caCORE SDK 4.0 Migration Guide

Center for Bioinformatics

This is a U.S. Government Work

October 26, 2007

Credits and Resources

caCORE SDK Contributors			
SDK Development Team	**Other Development Teams**	**Guide**	**Program Management**
Satish Patel [1]	Kunal Modi [1]	Satish Patel [1]	Denise Warzel [4]
Dan Dumitru [1]	Vijay Parmar [1]	Dan Dumitru [1]	Avinash Shanbhag [4]
Aynur Abdurazik [2]	Shaziya Muhsin [2]	Charles Griffin [1]	George Komatsoulis [4]
	Konrad Rokicki [2]	Wendy Erickson-Hirons [5]	Charles Griffin [1]
	Ye Wu [2]		
	Christophe Ludet [3]		
[1] Ekagra Software Technologies	[2] Science Applications International Corporation (SAIC)	[3] Oracle Corporation	[4] National Cancer Institute Center for Bioinformatics
[5] Northern Taiga Ventures, Inc.			

SDK Resources	
Name	***URL***
Mailing List	CACORESDK_USERS-L@mail.nih.gov
Mailing List Archive	https://list.nih.gov/archives/ cacore_sdk_users-l.html
Project Home (GForge)	https://gforge.nci.nih.gov/projects/cacoresdk/
SDK Support Tracker (GForge)	https://gforge.nci.nih.gov/tracker/?group_id=148&atid=731

Contacts and Support	
NCICB Application Support	http://ncicb.nci.nih.gov/NCICB/support Telephone: 301-451-4384 Toll free: 888-478-4423

Submitting a Support Issue

A GForge Support tracker group, which is actively monitored by caCORE SDK developers, has been created to track any support requests. If you believe there is a bug/issue in the caCORE SDK software itself, or have a technical issue that cannot be resolved by contacting the NCICB Application Support group, please submit a new support tracker using the following link: https://gforge.nci.nih.gov/tracker/?group_id=148&atid=731. Prior to submitting a new tracker, review any existing support request trackers in order to help avoid duplicate submissions.

Release Schedule

This guide has been updated for the caCORE SDK 4.0 release. It may be updated between releases if errors or omissions are found. The current document refers to the 4.0 version of caCORE SDK, released in October 2007 by the NCICB.

Table of Contents

Introduction

The SDK 4.0 Migration Guide contains instructions on migrating SDK generated applications and SDK clients from SDK 3.2.x to 4.0. Users of the SDK who intend to use the SDK 4.0 features will have to regenerate the system with SDK 4.0 options. Once the system is generated, users can follow the steps required for migration of a specific client.

The following is an overview of the main SDK areas requiring change:

- Enterprise Architect (EA) XMI Export Options:
  - The EA model needs to be exported to XMI using the *Full Roundtrip* option.
- User Interface:
  - A new *Home* page has been added that contains a *Login* form when security is enabled.
- GetXML Interface:
  - The authentication mechanism has changed when security is enabled.
- Java API:
  - An *ApplicationService* instance can now only be obtained via one of the *ApplicationServiceProvider* methods.
  - The *ApplicationServiceProvider* class has been moved to a new package. Client users of this class must change any import statement(s) accordingly.
  - The HQLCriteria wrapper query call has changed.
  - Clients can no longer specify the number of records to fetch from the server.
  - Impl class generation for the domain objects has been eliminated. Users must use the domain object for creating a new instance.
  - JAR files in the client distribution have changed; users should obtain and use the latest JAR files.
  - The Java API no longer provides a way of specifying case sensitivity during searches.
  - The remoteService.xml file is no longer available. Use the application-config-client.xml file instead to specify the serviceURL and serviceInterface property values.
  - Users of the old API used to have to specify the serviceURL as http://<<Server-name:port>>/<<project-name>>/http/remoteService. Now users only have to specify http://<<Server-name:port>>/<<project-name>>
- XML Utility:
  - The main XML Utility class now requires that a Marshaller and Unmarshaller instance be passed to the XML Utility constructor.
  - The XML Utility and related classes have been moved to a new package. Client users of these classes must change any import statement(s) accordingly.
- Web Services API:
  - The extraneous *.ws* package layer has been eliminated from the generated domain Java Bean packages, the WSDD, and the corresponding WSDL.
  - The Web Services API no longer provides a way to specify the number of records to retrieve.

