

# caCORE 2.0

## Technical Guide



NATIONAL<sup>®</sup>  
CANCER  
INSTITUTE

Center for  
Bioinformatics



U.S. Department of  
Health and Human Services

## TABLE OF CONTENTS

1.0	Overview of caCORE .....	1
1.1	Introduction: The NCICB Core Infrastructure.....	2
1.2	The caCORE Standards .....	2
1.3	Organization of this Guide.....	5
2.0	The Unified Modeling Language.....	6
2.1	Use Case diagrams.....	8
2.2	The Class diagram.....	9
2.3	The Rose Web Publisher Pages .....	13
2.4	Package diagrams.....	15
2.5	Collaboration and Sequence diagrams.....	16
3.0	The caBIO Domain Objects.....	19
3.1	The Object Hierarchies .....	20
3.2	The caBIO Domain Object Catalog.....	23
4.0	The EVS Domain Objects.....	31
4.1	The UMLS Metathesaurus.....	33
4.2	Knowledge Representations and Description Logic.....	34
4.3	Description Logic in the NCI Thesaurus .....	38
4.4	Concept Edit History in the NCI Thesaurus .....	39
4.5	The caBIO EVS API.....	40
4.6	The EVS Search Paradigm.....	42
4.7	The EVS Domain Object Catalog.....	46
4.8	Downloading the NCI Thesaurus .....	48
4.9	Mapping of the Gene Ontology to Ontylog .....	52
5.0	The caDSR Domain Objects.....	54
5.1	Modeling Metadata: The ISO/IEC 11179 Standard .....	55
5.2	The caDSR Metamodel.....	58
5.3	The caDSR API.....	63
5.4	Downloading the caDSR.....	66
5.5	The caDSR Domain Object Catalog.....	66
6.0	The caMOD Domain Objects .....	73
6.1	The Mouse Models of Human Cancers Consortium.....	74
6.2	The caMOD API.....	75
6.3	The caMOD Domain Object Catalog .....	78
7.0	The caCORE MAGE-OM API.....	86
7.1	The GEDP Project.....	87
7.2	The caCORE MAGE-OM API.....	88
7.3	Installing a MAGE-OM Java Client .....	90

7.4	The caBIO Bridge to MAGE .....	91
8.0	Search Criteria Objects and the caCORE APIs .....	92
8.1	The Java API Search/Retrieve Paradigm.....	94
8.2	How the caBIO Search Paradigm Operates .....	97
8.3	The SOAP API.....	98
8.4	The HTTP Interface .....	99
8.5	Summary of Search Controls in the Different APIs .....	99
8.6	The caBIO SearchCriteria Catalog .....	100
8.7	The EVS SearchCriteria Catalog .....	109
8.8	The caDSR SearchCriteria Catalog .....	110
8.9	The caMOD SearchCriteria Catalog.....	117
9.0	The caCORE Package Architecture.....	124
9.1	Organization of Packages in caCORE .....	125
9.2	The caBIO DAS Package.....	128
10.0	Advanced Search Methods .....	130
10.1	Basic and Advanced Search Methods.....	131
10.2	Constructing the query tree.....	132
10.3	Building the GridSearchCriteria .....	133
10.4	Executing the Search and Interpreting the Results .....	134
10.5	Building More Complex Queries.....	134
10.6	Roles and Attributes.....	137
11.0	The caBIO Java API .....	140
11.1	Installing a caBIO Client .....	141
11.2	Installing the caBIO Server.....	145
12.0	Code Examples .....	146
12.1	The caBIO GeneDemo program .....	147
12.2	The EVSDemo Program .....	148
12.3	The caDSR Demo Program .....	148
12.4	The caMOD Demo Program.....	149
12.5	The MAGE-OM Demo Program .....	149
13.0	The SOAP API and Web Services.....	151
13.1	The SOAP API and caBIO .....	152
13.2	Using the SOAP API with Perl and SOAP::LITE .....	153
13.3	The caBIO SOAP Services Catalog.....	157
13.4	The EVS SOAP Services Catalog.....	171
13.5	The caDSR SOAP Services Catalog.....	171
13.6	The caMOD Soap Services Catalog .....	181
14.0	The HTTP Interface .....	188
14.1	Overview.....	189
14.2	Using the HTTP Interface.....	190

14.3	Drilling Down Through XLinks .....	194
14.4	Controlling the Number of Items Returned .....	194
14.5	Specifying the IP Address and Port in the URL .....	194
14.6	Applying XSL to XML Output .....	194
14.7	The HTTP Operation Catalog .....	195
14.8	The caBIO HTTP Catalog .....	196
14.9	The EVS HTTP Catalog .....	201
14.10	The caDSR HTTP Catalog .....	202
14.11	The caMOD HTTP Catalog .....	205
15.0	The caCORE Data Sources .....	210
15.1	Data Sources in the caBIO Database .....	211
15.2	References .....	220
Appendix A: The GeneDemo Program .....		222
Appendix B: The EVSDemo Program .....		227
Appendix C: The CaseReportFormDemo Program .....		233
Appendix D: The CancerModelDemo Program .....		237
Appendix E: The MageTest Program .....		241
Appendix F: The caBIO_MageTest Program .....		246
Appendix G: The SearchPkgExample Program .....		250
Appendix H. geneClient.pl .....		253

## LIST OF FIGURES

Figure 2.1-1	CGAP Browser Use Case .....	8
Figure 2.2-1	The caBIO Domain Objects Class diagram .....	9
Figure 2.2-2	(a) Schematic for a UML class. (b) A simple class called <i>Gene</i> .....	10
Figure 2.2-3	Rational Rose access modifier representations .....	11
Figure 2.2-4	A one-to-one association with unidirectional navigability .....	11
Figure 2.2-5	A bidirectional many-to-one relation .....	12
Figure 2.2-6	Aggregation and association .....	12
Figure 2.2-7	Generalization relationship .....	13
Figure 2.3-1	The caBIO object managers .....	14
Figure 2.3-2	Interface as the stereotype name .....	15
Figure 2.4-1	The caCORE packages .....	15

Figure 2.5-1 Collaboration between objects .....	16
Figure 2.5-2 Collaboration diagram for retrieving genes .....	17
Figure 2.5-3 Sequence diagram for retrieving genes .....	17
Figure 3.1-1 The domain object hierarchy.....	20
Figure 3.1-2 The <i>SearchCriteria</i> inheritance hierarchy .....	21
Figure 4.2-1 An earthquake in a semantic network of news stories .....	35
Figure 4.2-2 Propositional versus first-order predicate logic .....	35
Figure 4.3-1. An overview of the NCI Thesaurus infrastructure.....	38
Figure 4.4-1 History records for the split action.....	40
Figure 4.5-1 The caBIO EVS API domain object classes .....	41
Figure 4.6-1 Middleware Interfaces to the EVS Databases.....	43
Figure 5.1-1 Representing data in the ISO/IEC 11179 model.....	56
Figure 5.1-2 Abstract and concrete components of the data representation.....	57
Figure 5.1-3 Many-to-one mappings of information elements in the metadata model .....	57
Figure 5.2-1 Information component infrastructure in the metamodel .....	58
Figure 5.2-2 Administrative and organizational components of the caDSR metamodel .....	61
Figure 5.2-3 Components in the caDSR metamodel for clinical trials data .....	62
Figure 5.3-1 The caDSR domain objects in the caCORE Java API.....	64
Figure 5.3-2 The caDSR API class hierarchy.....	65
Figure 6.2-1 The caMOD class hierarchy.....	76
Figure 6.2-2 The UML Class diagram for the caMOD Domain Objects .....	77
Figure 7.1-1 The caCORE MAGE-OM API to GEDP.....	88
Figure 7.2-1 A high-level view of the MAGE-OM API from a client perspective.....	89
Figure 8-1 The caBIO architecture.....	93
Figure 8.2-1 The logical deployment of the caBIO packages for data retrieval.....	97
Figure 9.1-1 The caCORE 1.0 package structure .....	125
Figure 10.4-1 Assembled Code for Example 2.....	136
Figure 10.4-2 Sample Output from Example 2.....	137
Figure 10.6-1 The association roles between Gene and Taxon .....	139
Figure 10.6-2 Objects with multiple association roles .....	139
Figure 11.1-1 The caBIO Java API .....	141
Figure 11.1-2 Screen shot of GeneDemo output .....	144

Figure 13.1-1 The caBIO architecture and the SOAP interface .....	153
Figure 14.1-1 The caBIO HTTP interface .....	189
Figure 14.7-1 XML excerpt in response to the Gene operation .....	196
Figure 15.1-1 caBIO objects supporting basic research .....	212
Figure 15.1-2 caBIO objects supporting clinical research.....	213

## LIST OF TABLES

Table 4.1-1 NCI local source vocabularies included in the Metathesaurus. ....	34
Table 4.4-1 The NCI Thesaurus concept history table .....	39
Table 4.8-1 Ontylog DTD to OWL conversions .....	51
Table 4.9-1 Ontylog elements used for GO mapping.....	52
Table 4.9-2 Mapping of GO term to Ontylog conceptDef.....	53
Table 5.2-1 Information components in the caDSR metamodel.....	59
Table 5.2-2 Class attributes of an <i>AdministeredComponent</i> .....	60
Table 5.2-3 Components in the caDSR metamodel for clinical trials data.....	63
Table 6.1-1 The web interface information pages for the Cancer Models Database.....	75
Table 8.1-1. Common methods implemented by all <i>SearchCriteria</i> objects .....	94
Table 8.5-1 Attributes controlling the way results are returned .....	100
Table 8.6-1 caBIO <i>putSearchCriteria</i> arguments.....	100
Table 8.8-1 caDSR <i>putSearchCriteria</i> arguments.....	110
Table 8.9-1 caMOD <i>putSearchCriteria</i> arguments .....	117
Table 9.1-1 The caCORE Packages.....	126
Table 9.1-2 caCORE Packages Summaries .....	126
Table 10.1-1 Central objects used in the advanced search methods.....	131
Table 13.2-1 Frequently used caBIO SOAP services.....	154
Table 13.3-1 Mapping web services to methods .....	168
Table 14.2-1 Summary of the HTTP syntax.....	193

## **1.0 OVERVIEW OF caCORE**

## 1.1 Introduction: The NCICB Core Infrastructure

The last decade has produced a wealth of genomic information that has just begun to be examined. With this accumulation of bioinformatic data has come a paradigm shift to translational research, and a directive to more quickly advance discoveries in basic research to multifaceted clinical settings and trials. This calls not only for advanced analytic tools and customized data warehouses, but, in addition, for computational environments and software tools that support the development of complex data-mining and information management tasks.

The National Cancer Institute's Center for Bioinformatics (NCICB) has as its mission the goal of bridging these diverse initiatives via a core infrastructure called caCORE. The collection of NCICB web sites described in this technical guide and in the accompanying user manual provide web-based analysis tools and integrated data repositories, as well as a rich development environment for implementing bioinformatics applications.

As described in the NCICB Applications User Manual, clinical and basic research scientists can find web-based tools for the analysis of genomic and clinical data as well as for the development of clinical trials protocols. For the clinical researcher, the Cancer Data Standards Repository (caDSR) provides metadata support for developing clinical trials protocols, and the controlled vocabularies available from the Enterprise Vocabulary Services (EVS) provide a semantic integration of the many diverse medical terminologies in use today.

For the cancer research scientist, these NCICB interfaces provide access to

- the Cancer Genome Anatomy Project (CGAP),
- the Cancer Models Database (caMOD),
- the Cancer Molecular Analysis Project (CMAP),
- the Cancer Images database (caIMAGE), and
- the Gene Expression Data Portal (GEDP).

Behind this array of web tools, data repositories, and biomedical informatics services is the “caCORE stack”—a set of core technologies providing the necessary middleware and knowledge infrastructure to serve the cancer research community. This document provides a technical guide to the caCORE, for users intending to make use of the application programming interfaces (APIs), and, more generally, for anyone who wants to look “under the hood,” to better understand the philosophies and vision at NCICB.

The accompanying NCICB Applications User Manual describes the various types of cancer research information and services available via the web, and includes step-by-step instructions on how to access these resources, along with simple examples.

## 1.2 The caCORE Standards

In addition to providing software and data repositories, the caCORE serves a critical role in defining standards—in biomedical nomenclature, data modeling, and shared data elements, as well as in the processes whereby these models and elements are developed. A guiding principle throughout all of the NCICB projects is the need to establish and/or adhere to agreed-upon standards of data representation, exchange, and manipulation.



The caCORE infrastructure is composed of three primary components: the Enterprise Vocabulary Services, the Cancer Data Standards Repository, and the Cancer Bioinformatics Infrastructure Objects (caBIO). The “standards stack” integrates

- controlled vocabularies (dictionaries, ontologies, and thesauri),
- common data elements (metadata), and
- object models of entities within and across each domain.

The EVS, managed cooperatively with the NCI Office of Communications, provides a set of standardized, controlled vocabularies for the life sciences, along with tools and guidelines for the development and curation of such vocabularies. The vocabularies and ontologies managed by the EVS span multiple disciplines and domains, including human and mouse pathology, epidemiology, molecular biology, genetics, clinical trials, patient care, and various other biomedical and bioinformatic application areas. The EVS team is also working closely with the developers of the Semantic Web, and the ontologies available through the EVS can be downloaded in both ASCII and OWL format.

The caDSR addresses a related but somewhat orthogonal aspect of data representation and exchange; specifically, the need to standardize the data representations, report forms, and protocols implemented in clinical trials. Although much data have accrued over the years in ongoing clinical trials, to date, little effort has been made to standardize the methods of record-keeping and reporting. As a result, an enormous amount of valuable information that could be used to advance efforts in related studies has become effectively inaccessible, and the capacity to generalize important results from these legacy data has been precluded.

Based on the ISO11179 standard for metadata, the caDSR manages the NCI Common Data Elements (CDEs) and provides a registry in an Oracle 8i database for agreed-upon clinical terms and their usage. In the previous release (caCORE 1.0), the EVS and caDSR were related but separate efforts. One of the new features of caCORE 2.0 is the interface between these two components: caDSR users can now access the EVS terminologies and definitions, and use these as the basis for curating new data elements.

This interaction between the two projects is further enhanced by the new EVS feature, "Suggest New Term," which allows curators to request new terms as needed. The EVS staff reviews such requests and, working with the caDSR curators, creates new terms to enrich the NCI vocabularies as well. This interface represents a first step towards establishing a well-defined process for curating new data elements, as well as towards achieving a global harmonization of the terms and concepts used in the controlled vocabularies and in clinical applications.

While the EVS and the caDSR address the representational needs and standardization issues involved in controlled vocabularies, report forms, and terminologies, the caBIO project provides a comprehensive set of predefined data structures, programming interfaces, and customized data sources to support the development of advanced software applications seeking to elucidate the molecular basis of cancer.

Many of the data structures and development tools provided by caBIO were initially developed in response to the need to directly access information provided by the Cancer Genome Anatomy Project web site. CGAP is an interactive web site providing access to vast reserves of

genomic information filtered by tissue type, histological status, chromosome location, and biological pathways. Some applications that have used the CGAP resources include

- Analysis of correlations between allelic variants of genes and disease states,
- Identification of single-nucleotide polymorphisms from EST chromatograms,
- Identification of potential tumor markers and antigens,
- *In silico* cloning of novel endothelial-specific genes, and
- Clustering of highly expressed genes in chromosomal domains.

But caBIO is more than a programmatic interface to CGAP; it provides access to many other data sources, as well as to software development tools that are customized for bioinformatic data-mining applications. One such application is NCI's Cancer Molecular Analysis Project, which enables researchers to identify and evaluate molecular targets in cancer. The CMAP web site was developed using the data structures and software tools provided by the caBIO infrastructure.

caBIO provides domain objects (Genes, Chromosomes, Sequences, etc.) that, in conjunction with *search criteria* objects, encapsulate the complexities of cross-platform data exchange and SQL query statements. For the reader familiar with caCORE 1.0, this second major release of caCORE introduces numerous additional domain objects, providing access to the EVS vocabularies, the Cancer Models Database, the GEDP database, and the Cancer Data Standards Repository. Some of the resources and data caBIO supplies access to include

- NCI's CGAP, CMAP, and Genetic Annotation Initiative (GAI) databases;
- Unigene, Homolgene, and LocusLink data from the National Center for Biotechnology Information (NCBI)
- The Distributed Annotation Server (DAS) at UCSC; and
- BioCarta Pathway data.

In keeping with the principle of conformance to emerging standards for data exchange, all of the caBIO data objects are "XML aware," and their design embodies many of the principles advocated by the Life Sciences Research Group at the Object Management Group (OMG). Several transparent programming interfaces are available, to support the developer's language of choice—including Java, Perl, C++, Python, or even HTML. All of the caBIO web services have associated Web Services Description Language files (WSDLs).

caBIO is available as open source, and was developed using best software practices. The development process is an integral part of the end product, and the philosophy at NCI has been to combine the principles of the Rational Unified Process with the agility of eXtreme programming methodology. The Unified Modeling Language (UML) is a critical part of this process, and UML diagrams that convey the internal design elements and implementation of caBIO are used throughout this guide.

In summary, the caCORE infrastructure brings a set of bridging technologies to the frontiers of cancer research. The EVS provides a web interface to vocabulary resources spanning over 70 controlled vocabularies specific to the areas of cancer research, prevention, and treatment. The caDSR provides a platform for registered common data elements to be used in the development of protocols, adverse event reports, and clinical report forms for use in clinical trials. The caBIO software development tools provide domain modeling of both the bioinformatic and the

administrative components of these efforts and supply access to both customized data warehouses and public databases.

### 1.3 Organization of this Guide

[Chapter 2](#) provides a brief review of the UML diagrams used to depict the caCORE Java classes, their relations to one another, and their interactions. [Chapter 3](#) then outlines the hierarchical relations and shared behaviors of the caBIO domain objects, and concludes with a catalog of these objects. Chapters 4 through 7 describe the additional domain objects which have evolved from this original set of caBIO domain objects. Each chapter is devoted to a new application domain, covering the EVS, caDSR, caMOD, and MAGE-OM APIs respectively. The first three of these are integrated with the caBIO objects at the package level, and share a common design framework. The [MAGE-OM](#) API is based on the [Object Management Group](#) (OMG) specifications for the Microarray Gene Expression (MAGE) object model, and, therefore has a different design structure.

[Chapter 8](#) defines the caBIO search criteria object paradigm. A general discussion of how these objects interact with the domain objects to retrieve data for the user is provided, along with specifications of the syntax and parameters to be used in their deployment. All of the interfaces, including the Simple Object Access Protocol (SOAP) API and the HTTP interface, operate on these domain objects and their associated search criteria objects. Thus, the discussion in Chapter 8 provides a basis for understanding the software mechanisms whereby all of the caCORE search capabilities are defined. A search criteria object catalog then summarizes this information for quick reference.

[Chapter 9](#) outlines the package structure used in the design of the caCORE APIs, and [Chapter 10](#) discusses the classes in one type of package in particular, the *search* packages. The search classes and methods defined in those packages are used in the implementation of the BIOgopher interface (described in the NCICB Applications User Manual); a discussion of it is included here for advanced programmers who may wish to deploy this capability in similar applications.

[Chapters 11](#) provides instructions for installing and using the caCORE Java API. The installation instructions document the client installation for users who intend to access data via the NCI servers. Detailed instructions for the complete installation of the caBIO server and database are provided in the Readme file that is included in the distribution package. [Chapter 12](#) discusses the sample demonstration programs that are included in the appendices.

[Chapter 13](#) introduces the caBIO Web Services API and their deployment via SOAP using tools such as SOAP: Lite for Perl. [Chapter 14](#) documents the HTTP interface and provides specifications of the methods and parameters to be used in its deployment. Finally, [Chapter 15](#) summarizes the data sources made available through these interfaces.

## **2.0 THE UNIFIED MODELING LANGUAGE**

The caCORE APIs were initially designed to provide programmatic access to the caBIO databases only. However, the confluence of independent projects at NCI has redefined and extended the requirements of these APIs, and they now provide an integration platform and infrastructure for several efforts, including

- The Enterprise Vocabulary Services,
- The Cancer Data Standards Repository,
- The Cancer Models Database,
- The Gene Expression Data Portal,
- The Cancer Images Database,
- The Cancer Genome Anatomy Project, and
- The Cancer Molecular Analysis Project.

These efforts are at various stages of integration with the larger caCORE design, and the intention is that the caBIO domain objects, which form the core of the APIs, will ultimately provide a variety of application programming interfaces for all of these resources. A good deal of documentation on these objects is available in the Unified Modeling Language at various NCICB web sites; the purpose of this chapter is to familiarize the reader who has not worked with UML with the notation and interpretation of these diagrams.

The UML is an international standard notation for specifying, visualizing, and documenting the artifacts of an object-oriented system. Defined by the [Object Management Group](#), the UML emerged as the result of several complementary systems of software notation and has now become the de facto standard for visual modeling.

In its entirety, the UML is composed of nine different types of diagrams. Perhaps the most intuitive of these are the Use Case and Class diagrams. A Use Case diagram uses simple ball and stick figures with labeled ellipses and arrows to signify how users or other software agents might interact with the system. A Class diagram offers a compact representation of the classes, their static relations to one another, and, optionally, the class attributes and operations.

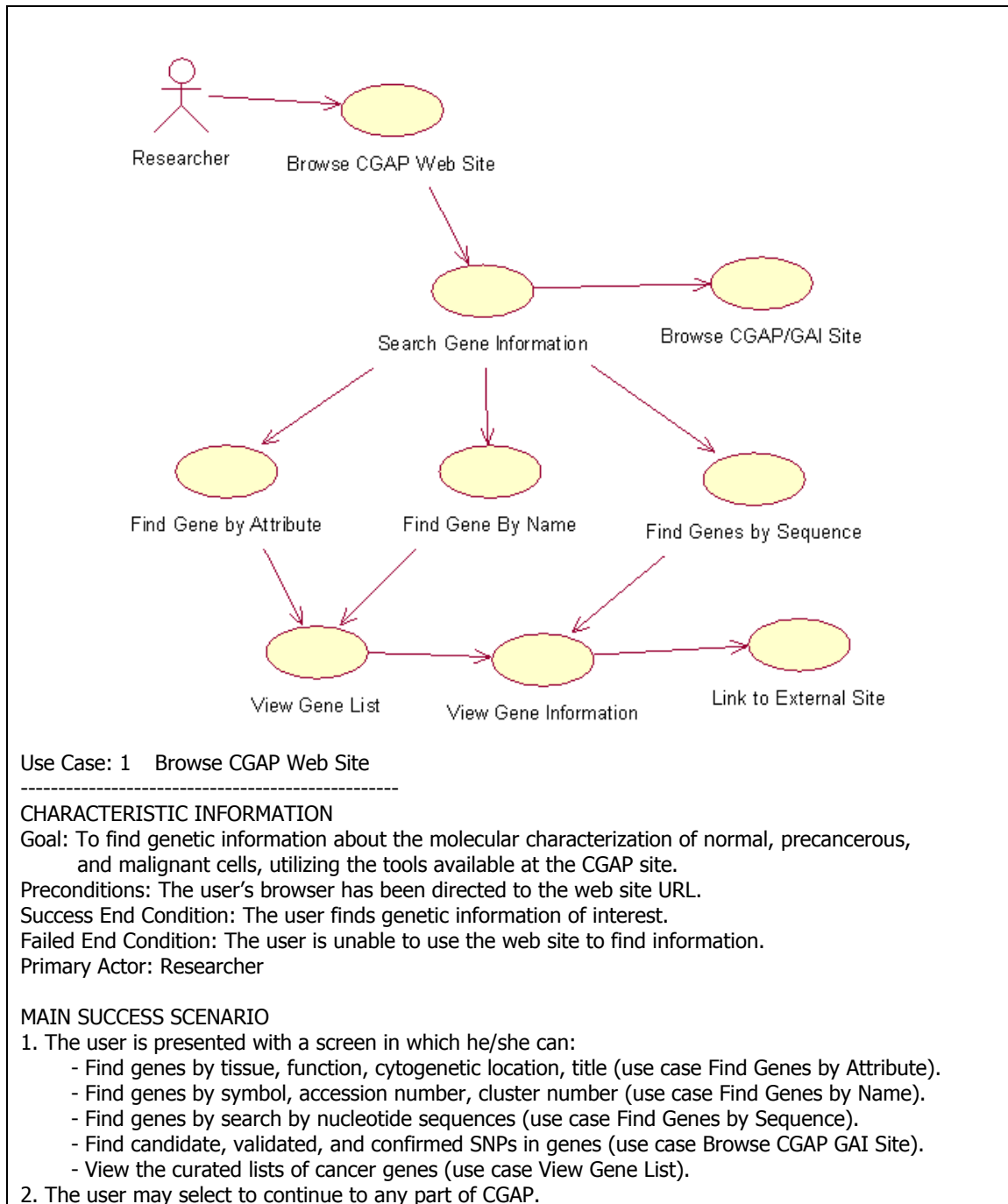
The caBIO development team applies Use Case analysis in the early design stages to informally capture high-level system requirements. Later in the design stage, as classes and their relations to one another begin to emerge, Class Diagrams help to define the static attributes, functionalities, and relations that must be implemented. As design progresses, other types of interaction diagrams are used to capture the dynamic behaviors and cooperative activities the objects must execute. Finally, additional diagrams, such as the Component, Package, and Deployment diagrams, can be used to represent pragmatic information such as the physical locations of source modules and the allocations of resources.

Many good development tools provide support for generating UML diagrams. In addition to providing a customizable interface for creating diagrams, the [Rational Rose™](#) software package also offers web publishing tools to disseminate the models by generating a suite of HTML pages. The resulting documents, originally generated during design and development, provide value throughout the software life cycle as they can rapidly familiarize new users of the system with the logic and structure of the underlying design elements. The Rose object model of the caBIO software can be browsed on the [caBIO Object Model](#) pages.

Each diagram type captures a different *view* of the system, emphasizing specific aspects of the design such as the class hierarchy, message-passing behaviors between objects, the configuration

of physical components, and user interface capabilities. Only a subset of these diagrams is discussed here, beginning with the Use Case diagram.

## 2.1 Use Case diagrams



**Figure 2.1-1 CGAP Browser Use Case**

A good starting point for capturing system requirements is to develop structured textual descriptions of how users will interact with the system. While there is no hard and fast predefined structure, Use Case descriptions typically consist of an Actor, a process, a list of

steps, and a set of pre- and post-conditions. In most cases, it is also good practice to show the post-conditions associated with success as well as failure. Figure 2.1-1 shows a sample Use Case diagram for browsing the CGAP web site, along with a textual description. Additional examples of Use Cases can be viewed by clicking [here](#).<sup>1</sup>

## 2.2 The Class diagram

The system designer utilizes the Use Case diagrams to identify the classes that must be implemented in the system, their attributes and behaviors, and the relationships and cooperative activities that must be realized. A Class diagram is used later in the design process to illustrate the hierarchy of classes and their static relationships at varying levels of detail. Figure 2.2-1 shows an abbreviated version of a UML Class diagram depicting many of the caBIO domain objects.

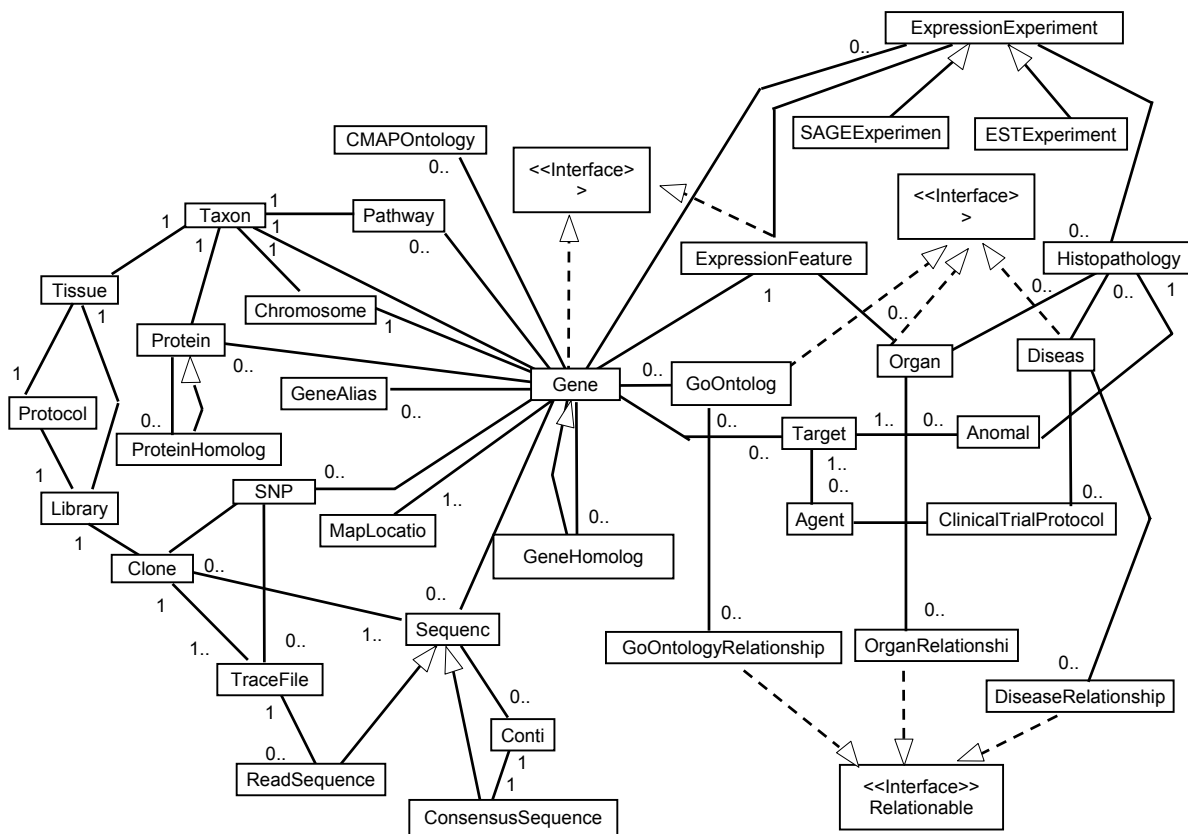


Figure 2.2-1 The caBIO Domain Objects Class diagram

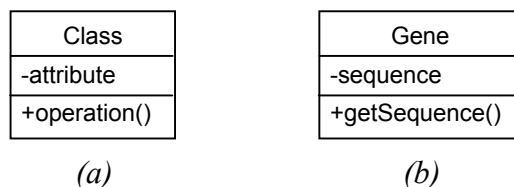
Several different types of objects are defined in the caBIO Java API; the two major types are *domain objects*, which are designed to represent class objects in specific scientific domains, and the *SearchCriteria* objects, which are associated with these domain objects. The domain objects

<sup>1</sup> Depending upon your browser, you may not be able to see the entire model. In Internet Explorer (IE), there may be no scroll bar in the left panel, depending on your Java Runtime Environment (JRE). To solve this problem, install the latest Java™ 2 Runtime Environment (the current Standard Edition is 1.4.0). In IE, then go to Tools → Internet Options → Advanced and uncheck the box under Java (Sun)—"Use Java 2 1.4.x\_xx for <applet>." Make sure also that the checkbox in the HTML published model "Display documentation" is checked.

encapsulate those attributes and properties that uniquely identify the object. For example, the *Gene* object encodes the gene’s *symbol* and *RefSeqId*, along with various other attributes. The associated *GeneSearchCriteria* object is engineered to use these attributes as selection criteria in a constructed SQL query.

Figure 2.2-1 is not an exhaustive catalog of the domain objects in caBIO but, instead, depicts a subset that is primary to bioinformatics applications and cancer research. Class objects can have a variety of possible relationships to one another, including “is derived from,” “contains,” “uses,” “is associated with,” etc. The UML provides specific notations to designate these different kinds of relations, and enforces a uniform layout of the objects’ attributes and methods — thus reducing the learning curve involved in interpreting new software specifications or learning how to navigate in a new programming environment.

Figure 2.2-2 (a) is a schematic for a UML class representation, and 2.2-2 (b) is an example of how a simple class might be represented in this scheme. The enclosing box is divided into three sections: The topmost section provides the name of the class, and is often used as the identifier for the class; the middle section contains a list of attributes (data members) for the class; the bottom section lists the object’s operations (methods). In the example below, (b) specifies the *Gene* class as having a single attribute called *sequence* and a single operation called *getSequence()*:



**Figure 2.2-2 (a) Schematic for a UML class. (b) A simple class called *Gene***

The operations and attributes of an object are called its features. The features, along with the class name, constitute the signature, or classifier, of the object. The UML provides explicit notation for the permissions assigned to a feature. UML tools vary with respect to how they represent their private, public, and protected notations for their Class diagrams.

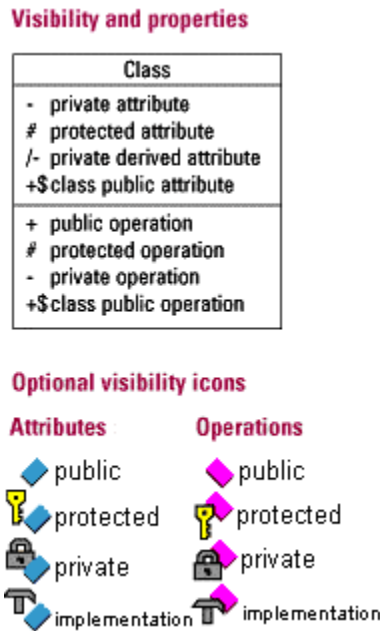
The caBIO classes represented in Figure 2.2-1 show only class names; both the operations and attributes are suppressed in that diagram. This is an example of a UML *view*: Details are hidden where they might obscure the bigger picture the diagram is intended to convey. Most UML design tools, such as Rational Rose, provide means for selectively suppressing visible details without removing the information from the underlying design model. In Figure 2.2-1, the emphasis is on the relationships that are defined among the objects, rather than on any particular class’s features.

Rational Rose uses variants on a lock and key icon (Figure 2.2-3) to indicate that a feature is public, protected, or private. Alternative notations use a “-” prefix for private features, a “+” for public features, and a “#” for protected features. In the above example, the *Gene* object’s *sequence* attribute is private and can only be accessed using the public *getSequence()* method.

In more detailed Class diagrams, it is common practice to display only those features that are part of the object’s interface. An interface is the externally visible behavior of a class or component. In most cases, this means that only the object’s public methods are shown, as the



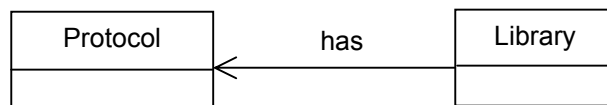
attributes are generally private or protected. The most salient information contained in Figure 2.2-1 is the objects' names and their relationships to one another, which are described next.



**Figure 2.2-3 Rational Rose access modifier representations**

A quick glance at Figure 2.2-1 shows that most of the other classes are organized around the *Gene* and *Sequence* classes. These two classes are themselves related to each other, by the *has-a* relation. More generally, the relationships occurring among the caBIO objects are of three types: association, aggregation, and generalization. The most primitive of these relationships is association, which represents the ability of one instance to send a message to another instance. The relationship between the *Gene* and *Sequence* classes is an example of an association and is depicted by a simple straight line connecting the two classes.

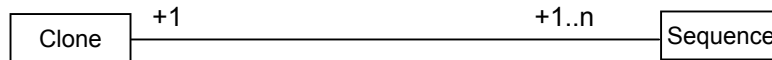
Optionally, a UML relation can have a label providing additional semantic information, as well as numerical ranges such as 1..n at its endpoints. These cardinality constraints indicate that the relationship is one-to-one, one-to-many, many-to-one, or many-to-many, according to the ranges specified and their placement. For example, the *Gene-to-Chromosome* relation in Figure 2.2-1 is many-to-one.



**Figure 2.2-4 A one-to-one association with unidirectional navigability**

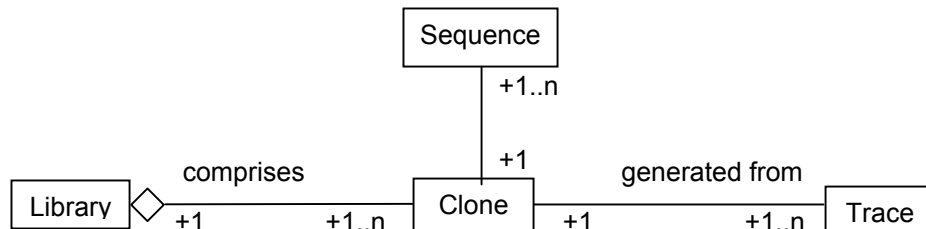
UML relations may also have directionality, as in Figure 2.2-4. Here, a *Library* object is uniquely associated with a *Protocol* object, with an arrow denoting unidirectional navigability. Specifically, the *Library* object has access to the *Protocol* object (i.e., there is a *getProtocol()* method), but the *Protocol* object does not have access to the *Library* object.

Figure 2.2-5 depicts a bidirectional many-to-one relation between *Sequence* objects and *Clone* objects. Each *Sequence* may have at most one *Clone* associated with it, while a *Clone* may be associated with many *Sequences*. To get information about a *Clone* from the *Sequence* object requires calling the *getSequenceClone()* method. Each *Clone* in turn can return its array of associated *Sequence* objects using the *getSequences()* method. This bidirectional relationship is shown using a single undirected line between the two objects.



**Figure 2.2-5 A bidirectional many-to-one relation**

Another relationship exhibited by caBIO objects is aggregation, which denotes a whole/part relationship. This relationship is exactly the same as an association with the exception that instances cannot have cyclic aggregation relationships (i.e., a part cannot contain its whole). So a *Library* can contain *Clones* but not vice-versa. Aggregation is represented by empty diamonds, as shown in the *Clone-to-Library* relation of Figure 2.2-6.



**Figure 2.2-6 Aggregation and association**

Figure 2.2-6 shows a more complex network of relations. This diagram indicates that:

- (a) one or more *Sequences* is associated with a *Clone*;
- (b) the *Clone* is contained in a *Library*, which comprises one or more *Clones*; and
- (c) the *Clone* may have one or more *Traces*.

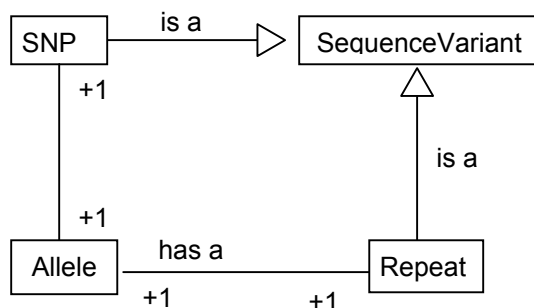
Only the relationship between the *Library* and the *Clone* is an aggregation. The others are simple associations.

In the UML, the empty diamond of aggregation designates that the whole maintains a *reference* to its part. More specifically, this means that while the *Library* is composed of *Clones*, these contained objects may have been created prior to the *Library* object's creation, and so will not be automatically destroyed when the *Library* goes out of scope.

All information retrieval in caBIO is implemented by search methods associated with the caBIO objects. Indeed, the quintessential operation is to instantiate a “blank” instance of the desired object type, define appropriate search criteria to select specific instances of that type from the databases, and use the search results to populate either the original instance itself or, alternatively, an array of instances, in the event that more than one match is found. More information about this paradigm will be detailed in Chapters 3 and 8, which discuss the caBIO domain objects and their associated search criteria and search result objects. A comprehensive listing of all of the domain objects, along with their attributes and methods, is available at the caBIO [JavaDocs](#) page.

The final relationship to be covered in this discussion of Class diagrams is generalization. Figure 2.2-7 depicts a generalization relationship between the *SequenceVariant* parent class and the *Repeat* and *SNP* classes. Classes participating in generalization relationships form a hierarchy, as depicted here.

