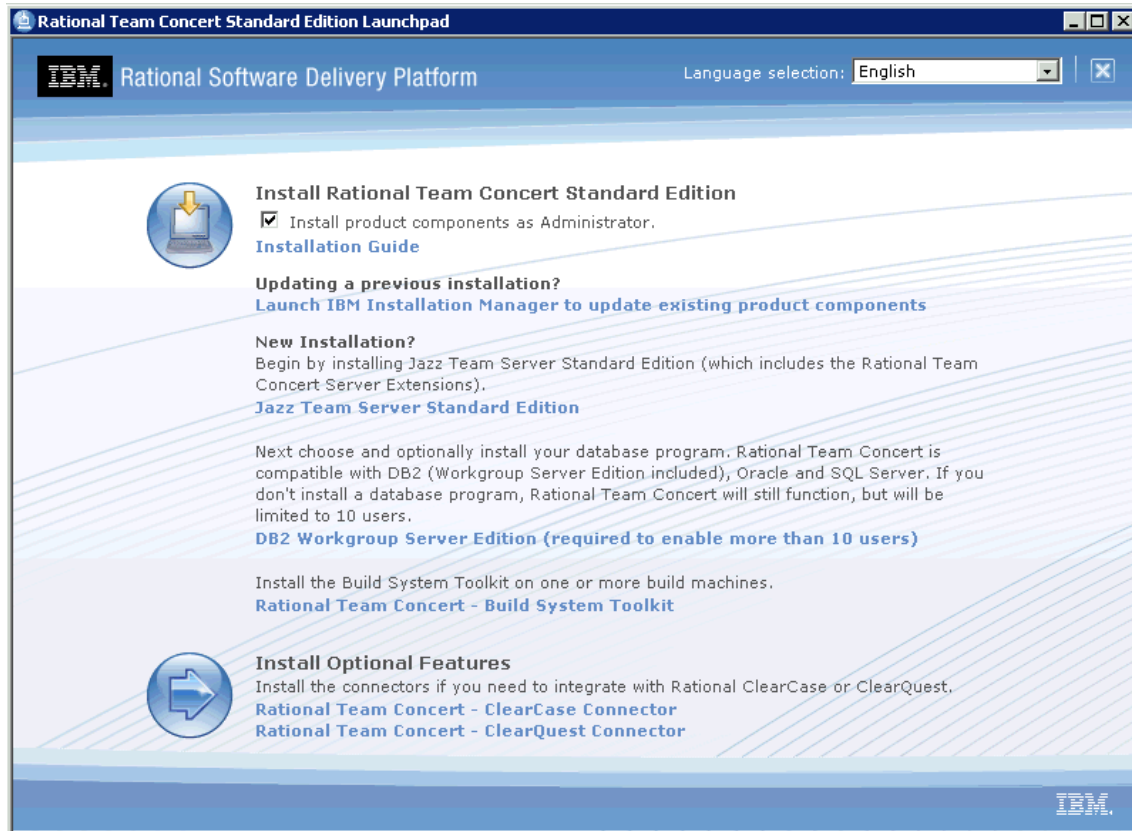


Figure A-48 Running the setup after starting Tomcat



- To select a setup path, click **Fast Path Setup**, which skips the configuration of the mail server for email notification and uses the built-in Derby database (Figure A-49). See the product help for information about using the Custom Setup option.

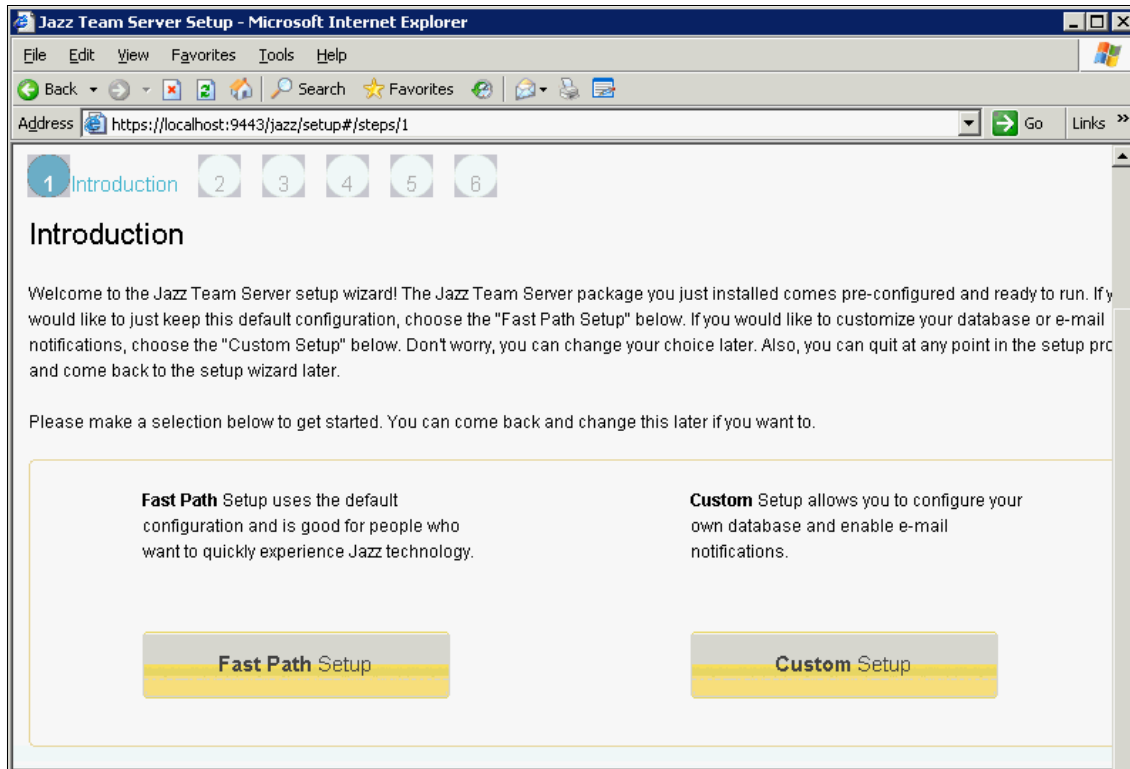


Figure A-49 Selecting Fast Path Setup

4. On the Setup User Registry page (Step 1), select **Tomcat User Database** (Figure A-50).

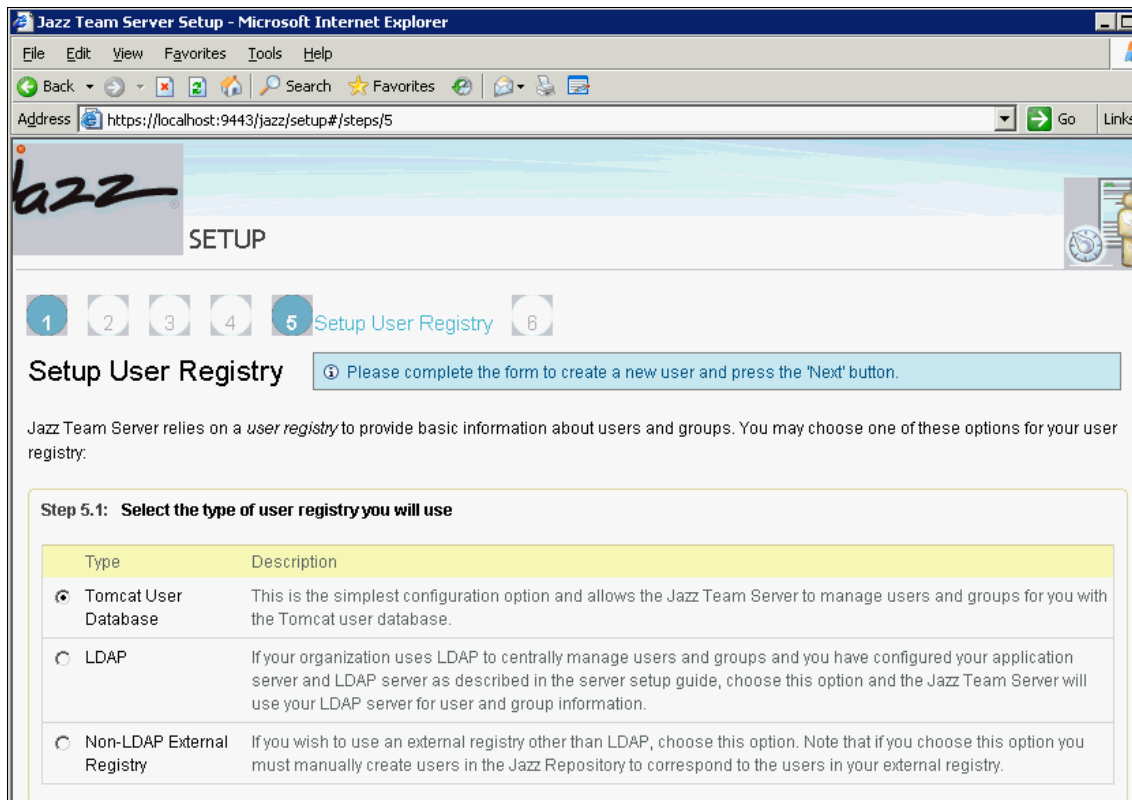


Figure A-50 Setting up the user registry

5. On the Setup User Registry page (Step 2), enter the user ID, Name, Password, and E-mail Address of the user (Figure A-51).

**Step 5.2: Create an administrator account**

You are currently logged in as the default user (ADMIN). Please create an administrator account for yourself using the form below. When you press the "Next" button, the new user account will be created in the Jazz Team Server. Your new account will have the JazzAdmins role, which will allow you to login to the Admin UI.

Property	Value	Description
User ID	jziosi	Your user ID (e.g. 'jsmith')
Name	Lara Ziosi	Your full name (e.g. 'John Smith')
Password	•••••	Enter a password that you can remember but won't be easy for others to guess.
Re-type Password	•••••	Re-type your password to help prevent typos.
E-mail Address	lara.ziosi@nl.ibm.com	Your e-mail address (e.g. 'jsmith@example.com')

Figure A-51 Creating the first user account

6. On the Setup User Registry page (Step 3), you can disable the default ADMIN user that was used to log on to the setup wizard (Figure A-52).
7. On the Setup User Registry page (Step 4), you can see three available Developer licenses. Assign one license to the user that you have created (Figure A-52). Click **Next**.

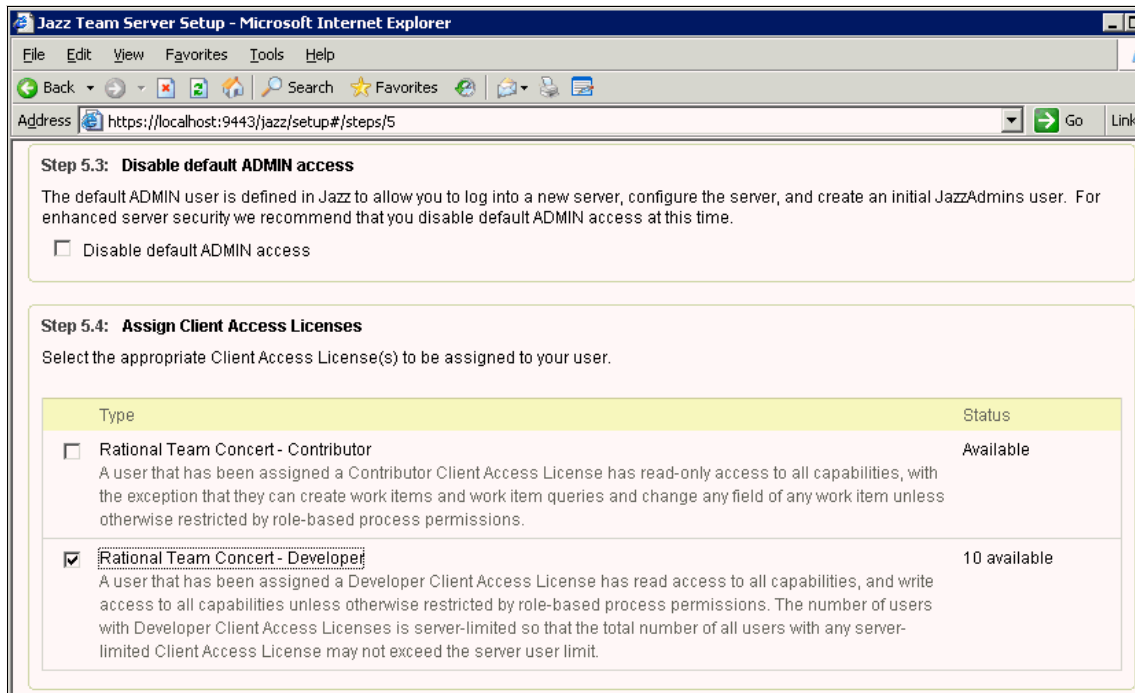


Figure A-52 Assigning developer licenses

You see a summary page from which you can terminate the Setup wizard (Figure A-53).

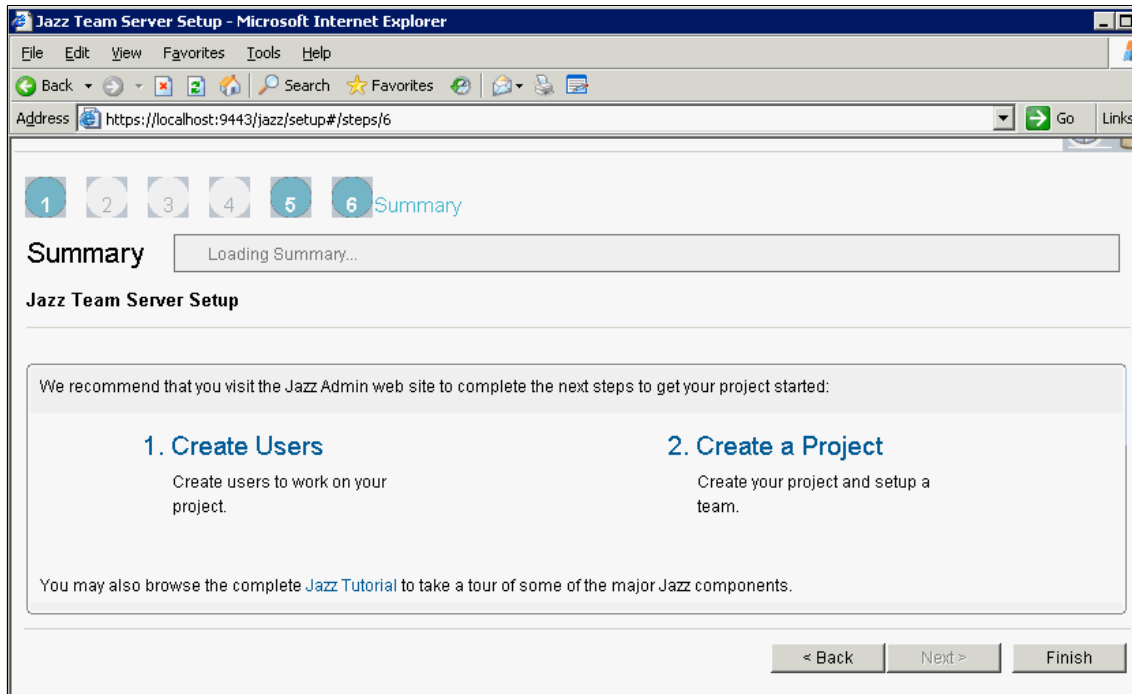


Figure A-53 Summary

8. Verify that the setup is correct by launching the Team Server Admin Web user interface (UI) by opening the following web address:  
`https://localhost:9443/jazz/admin`
9. If you do not want to have passwords in any files on the file system, inspect the contents and then remove the backup property files in the `C:\Program Files\IBM\JazzTeamserver\server\conf\jazz\teamserver-<digits>backup.properties` directory.
10. Connect to the server with the Rational Team Concert client or a web browser.
11. Stop the server from a command window by selecting **Start** → **All Programs** → **Jazz team Server** → **Stop Jazz Team server**.

The server was started as an application. You can make it become a service so that it starts automatically at boot time. See the following Jazz technote (web doc) for instructions:

<http://jazz.net/library/article/72>

## Installing Rational Team Concert Build Engine and Build Toolkit

You can install the Rational Team Concert Build Engine and Build Toolkit from the Launchpad. Follow these steps:

1. Change to the `RAD_SETUP` subdirectory of the directory where you extracted the “disks” for Rational Application Developer for WebSphere Software.
2. Start the Rational Team Concert Launchpad program (Figure A-43 on page 1825) and select the **Rational Team Concert Build System Toolkit** link.
3. This action launches Installation Manager, which is preconfigured for installing the following package, as shown in Figure A-54. Select **Next**.

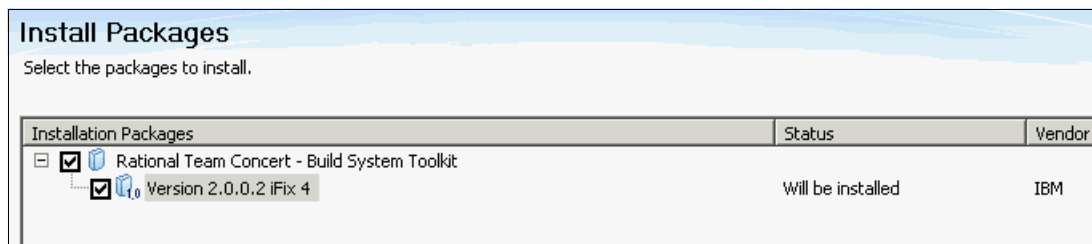


Figure A-54 Installation Package for Rational Team Concert Build System Toolkit

4. On the next page, accept the License Agreement and select **Next**.

5. On the next page, you are prompted to create a new package group (Figure A-55). Accept the defaults and select **Next**.