- Manual deployment of the Web Services API is no longer needed. The WSDD is now bundled with the application .war file, and is automatically deployed whenever the .war file is deployed.

Each of these areas is discussed in detail in the following sections.

Note: This migration guide is not intended to discuss or list the new SDK 4.0 features. For information related to the new SDK 4.0 features, see the section "New Features for caCORE SDK 4.0" in Chapter 1 of the *caCORE SDK 4.0 Developer's Guide.*

Changes Required When Migrating to SDK 4.0

Exporting the UML Model to XMI (EA Only)

Before the SDK can process a UML model created within Enterprise Architect (EA), it needs to be exported to XMI and then copied to the *\models* directory within the SDK root folder. This was true of prior versions of the SDK, and continues to be true with SDK 4.0. However, the options for exporting the model to XMI have changed for SDK 4.0.

Note: SDK 4.0 now supports models created using the ArgoUML tool. However, this section only applies to Enterprise Architect because the ArgoUML project file is stored in an XML format that the SDK Code Generator can understand and process directly; that is, the ArgoUML project file does *not* need to be exported to XMI prior to processing it via the SDK Code Generator.

Note: This section is also included in the main SDK 4.0 Guide and is provided here again as a convenience to the reader.

To export an EA model to XMI that is understood by the SDK 4.0 Code Generator, use the following steps:

1. In the EA **Project Browser**, select the **Logical View** package (Figure 1).

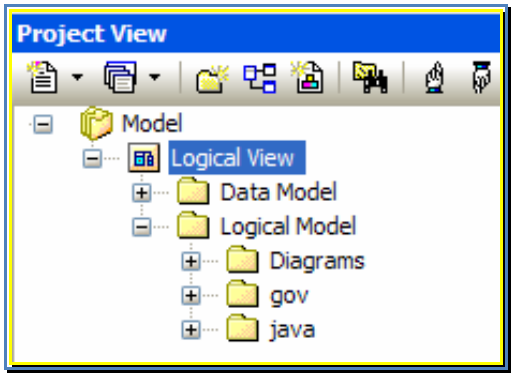

Figure 1 Model Logical View Package

2. Right click and select **Import/Export** or select **Project > Import/Export**. Select **Export Package to XMI**.

3. The **Export Package to XMI** dialog box opens (Figure 2).

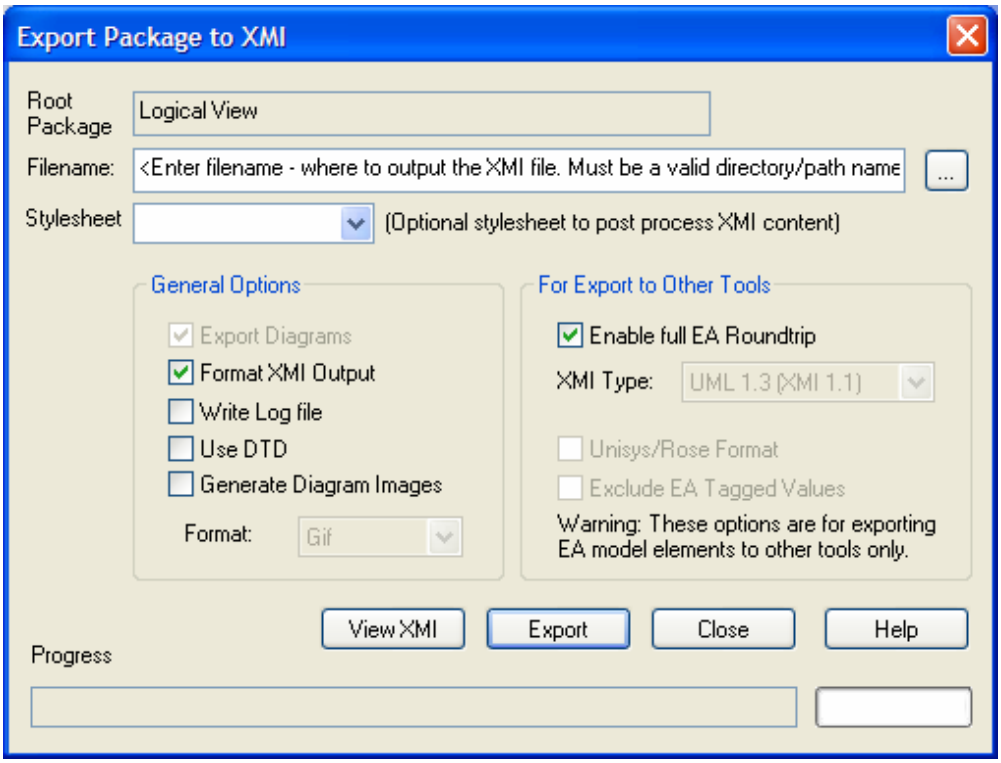

Figure 2 SDK 4.0 Enterprise Architect XMI Export Options

4. Set the Export options as follows:

   Note: The XMI export options that should be selected have changed for SDK 4.0. The new required options are:
   - Export Diagrams
   - Enable full EA Roundtrip

Export Option	***Description***
Filename	Used to indicate where to output the XMI file. *Enter a valid directory/path name. Also, make sure the file type suffix is .xmi.*
Stylesheet	Used to post-process XMI content before saving to file. *Leave unselected.*
Export Diagrams	*Leave checked.*
Use Unisys Rose Format	Used to indicate whether the model should be exported in Rose UML 1.3 or XMI 1.1 format. *Leave unchecked.*
Format XML output	Used to indicate whether or not to format output into readable XML (takes a few additional seconds at end of run). *Leave checked.*

Export Option	Description
Write log file	Used to indicate whether a log of export activity should be created (recommended). The log file is saved in the same directory to which the file is exported. (Optional.) *Leave checked if desired.*
Use DTD	Used to indicate whether to use the UML1.3 DTD. Using this option validates the correctness of the model and ensures that no syntactical errors have occurred. *Leave unchecked.*
Exclude EA Tagged Values	Used to indicate whether EA specific information should be excluded from the export to other tools. *The SDK now supports Full EA roundtrip.* *Leave unchecked.*

5. Click **Export**.

6. Once the XMI file has been exported, copy it to the *\models* directory within the SDK root folder.

Note: The XMI file name and the value of the *MODEL_FILE* property within the *deploy.properties* file must match. Otherwise, a *File Not Found* error will be reported when trying to process the XMI file through the SDK Code Generator.

Enabling Security

Many of the changes required when migrating to SDK 4.0 are a result of security enhancements to the SDK. This includes the addition of a configuration property to the *deploy.properties* file. To enable security within SDK 4.0, make sure the *SECURITY_ENABLED* property within the *deploy.properties* file is set to *true* (as highlighted in Figure 3) prior to building the SDK application. To generate an application using SDK 4.0, use the *build-system* target of the primary *build.xml* Ant script distributed with the SDK.

```
30 PROJECT_NAME=example
31 NAMESPACE_PREFIX=gme://caCORE.caCORE/3.2/
32
33 SECURITY_ENABLED=true
34 CSM_PROJECT_NAME=sdk
35 INSTANCE_LEVEL_SECURITY=false
36 ATTRIBUTE_LEVEL_SECURITY=false
37
38 WEBSERVICE_NAME=${PROJECT_NAME}Service
```

Figure 3 Enabling Security within SDK 4.0

User Interface Changes

The SDK generated Graphical User Interface (GUI) has received a cosmetic facelift for SDK 4.0, and the architecture is now based on the Struts 2 framework (http://struts.apache.org/2.x/). Along with these changes, a new *Home* page has been added to the original set of pages, which previously included the Package/Class browser, Search Criteria, and Results pages. As part of the security features implemented for SDK 4.0, a *Login* form containing *Login Id* and *Password* fields has been added to the new *Home* page (Figure 4).