Generalization denotes a taxonomic relationship between a more general element and a more specific element. The more specific element is fully consistent with the more general element (it has all of its properties, members, and relationships) and may contain additional information. Both the *SNP* and *Repeat* objects follow that definition. The superclass-to-subclass relationship is designated by connecting unidirectional empty arrow heads, as shown in the *SequenceVariant-to-Repeat* and *SequenceVariant-to-SNP* relations of Figure 2.2-7.



**Figure 2.2-7 Generalization relationship**

In summary, Class diagrams represent the static structure of a set of classes. Class diagrams, along with Use Cases, are the starting point when modeling a set of classes. Recall that an object is an instance of a class. Therefore, when the diagram references objects, it is representing dynamic behavior, whereas when it's referencing classes, it is representing the static structure.

### 2.3 The Rose Web Publisher Pages

As noted, the Class diagram in Figure 2.1.1 is a reduced form of the complete Class diagram. A more complete view can be found at the [caBIO Object Model](#) pages.<sup>2</sup> The contents of these pages were extracted automatically from the Java source code by the Rational Rose Web Publisher facility. Each page generates two frames; the tree structure on the left-hand side provides an index to the different views, and the right-hand side displays the diagrams associated with the currently selected view. To view the full caBIO Class diagram:

1. Expand the *Logical View* folder located in the upper-left frame.
2. Double click on the *Main* icon.
3. Click on any object in the model to view its related attributes and associates.

The tables displayed in response to selecting an element in the Class diagram are summaries of the class generated by the Rose Publisher. For example, selecting the *Protocol* class brings up a page showing several tables, including:

---

<sup>2</sup> See the previous footnote on updating your JRE if you cannot view the complete model.

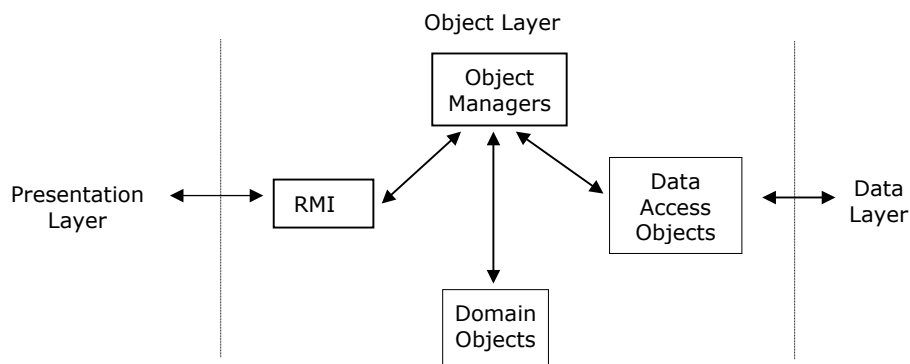
### Class Protocol {Analysis}

Parent Package	<a href="#">Logical View</a>	Abstract	No
Export Control	PublicAccess	Link Class for	None
Class Kind	NormalClass	Cardinality	n
Space		Concurrency	Sequential
Persistence	No		

### Attributes

Name	Class	Type	Initial Value
<a href="#">id</a>	<a href="#">Protocol</a>	Long	
<a href="#">name</a>	<a href="#">Protocol</a>	String	
<a href="#">type</a>	<a href="#">Protocol</a>	String	
<a href="#">description</a>	<a href="#">Protocol</a>	String	

The columns in these tables are grouped as attribute-value pairs. Thus, in the first table, the parent package for this class is the Logical View—not Export Control, which is another attribute whose value is PublicAccess.



**Figure 2.3-1 The caBIO object managers**

Double clicking on *Logical View/Managers* brings up a diagram illustrating the relationship between the various *Manager* objects and the objects that they manage. As depicted schematically in Figure 2.3-1, the *Manager* objects act as intermediaries to both the client side and the data sources on the back end, brokering requests and retrieving data as needed. The *Logical View/ExpressionClasses* detail shows both inheritance relationships among expression classes as well as their usage of related classes.

Some of these diagrams introduce additional UML notation not yet discussed — specifically, the **interface** notation. For example, the *Logical View/Beans* diagram shows a few beans and a few interfaces. A UML diagram may explicitly label an interface using the <<interface>> notation, thus making it part of its stereotype name:

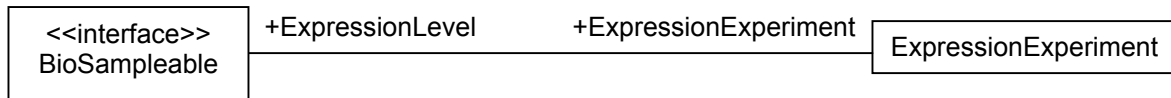
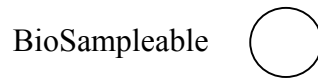


Figure 2.3-2 Interface as the stereotype name

Alternatively, a diagram may use the so-called “lollipop notation,” where an interface is represented by a label and a small circle, as in:



## 2.4 Package diagrams

Large-scale software design is a highly complex activity. As the number of classes grows to satisfy the evolving requirements of an application, the overall architectural design can quickly become obscured by this proliferation of design elements. A UML package is a logical grouping of semantically related elements, and is depicted as a labeled rectangle with a smaller rectangle attached to its upper left corner, somewhat resembling a file folder.

Packaging can be applied to any type of UML diagram, and a Package diagram is any UML diagram composed only of packages. Most commonly, packaging is used to simplify Use Case and Class diagrams. The Package diagram is not one of the nine standard UML diagrams, but as it provides a convenient way of depicting the organization of software components into packages it is described here.

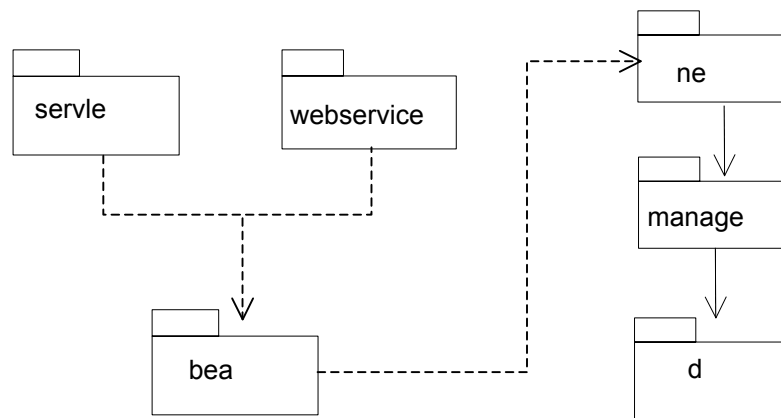


Figure 2.4-1 The caCORE packages

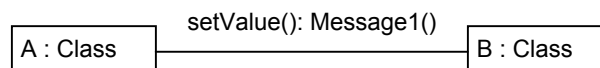
The concept of a package in a software application is similar but not identical to the notion of a UML package. Figure 2.4-1 is an abstract representation showing five logical packages that collaborate to implement the HTTP and SOAP interfaces in caCORE. The *db* package at the bottom provides access to the backend database; the *net* and *manager* packages provide middleware; and the *servlet* and *webservices* packages at the top implement the HTTP and SOAP interfaces, respectively. In the actual Java implementation, each database has its own *bean* package, for example, *gov.nih.nci.cabio.bean*, *gov.nih.nci.cadsr.bean*, etc. These packages are described in more detail in Chapter 9.

The organization of software components into packages is used to increase reusability and to minimize compile-time dependencies. It is highly unusual to reuse a single class, but quite common to reuse a collection of related classes that collaborate to produce some desired functionality. The UML models of the caCORE software that are available on the Rose Publisher pages approximately reflect the actual Java package structure but do not have a one-to-one correspondence.

The purpose of the next two types of diagrams is to describe the dynamic behavior of selected sets of objects—specifically, their communications and sequences of execution.

## 2.5 Collaboration and Sequence diagrams

Collaboration diagrams are used to describe interactions among different objects or classes. A Collaboration diagram can describe both the static structure and the dynamic behavior of a system. The structure of a Collaboration diagram generally will include two components. First, it may include a set of Object Instances and Class Names, or just the Class Names alone. Second, the messages and their directional nature are described by the arrows. Messages can be unidirectional or bidirectional and are typically method calls within the code.

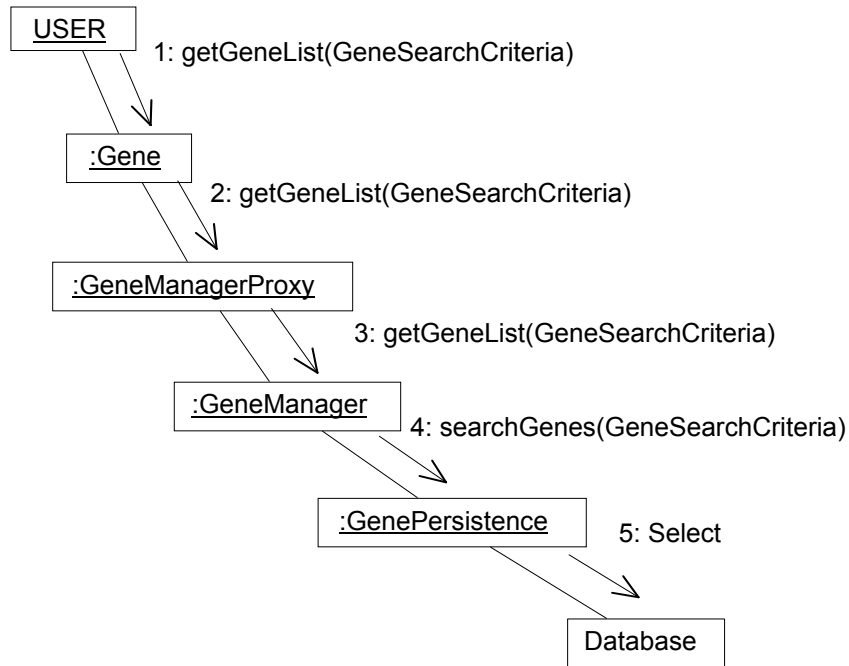


**Figure 2.5-1 Collaboration between objects**

Figure 2.5-1 is an example of collaboration between objects *A* and *B*, which are represented by rectangles. The black arrow and method call depict the dependency between these two objects: Object *A* calls the *setValue()* method that is on object *B*. *A* may have a dependency on the return value or may just need to change the state of *B*.

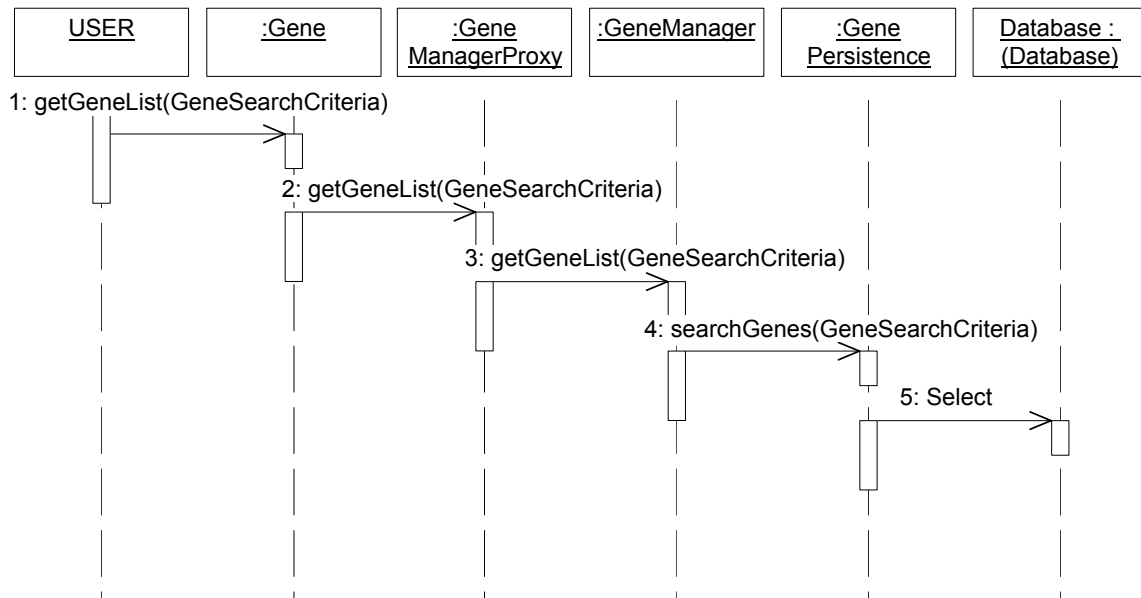
Figure 2.5-2 diagrams the collaborations of caBIO objects called into action when searching for a gene. Four objects participate in the collaboration, as indicated by the leading colons in the labels. The user initiates the activity by invoking the *getGeneList()* method on a *Gene* object. The *Gene* object passes this message on to the *GeneManagerProxy* object, which in turn passes it on to the *GeneManager* object. The *GeneManager* then invokes the *searchGenes()* method on a *GenePersistence* object, which finally forwards the request to the database as an SQL select statement.

As indicated by the messages, each of the first three objects implements the *getGeneList()* method, and the argument, in each case, is a *GeneSearchCriteria* object. This interaction between domain objects—such as the *Gene* in this case—and the associated search criteria object, is elaborated on in Chapters 3 and 8. The object managers and their proxies serve to insulate the domain objects from both the data and presentation layers, thus allowing for a clean separation between the logical design and the business rules of the implementation.



**Figure 2.5-2 Collaboration diagram for retrieving genes**

Figure 2.5-3 shows a Sequence diagram depicting the same activities over time. Sequence diagrams are very much like Collaboration diagrams in that they describe interactions among objects. The Sequence diagram, however, describes the interaction of objects in terms of an exchange of messages over time.



**Figure 2.5-3 Sequence diagram for retrieving genes**

The components of a Sequence diagram are very much the same as those of a Collaboration diagram. The Sequence diagram represents instantiated objects and their associated interactions as messages. However, the orientation of a Sequence diagram allows it to introduce a time component as represented by the horizontal bars. The diagram is read from top to bottom and describes the sequence of execution. The next several chapters provide more details of the domain objects developed for the various applications supported by the caCORE APIs.



### **3.0 THE caBIO DOMAIN OBJECTS**

### 3.1 The Object Hierarchies

Most of the packages prefixed by *gov.nih.nci.cabio* were initially defined in the caCORE 1.0 release. At that time, the primary application domains supported by the software involved genomic analysis, and the primary data types were centered around genes, taxa, sequences, diseases, and expression data. The term “domain objects” was used to distinguish those objects that modeled domain-specific entities from the more general implementation classes involved in the presentation and data layers.

The caBIO domain objects are implemented as Java beans in the [gov.nih.nci.caBIO.bean](#) package, and include those classes that correspond to biological entities and bioinformatic concepts. As depicted in Figure 3.1-1, the domain objects in the *bean* package define a wide, shallow hierarchy that is just three levels deep. The caBIO domain objects descend directly from *PersistentCaBIOBean*, and only four of the domain objects have subclasses.

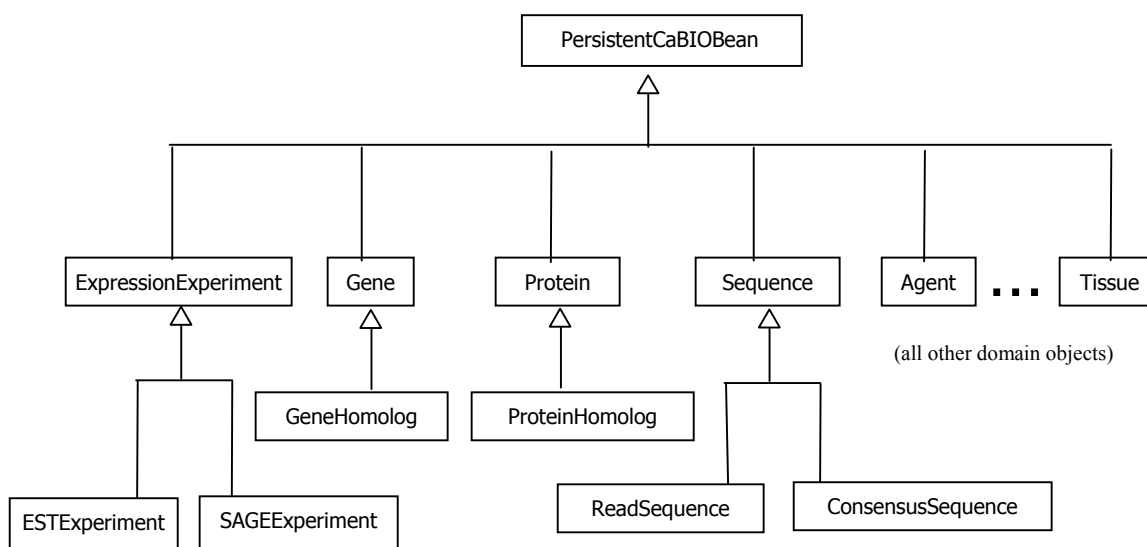
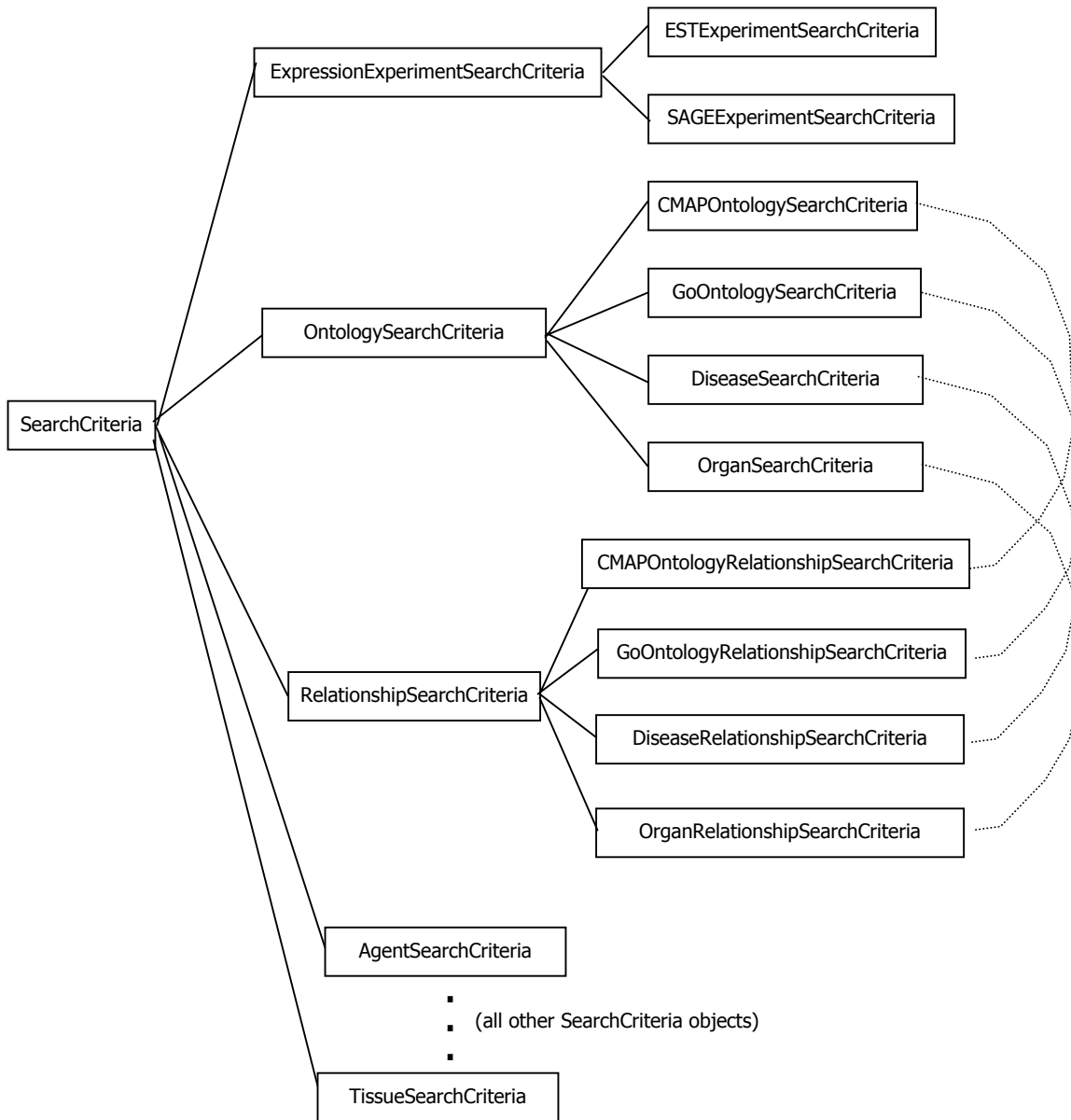


Figure 3.1-1 The domain object hierarchy

Additional logical and tactical structures, however, derive from the interfaces implemented by some of these domain objects, and from related bean objects, such as the *SearchCriteria* beans and a small set of “relationable” objects.

Each domain object *Xxx* has a corresponding *SearchCriteria* named *XxxSearchCriteria*. Thus, for example, there is a *GeneSearchCriteria* object and a *LibrarySearchCriteria* object corresponding to the *Gene* and *Library* domain objects, respectively. The inheritance relations that hold among the *SearchCriteria* objects reflect their associations with the domain objects and, in addition, capture the interfaces that are implemented by some of the domain objects. Figure 3.1-2 shows the *SearchCriteria* object hierarchy.

All objects in the *bean* package implement the *java.io.serializable* interface. In addition, all domain objects implement *gov.nih.nci.caBIO.util.XMLInterface*, thus facilitating their transport to the Presentation Layer where the SOAP and HTTP APIs are implemented.



**Figure 3.1-2 The *SearchCriteria* inheritance hierarchy**

Specifically, each domain object implements the following methods:

- *toXML()* – returns an XML-encoding of all of the object’s “top-level” attributes (i.e., number and character-valued features), with all “deeper” information (e.g., arrays, embedded objects, etc.) encoded as XLinks. This is the default XML-encoding for the SOAP and HTTP interfaces. The notion of an XLink is similar to a pointer or reference in a programming language; the XML Linking Language (XLink) allows complex elements to be embedded in XML documents as URLs that can be subsequently deployed to retrieve the elements themselves.

- *toXMLDOC()* – returns an XML-encoding of *all* of the object’s attributes; i.e., all XLinks are filled in one level deep. This method implements the *getHeavyXML* options used by the SOAP and HTTP interfaces.
- *toXMLProcessor(...)* – takes a list of *fillin* tags specifying which XLinks are to be selectively expanded in the XML-encoding.

Six additional interfaces are defined in *gov.nih.nci.caBIO.bean*: *Expressable*, *EVSInterface*, *ExpressionLevel*, *GeneInterface*, *Ontologable*, and *Relationable*. Several of these interfaces are implemented by only one or two classes as they represent a software design the caBIO project is either migrating towards or away from.

The *Expressable* and *GeneInterface* interfaces represent shared functionality that will be more fully implemented in future releases. Currently, only the *Gene* object implements *Expressable*. This interface defines a single method: *getExpression()*, which returns an array of *ExpressionExperiments* containing expression levels for the *Gene*. But the definition of “expressable” as an interface also posits an implicit relationship between the *Gene* class and the *ExpressionExperiment* objects in the hierarchy, as each of these implements the method *getExpressables()*, which returns an array of *Expressable* objects.

*Gene* objects are the effective portal to most of the genomic information provided by the caBIO data services; organs, diseases, chromosomes, pathways, sequence data, and expression experiments are among the many objects accessible via a gene. The *GeneInterface* defines a protocol of behavior that can be implemented by any class wishing to classify itself as a *Gene*. As an interface, it defines a set of methods but does not implement them. Any class that implements the *GeneInterface* agrees to implement all of the methods defined in the interface, thereby agreeing to certain behavior. Currently, the *GeneInterface* is implemented only by the *EngineeredGene* class in the *gov.nih.nci.caMOD.bean* package.

The *EVSInterface* has been deprecated and exists for historical reasons and backwards compatibility; this interface has effectively been replaced by the new *gov.nih.nci.evs* packages. The *ExpressionLevel* interface is implemented by the *CGAPExpressionLevel* class in the *gov.nih.nci.cabio.util* package. This interface supports measures of gene expression levels specialized for CGAP data, such as the “color” or “expression ratio” observed for the data.

A more complex set of relationships derives from the *Ontologable* and *Relationable* interfaces. The *Organ*, *Disease*, *CMAPOntology*, and *GoOntology* classes all implement the *Ontologable* interface, as each of these object types defines entities occurring in externally defined ontologies or taxonomies. The [Gene Ontology Consortium](#), for example, defines three gene ontologies, based on molecular function, biological process, and cellular location of the gene. Similarly, the CMAP ontology maps genes according to functional classifications. Other controlled vocabularies, such as those defined by the [Enterprise Vocabulary Services](#) at NCI, define disease and organ taxonomies.

Each instance of a relationship object stores its relationship *type* (child/parent) and the set of *Ontologable* objects participating in that relationship. For example, an *Organ* representing the heart might have a parent relationship to two other *Organs* representing the left and right ventricles. The parent’s method *getChildRelationships()* would return this (object-ified) relationship, and the relationship, in turn, would provide access to the *Ontologable* children

stored there. More specifically, all *Relationable* objects implement the *getOntologies()* method for just this purpose.

While the inheritance hierarchy for the domain objects does not reveal these interface implementations, the hierarchy of *SearchCriteria* objects (Figure 3.1-2) does. As described in Chapter 8, the *SearchCriteria* objects are a critical part of the infrastructure that provides caBIO's powerful database search mechanisms.

Most of the domain objects defined in the caBIO API are caBIO domain objects that specialize in bioinformatics applications and, in particular, gene expression analysis. This chapter concludes with a catalog of these domain objects. Chapters 4 through 7 describe the new domain objects introduced in Release 2.0 to provide programmatic interfaces to the EVS services, the caDSR project, and the GEDP and caMOD databases respectively. The online [JavaDocs](#) pages also provide comprehensive specifications of the objects, interfaces, and packages that together define the caBIO architecture and services.

## 3.2 The caBIO Domain Object Catalog

This catalog lists the objects defined in the *gov.nih.nci.cabio.bean* package. Items in the listing below should be interpreted as follows:

- *Application*: This field indicates in what contexts the object is most likely to be used.
- *Related domain objects*: A second domain object is “related” to the first domain object if that second object occurs anywhere in the signature of a method for the first object.
- *Extends*: This field reflects direct inheritance; i.e., the current class is a direct subclass of that which it extends.
- *Implements*: This field lists all interfaces implemented by the object.

In cases where no applications, relations, extensions, or interface implementations apply, the above fields are omitted.

### 3.2.1 [Agent](#)

A therapeutic agent (drug, intervention therapy) used in a clinical trial protocol.

*Application*: used primarily by CMAP and EVS applications.

*Related domain objects*: [ClinicalTrialProtocol](#), [Target](#)

*Extends*: PersistentCaBIOBean

*Implements*: XMLInterface, java.io.Serializable

### 3.2.2 [Anomaly](#)

An irregularity in either the expression of a gene or its structure (i.e., a mutation).

*Application*: defined and used by the CMAP project.

*Related domain objects*: [Histopathology](#), [Target](#)

*Extends*: PersistentCaBIOBean

*Implements*: XMLInterface, java.io.Serializable

### 3.2.3 [Chromosome](#)

An object representing a specific chromosome for a specific taxon; provides access to all known genes contained in the chromosome and to the taxon.

*Application:* used by CMAP and other applications to reason about the molecular basis of cancer.

*Related domain objects:* [Gene](#), [Taxon](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable

### **3.2.4 [ClinicalTrialProtocol](#)**

The protocol associated with a clinical trial; organizes administrative information about the trial such as Organization ID, participants, phase, etc., and provides access to the administered Agents.

*Application:* used primarily by CMAP.

*Related domain objects:* [Agent](#), [ProtocolAssociation](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable

### **3.2.5 [Clone](#)**

An object used to hold information pertaining to I.M.A.G.E. clones; provides access to sequence information, associated trace files, and the clone's library.

*Application:* imported from the CGAP web site databases.

*Related domain objects:* [Sequence](#), [Library](#), [TraceFile](#), [SNP](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable

### **3.2.6 [CMAPOntology](#)**

An object providing entry to the CMAP gene ontology, which categorizes genes by function; provides access to gene objects corresponding to the ontological term, as well as to ancestor and descendant terms in the ontology tree.

*Application:* defined and used by CMAP applications.

*Related domain objects:* [CMAPOntologyRelationship](#), [Gene](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable, [Ontologable](#)

### **3.2.7 [CMAPOntologyRelationship](#)**

An object specifying a child or parent relationship between CMAPOntology objects.

*Application:* used and defined by CMAP applications.

*Related domain objects:* [CMAPOntology](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable, [Relationable](#)

### **3.2.8 [ConceptSearch](#)**

Represents a searchable concept term in a controlled vocabulary occurring in the NCI Metathesaurus; used to find synonym or semantic types for the concept of interest. This object is now deprecated as it has been replaced by domain objects in the [EVS bean](#) package.

*Application:* used primarily by EVS applications.

*Extends:* PersistentCaBIOBean

### **3.2.9** [ConsensusSequence](#)

A specialization of the Sequence class; represents the consensus of a set of contigs, which it also provides access to.

*Application:* used by the GAI project to identify SNPs.

*Related domain objects:* [Contig](#), [Gene](#), [Protein](#), [Clone](#), [ExpressionMeasurement](#)

*Extends:* [Sequence](#)

*Implements:* XMLInterface, java.io.Serializable

### **3.2.10** [Contig](#)

One of the set of overlapping sequence fragments used to assemble a consensus sequence, which it also provides access to.

*Application:* Used by the GAI project to identify SNPs.

*Related domain objects:* [Sequence](#), [ConsensusSequence](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable

### **3.2.11** [Disease](#)

Specifies a disease name and ID; also provides access to: ontological relations to other diseases; clinical trial protocols treating the disease; and specific histologies associated with instances of the disease.

*Application:* used by the CMAP project.

*Related domain objects:* [ClinicalTrialProtocol](#), [Histopathology](#), [DiseaseRelationship](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable, [Ontologable](#)

### **3.2.12** [DiseaseRelationship](#)

An object specifying a child or parent relationship between Disease objects.

*Application:* used by the CMAP project.

*Related domain objects:* [Disease](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable, [Relationable](#)

### **3.2.13** [ESTExperiment](#)

An object representing data from an expression experiment using expressed sequence tags.

*Application:* caBIO's EST experiment data are downloaded from the CGAP databases.

*Related domain objects:* [ExpressionExperiment](#), [Gene](#), [Histopathology](#)

*Extends:* [ExpressionExperiment](#)

*Implements:* XMLInterface, java.io.Serializable

### **3.2.14** [ExpressionExperiment](#)

A virtual class defining the methods and attributes shared by various types of expression experiments, including ESTExperiment and SAGEExperiment objects.

*Related domain objects:* [Gene](#), [Histopathology](#), [ESTExperiment](#), [SAGEExperiment](#)  
*Extends:* PersistentCaBIOBean  
*Implements:* XMLInterface, java.io.Serializable

### **3.2.15** [ExpressionFeature](#)

Associated with a [Gene](#) object through the gene's method [getExpressionFeature\(\)](#); provides access to the list of organs where the gene is known to be expressed.  
*Application:* Expression information for a gene is extracted from the CGAP databases, which are based on the information in Unigene (see discussion of data sources in [Chapter 15 The CaBIO Data](#)).  
*Related domain objects:* [Organ](#), [Gene](#).  
*Extends:* PersistentCaBIOBean  
*Implements:* XMLInterface, java.io.Serializable

### **3.2.16** [ExpressionLevelDesc](#)

Used to capture descriptive, unquantifiable annotations of the observed expression levels.  
*Application:* Created to support caMOD gene expression data.  
*Extends:* PersistentCaBIOBean  
*Implements:* XMLInterface, java.io.Serializable

### **3.2.17** [ExpressionMeasurement](#)

An object representing a structure that is capable of measuring the absolute or relative amount of a given compound.  
*Related domain objects:* [Gene](#), [Sequence](#)  
*Extends:* PersistentCaBIOBean  
*Implements:* XMLInterface, java.io.Serializable

### **3.2.18** [ExpressionMeasurementArray](#)

An array of ExpressionMeasurement objects.  
*Related domain objects:* [Gene](#), [Sequence](#)  
*Extends:* PersistentCaBIOBean  
*Implements:* XMLInterface, java.io.Serializable

### **3.2.19** [Gene](#)

The effective portal to most of the genomic information provided by the caBIO data services; organs, diseases, chromosomes, pathways, sequence data, and expression experiments are among the many objects accessible via a gene.  
*Related domain objects:* [ExpressionFeature](#), [Organ](#), [Disease](#), [Chromosome](#), [Taxon](#), [Sequence](#), [GeneAlias](#), [GeneHomolog](#), [MapLocation](#), [Protein](#), [SNP](#), [Target](#), [ExpressionMeasurement](#), [Pathway](#), [GoOntology](#)  
*Extends:* PersistentCaBIOBean  
*Implements:* XMLInterface, java.io.Serializable, [Expressable](#)



### 3.2.20 [GeneAlias](#)

An alternative name for a gene; provides descriptive information about the gene (as it is known by this alias), as well as access to the Gene object it refers to.

*Related Domain Objects:* [Gene](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable

### 3.2.21 [GeneHomolog](#)

Defined only in relation to another Gene object of interest, the functional equivalent of that gene in another taxon (i.e., its ortholog). The GeneHomolog is a specialization of the parent Gene object; in addition to all of the methods provided by the gene interface, the homolog object provides the percent of sequence similarity of the homolog to the related gene of interest.

*Related domain objects:* [Gene](#), [ExpressionFeature](#), [Organ](#), [Disease](#), [Chromosome](#), [Taxon](#), [Sequence](#), [GeneAlias](#), [GeneHomolog](#), [MapLocation](#), [Protein](#), [SNP](#), [Target](#), [ExpressionMeasurement](#), [Pathway](#), [GoOntology](#)

*Extends:* [Gene](#)

*Implements:* XMLInterface, java.io.Serializable, [Expressable](#)

### 3.2.22 [GoOntology](#)

An object providing entry to a Gene object's position in the [Gene Ontology Consortium's](#) controlled vocabularies, as recorded by [LocusLink](#); provides access to Gene objects corresponding to the ontological term, as well as to ancestor and descendant terms in the ontology tree.

*Related domain objects:* [Gene](#), [GoOntologyRelationship](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable, [Ontologable](#)

### 3.2.23 [GoOntologyRelationship](#)

An object specifying a child or parent relationship between GoOntology objects.

*Related domain objects:* [GoOntology](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable, [Relationable](#)

### 3.2.24 [Histopathology](#)

An object representing anatomical changes in a diseased tissue sample associated with an expression experiment; captures the relationship between organ and disease.

*Application:* used by the CMAP project.

*Related domain objects:* [Anomaly](#), [Organ](#), [Disease](#), [ExpressionExperiment](#), [ESTExperiment](#), [SAGEExperiment](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable

### **3.2.25** [Library](#)

An object representing a CGAP library; provides access to information about: the tissue sample and its method of preparation, the library protocol that was used, the clones contained in the library, and the sequence information derived from the library.

*Application:* Extracted from the CGAP databases.

*Related domain objects:* [Clone](#), [Sequence](#), [Tissue](#), [Protocol](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable

### **3.2.26** [MapLocation](#)

Associated with a Gene object, the physical map location of the gene.

*Related domain objects:* [Chromosome](#), [Gene](#), [Taxon](#).

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable

### **3.2.27** [Organ](#)

A representation of an organ whose name occurs in a controlled vocabulary; provides access to any Histopathology objects for the organ and, because it is “ontolog-able,” provides access to its ancestral and descendant terms in the vocabulary.

*Related domain objects:* [Histopathology](#), [OrganRelationship](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable, [Ontologable](#)

### **3.2.28** [OrganRelationship](#)

An object specifying a child or parent relationship between Organ objects.

*Related domain objects:* [Organ](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable, [Relationable](#)

### **3.2.29** [Pathway](#)

An object representation of a molecular/cellular pathway compiled by [BioCarta](#). Pathways are associated with specific Taxon objects, and contain multiple Gene objects, which may be targets for treatment.

*Related domain objects:* [Gene](#), [Taxon](#), [TargetTarget](#).

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable

### **3.2.30** [Protein](#)

An object representation of a protein; provides access to the encoding gene via its [GenBank](#) ID, the taxon in which this instance of the protein occurs, and references to homologous proteins in other species.

*Related domain objects:* [Gene](#), [ProteinHomolog](#), [Taxon](#).

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable

### **3.2.31 [ProteinHomolog](#)**

Defined only in relation to another Protein object of interest, the functional equivalent of that protein in another taxon (i.e., its ortholog). The ProteinHomolog is a specialization of the parent Protein object; in addition to the methods provided by the parent class, the Homolog object provides the percent of sequence similarity of the homolog to the related protein of interest.

*Related domain objects:* [Gene](#), [Protein](#), [Taxon](#).

*Extends:* [Protein](#)

*Implements:* XMLInterface, java.io.Serializable

### **3.2.32 [Protocol](#)**

An object representation of the protocol used in assembling a clone library.

*Related domain objects:* [Library](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable

### **3.2.33 [ProtocolAssociation](#)**

An association class relating ClinicalTrialProtocols to Diseases.

*Application:* used primarily by the CMAP project.

*Related domain objects:* [ConceptSearch](#), [ClinicalTrialProtocol](#), [Disease](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable

### **3.2.34 [ReadSequence](#)**

The output of a TraceFile object, an ASCII representation of the nucleotide sequence; a read sequence is created by running PHRED.

*Application:* used in the GAI project.

*Related domain objects:* [TraceFile](#), [SNP](#)

*Extends:* [Sequence](#)

*Implements:* XMLInterface, java.io.Serializable

### **3.2.35 [SAGEExperiment](#)**

A specialization of the ExpressionExperiment class, used to represent serial analysis of gene expression (SAGE) data.

*Application:* derived from methods developed at NCI by the CGAP project in collaboration with Duke University.

*Related domain objects:* [ExpressionExperiment](#), [Gene](#), [Histopathology](#)

*Extends:* [ExpressionExperiment](#)

*Implements:* XMLInterface, java.io.Serializable

### **3.2.36 [Sequence](#)**

An object representation of a gene sequence; provides access to the clones from which it was derived, the ASCII representation of the residues it contains, and the sequence ID.

*Related domain objects:* [Clone](#), [Gene](#), [Protein](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable

### **3.2.37** [SNP](#)

A Single Nucleotide Polymorphism; provides access to the clones and trace files from which it was identified, the two most common substitutions at that position, the offset of the SNP in the parent sequence, and a confidence score.

*Application:* identified by the GAI project.

*Related domain objects:* [Clone](#), [TraceFile](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable

### **3.2.38** [Target](#)

A gene thought to be at the root of a disease etiology and targeted for therapeutic intervention.

*Application:* defined and used by the CMAP project.

*Related domain objects:* [Agent](#), [Anomaly](#), [Gene](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable

### **3.2.39** [Taxon](#)

An object representing the various names (scientific, common, abbreviated, etc.) for a species associated with a specific Gene, Chromosome, Pathway, Protein, or Tissue object.

*Related domain objects:* [Gene](#), [Chromosome](#), [Pathway](#), [Protein](#), [Tissue](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable

### **3.2.40** [Tissue](#)

A group of similar cells united to perform a specific function.

*Related domain objects:* [Disease](#), [Organ](#), [Protocol](#), [Taxon](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable

### **3.2.41** [TraceFile](#)

The recorded trace file used to identify a SNP, based on the observed intensities for the four possible bases at each position in the sequence.