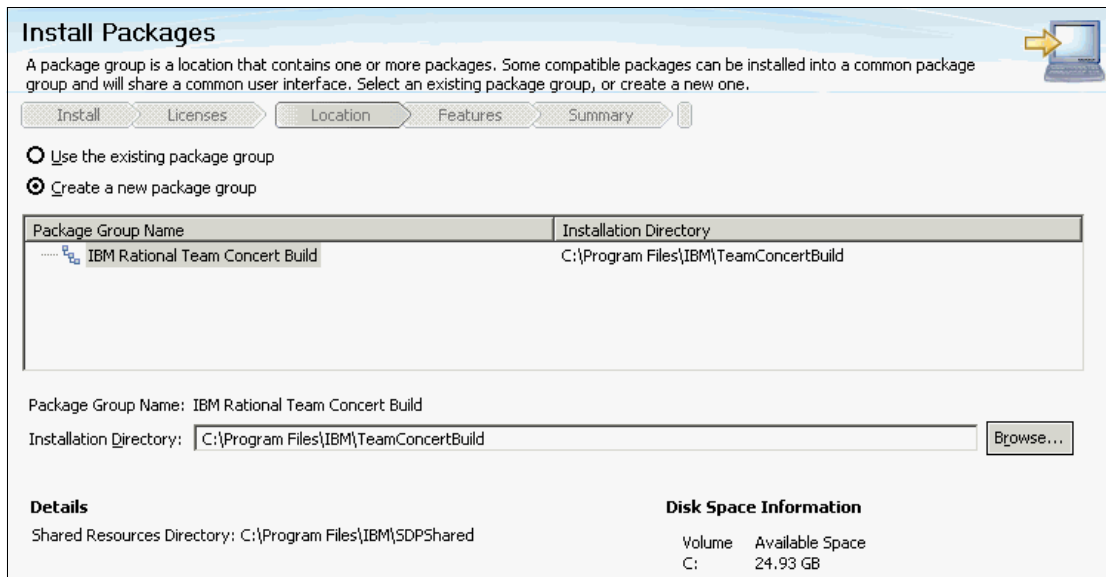


Figure A-55 Creating a new package group

6. On the next page, select the translations and select **Next**.
7. On the next page, select the features (there are no optional features) and select **Next**.
8. On the next page, you see a summary. Select **Install**.
9. After the installation completes, select **Finish**.

## Installing the client and the debug extensions

In addition to the Rational Team Concert server, install the following required components:

- ▶ Rational Team Concert client
- ▶ Debug Extension for Rational Team Concert client
- ▶ Team Debug Service Extension for Rational Team Concert server

### Rational Team Concert client

Rational Team Concert client is not packaged as an optional feature of Rational Application Developer. It must be installed separately. As of the time of writing, you can install it by using the Eclipse P2 installer.

Follow these instructions:

<http://www-01.ibm.com/support/docview.wss?uid=swg21444449>

**Using Installation Manager:** Starting with Rational Team Concert 2.0.0.2 iFix5, you can install the Rational Team Concert client in Rational Application Developer by using IBM Installation Manager, with the implementation of the following Request For Enhancement:

<https://jazz.net/jazz/web/projects/Rational%20Team%20Concert#action=com.ibm.team.workitem.viewWorkItem&id=132988>

## Rational Team Concert Server Debug Extension

The Rational Team Concert Debug Extension is an optional feature of Rational Application Developer that can be installed using IBM Installation Manager. You can modify an existing installation to include this feature, as shown in the following steps:

1. Change to the *RAD\_SETUP* subdirectory of the directory where you extracted the “disks” for Rational Application Developer for WebSphere Software.
2. Launch the Rational Application Developer Launchpad (Figure A-1 on page 1785).
3. Select **IBM Rational debug Extension for IBM Rational Team Concert Server**, which starts IBM Installation Manager and initiates the installation of the package. Select **Next**.
4. On the next page, you see the License agreement. If you accept it, select **I Agree** and **Next**.
5. On the next page, you are prompted to install Rational Team Concert Debug Extension inside the same package group as the Team Concert Server. Accept the default and select **Next** (Figure A-56 on page 1837).



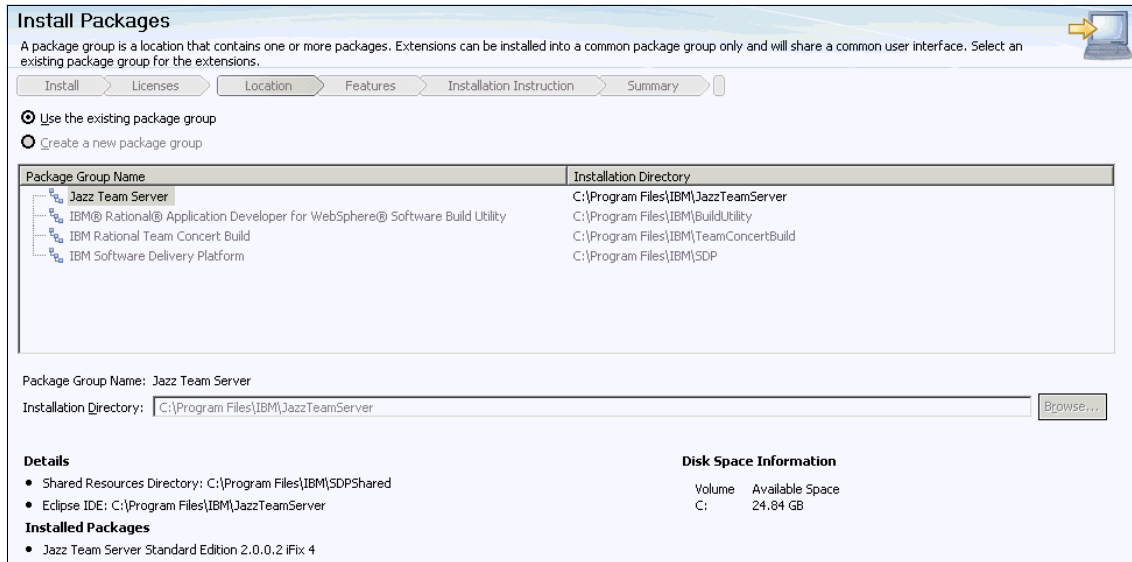


Figure A-56 Installing the Debug Extension in the same package group as the Team Concert Server

- On the next page, accept the defaults for the features. There are no optional features. Select **Next**.
- You see the following installation instruction:

After installing/uninstalling IBM Rational Debug Extension for IBM Rational Team Concert Server, you will need to restart the Team Concert Server for the change to be picked up. If the Team Concert Server is not running using the web server included with the installer, you will need to reset the server through this URL (<https://localhost:9443/jazz/admin?internal#action=com.ibm.team.repository.admin.serverReset>).
- On the Summary Page, select **Install**.

**Important:** In order to use this support, on each client installation, ensure that you select the **Collaborative Debug Extensions for Rational Team Concert Client** feature.

You can verify it by launching IBM Installation Manager, selecting **Modify**, and reviewing the features page of the wizard.

For more information, see the product help under **Developing** → **Debugging Applications** → **Debug Extensions for Rational Team Concert Client** → **Overview**.

For more information, see the information center:

<http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.debug.team.client.ui.doc/topics/cbtovrvw.html>

## Verifying the installation of the server debug extensions

To verify that the installation is correct, connect to the Rational Team Concert server with a web browser and verify that the following service is active (Figure A-57):

`com.ibm.debug.team.common.service.ITeamDebugService`

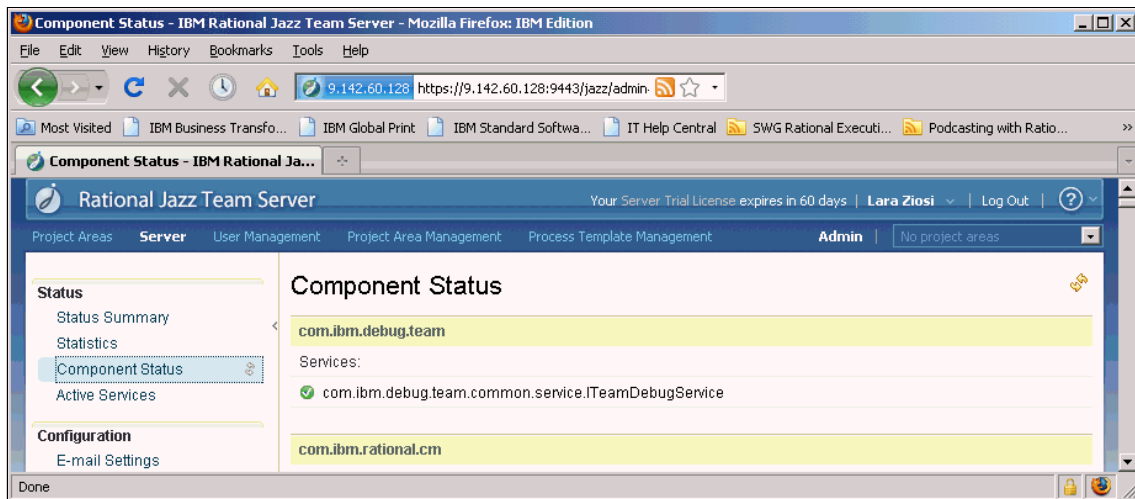


Figure A-57 Verifying that the debug extension is active on the Rational Team Concert server

## Rational Team Concert Server Code Coverage extension

The Rational Team Concert Server Code Coverage extension is an optional component that can be installed on the Rational Team Concert Server from the Rational Application Developer Launchpad. Complete these steps:

1. Change to the `RAD_SETUP` subdirectory of the directory where you extracted the “disks” for Rational Application Developer for WebSphere Software.
2. Launch the Rational Application Developer Launchpad (Figure A-1 on page 1785).
3. Select **IBM Rational Application Developer Code Coverage tools for IBM Rational Team Concert Builds**, which starts IBM Installation Manager and initiates the installation of the package. Select **Next** (Figure A-58 on page 1839).

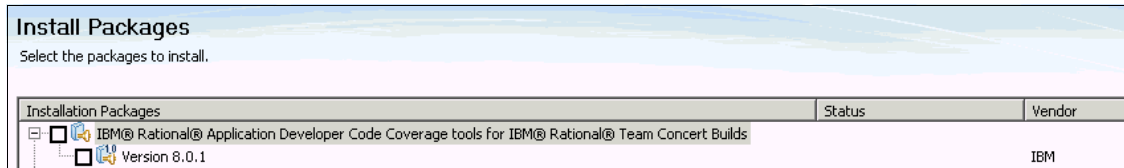


Figure A-58 Installing Code Coverage Tools for Team Concert builds

4.  On the next page, you see the License agreement. If you accept it, select **I Agree** and **Next**.
5. On the next page, you are prompted to select the existing package group that contains the IBM Rational Team Concert Build System Toolkit. Accept **Use the existing package group** (default) and select **Next** (Figure A-59).

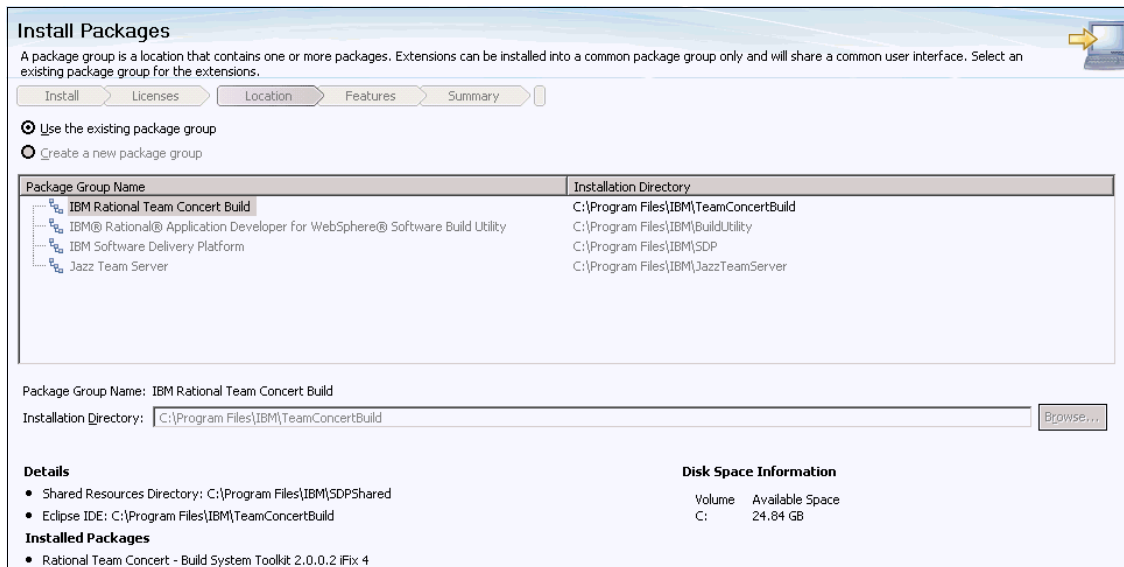


Figure A-59 Use the existing package group for Code Coverage tools

6. Accept all defaults until you reach the end of the wizard, select **Install**, and then click **Finish**.

To configure the Code Coverage Extension, see this website:

[http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=com.ibm.rad.install.doc/topics/t\\_cc\\_rtc\\_ext\\_config.html](http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=com.ibm.rad.install.doc/topics/t_cc_rtc_ext_config.html)

# Installing Rational Application Developer Build Utility

You can use the *build utility* to execute builds on a build server without having Rational Application Developer, WebSphere Application Server, or WebSphere Portal Server installed. The build utility does not contain any user interface code. By using the build utility, you can create Apache ANT scripts for running the available Ant tasks in headless mode.

In this section, we show you how to install only the core files and the WebSphere Application Server and WebSphere Portal Server stub files on Microsoft Windows. These files are required for executing Java Platform, Enterprise Edition (Java EE)-specific ANT tasks, such as *ejbDeploy*, targeting one of these servers.

For a complete description of the other platforms, see the product help under **Installing and Upgrading** → **Installing Supporting Software** → **Installing Rational Application Developer for WebSphere Software build utility**, which is also available at this website:

[http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/com.ibm.rad.install.doc/topics/t\\_install\\_build\\_utility\\_8.html](http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/com.ibm.rad.install.doc/topics/t_install_build_utility_8.html)

You can start the installation of the build utility from the corresponding link on **1 launchpad.exe**, or in IBM Installation Manager, you can configure the repository that is located in `BuildUtility\disk1\diskTag.inf`. Perform these steps:

1. On the Install Packages page, select **IBM Rational Application Developer for WebSphere Software Build Utility**, as shown in Figure A-60. Typically, you install the build utility on a computer that does not have Rational Application Developer installed, although there is no reason to avoid the coexistence of both packages on the same computer, if desired). Select **Next**.

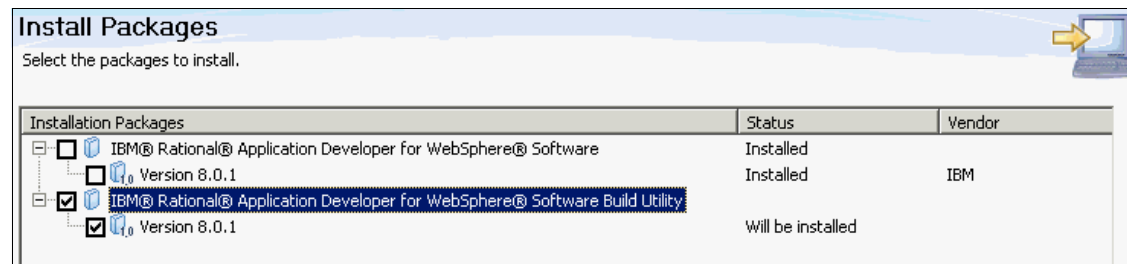


Figure A-60 Install Packages window

2. On the next page, review the License Agreement. If you agree, select **I Accept** and **Next**.

3. On the next page, select the Package Group Name. Enter the package group Installation Directory, which must be new, such as C:\Program Files\IBM\BuildUtility (Figure A-61). Select **Next**.

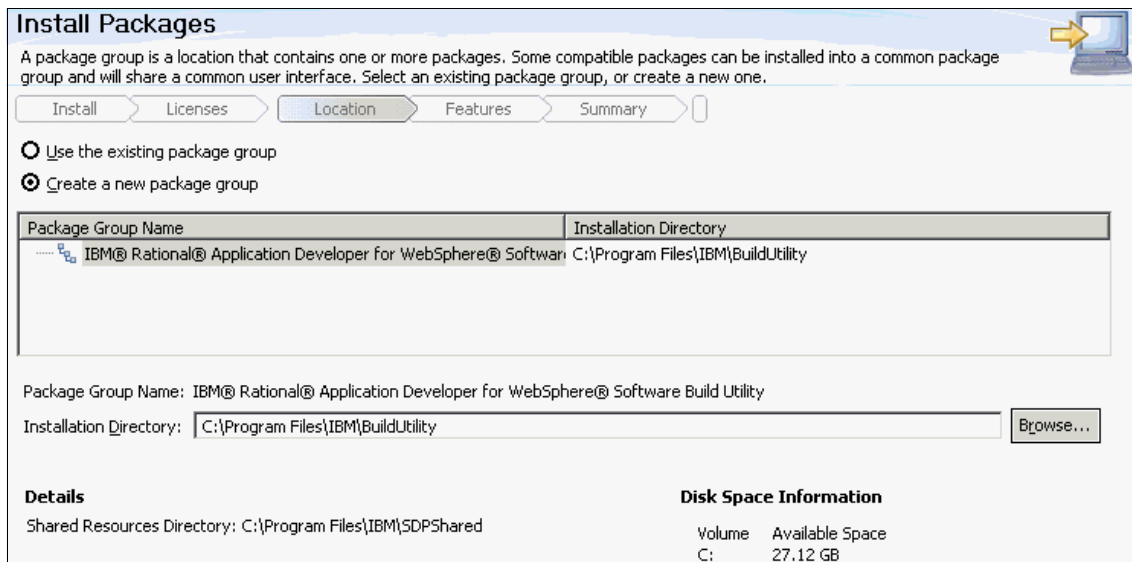


Figure A-61 Selecting a new package group installation directory for installing the build utility

4. On the next page, accept all of the defaults because we do not extend an Existing Eclipse. Select **Next**.
5. On the next page, select the translations to install. Select **Next**.
6. On the next page, select the desired WebSphere Application Server and WebSphere Portal Server stubs to install (Figure A-62). Select **Next**.

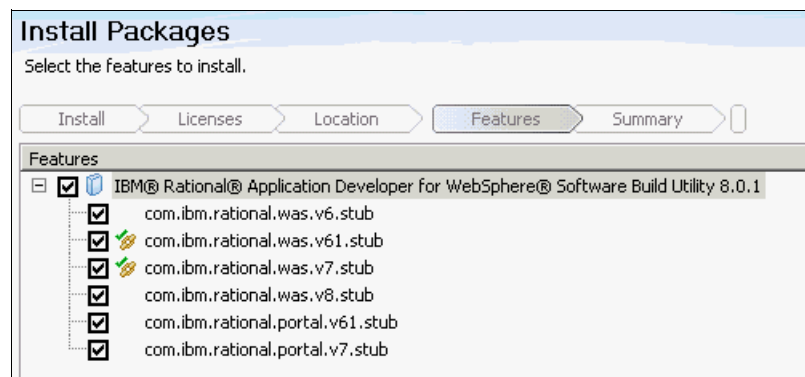


Figure A-62 Selecting features to install

7. On the summary page, review your selections and select **Install**.
8. On the last page, select **Finish**.

## Installing IBM Rational ClearCase

Rational Application Developer integrates with Rational ClearCase V7.1.1 and V7.1.2.