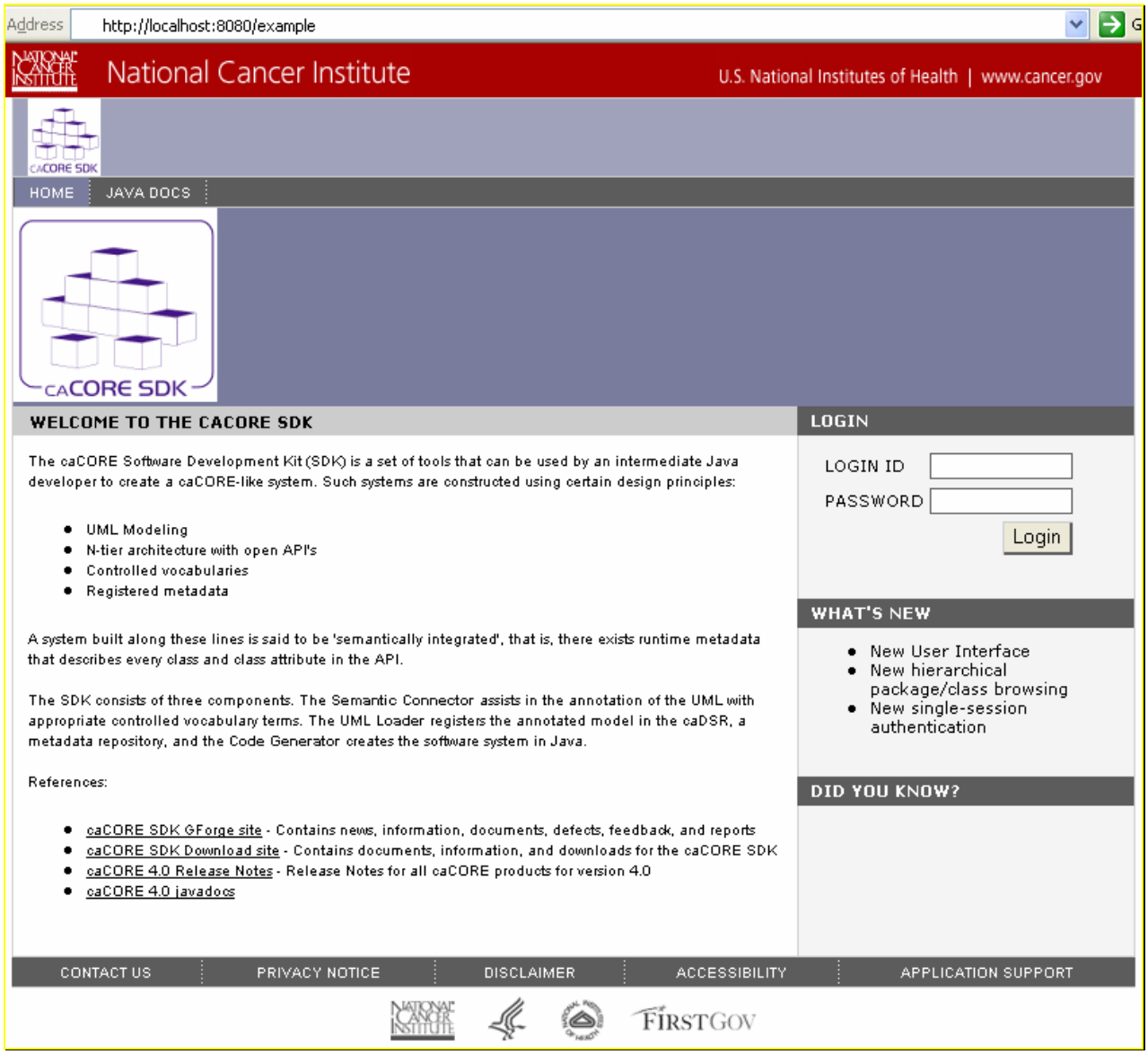

Figure 4 SDK 4.0 Home Page now has a Login form when security is enabled

For security-enabled SDK applications, a Common Security Module (CSM) database must be created, and user authentication and authorization information added to this database for

use with the SDK generated application.