*Application:* made available to caBIO from Washington University.

*Related domain objects:* [Clone](#), [ReadSequence](#), [SNP](#)

*Extends:* PersistentCaBIOBean

*Implements:* XMLInterface, java.io.Serializable

## **4.0 THE EVS DOMAIN OBJECTS**

The Enterprise Vocabulary Services project is a collaborative effort of the Center for Bioinformatics and the NCI [Office of Communications](#). The *NCI Thesaurus* is a biomedical thesaurus developed by EVS in response to a need for consistent shared vocabularies among the various projects and initiatives at the National Cancer Institute—as well as in the entire cancer research community. Controlled vocabularies are important to any application involving electronic data sharing; two areas where the need is perhaps most apparent are clinical trials data collection and reporting and more generally, data annotation of any kind. The EVS project also produces the *NCI Metathesaurus*, which is based on [NLM's Unified Medical Language System Metathesaurus](#) (UMLS) supplemented with additional cancer-centric vocabulary.

A critical need served by the EVS is the provision of a well designed ontology covering cancer science. Such an ontology is required for data annotation, inferencing and other functions. The data to be annotated might be anything from genomic sequences to case report forms to cancer image data. The Thesaurus covers all of these domains, as it includes vocabularies pertinent to disease, biomedical instrumentation, anatomical structure, and gene/protein information – to mention but a few of the included specialties. The NCI Thesaurus has recently been ranked by the National Center for Vital Health Statistics as one of the two best biomedical terminologies in the country, and has been nominated as a standard by the Consolidated Health Informatics initiative, the health-related component of the eGOV initiative (<http://aspe.hhs.gov/sp/nhii/News/hixs.htm>). The Thesaurus is updated monthly, keeping abreast of developments in cancer science.

The NCI Thesaurus is implemented as a Description Logic vocabulary and, as such, is a self-contained and logically consistent terminology. Unlike the NCI Thesaurus, the purpose of the NCI Metathesaurus is *not* to provide unequivocal or even necessarily consistent definitions. Like the UMLS Metathesaurus itself, the purpose of the NCI Metathesaurus is to provide mappings of terms across vocabularies. The caBIO objects described in this section provide access to both the NCI Thesaurus and the NCI Metathesaurus.

In previous releases of the caCORE, the EVS API provided access only to the NCI Metaphrase server, which hosts the Metathesaurus database. In this release, the API has been extended to provide access to both the [NCI Distributed Terminology Server](#) (DTS), which hosts the NCI Thesaurus and several other vocabularies, as well as access to [NCI Metaphrase](#).

NCI licenses the Metaphrase and DTS servers from Apelon Inc. Each server has a proprietary Java API. Because of the proprietary nature of these APIs, these interfaces cannot be made available to the public. Furthermore, NCI has extended and otherwise modified the Metaphrase and DTS servers to provide functionality that is not present in the commercial version of these products. Therefore, NCI developed a public domain open source wrapper that provides full access to the basic and enhanced capabilities of both servers. This public API is a component of caBIO in caCORE 2.0.

Before actually describing the caCORE Java API to the EVS, this chapter begins with a brief overview of the UMLS Metathesaurus, upon which the NCI Metathesaurus is based. This is followed by a short discussion of description logic, its role in the area of knowledge representation, and its implementation in the NCI Thesaurus. A sample Java program that uses the API described here is discussed in Chapter 12 and listed in Appendix B.

## 4.1 The UMLS Metathesaurus

As noted above, the NCI Metathesaurus is based on the UMLS Metathesaurus, supplemented with additional cancer-centric vocabulary. Excellent documentation on the UMLS is available at the UMLS Knowledge Sources web site at:

<http://www.nlm.nih.gov/research/umls/UMLSDOC.HTML>

A brief overview of the UMLS Metathesaurus is included here, but it is strongly recommended that users who wish to gain a deeper understanding refer to the above web site. Only those features of the UMLS Metathesaurus which are relevant to accessing the NCI Metathesaurus are described here.

The UMLS Metathesaurus is a unifying database of concepts that brings together terms occurring in over 100 different controlled vocabularies used in biomedicine. When adding terms to the Metathesaurus, the UMLS philosophy has been to preserve all of the original meanings, attributes, and relationships defined for those terms in the source vocabularies, and to retain explicit source information as well. In addition, the UMLS editors add basic information about each concept and introduce new associations which help to establish synonymy and other relationships among concepts from different sources.

Given the very large number of related vocabularies incorporated in the Metathesaurus, there are instances where the same concept may be known by many different names, as well as instances where the same names are intended to convey different concepts. To avoid ambiguity, the UMLS employs an elaborate indexing system, the central kingpin of which is the *concept unique identifier* (CUI). Similarly, each unique concept name or string in the Metathesaurus has a string unique identifier (SUI).

In cases where the same string is associated with multiple concepts, a numerical tag is appended to that string to render it unique as well as to reflect its multiplicity. In addition, the UMLS Metathesaurus editors may create an alternative name for the concept which is more indicative of its intended interpretation. In these cases, all three names for the concept are preserved.

Several types of relationships are defined in the UMLS Metathesaurus, and four of these are captured by the NCI Metaphrase interface:

Broader (RB)	The related concept has a more general meaning.
Narrower (RN)	The related concept has a more specific meaning.
Synonym (SY)	The two concepts are synonymous.
Other related (RO)	The relation is not specified but is something other than synonymous, narrower, or broader.

The UMLS *Semantic Network* is an independent construct whose purpose is to provide consistent categorization for all concepts contained in the UMLS Metathesaurus, and to define a useful set of relationships among these concepts. As of the 2003AB release, the Semantic Network defined a set of 135 basic semantic types or categories, which could be assigned to these concepts, and 54 relationships that could hold among these types.

The major groupings of semantic types include organisms, anatomical structures, biologic function, chemicals, events, physical objects, and concepts or ideas. Each UMLS Metathesaurus

concept is assigned at least one semantic type, and in some cases, several. In all cases, the most specific semantic type available in the network hierarchy is assigned to the concept.

The NCI Metathesaurus includes most of the UMLS Metathesaurus, with certain proprietary vocabularies of necessity excluded. In addition, the NCI Metathesaurus includes terminologies developed at NCI along with external vocabularies licensed by NCI. The local vocabularies developed at NCI are described in Table 4.1-1. As noted there, a limited model of the NCI Thesaurus is also accessible via the NCI Metathesaurus, as the NCI Source. Additional external vocabularies include [MedDRA](#), [SNOMED](#), [ICD-O-3](#), and other proprietary vocabularies.

The NCI Metathesaurus is available through the Metaphrase interface described in the NCICB Applications User Manual as well as through the Java API described in this chapter.

**Table 4.1-1 NCI local source vocabularies included in the Metathesaurus.**

<u>Vocabulary</u>	<u>Content</u>	<u>Usage</u>
NCI Source	Limited model of the NCI Thesaurus	Reference terminology for cancer research applications
NCIPDQ	Expanded and re-organized PDQ	CancerLit indexing and clinical trials accrual
NCISEER	SEER terminology	Incidence reporting
CTEP	CTEP terminology	Clinical trials administration
MDBCAC	Topology and Morphology	Cancer genome research
ELC2001	NCBI tissue taxonomy	Tissue classification for genetic data such as cDNA libraries.
ICD03	Oncology classifications	Cancer genome research and incidence reporting
MedDRA	Regulatory reporting terminology	Adverse event reporting
MMHCC	Mouse Cancer Database terminology	Mouse Models of Human Cancer Consortium
CTRM	Core anatomy, diagnosis and agent terminology	Translational research by NCICB applications

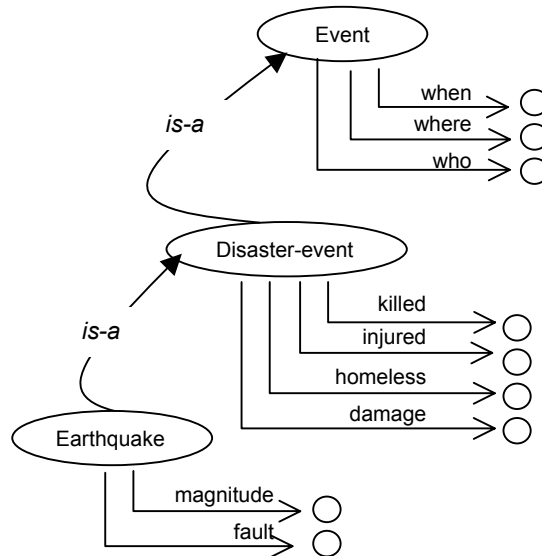
## **4.2 Knowledge Representations and Description Logic**

Knowledge representation has long been a prime focus in artificial intelligence research. This area of research asks how one can accurately encode the rich and highly detailed world of information that is required for the application area being modeled and yet, at the same time, capture the implicit commonsense knowledge. One of the most common approaches to this problem in the 1970s was to utilize *frame-based* representations.

The basic idea of a frame is that important objects in our world fall into natural classes, and that all members of these classes share certain properties or attributes, called *slots*. For example, all dogs have four legs, a tail (or vestige of one), whiskers, etc. Restaurants generally have tables, chairs, eating utensils, and menus. Thus, when we enter a new restaurant or encounter a



new dog, we already have a “frame of reference” and some expectations about the properties and behaviors of these entities.



**Figure 4.2-1 An earthquake in a semantic network of news stories**

In a seminal paper by Marvin Minsky published in 1975, he placed the frame representation paradigm in the context of a *semantic network* of nodes, attributes, and relations. Figure 4.2-1 shows a simple frame-based representation of an earthquake, as it might be used in a semantic network of news stories.<sup>3</sup>

At the same time that frame-based representations were being explored, a popular alternative approach was to use (some subset of) first-order predicate logic (FOL)—often implemented as a Prolog program. While propositional logic allows one to make simple statements about concrete entities, a complete first-order logic allows one to make general statements about anonymous elements, with the introduction of variables as placeholders. The example below contrasts the difference in expressivity between propositional logic and FOL:

<u>Propositional Logic</u>	<u>First-order Predicate Logic</u>
All men are mortal.	$\forall x: \text{Man}(x) \rightarrow \text{Mortal}(x)$
Socrates is a man.	$\text{Man}(\text{Socrates})$
Socrates is mortal.	

**Figure 4.2-2 Propositional versus first-order predicate logic**

In other words, in FOL it is possible to express general rules of inference that can be applied to all entities whose attributes satisfy the left-hand side of the  $\rightarrow$  inference operator. Thus, simply asserting  $\text{Man}(\text{Socrates})$  entails  $\text{Mortal}(\text{Socrates})$ .

Since logic programming is based on the tenets of classical logic and comes equipped with automated theorem-proving mechanisms, this approach allowed the development of inference

<sup>3</sup> This example is excerpted from *Artificial Intelligence*, by Patrick Winston, Addison-Wesley, 1984.

systems whose soundness and completeness could be rigorously demonstrated. But while many of these early inference systems were logically sound and complete, they were often not very useful, as they could only be applied to highly proscribed areas or “toy problems.” The problem was that a complete first-order predicate logic is itself computationally intractable, as certain statements may prove *undecidable*.

Suppose for example that we are trying to establish that some theorem,  $P(x)$ , is true. The way a theorem prover works is to first negate the theorem and, subsequently, to combine the negated theorem ( $\neg P(x)$ ) with stored axioms in the body of knowledge to show that this leads to a logical contradiction. Ultimately, when the theorem prover derives the conclusion  $P(x) \wedge \neg P(x)$ , the program terminates and the theorem is considered proven.

This method of proof by refutation is guaranteed to terminate when  $P(x)$  is indeed upheld by the body of knowledge. The problems arise when the initial theorem is not valid, as its negation may not produce a logical contradiction, and thus the program may not terminate.

In contrast, the frame representations offered a rich, intuitive means of expressing domain knowledge, yet they lacked the inference mechanisms and rigor that predicate logic systems could provide. As suggested by Figure 4.2-1, the frame representation captures a good deal of implicit knowledge. For example, we expect that all disaster events, including earthquakes, have information about fatalities and injuries and the extent of loss and property damage. In addition, we expect that these events will have locations, dates, and individuals associated with them.

Early efforts to apply predicate logics to frame representations in order to make this information explicit however, soon revealed that the problem was computationally intractable. This occurred for two reasons: (1) The frame representation was too permissive; more rigorous definitions were required to make the representation computational; and (2) the intractability of first-order predicate logic itself.

Several subsets of complete FOL have since been defined and successfully applied to develop useful computational models capable of significant reasoning. For example, the Prolog programming language is based on a subset of FOL that severely limits the use of negation. The family of description logic (DL) systems is a more recent development, and one that is especially well-suited to the development of ontologies, taxonomies, and controlled vocabularies, as an important function of a DL is as an auto-classifier.

#### **4.2.1 Description Logic**

Description logic can be viewed as a combination of the frame-based approach with FOL. In the process, both models had to be scaled back to achieve an effective solution. Like frames, the DL representation allows for concepts and relationships among concepts, including simple taxonomic relations as well as other meaningful types of association. Certain restrictions however, are placed on these relations. In particular, any relation that involves class membership, such as the *isa* or *inverse-isa* relations, must be strictly acyclic.

The predicate logic used in a description logic system is also limited in various ways, depending on the implementation. For example, the most minimal form of a DL does not allow any form of existential quantification. This limitation allows for a very easily computed solution space, but the resulting expressivity is severely diminished. The next step up in representational power allows limited existential quantification but without atomic negation.

Indeed, there is today a large family of description logics that have been realized, with varying levels of expressivity and resulting computational complexities. In general, DLs are decidable subsets of FOL, and the decidability is due in large part to their acyclicity. The theory behind these models is beyond the scope of this discussion, and the interested reader is referred to *The Description Logic Handbook*, by Franz Baader, et al. (eds.), Cambridge University Press, 1993, ISBN number 0-521-78176-0.

The two main ingredients of a DL representation are *concepts* and *roles*. A major distinction between description logics and other subsets of FOL is its emphasis on set notations. Thus a DL concept never corresponds to a particular entity but rather to a *set* of entities, and the notations used for logical conjunction and disjunction are set intersection and union.

DL concepts can also be thought of as unary predicates in FOL. Thus the DL expression  $Person \cap Young$  can be interpreted as the set of all children, with the corresponding FOL expression  $Person(x) \wedge Young(x)$ . Syntactically then, DL expressions are variable free, with the understanding that the concepts always reference sets of elements.

A DL *role* is used to indicate a relationship between the two sets of elements referenced by a pair of concepts. In general, DL notations are rather terse, and the concept (or set of elements) of interest is not explicitly represented. Thus, to represent the set of individuals whose children are all female, we would use:  $\forall hasChild.Female$ . The equivalent expression in FOL might be something like:

$$\forall x: hasChild(y, x) \rightarrow female(x).$$

In terms of set theory, a role potentially defines the Cartesian product of the two sets. Roles can have restrictions, however, which place limitations on the possible relations. A *value* restriction limits the type of elements that can participate in the relation; a *number* restriction limits the number of such relations an element can participate in.

In addition, each role defines a *directed* relation. For example, if  $x$  is the child of  $y$ ,  $y$  is not also the child of  $x$ . In the above example *hasChild*, the parent concept is considered the *domain* of the relation, and the child is considered the *range*. Elements belonging to the set of objects defined by the range concept are also called role fillers. Number restrictions apply to the number of role fillers that are required or allowed in a relation. For example, a parent can be defined as a person having at least one child:  $Person \cap (\geq 1 hasChild)$ .

A DL representation is constructed from a ground set of *atomic concepts* and *atomic roles*, which are simply asserted. *Defined concepts* and *defined roles* are then derived from these atomic elements, using the set operations of intersection, union, negation, etc. Most DLs also allow existential and universal quantifiers, as in the above examples. Note, however, that these quantifiers always apply to the role fillers only.

The fundamental inference operation in DL is *subsumption*, and is usually indicated with subset notation. Concept A is said to subsume B, or  $A \subseteq B$ , when all members of concept B are contained in the set of elements defined by concept A, but not vice versa. That is, if B is a proper subset of A, then A subsumes B. This capability has far-reaching repercussions for vocabulary and ontology developers, as it enables the system to automatically classify newly introduced concepts. Moreover, correct subsumption inferencing can be highly nontrivial, as, in general, this

requires examining all of the relationships defined in the system and the concepts that participate in those relations.

### 4.3 Description Logic in the NCI Thesaurus

The NCI Thesaurus is currently developed using the proprietary Apelon Inc. Ontylog™ implementation of description logic. Ontylog is distributed as a suite of tools for terminology development, management, and publishing. Although the underlying inference engine of Ontylog is not exposed, the implementation has the characteristics of what is called an AL<sup>-</sup> (attributive language) or FL<sup>-</sup> (“Frame Language”) description logic. It does not support atomic negation but does appear to provide all other basic description logic functionality.

The NCI Thesaurus is edited and maintained in the Terminology Development Environment (TDE) provided by Apelon. The TDE is an XML-based system that implements the DL model of description logic based on Apelon’s Ontylog Data Model. The Data Model uses four basic components: *Concepts*, *Kinds*, *Properties*, and *Roles*.

As in other DL systems, Concepts correspond to nodes in an acyclic graph, and Roles correspond to directed edges defining relations between concept members. Each Concept has a unique Kind. Formally, Kinds are disjoint sets of Concepts and represent major subdivisions in the NCI Thesaurus.

More concretely, Kinds are used in the Role definitions to constrain the *domain* and *range* values for that Role. Each Role is a *directed* relation that defines a triple consisting of two concepts and the way in which they are related. The domain defines the concept that the role applies to, and the range defines the possible values—in other words, concepts, that can fill that role. For example, the Role *geneEncodes* might have its domain restricted to the *Gene\_Kind* and its range to the *Protein\_Kind*. This Role then, essentially states that *Genes* encode *Proteins*.

As in all DLs, all roles are passed from parent to child in the inheritance hierarchy. For example, a “Malignant Breast Neoplasm” has the role *located-in*, connecting it to the concept “Breast.” Thus, since the concept “Breast Ductal Carcinoma” *is-a* “Malignant Breast Neoplasm,” it inherits the *located\_in* relation to the “Breast” concept. These lateral nonhierarchical relations among concepts are referred to as associative or semantic roles — in contrast to the hierarchical relations that reflect the *is-a* roles.

In the first-order algebra upon which Ontylog DL is based, every defined relationship also has a defined inverse relation. For example, if *A* is contained by *B*, then *B* contains *A*. Inverse relationships are useful and are expected by human users of ontologies. However, they have a computational cost. If the edges connecting concept nodes are bi-directional, then the computation quickly becomes intractable. Therefore in the Ontylog implementation of DL, inverse relationships are not stored explicitly but computed on demand.

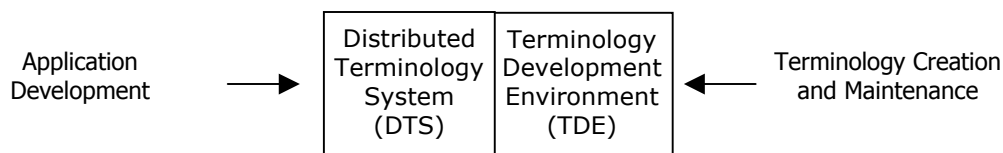


Figure 4.3-1. An overview of the NCI Thesaurus infrastructure

Figure 4.3-1 gives an overview of how the NCI Thesaurus is deployed. Apelon provides both graphical and programmatic interfaces to its Distributed Terminology System and its Terminology Development Environment.

The graphical interfaces to the DTS are available for browsing at the [EVS web sites](#); the APIs however are proprietary, and thus not available to the public. The domain objects described in this chapter have been implemented to provide a public API to the DTS, including the NCI-specific extensions to the DTS which support functionality such as concept history.

#### 4.4 Concept Edit History in the NCI Thesaurus

One of the primary uses of the NCI Thesaurus is as a resource for defining tags or retrieval keys for the curation of information artifacts in various NCI repositories. Since these tags are defined at a fixed point in time, however, they necessarily reflect the content and structure of the Thesaurus at that time only. Given the rapidly evolving terminologies associated with cancer research, there is no guarantee that the tags used at the time of curation in the repository will still have the same definition in subsequent releases of the Thesaurus. In most cases the deprecation or redefinition of a previously defined tag is not disastrous, but it may compromise the completeness of the information that can be retrieved.

In order to address this issue, the EVS team has developed a *history* mechanism for tracing the evolution of concepts as they are created, merged, modified, split, or retired. (In the NCI Thesaurus, no concept is ever deleted.) The basic idea is that each time an edit action is performed on a concept, a record is added to a history table. This record contains information about relations that held for that concept at the time of the action as well as other information, such as version number and timestamp that can be used to reconstruct the state when the action was taken. Table 4.4-1 summarizes the information stored in the history table.

The Reference\_Code column captures critical information concerning the impact of the edit actions on other concepts. This field contains the concept code of a second concept either participating in or affected by the editor's action. The value will always be null if the action is Create or Modify.

**Table 4.4-1 The NCI Thesaurus concept history table**

Column Name	Description
History_ID	Unique consecutive number for use as the database primary key
Concept_Code	The concept code for the concept currently being edited
Action	Edit Action: {Create, Modify, Split, Merge, Retire}
Baseline_Date	Date of NCI Thesaurus Baseline (see discussion below)
Reference_Code	The concept code for a second concept impacted by the action

Capturing the history data for a Split, Merge, or Retire action is more complicated. In a Split, a concept is redefined by partitioning its defining attributes between two concepts, one of which retains the original concept's code and one that is newly created. This action is taken when ambiguities in the original concept's meaning require clarification by narrowing its definition.

In the case of a Split, three history records will be created: one for the newly created concept, (with a null Reference\_Code), and two for the original concept that is being split. In the first of these two records, the Reference\_Code is the code for the new concept; in the second it is the code of the split concept.

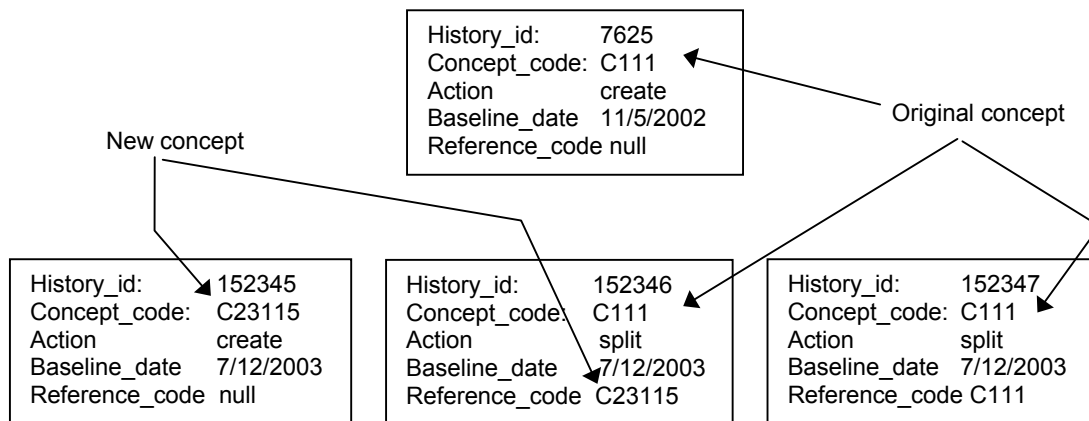


Figure 4.4-1 History records for the split action

For Merge actions, the situation is similar to a Split. In this case, two ambiguous concepts must be combined, and only one of the original concepts is retained. Again there will be three history records created: two for the concept that will be retired during the merge, and one for the "winning" concept. The Reference\_Code in the history record for the "winning" concept will be the same as the Concept\_Code; i.e., the concept points to itself as a descendant in the Merge action. The Reference\_Code will be null in one of the entries for the retiring concept, while the second entry will have the code of the "winning" concept; thus, this Reference column points to the concept into which the concept in the Concept\_Code column is being merged.

Finally, if the action is Retire, there will be as many history entries as the concept has parent concepts. The Reference column in these entries will contain the concept code of the parent concepts, one parent concept per history entry. The motivation for this is that end-users with documents coded by such retired concepts may find a suitable replacement among the concept's parents at the time of retirement.

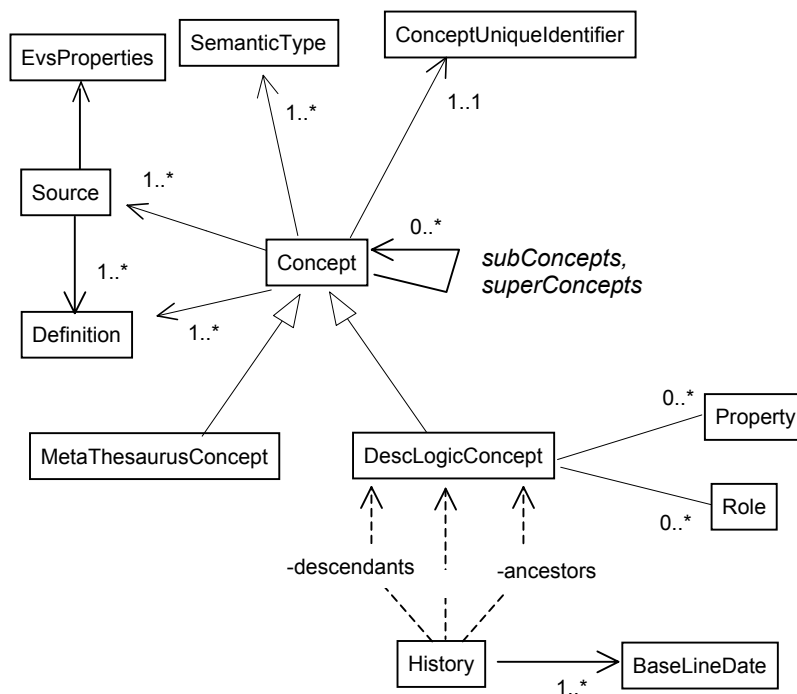
The new EVS APIs support concept history queries, and for programmatic consistency, a minimal history is added to all vocabularies served from the DTS that are *not* edited by the EVS group. Concepts in vocabularies that are not edited by EVS will have a single history entry associated with them—a *Create* action with date "May 1, 2003."

In the case of the NCI Thesaurus, concept history tracking has been ongoing internally since December 2002. However, for the purpose of publication in the DTS, a specific baseline has been selected to serve as "time zero" for concept history. This baseline is (internal) version 03.08c, which immediately preceded the NCI Thesaurus Version 2.0 released in caCORE 2.0. All of the concepts in this baseline have a *Create* action associated with them, dated "August 12, 2003", the date of the 03.08c build.

#### 4.5 The caBIO EVS API

The UML Class diagram in Figure 4.5-1 provides an overview of the caBIO EVS domain object classes, with the class attributes and operators suppressed to emphasize the general

organization. The central class is *Concept*, with most of the other classes organizing themselves around this entity.



**Figure 4.5-1 The caBIO EVS API domain object classes**

The *Concept* class is an abstract class and contains several private attributes, including:

- *name*: a string representation of the name for the concept;
- *definition*: the authoritative definition for the concept;
- *semanticTypes*: a list of *SemanticTypes* for the concept (see discussion below);
- *sources*: a list of vocabulary *Sources* containing that concept;
- *synonyms*: a list of string aliases for the concept; and
- *CUI*: the *ConceptUniqueIdentifier*.

All of these attributes are accessible via the public get methods listed in the EVS Javadocs. Four of these attributes—*definition*, *semanticTypes*, *sources*, and *CUI*, reference other class objects as depicted in the diagram.

The *SemanticType* is a simple class whose attributes include *name* and *id*; some example names for semantic types are: “Body Location or Region,” “Research Device,” “Cell Function,” etc. These names are the semantic types encoded for the associated concept in the UMLS Metathesaurus (see [Section 4.1](#)). All concepts in the UMLS Metathesaurus are associated with one or more semantic types, which broadly indicate the kind of information embodied by the concept. The semantic types and their relations are used to characterize where the concept occurs in various vocabularies and what relations may hold for that concept.

All concepts in the NCI Metathesaurus have at least one source, and some concepts may have several. In contrast, each concept in the NCI Thesaurus has exactly one source. In addition to

having a name attribute, a *Source* object may have a definition, as well as a description and an abbreviation or acronym associated with it.

For NCI Metathesaurus concepts, the *CUI*, or *ConceptUniqueIdentifier*, is either the UMLS id for the concept, or the local id. In either case, the *CUI* is a string that uniquely identifies the concept. All concept identifiers begin with the letter *C*; those whose second character is also a letter are local identifiers used at NCI.

Other relations to *Concept* in the Class diagram derive from the applicable operations for the class, such as *getSubConcepts()* and *getSuperConcepts()*. These methods return *Concept* objects that are specializations and generalizations, respectively, of the current concept. Like the caBIO domain objects, the *Concept* class implements the *XMLInterface* described in the previous section, and is associated with a *SearchCriteria* object called *ConceptSearchCriteria*.

Two subordinate classes in Figure 4.5-1 are derived from *Concept*: *DescLogicConcept* and *MetathesaurusConcept*. Like the *Concept* class, each of these derived classes implements *XMLInterface* and has an associated search criteria object specialized for it.

The *MetaThesaurusConcept* class inherits most of its attributes from the parent *Concept* class, adding just one additional attribute: *preferredName*. In most cases the value of this attribute is the same as *name*. The *DescLogicConcept* class adds several new attributes to those inherited from the *Concept* class. The *properties* and *roles* attributes define links to *Property* and *Role* objects. Both of these constructs serve to capture the DL encoding of the NCI Thesaurus concepts. Two additional attributes, *creationDate* and *retired* provide access to history and status information for the concept.

The *History* class is defined independently of the *DescLogicConcept* class, but makes reference to instances of these elements via their associated concept codes. Figure 4.5-1 reflects this indirect referencing with dashed lines.

The EVS API diverges somewhat from the other caCORE domain models in its search mechanisms, as described in the next section. While the other APIs have direct access to their databases, the EVS API does not. Since all EVS queries are passed through the proprietary APIs provided by Apelon, the search and retrieval capabilities are effectively proscribed by the features implemented by these third-party tools.

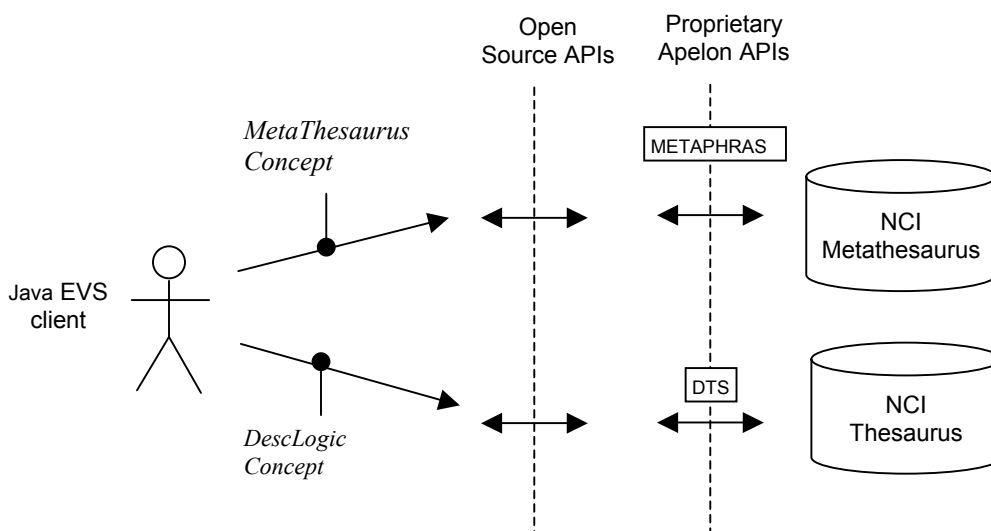
## 4.6 The EVS Search Paradigm

Figure 4.6-1 depicts the configuration used in accessing the EVS vocabularies. As indicated there, each of the two servers has a somewhat different interface. The Metaphrase interface, which serves the NCI Metathesaurus users, provides access to the complete concept in a single query. The DTS interface however, which serves the NCI Thesaurus users, provides access to only one feature per query. Thus, in order to retrieve an NCI Thesaurus concept—with all of its attributes fully populated—a large number of atomic queries would need to be issued, with each requiring its own database lookup

Java clients accessing the NCI Thesaurus and Metathesaurus vocabularies using the caCORE 2.0 domain objects communicate their requests to these proprietary APIs via the open source



APIs, which are included in the jar files of the caCORE 2.0 distribution.<sup>4</sup> The two data sources can be accessed via the *DescLogicConcept* and *MetaThesaurusConcept* objects. Both classes inherit from the *Concept* class, as described in the previous section.



**Figure 4.6-1 Middleware Interfaces to the EVS Databases**

Working closely with the EVS user community, a decision was made to avoid the overhead of issuing multiple queries for the retrieval of a single description logic concept. The design solution uses a kind of “lazy evaluation” in its interface to the NCI Thesaurus: attributes of the retrieved description logic concepts are not populated until they are explicitly accessed.

As a result of this design decision, the *DescLogicConcept search()* method returns a lightweight object, where only a few of the object’s attributes are populated. While all of the other attributes are still defined, they are valueless, and any method which accesses them will return the *null* value. This information is of course still available, but must be accessed via explicit method calls to the DTS interface. Users who wish to retrieve a “heavyweight object” for a description logic concept—i.e., one whose attributes are fully populated—must instead use the convenience method *getConceptByName()*.

In contrast, the Metathesaurus interface does not require this design, as a single query to the proprietary interface returns the complete concept. These heavyweight objects can be obtained directly via the *MetaThesaurusConcept*’s generic *search()* method.

In both cases, the heavyweight objects provide direct access to all of the concepts’ attributes. The basic workflow in the EVS API is to define various search parameters, such as the name of the vocabulary to be searched, the search term to which concept names should be matched, and the number of results to be returned, and then call the appropriate methods for executing the search using either a *DescLogicConcept* or *MetaThesaurusConcept*. The following sub-sections provide examples of this workflow.

<sup>4</sup> The open sources interfaces are included only in the jar file for the full installation of the caBIO server – they are not in the distribution for the client installation

## 4.6.1 Description Logic Concepts

Similar to other caBIO domain objects, the *DescLogicConcept* class has a generic *search()* method which takes a *DescLogicConceptSearchCriteria* as its sole argument. Two attributes must be defined for the search criteria object before it can be used—the *searchTerm* and the *vocabularyName*:

```
//Instantiate DescLogicSearchCriteria
DescLogicConceptSearchCriteria dlscsc = new DescLogicConceptSearchCriteria();

//Instantiate DescLogicConcept
DescLogicConcept dlc = new DescLogicConcept();

//Set input parameters
dlscsc.setSearchTerm("gen*"); //mandatory
dlscsc.setVocabularyName("NCI_Thesaurus"); //mandatory
dlscsc.setLimit(10); //optional default is 100

//Call the method in DescLogicConcept passing dlscsc
Concept[] conceptArray = dlc.search(dlscsc);
```

The *searchTerm* can be generalized to include wildcard matching by inserting an asterisk where appropriate. In the above example, the search will retrieve all concepts in the NCI Thesaurus whose concept name begins with the string “gen”. If no asterisks occur in the search term then only exact matches will be retrieved.

The call to the Concept object’s *search()* method returns an array of lightweight *Concept* objects whose names contain the specified search term. The only attribute that these *Concept* objects will have populated in this case is *name*. Once the name is obtained however, it becomes possible to use this to invoke other methods, such as *getConceptByName()*. Alternatively, if the user already knows the name, this method can be invoked directly as in:

```
DescLogicConceptSearchCriteria dlscsc = new DescLogicConceptSearchCriteria ();
DescLogicConcept dlc = new DescLogicConcept ();
dlscsc.setSearchTerm("gene");
dlscsc.setVocabularyName("NCI_Thesaurus");
dlc = dlscsc.getConceptByName(dlscsc);
```

The *DescLogicConcept* returned in this case is now a fully-populated heavyweight object, providing direct access to all of the attributes for the associated NCI Thesaurus concept.

Given the exact concept name, several convenience methods also become available, including *getConceptCodeByName()*, *getPropertiesByConceptName()*, and *getRolesByConceptName()*. In the following example, an “empty” *DescLogicConcept* – one that has been instantiated but is not associated with any specific concept – is used to anonymously invoke these methods:

```
DescLogicConceptSearchCriteria dlscsc = new DescLogicConceptSearchCriteria ();
DescLogicConcept dlc = new DescLogicConcept ();
dlscsc.setSearchTerm("gene");
dlscsc.setVocabularyName("NCI_Thesaurus");
String conceptCode = dlc.getConceptCodeByName(dlscsc);
Property[] properties = dlc.getPropertiesByConceptName("gene");
Role[] roles = dlc.getRolesByConceptName("gene");
```

Referring back to our earlier description of the DTS interface to the NCI thesaurus, we see that each of the above invocations corresponds to a single query. In contrast, the heavyweight method *getConceptByName()*, which returns a fully-populated object, entails a sequence of queries. The *DescLogicConcept* class also defines parameter-free methods which are applicable to such heavyweight objects, such as *getProperties()*.

*Property* and *Role* are simple classes used to hold the tag-value pairs associated with a description logic concept. One such example property is the concept's preferred name. The associated *Property* object for the Gene concept would return "Preferred\_Name" for the *getName()* method and "Gene" for the *getValue()* method. Similarly, the *Role* objects implement *getName()* and *getValue()* methods. As described in [Section 4.2.1](#), roles define the non-hierarchical relations that hold among concepts.

Another convenience method that requires only the vocabulary name and concept code is the *isRetired()* method, which returns true when the concept is no longer in use:

```
DescLogicConceptSearchCriteria dlsc = new DescLogicConceptSearchCriteria ();
DescLogicConcept dlc = new DescLogicConcept ();
dlsc.setSearchTerm("gene");
dlsc.setVocabularyName("NCI_Thesaurus");
String conceptCode = dlc.getConceptCodeByName(dlsc);
boolean retired = dlc.isRetired("NCI_Thesaurus", "gene");
```

The use of concept editing history in the EVS was described in [Section 4.4](#); access to this information is provided via the *History* object, and specifically, through the methods which access the concepts' ancestors and descendants. The ancestor and descendant relations associated with *History* objects provide information concerning how a concept has evolved over time.

Note that ancestors and descendants do *not* refer to the relations between nodes occurring in the Thesaurus's tree-structured concept hierarchy. Parent/child relations in the concept hierarchy are accessed as superconcept/subconcept relations. Thus, while a superconcept refers to a second more general concept, an *ancestor* captures information about that *same* concept as it was published in a previous release of the NCI Thesaurus

A *History* object's methods can be invoked directly without involving any intermediate search criteria or concept objects. The key convenience method in the *History* object is the *getDescendants()* method. This method provides access to the descendants of a concept code or concept name and can be invoked as follows:

```
History hist = new History();
String[] descendantsArray =
    = hist.getDescendants("NCI_Thesaurus", "Gene", true, "1/1/2000", "1/1/2003");
```

The input parameters used above in the *getDescendants()* method are: the vocabulary name, the concept name (or code), a flag indicating how the search should be performed, and the starting and ending dates for the NCI Thesaurus releases. In this case, the flag's value of `true` indicates that only active descendant concepts should be returned. The *descendantsArray* returns a *String* array of all the descendant concept names of "Gene" between the timelines "1/1/2000" and "1/1/2003".