Installing Rational ClearCase requires planning. See the following information in the ClearCase Information Center:

- ▶ [https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m0/index.jsp?topic=/com.ibm.rational.clearcase.help.ic.doc/helpindex\\_clearcase.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m0/index.jsp?topic=/com.ibm.rational.clearcase.help.ic.doc/helpindex_clearcase.htm)
- ▶ [https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.help.ic.doc/helpindex\\_clearcase.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.help.ic.doc/helpindex_clearcase.htm)

In the following sections, we describe the minimal number of required steps to run the example scenarios that are covered in this book. We ran these scenarios on a single machine that acted as both server and client. In this example, we installed Rational ClearCase V7.1.2 on Microsoft Windows.

For the download location, refer to this website:

<http://www-01.ibm.com/support/docview.wss?rs=984&uid=swg24028116>

The package name of Rational ClearCase V7.1.2 Windows is CZJG5ML.

Before installing, review the following planning information:

[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc\\_ms\\_install.doc/topics/c\\_inst\\_planning\\_c.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc_ms_install.doc/topics/c_inst_planning_c.htm)

The following list is an overview of the installation steps:

1. Extract CZJG5ML.zip to a temporary directory.
2. If you have not installed IBM Installation Manager yet, perform these steps:
  - a. Inside the folder called disk1, execute **launchpad.exe**.
  - b. Select **Install IBM Rational ClearCase v7.1.2**, which installs IBM Installation Manager 1.4.1 and preconfigures the installation of Rational ClearCase and the installation of the license key administrator.
  - c. Figure A-63 on page 1843 shows the features that we chose.

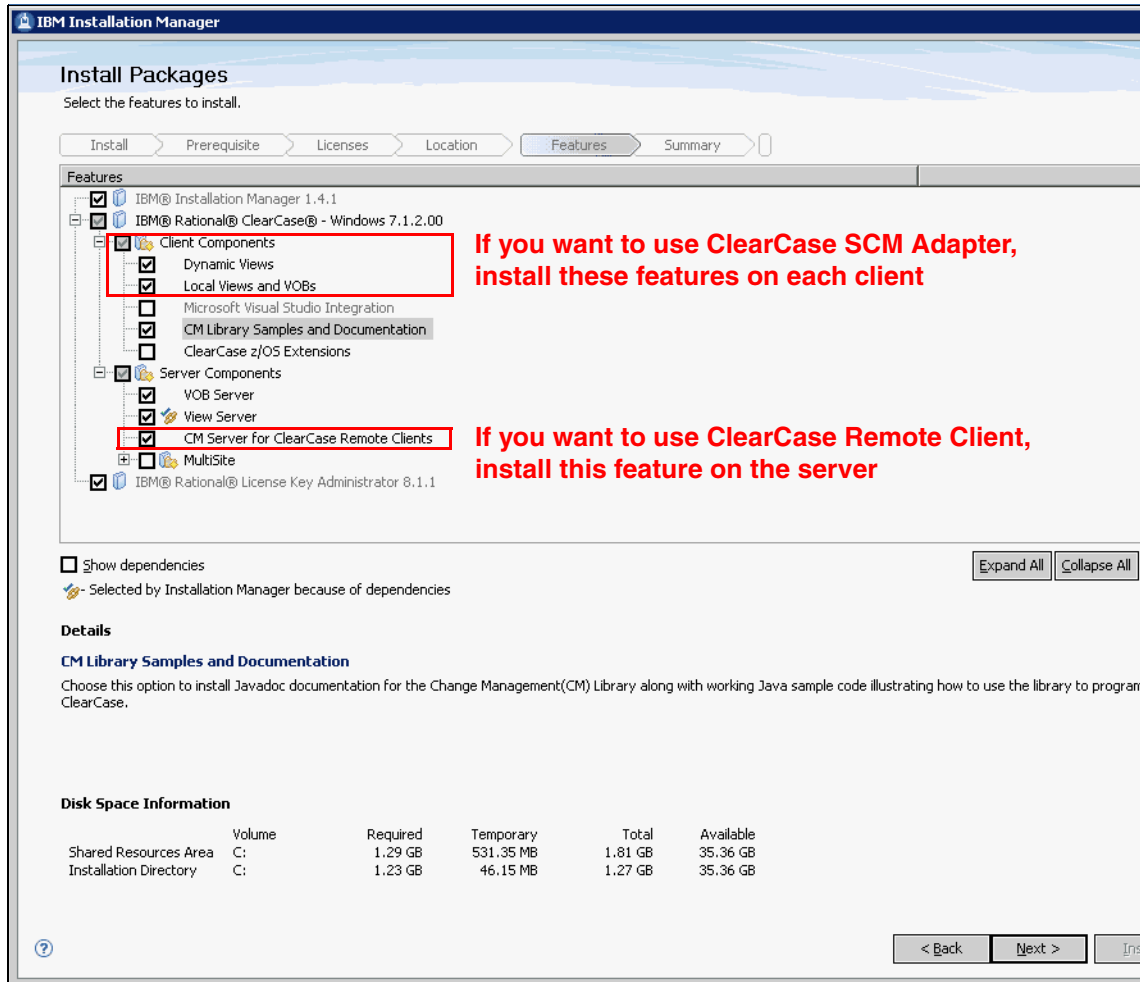


Figure A-63 Choosing features during ClearCase installation

- d. For more information about this scenario, see this website:
  - [https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/topic/com.ibm.rational.clearcase.cc\\_ms\\_install.doc/topics/t\\_start\\_install\\_launchpad.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/topic/com.ibm.rational.clearcase.cc_ms_install.doc/topics/t_start_install_launchpad.htm)
3. If you have installed IBM Installation Manager 1.4.1 already, perform these steps:
  - a. Launch IBM Installation Manager.
  - b. Select **File** → **Preferences** → **Repositories**.
  - c. Configure a new Repository by browsing to the `disk1/diskTag.inf` file.

- d. Select **OK**.
- e. In the major Installation Manager page, select **Install**.
- f. For more information about this scenario, see this website:

[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc\\_ms\\_install.doc/topics/t\\_start\\_install\\_launchpad.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc_ms_install.doc/topics/t_start_install_launchpad.htm)

**Important:** If you plan to use the ClearCase SCM Adapter, you must install the ClearCase Client Components (Figure A-63 on page 1843) on each client machine.

If you plan to use the ClearCase Remote Client, you must install the CM Server for ClearCase Remote Clients (Figure A-63 on page 1843) on a server machine.

4. During the installation, the wizard asks for various parameters, for which you can refer to the ClearCase documentation. The one parameter that you must set, especially for working with Rational Application Developer, is related to MVFS case sensitivity. You must select the **Case Preserving** option, as shown in Figure A-64.

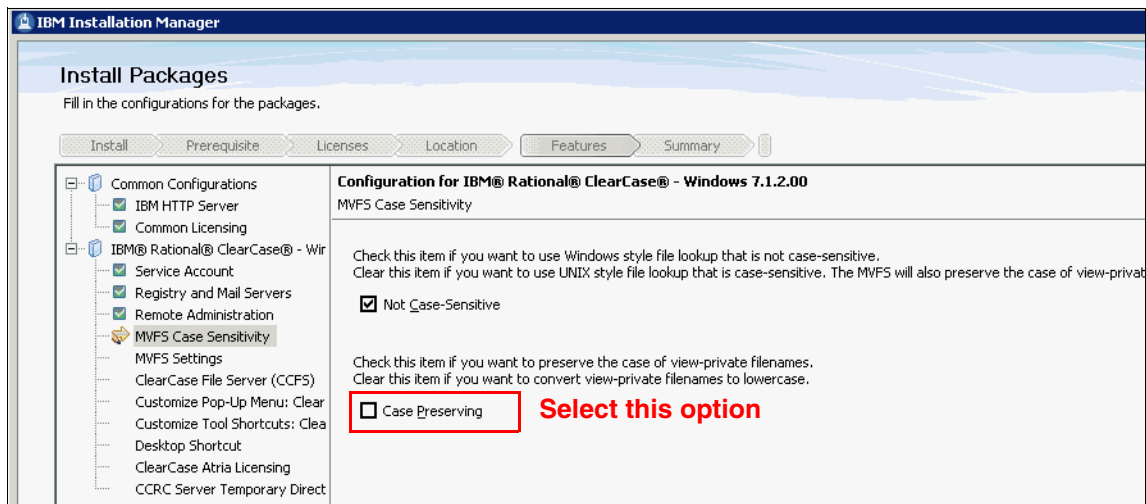


Figure A-64 Selecting the Case Preserving option in the MVFS Case Sensitivity window

After you complete the installation wizard, review the following information:

[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc\\_ms\\_install.doc/topics/c\\_post\\_inst\\_config.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc_ms_install.doc/topics/c_post_inst_config.htm)



## Creating a Storage Location

Before you can create a versioned object base (VOB) and a view, you must create a minimal storage location. For more information about storage locations, see this website:

[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc\\_admin.doc/topics/c\\_rgy\\_data\\_other\\_stgloc.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc_admin.doc/topics/c_rgy_data_other_stgloc.htm)

1. Create a folder, such as C:\Views, and share it. For required permissions, see this website:

[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/topic/com.ibm.rational.clearcase.cc\\_admin.doc/topics/c\\_access\\_fs\\_shares.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/topic/com.ibm.rational.clearcase.cc_admin.doc/topics/c_access_fs_shares.htm)

2. Select **Start** → **All programs** → **IBM Rational ClearCase** → **Administration** → **Server Storage Wizard**.
3. In the ClearCase Server Storage Configuration Wizard, select **Yes, start storage configuration now**.
4. Select **Let me select local drives and let the Storage Configuration Wizard configure the locations for VOB and view storage automatically**, as shown in Figure A-65.

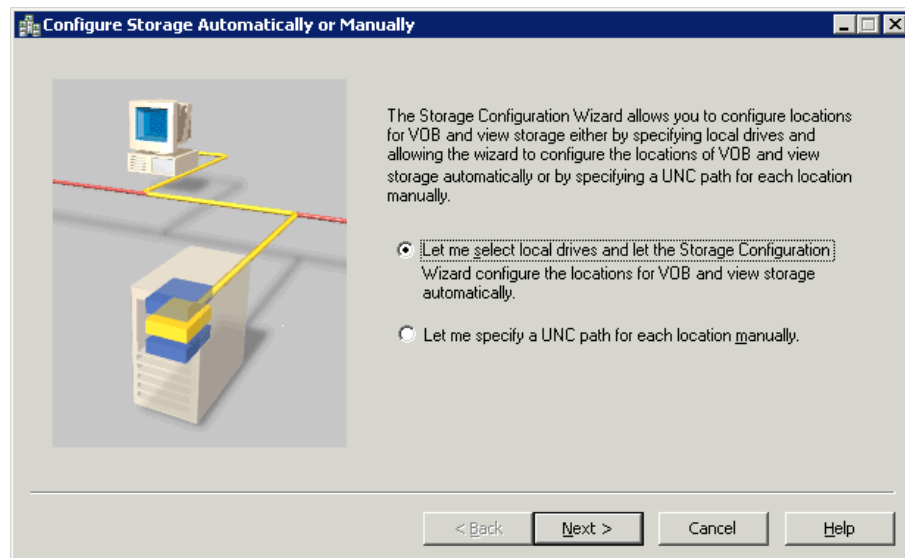


Figure A-65 Select to configure storage automatically in the Storage Location Wizard

5. Select **Next**.

6. Accept all of the defaults, as shown in Figure A-66.
7. Select **Next**.

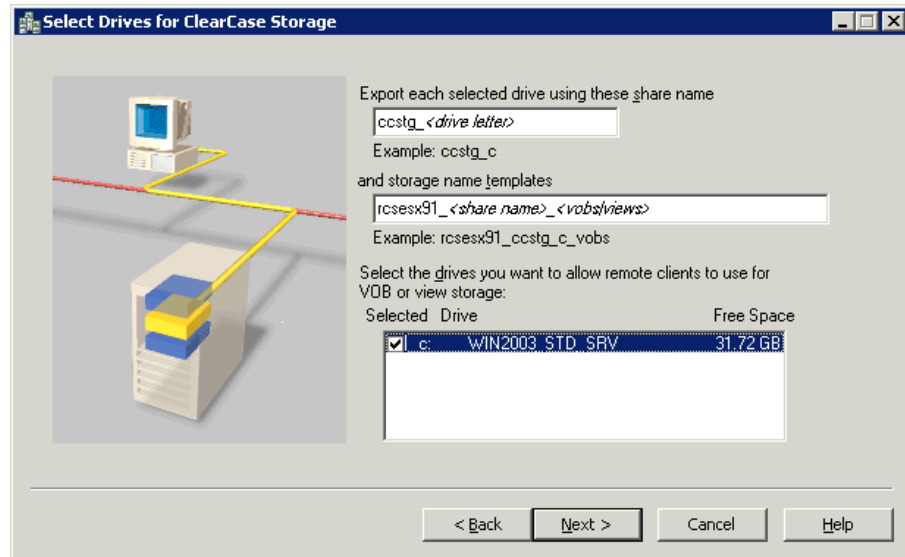


Figure A-66 Select Drives for ClearCase Storage page

8. Review the results in the summary page, as shown in Figure A-67.

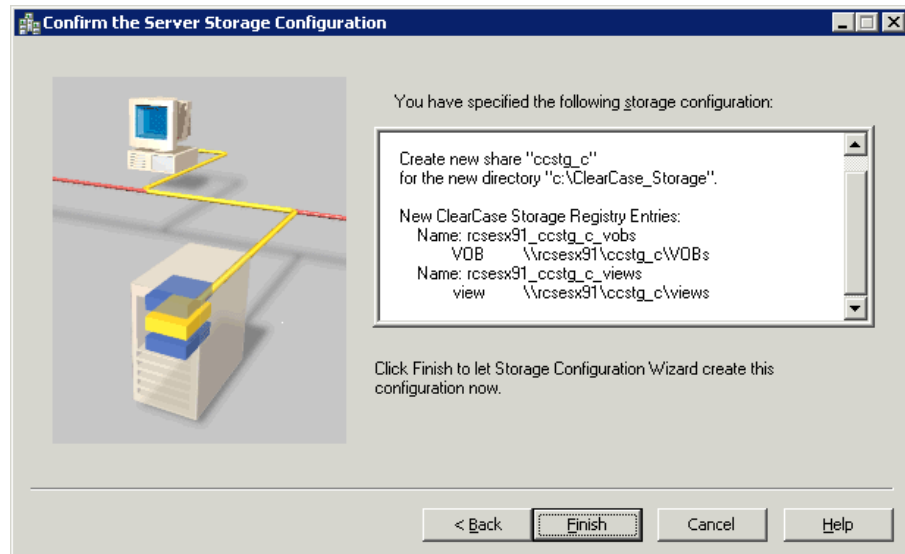


Figure A-67 Server Storage Configuration summary page

For guidelines for locating dynamic view storage directories, see this website:  
[https://publib.boulder.ibm.com/infocenter/cche1p/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.dev.doc/topics/cc\\_mvfs/glines\\_locate\\_storage\\_dirs.htm](https://publib.boulder.ibm.com/infocenter/cche1p/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.dev.doc/topics/cc_mvfs/glines_locate_storage_dirs.htm)

## Creating a VOB for use in Base ClearCase

To create a VOB for use in Base ClearCase, perform these steps:

1. Select **Start** → **All programs** → **IBM Rational ClearCase** → **Administration** → **Create VOB**.
2. On the VOB Creation Wizard - Name and Major Parameters page (See Figure A-68), perform these steps:
  - a. For the new VOB name, enter RAD8RedbookBase.
  - b. Clear **This VOB will contain UCM Components**.
  - c. Clear **Create as a UCM project VOB** and select **Next**.

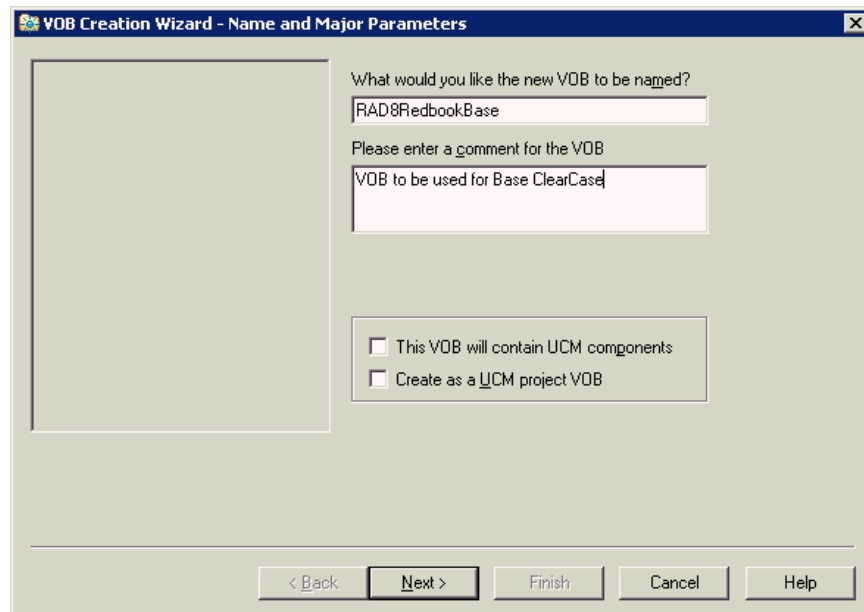


Figure A-68 Vob Creation Wizard - Name and Major Parameters page

3. On the Vob Creation Wizard - Storage page, select **Use Explicit Path**, as shown in Figure A-69 on page 1848.

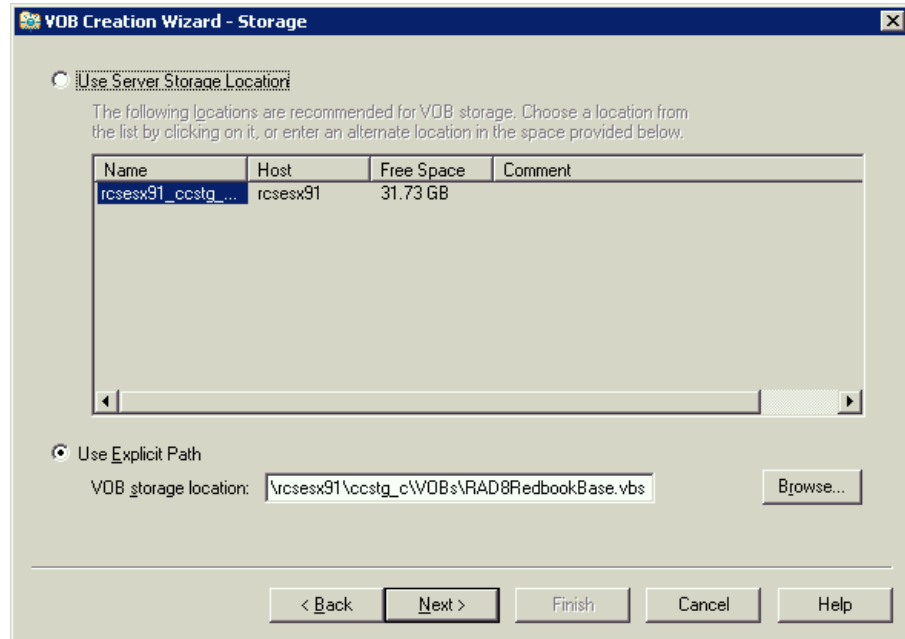


Figure A-69 VOB Creation Wizard - Storage page

4. On the Vob Creation Wizard - Options page, you can select **Make this a public VOB**, provided that you have set a password for the ClearCase registry host. See Figure A-70 on page 1849.