The *Home* page can be accessed via the following URL pattern:

SDK GUI URL Pattern	http://<server_name>:<server_port>/<project_name>

A sample URL from the sample SDK model is:

Sample SDK GUI URL	http://localhost:8080/example

GetXML Interface Changes

The GetXML REST Interface query syntax remains unchanged in SDK 4.0. However, there are two notable exceptions:

- The authentication login process has changed when security is enabled.
- An optional request parameter, *rolename*, is now supported.

The GetXML authentication process previously required that a *username* and *password* must be supplied as part of the query string (URL). This method posed more of a security threat, as the credentials could easily be read if the transmission were intercepted. Now, the GetXML function is secured by BASIC authentication (http://en.wikipedia.org/wiki/Basic_access_authentication). This implies that, though the *username* and *password* are still passed as plaintext, they are now at least encoded as a sequence of base-64 characters before transmission to prevent them from being read directly by a person.

The *username* and *password* must now be supplied via an *Authorization* HTTP header, as illustrated in the highlighted portion of the following sample test code:

```
String searchUrl = serverUrl+"/GetXML?query="+klass.getName()+"&"+klass.getName();
URL url = new URL(searchUrl);
URLConnection conn = url.openConnection();

//Uncomment the following two lines for secured system
//and provide proper username and password
//
//String base64 = "userId" + ":" + "password";
//conn.setRequestProperty("Authorization", "Basic " +
//                        new String(Base64.encodeBase64(base64.getBytes())));

File myFile = new File("./output/" + klass.getName() + "_test-getxml.xml");

FileWriter myWriter = new FileWriter(myFile);
DataInputStream dis = new DataInputStream(conn.getInputStream());
```

Figure 5 SDK 4.0 GetXML sample code illustrating BASIC authentication

The second notable change with the GetXML Interface is that it now supports an optional request parameter, *rolename*, as part of the URL query string. The syntax of the URL query string is shown below, where variable items are enclosed in “<>”:

REST Interface URL Pattern	http://<server_name>:<server_port>/<project_name>/ GetXML?query=<target>&<criteria>[&rolename=<rolename>]

The following table describes each of the variable properties:

ApplicationService Parameter	***Description***
server_name	A *string* identifying the server, or host, name. Examples include *localhost* and *cbiodb30.nci.nih.gov*.
server_port	A *string* identifying *port number* the SDK server is listening on. Examples include *80* or *8080*.
project_name	A *string* identifying the *project name* used when building and deploying the SDK application. Examples include *example* and *cabio*. *Note: this value is same as the PROJECT_NAME property found within the deploy.properties file.*
target	A *string* identifying the qualified or non-qualified query *target/result* class name. Examples include: • *Bank* • *gov.nih.nci.cacoresdk.domain.inheritance.childwithassociation .Bank*
criteria	A *string* identifying the qualified or non-qualified criteria class name to be used as a filter/constraint on the result set. An example would be the SDK sample model *Credit* class that has an association to the *Bank* class via its *issuingBank* attribute. If desired, the value of the *id* attribute of the criteria class instance can also be supplied in order to further constrain the result set. The pattern for such a criteria string is *<criteria_class_name>[@id=<id_value>]*. An example might be *Credit[@id=3]*, which indicates that only target/result class instances are returned that are associated to the *Credit* record with an *id* value of *3*.
rolename	The name of the attribute within the criteria class that identifies the association to be traversed when retrieving the target/result class(es). An example would be the *issuingBank* attribute of the *Credit* class found within the sample SDK model. The *rolename* property must be specified whenever the *Criteria* class has two or more attributes representing associations to the same target/result class type. One example would be the *Child* class within the sample SDK model that contains two attributes, *mother* and *father*, that both represent instances of the *Parent* class. In this

	scenario, specifying a value of *rolename=mother* or *rolename=father* within the REST URL would ensure that the correct *Parent* instance would be returned.