## 4.6.2 MetaThesaurus Concepts

The *MetaThesaurusConcept* class provides access to the NCI Metathesaurus data source. The procedure for calling a method is similar to that of a *DescLogicConcept*. But in this case, the generic *search()* method returns a fully-populated heavyweight object:

```
//Instantiate SearchCriteria
MetaThesaurusConceptSearchCriteria mtcsc = new
    MetaThesaurusConceptSearchCriteria();
```

```
//Instantiate Concept
MetaThesaurusConcept mtc = new MetaThesaurusConcept();

//Set input parameters
mtcsc.setSearchTerm("gen*"); // mandatory
mtcsc.setLimit(10); // optional default is 100

//Call the method in MetaThesaurusConcept passing mtcsc
Concept[] conceptArray = mtc.search(mtcsc);
```

Unlike the DTS server which provides access to several vocabularies including the NCI Thesaurus, the Metaphrase server provides access to the NCI Metathesaurus only. Thus it is not necessary to define a vocabulary name. Also, since the server returns a complete concept on a single query, each *Concept* in the resulting *conceptArray* is a fully-populated heavyweight object. These *Concept* objects provide direct access to information such as the *ConceptUniqueIdentifier*, *Definition*, *Source* and *SemanticType*. From the above result each object can be retrieved as:

```
//Retrieve each Concept object
Concept conceptObject = (Concept)ConceptArray[0];

//Get Source array
Source[] sourceArray = conceptObj.getSources();

//Get Concept unique identifier
ConceptUniqueIdentifier conceptUID =
    conceptObj.getConceptUniqueIdentifier();

//Get Definitions
Definition[] definitionArray = conceptObj.getDefinitions();

//Get Source of a each definition from the above definitionArray
Source defintionSource = definitionArray[0].getDefinition();
```

Sample code demonstrating how to use each method in the EVS Java API is included in both Appendix B of this guide as well as in the caCORE 2.0 distribution package. For non-Java developers, a SOAP API is also available. The details on using this interface are discussed in [Section 8.6](#) and in [Section 13.4](#).

## 4.7 The EVS Domain Object Catalog

This catalog lists the objects defined in the *gov.nih.nci.evs.bean* package; the fields listed here should be interpreted as follows:

- *Application*: Indicates in what contexts the object is most likely to be used.
- *Related domain objects*: A second domain object is “related” to the first domain object if that second object occurs anywhere in the signature of a method for the first object.
- *Extends*: Reflects direct inheritance; i.e., the current class is a direct subclass of that which it extends.

In cases where no applications, relations, or extensions apply, those fields are omitted. The only interface implemented by the EVS domain objects is *java.io.Serializable*.

### 4.7.1 [BaseLineDate](#)

Reflects the release date of the NCI Thesaurus when this definition went public.

*Application:* defined and used by the EVS project for maintaining historical context information.  
*Related domain objects:* [DescLogicConcept](#)  
*Extends:* java.lang.Object

#### **4.7.2 [Concept](#)**

Some “thing” in the knowledge domain, such as a disease, an organ, or a location. More formally, describes the properties of a collection of individuals and can be interpreted as a unary predicate.

*Application:* defined and used by the EVS.  
*Related domain objects:* [DescLogicConcept](#), [MetathesaurusConcept](#), [SemanticType](#)  
*Extends:* java.lang.Object

#### **4.7.3 [ConceptUniqueIdentifier](#)**

An alphanumeric string associated with a concept that uniquely identifies the concept.

*Application:* defined and used by the UMLS and by EVS.  
*Extends:* java.lang.Object

#### **4.7.4 [Definition](#)**

An authoritative definition of a concept.

*Application:* defined and used by the EVS.  
*Extends:* java.lang.Object

#### **4.7.5 [DescLogicConcept](#)**

A description logic concept represented in the NCI Thesaurus.

*Application:* defined and used by the EVS.  
*Related domain objects:* [Concept](#), [History](#), [SemanticType](#)  
*Extends:* Concept

#### **4.7.6 [EvsProperties](#)**

An object used to store connection properties (host, port) for an EVS session.

*Application:* defined and used by the EVS.  
*Related domain objects:* [DescLogicConcept](#), [MetathesaurusConcept](#), [SemanticType](#)  
*Extends:* java.lang.Object

#### **4.7.7 [History](#)**

A history record associated with a description logic concept in the NCI Thesaurus; used to track concept identities and definitions over multiple releases.

*Application:* defined and used by the EVS.  
*Related domain objects:* [DescLogicConcept](#), [BaseLineDate](#)  
*Extends:* java.lang.Object

#### **4.7.8 [MetaThesaurusConcept](#)**

A concept in the NCI Metathesaurus.

*Application:* defined and used by the EVS.

*Related domain objects:* [Concept](#), [SemanticType](#)

*Extends:* Concept

#### **4.7.9 [Property](#)**

A feature or attribute of a description logic concept.

*Application:* defined and used by the EVS.

*Related domain objects:* [DescLogicConcept](#)

*Extends:* java.lang.Object

#### **4.7.10 [Role](#)**

A relationship defined between two description logic concepts in the NCI Thesaurus.

*Application:* defined and used by the EVS.

*Related domain objects:* [DescLogicConcept](#), [MetathesaurusConcept](#), [SemanticType](#).

*Extends:* java.lang.Object

#### **4.7.11 [SemanticType](#)**

A categorization used to broadly characterize the type of information represented by a concept.

*Application:* defined and used by the EVS and UMLS.

*Related domain objects:* [MetathesaurusConcept](#)

*Extends:* java.lang.Object

#### **4.7.12 [Source](#)**

The named source for a concept definition.

*Application:* defined and used by the EVS.

*Related domain objects:* [EVSProperties](#), [Definition](#).

*Extends:* java.lang.Object

### **4.8 Downloading the NCI Thesaurus**

The NCI Thesaurus is can be downloaded in several formats, including simple tab-delimited ASCII format, Apelon's proprietary Ontylog XML format, and [OWL](#) format (the Web Ontology Language). The ASCII- and XML-formatted files are available for download at the [NCICB download](#) site, as ThesaurusV2\_0Flat.zip and ThesaurusV2\_0XML.zip. The OWL formatted version is available at <http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl>. Users who prefer to use FTP for download can go to the [caCORE FTP site](#).

The format of the ASCII flat file is extremely simple. For each concept, the download file includes the following information:

1. The concept code: all terms have the “C” prefix, followed by its integer index;
2. The concept name: this name may contain embedded punctuation and spaces;
3. A pipe-delimited list of parent concepts, as identified in the NCI Thesaurus by *isa* relations;
4. A pipe-delimited list of synonyms, the first of which is the preferred name; and
5. One of the NCI definitions for the term—if one exists.

Each of these separate types of information is tab-delimited; within a given category, the individual entries are separated by pipes (“|”). Only the third and fourth categories, i.e., the parent concepts and synonyms, have multiple entries requiring the pipe separators. Note that while much of the information available from the interactive Metaphrase server is included in the download, any information outside the NCI Thesaurus description logic vocabulary (e.g., Diagnosis, Laboratory, Procedures, etc.) is not.

For example, the flat file download for the term “*Mercaptopurine*” is as follows:

```
C6 Mercaptopurine Immunosuppressants|Purine Antagonists
Mercaptopurine|1,3-AZP|1,7-Dihydro-6H-purine-6-thione|3H-Purine-6-thiol|6
Thiohypoxanthine|6 Thiopurine|6-MP|6-Mercaptopurine|6-Mercaptopurine
Monohydrate|6-Purinethiol|6-Thiopurine|6-Thioxopurine|6H-Purine-6-thione,
1,7-dihydro- (9CI)|6MP|7-Mercapto-1,3,4,6-tetrazaindene|AZA|Alti-
Mercaptopurine|Azathiopurine|BW 57-323H|CAS
50442|Flocofil|Ismipur|Leukerin|Leupurin|MP|Mercaleukim|Mercaleukin|Mercap|Me
rcaptina|Mercapto-6-purine|Mercaptopurinum|Mercapurin|Mern|NCI-C04886|NSC
755|Puri-Nethol|Purimethol|Purine-6-thiol (8CI)|Purine-6-thiol
Monohydrate|Purine-6-thiol, Monohydrate|Purinethiol|Purinethol|U-4748|WR-2785
An anticancer drug that belongs to the family of drugs called
antimetabolites.
```

Users who have access to the Apelon Ontylog software may wish to download the XML encoded file. All other users who prefer to use an encoded format rather than the simple ASCII form should download the OWL encoding of the NCI Thesaurus, which is described below.

#### 4.8.1 The OWL Encoding of the NCI Thesaurus

[OWL](#), as specified and proposed by the World Wide Web Consortium ([W3C](#)), is an emerging standard for the representation of semantic content on the web. Building on the earlier groundwork laid by XML, the Resource Description Framework (RDF) and RDF schema; and subsequently, by DAML+OIL, OWL represents the culmination of what has been learned from these previous efforts.

While XML provides surface syntax rules and XML Schema provides methods for validating a document's structure, neither of these can in itself impose semantic constraints on how a document is interpreted. RDF provides a data model for specifying objects (resources) and their relations, and RDF Schema allows one to associate properties with the individual resources as well as taxonomic relations among the objects. Yet even these extensions could not provide the breadth and depth of representation needed to encode nontrivial real-world information. OWL adds vocabulary for describing arbitrary nonhierarchical relations between classes, cardinality constraints, resource equivalences, richer typing of properties, and enumerated classes.

A major focus of the W3C is the establishment of the [The Semantic Web](#)—a far-reaching infrastructure whose purpose is to provide a framework whereby autonomous self-documenting agents and web services can exchange meaningful information without human intervention. OWL is the first step towards realizing this vision. As a result of collaborative efforts with Dr. James Hendler and the University of Maryland, the NCI Thesaurus is now available for download in OWL format; this section describes the mapping of the NCI Thesaurus to OWL.

The mapping of the NCI Thesaurus into OWL format proceeds via the Ontylog XML elements declared in Apelon's Ontylog DTD. The four basic elements are *Kinds*, *Concepts*, *Roles*, and *Properties*, where:

- Kinds are the top-level super classes in the Thesaurus; they enumerate the different possible categories of all concepts, and include such things as Anatomy, Biological Processes, Chemicals and Drugs, etc. *Each NCI Thesaurus Kind is converted to an owl:Class.*
- An NCI Thesaurus Concept describes a specific concept under one of the Kind categories. *Each NCI Thesaurus Concept is converted to an owl:Class.*
- Roles capture how concepts relate to one another. Generally, Roles have restricted domains and ranges, that limit the sets of concepts which can participate in the Role according to their categories—i.e., Kinds. The "defining roles" within a concept definition provide these local restrictions on the ranges of roles. *Each NCI Thesaurus Role is converted to an owl:ObjectProperty.*
- NCI Thesaurus Properties encode the attributes that pertain to a class; they contain metadata that describes the class, but not its instantiations or subclasses. *Each NCI Thesaurus Property is converted to an owl:AnnotationProperty.*

The bulk of the Thesaurus comprises concept definitions; this is also where the most complex semantics occur. Each concept in the Thesaurus has three main types of associated data: defining concepts, defining roles, and properties. A "defining concept" is essentially a super class; the defined concept in OWL has an *rdfs:subClassOf* relationship to the defining concept.

The defining roles and properties are mapped as described above; the *owl:AnnotationProperty* is actually a subclass of *rdf:Property*, and, like *rdfs:comment* and *rdfs:label*, can be attached to any class, property or instance. This allows properties from the Thesaurus to be associated directly with a concept's corresponding class, without violating the rules of OWL.

In addition to any explicitly named properties, each element in the Thesaurus also has a uniquely defined "code" and "id" attribute associated with it. These are used as unique identifiers in the Apelon development software, and, as such, are not defined explicitly as roles or properties. In mapping these identifying attributes to OWL, we have treated these as special cases of the explicit property elements, and just like other properties in the Thesaurus, they are mapped as *owl:AnnotationProperties*. Table 4.7-1 summarizes the mapping of elements in the Ontylog DTD to OWL elements.

### Ontylog Name Conversion

In mapping to OWL, all Ontylog concept *names* must be converted to proper RDF identifiers (*rdf:id*) following the RDF naming rules. This is achieved by removing any spaces in the original names and substituting all illegal characters with underscores. Names that begin with numbers are also prefixed with underscores to make them legal. The original concept name however, is preserved as an *rdfs:label*. The following steps summarize the conversion of names:

1. Any "+" characters are replaced with the text "plus."
2. All role names are prefixed with an "r" to ensure that roles and properties with the same name do not clash.
3. Any characters that are not alphanumeric, or one of "-" and "\_," are replaced with an underscore ("\_").
4. All names with leading digits are prefixed with an underscore.



5. Multiple adjacent underscores in the corrected name are replaced with a single underscore.

Table 4.8-1 Ontylog DTD to OWL conversions

Ontylog Element	Owl Element	Comment
kindDef	owl:Class	
roleDef	owl:ObjectProperty	
propertyDef	owl:AnnotationProperty	
conceptDef	owl:Class	
name*	rdf:ID	Applies to the name subelement of <i>kindDef</i> , <i>roleDef</i> , <i>propertyDef</i> , and <i>conceptDef</i> .*
name	rdfs:label	Because the <i>conceptDef</i> <i>name</i> contains some useful semantics, the original form is retained as an <i>rdfs:label</i> . No other name elements are retained in <i>rdfs:label</i> .
code	owl:AnnotationProperty	Defined as an <i>owl:AnnotationProperty</i> with <i>rdf:ID</i> ="code". Code values remain the same for each concept.
id	owl:AnnotationProperty	Defined as an <i>owl:AnnotationProperty</i> with <i>rdf:ID</i> ="ID". ID values remain the same for each concept.
definingConcepts	rdfs:subClassOf	The <i>concept</i> subelement of <i>definingConcepts</i> is mapped to the <i>rdf:resource</i> attribute of the <i>rdfs:subClassOf</i> element.
domain	rdfs:domain	
range	rdfs:range	
definingRoles / role / name	owl:onProperty	<i>definingRoles</i> are converted to owl restrictions on properties. The <i>name</i> child element of <i>definingRoles/role</i> is taken as the <i>rdf:resource</i> attribute of the <i>owl:onProperty</i> element.
definingRoles / role / value	owl:someValuesFrom	<i>definingRoles</i> are converted to owl restrictions on properties. The <i>value</i> child element of <i>definingRoles/role</i> is taken as the <i>rdf:resource</i> attribute of the <i>owl:someValuesFrom</i> element.

\* *name* Ontylog elements are converted to *rdf:ID* as described in the *Ontylog Name Conversion* section. *namespaceDef* and *namespace* elements are not mapped to OWL.

Additional information about the Ontylog encoding is available in the Ontylog DTD, which can be downloaded from the NCICB EVS [FTP site](#), along with the zipped ASCII flat file and the Ontylog XML encoding. The current OWL translation of the NCI Thesaurus contains over 500,000 triples and is available in zipped format from the FTP site, as well as in unzipped format at <http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl>, the [mindswap](#) web site for download or online viewing.

## 4.9 Mapping of the Gene Ontology to Ontylog

As of the caCORE 2.0 release, the NCI DTS now provides access to the Gene Ontology™ Consortium’s (GO) controlled vocabulary. The GO ontologies are widely used—most likely due to their simplicity of design and their potential for automated transfer of biological annotations, from model organisms to more complex organisms based on sequence similarities. GO comprises three independent controlled vocabularies (ontologies) encoding biological process, molecular function, and cellular components for eukaryotic genes. GO terms are connected via two relations, *is-a* and *part-of*, that define a directed acyclic graph. Although concepts in the ontologies were initially derived from only three model systems (yeast, worm, and fruitfly), the goal was to encode concepts in such a way that the information is applicable to *all* eukaryotic cells. Thus, species-specific anatomies are not represented, as this would not support a unifying reference for species-divergent nomenclatures.

Each month NCI will load the latest version of GO into a test instance of the DTS server, and, following validation in the Ontylog environment, will promote it to a production server for programmatic access by NCI applications. NCI converts GO into the Ontylog XML representation (necessary for import into the DTS server) via a stylesheet transformation followed by some post-processing to satisfy Ontylog constraints. It is NCI’s intent that the version of GO on the DTS server will not be more than a month behind the current version available from <http://www.geneontology.org>. However, it might be necessary to skip releases if unforeseen complications arise.

The tables in this section summarize the encoding of GO elements into Ontylog.

Table 4.9-1 Ontylog elements used for GO mapping

Ontylog Entity	Instance Name (and optional description)
namespaceDef	<b>GO</b>
kindDef	<b>GO_Kind</b>
RoleDef	<b>part-of:</b> This role is unused; however, the software requires that at least one role be declared.
propertyDef	<b>Preferred_Name</b>
propertyDef	<b>Synonym</b>
propertyDef	<b>DEFINITION</b>
propertyDef	<b>dbxref:</b> complex property containing two XML-marked up GO entities: “go:database_symbol,” and “go:reference,” using tags “database_symbol” and “reference,” respectively.
propertyDef	<b>part-of:</b> complex property containing two XML-marked up GO entities: “go:name” and “go:accession,” using tags “go-term” and “go-id,” respectively.

The go:name stored in Preferred\_Name is as declared in GO. However, the go:name used in the Ontylog name might have been modified during the conversion process (by appending underscores) to make the Ontylog name unique.

**Table 4.9-2 Mapping of GO term to Ontylog conceptDef**

<b>GO term element</b>	<b>conceptDef element</b>	<b>(propertyDef)</b>
go:accession	code	
go:name	name	
go:isa	definingConcepts	
go:name	property	Preferred_Name
go:synonym	property	Synonym
go:definition	property	DEFINITION
go:part-of	property	part-of
go:dbxref	property	dbxref

## **5.0 THE caDSR DOMAIN OBJECTS**

The Cancer Data Standards Repository at NCI is part of a larger effort associated with the 11179 standard defined by the ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission). The purpose of the [ISO/IEC 11179](#) is to regularize the vocabularies used in representing and annotating shared electronic data. A more complete description of the caDSR is provided in the NCICB Applications User Manual, where several web-based interface tools to the repository are also described. This chapter describes the Java API to the repository and introduces the Java classes which participate in this programmatic interface. The first two sections provide a brief review of the ISO/IEC 11179 standard and its realization as an Oracle database at NCI.

## 5.1 Modeling Metadata: The ISO/IEC 11179 Standard

Regardless of the application domain, any particular data item must have associated with it a variable name or tag, a conceptualization of what the item signifies, a value, and an intended interpretation of that value. For example, an entry on a case report form may be intended to capture the patient's place of birth, and the corresponding value may be tagged electronically as *Patient\_placeOfBirth*. But what is the intended concept? Is the data element designed to capture the country, the city, or the specific hospital where the person was born? Assuming that the intended concept is country, how is the resulting value to be represented electronically? Possible representations might include the full name of the country, a standard two- or three-letter abbreviation, a standard country code, or perhaps a specific encoding unique to the application.

Metadata is “data about data,” and refers to just this type of intentional information that must be made explicit in order to ensure that electronically exchanged data can be correctly interpreted. The purpose of the ISO/IEC standard is to define a framework and protocols for how such metadata can be specified, consistently maintained, and shared across diverse domains. The caDSR conforms to this standard; while it was developed specifically for the support of clinical trials data, usage of the caDSR is not limited to clinical applications.

The ISO/IEC 11179 standard defines a fairly complex model, and even the notion of metadata itself can be somewhat abstruse as it is a rather abstract concept. To facilitate understanding the model, this discussion uses a divide-and-conquer approach, and defines two very general types of components:

1. Information components whose purpose is to represent content; and
2. Organizational and administrative components whose purpose is to manage the repository.

This partitioning is not intrinsic to the ISO/IEC 11179, and indeed, some of the components do not neatly fit into the separate categories. Nevertheless, it provides a useful framework.

The fundamental information component in the ISO/IEC 11179 model is the *data element*, which constitutes a single unit of data considered indivisible in the context in which it is used. Another way of saying this is that a data element is the smallest unit of information that can be exchanged in a transaction between cooperating systems. More specifically, a data element is

used to convey the *value* of a selected *property* of a defined *object*<sup>5</sup>, using a particular *representation* of that value.

A critical notion in the metadata model is that any concept represented by a data element must have an explicit definition that is independent of any particular representation. In order to achieve this in the model, the ISO/IEC 11179 standard specifies the following four components:

1. A *DataElementConcept* consists of an *object* and a selected *property* of that object;
2. The *ConceptualDomain* is the set of all intended meanings for the possible values of an associated *DataElementConcept*;
3. The *ValueDomain* is a set of accepted representations for these intended meanings; and
4. A *DataElement* is a combination of a selected *DataElementConcept* and a *ValueDomain*.

A simple example should help to clarify these definitions:

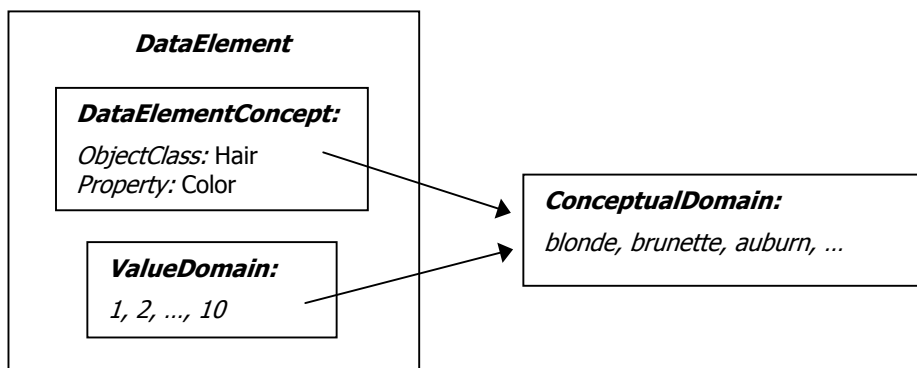


Figure 5.1-1 Representing data in the ISO/IEC 11179 model

Figure 5.1-1 shows a *DataElement* that might be used to represent hair color. The associated *DataElementConcept* uses the *ObjectClass* `Hair` and the *Property* `Color` to define the intended concept. The intended meanings for this data element are the familiar hair colors blonde, brunette, etc., but the *ValueDomain* uses a numeric representation that is mapped to these intended meanings. Both the *DataElementConcept* and the *ValueDomain* are components of the *DataElement*, and each references the same *ConceptualDomain*, which is defined outside the *DataElement*. Important principles of this design are:

- The *DataElementConcept* (DEC) is used to signify a concept *independent of representation*.
- The *ValueDomain* (VD) specifies a set of representational values *independent of meaning*.
- The *DataElement* (DE) combines a specific object and property with a value representation.
- The *ConceptualDomain* (CD) specifies the complete set of value meanings for the concept and allows the interpretation of the representation.

---

<sup>5</sup> The term *object* is used here in the sense defined by the ISO/IEC 11179 (see definition in Table 5.2-1)—and does not have any literal correspondence to a caBIO Java object.

Figure 5.1-2 uses a UML Class diagram to show the cardinality constraints that hold for these relations. Each *DataElement* must specify exactly one *DataElementConcept* and one *ValueDomain*, in order to fully specify the data element. Similarly, each *DataElementConcept* and *ValueDomain* must specify exactly one *ConceptualDomain*. Conversely, a *ConceptualDomain* may be associated with any number of *ValueDomains* and any number of *DataElementConcepts*. Figure 5.1-3 shows an example of this, using the `color` property of different geometric objects as *DataElementConcepts*, and alternate color representations for the *ValueDomains*.

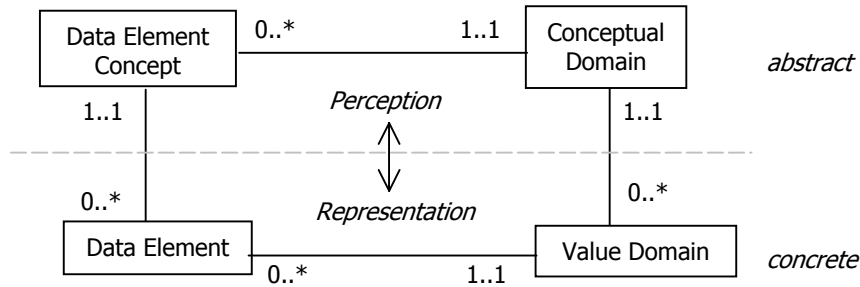


Figure 5.1-2 Abstract and concrete components of the data representation

A constraint not shown in any of these figures is that it is not possible to reuse the same *DataElementConcept-ValueDomain* pair to define a new *DataElement*, as this defines a logical redundancy. Thus, the “0..\*” cardinality constraints implied by Figure 5.1-2 are not quite as open-ended as they imply. Specifically,

- a *DataElement* specifies exactly one *DataElementConcept* and one *ValueDomain*;
- a *DataElementConcept* specifies exactly one *ConceptualDomain*;
- a *ValueDomain* specifies exactly one *ConceptualDomain*;
- a *ConceptualDomain* may be associated with any number of *ValueDomains*;
- a *ConceptualDomain* may be associated with any number of *DataElementConcepts*;
- a *DataElementConcept* may be associated with as many *DataElements* as there are *ValueDomains* (i.e. alternate representations) associated with the *ConceptualDomain*; and
- a *ValueDomain* may be associated with as many *DataElements* as there are *DataElementConcepts* associated with the *ConceptualDomain*.

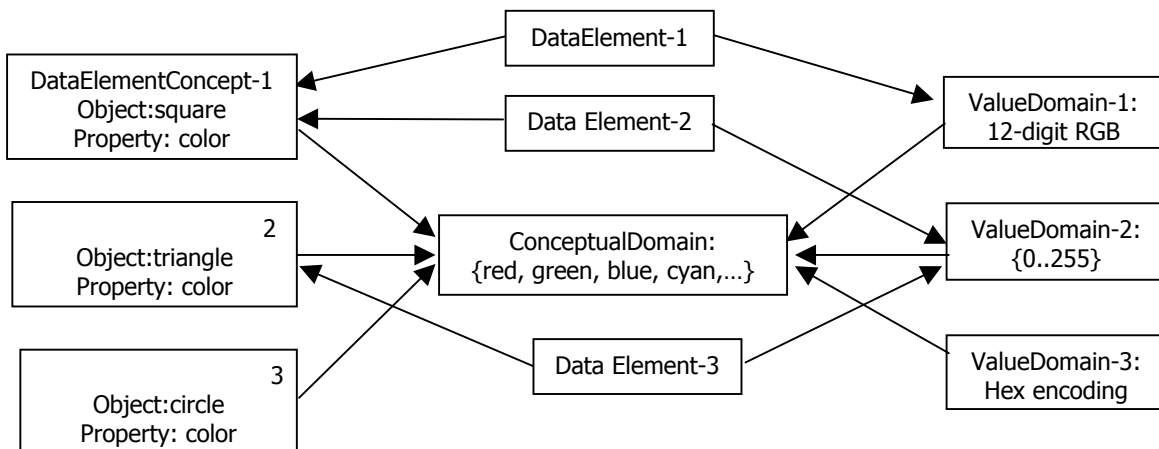


Figure 5.1-3 Many-to-one mappings of information elements in the metadata model

Many additional information components collaborate with these four core elements to provide the ISO/IEC 11179 infrastructure for content representation. These are described in the caDSR model in the next section, along with the organizational and administrative components that are used to document, classify, and in general, manage the information components.

## 5.2 The caDSR Metamodel

Figure 5.2-1 again shows the four elements discussed thus far, but this time in the context of other components that collectively define the infrastructure for content representation.

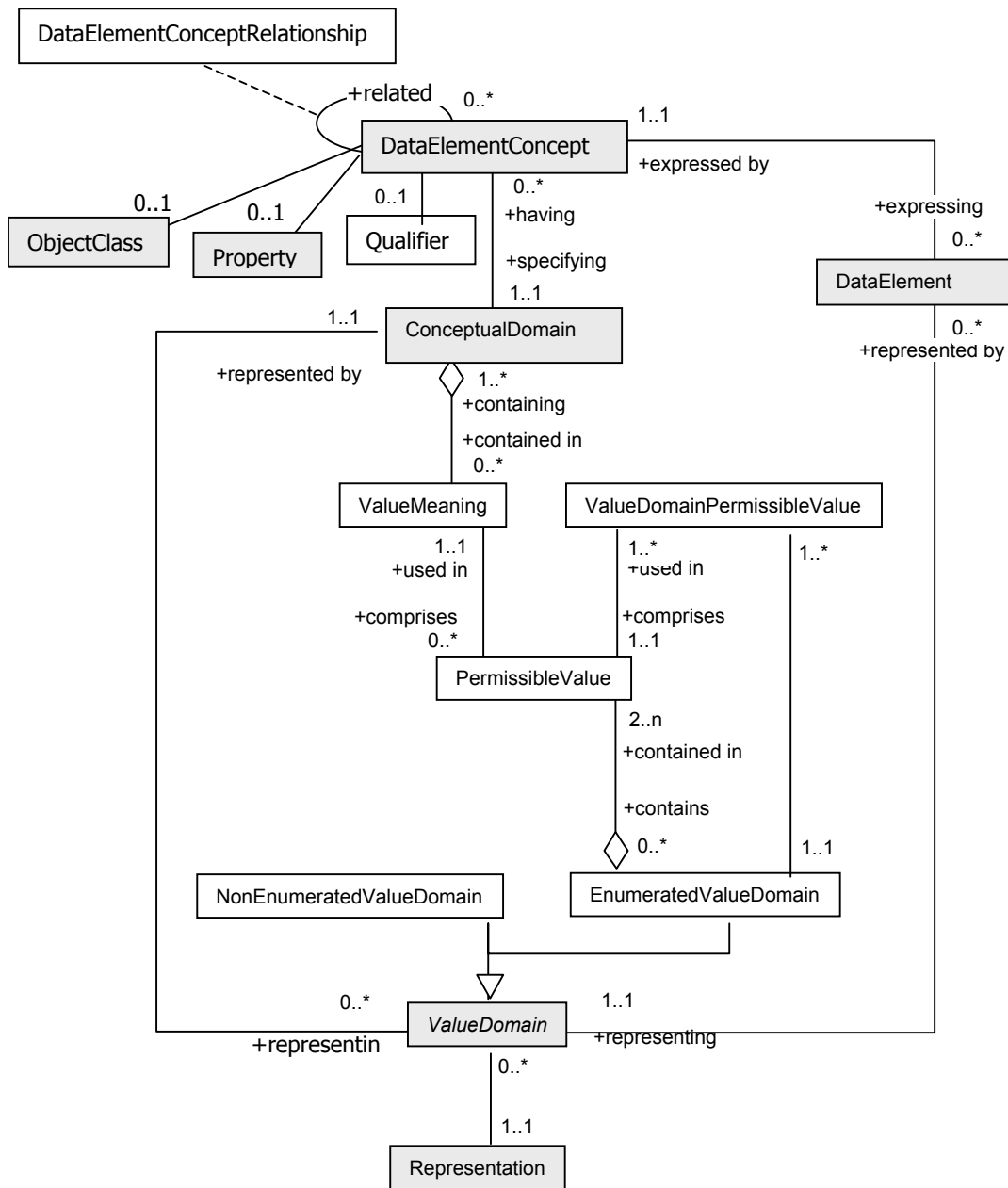


Figure 5.2-1 Information component infrastructure in the metamodel



All of the components in Figure 5.2-1 that are highlighted in light gray must be *administered*. Pragmatically, this means that there is a formal protocol for creating these components; that there is an approval process in place for accepting newly proposed elements; and that there is a designated authority in charge of stewarding the component. Technically, this means that each of the highlighted components is derived from a parent class named *AdministeredComponent*. Table 5.2-1 provides definitions for the new components introduced in Figure 5.2-1.

**Table 5.2-1 Information components in the caDSR metamodel**

<b>Component Name</b>	<b>Definition</b>
<i>ConceptualDomain</i>	The set of all valid value meanings of a Data Element Concept expressed without representation.
<i>DataElement</i>	A unit of data for which the definition, identification, representation, and permissible values are specified by means of a set of attributes.
<i>DataElementConcept</i>	A concept that can be represented in the form of a data element, independent of any particular representation.
<i>DataElementConceptRelationship</i>	An affiliation between two instances of Data Element Concepts.
<i>EnumeratedValueDomain</i>	A value domain expressed as a list of all permissible values.
<i>NonenumeratedValueDomain</i>	A value domain expressed by a generative rule or formula; for example: "all even integers less than 100."
<i>ObjectClass</i>	A set of ideas, abstractions, or things in the real world that can be identified with explicit boundaries and meaning and whose properties and behavior follow the same rules.
<i>PermissibleValue</i>	The exact names, codes, and text that can be stored in a data field in an information management system.
<i>Property</i>	A characteristic common to all members of an Object Class. It may be any feature naturally used to distinguish one individual object from another. It is conceptual and thus has no particular associated means of representation.
<i>Qualifier</i>	A term that helps define and render a concept unique. For example, given the ObjectClass <code>household</code> and the Property <code>annual income</code> , a Qualifier could be used to indicate <code>previous year</code> .
<i>Representation</i>	Mechanism by which the functional and/or presentational category of an item may be conveyed to a user. Examples: 2-digit country code, currency, YYYY-MM-DD, etc.
<i>ValueDomain</i>	A set of permissible values for a data element.
<i>ValueDomainPermissibleValue</i>	The many-to-many relationship between value domains and permissible values; allows one to associate a value domain to a permissible value.
<i>ValueMeaning</i>	The significance or intended meaning of a permissible value.

An *AdministeredComponent* is literally a component for which administrative information must be recorded. It may be a *DataElement* itself or one of its associated components (*Representation*, *ValueDomain*, *DataElementConcept*, *ConceptualDomain*, *ObjectClass*, or *Property*) that requires explicit specifications for reuse in or among enterprises—an *AdministeredComponent* is a generalization for all of the descendant components that are highlighted in Figure 5.2-1. Table 5.2-2 lists the class attributes of an *AdministeredComponent*.

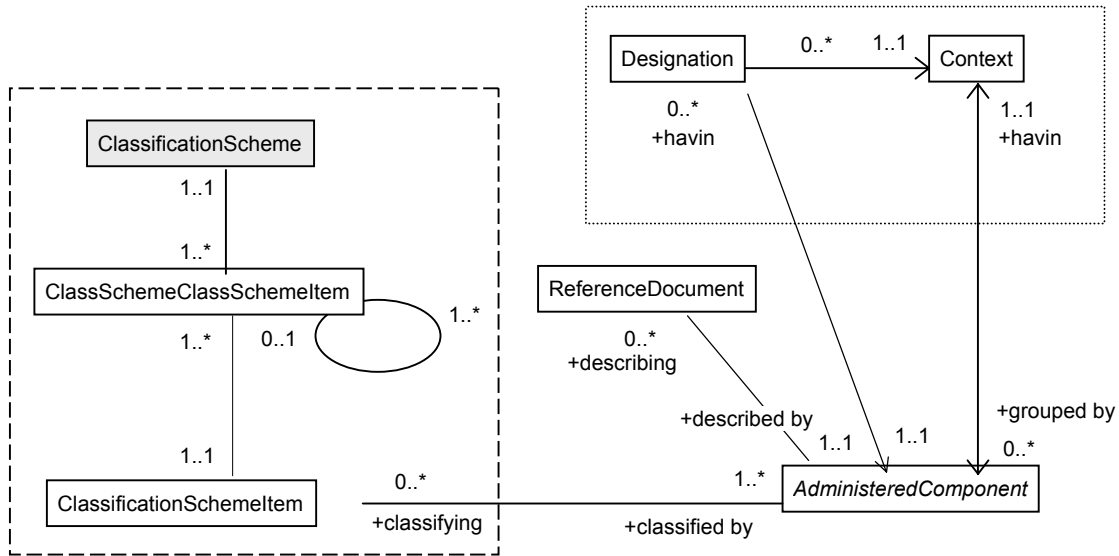
**Table 5.2-2 Class attributes of an *AdministeredComponent***

Attribute Name	Type	
id	String	required
preferredName	String	required
preferredDefinition	String	required
longName	String	required
version	Float	required
workflowStatusName	String	required
workflowStatusDescription	String	required
latestVersionIndicator	Boolean	required
beginDate	Date	required
endDate	Date	required
deletedIndicator	Boolean	optional
changeNote	String	optional
unresolvedIssue	String	optional
origin	String	optional
dateCreated	Date	required
dateModified	Date	required
registration	String	optional

The attributes listed in Table 5.2-2 tell only half the story; additional critical information about an *AdministeredComponent* derives from its associations with the organizational and administrative components depicted in Figure 5.2-2. Of these components, the only element that is also itself an administered component is the *ClassificationScheme*.

Two “regions” are outlined in Figure 5.2-2: (1) the Naming and Identification region (upper right), and (2) the Classification region (lower left). The *ReferenceDocument* component is not included in either region. Each *AdministeredComponent* may be associated with one or more *ReferenceDocuments* that identify where and when the component was created and provide contact information for the component’s designated registration authority.

The purpose of the Naming and Identification region is to manage the various names by which components are referenced in different contexts. Many components may be referenced by different names depending on the discipline, locality, and technology in which they are used. In addition to the name attributes contained in the component itself (*preferredName*, *longName*), an administered component may have any number of alternative *Designations*. Each *Designation* is associated with exactly one *Context* reflecting its usage.



**Figure 5.2-2 Administrative and organizational components of the caDSR metamodel**

The Classification region is used to manage classification schemes and the administered components that are in those classification schemes. Classification is a very fundamental and powerful way of organizing information to make the contents more accessible. Abstractly, a classification *scheme* is any set of organizing principles or dimensions along which data can be organized. In the ISO/IEC 11179 model, a *ClassificationScheme* may be something as simple as a collection of keywords or as complex as an ontology. The classification scheme element in Figure 5.2-2 is highlighted in light gray to reflect that it is an administered component.

Classification schemes that define associations among components can greatly assist navigation through a large network of elements; the associations may describe simple subsumption hierarchies or more complex relations such as causal or temporal relations. In particular, classification schemes with inheritance can enhance self-contained definitions by contributing the definition of one or more ancestors.

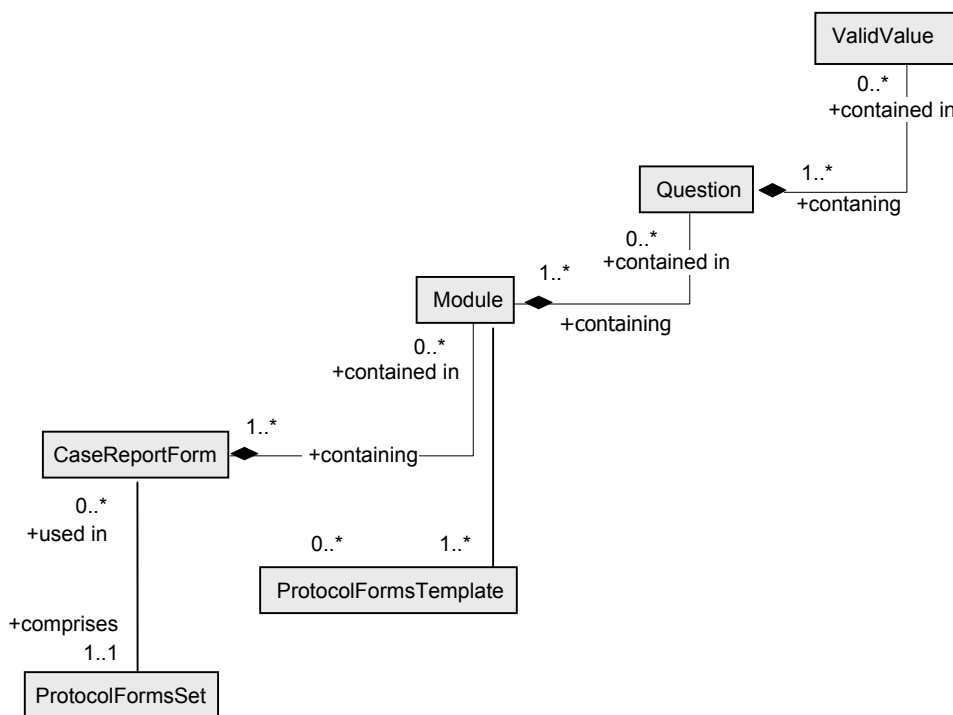
The *ClassificationScheme* component serves as a container-like element that collects the *ClassificationSchemeItems* participating in the scheme. In addition, the *ClassificationScheme* component identifies the source of the classification system and contains an indicator specifying that the scheme is alphanumeric, character, or numeric.