For more information, see these websites:

- For more information about setting the ClearCase registry password, see this website:

[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc\\_admin.doc/topics/t\\_rgy\\_password.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc_admin.doc/topics/t_rgy_password.htm)

- For more information about public and private VOBs, see this website:

[https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc\\_admin.doc/topics/t\\_rgy\\_password.htm](https://publib.boulder.ibm.com/infocenter/cchelp/v7r1m2/index.jsp?topic=/com.ibm.rational.clearcase.cc_admin.doc/topics/t_rgy_password.htm)

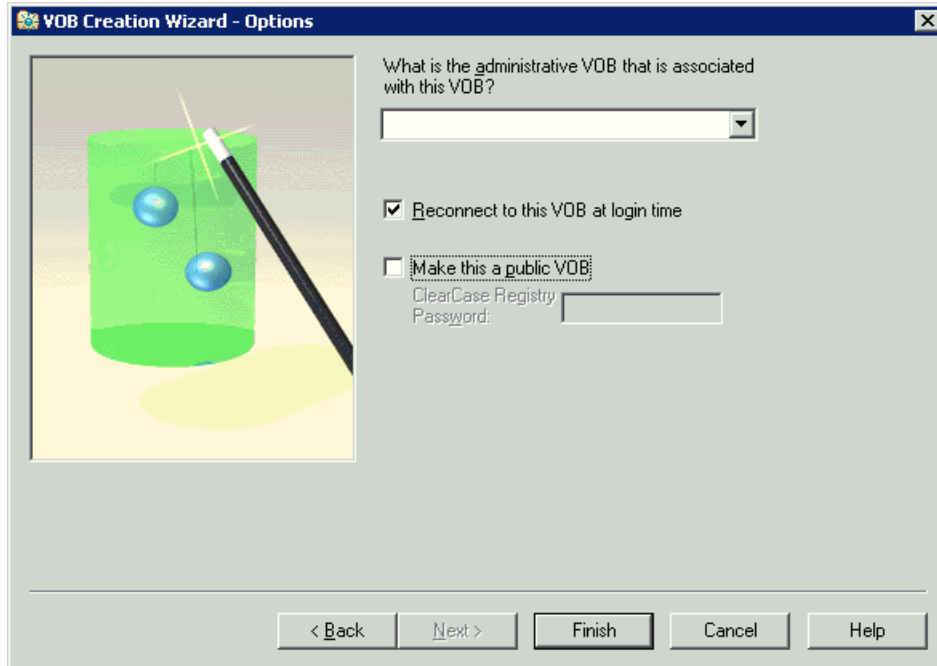


Figure A-70 VOB Creation Wizard - Options page

## Creating a dynamic view

To create a dynamic view for working in Base ClearCase, follow these steps:

1. Launch Rational ClearCase Explorer, as shown in Figure A-71 on page 1850.

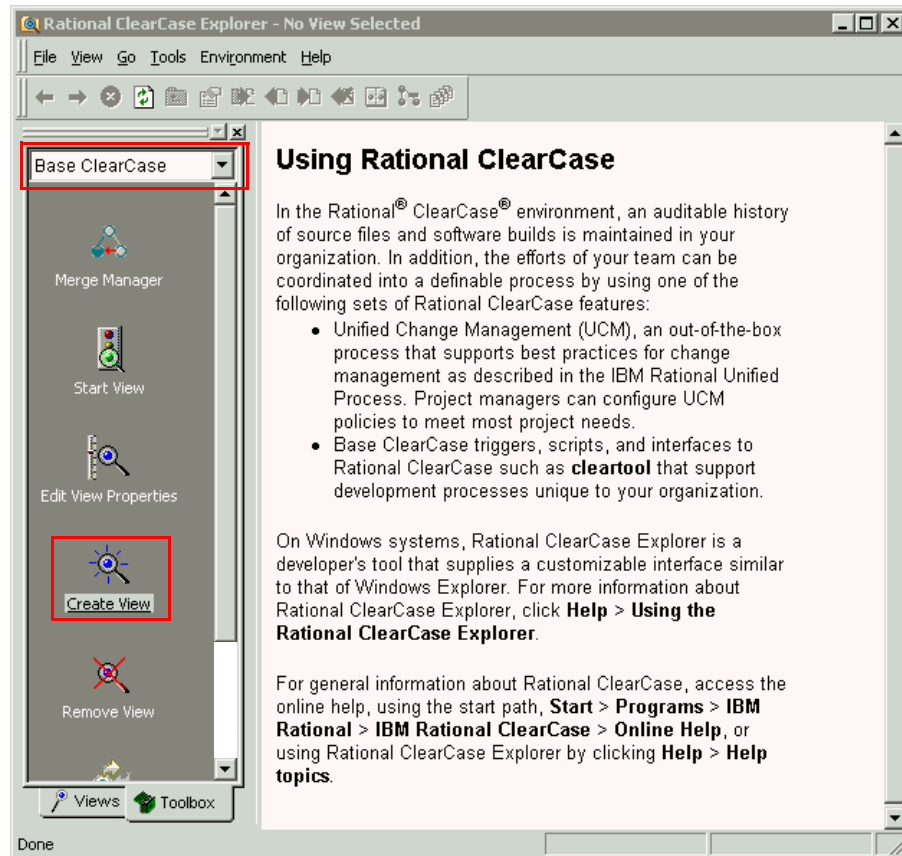


Figure A-71 Creating a view from the Rational ClearCase Explorer

2. From the drop-down list, select **Base ClearCase**.
3. Select **Create View**.
4. On the View Creation Wizard-Choose a Project page, select **No** because this view is a view for working in Base ClearCase. Projects are only used in Unified Change Management (UCM).
5. Select **Next**.
6. On the View Creation Wizard - Choose Snapshot View or Dynamic View page, select **Dynamic View**, as shown in Figure A-72 on page 1851.

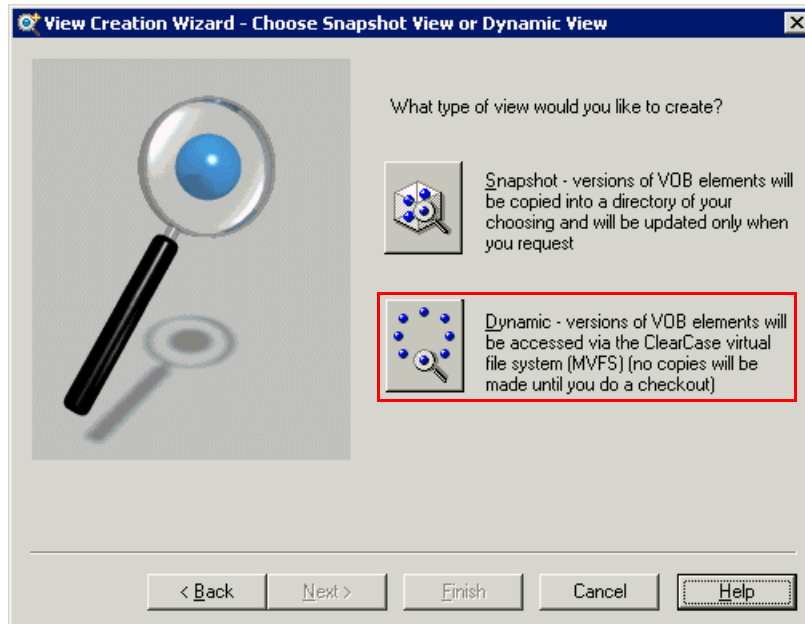


Figure A-72 View Creation Wizard - Choose Snapshot View or Dynamic View

7. On the View Creation Wizard - Choose Name and Drive for a Dynamic View page, complete these steps:
  - a. For the new view name, enter `lziosi_base_view`, as shown in Figure A-73 on page 1852.
  - b. Select the **Y:** drive to use to access this view. Dynamic views are accessible collectively from the M: drive, also.
  - c. Select **Advanced Options**.

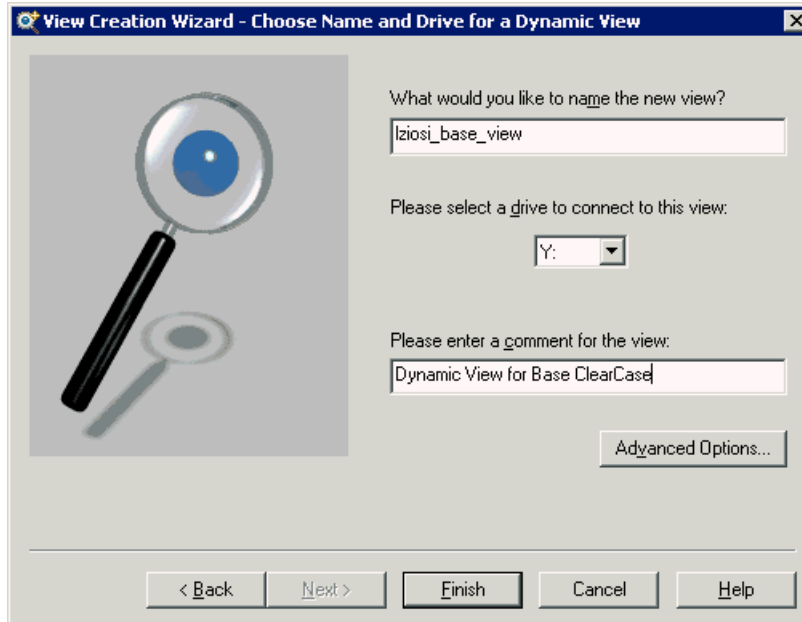


Figure A-73 View Creation Wizard - Choose Name and Drive for a Dynamic View

8. On the Advanced View Options page, select **Use Explicit Path**. Browse to the network share storage location that was created in step 7.a, as shown in Figure A-74 on page 1853.



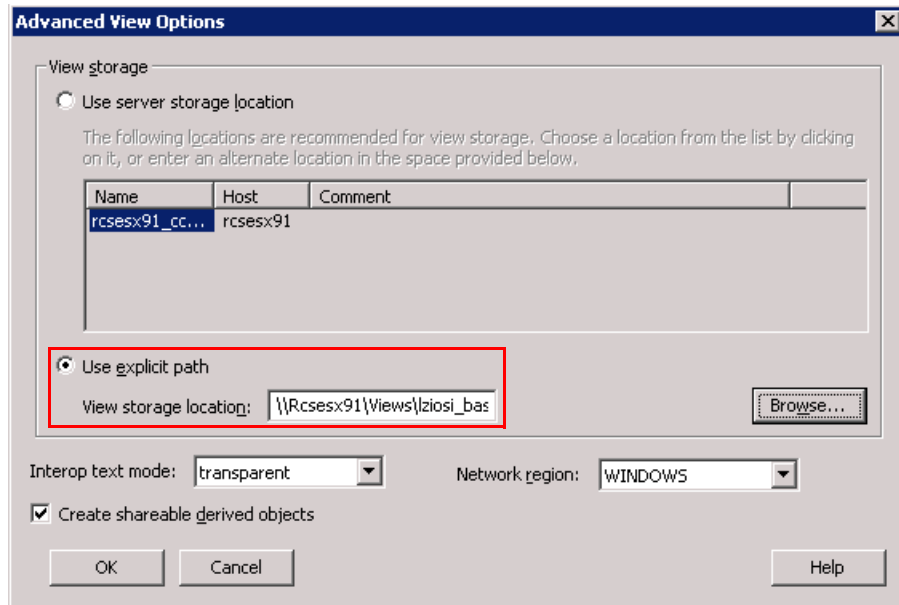


Figure A-74 Advanced View Options

9. You see a Confirm window, as shown in Figure A-75 on page 1854. Complete these tasks:
  - a. Select **Inspect Config Spec** to show the default Config Spec, which contains the rules:
    - element \* CHECKOUT
    - element \* /main/LATEST

By default, this view shows all of the versions that are checked out, and for those versions that are checked in, this view shows the latest version from the main branch. See Figure A-76 on page 1854.

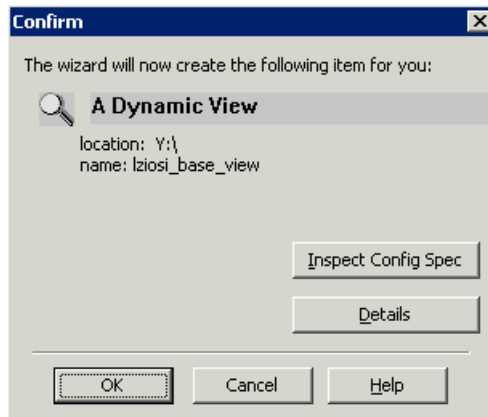


Figure A-75 Selecting *Inspect Config Spec* in the view creation *Confirm* window

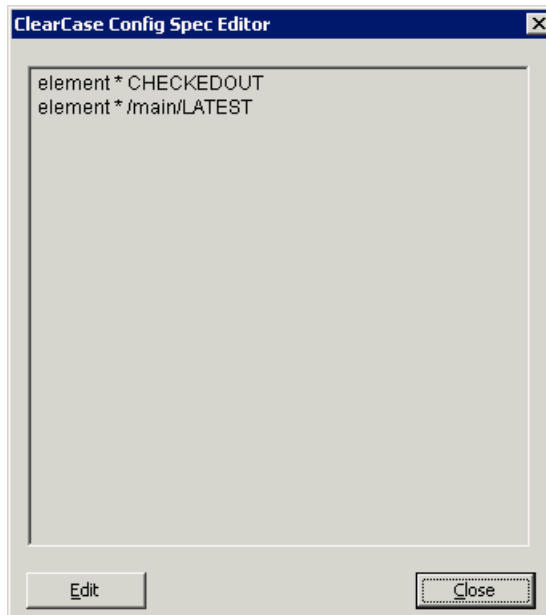


Figure A-76 *ClearCase Config Spec Editor* rules

10. You now see a confirmation dialog window indicating that the view was created successfully.

# Installing IBM Rational ClearCase Remote Client Extension

There are two builds of Rational ClearCase Remote Client:

- ▶ You can download the build for stand-alone use from Passport Advantage, but that build is unsuitable for shell-sharing with Rational Application Developer.
- ▶ Rational ClearCase Remote Client Extension is a plug-in installable from IBM Installation Manager. You can install it in the same package group as Rational Application developer, which is called *shell-sharing*.

You can download Rational ClearCase Remote Client Extension 7.1.1 or 7.1.2 from this website:

<http://www-01.ibm.com/support/docview.wss?uid=swg21431506>

The following steps illustrate how to install Rational ClearCase Remote Extension 7.1.2 in the same package group as Rational Application Developer:

1. Download the 7.1.2.0-RATL-RCCRC-EXT-FP00.zip file.
2. Extract the file to a temporary location, such as  
C:\download\7.1.2.0-RATL-RCCRC-EXT-FP00.
3. Launch IBM Installation Manager.
4. Select **File** → **Preferences** → **Repositories**.
5. Enter a new repository by browsing to the repository.config file inside the extracted package, for example:  
C:\download\7.1.2.0-RATL-RCCRC-EXT-FP00\repository.config
6. Select **OK**.
7. On the main Installation Manager page, select **Install**.
8. Select **IBM Rational ClearCase Remote Client Extension**, as shown in Figure A-77 on page 1856.