A sample URL from the sample SDK model is provided below:

Sample GetXML URL	http://localhost:8080/example/GetHTML?query=Bank&Credit[@id=3]&roleName=issuingBank

Java API Changes

Security Changes

In earlier versions of the SDK, making secure calls via the Java API essentially required the following process:

- Create a *ClientSession* instance containing *username* and *password* credentials
- Start the *ClientSession* instance
- Obtain an *ApplicationService* instance directly (via one of the *ApplicationService.getxxx()* methods) or indirectly via the *ApplicationServiceProvider* class
- Invoke one of the wrapper *ApplicationService* search methods

An example of this process is shown in the sample code below:

```
public static void main(String[] args) {

    System.out.println("*** Test Secured Client...");

    try{

        ClientSession session = ClientSession.getInstance();
        session.startSession("userId", "password");
        ApplicationService appService = ApplicationServiceProvider.getApplicationService();

        try {

            System.out.println("Retrieving all genes based on symbol=IL*");
            Gene gene = new GeneImpl();
            gene.setSymbol("IL*");

            try {
                List resultList = appService.search(Gene.class, gene);
                for (Iterator resultsIterator = resultList.iterator(); resultsIterator.hasNext();) {
```

Figure 6 Making Secured ApplicationService query calls with previous SDK versions

With SDK 4.0, the *ClientSession* class has been eliminated, and the *ApplicationServiceProvider* class is used to both supply the username and password credentials and to obtain an instance of the *ApplicationService*. An example of the new process is shown in Figure 7.

```
public void testSearch() throws Exception
{
    //Application Service retrieval for secured system
    //ApplicationService appService = ApplicationServiceProvider.getApplicationService("userId","password");

    ApplicationService appService = ApplicationServiceProvider.getApplicationService();
    Collection<Class> classList = getClasses();
    for(Class klass:classList)
    {
        Object o = klass.newInstance();
        System.out.println("Searching for "+klass.getName());
        try
        {
            Collection results = appService.search(klass, o);
```

Figure 7 SDK 4.0 Making ApplicationService query calls in a secured system

Obtaining an ApplicationService Instance

Unlike earlier versions of the SDK, the *ApplicationService* instance can no longer be obtained directly via one of the *ApplicationService.getxxx()* methods. Instead, it must now be obtained by using one of the *ApplicationServiceProvider* methods shown in Figure 8.