A *ClassificationSchemeItem* may be a node in a taxonomy, a term in a thesaurus, a keyword in a collection, or a concept in an ontology—in all cases, it is an element that is used to *classify* administered components. It is quite natural for an administered component that is used in different contexts to participate in several classification schemes. Classification schemes may

coexist and a classified component may have a different name in each one, since each scheme is from a different context.

The *ClassSchemeClassSchemeItem* in the caDSR model is not a component of the ISO/IEC 11179 metamodel, but serves an important role in the implementation of the many-to-many mappings between *ClassificationSchemeItems* and *ClassificationSchemes*. This component is used to associate a set of classification scheme items with a particular classification scheme, and to store details of that association such as the display order of the items within that scheme.

In addition to the caDSR components corresponding to elements of the ISO/IEC 11179 metamodel, the caDSR model defines a collection of domain-specific elements for capturing clinical trials data. All of the components described up to this point provide the infrastructure for managing shared data. The clinical trials components exercise the representational power of the metamodel, and are used to specify how clinical trials data should be captured and exchanged.



**Figure 5.2-3 Components in the caDSR metamodel for clinical trials data**

All of the components in Figure 5.2-3 are highlighted in light gray, as they are *AdministeredComponents* designed for use in NCI-sponsored clinical trials. Note that because these elements are *not* part of the ISO-11179 specification, they are not technically speaking, ISO administered components. This caDSR design decision was made to ensure that these shared data elements could be stewarded and controlled adequately.

NCI-sponsored programs can populate the registry with instances of these components as needed to specify the metadata descriptors needed for that program. Programs currently participating in this effort include:

- The Cancer Therapy Evaluation Project ([CTEP](#))

- Specialized Programs of Research Excellence ([SPORes](#))
- The Early Detection Research Network ([EDRN](#))
- The Division of Cancer Prevention ([DCP](#))
- The Cancer Imaging Program ([CIP](#))
- The Division of Cancer Epidemiology and Genetics ([DECG](#))
- The Cancer Bioinformatics Infrastructure Objects Project (caBIO)

**Table 5.2-3 Components in the caDSR metamodel for clinical trials data**

<b>Component Name</b>	<b>Component Description</b>
CaseReportForm (CRF)	A questionnaire that is a collection of data elements used to document patient information stipulated in the protocol. A CRF is used by clinicians to record information about patients' visits in a clinical trial.
Question	The text that accompanies a data element on a CRF; used to clarify the information being requested.
Module	A logical grouping of data elements on a CRF.
ProtocolFormsSet	A specific clinical trial protocol document and its collection of associated CRFs. Clinical trial protocols, along with their associated CRFs, stipulate the execution of clinical trials. A protocol is uniquely identify by a protocol ID, protocol version, and Context name.
ProtocolFormsTemplate	A boilerplate collection of components (modules, questions and valid values) to be included in a Case Report Form. The template form is not associated with any particular clinical trial.
ValidValue	An allowable value for a data element (question) on a CRF.

### 5.3 The caDSR API

The previous section described three broad categories of component in the caDSR metamodel and presented each of these independently, thus implying that there are no dependencies among these groupings. Figure 5.3-1 brings these components together and exposes the associations actually occurring between components in different categories.

As in the previous diagram, all components highlighted in gray are descendants of the *AdministeredComponent* class. We emphasize, however, that some of these elements—i.e., those supporting clinical trials specific data—are not defined in the ISO/IEC 11179 standard, but are nevertheless implemented as subclasses of the *AdministeredComponent* class in the caDSR implementation for pragmatic reasons.

Because so many components are *AdministeredComponent* subclasses, we use color coding instead of the standard UML generalization notation (a line ending in an open triangle) to indicate this. Other superclass-subclass relations, such as the *ValueMeaning* class derived from *PermissibleValue*, do however use the standard UML notation.

The three categories of components are also outlined in the figure: administrative and organizational components are in the upper left, clinical trials components are in the upper right, and information components are centered underneath these two.

Figure 5.3-2 summarizes the caDSR API class hierarchy. At the most abstract level, a single distinguished object called the *DomainObject* class is the ancestor to all other classes. At the next level, the *AdministeredComponent* class is defined, along with all other classes which do not represent elements requiring administration. Among the *AdministeredComponent* subclasses, only the *ValueDomain* class has further specialization, i.e., the *EnumeratedValueDomain* and *NonEnumeratedValueDomain* classes.

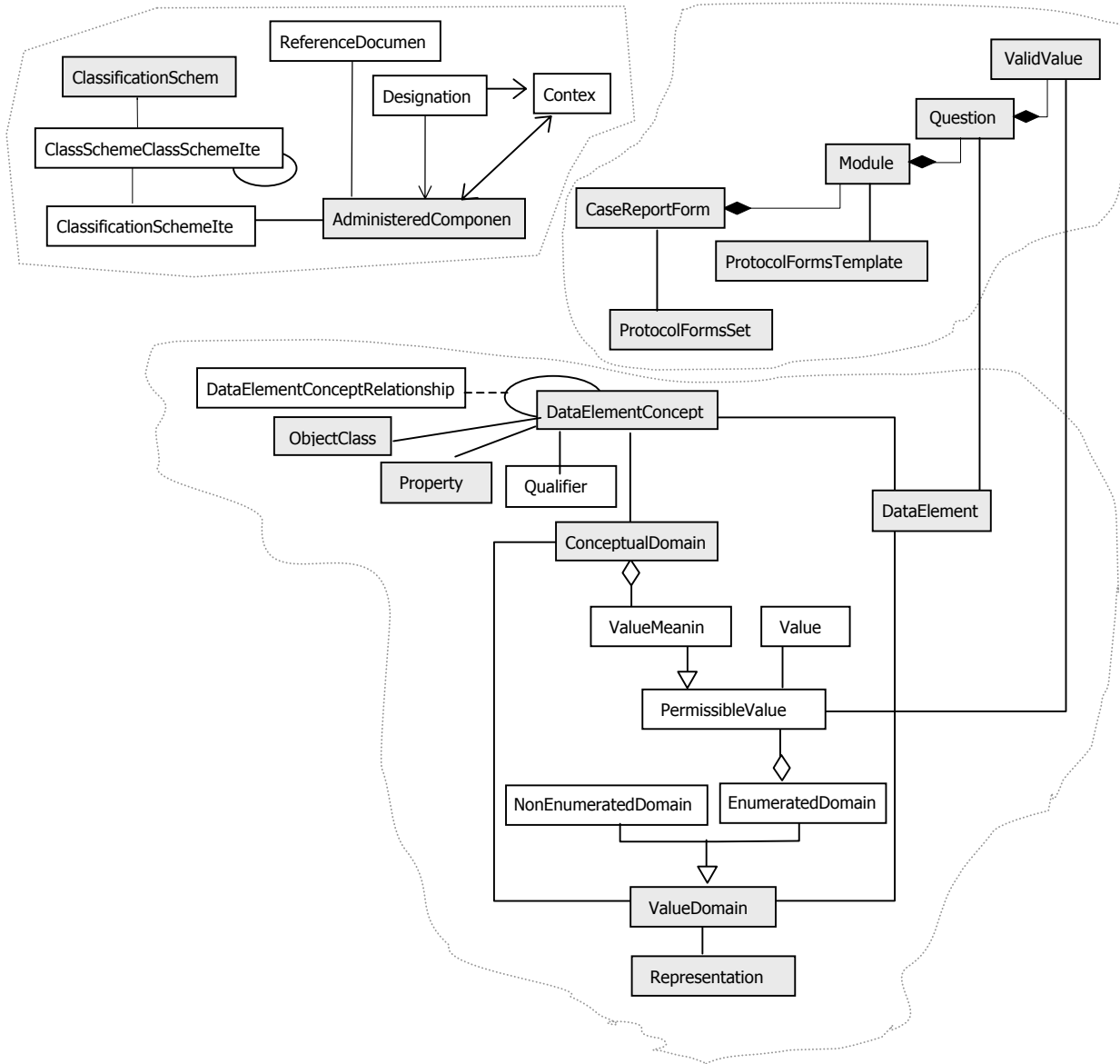


Figure 5.3-1 The caDSR domain objects in the caCORE Java API

This discussion was intended to provide a descriptive overview of the domain objects included in the caCORE API to the caDSR. More detailed modeling information can be found on the caDSR Rose Web Publisher pages, and more concrete specifications are available on the [JavaDocs](#) pages. Sample code that exercises the Java API is described in [Section 12.3](#) and listed in [Appendix C](#). [Section 5.5](#) summarizes the classes described here.

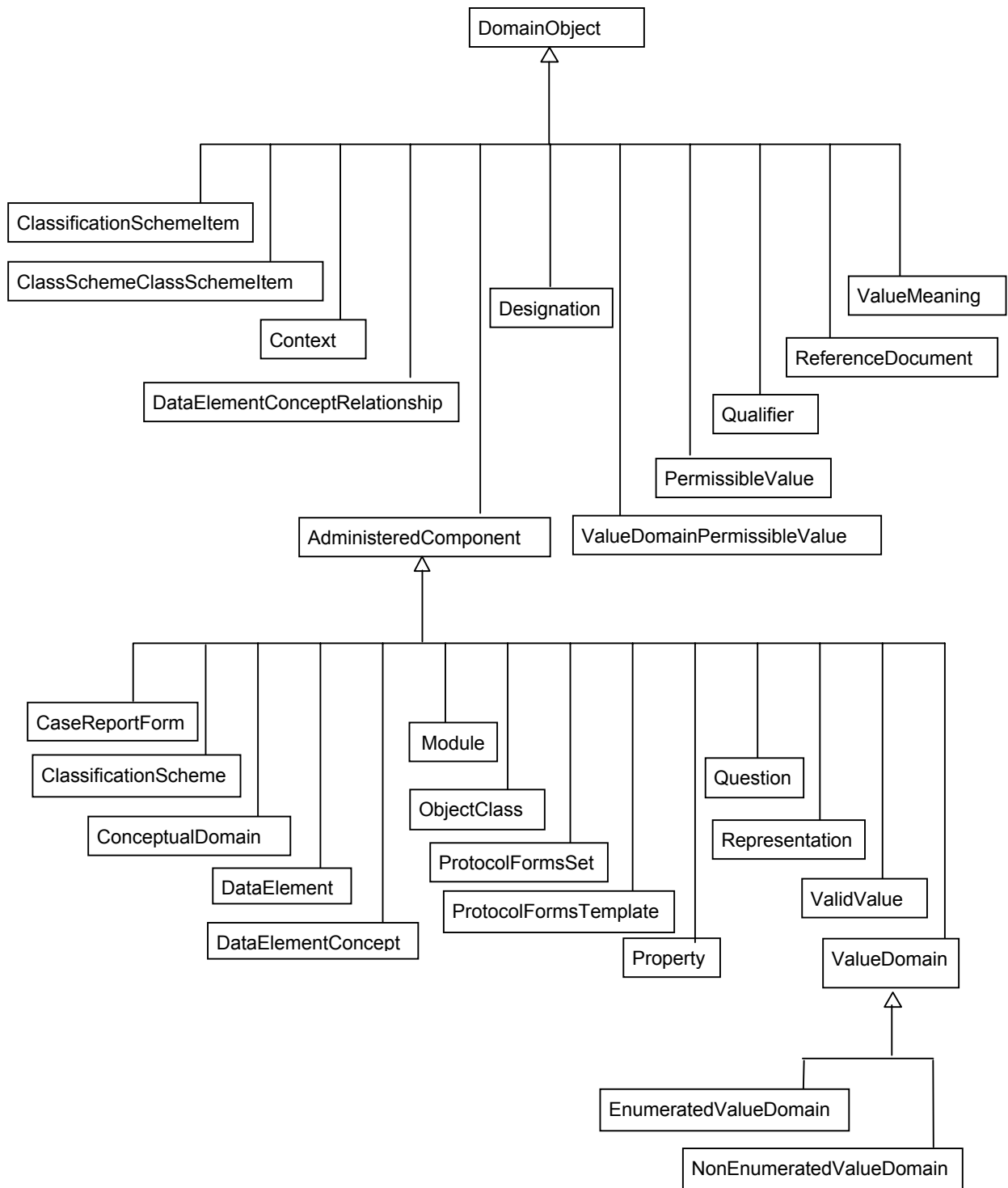


Figure 5.3-2 The caDSR API class hierarchy

## 5.4 Downloading the caDSR

The caDSR repository and admin tool, along with Java source code for the CDE Curation Tool and the CDE Browser, are available for download from the [NCICB download](#) site. Step-by-step instructions are provided there, along with the data files and source code.

The installation guides include:

- caDSR\_2.0\_Install\_Guide.doc
- CDE Curation Tool V2.0 Installation.doc
- Install\_CDBrowser\_2.0\_9ias.txt (versions for Unix and Windows NT)
- Install\_CDBrowser\_2.0\_On\_OC4J.txt (versions for Unix and Windows NT)

The caDSR repository/admin tool install requires the prior installation and configuration of the Oracle Relational Database Management System (RDBMS) (8.1.7 or higher) and Oracle 9ias Release 9.0.2.2.3. Both the CDE Curation Tool and the CDE Browser require that you have previously downloaded and installed the caDSR repository/admin tool. The CDE Browser requires either Oracle 9ias 9.0.2.2.3 or Oracle OC4J 902. The CDE Curation tool also requires the following components to successfully install the tool:

- JDK 1.4.1\_02 or higher
- Tomcat 4.1.18 or equivalent J2EE container
- Oracle 9i client software

## 5.5 The caDSR Domain Object Catalog

This catalog lists the objects defined in the *gov.nih.nci.cadsr.bean* package. Items in the listing below should be interpreted as follows:

- *Application*: This field indicates whether or not the object is a component defined in the ISO/IEC 11179 model, an object introduced to support a specific application area, or an object introduced by the caDSR for general implementation purposes.
- *Related domain objects*: A second domain object is “related” to the first domain object if that second object occurs anywhere in the signature of a method for the first object.
- *Extends*: This field reflects direct inheritance; i.e., the current class is a direct subclass of that which it extends.
- *Implements*: This field lists all interfaces implemented by the object.

In cases where no applications, relations, extensions, or interface implementations apply, the above fields are omitted. Note that the terms “related” and “extends” are very narrowly defined here and refer explicitly to the Java implementation. For example, although a *CaseReportForm* may “contain” *Questions*, the class definition does not include any methods that specify a *Question* as either one of its input or output parameters. Thus, a *Question* is not “related” to a *CaseReportForm* object. This actually reflects the information hiding in use, as *Questions* are contained in *CaseReportForm* objects only indirectly, via *ProtocolFormsTemplate* objects.

### 5.5.1 [AdministeredComponent](#)

Literally, a component for which administrative information must be recorded.

*Application*: A component of the ISO/IEC 11179 model, implemented in caDSR.

*Extends*: DomainObject



*Implements:* java.io.Serializable

### **5.5.2 CaseReportForm**

A questionnaire that is a collection of data elements used to document patient information stipulated in the protocol.

*Application:* Used in clinical trials applications.

*Related domain objects:* [ClassificationSchemeItem](#), [Context](#), [Designation](#), [ProtocolFormsSet](#), [ReferenceDocument](#)

*Extends:* AdministeredComponent

*Implements:* XMLInterface, java.io.Serializable

### **5.5.3 ClassificationScheme**

Any set of organizing principles or dimensions along which data can be organized. A classification scheme may be a simple collection of keywords or a complex ontology.

*Application:* A component of the ISO/IEC 11179 model, implemented in caDSR.

*Related domain objects:* [ClassificationSchemeItem](#), [ClassSchemeClassSchemeItem](#), [Context](#), [Designation](#), [ReferenceDocument](#)

*Extends:* AdministeredComponent

*Implements:* XMLInterface, java.io.Serializable

### **5.5.4 ClassificationSchemeItem**

An item or category in a classification scheme used to classify other components; for example, a node in a taxonomy.

*Application:* A component of the ISO/IEC 11179 model, implemented in caDSR.

*Related domain objects:* [CaseReportForm](#), [ClassificationSchemeItem](#), [ClassSchemeClassSchemeItem](#), [ConceptualDomain](#), [DataElementConcept](#), [DataElement](#), [EnumeratedValueDomain](#), [Module](#), [NonEnumeratedValueDomain](#), [ObjectClass](#), [Property](#), [ProtocolFormsSet](#), [Question](#), [Representation](#), [ValidValue](#)

*Extends:* DomainObject

*Implements:* XMLInterface, java.io.Serializable

### **5.5.5 ClassSchemeClassSchemeItem**

Used to associate a set of classification scheme items with a particular classification scheme, and to store details of that association such as the display order of the items within that scheme.

*Application:* Defined and used by the caDSR project for implementation purposes.

*Related domain objects:* [ClassificationScheme](#), [ClassificationSchemeItem](#)

*Extends:* DomainObject

*Implements:* XMLInterface, java.io.Serializable

### **5.5.6 ConceptualDomain**

The set of all possible Valid Value meanings of a Data Element Concept expressed without representation.

*Application:* A component of the ISO/IEC 11179 model, implemented in caDSR.

*Related domain objects:* [DataElementConcept](#), [Designation](#), [EnumeratedValueDomain](#), [NonEnumeratedValueDomain](#), [ReferenceDocument](#), [ValueMeaning](#)

*Extends:* AdministeredComponent

*Implements:* XMLInterface, java.io.Serializable

### **5.5.7 [Context](#)**

A designation or description of the application environment or discipline in which a name is applied or from which it originates.

*Application:* A component of the ISO/IEC 11179 model, implemented in caDSR.

*Related domain objects:* [DataElementConcept](#), [DataElement](#), [Designation](#), [EnumeratedValueDomain](#), [Module](#), [NonEnumeratedValueDomain](#), [ObjectClass](#), [Property](#), [ProtocolFormsSet](#), [ProtocolFormsTemplate](#), [Question](#), [Representation](#), [ValidValue](#)

*Extends:* DomainObject

*Implements:* XMLInterface, java.io.Serializable

### **5.5.8 [DataElement](#)**

A unit of data for which the definition, identification, representation, and permissible values are specified by means of a set of attributes.

*Application:* A component of the ISO/IEC 11179 model, implemented in caDSR.

*Related domain objects:* [ClassificationSchemeItem](#), [Context](#), [DataElementConcept](#), [Designation](#), [EnumeratedValueDomain](#), [NonEnumeratedValueDomain](#), [Question](#), [ReferenceDocument](#)

*Extends:* AdministeredComponent

*Implements:* XMLInterface, java.io.Serializable

### **5.5.9 [DataElementConcept](#)**

A concept that can be represented in the form of a data element and described independent of any particular representation.

*Application:* A component of the ISO/IEC 11179 model, implemented in caDSR.

*Related domain objects:* [DataElementConceptRelationship](#), [Property](#), [Qualifier](#)

*Extends:* AdministeredComponent

*Implements:* XMLInterface, java.io.Serializable

### **5.5.10 [DataElementConceptRelationship](#)**

A description of the affiliation between two occurrences of Data Element Concepts.

*Application:* A component of the ISO/IEC 11179 model, implemented in caDSR.

*Related domain objects:* [DataElementConcept](#)

*Extends:* DomainObject

*Implements:* XMLInterface, java.io.Serializable

### **5.5.11 [Designation](#)**

A name by which an Administered Component is known in a specific context, user database, or application. Also a placeholder to track the usage of Administered Components by different Contexts.

*Application:* A component of the ISO/IEC 11179 model, implemented in caDSR.

*Extends:* gov.nih.nci.caDSR.bean.DomainObject

*Implements:* XMLInterface, java.io.Serializable

### **5.5.12 [DomainObject](#)**

An abstract class that serves as the parent or ancestor to all other classes in the caDSR *bean* package.

*Application:* An implementation class in the caDSR model.

*Related domain objects:* [CaseReportForm](#), [ClassificationScheme](#), [ConceptualDomain](#), [Context](#), [DataElement](#), [DataElementConcept](#), [EnumeratedValueDomain](#), [Module](#), [NonEnumeratedValueDomain](#), [ObjectClass](#), [Property](#), [ProtocolFormsSet](#), [ProtocolFormsTemplate](#), [Question](#), [Representation](#), [ValidValue](#)

*Extends:* java.lang.Object

### **5.5.13 [EnumeratedValueDomain](#)**

A value domain expressed as a list of all permissible values.

*Application:* A component of the ISO/IEC 11179 model, implemented in caDSR.

*Related domain objects:* [ClassificationSchemeItem](#), [Context](#), [DataElement](#), [Designation](#)

*Extends:* ValueDomain

*Implements:* XMLInterface, java.io.Serializable

### **5.5.14 [Module](#)**

A collection of data elements logically grouped on a CRF.

*Application:* Used in clinical trials applications.

*Related domain objects:* [ClassificationScheme](#), [Context](#), [Designation](#), [Question](#), [ReferenceDocument](#)

*Extends:* AdministeredComponent

*Implements:* XMLInterface, java.io.Serializable

### **5.5.15 [NonenumeratedValueDomain](#)**

A value domain expressed by a generative formula or range of allowed values.

*Application:* A component of the ISO/IEC 11179 model, implemented in caDSR.

*Related domain objects:* [ClassificationSchemeItem](#), [ConceptualDomain](#), [Context](#), [DataElement](#), [Designation](#), [Qualifier](#), [ReferenceDocument](#), [Representation](#)

*Extends:* ValueDomain

*Implements:* XMLInterface, java.io.Serializable

### **5.5.16 [ObjectClass](#)**

A set of ideas, abstractions, or things in the real world that can be identified with explicit boundaries and meaning and whose properties and behavior follow the same rules.

*Application:* A component of the ISO/IEC 11179 model, implemented in caDSR.

*Related domain objects:* [ClassificationSchemeItem](#), [Context](#), [DataElementConcept](#), [Designation](#), [ReferenceDocument](#)

*Extends:* AdministeredComponent

*Implements:* XMLInterface, java.io.Serializable

### **5.5.17 [PermissibleValue](#)**

The exact names, codes, and text that can be stored in a data field in an information management system.

*Application:* A component of the ISO/IEC 11179 model, implemented in caDSR.

*Related domain objects:* [EnumeratedValueDomain](#), [ValueDomainPermissibleValue](#), [ValueMeaning](#)

*Extends:* DomainObject

*Implements:* XMLInterface, java.io.Serializable

### **5.5.18 [Property](#)**

A characteristic common to all members of an Object Class. It may be any feature that humans naturally use to distinguish one individual object from another. It is conceptual and thus has no particular associated means of representation by which property can be communicated.

*Application:* A component of the ISO/IEC 11179 model, implemented in caDSR.

*Related domain objects:* [ClassificationSchemeItem](#), [Context](#), [DataElementConcept](#), [Designation](#), [ReferenceDocument](#)

*Extends:* AdministeredComponent

*Implements:* XMLInterface, java.io.Serializable

### **5.5.19 [ProtocolFormsSet](#)**

A specific clinical trial protocol document and its collection of associated CRFs. Clinical trial protocols, along with their associated CRFs, stipulate the execution of clinical trials.

*Application:* Used in clinical trials applications.

*Related domain objects:* [CaseReportForm](#), [ClassificationSchemeItem](#), [Context](#), [Designation](#), [ReferenceDocument](#)

*Extends:* AdministeredComponent

*Implements:* XMLInterface, java.io.Serializable

### **5.5.20 [ProtocolFormsTemplate](#)**

A boilerplate collection of elements (modules, questions, valid values) to be included in a CRF.

*Application:* Used in clinical trials applications.

*Related domain objects:* [ClassificationSchemeItem](#), [Context](#), [Designation](#), [ReferenceDocument](#)

*Extends:* AdministeredComponent

*Implements:* XMLInterface, java.io.Serializable

### **5.5.21 [Qualifier](#)**

A term that helps define and render a concept unique; criterion that further defines or describes an aspect (object class, property, representation) of a DataElementConcept or ValueDomain.

*Application:* A component of the ISO/IEC 11179 model, implemented in caDSR.

*Related domain objects:* [DataElementConcept](#), [EnumeratedValueDomain](#), [NonEnumeratedValueDomain](#)

*Extends:* DomainObject

*Implements:* XMLInterface, java.io.Serializable

### 5.5.22 [Question](#)

The actual text of the data element as specified on a Case Report Form of a Protocol.

*Application:* Used in clinical trials applications.

*Related domain objects:* [ClassificationSchemeItem](#), [Context](#), [DataElement](#), [Designation](#), [Module](#), [ReferenceDocument](#), [ValidValue](#)

*Extends:* AdministeredComponent

*Implements:* XMLInterface, java.io.Serializable

### 5.5.23 [ReferenceDocument](#)

A place to document additional information about Administered Components that is not readily stored elsewhere.

*Application:* A component of the ISO/IEC 11179 model, implemented in caDSR.

*Related domain objects:* [CaseReportForm](#), [ClassificationScheme](#), [ConceptualDomain](#), [DataElement](#), [DataElementConcept](#), [EnumeratedValueDomain](#), [Module](#), [NonEnumeratedValueDomain](#), [ObjectClass](#), [Property](#), [ProtocolFormsSet](#), [ProtocolFormsTemplate](#), [Question](#), [Representation](#), [ValidValue](#)

*Extends:* DomainObject

*Implements:* java.io.Serializable

### 5.5.24 [Representation](#)

Mechanism by which the functional and/or presentational category of an item may be conveyed to a user. Component of a Data Element Name that describes how data are represented (i.e., the combination of a ValueDomain, data type, and, if necessary, a unit of measure or a character set). The Representation occupies the last (rightmost) position in the DataElement name.

*Application:* A component of the ISO/IEC 11179 model, implemented in caDSR.

*Related domain objects:* [ClassificationSchemeItem](#), [Context](#), [Designation](#), [EnumeratedValueDomain](#), [NonEnumeratedValueDomain](#), [ReferenceDocument](#)

*Extends:* AdministeredComponent

*Implements:* XMLInterface, java.io.Serializable

### 5.5.25 [ValidValue](#)

The allowable values for a given data element (question) on a CRF.

*Application:* Used in clinical trials applications.

*Related domain objects:* [ClassificationSchemeItem](#), [Context](#), [Designation](#), [Question](#), [ReferenceDocument](#), [ValueDomainPermissibleValue](#)

*Extends:* AdministeredComponent

*Implements:* XMLInterface, java.io.Serializable

### 5.5.26 [ValueDomain](#)

A set of permissible values for a data element.

*Application:* A component of the ISO/IEC 11179 model, implemented in caDSR.

*Related domain objects:* [CaseReportForm](#), [ClassificationScheme](#), [ClassificationSchemeItem](#), [ClassSchemeClassSchemeItem](#), [ConceptualDomain](#), [Context](#), [DataElementConcept](#), [DataElement](#), [Designation](#), [EnumeratedValueDomain](#), [Module](#), [NonEnumeratedValueDomain](#),

[ObjectClass](#), [Property](#), [ProtocolFormsSet](#), [ProtocolFormsTemplate](#), [Qualifier](#), [Question](#), [ReferenceDocument](#), [Representation](#), [ValidValue](#)

*Extends:* AdministeredComponent

*Implements:* XMLInterface, java.io.Serializable

### **5.5.27 [ValueDomainPermissibleValue](#)**

Captures the many-to-many relationships between value domains and permissible values and allows one to associate a value domain to a permissible value.

*Application:* A component of the ISO/IEC 11179 model, implemented in caDSR.

*Related domain objects:* [EnumeratedValueDomain](#), [PermissibleValue](#), [ValidValue](#)

*Extends:* DomainObject

*Implements:* XMLInterface, java.io.Serializable

### **5.5.28 [ValueMeaning](#)**

The significance associated with an allowable/permissible value.

*Application:* A component of the ISO/IEC 11179 model, implemented in caDSR.

*Related domain objects:* [ConceptualDomain](#), [PermissibleValue](#)

*Extends:* DomainObject

*Implements:* XMLInterface, java.io.Serializable

## **6.0 THE caMOD DOMAIN OBJECTS**

Animal models that mimic the course of human cancers can provide critical insight to the molecular etiology of the associated disease processes. Most cancers result from a complex of disorders involving multiple biologic pathways. Murine (rat and mouse) models can be used to study these disorders—both in isolation and in combination. These models can be manipulated by a variety of techniques, including genetic engineering, chemical treatment, and exposure to carcinogenic levels of radiation, to produce

- underexpression of tumor suppressor genes;
- overexpression of oncogenes;
- impaired immune functions;
- induced tumors; and
- mutagenized strains with germline mutations of relevant genes.

Each such treatment or combination of treatments yields strain-specific differences in lifespan, tumor location, histology, and time of onset. Careful manipulation of these factors can be applied to selectively model many aspects of human cancers and potentially, to shed light on the differential roles played by the affected genes in each strain.

The NCI Mouse Models of Human Cancers Consortium ([MMHCC](#)) is a collaborative program designed to derive and characterize mouse models, and to generate resources, information, and innovative approaches to the application of these models in cancer research. It is the goal of the consortium to make information and materials concerning animal models of human cancer as widely available as possible to the entire cancer research community.

In order to achieve this goal the MMHCC has initiated the development of three web-based resources:

- The [Emice](#) web site
- The Cancer Models Database ([caMOD](#))
- The Cancer Images Database ([caIMAGE](#))

This chapter describes an application programming interface to the Cancer Models Database at NCI. The database contains information about animal models that has been contributed by the broader research community, including the consortium members. Many of the developed strains are available from the [MMHCC Repository](#) at NCI-Frederick, from the [Jackson Laboratory](#) in Maine, or directly from the principal investigators.

The NCICB Applications User Manual provides a broader discussion of both the caMOD and caIMAGE databases, and includes detailed instructions on how to use the web interfaces to these resources.

## **6.1 The Mouse Models of Human Cancers Consortium**

Although initially driven by the MMHCC, the Cancer Models Database was designed to represent the full range of possible cancer models—not just murine models. Currently, the database stores information on murine models only, but the foundations and infrastructure to extend this to all animal models has been implemented.

In order to achieve this flexibility and yet maintain data accessibility, the model requires the explicit specification of various parameters and identifying characteristics. This section describes the type of information that is submitted and maintained for each model. The web interface presents these different types of information as a collection of separate web pages. This structure



is not intrinsic to the API implementation, but provides a useful framework for discussion. Table 6.1-1 describes some of the information available on these web pages.

**Table 6.1-1 The web interface information pages for the Cancer Models Database**

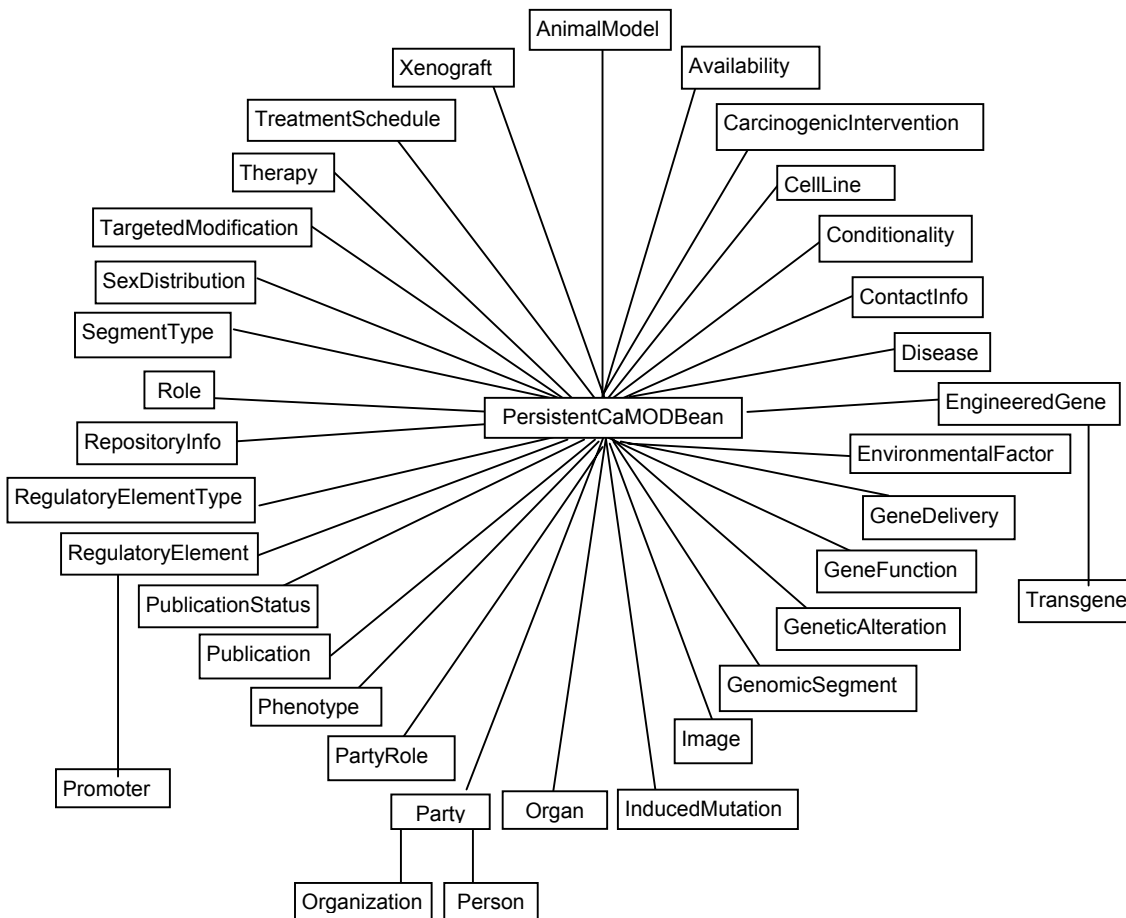
<u>Web Page</u>	<u>Description</u>
General Information	Provides an overview of the experimental design, phenotype and availability of the model; elements on this page include: model descriptor (name), official nomenclature, genotype, principal investigator's laboratory, location where the model is available, species, genetic background, experimental design, phenotype, sex distribution of phenotype, breeding notes, and record release date.
Genetic Description	Information about transgenes, targeted mutations, and targeted transgenes.
Carcinogenic Interventions	Information about chemicals/drugs used, relevant growth factors, hormones, radiation treatments, viral agents, and surgical procedures. This page also contains information on xenografts and allografts.
Publications	Used to specify any citations associated with the generation of the model, phenotype, therapeutic experiments, or cell lines and experiments conducted on those cell lines.
Histopathology	Used to capture information about the organ where the tumor/lesion arises, the diagnosis, and other tumor-related data like time of onset, tumor incidence, and any genetic aberrations observed in the lesion.
Therapeutic Approaches	Cites any therapeutic trials associated with the model.
Cell Lines	Used to capture the name of the cell line, the organ of origin, and experiments conducted with the cell line that was generated from the model.
Images	Provides links to any image data associated with the model.
Microarray	Provides links to any microarray data associated with the model.

The caMOD web interface provides both simple and advanced search methods based on these stored characteristics. The simple search form retrieves models based on the principal investigator's name, the model descriptor (model name), the site of the lesion or tumor, and/or the species. The advanced search mode extends the basic search and includes options for searching on the genetic description, carcinogenic agents, phenotype, cell lines, therapeutic approaches, and microarray data associated with the model.

## 6.2 The caMOD API

Figure 6.2-1 features the *PersistentCaMODBean* class at the center of a broad, shallow hierarchy of classes defined in the *gov.nih.nci.caMOD.bean* package. Like the caBIO domain

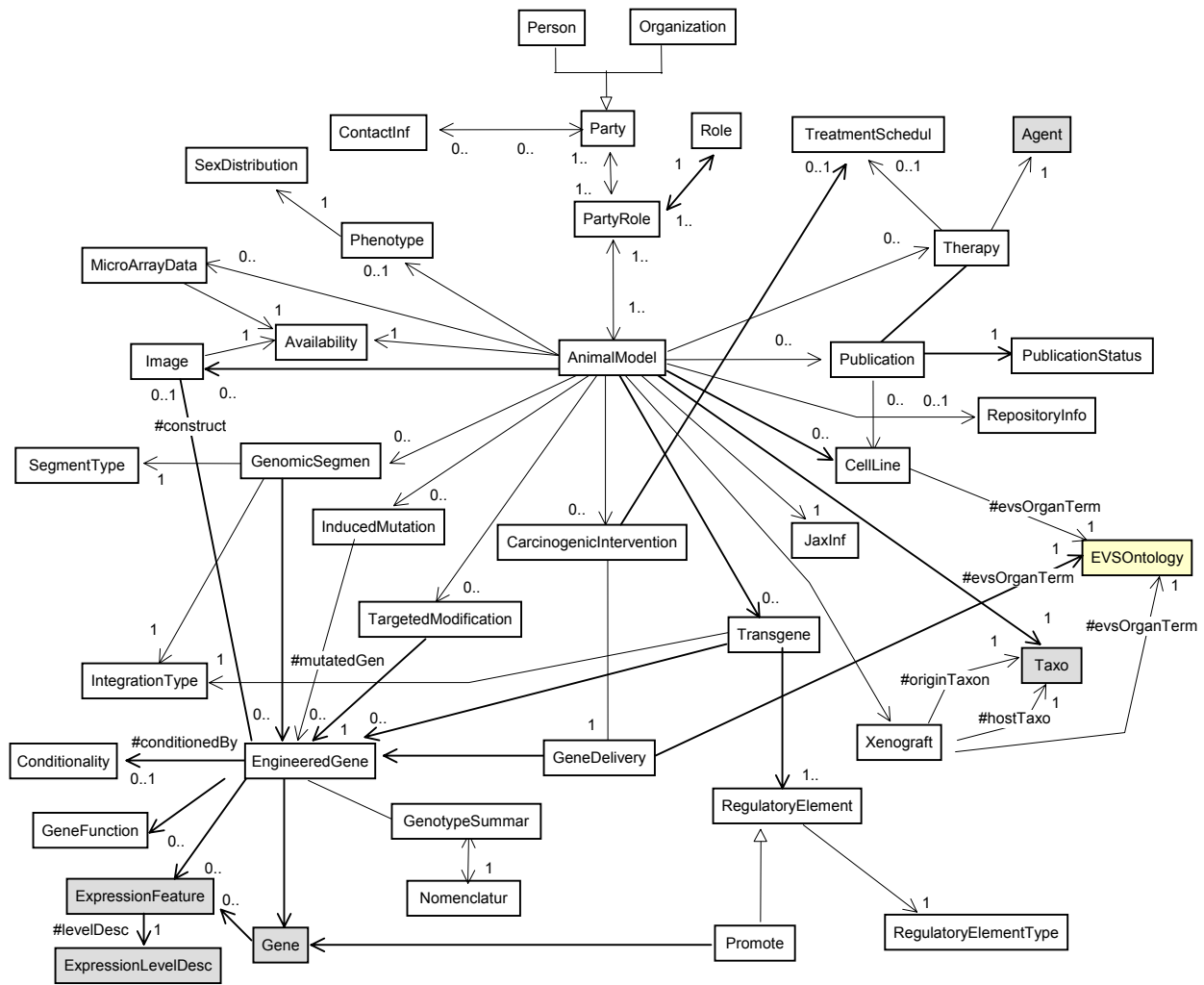
objects defined in Chapter 3, all of the classes in Figure 6.2-1 have associated search criteria objects, which can be deployed to retrieve stored instances of these classes satisfying the specified attributes. And like all search criteria objects, these objects can be nested to develop complex and precise queries. In this case however, the search criteria objects are defined in the *search* package.