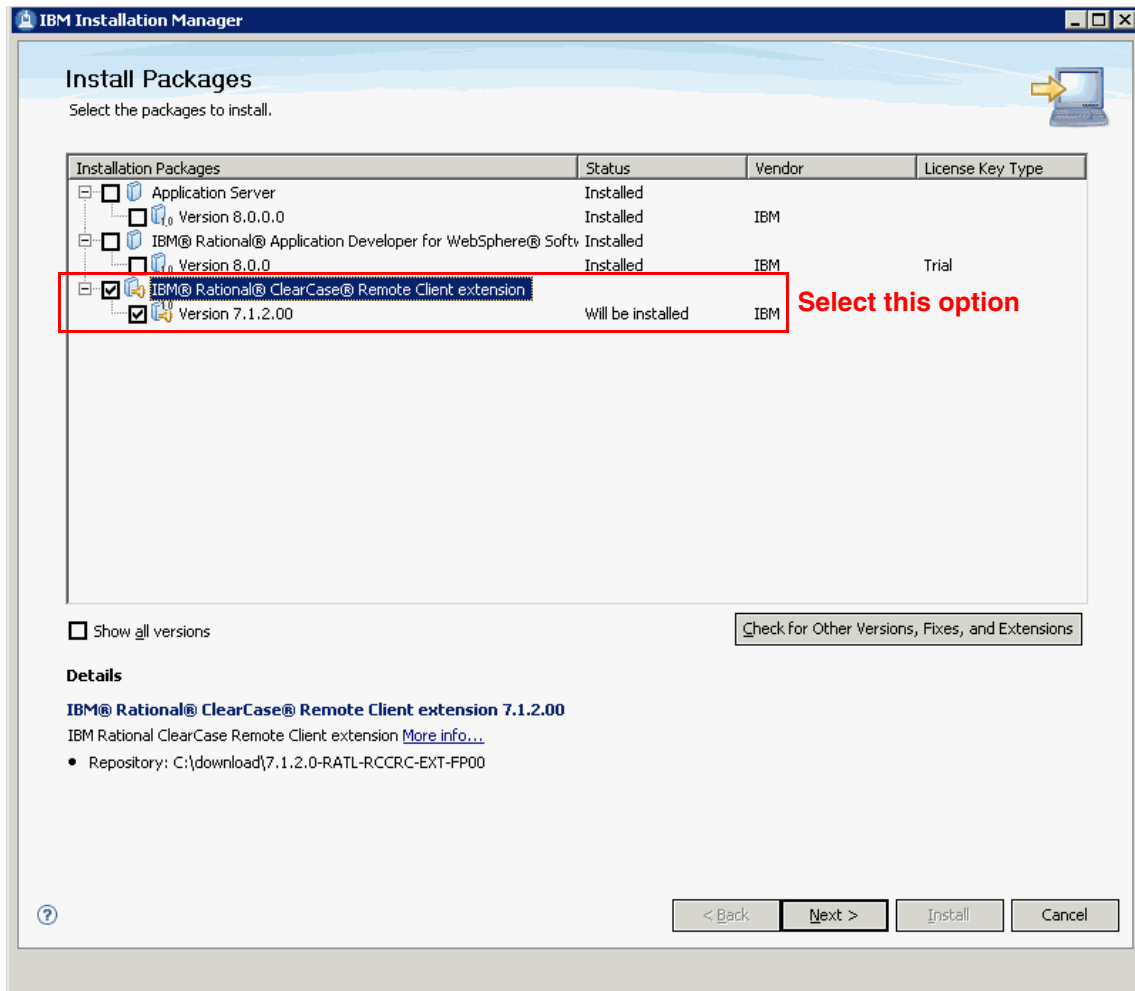


Figure A-77 Installing Rational ClearCase Remote Client Extension

9. If you see a prompt to stop the VOB server and view server processes, accept the option to stop these processes (Figure A-78 on page 1857).

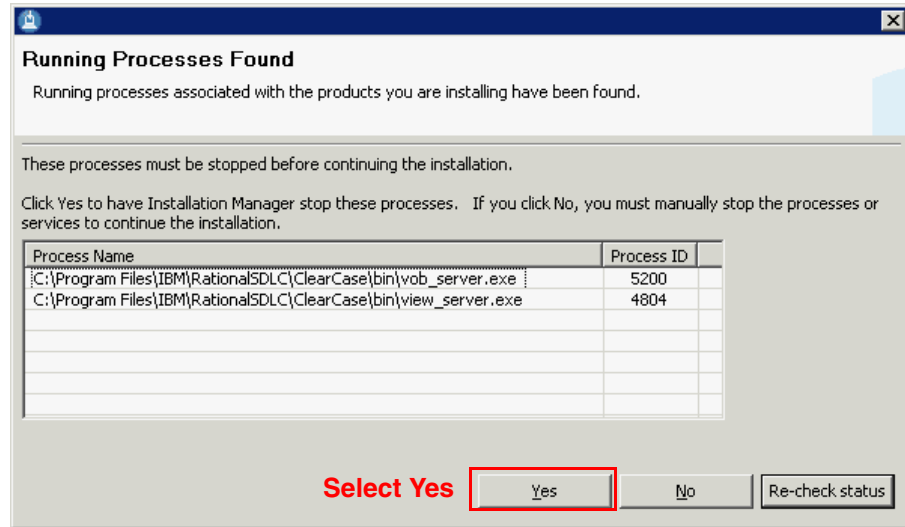


Figure A-78 Prompt to stop the VOB server and view server processes

10. Select **Next**.
11. On the License tab, you see a page instructing you to stop antivirus software. Complete these steps:
  - a. Review the information that is provided in the supplied link.
  - b. Take appropriate action as required and select **Next**.
12. Read and accept the license agreement. Select **Next**.
13. On the Location tab, you see a page with a list of available package groups. This list varies, depending on the products that you have installed. Select the package group that contains Rational Application Developer:
  - a. Select **Use existing Package Group**.
  - b. Installation Manager selects a compatible package group, typically, **IBM Software Delivery Platform**, as shown in Figure A-79 on page 1858.

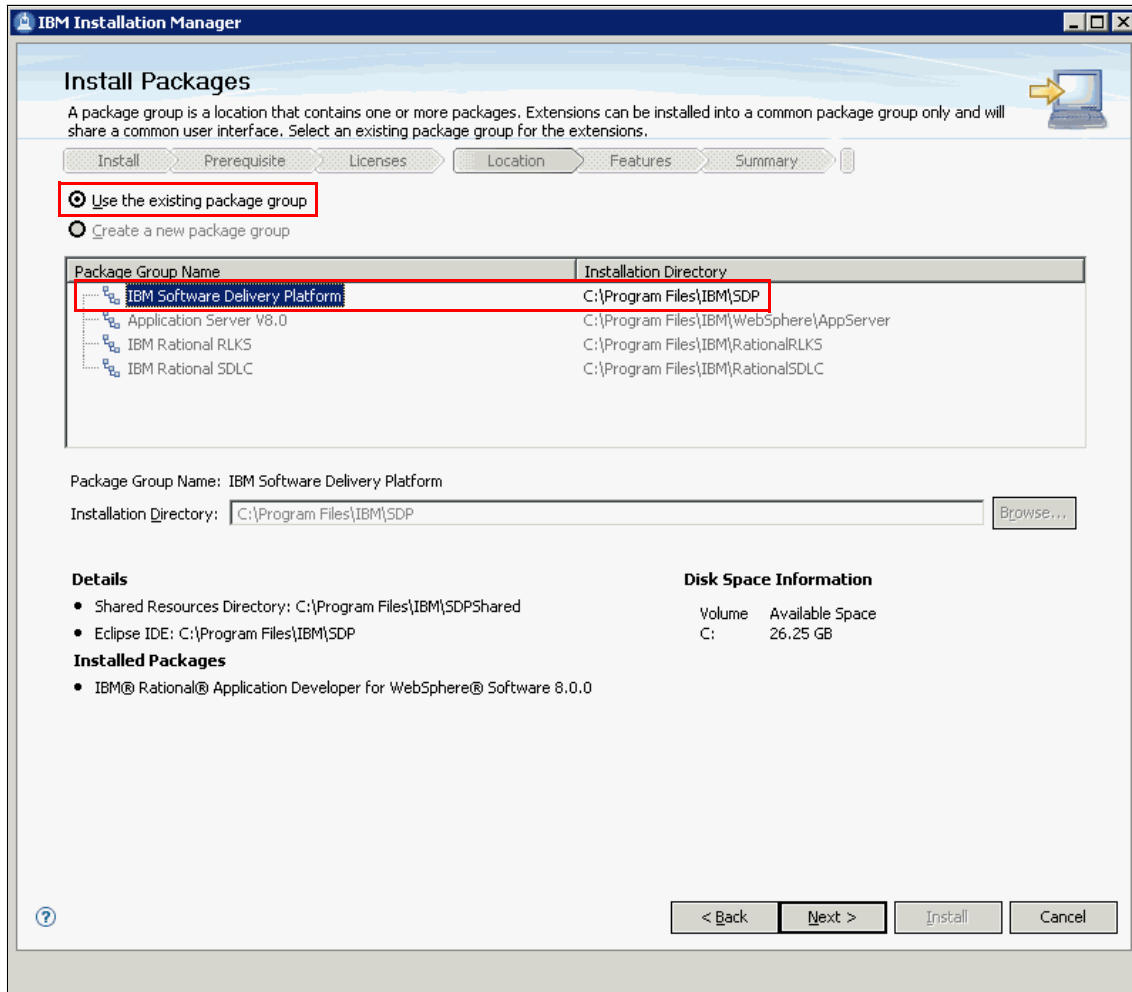


Figure A-79 Selecting to install the extension in the package group with Rational Application Developer

14. On the Features tab, accept the default, because there are no optional features. Select **Next**.
15. On the Summary tab, select **Install**.

## Verifying the installation

After you have installed Rational ClearCase Remote Client Extension, launch Rational Application Developer on a new workspace to verify the installation.

You see a Welcome page that contains links to the ClearCase Remote Client documentation. You also see, in the Java EE perspective, a ClearCase menu with contents provided by the Rational ClearCase SCM Adapter, as shown in Figure A-80.

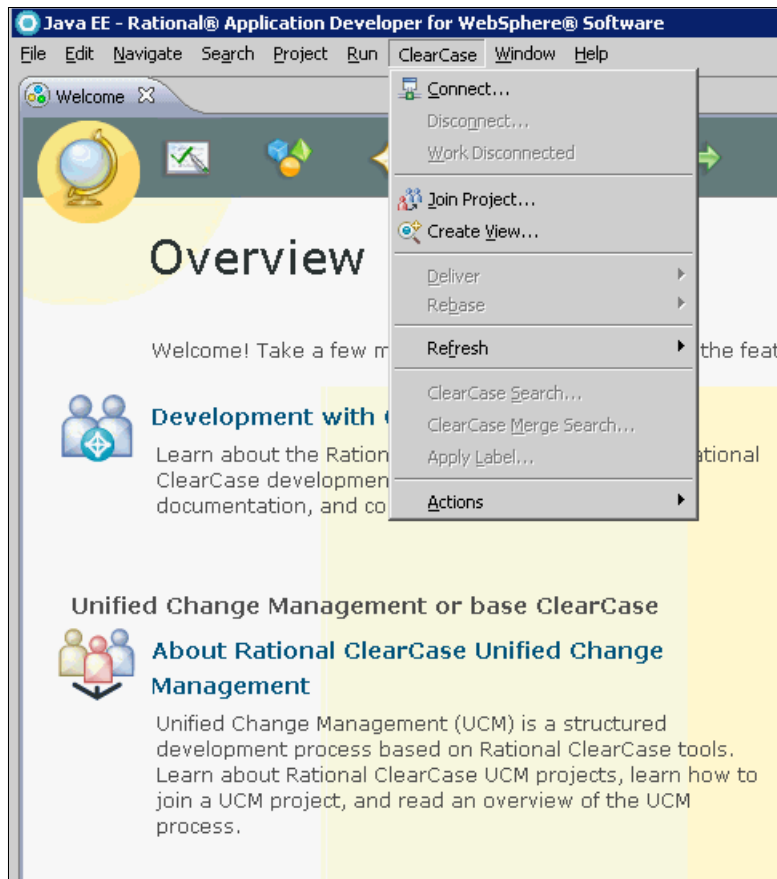


Figure A-80 ClearCase Remote Client main menu and updated Welcome page

You can also verify that the enabled capability corresponds to ClearCase Remote Client by accessing the following menu:

1. Click **Window** → **Preferences**.
2. On the Preferences page, navigate to **General** → **Capabilities** → **Team**.
3. Select **Advanced**.
4. The following options are preselected, as shown in Figure A-31 on page 1428:
  - **Core Team Support**

– **ClearCase Remote Client**

5. Clear all other options (**CVS Support** and **ClearCase SCM Adapter**), because you typically work with one team provider at a time.

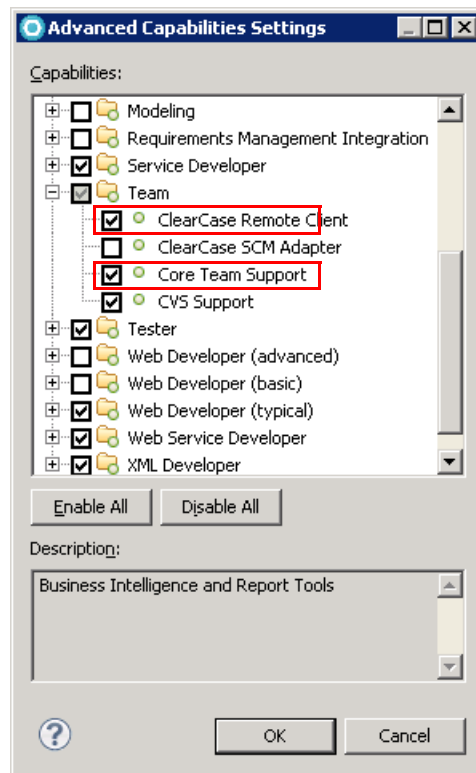


Figure A-81 Required Capabilities: Core Team Support and ClearCase Remote Client

## Configuring ClearCase for UCM development

The following list shows the minimum requirements to set up a Unified Change Management (UCM) configuration:

- ▶ Create a Project VOB (PVOB)
- ▶ Create a VOB, which is associated with the PVOB, to hold UCM components
- ▶ Create a UCM project
- ▶ As a developer, join the project (possibly creating a development stream and integration and development views)



In the following paragraphs, we show an example of performing the previous tasks by using the Project Explorer, the ClearCase Remote Client, and web views.

Follow these steps to create a PVOB:

1. Select **Start** → **All programs** → **IBM Rational ClearCase** → **Administration** → **Create VOB**.
2. On the VOB Creation Wizard - Name and Major Parameters page, complete these steps:
  - a. For the new VOB name, enter RedbookPVOB.
  - b. Clear **This VOB will contain UCM Components**.
  - c. Select **Create as a UCM project VOB** (See Figure A-82).

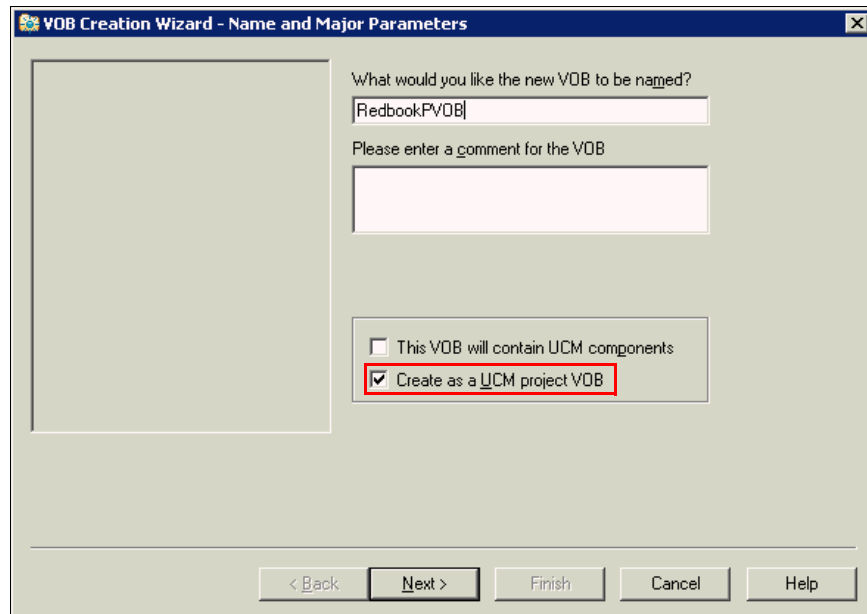


Figure A-82 Creating a UCM project VOB

- d. Select **Next**.
3. On the Vob Creation Wizard - Storage page, select **Use explicit path**. Select **Next**.
4. Optional: On the Vob Creation Wizard - Options page, select **Make this a public VOB** and enter the registry password.
5. On the confirmation dialog window, select **OK**.

Follow these steps to create a VOB, which is associated with the PVOB, to hold UCM components:

1. Select **Start** → **All programs** → **IBM Rational ClearCase** → **Administration** → **Create VOB**.
2. On the VOB Creation Wizard - Name and Major Parameters page, complete these steps:
  - a. For the new VOB name, enter RAD8RedbookUCM.
  - b. Select **This VOB will contain UCM Components**.
  - c. Clear **Create as a UCM project VOB**.
  - d. See Figure A-83.

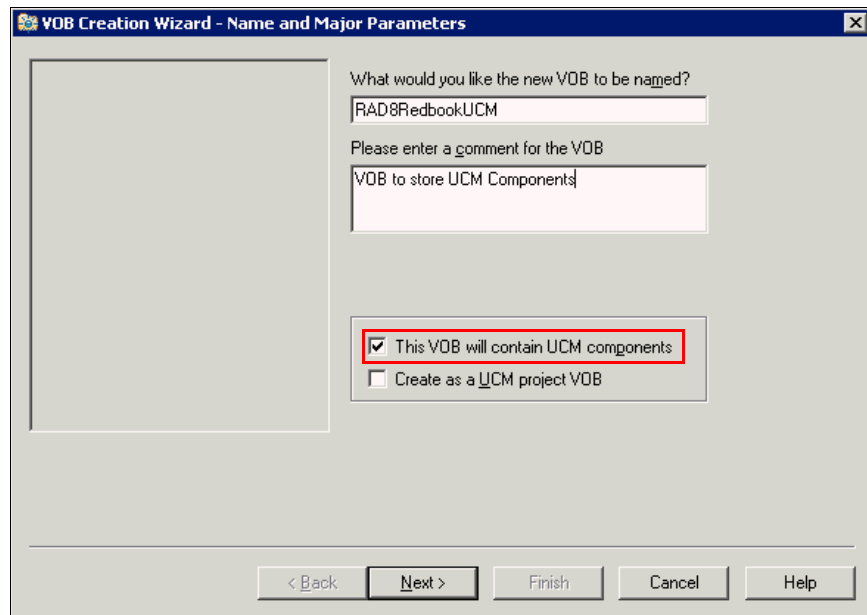


Figure A-83 Create a VOB to store UCM components

- e. Select **Next**.
3. On the VOB Creation Wizard - Components page, complete these tasks:
    - a. Select **Allow this VOB to contain multiple components**.
    - b. Add a new component called WebDev with a root directory called WebDev.
    - c. Select an existing view, such as lziosi\_base\_view.
    - d. See Figure A-84 on page 1863.