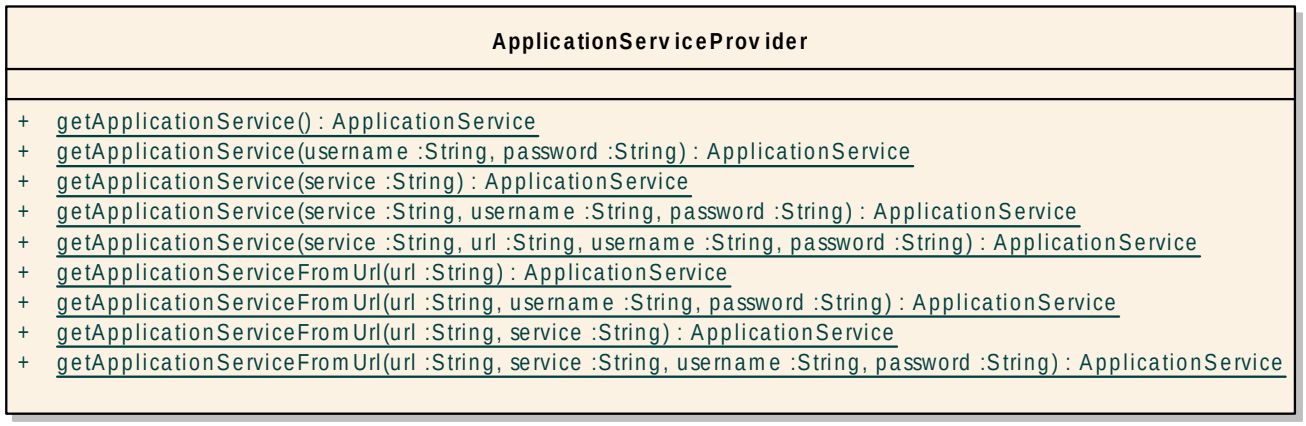

Figure 8 SDK 4.0 Obtaining an ApplicationService instance

HQLCriteria Wrapper Query Calls

As part of the SDK 4.0 refactoring process, some of the wrapper query classes have been moved to a different package. The HQLCriteria class has moved from *gov.nih.nci.common.util package* to *gov.nih.nci.system.query.hibernate*. SDK 4.0 users should change the import statements in their client applications.

Impl Class Generation for Domain Objects

The Impl class generation process for the domain objects has been eliminated. Users of SDK 3.x had an option to create a new instance of a domain object using either class for the

domain object or class for the Impl of the domain object. For example, if the domain contained an object called Bank, then users could create an instance of Bank as:

Option 1: Bank bank = new Bank(); or

Option 2: Bank bank = new BankImpl();

Users of the SDK 4.0 version can no longer use the second option. If users have been using option 2, code should be changed to use option 1.

JAR files in the Client Distribution

As part of the technology stack upgrade, many of the JARs have been changed. In addition, the client.jar that SDK previously produced has been divided into multiple JAR files. Users of the SDK 4.0 version should pick up all the JAR files being produced under the respective client's distribution directory.

XML Utility Changes

While used primarily by caGRID, the caCORE SDK does provide a class, *XMLUtility*, that can be used to marshal (serialize) the generated domain Java Beans to XML, and unmarshal (deserialize) XML data back to the generated domain Java Beans. In prior versions of the SDK, the *XMLUtility* class could be instantiated using the default, no-argument constructor, as highlighted in the sample code in Figure 9.

```
XMLUtility myUtil = new XMLUtility();
List resultList = appService.search(Gene.class, gene);
System.out.println("Result list size: " + resultList.size() + "\n");
long startTime = System.currentTimeMillis();
for (Iterator resultsIterator = resultList.iterator(); resultsIterator.hasNext();) {
    Gene returnedGene = (Gene)resultsIterator.next();
    File myFile = new File("C:/temp/test.xml");
    FileWriter myWriter = new FileWriter(myFile);
```

Figure 9 XMLUtility Usage Example from Previous SDK Versions

In SDK 4.0, this is no longer the case. The main XMLUtility class now requires that a Marshaller and Unmarshaller instance be created and passed to the XMLUtility constructor, as shown in the highlighted sample code in Figure 10.

```
Marshaller marshaller = new caCOREMarshaller("xml-mapping.xml", false);
Unmarshaller unmarshaller = new caCOREUnmarshaller("unmarshaller-xml-mapping.xml", false);
XMLUtility myUtil = new XMLUtility(marshaller, unmarshaller);
for(Class klass:classList)
{
    Object o = klass.newInstance();
    System.out.println("Searching for "+klass.getName());
    try
    {
        Collection results = appService.search(klass, o);
        for(Object obj : results)
        {
```

Figure 10 SDK 4.0 XMLUtility Usage Example

Note that the default Marshaller and Unmarshaller classes (caCOREMarshaller and caCOREUnmarshaller, respectively) are provided with the SDK. A class diagram of the SDK 4.0 *XMLUtility* class is provided for convenience (Figure 11).

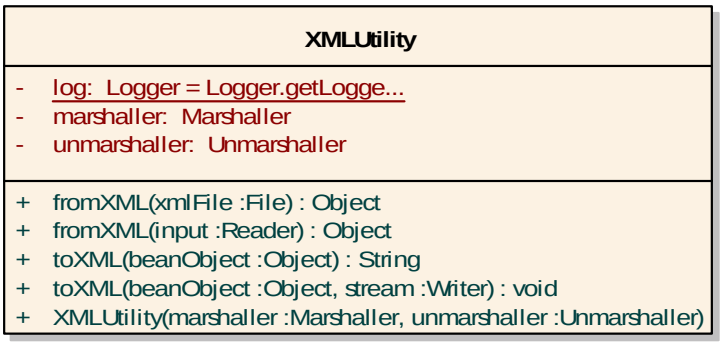

Figure 11 XML Utility Class Diagram

As implied by the XMLUtility Constructor method, the *XMLUtility* class wraps both a SDK *Marshaller* and *Unmarshaller* class, which it requires in order to perform its work. See the caCORE SDK 4.0 Developer's Guide for more information.