**Figure 6.2-1 The caMOD class hierarchy**

The associated search criteria classes are likewise derived from a central parent class in the *gov.nih.nci.common.search* package called *SearchCriteria*. A similar hierarchy is defined, with all classes descending from *SearchCriteria*, with the exception of the *PromoterSearchCriteria*, *TransgeneSearchCriteria*, *OrganizationSearchCriteria* and *PersonSearchCriteria* classes. Like their associated domain objects, these classes are one level deeper in the hierarchy.

Figure 6.2-2 shows a partial class diagram for the domain objects contained in the *gov.nih.nci.caMOD.bean* package. The *AnimalModel* class sits at the center, as all of the other classes are either direct or indirect attributes of the model. For each relation emanating from *AnimalModel*, the *AnimalModel* class defines a *get* method, e.g., *getHistopathologies()*, *getInducedMutations()*, *getCarcinogenicInterventions()*, *getGenomicSegments()*, *getPhenotype()*, etc. In addition, for those methods which return an array of elements, an auxiliary method to determine the size of the array is provided, e.g., *getHistopathologiesCount()*.



**Figure 6.2-2 The UML Class diagram for the caMOD Domain Objects**

The diagram has been simplified by eliminating the relationship labels where they correspond directly to the class names. For example, the class *PartyRole* has a relation named “party” that associates it with an instance of class *Party*. In cases where the relation names cannot be inferred from the class names, the labels are made explicit. For relations that have arities greater than one (such as 0..\*), the named relation is pluralized, as in *EngineeredGene*’s relation to *Gene*, “genes.”

Two colors are used to highlight classes in the diagram that are not defined in the caMOD *bean* package. Classes highlighted in gray are defined in the *gov.nih.nci.caBIO.bean* package. The inclusion of the three gene-related classes reflects the close collaboration among gene-related objects defined in the caBIO and caMOD packages.

The *EVSOntology* class (highlighted in yellow) is defined in the *gov.nih.nci.common.domain* package, and is used by the caMOD classes to access organ names curated in the EVS vocabularies.

Several lightweight classes are not included in either of the above figures; these are the classes that are used to track the status of models in the database, and each of these implements the *ApprovalStatus* interface. The status of a model is used to encode the various stages it passes through before becoming accessible to the public. Although these classes are defined in the caMOD *bean* package, they are direct subclasses of *java.lang.Object*.

### 6.3 The caMOD Domain Object Catalog

This catalog lists the objects defined in the *gov.nih.nci.caMOD.bean* package. Items in the listing below should be interpreted as follows:

- *Related MMHCC domain objects*: A second MMHCC domain object is “related” to the first domain object if that second object occurs anywhere in the signature of a method for the first object.
- *Other related domain objects*: The same as above, but in this case the second domain object is defined outside the *gov.nih.nci.caMOD.bean* package.
- *Extends*: This field reflects direct inheritance; i.e., the current class is a direct subclass of that which it extends.
- *Implements*: This field lists all interfaces implemented by the object.

In cases where no relations, extensions, or interface implementations apply, the above fields are omitted.

#### 6.3.1 [AnimalModel](#)

A strain of animals used to study the various types of cancer.

*Related caMOD domain objects*: [Availability](#), [CarcinogenicIntervention](#), [CellLine](#), [GenomicSegment](#), [Image](#), [InducedMutation](#), [JaxInfo](#), [MicroArrayData](#), [PartyRole](#), [Phenotype](#), [Publication](#), [RepositoryInfo](#), [TargetedModification](#), [Therapy](#), [Transgene](#), [Xenograft](#)

*Other related objects*: [gov.nih.nci.cabio.bean.Taxon](#), [gov.nih.nci.cabio.bean.Histopathology](#)

*Extends*: PersistentCaMODBean

*Implements*: XMLInterface, java.io.Serializable

#### 6.3.2 [Availability](#)

The availability status of a developed strain from the MMHCC Repository, the Jackson Laboratory, or directly from the principal investigator.

*Extends*: PersistentCaMODBean

*Implements*: XMLInterface, java.io.Serializable

#### 6.3.3 [CarcinogenicIntervention](#)

A treatment (chemical or drug administration, radiation, etc) applied to an animal model to induce a disease state.

*Related caMOD domain objects*: [EnvironmentalFactor](#), [GeneDelivery](#), [TreatmentSchedule](#)

*Extends*: PersistentCaMODBean

*Implements*: XMLInterface, java.io.Serializable

#### **6.3.4**    [CellLine](#)

A cell line generated from a particular strain of animal model.

*Related caMOD domain objects:* [Publication](#)

*Other related objects:* gov.nih.nci.common.domain.EVSOntology

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

#### **6.3.5**    [CompleteNotScreened](#)

Used to indicate the status of a model which has been submitted but not yet screened.

*Extends:* java.lang.Object

*Implements:* ApprovalStatus, java.io.Serializable

#### **6.3.6**    [Conditionality](#)

Indicates if a transgene or targeted modification is time- or tissue-specific.

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

#### **6.3.7**    [ContactInfo](#)

Contact information for the person who submitted data for the selected model.

*Related caMOD domain objects:* [Party](#)

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

#### **6.3.8**    [EditorApproved](#)

Used to indicate the status of a model that has been approved by an editor.

*Extends:* java.lang.Object

*Implements:* ApprovalStatus, java.io.Serializable

#### **6.3.9**    [EditorAssigned](#)

Used to indicate the status of a model that has been assigned to an editor but not yet approved.

*Extends:* java.lang.Object

*Implements:* ApprovalStatus, java.io.Serializable

#### **6.3.10**   [EditorMoreInfo](#)

Used to indicate the status of a model that has been assigned to an editor who has requested additional information.

*Extends:* java.lang.Object

*Implements:* ApprovalStatus, java.io.Serializable

### **6.3.11 [EngineeredGene](#)**

A gene sequence that has been genetically modified to induce a desired state in an animal model.

*Related caMOD domain objects:* [Conditionality](#), [GeneFunction](#), [GenomicSegment](#), [GenotypeSummary](#), [Image](#)

*Other related objects:* gov.nci.nih.gov.cabio.bean.Gene,  
gov.nci.nih.gov.cabio.bean.ExpressionFeature

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.12 [EnvironmentalFactor](#)**

A chemical, radiation, or hormone treatment, or other environmental factor that initiates or supports development of neoplasia in the animal model.

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.13 [GeneDelivery](#)**

The method of introducing a modified gene to the recipient animal.

*Related caMOD domain objects:* [EngineeredGene](#)

*Other related objects:* gov.nih.nci.common.domain.EVSOntology

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.14 [GeneFunction](#)**

The known or hypothesized function of a gene.

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.15 [GeneticAlteration](#)**

Genetic alterations found in the diseased tissue other than the engineered genetic changes.

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.16 [GenomicSegment](#)**

A region or set of regions of a genome including chromosome, gene, breakpoint etc. The size of a genomic segment varies from a fraction of a gene to a region containing many genes surrounded by non-coding sequences, and can be as large as a chromosome.

*Related caMOD domain objects:* [EngineeredGene](#), [IntegrationType](#), [SegmentType](#)

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.17** [GenotypeSummary](#)

Summary of genotype information for a particular model

*Related caMOD domain objects:* [Nomenclature](#)

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.18** [Incomplete](#)

Used to indicate the status of a model which has not yet been completed.

*Extends:* java.lang.Object

*Implements:* ApprovalStatus, java.io.Serializable

### **6.3.19** [Image](#)

Images and image annotations associated with animal models.

*Related caMOD domain objects:* [Availability](#)

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.20** [InducedMutation](#)

A mutation in a gene caused by exposure to chemicals, radiation or other types of mutagens. By definition, an induced mutation is inherited by the next generation.

*Related caMOD domain objects:* [EnvironmentalFactor](#), [EngineeredGene](#)

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.21** [IntegrationType](#)

Location of the integration of the engineered gene, e.g. “random” or “targeted”.

*Related caMOD domain objects:* [GenomicSegment](#), [Transgene](#)

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.22** [JaxInfo](#)

Information associated with an animal model in The Jackson Laboratory; for example: stock number, strain name, etc.

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.23** [MicroArrayData](#)

Gene expression data from microarray experiments using cells or tissues from an animal model.

*Related caMOD domain objects:* [Availability](#)

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.24** [ModificationType](#)

The type of gene modification in the target gene, such as the *Null* modification, amino acid substitution, deletion, insertion, misense, nonsense, point mutation, etc.

*Related caMOD domain objects:* [TargetedModification](#)

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.25** [Nomenclature](#)

Official nomenclature name of the animal model, following the recommendations of the International Committee on Standardized Genetic Nomenclature for Mice; nomenclature is also applicable to rats starting in 2001.

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.26** [Organization](#)

The institution (university, institute, laboratory) where a contributing scientist works.

*Extends:* [Party](#)

*Implements:* XMLInterface, java.io.Serializable

### **6.3.27** [Party](#)

A *Person* or group of persons (*Organization*) having a designated relationship (*PartyRole*) to parts of the MMHCC data with explicit access permissions.

*Related caMOD domain objects:* [ContactInfo](#), [PartyRole](#)

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.28** [PartyRole](#)

The PartyRole is used to map designated MMHCC users to the read/write access privileges associated with MMHCC data.

*Related caMOD domain objects:* [AnimalModel](#), [Party](#), [Role](#)

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.29** [Person](#)

An individual involved in the deposition, review, and/or approval of data in MMHCC.

*Extends:* [Party](#)

*Implements:* XMLInterface, java.io.Serializable

### **6.3.30** [Phenotype](#)

The physical appearance or otherwise observable characteristics of a model animal.

*Related caMOD domain objects:* [SexDistribution](#)



*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.31 [Promoter](#)**

A region of DNA sequence upstream of the coding region to which RNA polymerase will bind and initiate replication.

*Related caMOD domain objects:* [RegulatoryElement](#)

*Other related domain objects:* [gov.nih.nci.cabio.bean.Taxon](#)

*Extends:* [RegulatoryElement](#)

*Implements:* XMLInterface, java.io.Serializable

### **6.3.32 [Publication](#)**

Publications describing the animal model itself or experiments in which the animal model was used

*Related caMOD domain objects:* [PublicationStatus](#)

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.33 [PublicationStatus](#)**

Status regarding a scientific paper, e.g., “unpublished”, “submitted”, “published”.

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.34 [RegulatoryElement](#)**

A region of DNA sequence controlling the transcription/expression of a gene.

*Related caMOD domain objects:* [Promoter](#), [RegulatoryElementType](#),

*Other related domain objects:* [gov.nih.nci.cabio.bean.Taxon](#)

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.35 [RegulatoryElementType](#)**

The type of regulation imposed by the element, e.g., suppressor, promoter, etc.

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.36 [RepositoryInfo](#)**

A RepositoryInfo object contains information about the availability of a particular model from the MMHCC repository. Data submitters to the cancer models database can indicate that their model should be submitted to the repository for acceptance for acceptance.

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.37** [ReviewerApproved](#)

Used to indicate the status of a model that has been approved by a reviewer.

*Extends:* java.lang.Object

*Implements:* ApprovalStatus, java.io.Serializable

### **6.3.38** [ReviewerAssigned](#)

Used to indicate the status of a model that has been assigned to a reviewer but not yet approved.

*Extends:* java.lang.Object

*Implements:* ApprovalStatus, java.io.Serializable

### **6.3.39** [ReviewerRejected](#)

Used to indicate the status of a model that has been rejected by the assigned reviewer.

*Extends:* java.lang.Object

*Implements:* ApprovalStatus, java.io.Serializable

### **6.3.40** [Role](#)

Role that a person or organization plays; for example, submitter, reviewer, screener, etc.

*Related caMOD domain objects:* [PartyRole](#)

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

### **6.3.41** [ScreenerApproved](#)

Used to indicate the status of a model that has been approved by a screener.

*Extends:* java.lang.Object

*Implements:* ApprovalStatus, java.io.Serializable

### **6.3.42** [ScreenerAssigned](#)

*Related caMOD domain objects:*

*Extends:* PersistentCaMODBean

*Implements:* ApprovalStatus, java.io.Serializable

### **6.3.43** [ScreenerRejected](#)

*Related caMOD domain objects:*

*Extends:* PersistentCaMODBean

*Implements:* ApprovalStatus, java.io.Serializable

### **6.3.44** [SegmentType](#)

Genetic segment type such as chromosome, contig, CpG islands, repetitive DNA (e.g. Alu, LINE, SINE etc.), gene, breakpoint etc. (see GenomicSegment).

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

#### **6.3.45** [SexDistribution](#)

The observable distribution of phenotypes between sexes.

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

#### **6.3.46** [TargetedModification](#)

Modification of a specific gene (versus one randomly selected) by a genetic engineering technology called gene targeting through homologous recombination; usually achieved using gene targeting vectors.

*Related caMOD domain objects:* [EngineeredGene](#)

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

#### **6.3.47** [Therapy](#)

A defined treatment protocol for testing the efficacy of the treatment on an engineered animal model.

*Related caMOD domain objects:* [TreatmentSchedule](#),

*Other related domain objects:* gov.nih.nci.cabio.bean.Agent

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

#### **6.3.48** [Transgene](#)

A foreign gene that has been integrated into the genome of an animal.

*Related caMOD domain objects:* [IntegrationType](#), [RegulatoryElement](#)

*Other related domain objects:* [gov.nih.nci.cabio.bean.Taxon](#)

*Extends:* [EngineeredGene](#)

*Implements:* GeneInterface, XMLInterface, java.io.Serializable

#### **6.3.49** [TreatmentSchedule](#)

The dosage and regimen for treating cancer in an animal model.

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

#### **6.3.50** [Xenograft](#)

A surgical graft of tissue from one species onto or into individuals of unlike species, genus or family.

*Other related domain objects:* gov.nih.nci.common.domain.EVSOntology,

[gov.nih.nci.cabio.bean.Taxon](#)

*Extends:* PersistentCaMODBean

*Implements:* XMLInterface, java.io.Serializable

## **7.0 THE caCORE MAGE-OM API**

## 7.1 The GEDP Project

The recently completed draft human genome sequence indicates that the human genome comprises only about 30,000 genes—just twice the number as the fly or worm. Yet with these 30,000 genes the human genome generates approximately 90,000 proteins, due to alternative splicing of the original nucleotide sequences. Only a fraction of these genes and their products are currently associated with known function, however, and the further elucidation of their roles in disease processes has become a central focus in today's research.

Microarray technology can provide valuable clues to function via the study of variable expression levels in different cells at various times and while undergoing different processes. For example, increased expression of a gene in a disease state may implicate that gene in the etiology of the disease, while underexpression may indicate that the gene normally provides protection against the disease. More generally, the study of up- and down-regulation of any particular gene among different tissues, and even among individual cells within a tissue, can provide important insight to function.

In the past, molecular biologists were only able to detect and analyze the expression of one or a few genes in one experiment. With the advent of DNA microarray technology, it is now possible to monitor the expression of almost every gene in a given genome on a single chip, and a typical microarray experiment will yield millions of data points.

The massive amount of microarray data being generated today presents a significant challenge for analysis, storage, and exchange of data. The Gene Expression Data Portal ([GEDP](#)), which is part of the NCICB's cancer array informatics project (caARRAY), was developed to address this problem. GEDP is not only a repository for secure storage of researchers' microarray data, it also functions as a depot for the exchange of pre- and post-publication data. The NCICB Applications User Manual describes in greater detail the various user interfaces and analysis tools that are available on the GEDP web sites. This chapter describes the caCORE MAGE-OM application programming interface to the GEDP.

Due to the many complex parameters and variables that must be used in microarray experiments—both before and after data collection—the technology itself has spearheaded new and more rigorous requirements for scientific data sharing. In particular, the ever-present need for agreed-upon standards in representation and content has become critical. Thus, in reporting the results of a microarray experiment, it is equally important that the researcher specify just how and under what conditions these results were obtained.

The [MIAME](#) standard (minimal information for the annotation of a microarray experiment) provides a template for supplemental information and data that must be provided to achieve reproducible and machine-readable microarray expression data. This standard was arrived at by a consensus of hundreds of scientists in cooperation with the Microarray Gene Expression Data ([MGED](#)) Society. While the standard is still evolving, even today many journals endorse it—and indeed, some require it for publication.

The electronic sharing of microarray data is further enhanced by an agreed-upon exchange format known as [MAGE-ML](#), for the XML-encoding of MicroArrayGeneExpression data. MAGE-ML is in turn based on an object model (the [MAGE-OM](#)), which captures the specifications of the MIAME standard. While the MIAME standard specifies a framework for

data reporting, it is the MAGE-OM that captures the logical relations among the terms defined in the standard.

The data model that defines the GEDP database at NCI is derived from the MAGE-OM, but is not identical to it, as the MAGE-OM is still evolving. Like MAGE-OM, the GEDP data model supports all DNA arrays, including spotted and synthesized arrays, and oligo-nucleotide and cDNA arrays—independent of image analysis and/or data normalization methods. The MAGE-OM API to the GEDP database described in Figure 7.1-1 implements a transparent MAGE interface to the GEDP data.

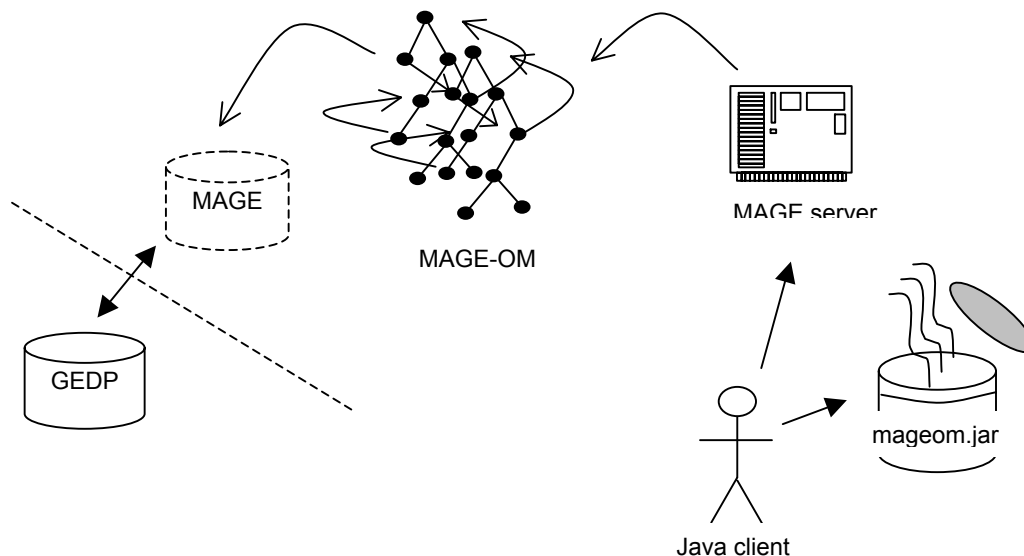


Figure 7.1-1 The caCORE MAGE-OM API to GEDP

## 7.2 The caCORE MAGE-OM API

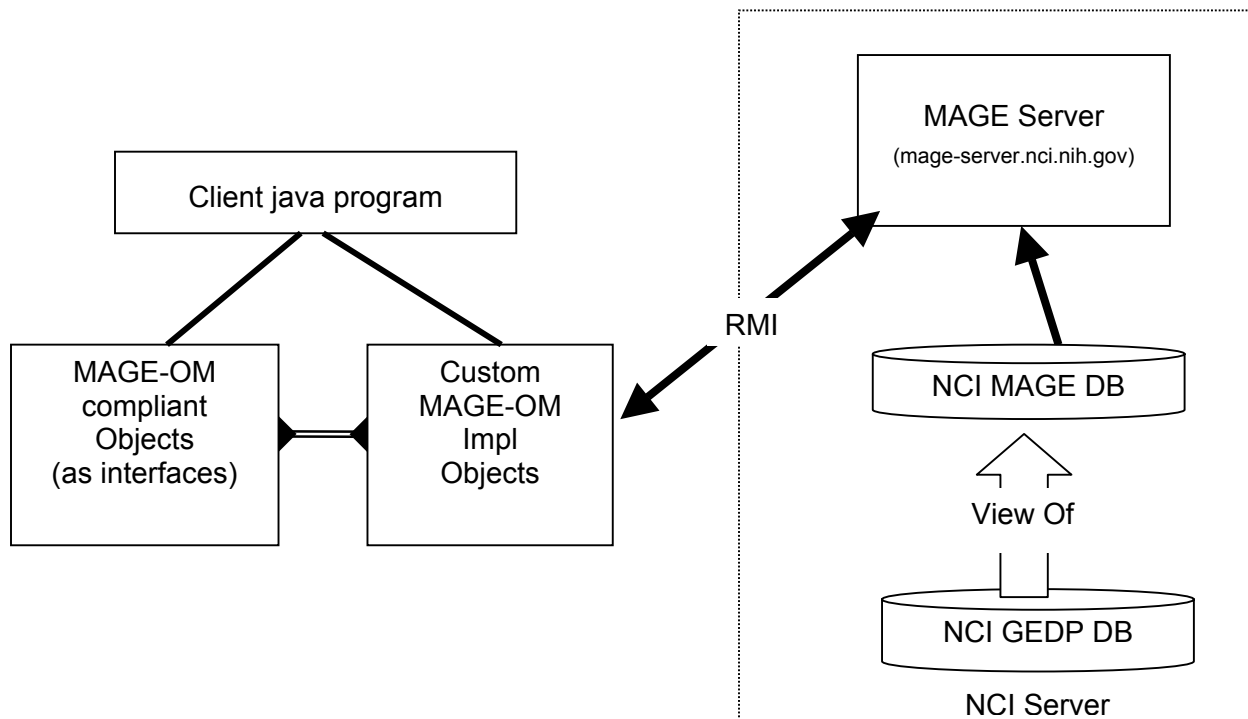
The MAGE-OM API is a set of java objects that adhere to the object model defined by the OMG as Gene Expression v1.0 on 2003-02-03.<sup>6</sup> The MGED Society web site found at <http://www.mged.org/Workgroups/MAGE/mage.html> is also a good source for supplemental material on the MAGE object model.

The caCORE MAGE-OM API objects access data in the GEDP database via remote method invocations issued to a dedicated MAGE server at NCI. There are two primary types of objects defined in the API: (1) MAGE-OM-compliant objects, and (2) custom MAGE-OM *Impl* (implementation) objects. The MAGE-compliant objects are implemented as Java interfaces, which the custom MAGE-OM *Impl* objects implement as concrete java classes.

These interface objects should always be used when strict adherence to the OMG Gene Expression definition is required. Even in these cases, however, the *Impl* classes must first be deployed to extract the data from the GEDP. Once the data have been extracted, the application

<sup>6</sup> See [http://www.omg.org/technology/documents/formal/gene\\_expression.htm](http://www.omg.org/technology/documents/formal/gene_expression.htm).

can use just the interface objects to comply with the standards. Alternatively, the *Impl* objects can be used directly, as this will provide access to several convenient methods and attributes that are not defined in the OMG specification. These enhancements were added to improve both performance and ease of access to certain attributes. Figure 7.2-1 is a high-level view of the API architecture from the java client's perspective.



**Figure 7.2-1 A high-level view of the MAGE-OM API from a client perspective**

A few small examples here illustrate usage of the MAGE-OM API objects. More detailed information is available with the [JavaDocs](#) at:

[ncicb.nci.nih.gov/content/coreftp/MAGE-OM1-0\\_JavaDocs/index.html](http://ncicb.nci.nih.gov/content/coreftp/MAGE-OM1-0_JavaDocs/index.html)

The first example obtains the list of BioAssay components used in a specific experiment (with an id of 25), and prints out whether or not each bioassay is derived or measured. Note that in this example, there is no attempt to first *find* an experiment with Id 25; a new `ExperimentImpl` object is simply instantiated with that Id.

In the second example, a `SearchCriteria` object is enlisted to search for experiments. The caBIO search criteria objects were initially introduced in [Chapter 2](#) and [Chapter 3](#); [Chapter 8](#) discusses these objects and their search mechanisms in more depth. Suffice it for now to say that all of the caBIO domain object classes—that is, all of the classes implemented to represent scientific objects from specific domains—have associated search criteria objects that can be deployed to extract instances of those objects from the databases.

In most cases, the user sets specific criteria, such as “id = 25,” for filtering out irrelevant instances. In this example, however, the resulting query to the database will retrieve *all* experiments, as no selection criteria were specified. Once the results are collected in the `Experiment[]` array, an iterative loop prints the experiment Ids to the screen.

### 7.2.1 Example 1:

```
ExperimentImpl experiment=null;
try {
    experiment = new ExperimentImpl(new Long(25));
} catch(Exception e) {
    System.out.println("Error getting experiment:" + e.getMessage());
    e.printStackTrace();
}

BioAssayImpl[] bioassays = null;
try{
    bioassays = experiment.getBioAssaysImpl();
}catch (Exception e){
    System.out.println("Error getting bioassays from experiment:" +
        e.getMessage());
    e.printStackTrace();
}
    if(bioassays.length > 0)
        System.out.println("Experiment: " + experiment.getId() +
            " returned " + bioassays.length + " bioassays.");}

for(int y = 0; y < bioassays.length; y++) {
    if(bioassays[y] instanceof MeasuredBioAssay) {
        System.out.println("Bioassay #" + y + " is measured, id=" +
            bioassays[y].getId());
    }else if (bioassays[y] instanceof DerivedBioAssay){
        System.out.println("Bioassay #" + y + " is derived, id=" +
            bioassays[y].getId());
    }else{
        System.out.println("Bioassay #" + y + " is unknown type?!?");
    }
}
```

### 7.2.2 Example 2:

```
//get all experiments by using an empty search criteria
ExperimentImplSearchCriteria criteria = new ExperimentImplSearchCriteria();
ExperimentImpl exp = new ExperimentImpl();
SearchResult results=null;
try {
    results = exp.search(criteria);
}catch (ManagerException me){
    System.out.println("Search problem: " + me);
}

Experiment[] experiments = (Experiment[])results.getResultSet();
for ( int i=0; i<100; i++){
    System.out.println("Exp#" + i + " = " + experiments[i].getId());
}
```

## 7.3 Installing a MAGE-OM Java Client

To use the NCI MAGE-OM API, you will first need to download the *mageom.jar* support file from NCI from the [NCICB Download site](#). This jar file contains all of the classes listed in the MAGE JavaDocs pages at NCI. Because remote method invocation (RMI) is used, you will also need to specify a *java.policy* file. This policy file controls RMI security for your local machine.



A sample policy file is provided on the downloads page. This sample is totally permissive, however, as it opens up all access to your client machine. You may want to verify with your network security experts the proper java.policy settings for your site. After the jar file has been downloaded, remember to add the jar file to the classpath for compilation as well as execution.

A sample client program called *MageTest.java* is also discussed in [Chapter 12](#). The source code can be found at the downloads page, and its listing is also included in [Appendix E](#). To compile this program use:

```
javac -classpath mageom.jar MageTest.java
```

To then execute the compiled program use:

```
java -classpath "mageom.jar;." \
-Djava.security.policy=java.policy MageTest <experiment Id>
```

This example takes an experiment Id as an argument; try using 2915 or 3005 as the Id to be sure you get a hit.

## 7.4 The caBIO Bridge to MAGE

One of the most important features of the caCORE MAGE API is its potential to integrate the microarray data stored in the GEDP with other types of information maintained by other databases at NCI. This potential has only begun to be explored; this first release of the MAGE API provides a bridge between the caBIO domain objects and the MAGE-OM API via the *Gene* and *ReporterImpl* objects defined in those packages, respectively. For example, given a *ReporterImpl* object, the following method will display related gene information:

```
public static void getCabioGeneInfo(ReporterImpl theReporter) {
    try {
        Long theId = theReporter.getId();
        if ( theId == null ) return;

        GeneSearchCriteria caCrit = new GeneSearchCriteria();
        caCrit.setExpressionMeasurementId(theId);
        gov.nih.nci.caBIO.bean.SearchResult caRes =
            (new Gene()).search(caCrit);
        Gene[] caGenes = (Gene[]) caRes.getResultSet();

        for (int ig = 0; ig < caGenes.length; ig++) {
            System.out.println(
                "caBIO Gene #" + ig + " id = " + caGenes[ig].getId());
            System.out.println(
                "caBIO Gene #" + ig + " nam= " + caGenes[ig].getName());
        }
    } catch (Exception e) {
        System.out.println("Error getting cabio gene info");
        e.printStackTrace();
    }
}
```

The bridge between these two domains is obtained by setting the *ExpressionMeasurementId* attribute of the *GeneSearchCriteria* object to the “id” feature of the *ReporterImpl* object. A more complete example incorporating this method is listed in Appendix F. Note that in the complete example, the *getCabioGeneInfo()* method is invoked in two places. In the first case, the input argument is a *MeasuredBioAssayDataImpl* object; in the second case, the input parameter is a *DerivedBioAssayDataImpl* object. As both of these are different types of reporter objects, they are subclasses to the *ReporterImpl* class and, thus, can be used in this context.

## **8.0 SEARCH CRITERIA OBJECTS AND THE caCORE APIs**

The caCORE infrastructure comprises an *n*-tiered architecture, as depicted in Figure 8-1. The architecture and APIs are provided by caBIO. Thus, caBIO represents the system architecture and public programming interfaces to caCORE. At the backend are various local databases, flat files, and URLs to external databases and public web sites. At the front end is a Presentation Layer providing APIs capable of supporting a wide variety of programming languages. At the heart of the caCORE architecture are the classes that comprise the caBIO Object Layer. To this point most of these discussions have been concerned with the various domain objects in the *bean* packages. This chapter turns to the *SearchCriteria* objects that are associated with these domain objects.

Although the complete caCORE implementation includes a number of packages, this discussion is limited to the *bean*, *das*, *manager*, *servlet*, *search*, *util*, and *webservices* packages that together implement the APIs. The focus in particular is on the driving force behind these APIs—the Search/Retrieve paradigm, and how the caBIO search criteria objects are used to realize the underlying design.

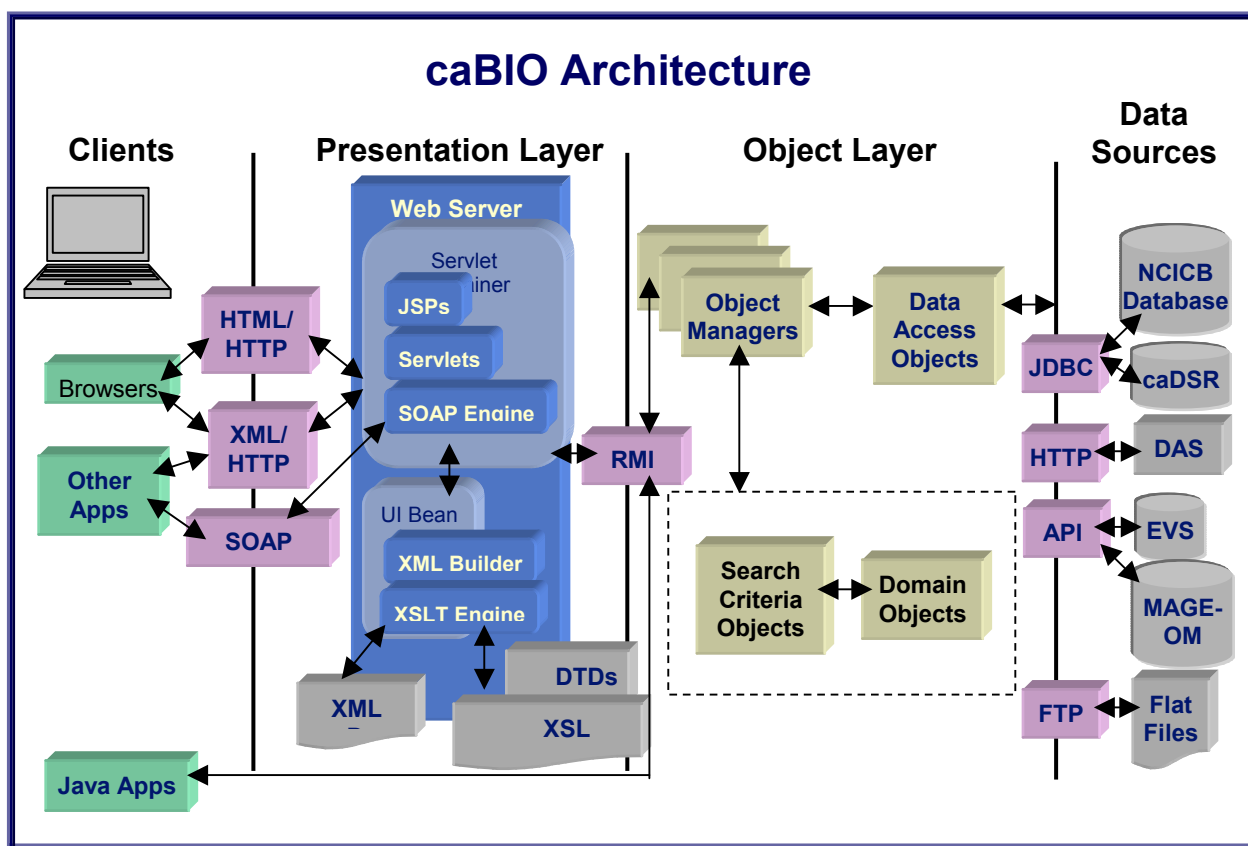


Figure 8-1 The caBIO architecture

As depicted in Figure 8-1, the classes in the object layer fall into four primary groupings: (1) domain objects that represent terminological, bioinformatic, clinical, microarray, and cancer image data, (2) the search criteria objects associated with these domain objects, (3) managerial objects supporting the infrastructure, and (4) data access objects interfacing to the backend data sources.

Java applications send requests directly to the Object Layer using RMI, effectively bypassing the Presentation Layer. All other applications require some type of interface, such as the SOAP Engine or the Java server pages (JSPs) provided by the Presentation Layer. The next section considers first how the Java API operates, as these same processes must execute on the backend of all other interfaces.

## 8.1 The Java API Search/Retrieve Paradigm

The driving force behind the caCORE design is a data-mining paradigm, and the basic operation is a database search and retrieval, using a newly instantiated domain object as the target to be populated with the data that are returned. All of the domain objects are contained in the *bean* packages. In addition to the methods that are particular to the individual classes, all domain objects also inherit the *search()* method. Each domain object has an associated search criteria object, which is the sole argument to the domain object's *search()* method. For example, corresponding to the *Gene* domain object, there is a *GeneSearchCriteria* object, and the syntax of the *Gene*'s search method is:

```
myGene.search(myGeneSearchCriteria)
```

Each search criteria object has attributes that are used to constrain the search and define the type of information that will be returned in the result set. The desired values for a query are defined using an appropriate *setXxx()* method of the search criteria object. For example, the sequence of statements:

```
Gene myGene = new Gene();
GeneSearchCriteria myGeneSearchCriteria = new GeneSearchCriteria();
MyGeneSearchCriteria.setSymbol("pTEN");
SearchResult result = myGene.search(myGeneSearchCriteria);
```

is effectively equivalent to the following SQL query:

```
Select Gene from <database table> where symbol = 'PTEN'
```

Many object-specific attributes are settable for the *SearchCriteria* objects associated with specific domain objects. For example, the *GeneSearchCriteria* object includes methods for specifying the desired gene symbol, chromosome id, organism, etc.

In addition to attributes that are particular to the associated domain objects, the object-specific search criteria classes inherit methods and attributes from a generic *SearchCriteria* object that constrain *how* the information will be returned. Table 8.1-1 lists some of the more important inherited methods.

**Table 8.1-1. Common methods implemented by all *SearchCriteria* objects**

Method	Description
<i>setMaxRecordset()</i>	Sets the maximum number of result objects to return (default 1,000)
<i>setOrderBy()</i>	Sets the order by clause for the SQL
<i>setReturnCount()</i>	Specifies that the number of objects found should be included in the search result (default false)

<i>setReturnObjects()</i>	Specifies that the objects themselves should be included in the search result (default true)
<i>setStartAt()</i>	Sets the number of the first member of the result array

The return value of a domain object's *search()* method is always an object of type *SearchResult*, whose attributes and methods approximately mirror the attributes and methods of the generic *SearchCriteria* object. For example, the *SearchResult* object has a method *getCount()* that returns the number of objects that matched the specified criteria. This method's return value is only defined, however, when the associated *SearchCriteria* object specifies that the *number* of objects found should be included in the search result.

Similarly, the *SearchResult* object's method *getResultSet()* returns an array of objects only if the *SearchCriteria* specifies that the objects themselves should be included in the search result. Setting this last option to *false* is useful in situations where the only information that is needed is the count of objects satisfying the criteria, and not the objects themselves. By default, *setReturnCount()* is *false* and *setReturnObjects()* is true, unless these options are explicitly reset using these methods.

The *SearchResult* object's methods *getStartsAt()* and *getEndsAt()* return the array index of the first and last objects in the result array, respectively. While these methods might seem at first to be gratuitous, they are actually a critical part of the caBIO API design, which provides a "throttling" mechanism to limit the number of results returned on any single query. By default, the maximum number is 1000, but, as indicated here, this can be reset to a smaller or larger number as desired.

These same mechanisms are implemented in the subsequent API layers providing interfaces to other programming languages. Given the enormous amount of currently available bioinformatic data and their exponential rate of growth, this ability to receive large amounts of data in bursts is indispensable.

Several additional methods further facilitate situations where a very large set of objects match the search criteria. The *SearchResult* object's method *hasMore()* returns *true* when further results are available that are not included in the current *SearchResult*. In this case, the *SearchResult* object's method *getNextCriteria()* can be used to return a new *SearchCriteria* object whose starting index picks up where the previous result set left off. Alternatively, the user can use the *setStartAt()* method to control the starting index of the result set.

Other methods provided by the generic *SearchCriteria* object provide means of determining whether or not a specific criterion has been defined, removing previously set criteria, and/or adding new criteria to the current collection.

### 8.1.1 Constructing More Complex Queries

By default, all of the domain object-specific attributes which have been set for a search criteria object are treated as a logical conjunction. For example, if both the *setGoOntologyId()* and *setKeyword()* methods have been applied to a *GeneSearchCriteria* object, then only those genes satisfying both of these criteria will be returned.

This is accomplished using a *CriteriaElement* helper class defined in the *util* package. For example, the expressions:

```
MyGeneSearchCriteria.setSymbol("pTEN");  
MyGeneSearchCriteria.setKeywords("mouse");
```

are interpreted as

```
MyGeneSearchCriteria.setSymbol("pTEN",  
    gov.nih.caBIO.util.CriteriaElement.AND)  
MyGeneSearchCriteria.setSymbol("pTEN",  
    gov.nih.caBIO.util.CriteriaElement.AND)
```

This default behavior can however, be overridden, using the *CriteriaElement*'s *OR* element in place of *AND*. By default, all searches will be performed in a case-insensitive manner. So in the above example, the search strings "pten", "PTEN", and "pTeN" would all produce the same results. The *CriteriaElement* helper class also provides to override this behavior:

```
MyGeneSearchCriteria.setSymbol("pTEN",  
    gov.nih.caBIO.util.CriteriaElement.CASESENSITIVE);
```

This will force the search engine to return only case sensitive matches.

*SearchCriteria* objects can also be embedded in one another to specify more complex queries, via the *putSearchCriteria()* method. For example, to extract the set of all pathways containing the gene PTEN, one can:

1. Create a *GeneSearchCriteria()*, *myGenesCriteria*.
2. Invoke: *myGenesCriteria.SetSymbol("PTEN")*.
3. Create a *PathwaySearchCriteria()*, *myPathsCriteria*.
4. Invoke *myPathsCriteria.putSearchCriteria(myGenesCriteria)*.
5. Create a *Pathway* domain object, *myPathway*.
6. Invoke *myPathway.search(myPathsCriteria)*.