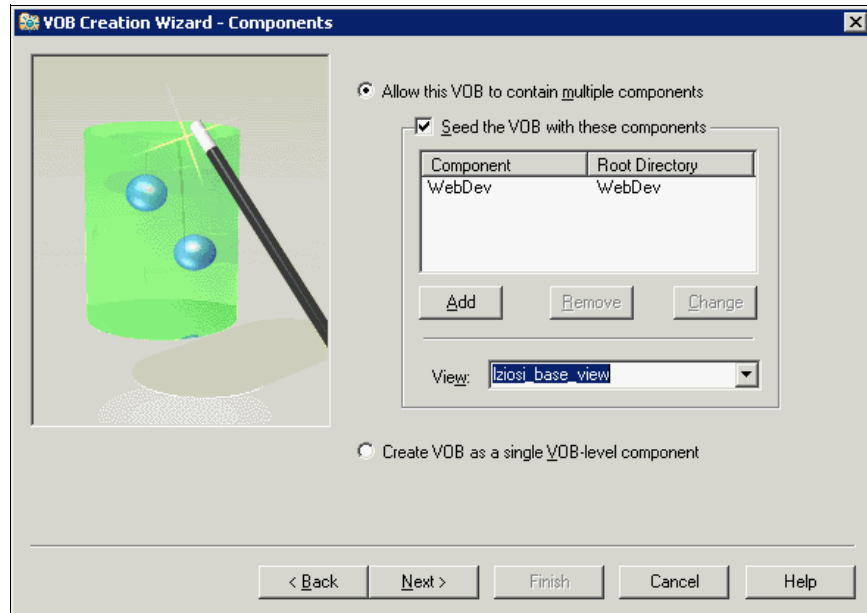


Figure A-84 Creating components for the UCM VOB

- e. Select **Next**.
4. On the VOB Creation Wizard - Storage page, select **Use explicit path**. Select **Next**.
5. On the VOB Creation Wizard - Options page, complete these steps:
  - a. For “What is the project VOB?”, select **RedbookPVOB**.
  - b. Optional: Select **Make this a public VOB** and enter the registry password.
6. On the confirmation window, select **OK**.

To create a UCM project, follow these steps:

1. Select **Start** → **All Programs** → **IBM Rational ClearCase** → **Project Explorer**. You see the PVOB RedbookPVOB and the component WebDev, as shown in Figure A-85 on page 1864.

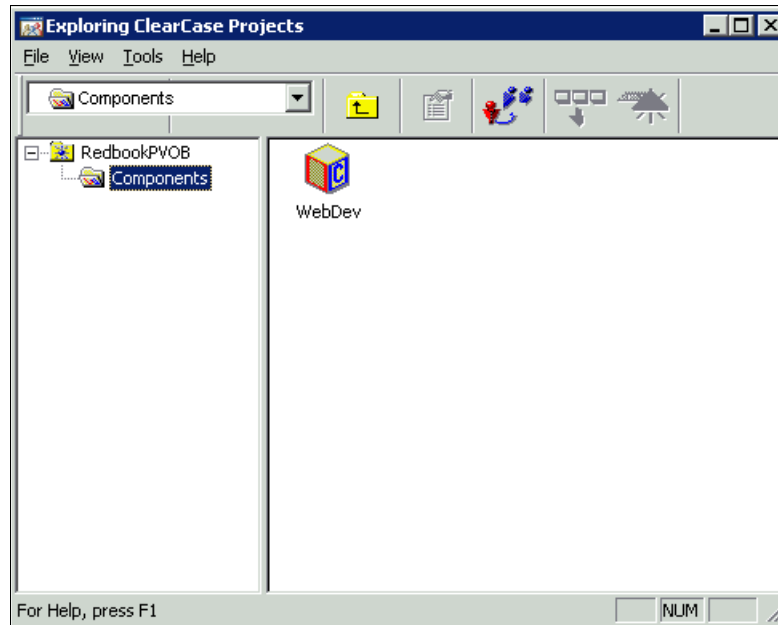


Figure A-85 ClearCase Project Explorer showing PVOB and component

2. Right-click the **RedbookPVOB** PVOB and select **New** → **Project**, as shown in Figure A-86 on page 1865.

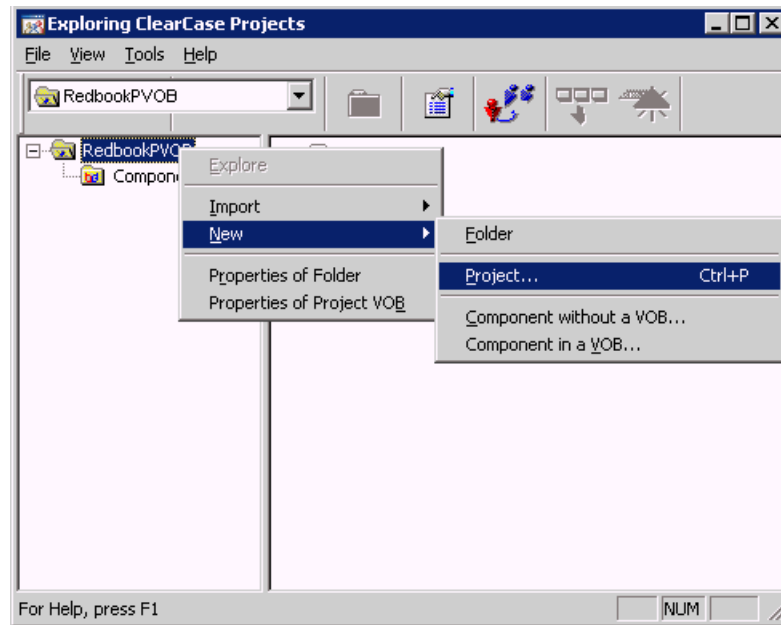


Figure A-86 Creating a new project in the ClearCase Project Explorer

3. On the New Project - Step 1 page, complete these steps:
  - a. For Project Name, enter RAD8Redbook.
  - b. For Integration Stream Name, enter RAD8Redbook\_integration.
  - c. For Project Type, select **Traditional parallel development**, as shown in Figure A-87 on page 1866.
  - d. Select **Next**.

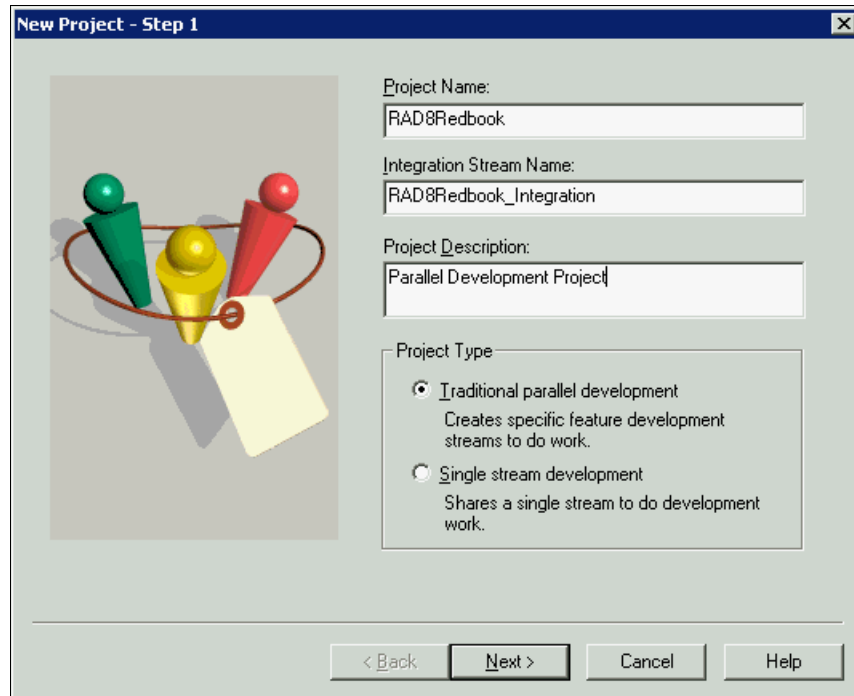


Figure A-87 Configuring the project for parallel development

4. On the New Project - Step 2 page, complete these steps:
  - a. For the question, “Would you like to seed this project with the recommended baselines?”, select **No**. There are no recommended baselines yet.
  - b. Select **Next**.
5. On the New Project - Step 3 page, complete these steps:
  - a. Add the component baselines to use in this project and select **Add**, as shown in Figure A-88 on page 1867.

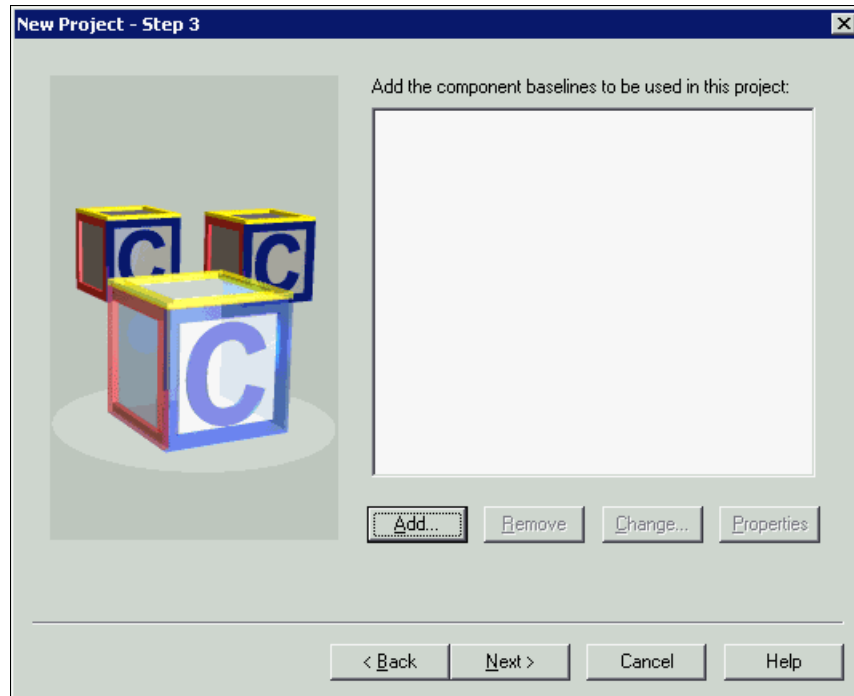


Figure A-88 Adding the component baselines to use in the UCM project

- b. Select the component **WebDev**.
- c. For From stream, select **all streams**.
- d. Select the Baseline Name **WebDev\_INITIAL**, as shown in Figure A-89.

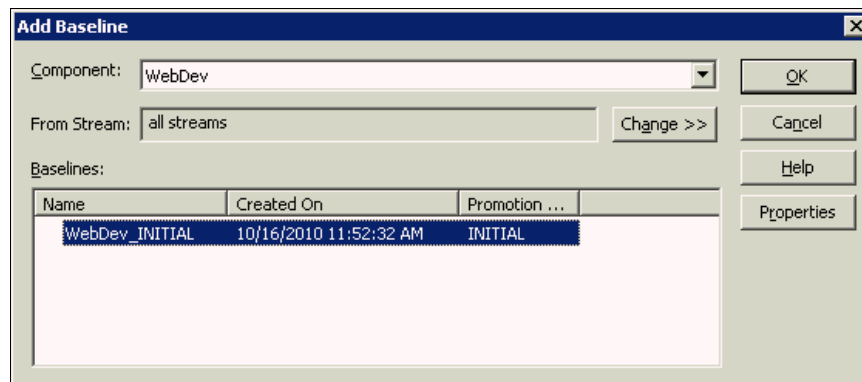


Figure A-89 Adding the Baseline WebDev\_INITIAL

- e. Select **OK** to close the Add Baseline window.
- f. Select **Next**.
1. On the New project - Step 4 page, perform these tasks:
  - a. For “Make the following components modifiable”, select **WebDev**.
  - b. Figure A-90 shows the multiple default promotion levels for recommending new baselines. Accept **INITIAL**.
  - c. Select **Next**.

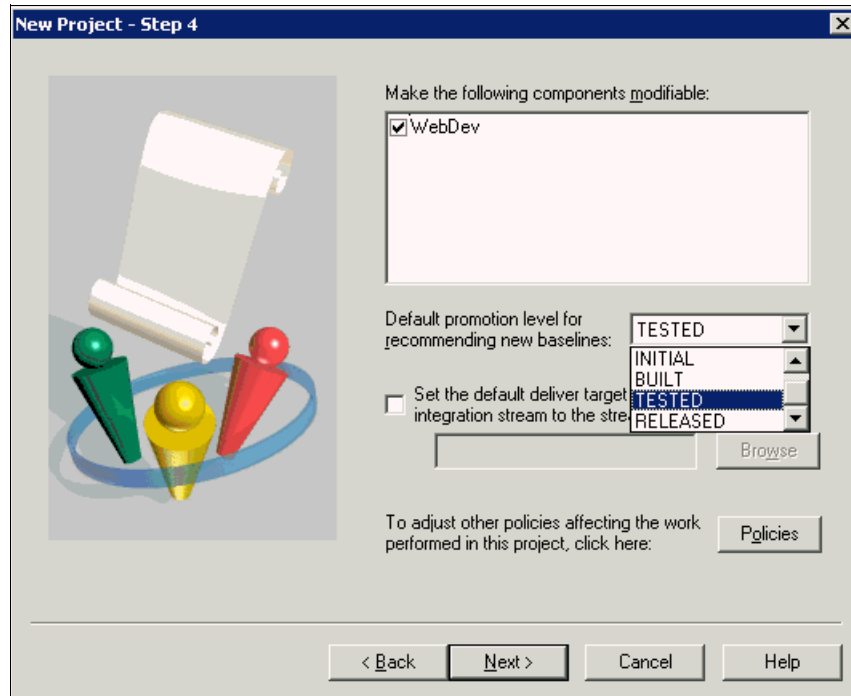


Figure A-90 Making components modifiable

2. On the New Project - Step 5 page, complete these tasks:
  - a. For “Should this project be ClearQuest-enabled?”, select **No**.
  - b. Select **Finish**.
3. On the New project - Confirmation page, select **OK**.

In the Project Explorer, you now see the PVOB, RedbookPV0B, the UCM Project RAD8Redbook, the integration stream, RAD8Redbook\_Integration, as shown in Figure A-91 on page 1869.



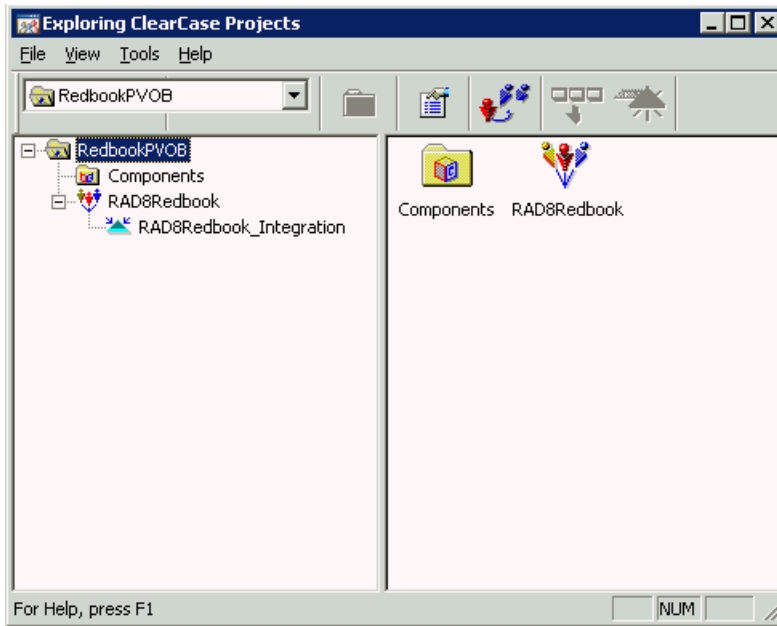


Figure A-91 ClearCase Project Explorer after completion of project creation





# B

## Performance tips for Rational Application Developer

This appendix describes ten performance tips for improving your development experience while using Rational Application Developer. You can obtain additional performance information in the Rational Application Developer help text, in the topic **Troubleshooting and Support** → **Improving Performance**.

## Better hardware

Rational Application Developer was built to make the best possible use of today's hardware, so using modern hardware runs everything faster. Rational Application Developer is multithreaded, so having a multiple core machine (dual core or quad core) allows certain operations to run in parallel and provides improved user interface (UI) responsiveness.

Rational Application Developer is a large application and our clients tend to build large applications. Installing more memory improves performance. When planning how much memory you need, you also need to consider what is running in addition to Rational Application Developer, and adjust accordingly.

If you need more than 3.5 GB of memory, we recommend using a 64-bit operating system. A 32-bit Microsoft Windows operating system typically cannot exploit more than 3.5 GB because of the way that device drivers reserve memory addresses.

Performance is not merely about the CPU; everything helps: faster bus, faster memory, and faster disks. You do not need the fastest, most expensive system that is currently available, but usually about two models down from that system is optimum in terms of price and performance.

Improving your hardware makes everything faster.

## Shared EARs (binary modules)

With large development projects, not every team member needs to change every project, every day. You can achieve a large performance win by following this guideline and using shared Enterprise Application Archives (EARs).

Keep code that you need to change in source form and keep everything else in binary form. Rational Application Developer automatically attaches source to the binary modules so that you can debug and view all the code, almost as if all of the code was in source form.

Build and validation are much quicker because only the source projects are built and validated. For more information, see the following paper *Using binary modules to optimize Rational Application Developer in a team environment*.

[http://www-128.ibm.com/developerworks/rational/library/07/0619\\_karasiuk\\_sho11/](http://www-128.ibm.com/developerworks/rational/library/07/0619_karasiuk_sho11/)

The effect of this tip is faster builds, publishing, refactoring, and less memory usage. The more modules that you use in their binary form, the bigger the performance benefit.

## Annotations

Annotations are expensive to process. You can reduce this time greatly by providing instructions about what does not need to be processed. The effect of this tip is faster publishing.

For more information, see the Rational Application Developer Help text by selecting **Troubleshooting and Support** → **Improving Performance** → **Performance Tips** → **Publishing and annotations**.

## Publishing

There are many ways to improve the amount of time that it takes to publish your application:

- ▶ Ensure that your projects are single-rooted. Projects that are single-rooted can be directly consumed by WebSphere Application Server. If the project is not single-rooted, it needs to be copied as part of the publishing operation. If a project is not single-rooted, you see a warning message in the Marker's view.
- ▶ Minimize the number of Java archives (JARs) in the WEB-INF/lib directories. If the JAR is no longer needed, remove it. If you see that the same jar is repeated in many lib directories, see if it can be moved to a shared directory on the server.
- ▶ Start the test server in debug mode. This approach, which allows you to change the code by using the "hot method replace" and directly inserting the code into the running server, eliminates the need to perform a publish. There are cases where the Java virtual machine (JVM) cannot replace a method. In those cases, you get a warning message and then you need to publish the application.
- ▶ Use Remote Method Invocation (RMI) as the connection type, because no polling is involved.
- ▶ Remove unused applications. Removing unused applications shortens up your server start-up time.
- ▶ Turn off your server's security, which improves the start-up time for your server as well as improving your publish times.