In addition, the *XMLUtility* class and related classes have been moved to a new package: *gov.nih.nci.system.client.util.xml*. Users of these classes must change any import statement(s) accordingly.

Web Service Changes

While the SDK 4.0 Web Service continues to be based on the Axis 1.4 framework, the extraneous *.ws* translation layer found in previous SDK versions has been eliminated from both the generated Java Bean packages and the Axis Web Service Deployment Descriptor (WSDD). The WSDD, in turn, is used to generate the SDK Web Service Description Language (WSDL). This implies that the .ws packages have also been removed from the SDK WSDL. Consequently, those SDK users that used earlier versions of the SDK WSDL to generate client-side artifacts (including stubs) must regenerate these artifacts using the new SDK 4.0 WSDL.

In addition, the SDK WSDD is now packaged along with the rest of the SDK generated .war file, thus allowing for automatic deployment of the Web Service; that is, deploying the Web Service separately from the client is no longer necessary.

Finally, if security is enabled, a SOAP Security header containing credential (*username* and *password*) information must be inserted into the call prior to invocation, as shown in the highlighted portion of the sample code in Figure 12.

```
call = (Call) service.createCall();

for(Class klassToMap:classList)
{
    QName searchClassQNameToMap = new QName("urn:"+getInversePackageName(klassToMap), klassToMap.getSimpleName());
    call.registerTypeMapping(klassToMap, searchClassQNameToMap,
            new org.apache.axis.encoding.ser.BeanSerializerFactory(klassToMap, searchClassQNameToMap),
            new org.apache.axis.encoding.ser.BeanDeserializerFactory(klassToMap, searchClassQNameToMap));
}

QName searchClassQName = new QName("urn:"+getInversePackageName(klass), klass.getSimpleName());

call.setTargetEndpointAddress(new java.net.URL(url));
call.setOperationName(new QName("exampleService", "queryObject"));
call.addParameter("arg1", org.apache.axis.encoding.XMLType.XSD_STRING, ParameterMode.IN);
call.addParameter("arg2", searchClassQName, ParameterMode.IN);
call.setReturnType(org.apache.axis.encoding.XMLType.SOAP_ARRAY);

/*
//This block inserts the security headers in the service call
SOAPHeaderElement headerElement = new SOAPHeaderElement(call.getOperationName().getNamespaceURI(),"SecurityHeader");
headerElement.setPrefix("security");
headerElement.setMustUnderstand(false);
SOAPElement usernameElement = headerElement.addChildElement("username");
usernameElement.addTextNode("userId");
SOAPElement passwordElement = headerElement.addChildElement("password");
passwordElement.addTextNode("password");
call.addHeader(headerElement);
*/

Object o = klass.newInstance();

System.out.println("Searching for "+klass.getName());
Object[] results = (Object[])call.invoke(new Object[] { klass.getName(), o });
```

Figure 12 SDK 4.0 Web Service SOAP Security Header

In prior versions of the SDK, users could specify the number of records to retrieve by providing a value for the recordCounter parameter of the following query method:

```
public List query(String targetClassName, Object criteria, int
startIndex, int recordCounter);
```

In SDK 4.0, however, the recordCounter parameter has been removed, and the Web Services API no longer provides a way to specify the number of records to retrieve. The query method now has the following signature:

```
public List query(String targetClassName, Object criteria, int
startIndex);
```

Users should now only provide the start index. The end index is derived as the start index + the maximum number of records that can be retrieved. Users can determine the maximum number of records that can be retrieved by invoking the following Web Service method:

```
public int getMaximumRecordsPerQuery();
```