The result of step 6 will produce an array of pathways containing the PTEN gene. This example demonstrates another feature of the caBIO design: Each domain object also serves as a factory for creating multiple instances of that object. Reviewing the steps outlined above allows one to generalize the factory process as follows:

1. Instantiate a *new* domain object of the desired type.
2. Create a *new SearchCriteria* for that object, and set its attributes.
3. Execute the domain object's *search()* method on that *SearchCriteria*, and store the results in a generic *SearchResult* object.
4. Invoke the *getResultSet()* method on the *SearchResult* object and typecast its return value to an array of the same type as the original domain object.

Advanced users may want to take advantage of the enhanced search capabilities which are defined in the search packages. These are somewhat complicated, and a separate chapter ([10](#)) is provided for that discussion. Included in the new objects, features, and methods defined in the advanced search interface is an extension of the *putSearchCriteria()* called *putCriteria()*. This is also a method implemented by the search criteria classes, and can be used to "put" an *array* of criteria values, as in:

```
List geneNames = new ArrayList();
```

```

geneNames.add( "PTEN" );
geneNames.add( "TP53" );
geneNames.add( "BRCA1" );
GeneSearchCriteria gsc = new GeneSearchCriteria();
gsc.putCriteria( "name", geneNames );

```

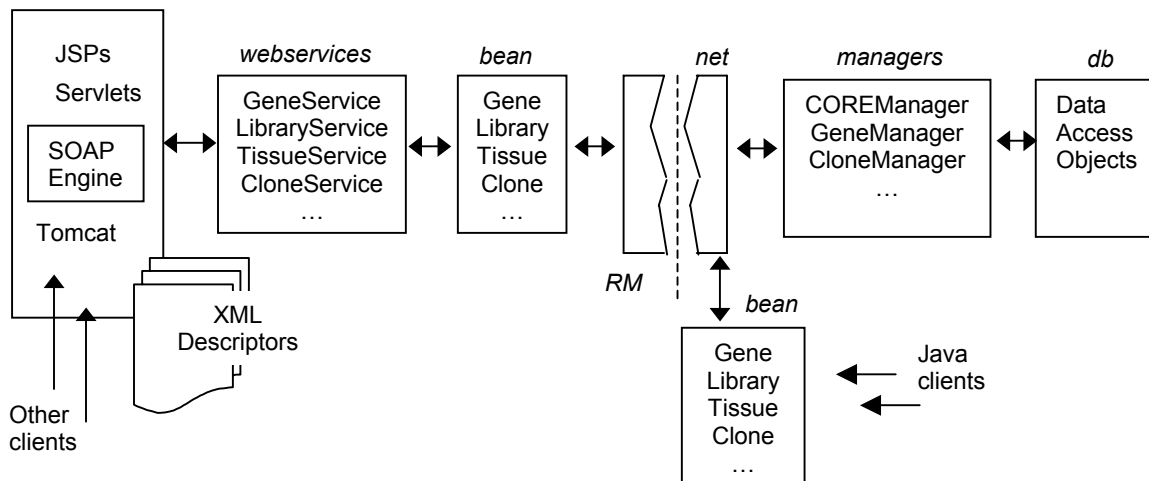
The usage of this method is demonstrated further in [Chapter 10](#); the remainder of this chapter considers how the basic domain object/search criteria paradigm is implemented and how it is used by the various Java clients and interfaces.

## 8.2 How the caBIO Search Paradigm Operates

The installation of the caBIO software for a Java client (see [Chapter 11](#)) includes a Java archive file (*cabio.jar*) defining all of the domain objects as well as the protocols and server information required to issue RMI requests to the caBIO servers. The logistics of retrieving data for a Java client are as follows. First, the Java client application declares a new instance of the object type of interest. For example, the client executes the statement:

```
Gene myGene = New Gene ();
```

This instantiation of the new *Gene* object alerts the *GeneManager* on the caBIO server, and causes a proxy for that manager to become resident on the client machine for the duration of the application. All RMI requests on the *Gene* object from that point forward will be handled by the protocols defined for the manager and proxy objects.



**Figure 8.2-1** The logical deployment of the caBIO packages for data retrieval

If a subsequent request such as *myGene.getSequences()* is issued, the instantiation of the resulting *Sequence* objects will in turn be handled by a remote *SequenceManager* and a local proxy. A logical view of the caBIO object managers is available in the [caBIO Object Model](#) pages, and their specifications can be viewed on the [JavaDocs](#) pages. By virtue of the object managers, the network details of the communication to the caBIO server are abstracted away from the Java developer. Figure 8.2-1 is an alternative representation of the caBIO architecture, emphasizing how some of the Java packages are deployed to implement the different APIs.

A general rule of thumb for all domain objects is that only those attributes represented by primitive data types (e.g., integer, string, float, etc.) are returned directly with a retrieved object.

For more complex types such as objects and arrays, access to these entities is provided, but the objects themselves are not. Instead, the “containing” object provides methods for retrieving these embedded objects recursively. Thus, for example, a *Gene* object provides a method called *getSequences()* to retrieve its associated genomic sequences once the *Gene* itself has been obtained.

Similar mechanisms for returning only top-level information — with the option of drilling down further where desired — are implemented in the SOAP and HTTP interfaces described below. The SOAP and HTTP interfaces also provide mechanisms for controlling the number of objects returned for a single query, corresponding to the *SearchCriteria* object’s methods *setMaxRecordset()* and *setStartAt()*.

An important difference between the Java API and other interfaces to caBIO is that only Java applications have direct access to the domain objects and their methods, as they are defined in a local jar file.

### 8.3 The SOAP API

caBIO’s SOAP API is provided for non-Java applications written in languages such as Perl, C, Python, etc. SOAP is a lightweight XML-based protocol for the exchange of information in a decentralized, distributed environment. It consists of an envelope that describes the message and a framework for message transport. caBIO uses the open-source Apache SOAP package, in combination with Jakarta Tomcat, to provide its web services to users.

It is up to the application developer to select and install a SOAP client for the development environment. [Chapter 13](#) provides a more in-depth discussion of the SOAP API, and includes an example of using SOAP::Lite for Perl. This section considers how the SOAP services also implement a Search/Retrieve paradigm, using the Java search criteria objects on the backend.

All SOAP requests issued by clients are formatted as XML documents that are posted to the caBIO server at a listening port reserved for SOAP requests. All of the caBIO domain objects are capable of serializing themselves to XML, and the response returned by the server is also an XML document. These domain objects, however, do not directly receive the SOAP requests, as they are first parsed by objects in the Presentation Layer.

Specifically, the caBIO SOAP server receives requests from the remote SOAP client, and forwards these to an appropriate class in the *webservices* package. Each SOAP web service defined in the *webservices* package has methods mirroring those defined for a corresponding domain object in the *bean* package. For example, the SOAP *GoOntologyService* has methods called *getChildRelationships()*, *getParentRelationships()*, *getHomoSapienGenes()*, and *getMouseGenes()*, corresponding to the *GoOntology* domain object’s methods of the same name.

The SOAP service classes inherit methods from a single parent class—*WebCriteriaInterpreter*, whose two methods are *readInCriteria()* and *searchObjects()*. Each of these methods takes two arguments:

- (1) a hashtable of tag/value pairs, and
- (2) a string specifying the object type to search for.

The return value of *readInCriteria()* is a *SearchCriteria* object; the return value of *searchObjects()* is a *SearchResult* object. Thus, the SOAP service classes in the *webservices* package can use the parent class’s methods, in combination with their own method



specializations, to transform the incoming HTTP requests to equivalent Java method invocations that can be passed on to domain and search criteria objects in the *bean* package.

Once the results of the search have been obtained from the back-end data sources, the domain objects' *toXML()* methods are applied to return XML-encoded responses to the SOAP service classes in the *webservices* package. There, the service classes can forward these responses to the SOAP server, which, in turn, forwards these as strings to the SOAP client.

The SOAP API provides a throttling mechanism that is similar to that described above for the Java API. In the SOAP API, this is achieved by using the XML Linking Language (XLink), which effectively provides a way of embedding “pointers” in the XML output, thus reducing the amount of information returned. As with the Java API, all attributes represented by simple data types (i.e., numbers and strings) are included directly in the SOAP output. More complex data types, such as structured objects and arrays, are returned as XLinks providing URLs where the data can be fetched recursively.

Two additional parameters can be included in the SOAP request, however, which specify that the XML-encoded response should also expand either all or only selected XLinks. These additional parameters are *returnHeavyXML*, which takes the values 0/1, and *fillInObjects*, which takes a list of comma-separated arguments specifying the Xlink tags that should be expanded. The *returnHeavyXML*, when set to true, opens up all of the embedded XLinks one level deep. Specific instructions on how to use these parameters are provided in [Chapter 13](#).

## 8.4 The HTTP Interface

The Hypertext Transfer Protocol provides a non-programmatic interface for accessing caBIO data. Using the HTTP interface does not require any additional software other than a web browser such as Internet Explorer or Netscape Communicator. Like the SOAP API, the HTTP interface uses the domain and search criteria objects in the Object Layer to communicate with the backend data sources. The HTTP interface forwards its requests as URLs to a Java servlet running in the Presentation Layer, called *getXML*. The *getXML* servlet is defined in caBIO's *servlet* package, and has methods that receive requests from HTTP clients, forward messages to the respective domain objects for processing, and return the results as XML-encoded responses to the HTTP clients.

The HTTP request parameters correspond to methods in the respective search criteria object associated with the domain object being queried. One can narrow down a search by using as many parameters as required. The XML output returned from an HTTP request is similar to the XML output from a SOAP client request. The XML output received by the HTTP client also embeds XLinks to limit the amount of information returned in a single response.

As with the SOAP API, the HTTP interface allows the user to further expand these XLinks by using *returnHeavyXML* and *fillInObjects* as additional parameters in the HTTP request. The number of top-level objects returned can also be controlled by specifying values for *StartAt* and *EndAt* in the HTTP request or, alternatively, by using the *ResultCount* parameter. Details and examples of using the HTTP interface are provided in [Chapter 14](#).

## 8.5 Summary of Search Controls in the Different APIs

The SOAP and HTTP interfaces are described in more detail in subsequent chapters, but it is useful at this point to compare the generic controls that are available in these interfaces with

those provided by the Java API. Table 8.1-1 summarized the generic *methods* for controlling the results in the Java API. In the SOAP and HTTP interfaces, *attributes*, specified as <tag=value> pairs, are used in place of methods. Table 8.5-1 summarizes the attribute names that control various aspects of the results for the different interfaces.

**Table 8.5-1 Attributes controlling the way results are returned**

Functionality	Attribute	Value type	Java	SOAP	HTTP
Starting element	startAt	Integer	✓		✓
	resultStart	Integer		✓	✓
Number of results	maxRecordset	Integer	✓		✓
	resultCount			✓	✓
Drill down (all)	fillInAllObjects	Boolean	✓		✓
	returnHeavyXML	Boolean		✓	✓
Drill down (selective)	fillInObjects	List of objects	✓	✓	✓
Ordering of results	orderBy	String	✓	✓	✓

As described in [Chapter 14](#), the HTTP interface currently provides two different protocols for issuing an HTTP request. The *operation=* syntax, which was introduced in caCORE 1.0, conforms to the SOAP syntax in its attribute names; the new *query=* syntax conforms to the Java API attributes. The HTTP *operation=* syntax is deprecated, and in future releases of caCORE, the SOAP attributes will also conform to the Java API attributes.

## 8.6 The caBIO SearchCriteria Catalog

As described in [Section 8.1](#), it is possible to set the desired attributes of one *SearchCriteria* object and, subsequently, embed that object in a second *SearchCriteria* object using the *putSearchCriteria()* method.

However, not all *SearchCriteria* are “compatible” with one another. Table 8.6-1 lists the search criteria objects implementing the *putSearchCriteria()* along with the related search criteria objects that can be used as arguments to that method.

**Table 8.6-1 caBIO *putSearchCriteria* arguments**

<u>SearchCriteria #1</u>	<u>(accepts) SearchCriteria #2</u>
AgentSearchCriteria	ClinicalTrialProtocolSearchCriteria, TargetSearchCriteria
AnomalySearchCriteria	TargetSearchCriteria, HistopathologySearchCriteria, VocabularySearchCriteria
ChromosomeSearchCriteria	TaxonSearchCriteria, GeneSearchCriteria
ClinicalTrialProtocolSearchCriteria	AgentSearchCriteria, DiseaseSearchCriteria, ProtocolAssociationSearchCriteria
CloneSearchCriteria	LibrarySearchCriteria, SequenceSearchCriteria, SNPSearchCriteria
CMAPOntologyRelationshipSearchCriteria	CMAPOntologySearchCriteria
CMAPOntologySearchCriteria	GeneSearchCriteria, CMAPOntologyRelationshipSearchCriteria,

SearchCriteria #1

DiseaseRelationshipSearchCriteria  
DiseaseSearchCriteria  
  
ExpressionFeatureSearchCriteria  
ExpressionMeasurementArraySearchCriteria  
ExpressionMeasurementSearchCriteria  
  
GeneAliasSearchCriteria  
GeneHomologSearchCriteria  
GeneSearchCriteria  
  
GoOntologyRelationshipSearchCriteria  
GoOntologySearchCriteria  
  
HistopathologySearchCriteria  
MapLocationSearchCriteria  
  
OrganRelationshipSearchCriteria  
OrganSearchCriteria  
  
PathwaySearchCriteria  
ProteinHomologSearchCriteria  
ProtocolAssociationSearchCriteria  
ProtocolSearchCriteria  
SequenceSearchCriteria  
SNPSearchCriteria  
TargetSearchCriteria  
  
TaxonSearchCriteria  
TissueSearchCriteria

*(accepts)* SearchCriteria #2

CMAPOntologySearchCriteria  
DiseaseSearchCriteria  
HistopathologySearchCriteria,  
ClinicalTrialProtocolSearchCriteria,  
DiseaseRelationshipSearchCriteria, DiseaseSearchCriteria  
OrganSearchCriteria  
ExpressionMeasurementSearchCriteria  
GeneSearchCriteria, SequenceSearchCriteria,  
ExpressionMeasurementArraySearchCriteria  
GeneSearchCriteria  
GeneSearchCriteria  
ChromosomeSearchCriteria,  
CMAPOntologySearchCriteria,  
ExpressionFeatureSearchCriteria,  
GoOntologySearchCriteria, GeneAliasSearchCriteria,  
GeneHomologSearchCriteria,  
MapLocationSearchCriteria, OrganSearchCriteria,  
PathwaySearchCriteria, ProteinSearchCriteria,  
SequenceSearchCriteria, SNPSearchCriteria,  
TargetSearchCriteria, TaxonSearchCriteria  
GoOntologySearchCriteria  
GeneSearchCriteria,  
GoOntologyRelationshipSearchCriteria,  
GoOntologySearchCriteria  
TissueSearchCriteria, ProtocolSearchCriteria  
TaxonSearchCriteria, GeneSearchCriteria,  
ProteinHomologSearchCriteria  
OrganSearchCriteria  
OrganRelationshipSearchCriteria, OrganSearchCriteria,  
GeneSearchCriteria, ExpressionFeatureSearchCriteria  
TaxonSearchCriteria, GeneSearchCriteria  
ProteinSearchCriteria  
ClinicalTrialProtocolSearchCriteria  
LibrarySearchCriteria  
CloneSearchCriteria  
GeneSearchCriteria, CloneSearchCriteria  
AgentSearchCriteria, AnomalySearchCriteria,  
GeneSearchCriteria  
GeneSearchCriteria  
TaxonSearchCriteria, ProtocolSearchCriteria,  
LibrarySearchCriteria

Column 1 in Table 8.6-1 lists those SearchCriteria that support the *putSearchCriteria()* method; column 2 lists the secondary SearchCriteria objects that can be provided as arguments. This list is dynamic and new entries are added as necessary.

For example, the following code snippet will retrieve all pathways containing genes whose symbols match the string “vegf”:

```
// define the criteria for the genes we are interested in:
GeneSearchCriteria GeneCriteria = new GeneSearchCriteria();
GeneCriteria.setSymbol("vegf");

// now define the criteria for pathways, and embed the gene criteria:
PathwaySearchCriteria PathCriteria = new PathwaySearchCriteria();
PathCriteria.PutSearchCriteria(GeneCriteria);

// create a pathway object and invoke its search method:
Pathway myPath = new Pathway();
SearchResult result = MyPath.Search(PathCriteria);
if (result != null){
    Pathway[] myPaths = (Pathway[]) result.getResultSet();
    // ... do something interesting with the paths
}
```

### 8.6.1 The caBIO SearchCriteria-Attribute Map

For convenience, this section summarizes the object-specific settable attributes of the various search criteria, which can be used to narrow the search for an associated class of objects. Each of these attributes is a private data member of the class, but is settable via the *set* method of the same name. In addition to these object-specific attributes, each search criteria object also implements the *setOrderBy()* method, which controls the order in which the results are returned.

#### ***AgentSearchCriteria***

agentNSCNumber	java.lang.Long
clinicalTrialProtocolId	java.lang.Long
comment	java.lang.String
evsId	java.lang.String
isCMAPAgent	java.lang.Boolean
name	java.lang.String
source	java.lang.String
targetId	java.lang.Long
therapyId	java.lang.Long

#### ***AnomalySearchCriteria***

anomalyDescription	java.lang.String
contextCode	java.lang.Long
HistopathologyId	java.lang.Long
organId	java.lang.Long
targetId	java.lang.Long

#### ***ChromosomeSearchCriteria***

name	java.lang.String
------	------------------

### ***ClinicalTrialProtocolSearchCriteria***

agent	java.lang.String
agentId	java.lang.Long
conceptId	java.lang.String
ctepName	java.lang.String
diseaseCategory	java.lang.String
diseaseId	java.lang.Long
diseaseName	java.lang.String
documentNumber	java.lang.String
imtCode	java.lang.Long
leadOrganizationId	java.lang.String
leadOrganizationName	java.lang.String
nihAdminCode	java.lang.String
pdqIdentifier	java.lang.String
phase	java.lang.String
piName	java.lang.String
protocolAssociationId	java.lang.Long
title	java.lang.String
treatmentFlag	java.lang.String

### ***CloneSearchCriteria***

geneId	java.lang.Long
name	java.lang.String
sequenceId	java.lang.Long
snpId	java.lang.Long
verified	java.lang.Boolean

### ***CMAPOntologySearchCriteria***

CMAPOntologyChildId	java.lang.Long
CMAPOntologyGeneId	java.lang.Long
CMAPOntologyName	java.lang.String
CMAPOntologyOntologyId	java.lang.Long
CMAPOntologyParentId	java.lang.Long
includeBoth	java.lang.Boolean
includeParents	java.lang.Boolean
includeChildren	java.lang.Boolean
relationshipParentId	java.lang.Long
relationshipChildId	java.lang.Long
relationshipType	java.lang.Long

### ***CMAPOntologyRelationshipSearchCriteria***

relationshipChildId	java.lang.Long
relationshipParentId	java.lang.Long
relationshipType	java.lang.String

<b><i>ConsensusSequenceSearchCriteria</i></b>	
consensusSequenceType	java.lang.String
contigId	java.lang.Long
geneId	java.lang.Long
proteinId	java.lang.Long
refGeneId	java.lang.Long
<b><i>ContigSearchCriteria</i></b>	
name	java.lang.String
sequenceId	java.lang.Long
<b><i>DiseaseSearchCriteria</i></b>	
diseaseId	java.lang.Long
histopathologyId	java.lang.Long
geneId	java.lang.Long
includeBoth	java.lang.Boolean
includeParents	java.lang.Boolean
includeChildren	java.lang.Boolean
relationshipParentId	java.lang.Long
relationshipChildId	java.lang.Long
relationshipType	java.lang.Long
<b><i>DiseaseRelationshipSearchCriteria</i></b>	
relationshipChildId	java.lang.Long
relationshipParentId	java.lang.Long
relationshipType_attribute	java.lang.String
<b><i>EstExperimentSearchCriteria</i></b>	
contextId	java.lang.Long
expressionFeatureId	java.lang.Long
gene	java.lang.String
geneId	java.lang.Long
organ	java.lang.String
proteinId	java.lang.Long
taxonId	java.lang.Long
threshold	java.lang.Float
type	java.lang.String
<b><i>ExpressionExperimentSearchCriteria</i></b>	
expressionFeatureId	java.lang.Long
gene	java.lang.String
geneId	java.lang.Long
organ	java.lang.String
proteinId	java.lang.Long
taxonId	java.lang.Long
threshold	java.lang.Float
type	java.lang.String

***ExpressionFeatureSearchCriteria***

expressionLevelDescId	java.lang.Long
geneId	java.lang.Long

***ExpressionLevelDescSearchCriteria***

name	java.lang.String
------	------------------

***ExpressionMeasurementSearchCriteria***

accessionNumber	java.lang.String
expressionMeasurementArrayId	java.lang.Long
geneId	java.lang.Long
name	java.lang.String
sequenceId	java.lang.Long

***ExpressionMeasurementArraySearchCriteria***

accessionNumber	java.lang.String
expressionMeasurementId	java.lang.Long
name	java.lang.String

***GeneSearchCriteria***

allPathwayId	java.lang.Long
bcId	java.lang.String
chromosomeId	java.lang.Long
cloneName	java.lang.String
cMAPOntologyId	java.lang.Long
cytogenicLocation	java.lang.String
expressedPathwayId	java.lang.Long
expressionMeasurementId	java.lang.Long
genBankAccessionNumber	java.lang.String
geneKeyword	java.lang.String
geneNameKeyword	java.lang.String
goOntologyHomoSapienId	java.lang.Long
goOntologyId	java.lang.Long
goOntologyMouseId	java.lang.Long
keyword	java.lang.String
mutatedGenePathwayId	java.lang.Long
organism	java.lang.String
overExpressedPathwayId	java.lang.Long
PathwayId	java.lang.Long
symbol	java.lang.String
targetId	java.lang.Long
taxonId	java.lang.Long
tissueType	java.lang.String
underExpressedPathwayId	java.lang.Long
unigeneClusterId	java.lang.String
uniqueIdentifier	java.lang.String

***GeneAliasSearchCriteria***

description	java.lang.String
-------------	------------------

geneId	java.lang.Long
type	java.lang.String
<b><i>GeneHomologSearchCriteria</i></b>	
geneId	java.lang.Long
<b><i>GoOntologySearchCriteria</i></b>	
geneId	java.lang.Long
includeBoth	java.lang.Boolean
includeParents	java.lang.Boolean
includeChildren	java.lang.Boolean
relationshipParentId	java.lang.Long
relationshipChildId	java.lang.Long
relationshipType	java.lang.Long
<b><i>GoOntologyRelationshipSearchCriteria</i></b>	
relationshipChildId	java.lang.Long
relationshipParentId	java.lang.Long
relationshipType_attribute	java.lang.String
<b><i>HistopathologySearchCriteria</i></b>	
animalModelId	java.lang.Long
comments	java.lang.String
dataType	java.lang.String
diseaseId	java.lang.Long
grossDescription	java.lang.String
metastasisOf	java.lang.Long
microscopicDescription	java.lang.String
organId	java.lang.Long
relationalOperation	java.lang.String
survivalInfo	java.lang.String
tumorIncidenceRate	java.lang.Float
tumorType	java.lang.String
<b><i>LibrarySearchCriteria</i></b>	
geneId	java.lang.Long
libraryGroup	java.lang.String
libraryName	java.lang.String
libraryProtocol	java.lang.String
organism	java.lang.String
sortOrder	java.lang.String
tissueHistology	java.lang.String
tissueName	java.lang.String
tissuePreparation	java.lang.String
tissueType	java.lang.String
<b><i>MapLocationSearchCriteria</i></b>	
geneId	java.lang.Long
location	java.lang.String



type	java.lang.String
<b><i>OntologySearchCriteria</i></b>	
includeBoth	java.lang.Boolean
includeChildren	java.lang.Boolean
includeParents	java.lang.Boolean
name	java.lang.String
relationshipParentId	java.lang.Long
relationshipChildId	java.lang.Long
relationshipType	java.lang.Long
<b><i>OrganSearchCriteria</i></b>	
anomalyId	java.lang.Long
expressionFeatureId	java.lang.Long
diseaseId	java.lang.Long
histopathologyId	java.lang.Long
geneId	java.lang.String
includeBoth	java.lang.Boolean
includeChildren	java.lang.Boolean
includeParents	java.lang.Boolean
name	java.lang.String
relationshipParentId	java.lang.Long
relationshipChildId	java.lang.Long
relationshipType	java.lang.Long
<b><i>OrganRelationshipSearchCriteria</i></b>	
relationshipChildId	java.lang.Long
relationshipParentId	java.lang.Long
relationshipType	java.lang.String
<b><i>PathwaySearchCriteria</i></b>	
bioProcessId	java.lang.Long
context	java.lang.String
displayValue	java.lang.String
geneId	java.lang.Long
name	java.lang.String
pathwayDescription	java.lang.String
pathwayDiagram	java.lang.String
taxonId	java.lang.Long
<b><i>ProteinSearchCriteria</i></b>	
accessionNumber	java.lang.String
description	java.lang.String
geneId	java.lang.Long
<b><i>ProteinHomologSearchCriteria</i></b>	
proteinId	java.lang.Long
<b><i>ProtocolSearchCriteria</i></b>	
name	java.lang.String

***ProtocolAssociationSearchCriteria***

clinicalTrialProtocolId	java.lang.Long
protocolId	java.lang.Long

***ReadSequenceSearchCriteria***

CloneId	java.lang.Long
GeneId	java.lang.Long
proteinID	java.lang.Long
ReadSequenceId	java.lang.String
refGeneId	java.lang.Long
tracefileID	java.lang.Long

***RelationshipSearchCriteria***

RelationshipChildId	java.lang.Long
RelationshipParentId	java.lang.Long
relationshipType	java.lang.String

***SAGEExperimentSearchCriteria***

contextId	java.lang.Long
expressionFeatureId	java.lang.Long
gene	java.lang.String
geneId	java.lang.Long
organ	java.lang.String
proteinId	java.lang.Long
taxonId	java.lang.Long
threshold	java.lang.Float
type	java.lang.String

***SequenceSearchCriteria***

accessionNumber	java.lang.String
accessionNumberVersion	java.lang.String
cloneId	java.lang.Long
contigId	java.lang.Long
expressionMeasurementId	java.lang.Long
geneId	java.lang.Long
isRefSeq	java.lang.Boolean
refGeneId	java.lang.Long
returnDNA	java.lang.Boolean
sequenceType	java.lang.String

***SNPSearchCriteria***

geneId	java.lang.Long
--------	----------------

***TargetSearchCriteria***

agentId	java.lang.Long
anomalyDescription	java.lang.String
anomalyId	java.lang.Long
cancerType	java.lang.String
conceptID	java.lang.Long

geneId	java.lang.Long
<b><i>TaxonSearchCriteria</i></b>	
abbreviation	java.lang.String
animalModelId	java.lang.Long
chromosomeId	java.lang.Long
commonName	java.lang.String
isPreferred	java.lang.Boolean
name	java.lang.String
regulatoryElementId	java.lang.Long
scientificName	java.lang.String
strainId	java.lang.Long
xenograftId	java.lang.Long
<b><i>TissueSearchCriteria</i></b>	
libraryId	java.lang.Long
<b><i>TraceFileSearchCriteria</i></b>	
cloneId	java.lang.Long
name	java.lang.String
snpld	java.lang.Long

## 8.7 The EVS SearchCriteria Catalog

The caBIO EVS API contains only three search criteria classes: *ConceptSearchCriteria*, *DescLogicConceptSearchCriteria*, and *MetathesaurusConceptSearchCriteria*, where the latter two are subclasses of the *ConceptSearchCriteria* class. Given the simplicity of this design, and the orthogonality of the two descendant classes, the *putSearchCriteria()* methods are not applicable in the EVS domain.

### 8.7.1 The EVS SearchCriteria-Attribute Map

The listing below summarizes the settable attributes for the three EVS search criteria.

#### ***ConceptSearchCriteria***

allSource	gov.nih.nci.evs.Source[]
limit	java.lang.Int
searchTerm	java.lang.String
source	gov.nih.nci.evs.Source

#### ***DescLogicConceptSearchCriteria***

allSource	gov.nih.nci.evs.Source[]
limit	java.lang.Int
searchTerm	java.lang.String
source	gov.nih.nci.evs.Source
conceptCode	java.lang.String
initialDate	gov.nih.nci.evs.BaseLineDate
property	gov.nih.nci.evs.Property
role	gov.nih.nci.evs.Role
vocabularyName	java.lang.String

### ***MetathesaurusConceptSearchCriteria***

allSource	gov.nih.nci.evs.Source[]
limit	java.lang.Int
searchTerm	java.lang.String
source	gov.nih.nci.evs.Source
code	java.lang.Boolean
score	java.lang.Boolean
semanticType	gov.nih.nci.evs.SemanticType
shortResult	java.lang.Boolean

## **8.8 The caDSR SearchCriteria Catalog**

Table 8.8-1 lists the caDSR search criteria objects that implement the *putSearchCriteria()* method and the arguments which these objects accept.

**Table 8.8-1 caDSR *putSearchCriteria* arguments**

<u>SearchCriteria #1</u>	<u>(accepts) SearchCriteria #2</u>
CaseReportFormSearchCriteria	ClassificationSchemeItemSearchCriteria, ContextSearchCriteria, DesignationSearchCriteria, ReferenceDocumentSearchCriteria
ClassificationSchemeItemSearchCriteria	CaseReportFormSearchCriteria, ClassSchemeClassSchemeItemSearchCriteria, ClassificationSchemeSearchCriteria, ConceptualDomainSearchCriteria, DataElementConceptSearchCriteria, DataElementSearchCriteria, EnumeratedValueDomainSearchCriteria, ModuleSearchCriteria, NonenumeratedValueDomainSearchCriteria, ObjectClassSearchCriteria, PropertySearchCriteria, ProtocolFormsSetSearchCriteria, ProtocolFormsTemplateSearchCriteria, QuestionSearchCriteria, RepresentationSearchCriteria, ValidValueSearchCriteria
ClassificationSchemeSearchCriteria	ClassSchemeClassSchemeItemSearchCriteria, ContextSearchCriteria, DesignationSearchCriteria, ReferenceDocumentSearchCriteria
ClassSchemeClassSchemeItemSearchCriteria	ClassificationSchemeItemSearchCriteria, ClassificationSchemeSearchCriteria, ClassSchemeClassSchemeItemSearchCriteria
ConceptualDomainSearchCriteria	ClassificationSchemeItemSearchCriteria, DataElementConceptSearchCriteria, DesignationSearchCriteria, EnumeratedValueDomainSearchCriteria, NonenumeratedValueDomainSearchCriteria, ReferenceDocumentSearchCriteria,

SearchCriteria #1

ContextSearchCriteria

DataElementConceptRelationshipsSearchCriteria

DataElementConceptSearchCriteria

DataElementSearchCriteria

DesignationSearchCriteria

EnumeratedValueDomainSearchCriteria

*(accepts)* SearchCriteria #2

ValueMeaningSearchCriteria

CaseReportFormSearchCriteria,  
ClassificationSchemeSearchCriteria,  
DataElementConceptSearchCriteria,  
DataElementSearchCriteria,  
DesignationSearchCriteria,  
EnumeratedValueDomainSearchCriteria,  
ModuleSearchCriteria,  
NonenumeratedValueDomainSearchCriteria,  
ObjectClassSearchCriteria, PropertySearchCriteria,  
ProtocolFormsSetSearchCriteria,  
ProtocolFormsTemplateSearchCriteria,  
QuestionSearchCriteria,  
RepresentationSearchCriteria,  
ValidValueSearchCriteria

DataElementConceptSearchCriteria,  
ParentDataElementConceptSearchCriteria

DataElementConceptRelationshipSearchCriteria,  
DataElementSearchCriteria,  
ObjectClassSearchCriteria,  
ObjectClassQualifierSearchCriteria,  
PropertySearchCriteria

ClassificationSchemeItemSearchCriteria,  
ContextSearchCriteria, DesignationSearchCriteria,  
NonenumeratedValueDomainSearchCriteria,  
QuestionSearchCriteria,  
ReferenceDocumentSearchCriteria

CaseReportFormSearchCriteria,  
ClassificationSchemeSearchCriteria,  
ConceptualDomainSearchCriteria,  
ContextSearchCriteria,  
DataElementConceptSearchCriteria,  
DataElementSearchCriteria,  
EnumeratedValueDomainSearchCriteria,  
ModuleSearchCriteria,  
NonenumeratedValueDomainSearchCriteria,  
ObjectClassSearchCriteria, PropertySearchCriteria,  
ProtocolFormsSetSearchCriteria,  
ProtocolFormsTemplateSearchCriteria,  
QuestionSearchCriteria,  
RepresentationSearchCriteria,  
ValidValueSearchCriteria

ClassificationSchemeItemSearchCriteria,  
ConceptualDomainSearchCriteria,  
ContextSearchCriteria, DataElementSearchCriteria,  
DesignationSearchCriteria,  
PermissibleValueSearchCriteria,

<u>SearchCriteria #1</u>	<i>(accepts)</i> <u>SearchCriteria #2</u>
ModuleSearchCriteria	QualifierSearchCriteria, ReferenceDocumentSearchCriteria, RepresentationSearchCriteria, ValueDomainPermissibleValueSearchCriteria
NonenumeratedValueDomainSearchCriteria	ClassificationSchemeItemSearchCriteria, ContextSearchCriteria, DesignationSearchCriteria, QuestionSearchCriteria, ReferenceDocumentSearchCriteria
ObjectClassSearchCriteria	ClassificationSchemeItemSearchCriteria, ConceptualDomainSearchCriteria, ContextSearchCriteria, DataElementSearchCriteria, DesignationSearchCriteria, QualifierSearchCriteria, ReferenceDocumentSearchCriteria, RepresentationSearchCriteria
PermissibleValueSearchCriteria	ClassificationSchemeItemSearchCriteria, ContextSearchCriteria, DataElementConceptSearchCriteria, DesignationSearchCriteria, ReferenceDocumentSearchCriteria
PropertySearchCriteria	EnumeratedValueDomainSearchCriteria, ValueDomainPermissibleValueSearchCriteria, ValueMeaningSearchCriteria
ProtocolFormsSetSearchCriteria	ClassificationSchemeItemSearchCriteria, ContextSearchCriteria, DataElementConceptSearchCriteria, DesignationSearchCriteria, ReferenceDocumentSearchCriteria
ProtocolFormsTemplateSearchCriteria	CaseReportFormSearchCriteria, ClassificationSchemeItemSearchCriteria, ContextSearchCriteria, DesignationSearchCriteria, ReferenceDocumentSearchCriteria
QualifierSearchCriteria	ClassificationSchemeItemSearchCriteria, ContextSearchCriteria, DesignationSearchCriteria, ReferenceDocumentSearchCriteria
QuestionSearchCriteria	EnumeratedValueDomainSearchCriteria, NonenumeratedValueDomainSearchCriteria, DataElementConceptSearchCriteria
ReferenceDocumentSearchCriteria	ClassificationSchemeItemSearchCriteria, ContextSearchCriteria, DataElementSearchCriteria, DesignationSearchCriteria, ModuleSearchCriteria, ReferenceDocumentSearchCriteria, ValidValueSearchCriteria
	CaseReportFormSearchCriteria, ClassificationSchemeSearchCriteria, ConceptualDomainSearchCriteria, DataElementConceptSearchCriteria,

<u>SearchCriteria #1</u>	<i>(accepts)</i> <u>SearchCriteria #2</u> DataElementSearchCriteria, EnumeratedValueDomainSearchCriteria, ModuleSearchCriteria, NonenumeratedValueDomainSearchCriteria, ObjectClassSearchCriteria, PropertySearchCriteria, ProtocolFormsSetSearchCriteria, ProtocolFormsTemplateSearchCriteria, QuestionSearchCriteria, RepresentationSearchCriteria, ValidValueSearchCriteria
RepresentationSearchCriteria	ClassificationSchemeItemSearchCriteria, ContextSearchCriteria, DesignationSearchCriteria, EnumeratedValueDomainSearchCriteria, NonenumeratedValueDomainSearchCriteria, ReferenceDocumentSearchCriteria
ValidValueSearchCriteria	ClassificationSchemeItemSearchCriteria, ContextSearchCriteria, DesignationSearchCriteria, QuestionSearchCriteria, ReferenceDocumentSearchCriteria, ValueDomainPermissibleValueSearchCriteria
ValueDomainPermissibleValueSearchCriteria	EnumeratedValueDomainSearchCriteria, PermissibleValueSearchCriteria, ValidValueSearchCriteria
ValueMeaningSearchCriteria	ConceptualDomainSearchCriteria, PermissibleValueSearchCriteria

### 8.8.1 The caDSR SearchCriteria-Attribute Map

This section summarizes the object-specific settable attributes for the various caDSR search criteria. Each of these attributes is a private data member of the class, but is settable via the *set* method of the same name.

#### *AdministeredComponentSearchCriteria*

beginDate	java.util.Date
changeNote	java.lang.String
dateCreated	java.util.Date
dateModified	java.util.Date
deletedIndicator	java.lang.Boolean
endDate	java.util.Date
latestVersionIndicator	java.lang.Boolean
longName	java.lang.String
origin	java.lang.String
preferredDefinition	java.lang.String
preferredName	java.lang.String
publicId	java.lang.Long
unresolvedIssue	java.lang.String

version	java.lang.String
workflowStatusDescription	java.lang.String
workflowStatusName	java.lang.String
<b><i>CaseReportFormSearchCriteria*</i></b>	
displayName	java.lang.String
<b><i>ClassificationSchemeItemSearchCriteria</i></b>	
comments	java.lang.String
dateCreated	java.util.Date
dateModified	java.util.Date
description	java.lang.String
name	java.lang.String
type	java.lang.String
<b><i>ClassificationSchemeSearchCriteria*</i></b>	
labeltypeflag	java.lang.String
type	java.lang.String
<b><i>ClassSchemeClassSchemeItemSearchCriteria</i></b>	
dateCreated	java.util.Date
dateModified	java.util.Date
displayOrder	java.lang.Integer
label	java.lang.String
<b><i>ConceptualDomainSearchCriteria*</i></b>	
dimensionality	java.lang.String
<b><i>ContextSearchCriteria</i></b>	
dateCreated	java.util.Date
dateModified	java.util.Date
description	java.lang.String
designationId	java.lang.String
languageName	java.lang.String
name	java.lang.String
version	java.lang.Float
<b><i>DataElementConceptRelationshipsSearchCriteria</i></b>	
dateCreated	java.util.Date
dateModified	java.util.Date
description	java.lang.String
name	java.lang.String
<b><i>DataElementConceptSearchCriteria*</i></b>	
<b><i>DataElementSearchCriteria*</i></b>	
<b><i>DesignationSearchCriteria</i></b>	

---

\* Also inherits all attributes from *AdministeredComponentSearchCriteria*



dateCreated	java.util.Date
dateModified	java.util.Date
languageName	java.lang.String
name	java.lang.String
type	java.lang.String

***EnumeratedValueDomainSearchCriteria §***

***ModuleSearchCriteria\****

displayOrder	java.lang.Integer
--------------	-------------------

***NonenumeratedValueDomainSearchCriteria §***

***ObjectClassSearchCriteria\****

definitionSource	java.lang.String
------------------	------------------

***PermissibleValueSearchCriteria***

beginDate	java.util.Date
dateCreated	java.util.Date
dateModified	java.util.Date
endDate	java.util.Date
highValueNumber	java.lang.String
lowValueNumber	java.lang.String
value	java.lang.String

***PropertySearchCriteria\****

definitionSource	java.lang.String
------------------	------------------

***ProtocolFormsSetSearchCriteria\****

approvedBy	java.lang.String
approvedDate	java.lang.Date
changeNumber	java.lang.String
changeType	java.lang.String
leadOrganizationName	java.lang.String
phase	java.lang.String
protocolId	java.lang.String
reviewedBy	java.lang.String
reviewedDate	java.lang.Date
type	java.lang.String

***ProtocolFormsTemplateSearchCriteria\****

***QualifierSearchCriteria***

comments	java.lang.String
dateCreated	java.util.Date
dateModified	java.util.Date
description	java.lang.String

---

§ Also inherits attributes from *ValueDomainSearchCriteria*, which in turn inherits from *AdministeredComponentSearchCriteria*.

name	java.lang.String
<b><i>QuestionSearchCriteria*</i></b>	
displayOrder	java.lang.Integer
<b><i>ReferenceDocumentSearchCriteria</i></b>	
dateCreated	java.util.Date
dateModified	java.util.Date
displayOrder	java.lang.Integer
docText	java.lang.String
languageName	java.lang.String
name	java.lang.String
organizationId	java.lang.String
rdtlName	java.lang.String
type	java.lang.String
url	java.lang.String
<b><i>RepresentationSearchCriteria*</i></b>	
definitionSource	java.lang.String
<b><i>ValidValueSearchCriteria</i></b>	
displayOrder	java.lang.Integer
<b><i>ValueDomainPermissibleValueSearchCriteria</i></b>	
dateCreated	java.util.Date
dateModified	java.util.Date
<b><i>ValueDomainSearchCriteria</i></b>	
characterSetName	java.lang.String
dataTypeName	java.lang.String
decimalPlace	java.lang.Integer
formatName	java.lang.String
highValueNumber	java.lang.String
lowValueNumber	java.lang.String
maximumLengthNumber	java.lang.Integer
minimumLengthNumber	java.lang.Integer
uomName	java.lang.String
<b><i>ValueMeaningSearchCriteria</i></b>	
beginDate	java.util.Date
comments	java.lang.String
dateCreated	java.util.Date
dateModified	java.util.Date
description	java.lang.String
endDate	java.util.Date
shortMeaning	java.lang.String

## 8.9 The caMOD SearchCriteria Catalog

Table 8.9-1 lists the caDSR search criteria objects that implement the *putSearchCriteria()* method and the arguments which these objects accept.