- ▶ Periodically clean out the server's `ws-temp` and `temp` directories.
- ▶ Publish is an expensive operation because of all of the Rational Application Developer code, WebSphere Application Server code, and application code, for example, servlet destruction code, that must be processed. The best publish is no publish. It is best if you are in control of when publishes occur, so we recommend turning off automatic publishes (Figure B-1).

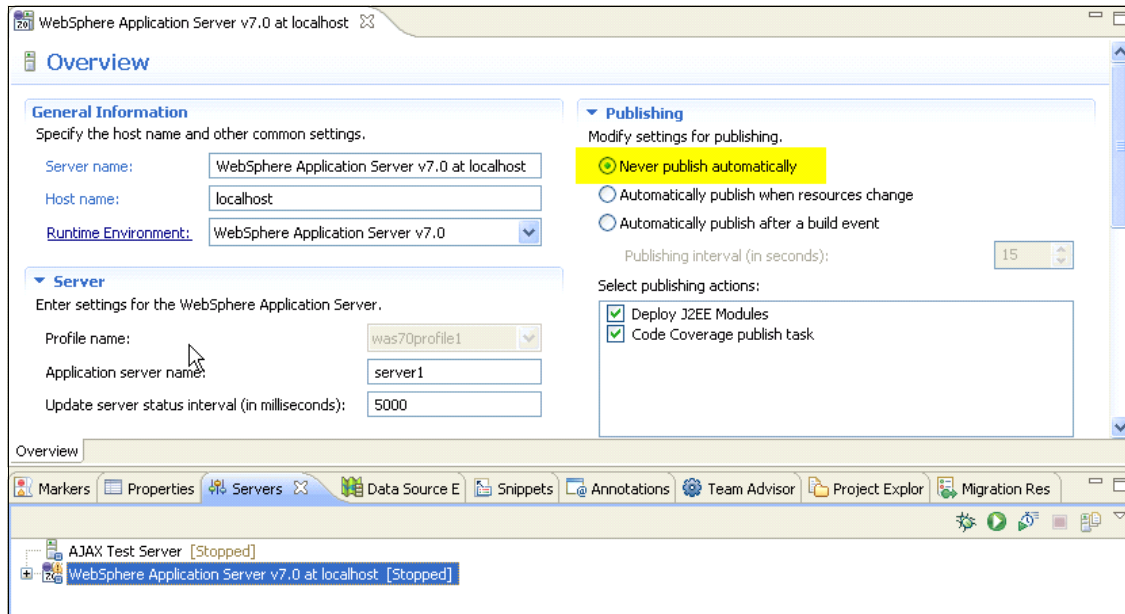


Figure B-1 Turning off automatic publishing

## Shorter build time by tuning validation

Many customization options exist for validation. You can set when to perform validation, turn individual validators on or off, and control the types of resources that are validated. Explore the Validation preference page **Windows** → **Preferences** → **Validation** to see the customization options.

Many clients use a strategy to turn off build-time validation and to perform periodic manual validations instead. The most expensive validators are the *JavaServer Faces (JSF)*, *JavaServer Pages (JSP)*, and *HTML* validators.

By tuning validation, you see the largest improvement in the full builds and clean builds.

## Only install what you need

Install only the components that you use. Plug-ins, which are not activated, still have a memory cost. It is better to add components when you need them, rather than adding them in anticipation of needing them in the future. This approach saves memory, time, and disk space.

Also, download the install and update images one time only and share them among your team members.

## No circular dependencies

Ensure that your build paths are clean. Projects must not have dependency cycles. For example, if project A depends on project B, project B must not depend on project A. Having cycles forces the builds to iterate. Rational Application Developer warns you if you have circular dependencies. Circular dependencies are a sign of poor coding practice and an indication that your projects are structured improperly.

Use Code Coverage to identify “dead” code and use refactoring to eliminate dead code.

Implementing this tip improves the speed of your builds and gives you code that is easier to maintain.

## Using a remote test server

If you realize that you do not have enough memory, that you cannot add more memory, and that your system is thrashing, consider using a remote test server instead of running a local test server.

Your test server is often the largest or second largest process on your system. By running your test server on another system, you free up more resources for Rational Application Developer.

The *Setting up a Remote WebSphere Test Server for Multiple Developers* paper explains this concept in more detail. Although this paper applies to a much earlier version of Rational Application Developer, the concepts still apply today:

[http://www-128.ibm.com/developerworks/websphere/library/techarticles/03\\_03\\_karasiuk/karasiuk.html](http://www-128.ibm.com/developerworks/websphere/library/techarticles/03_03_karasiuk/karasiuk.html)

Before moving your test server, consider moving several of your other large processes first. For example, if you run a database server on your local machine, move it before you move your test server.

One advantage in keeping your test server on your local machine is that your publishing process can be faster.

## Tuning your anti-virus program

Try to store Rational Application Developer in a directory that does not need continuous anti-virus scans. Also, try to store your workspace in a directory that does not need continuous anti-virus scans

If you run an anti-virus program, try to use the current version of it. The anti-virus program that you use can make a big difference.

For example, one user described this experience:

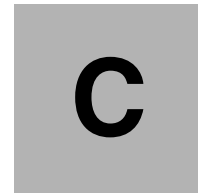
- ▶ First-time user opens a JSP file in Rational Application Developer:
  - Anti-virus program on: 90 seconds
  - Anti-virus program off: 10 seconds
- ▶ Cold start-up of Rational Application Developer:
  - Anti-virus program on: 3 minutes
  - Anti-virus program off: 1 minute

## Defragmenting disks

Reducing the amount of fragmentation means that your files can be read quicker. The disk arm does not have to move as much if the pieces of the file are not scattered in various disk locations. Defragmenting your disks weekly is a good idea. Set it up and forget it.

This tip affects operations that need to read many files, such as Rational Application Developer. We measured a 7% improvement in Rational Application Developer's cold start-up time by defragmenting the disk.





## Additional material

The additional material is a web download of the sample code for this book. This appendix describes how to download, unpack, and describe the contents, and import the project interchange file.

This appendix is organized in the following sections:

- ▶ Locating the web material
- ▶ Unpacking the sample code
- ▶ Description of the sample code
- ▶ Setting up the ITSOBANK database
- ▶ Configuring the data source in WebSphere Application Server
- ▶ Importing sample code from a project archive file

## Locating the web material

The web material that is associated with this book is available in softcopy on the Internet from the IBM Redbooks web server. Enter the following URL in a web browser and then download the two ZIP files:

<ftp://www.redbooks.ibm.com/redbooks/SG247835>

Alternatively, you can go to the IBM Redbooks website:

<http://www.ibm.com/redbooks>

## Accessing the web material

Select **Additional materials** and open the directory that corresponds with the IBM Redbooks publication form number, SG24-7835.

The additional web material that accompanies this IBM Redbooks publication includes the following files:

<i>File name</i>	<i>Description</i>
<b>7835code.zip</b>	Compressed file that contains sample code
<b>7835codesolution.zip</b>	Compressed file that contains solution interchange files

## System requirements for downloading the web material

We recommend the following system configuration:

<b>Hard disk space:</b>	20 GB minimum
<b>Operating system:</b>	Microsoft Windows or Linux
<b>Processor:</b>	2 GHz
<b>Memory:</b>	2 GB

## Using the sample code

In this section, we provide a description of the sample code and how to use it.

### Unpacking the sample code

After you have downloaded the two compressed files, extract the files to your local file system using 7Zip, or similar software. For example, we extract the 7835code.zip and 7835codesolution.zip files to the C:\ drive, which creates

C:\7835code. Throughout the samples, we reference the sample code as though you have unpacked the files to the C drive.

## Description of the sample code

Table C-1 lists the contents of the sample code after unpacking. The 7835code folder has the following major sections:

- ▶ Code sample to follow the instructions in a chapter
- ▶ Interchange files with the solution for each chapter

*Table C-1 Sample code description*

Directory	Specific chapter for this code
c:\7835code	Root directory after unpacking the sample code
..\java	Chapter 7, “Developing Java applications” on page 229
..\xml	Chapter 8, “Developing XML applications” on page 331
..\database	Chapter 9, “Developing database applications” on page 393. Also, this directory includes the code to set up the ITSOBANK database in either Derby or DB2.
..\jpa	Chapter 10, “Persistence using the Java Persistence API” on page 443
..\ejb	Chapter 12, “Developing Enterprise JavaBeans (EJB) applications” on page 577
..\j2eeclient	Chapter 13, “Developing Java Platform, Enterprise Edition (Java EE) application clients” on page 649
..\webservice	Chapter 14, “Developing web services applications” on page 681
..\osgi	Chapter 15, “Developing Open Services Gateway initiative (OSGi) applications” on page 837
..\sca	Chapter 16, “Developing Service Component Architecture (SCA) applications” on page 885
..\webapp	Chapter 18, “Developing web applications using JavaServer Pages (JSP) and servlets” on page 981
..\jsf	Chapter 19, “Developing web applications using JavaServer Faces” on page 1057
..\web20	Chapter 20, “Developing web applications using Web 2.0” on page 1097
..\portal	Chapter 21, “Developing portal applications” on page 1131
..\ant	Chapter 24, “Building applications with Apache Ant” on page 1279

Directory	Specific chapter for this code
..\jython	Chapter 25, "Deploying enterprise applications" on page 1309
..\junit	Chapter 26, "Testing using JUnit" on page 1365
..\js	Chapter 28, "Debugging local and remote applications" on page 1461
..\codecoverage	Chapter 32, "Code Coverage" on page 1697
..\sip	Chapter 33, "Developing Session Initiation Protocol applications" on page 1727
c:\7835codesolution	Solution files for multiple chapters

## Importing sample code from a project archive file

In this section, we explain how to import the sample code (for this book) from the compressed project archive files into Rational Application Developer. This section applies to each of the chapters that contain sample code, which has been packaged as a compressed project interchange file.

To import an existing project into a Rational Application Developer workspace, perform the following steps:

1. In the Java EE (or web) perspective, Enterprise Explorer, select **File** → **Import**.
2. Select **Import**, expand **General**, and select **Existing Projects into Workspace** from the list of import sources. Then click **Next**.
3. In the Import Projects window, select **Select archive file** and click **Browse**. Locate the appropriate compressed file in the c:\7835code\ or c:\7835codesolution\ folder and click **Open**.
4. Click **Select All** to select all projects and then click **Finish**.

## Setting up the ITSOBANK database

We provide two implementations of the ITSOBANK database: Derby and DB2 Universal Database. You can choose to implement either or both databases and then set up the enterprise applications to use one of the databases. The Derby database system ships with WebSphere Application Server.

## Derby

The C:\7835code\database\derby directory provides command files to define and load the ITSOBANK database in Derby. For the DerbyCreate.bat, DerbyLoad.bat, and DerbyList.bat files, you must have installed WebSphere Application Server in the C:\IBM\WebSphere\AppServer folder. You must edit these files to point to your WebSphere Application Server installation directory if you installed the product in a separate folder.

In the C:\7835code\database\derby directory, you can perform the following actions:

- ▶ Execute the DerbyCreate.bat file to create the database and table.
- ▶ Execute the DerbyLoad.bat file to delete the existing data and add records.
- ▶ Execute the DerbyList.bat file to list the contents of the database.

These command files use the SQL statements and helper files that are provided in the following files:

- ▶ itsobank.ddl: Database and table definition
- ▶ itsobank.sql: SQL statements to load sample data
- ▶ itsobanklist.sql: SQL statement to list the sample data
- ▶ tables.bat: Command file to execute itsobank.ddl statements
- ▶ load.bat: Command file to execute itsobank.sql statements
- ▶ list.bat: Command file to execute itsobanklist.sql statements

The Derby ITSOBANK database is created in the C:\7835code\database\derby\ITSOBANK directory.

## DB2

The C:\7835code\database\db2 folder provides the DB2 command files to define and load the ITSOBANK database. You can perform the following actions:

- ▶ Execute the createbank.bat file to define the database and table.
- ▶ Execute the loadbank.bat file to delete the existing data and add records.
- ▶ Execute the listbank.bat file to list the contents of the database.

These command files use the SQL statements that are provided in the following files:

- ▶ itsobank.ddl: Database and table definition
- ▶ itsobank.sql: SQL statements to load sample data
- ▶ itsobanklist.sql: SQL statement to list the sample data

# Configuring the data source in WebSphere Application Server

In this section, we explain how to configure the data source in the WebSphere administrative console. We configure the data source against the WebSphere Application Server v8.0 Beta test environment that ships with Rational Application Developer.

We followed these high-level configuration steps to configure the data source within WebSphere Application Server for the ITSOBANK database:

1. Starting the WebSphere Application Server
2. Configuring the environment variables
3. Configuring J2C authentication data
4. Configuring the JDBC provider
5. Creating the data source

## Starting the WebSphere Application Server

If you use a stand-alone WebSphere Application Server v8.0 Beta, enter the following commands in a command window:

```
cd \IBM\WebSphere\AppServer\profiles\AppSrv01\bin
startServer.bat server1
```

If you use the WebSphere Application Server V8 Beta test environment that ships with Rational Application Developer, in the Servers view, right-click **WebSphere Application Server v8.0 Beta at localhost** and select **Start**.

## Configuring the environment variables

Prior to configuring the data source, ensure that the environment variables are defined for the desired database server type. This step does not apply to Derby because we use the embedded Derby, which already has the variables defined. For example, if you choose to use DB2 Universal Database, you must verify the path of the driver for DB2 Universal Database. Perform these steps:

1. Start the WebSphere administrative console:
  - If you use the WebSphere Application Server V7.0 test environment that ships with Application Developer, right-click **WebSphere Application Server v8.0 Beta** and select **Administration** → **Run administrative console**.
  - For a stand-alone server, type the following URL in a web browser:

<http://localhost:9062/ibm/console>

You might need to specify a port other than 9062. The administrative console port was chosen during the installation of the server profile.

2. Click **Log in**. If you installed WebSphere with administrative security enabled, use the user ID and password that were chosen at installation time (for example, admin/admin).
3. Expand **Environment** → **WebSphere variables**.
4. Scroll down the page, click the desired variable, and update the path accordingly for your installation:
  - For Derby, use DERBY\_JDBC\_DRIVER\_PATH.  
By default, this variable is already configured because Derby is installed with WebSphere Application Server:  
`${WAS_INSTALL_ROOT}/derby/lib`
  - For DB2, use DB2UNIVERSAL\_JDBC\_DRIVER\_PATH.  
Edit the value and provide the DB2 installation directory, for example, C:\IBM\SQLLIB\java or C:\SQLLIB\java.
5. Click **Save** (at the top).

## Configuring J2C authentication data

In this section, we explain how to configure the J2EE Connector architecture (J2C) authentication data (database login and password) for WebSphere Application Server from the WebSphere administrative console. This step is required for DB2 Universal Database, but it is optional for Derby.

If you use DB2 Universal Database, configure the J2C authentication data (database login and password) for WebSphere Application Server from the administrative console:

1. Select **Security** → **Global security**.
2. Under the Authentication properties, expand **Java Authentication and Authorization Service** and select **J2C authentication data**.
3. Click **New**.
4. Enter the Alias, User ID, and Password in the Java Authentication and Authorization Service (JAAS) J2C Authentication data page. For example, create an alias called db2user with the user ID and password that were used when installing DB2.
5. Click **Save**.

## Configuring the JDBC provider

In this section, we explain how to configure the Java Database Connectivity (JDBC) provider for the selected database type. In the following procedure, we demonstrate how you can configure the JDBC provider for Derby. We provide notes about how to perform the equivalent steps in DB2 Universal Database.

To configure the JDBC provider from the WebSphere administrative console, follow these steps:

1. Select **Resources** → **JDBC** → **JDBC providers**.

2. Select the scope settings.

Select the server scope from the drop-down menu. In our case, we select **Node=<machine>NodeXX, Server=server1**.

3. Click **New**.

4. On the New JDBC Providers page, follow these steps:

a. For the Database Type, select **Derby**.

b. For the JDBC Provider, select **Derby JDBC Provider**.

c. For the Implementation type, select **XA data source**.

d. Click **Next**.

For DB2 Universal Database, follow these steps on the New JDBC Providers page:

a. For the Database Type, select **DB2**.

b. For the JDBC Provider, select **DB2 Universal JDBC Driver Provider**.

c. For the Implementation type, select **XA data source**.

d. Click **Next**.

5. Click **Finish**.

## Creating the data source

To create the data source for Derby, follow these steps:

1. Select the **Derby JDBC Provider (XA)**.

2. Under Additional Properties, select **Data sources**.

3. Click **New**.

4. For the Data source name, type `ITSOBANKderby`, and for Java Naming and Directory Interface (JNDI) name, type `jdbc/itsobank`. Click **Next**.

5. For the Database name, type `C:\7835code\database\derby\ITSOBANK`. Clear **Use this data source in container managed persistence (CMP)**, because Java Persistence API (JPA) does not use CMP. Click **Next**.



6. Skip the Set up security aliases page. Click **Next**.
7. Click **Finish** and then click **Save**.
8. Verify the connection by selecting **ITSOBANKderby** (the check box) and then click **Test connection**. You see the following message:

The test connection operation for data source ITSOBANKderby on server server1 at node xxxxxNodexx was successful.

If you use DB2, create a data source for **JDBC Providers** → **DB2 Universal JDBC Provider Driver (XA)**:

1. Select **Data sources** and click **New**.
2. For Data source name, type ITSOBANKdb2, and for JNDI name, type jdbc/itsobankdb2.
3. For Database name, type ITSOBANK, and for Server name, type localhost. For Driver type, leave **4** as the value, and for Port number, leave **50000** as the value. Clear **Use this data source in container managed persistence (CMP)**.
4. Select the **db2user** alias for Authentication alias for XA recovery and for Component-managed authentication alias. Click **Next**.
5. Click **Finish** and then click **Save**.
6. Verify the connection by selecting **ITSOBANKdb2** (the check box) and then click **Test connection**. You see a message indicating that the connection is successful.

**DB2 for the ITSOBANK database:** If you always want to use DB2 for the ITSOBANK database, set the JNDI name to jdbc/itsobank for the DB2 data source and to jdbc/itsobankderby for Derby. The sample application uses jdbc/itsobank to access the database, which can be Derby or DB2.



# Abbreviations and acronyms

<b>AOP</b>	aspect-oriented programming	<b>CVS</b>	Concurrent Versioning System
<b>API</b>	application programming interface	<b>DI</b>	dependency injection
<b>ATM</b>	automatic teller machine	<b>DMS</b>	Data Mediator Services
<b>AWT</b>	Abstract Window Toolkit	<b>DOM</b>	Document Object Model
<b>B2BUA</b>	Back-to-Back User Agent	<b>DTD</b>	document type definition
<b>BCI</b>	byte-code instrumentation	<b>EAI</b>	Enterprise Application Integration
<b>BIRT</b>	Business Intelligence and Reporting Tools	<b>EAM</b>	Enterprise Asset Management
<b>BLOBs</b>	Binary Large Objects	<b>EAR</b>	Enterprise Application Archive
<b>BMC</b>	bean-managed concurrency	<b>EBA</b>	Enterprise bundle archives
<b>BSF</b>	Bean Scripting Framework	<b>EIS</b>	enterprise information systems
<b>BVT</b>	build verification test	<b>EJB</b>	Enterprise JavaBean
<b>CAR</b>	configuration archive	<b>EL</b>	Expression Language
<b>CBA</b>	Composite bundle archives	<b>EPR</b>	endpoint reference
<b>CCI</b>	Common Client Interface	<b>FVT</b>	function verification test
<b>CEA</b>	Communications Enabled Applications	<b>GIS</b>	geographic information system
<b>CEI</b>	Common Event Infrastructure	<b>GUI</b>	graphical user interface
<b>CI</b>	Compute-Intensive	<b>HTML</b>	Hypertext Markup Language
<b>CLI</b>	command-line interface	<b>HTTP</b>	Hypertext Transfer Protocol
<b>CLOB</b>	Character Large Object	<b>IDE</b>	integrated development environment
<b>CMC</b>	container-managed concurrency	<b>IDL</b>	Interface Definition Language
<b>CMP</b>	Container Managed Persistence	<b>IMS</b>	Information Management System
<b>CMTs</b>	container-managed transactions	<b>IP</b>	Internet Protocol
<b>CORBA</b>	Common Object Request Broker Architecture	<b>IPC</b>	Inter Process Communication
<b>CSS</b>	Cascading Style Sheet	<b>J2C</b>	Java EE Connector
<b>CSTE</b>	Certified Software Tester	<b>J2EE</b>	Java 2 Platform Enterprise Edition
<b>CSV</b>	comma-separated values		

<b>JAAS</b>	Java Authentication and Authorization Service	<b>MVC</b>	model view controller
<b>JAF</b>	Java Activation Framework	<b>MVFS</b>	multiversion file system
<b>JAR</b>	Java archive	<b>OC4J</b>	Oracle Containers for J2EE
<b>JAXB</b>	Java Architecture for XML Binding	<b>ODA</b>	Open Data Access
<b>JAXP</b>	Java API for XML Processing	<b>OMG</b>	Object Management Group
<b>JAXR</b>	Java API for XML Registries	<b>ORM</b>	Object Relational Mapping
<b>JCA</b>	Java EE Connector Architecture	<b>OSGi</b>	Open Services Gateway initiative
<b>JDBC</b>	Java Database Connectivity	<b>OSOA</b>	Open Service Oriented Architecture
<b>JDO</b>	Java Data Objects	<b>PDA</b>	personal digital assistant
<b>JDT</b>	Java development tools	<b>PDD</b>	Portlet Deployment Descriptor
<b>JET</b>	Java Emitter Templates	<b>PKI</b>	public key infrastructure
<b>JMC</b>	Job Management Console	<b>POJI</b>	plain old Java interface
<b>JMS</b>	Java Message Service	<b>POJO</b>	Plain Old Java Object
<b>JMX</b>	Java Management Extension	<b>PTP</b>	Point-to-point
<b>JNDI</b>	Java Naming and Directory Interface	<b>RA</b>	resource adapter
<b>JNI</b>	Java Native Interface	<b>RAMP</b>	Reliable Asynchronous Messaging Profile
<b>JOnAS</b>	Java Open Application Server	<b>RAR</b>	resource adapter archive
<b>JPA</b>	Java Persistence API	<b>RCP</b>	Rich Client Platform
<b>JPQL</b>	Java Persistence Query Language	<b>REST</b>	Representational State Transfer
<b>JRE</b>	Java Runtime Environment	<b>RMI</b>	Remote Method Invocation
<b>JSF</b>	JavaServer Faces	<b>RPC</b>	Remote Procedure Call
<b>JSON</b>	JavaScript Object Notation	<b>RTF</b>	Result Tree Fragment
<b>JSP</b>	JavaServer Pages	<b>SAML</b>	Security Assertion Markup Language
<b>JSTL</b>	JavaServer Pages Standard Tag Library	<b>SAX</b>	Simple API for XML
<b>JTA</b>	Java Transaction API	<b>SCA</b>	Service Component Architecture
<b>JVM</b>	Java Virtual Machine	<b>SDO</b>	Service Data Object
<b>JVMTI</b>	JVM Tools Interface	<b>SEI</b>	service endpoint interface
<b>MDB</b>	Message-driven bean	<b>SGML</b>	Standard Generalized Markup Language
<b>MIME</b>	Multipurpose Internet Mail Extensions	<b>SIB</b>	service integration bus
<b>MTOM</b>	Message Transmission Optimization Mechanism	<b>SIP</b>	Session Initiation Protocol

<b>SLF4J</b>	Simple Logging Facade for Java	<b>XOP</b>	XML-binary Optimized Packaging
<b>SOA</b>	service-oriented architecture	<b>XPath</b>	XML Path Language
<b>SPF</b>	Struts Portal Framework	<b>XQuery</b>	XML Query Language
<b>SPI</b>	Service Programming Interfaces	<b>XSD</b>	XML Schema Definition
<b>SSH</b>	Secure Shell	<b>XSLT</b>	XSL Transformation
<b>SSN</b>	Social Security number		
<b>SSO</b>	single sign-on		
<b>STS</b>	Security Token Service		
<b>SVG</b>	Scalable Vector Graphics		
<b>SVT</b>	system verification test		
<b>SWT</b>	Standard Widget Toolkit		
<b>StAX</b>	Streaming API for XML		
<b>TPTP</b>	Test and Performance Tools Platform		
<b>UCM</b>	Unified Change Management		
<b>UDDI</b>	Universal Description, Discovery, and Integration		
<b>UDF</b>	user-defined function		
<b>UEL</b>	Unified Expression Language		
<b>UI</b>	User interface		
<b>UML</b>	Unified Modeling Language		
<b>URI</b>	Uniform Resource Identifier		
<b>UTC</b>	Universal Test Client		
<b>UUID</b>	universally unique identifier		
<b>VOIP</b>	Voice over IP		
<b>W3C</b>	World Wide Web Consortium		
<b>WAP</b>	Wireless Application Protocol		
<b>WAR</b>	web archive		
<b>WML</b>	Wireless Markup Language		
<b>WSDL</b>	Web Services Description Language		
<b>WSRP</b>	Web Services for Remote Portlets		
<b>WTP</b>	Web Tools Platform		



# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics that are covered in this book.

## IBM Redbooks publications

For information about ordering these publications, see “How to get IBM Redbooks publications” on page 1897. Note that several of the documents referenced here might be available in softcopy only.

- ▶ *Building Composite Applications*, SG24-7367
- ▶ *Building Dynamic Ajax Applications Using WebSphere Feature Pack for Web 2.0*, SG24-7635
- ▶ *Building SOA Solutions Using the Rational SDP*, SG24-7356
- ▶ *Experience Java EE! Using WebSphere Application Server Community Edition 2.1*, SG24-7639
- ▶ *Getting Started with WebSphere Enterprise Service Bus V6*, SG24-7212
- ▶ *IBM Rational Application Developer V6 Portlet Application Development and Portal Tools*, SG24-6681
- ▶ *IBM WebSphere Application Server V6.1 Security Handbook*, SG24-6316
- ▶ *IBM WebSphere Portal V5 A Guide for Portlet Application Development*, SG24-6076
- ▶ *Patterns: Extended Enterprise SOA and Web Services*, SG24-7135
- ▶ *Portal Application Design and Development Guidelines*, REDP-3829
- ▶ *Rational Application Developer V7 Programming Guide*, SG24-7501
- ▶ *Topics on Version 7 of IBM Rational Developer for System z and IBM WebSphere Developer for System z*, SG24-7482
- ▶ *Using Rational Performance Tester Version 7*, SG24-7391
- ▶ *Web Services Feature Pack for WebSphere Application Server V6.1*, SG24-7618
- ▶ *Web Services Handbook for WebSphere Application Server 6.1*, SG24-7257
- ▶ *WebSphere Application Server V6.1: Planning and Design*, SG24-7305

- ▶ *WebSphere Application Server V6.1: System Management and Configuration*, SG24-7304
- ▶ *WebSphere Application Server V7.0: Technical Overview*, REDP-4482
- ▶ *WebSphere Application Server Version 6.1 Feature Pack for EJB 3.0*, SG24-7611
- ▶ *WebSphere Studio 5.1.2 JavaServer Faces and Service Data Objects*, SG24-6361
- ▶ *Rational Application Developer V7.5 Programming Guide*, SG24-7672
- ▶ *Getting Started with the WebSphere Application Server Feature Pack for Communications Enabled Applications V1.0*, REDP-4613
- ▶ *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework*, SG24-6302

## Other publications

These publications are also relevant as further information sources:

- ▶ Eric Gamma, et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995, ISBN 0-201-63361-2
- ▶ D'Anjou, et al., *The Java Developer's Guide to Eclipse-Second Edition*, Addison Wesley, 2004, ISBN 0-321-30502-7

## Online resources

These websites are also relevant as further information sources:

- ▶ IBM WebSphere software  
<http://www.ibm.com/software/websphere>
- ▶ IBM Rational software  
<http://www.ibm.com/software/rational>
- ▶ WebSphere Information Center  
<http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp>
- ▶ Rational Application Developer Information Center  
<http://publib.boulder.ibm.com/infocenter/radhelp/v7r5/index.jsp>
- ▶ IBM Education Assistant  
<http://www.ibm.com/software/info/education/assistant>



- ▶ developerWorks  
<http://www.ibm.com/developerworks>
- ▶ alphaWorks  
<http://www.ibm.com/alphaworks>
- ▶ Eclipse  
<http://www.eclipse.org>
- ▶ Sun Java  
<http://java.sun.com>
- ▶ Java Community Process  
<http://www.jcp.org>
- ▶ Apache Derby database  
<http://db.apache.org/derby>
- ▶ OASIS  
<http://www.oasis-open.org>
- ▶ Jython  
<http://www.jython.org>
- ▶ Web Services Interoperability Organization  
<http://www.ws-i.org>
- ▶ *Core J2EE Patterns: Best Practices and Design Strategies* by Crupi, et al.  
<http://java.sun.com/blueprints/corej2eepatterns>
- ▶ *EJB Design Patterns: Advanced Patterns, Processes and Idioms* by Marinescu  
<http://www.theserverside.com/news/1369776/Free-Book-EJB-Design-Patterns>
- ▶ *The Java Developer's Guide to Eclipse-Second Edition*, by D'Anjou et al.  
<http://jdg2e.com>
- ▶ IBM developerWorks for Rational  
<http://www.ibm.com/developerworks/rational/>
- ▶ Rational Edge articles focusing on UML topics  
<http://www.ibm.com/developerworks/rational/library/content/RationalEdge/archives/uml.html>
- ▶ IBM Rational Software UML Resource Center  
<http://www-01.ibm.com/software/rational/uml/index.html>

- ▶ Object Management Group (OMG):
  - <http://www.omg.org>
  - <http://www.uml.org>
- ▶ Craig Larman's home page  
<http://www.craiglarman.com/>
- ▶ Oracle Java SE  
<http://www.oracle.com/technetwork/java/javase/index.html>
- ▶ IBM developerWorks Java Technology  
<http://www.ibm.com/developerworks/java/>
- ▶ What's new in Eclipse 3.6  
[http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.jdt.doc.user/whatsNew/jdt\\_whatsnew.html](http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.jdt.doc.user/whatsNew/jdt_whatsnew.html)
- ▶ Eclipse tips and tricks  
[http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.jdt.doc.user/tips/jdt\\_tips.html](http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.jdt.doc.user/tips/jdt_tips.html)
- ▶ SAX  
<http://www.saxproject.org/>
- ▶ DOM  
<http://www.w3.org/DOM/>
- ▶ StAX  
<http://www.jcp.org/en/jsr/detail?id=173>
- ▶ Extensible Markup Language 1.0 (Fourth Edition)  
<http://www.w3.org/TR/2006/REC-xml-20060816/>
- ▶ XML Schema 1.1 status  
<http://www.w3.org/XML/Schema>
- ▶ XSLT  
<http://www.w3.org/TR/xslt.html>
- ▶ XSLT 2.0  
<http://www.w3.org/TR/xslt20/>
- ▶ XPath  
<http://www.w3.org/TR/xpath>
- ▶ XPath 2.0  
<http://www.w3.org/TR/xpath20/>

- ▶ WebSphere Application Server V7 Feature Pack for XML  
[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.xmlfep.multiplatform.doc/info/ae/ae/welcome\\_fep\\_xml.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.xmlfep.multiplatform.doc/info/ae/ae/welcome_fep_xml.html)
- ▶ XML schemas  
<http://www.w3.org/XML/Schema>
- ▶ XML  
<http://www.w3.org/XML/>
- ▶ Xerces (XML parser - Apache)  
<http://xml.apache.org/xerces2-j>
- ▶ Xalan (XSLT processor - Apache)  
<http://xml.apache.org/xalan-j>
- ▶ JAXP (XML parser - Sun)  
<http://java.sun.com/xml/jaxp>
- ▶ SAX2 (XML API)  
<http://sax.sourceforge.net>
- ▶ Introduction to Service Data Objects on developerWorks  
<http://www-128.ibm.com/developerworks/java/library/j-sdo/>
- ▶ Open service-oriented architecture: SDO Resources  
<http://www.osoa.org/display/Main/SDO+Resources>
- ▶ JAXB specification  
<http://jcp.org/en/jsr/detail?id=222>
- ▶ Oracle Sun Developer Network JDBC Documentation  
<http://java.sun.com/products/jdbc/overview.html>
- ▶ Oracle Sun Developer Network Java SE Technologies: Database  
<http://java.sun.com/javase/technologies/database/>
- ▶ Oracle Sun Developer Network JDBC Data Access API  
<http://developers.sun.com/product/jdbc/drivers>
- ▶ “SQLJ: The “open sesame” of Java database applications” by Shirley Ann Stern, InfoWorld Java World: JavaWorld.com, May 1, 1999  
<http://www.javaworld.com/javaworld/jw-05-1999/jw-05-sqlj.html>
- ▶ *Java Specification Request (JSR) 317: Java Persistence API, Version 2.0*  
<http://jcp.org/en/jsr/summary?id=317>

- ▶ Refer to the WebSphere Application Server V8 Beta Information Center for detailed descriptions about JPA application development:
  - [http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.jpa.doc/topics/c\\_jpa.html](http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.jpa.doc/topics/c_jpa.html)
  - <http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/com.ibm.servertools.doc/topics/tjpaautv7.html>
  - <http://publib.boulder.ibm.com/infocenter/radhelp/v8/index.jsp?topic=/com.ibm.ertools.webtoolscore.doc/topics/tjpaconfigmrbearother.html>
- ▶ Generating a J2C bean using the J2C Tools in Rational Application Developer V7.0  
[http://www.ibm.com/developerworks/rational/library/06/1212\\_nigul/](http://www.ibm.com/developerworks/rational/library/06/1212_nigul/)
- ▶ Create a J2C application for an Information Management System (IMS) phone book transaction using IMS Resource Adapter  
<http://www.ibm.com/developerworks/rational/library/08/dw-r-j2cimsresource/>
- ▶ Working with J2C Ant Scripts in Rational Application Developer V7  
[http://www.ibm.com/developerworks/rational/library/06/1205\\_ho-benedek/](http://www.ibm.com/developerworks/rational/library/06/1205_ho-benedek/)
- ▶ *JSR 318: Enterprise JavaBeans 3.1*  
<http://jcp.org/en/jsr/summary?id=318>
- ▶ WebSphere Application Server Information Center  
[http://publib.boulder.ibm.com/infocenter/wasinfo/beta/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tejb\\_timerserviceejb\\_enh.html](http://publib.boulder.ibm.com/infocenter/wasinfo/beta/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tejb_timerserviceejb_enh.html)
- ▶ Roy Fielding in his dissertation “Architectural Styles and the Design of Network-based Software Architectures”  
<http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

## How to get IBM Redbooks publications

You can search for, view, or download IBM Redbooks publications, IBM Redpaper publications, web docs, draft publications, and Additional materials, as well as order hardcopy Redbooks publications, at this website:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Help from IBM

IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

IBM Global Services

[ibm.com/services](http://ibm.com/services)







**Redbooks**

# **Rational Application Developer for WebSphere Software V8 Programming Guide**

(2.5" spine)  
2.5" <-> mm, n"  
1315 <-> mmn pages







# Rational Application Developer for WebSphere Software V8 Programming Guide



**Develop applications  
using Java EE 6 and  
beyond**

**Test, debug, and  
profile with local and  
remote servers**

**Deploy applications  
to WebSphere  
servers**

IBM Rational Application Developer for WebSphere Software V8 is the full-function Eclipse 3.6 technology-based development platform for developing Java Standard Edition Version 6 (Java SE 6) and Java Enterprise Edition Version 6 (Java EE 6) applications. In addition, Rational Application Developer provides development tools for technologies, such as OSGi, Service Component Architecture (SCA), Web 2.0, and XML. Rational Application Developer focuses on applications to be deployed to IBM WebSphere Application Server and IBM WebSphere Portal.

Rational Application Developer provides integrated development tools for all development roles, including web developers, Java developers, business analysts, architects, and enterprise programmers.

This IBM Redbooks publication is a programming guide that highlights the features and tooling included with Rational Application Developer V8.0.1. Many of the chapters provide working examples that demonstrate how to use the tooling to develop applications, as well as achieve the benefits of visual and rapid application development. This publication is an update of *Rational Application Developer V7.5 Programming Guide*, SG24-7672.

## **INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION**

### **BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
[ibm.com/redbooks](http://ibm.com/redbooks)