**Table 8.9-1 caMOD *putSearchCriteria* arguments**

<u>SearchCriteria #1</u>	<u>(accepts) SearchCriteria #2</u>
domain.AnimalModelSearchCriteria	AvailabilitySearchCriteria, CarcinogenicInterventionSearchCriteria, CellLineSearchCriteria, GenomicSegmentSearchCriteria, HistopathologySearchCriteria, ImageSearchCriteria, InducedMutationSearchCriteria, JaxInfoSearchCriteria, MicroArrayDataSearchCriteria, PartyRoleSearchCriteria, PhenotypeSearchCriteria, PublicationSearchCriteria, RepositoryInfoSearchCriteria, TargetedModificationSearchCriteria, TaxonSearchCriteria, TherapySearchCriteria, TransgeneSearchCriteria, XenograftSearchCriteria
CarcinogenicInterventionSearchCriteria	EnvironmentalFactorSearchCriteria, GeneDeliverySearchCriteria, TreatmentScheduleSearchCriteria
CellLineSearchCriteria	OrganSearchCriteria, PublicationSearchCriteria
ConditionalitySearchCriteria	PartySearchCriteria
EngineeredGeneSearchCriteria	ConditionalitySearchCriteria, ExpressionFeatureSearchCriteria, GeneSearchCriteria, GeneFunctionSearchCriteria, GenotypeSummarySearchCriteria, ImageSearchCriteria, PromoterSearchCriteria
GeneDeliverySearchCriteria	EngineeredGeneSearchCriteria, OrganSearchCriteria
GenomicSegmentSearchCriteria	EngineeredGeneSearchCriteria, IntegrationTypeSearchCriteria, SegmentTypeSearchCriteria
GenotypeSummarySearchCriteria	NomenclatureSearchCriteria
ImageSearchCriteria	AvailabilitySearchCriteria
InducedMutationSearchCriteria	EngineeredGeneSearchCriteria, EnvironmentalFactorSearchCriteria
MicroArrayDataSearchCriteria	AvailabilitySearchCriteria
ModificationTypeSearchCriteria	TargetedModificationSearchCriteria
PartySearchCriteria	ContactInfoSearchCriteria, PartyRoleSearchCriteria
PartyRoleSearchCriteria	AnimalModelSearchCriteria
PhenotypeSearchCriteria	SexDistributionSearchCriteria
PublicationSearchCriteria	PublicationStatusSearchCriteria
RegulatoryElementSearchCriteria	RegulatoryElementTypeSearchCriteria, TaxonSearchCriteria
RoleSearchCriteria	PartyRoleSearchCriteria
TargetedModificationSearchCriteria	EngineeredGeneSearchCriteria, ModificationTypeSearchCriteria
TherapySearchCriteria	AgentSearchCriteria, PublicationSearchCriteria,

<u>SearchCriteria #1</u>	<u>(accepts) SearchCriteria #2</u>
domain.AnimalModelSearchCriteria	AvailabilitySearchCriteria, CarcinogenicInterventionSearchCriteria, CellLineSearchCriteria, GenomicSegmentSearchCriteria, HistopathologySearchCriteria, ImageSearchCriteria, InducedMutationSearchCriteria, JaxInfoSearchCriteria, MicroArrayDataSearchCriteria, PartyRoleSearchCriteria, PhenotypeSearchCriteria, PublicationSearchCriteria, RepositoryInfoSearchCriteria, TargetedModificationSearchCriteria, TaxonSearchCriteria, TherapySearchCriteria, TransgeneSearchCriteria, XenograftSearchCriteria
CarcinogenicInterventionSearchCriteria	EnvironmentalFactorSearchCriteria, GeneDeliverySearchCriteria, TreatmentScheduleSearchCriteria
CellLineSearchCriteria	OrganSearchCriteria, PublicationSearchCriteria
ConditionalitySearchCriteria	PartySearchCriteria TreatmentScheduleSearchCriteria
TransgeneSearchCriteria	IntegrationTypeSearchCriteria, RegulatoryElementSearchCriteria, TaxonSearchCriteria
XenograftSearchCriteria	HostTaxonSearchCriteria, OrganSearchCriteria, OriginTaxonSearchCriteria

### 8.9.1 The caMOD Criteria-Attribute Map

This section summarizes the object-specific settable attributes for the various caMOD search criteria. Each of these attributes is a private data member of the class, but is settable via the *set* method of the same name.

#### *AnimalModelSearchCriteria*

agentId	java.lang.Long
availabilityId	java.lang.Long
diseaseId	java.lang.Long
experimentDescription	java.lang.String
modelDescriptor	java.lang.String
partyRoleId	java.lang.Long
phenotypeId	java.lang.Long
taxonId	java.lang.Long

#### *ApprovalStatusSearchCriteria*

animalModelId	java.lang.Long
approvalStatusName	java.lang.String

#### *AvailabilitySearchCriteria*

animalModelId	java.lang.Long
enteredDate	java.util.Date
modifiedDate	java.util.Date
releaseDate	java.util.Date

visibleTo	java.lang.String
<b><i>CarcinogenicInterventionSearchCriteria</i></b>	
animalModelId	java.lang.Long
environmentalFactorId	java.lang.Long
geneDeliveryId	java.lang.Long
treatmentScheduleId	java.lang.Long
<b><i>CellLineSearchCriteria</i></b>	
animalModelId	java.lang.Long
comments	java.lang.String
experiment	java.lang.String
name	java.lang.String
organId	java.lang.Long
<b><i>ConditionalitySearchCriteria</i></b>	
conditionedBy	java.lang.String
Desc	java.lang.String
<b><i>ContactInfoSearchCriteria</i></b>	
city	java.lang.String
email	java.lang.String
fax	java.lang.String
institute	java.lang.String
labName	java.lang.String
partyId	java.lang.Long
phoneNumber	java.lang.String
state	java.lang.String
street	java.lang.String
zip	java.lang.String
<b><i>EngineeredGeneSearchCriteria</i></b>	
caBioId	java.lang.Long
conditionalityId	java.lang.Long
dbCrossRefs	java.util.Hashtable
genomicSegmentId	java.lang.Long
genotypeSummaryId	java.lang.Long
imageId	java.lang.Long
inducedMutationId	java.lang.Long
locusLinkSummary	java.lang.String
name	java.lang.String
targetedModificationId	java.lang.Long
title	java.lang.String
<b><i>EnvironmentalFactorSearchCriteria</i></b>	
carcinogenicId	java.lang.Long
name	java.lang.String
type	java.lang.String

<b><i>GeneDeliverySearchCriteria</i></b>	
engineeredGeneId	java.lang.Long
geneId	java.lang.Long
organId	java.lang.Long
viralVector	java.lang.String
<b><i>GeneFunctionSearchCriteria</i></b>	
engineeredGeneId	java.lang.Long
geneFunction	java.lang.String
<b><i>GeneticAlterationSearchCriteria</i></b>	
histopathologyId	java.lang.Long
methodOfObservation	java.lang.String
observation	java.lang.String
<b><i>GenomicSegmentSearchCriteria</i></b>	
animalModelId	java.lang.Long
cloneDesignator	java.lang.String
integrationTypeId	java.lang.Long
LocationOfIntegration	java.lang.String
segmentSize	java.lang.Float
segmentTypeId	java.lang.Long
<b><i>GenotypeSummarySearchCriteria</i></b>	
genotype	java.lang.String
nomenclatureID	java.lang.Long
summary	java.lang.String
<b><i>ImageSearchCriteria</i></b>	
animalModelId	java.lang.Long
description	java.lang.String
image	java.lang.String
staining	java.lang.String
title	java.lang.String
<b><i>InducedMutationSearchCriteria</i></b>	
animalModelId	java.lang.Long
environmentalFactorId	java.lang.Long
<b><i>IntegrationTypeSearchCriteria</i></b>	
integrationTypeName	java.lang.String
<b><i>JaxInfoSearchCriteria</i></b>	
jaxStockNo	java.lang.Long
<b><i>MicroArrayDataSearchCriteria</i></b>	
animalModelId	java.lang.Long
experimentId	java.lang.Long
experimentName	java.lang.String
<b><i>ModificationTypeSearchCriteria</i></b>	
modificationTypeName	java.lang.String

targetModificationId	java.lang.Long
<b><i>NomenclatureSearchCriteria</i></b>	
name	java.lang.String
<b><i>OrganizationSearchCriteria*</i></b>	
lastName	java.lang.String
<b><i>PartyRoleSearchCriteria</i></b>	
animalModelId	java.lang.Long
partyId	java.lang.Long
roleId	java.lang.Long
<b><i>PartySearchCriteria</i></b>	
contactInfo	java.lang.String
<b><i>PersonSearchCriteria*</i></b>	
firstName	java.lang.String
lastName	java.lang.String
<b><i>PhenotypeSearchCriteria</i></b>	
animalModelId	java.lang.Long
breedingNotes	java.lang.String
desc	java.lang.String
sexDistributionId	java.lang.Long
<b><i>PromoterSearchCriteria§</i></b>	
<b><i>PublicationSearchCriteria</i></b>	
animalmodelId	java.lang.Long
authors	java.lang.String
cellLineId	java.lang.Long
endPage	java.lang.Long
journal	java.lang.String
pmId	java.lang.Long
publicationStatusId	java.lang.Long
startPage	java.lang.Long
status	java.lang.String
therapyId	java.lang.Long
title	java.lang.String
volume	java.lang.String
year	java.lang.Long
<b><i>PublicationStatusSearchCriteria</i></b>	
publicationStatusName	java.lang.String

---

\* Also inherits from *PartySearchCriteria*

§ Also inherits from *RegulatoryElementSearchCriteria*

<b><i>RegulatoryElementSearchCriteria</i></b>	
name	java.lang.String
regulatoryElementTypeId	java.lang.Long
speciesId	java.lang.Long
transGeneId	java.lang.Long
<b><i>RegulatoryElementTypeSearchCriteria</i></b>	
regulatoryElementTypeName	java.lang.String
<b><i>RepositoryInfoSearchCriteria</i></b>	
inTheRepository	java.lang.Long
sentEmailContent	java.lang.String
suggestSubmission	java.lang.Long
<b><i>RoleSearchCriteria</i></b>	
roleName	java.lang.String
<b><i>SegmentTypeSearchCriteria</i></b>	
segmentTypeName	java.lang.String
<b><i>SexDistributionSearchCriteria</i></b>	
sexDistributionTypeName	java.lang.String
<b><i>TargetedModificationSearchCriteria</i></b>	
animalModelId	java.lang.Long
blastocystName	java.lang.String
engineeredGeneId	java.lang.Long
escellLineName	java.lang.String
geneId	java.lang.Long
modificationTypeId	java.lang.Long
<b><i>TherapySearchCriteria</i></b>	
agentId	java.lang.Long
animalModelId	java.lang.Long
comments	java.lang.String
experiment	java.lang.String
treatmentScheduleId	java.lang.Long
<b><i>TransgeneSearchCriteria</i><sup>±</sup></b>	
animalModelId	java.lang.Long
integrationTypeId	java.lang.Long
locationOfIntegration	java.lang.String
speciesId	java.lang.Long
<b><i>TreatmentScheduleSearchCriteria</i></b>	
carcinogenicInterventionId	java.lang.Long
dosage	java.lang.String
regimen	java.lang.String

---

<sup>±</sup> Also inherits from *EngineeredGeneSearchCriteria*

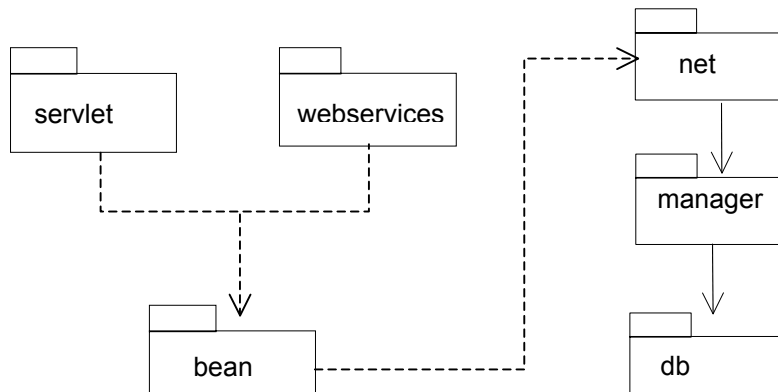


therapyId	java.lang.Long
<b><i>XenograftSearchCriteria</i></b>	
administrativeSite	java.lang.String
animalModelId	java.lang.Long
geneticManipulation	java.lang.String
hostSpeciesId	java.lang.Long
modificationDescription	java.lang.String
name	java.lang.String
organId	java.lang.Long
originSpeciesId	java.lang.Long
parentCellLineName	java.lang.String
type	java.lang.String

## **9.0 THE caCORE PACKAGE ARCHITECTURE**

## 9.1 Organization of Packages in caCORE

The caCORE 2.0 package structure reflects two perspectives: (1) a reorganization of packages that the current implementation is migrating towards, and (2) vestiges of the previous major release that are being maintained for backwards compatibility. In the caCORE 1.0 design, the caBIO *bean* package was used to represent application-specific objects for the single domain that the entire framework supported at that time. Figure 9.1-1 summarizes the main features of the original package structure used in release 1.0:



**Figure 9.1-1 The caCORE 1.0 package structure**

A user could interact with the caBIO server in one of three ways. First, using a browser, a user could enter the URL for a servlet contained in the *servlet* package. For example, the user could access the *getXML* servlet, which provides an HTTP interface to perform searches on domain objects. Second, a user could use a SOAP client to invoke various Web Services methods contained in the *webservices* package. In both cases, the initial request was serviced by the caBIO server by instantiating appropriate domain objects in the *bean* package, and forwarding the requests to manager objects in the *net* and *manager* packages. Java clients bypassed this first step, as their applications instantiate the domain objects directly.

While the package structure has expanded and evolved, this basic framework is still in place in the 2.0 release; the difference is that the framework has been extended to support additional application domains. The newly introduced application domains share many of the same requirements as caBIO, and an obvious choice was to refactor those components that had previously been deployed to support the caBIO domain objects only.

This led to the definition of five major modules: *common*, *cabio*, *cadrs*, *camod*, and *evs*, corresponding to the four domains currently supported and a single shared module. The idea is that each of these modules has an internal structure similar to that shown in Figure 9-1; the packages contained in the *common* module implement the basic components shared by all of the applications.

While the three new modules corresponding to the new domains follow this design principle fairly rigorously, the *cabio* module retains much of its original “stand-alone” implementation, for backwards compatibility. The problem with making a clean sweep is that applications based on the original design would no longer be supported. Thus, the package design in release 2.0 applies the new design structure to all of the new domains, but retains much of the previous design

internal to the *cabio* packages. In future releases the *cabio* module will also conform more rigorously to the new design.

Table 9.1-1 summarizes the primary packages contained in caCORE 2.0. For convenience, the package names in the first column drop the name's prefix and retain only the last part of the full name. The actual name for each package is composed as:

*gov.nih.nci.<component name>.<package name>*

**Table 9.1-1 The caCORE Packages**

Package	common	caBIO	caDSR	caMOD	EVS
bean		✓	✓	✓	✓
db	✓				
domain	✓				
exception	✓			✓	
manager		✓	✓	✓	✓
net	✓	✓	✓	✓	✓
search		✓	✓	✓	✓
servlet		✓		✓	
util	✓	✓	✓	✓	✓
webservices		✓	✓		

A checkmark in the component column indicates that the component contains a package of the same name as the corresponding row. Thus, there are six packages in the *evs* component, named *gov.nih.nci.evs.bean*, *gov.nih.nci.evs.exception*, *gov.nih.nci.evs.manager*, *gov.nih.nci.evs.net*, *gov.nih.nci.evs.search*, and *gov.nih.nci.evs.util*.

As mentioned, all of the application domains have a similar organization of packages, with variations depending on the specific application. For example, the EVS has its own proprietary data access layer powered by the Apelon vocabulary server. Table 9-2 summarizes the caCORE package organization and provides brief descriptions of the functionalities located in the packages.

**Table 9.1-2 caCORE Packages Summaries**

Domains	Package	Description
<ul style="list-style-type: none"> <li>• caBIO</li> <li>• caDSR</li> <li>• EVS</li> <li>• caMOD</li> </ul>	<i>bean</i>	Holds domain-specific objects such as <i>Gene</i> , <i>Chromosome</i> (caBIO); <i>DataElement</i> , <i>AdministeredComponent</i> (caDSR); <i>Concept</i> , <i>Definition</i> (EVS); <i>AnimalModel</i> , <i>Availability</i> (caMOD), etc.
<ul style="list-style-type: none"> <li>• common</li> </ul>	<i>db</i>	Contains the logic to perform queries against the databases and return results as collections or arrays of the associated domain objects; contains shared code to hold JDBC connection information and deal with errors related to connection pooling.

<ul style="list-style-type: none"> <li>• common</li> </ul>	<i>domain</i>	Holds the parent class to all caMOD domain objects.
<ul style="list-style-type: none"> <li>• common</li> <li>• caMOD</li> <li>• EVS</li> </ul>	<i>exception</i>	Holds the exception classes for error handling.
<ul style="list-style-type: none"> <li>• caBIO</li> <li>• caDSR</li> <li>• caMOD</li> <li>• EVS</li> </ul>	<i>manager</i>	Holds the object manager classes to support RMI on the server.
<ul style="list-style-type: none"> <li>• common</li> <li>• caBIO</li> <li>• caDSR</li> <li>• caMOD</li> <li>• EVS</li> </ul>	<i>net</i>	Holds the proxy manager classes to support RMI on the client – these objects encapsulate the network protocols used by the domain objects to communicate with the server, and thus abstract RMI implementations away from the user.
<ul style="list-style-type: none"> <li>• common</li> <li>• caBIO</li> <li>• caDSR</li> <li>• caMOD</li> <li>• EVS</li> </ul>	<i>search</i>	Holds the classes that collectively implement both the basic and advanced search operations. For caDSR, EVS, and caMOD, the search criteria objects are in this package.. For caBIO only, the search criteria classes are in the <i>bean</i> package in release 2.0. In future releases however, the caBIO search criteria classes will also be located in the <i>search</i> package.
<ul style="list-style-type: none"> <li>• caBIO</li> <li>• caMOD</li> </ul>	<i>servlet</i>	Holds classes supporting the JSP pages in the Presentation Layer; encapsulates the caBIO servlet implementations.
<ul style="list-style-type: none"> <li>• common</li> <li>• caBIO</li> <li>• caDSR</li> <li>• caMOD</li> <li>• EVS</li> </ul>	<i>util</i>	Holds the classes supporting XML-encoding and other types of serialization; includes the Document Object Model (DOM) parser and Scalable Vector Graphics (SVG) pathway wrappers.
<ul style="list-style-type: none"> <li>• caBIO</li> </ul>	<i>util.das</i>	Classes providing access to the Distributed Annotation Server (DAS) at UCSC. The classes in this package were auto-generated from the DAS DTDs using the Sun JAXB "xjc" tool.
<ul style="list-style-type: none"> <li>• caBIO</li> <li>• caDSR</li> <li>• caMOD</li> </ul>	<i>webservices</i>	Contains the caBIO webservices classes that receive and respond to SOAP requests from a SOAP client or consumer by rendering a response as a SOAP message.

A second factor contributing to the reorganization of the package structure was the emergence of a more complete set of search utilities. The central paradigm of the caCORE 1.0 release was based on the usage of instantiated caBIO domain objects in collaboration with their associated search criteria objects to generate SQL queries. In release 1.0, these search criteria objects were defined in the *bean* package, along with the domain objects.

In release 2.0, this same organization of classes is found in the *cabio.bean* package only; all of the other modules locate their search criteria objects in the newly defined *search* packages. In addition, all of these individual search criteria objects are direct subclasses of the *SearchCriteria* class defined in the *gov.nih.nci.common.search* package. In contrast, search criteria objects in the

*cabio.bean* package are indirect subclasses of the common *SearchCriteria* object. The next chapter describes the classes contained in the *search* packages in more depth.

## 9.2 The caBIO DAS Package

The caBIO package structure also differs from the other components in that it includes an interface to the Distributed Annotation Server. Excellent documentation about these servers is available on the [DAS](#) home page. For a brief summary of DAS, visit the [Overview](#) page; for a more in-depth discussion, see the [DAS/1 Specification](#) pages. This section provides a general overview of the DAS<sup>7</sup>, and describes the interface to these services implemented in the *gov.nih.nci.caBIO.util.das* package.

The DAS provides a framework for sharing annotations associated with entire chromosomes, complete sequences, and sequence fragments such as contigs. The key to how this knowledge is shared rests in the idea of *reference sequences*, which can be used to locate nested indexes in hierarchical structures. A reference sequence is defined as a set of entry points into a sequence, along with a set of sequence lengths to be associated with these entry points. Each entry point itself may have a substructure that recursively defines its own set of entry points, thus effecting a kind of “zoom-in/out” capability. The annotations are in turn associated with selected regions of the referred-to genomic segment by unambiguous start and stop positions relative to their respective reference sequences.

The DAS uses a web-based client-server architecture consisting of a *reference sequence server* in collaboration with several *annotation servers*. Clients query these servers by sending formatted URL requests using the HTTP/1.0 protocol, and, in response, receive formatted XML documents.

The basic operation is to retrieve annotations according to the type of construct annotated, the methods used for discovery, and the broad functional category of the annotation. Annotation *types* include examples such as “exon,” “intron,” and “CDS.” The *categories* are used to filter, group and sort annotations, and include things like “homology,” “variation,” and “transcribed.” Different annotation servers specialize in retrieving annotations across different regions of the genomes. The reference sequence server is an annotation server that, when given a reference sequence id, can provide additional information concerning the raw DNA as well as how the sequence information was assembled.

The [DAS/1 Specification](#) pages provide catalogs of the available query commands, along with detailed explanations of how to use these commands and how to interpret the XML response documents. Classes in the *gov.nih.nci.caBIO.util.das* package were auto-generated from the DAS DTDs using the Sun JAXB “xjc” tool. For example, the class *DasDsn* was generated from “<http://www.biodas.org/dtd/dasdsn.dtd>.”

### 9.2.1 Accessing the DAS Server Using caBIO Objects

Three types of retrievals can be performed against a DAS server using caBIO objects:

1. Retrieval of annotation types.
2. Retrieval of annotations.

---

<sup>7</sup> This description is based on the [DAS/1 Specification](#) pages and includes excerpts from that document.

### 3. Retrieval of DNA base pair sequences.

The first type of query will retrieve a list of annotation types, filtered by specific segments or type subsets if desired. The second type of query will fetch a set of features/annotations for a given segment. This type of query can be restricted to return only certain types or categories, or filtered by a gene or sequence of interest. Finally, the third type of query will retrieve a DNA base pair sequence for a given gene or sequence.

There are several classes in the caBIO hierarchy that are used to perform DAS queries, but the most important are the search criteria objects *DasDnaSearchCriteria*, *DasTypeSearchCriteria*, and *DasGffFeatureSearchCriteria*—which are defined in the *gov.nih.nci.caBIO.util.das* package.

#### Example 1: Search for annotation types

```
DasTypeSearchCriteria criteria = new DasTypeSearchCriteria();
SearchResult results = null;
try {
    results = criteria.search();
} catch (Exception e) {
    MessageLog.printlnInfo("Search exception " + e);
}
String[] types = (String [])results.getResultSet();
```

#### Example 2: Search for annotations:

```
SearchResult results=null;
DasGffFeatureSearchCriteria criteria=new DasGffFeatureSearchCriteria();
try{
    criteria.setGenes(new Gene(new Long(10)));
    criteria.setGenes(genes);
    results=criteria2.search();
} catch (Exception e){
    MessageLog.printlnInfo("Search exception " + e);
}
DasGffFeature[] features=(DasGffFeature [])results.getResultSet();
```

#### Example 3: Search for DNA sequences:

```
SearchResult results=null;
try{
    DasDnaSearchCriteria criteria=new DasDnaSearchCriteria();
    DasDataSource source = new DasDataSource();
    source.setURI("http://genome-test.cse.ucsc.edu/cgi-bin/das/hg7");
    criteria.setDataSource(source);
    criteria.setSequences(new Sequence(new Long(2709864)));
    results=criteria.search();
} catch (Exception e){
    MessageLog.printlnInfo("Search exception " + e);
}
DasDnaDna[] dnas=(DasDnaDna [])results.getResultSet();
MessageLog.printlnInfo(" DNA -> " + dnas[0].getContent());
```

This last example retrieves a DNA string using a sequence id, and uses the *DasDataSource* to specify a server different from the caBIO default DAS server.

## **10.0 Advanced Search Methods**



## 10.1 Basic and Advanced Search Methods

We begin this discussion of the advanced search interface with a review of the basic search paradigm presented in [Chapter 8](#). The sequence described there was:

1. Instantiate a new domain object of the desired type;
2. Create a new *SearchCriteria* for that domain object, and set its attributes;
3. Invoke the domain object's *search()* method on that *SearchCriteria*;
4. Invoke the *getResultSet()* method on the returned *SearchResult* object.

This process is well-suited to applications where the focus is not on the queries themselves but rather, on the subsequent analysis of the query results. For applications such as BIOgopher however, where the primary activity is issuing queries, a more advanced capability for executing batch searches is needed. This is exactly what the *search* packages provide.

The advanced search paradigm implemented by these packages assembles a tree-structured list of search criteria which can be used to issue highly complex search queries. The implementation introduces several new objects which are summarized in **Error! Reference source not found.** Like the basic search paradigm, the advanced interface is centered around search criteria objects. In this case however, it is no longer necessary to instantiate an associated domain object for each search criteria.

**Table 10.1-1 Central objects used in the advanced search methods**

<u>Object</u>	<u>Description</u>
<i>SelectionNode</i>	For each search criteria object to be used in the query, a selection node is added to a growing query tree. Each node specifies a search criteria, a list of the relevant fields (attribute names) for that object, and a name for the node. The root of the tree is itself a selection node, which may have (0) or more child nodes to which it is linked.
<i>SearchCriteriaMapping</i>	The <i>SearchCriteriaMapping</i> objects serve two functions. Firstly, they are used to define the values which will be associated with the search criteria attributes in the query tree. Secondly, they provide a mechanism for associating labels with the results. The results are returned in a data structure that operates as an associative array; these labels can then be used to extract specific elements.
<i>GridSearchCriteria</i>	A <i>GridSearchCriteria</i> should be thought of as a structured collection of <i>SearchCriteria</i> objects. The two arguments to the constructor are the root of a query tree (a <i>SelectionNode</i> ), and an array of <i>SearchCriteriaMapping</i> [].
<i>ObjectGrid</i>	Just as the domain objects provided a generic <i>search()</i> method for executing simple queries, the <i>ObjectGrid</i> provides a a generic <i>search()</i> method for operating on a <i>GridSearchCriteria</i> . The search method returns a <i>GridSearchResultMapping</i> .

<u>Object</u>	<u>Description</u>
<i>GridSearchResultMapping</i>	As implied by this object's name, the results of the search are combined with the <i>SearchCriteriaMappings</i> which were provided in the <i>GridSearchCriteria</i> object. The combined results and mappings are returned as <i>GridRow</i> objects.
<i>GridRow</i>	A <i>GridRow</i> acts as an associative array, in that results are retrieved by name. The "name" of a cell is defined through a combination of the node names defined in the <i>SelectionNodes</i> and the labels defined in the <i>SearchCriteriaMappings</i> .

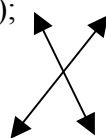
Abstracting away the complexity involved in constructing a *GridSearchCriteria*, we can find clear parallels to the basic search paradigm, summarized below. In this summary, we have simplified the number of objects involved in the advanced search to emphasize the parallels:

#### Basic Search

1. `Gene myGene = new Gene();`
2. `GeneSearchCriteria GSC = new GeneSearchCriteria();`
3. `GSC.setName("PTEN");`
4. `myGene.search (GSC)`

#### Advanced Search

1. `// define the Tree and Mappings data structures`
2. `GridSearchCriteria GSC = new GridSearchCriteria(Tree, Mappings);`
3. `ObjectGrid myGrid = new ObjectGrid();`
4. `myGrid.search (GSC)`



As suggested by this side-by-side comparison, the advanced search has effectively reversed steps (1) and (3) of the basic search. In the basic search method, we invoke the parameter-free constructor to instantiate a new search criteria object, and subsequently, use its *set* methods to define the attributes we wish to use as a filter. In contrast, the advanced search first defines all of the attributes and values to be searched for, and subsequently, uses these to initialize a new *GridSearchCriteria* object.

## 10.2 Constructing the query tree

*SelectionNode* objects are extensions of the Java class *DefaultMutableTreeNode* contained in the package *javax.swing.tree*. Each *SelectionNode* has a *SearchCriteria* object (the "filter"), a list of attributes, and a node name associated with it. To build a query tree, we begin by defining its root selection node, and to do that, we begin by defining that node's search criteria object.

The following code defines a *GeneSearchCriteria* object whose selection attribute is the name of the gene:

```
List geneNames = new ArrayList();
geneNames.add( "PTEN" );
geneNames.add( "TP53" );
geneNames.add( "BRCA1" );
GeneSearchCriteria gsc = new GeneSearchCriteria();
gsc.putCriteria( "name", geneNames );
```

This example uses the *putCriteria()* method to define the actual values we would like to filter the gene names by. Note that this expression will *not* be used to extract an individual gene whose name satisfies *all* of the specified values – that would not be possible. Instead, the expression will be used to find all genes whose names match any *one* of the specified values.

Having defined our search criteria's attribute and values, our next step is to build the selection node's attribute list. In this case we have only one attribute:

```
List geneAtts = new ArrayList();
geneAtts.add( "name" );
```

With this we can now initialize a new selection node\* named "Gene" using these constructs:

```
SelectionNode tree = new SelectionNodeImpl( "Gene", gsc, geneAtts );
```

At this point, we have constructed a query tree with a single selection node that can be used to extract the set of genes whose names match the specified values. In order to use this query tree in a search, we will need to (1) define the *ObjectGrid* that will be used to invoke the search; and (2) define the *GridSearchCriteria* object that will be required for that invocation.

### 10.3 Building the GridSearchCriteria

The two arguments to the *GridSearchCriteria* constructor are the root of a query tree (a *SelectionNode*), and an array of *SearchCriteriaMappings*. Note that while a query tree can have many selection nodes, only one of these can be defined as the root.

When our search is finally executed, the results will be returned as rows, with each row corresponding to a branch of the query tree. The branch is "flattened" into a linear sequence that behaves like an associative array. Each *SearchCriteriaMapping* provides a tag that will be used to access an individual cell in a result row. The *SearchCriteriaMapping* also contains a single search criteria object that will be associated with a specific selection node in the query tree.

Recall that our query tree contains just one selection node whose *single* search criteria specifies a *set* of gene names. For each name, a separate *SearchCriteriaMapping* must now be created. For convenience, we will use the gene name itself for the tag, and create a *GeneSearchCriteria* whose attribute and value correspond to this tag:

```
SearchCriteriaMapping[] myMappings =
    new SearchCriteriaMapping[geneNames.size()];

for( ListIterator i = geneNames.listIterator(); i.hasNext(); ){
    GeneSearchCriteria sc = new GeneSearchCriteria();
    String myTag = (String)i.next();
    sc.putCriteria( "name", myTag);
    SearchCriteriaMapping scm = new SearchCriteriaMapping(myTag, sc);
    MyMappings[i.previousIndex()] = scm;
}
```

Although the tags are entirely arbitrary, the value specified in the above call to *putCriteria()* is *not*. The tags could just as well have been zipcodes, but the values to *putCriteria()* must correspond to those specified in the selection node's search criteria object.

Our *GridSearchCriteria* can now be constructed as follows:

```
GridSearchCriteria gridCriteria = new GridSearchCriteria( tree, myMappings );
```

---

\* The *SelectionNode* class, like many of the objects in the search packages, is actually an interface class that is implemented by an *Impl* class.

## 10.4 Executing the Search and Interpreting the Results

We are now ready to execute the search by constructing an *ObjectGrid* and invoking its search method:

```
ObjectGrid myObjGrid = new ObjectGridImpl();
GridSearchResultMapping[] results = myObjGrid.search(gridCriteria);
```

The results of the search are returned as an array of *GridSearchResultMappings*. All of these results refer to the root node of the query tree. For each attribute value that was originally supplied to the root node's search criteria object, a new *GridSearchResultMapping* is included in the results. The order in which these results are returned exactly follows the order in which the attribute values were provided.

Each *GridSearchResultMapping* contains (1) a string specifying the attribute value that the result corresponds to; and (2) an array of type *GridRow[]*. The attribute value can be extracted using *GridSearchResultMapping.getClientData()*. To extract the array of *GridRows*, the method *GridSearchResultMapping.getResult()* is used.

In the simple example we have built thus far, each result mapping will contain a *GridRow* array of size (1), as our query tree contained only a single selection node. For each attribute that was specified, the grid row will contain a cell holding the value for that attribute. Because we used only the gene's name as a search criteria, each grid row will in this case contain a single cell holding the value of *name* for that gene.

In summary, our simple example will return at most (3) *GridSearchResultMappings*, where each of these will provide access to a single *GridRow* object containing a single value. The cells are accessed using the row's *getCell()* method, and the values are then accessed using the cell's *getObject()* method.

The argument to the *getCell()* method is a string constructed by appending each of the node names together using a dot (".") to concatenate them, followed by the attribute name. Thus, to access our results, we could use the following code:

```
for( int i = 0; i < results.length; i++ ){
    GridSearchResultMapping resultMapping = results[i];
    String attVal = (String)resultMapping.getClientData();
    GridRow[] rows = resultMapping.getResult();
    GridCell myCell1 = rows[0].getCell( "Gene.name" )

    System.out.println( "for criteria = " + attVal +
        " the gene name is: " + myCell.getObject() );
}
```

Here, the single node is named "Gene", and the attribute name "name" is appended to it to identify the cell in the row.

We began with an exceedingly simple example in order to lay out the complexities of using the advanced search paradigm in a straightforward manner. The potential power of the advanced search is better illustrated in the next example.

## 10.5 Building More Complex Queries

This example extends the previous case by inserting a new search criteria to the original query tree. As before, we begin by initializing the query tree with a *GeneSearchCriteria* that

searches for genes by name We then add a *PathwaySearchCriteria* to the tree, and select a couple of attributes of interest for the associated pathways. The *PathwaySearchCriteria* is contained in a new selection node that is embedded in the tree using the *DefaultMutableTreeNode.insert()* method. Figure 10.5-1 shows the complete code.

```
// initialize the query tree with a GeneSearchCriteria
List geneNames = new ArrayList();
geneNames.add( "PTEN" );
geneNames.add( "TP53" );
geneNames.add( "BRCA1" );
GeneSearchCriteria gsc = new GeneSearchCriteria();
gsc.putCriteria( "name", geneNames );

List geneAtts = new ArrayList();
geneAtts.add( "name" );
SelectionNode tree = new SelectionNodeImpl( "Gene", gsc, geneAtts );

// now embed a PathwaySearchCriteria
List pathAtts = new ArrayList();
pathAtts.add( "name" );
pathAtts.add("displayValue");
SelectionNode pathNode =
    new SelectionNodeImpl( "pathways", new PathwaySearchCriteria(), pathAtts );
tree.insert( pathNode, 0 );

// Construct the SearchCriteriaMappings array and GridSearchCriteria
SearchCriteriaMapping[] myMappings =
    new SearchCriteriaMapping[ geneNames.size() ];

for( ListIterator i = geneNames.listIterator(); i.hasNext(); ){
    GeneSearchCriteria sc = new GeneSearchCriteria();
    String myTag = (String)i.next();
    sc.putCriteria( "name", myTag);
    SearchCriteriaMapping scm = new SearchCriteriaMapping(myTag, sc);
    myMappings[i.previousIndex()] = scm;
}

GridSearchCriteria gridCriteria = new GridSearchCriteria( tree, myMappings );

// Execute the search and process the results
ObjectGrid myObjGrid = new ObjectGridImpl();
GridSearchResultMapping[] results = myObjGrid.search(gridCriteria);

for( int i = 0; i < results.length; i++ ){
    GridSearchResultMapping resultMapping = results[i];
    String attVal = (String)resultMapping.getClientData();
    GridRow[] rows = resultMapping.getResult();
    GridCell myCell1 = rows[0].getCell( "Gene.name" );

    System.out.println( "\n" + rows.length + " results for " + attVal + "\n");
    for( int j = 0; j < rows.length; j++ ){
        GridCell cell1 = rows[j].getCell( "Gene.name" );
        GridCell cell2 = rows[j].getCell( "Gene.pathways.name" );
        GridCell cell3 = rows[j].getCell( "Gene.pathways.displayValue" );
        if (cell1 != null && cell2 != null && cell3 != null){
            System.out.println( cell1.getObject() + ": " +
                cell2.getObject() + "( " +
                cell3.getObject() + " )" );
        }
    }
}
}
```

**Figure 10.5-1 Assembled Code for Example 2**

In this example we are looking for pathways relative to the genes we are searching for, and the tree we have constructed is analogous to a *GeneSearchCriteria* containing an embedded *PathwaySearchCriteria*. In the basic search method, we would have constructed this by invoking the *GeneSearchCriteria*'s *putSearchCriteria ()* method on the *PathwaySearchCriteria*.

In the advanced search method, each time we insert a new selection node in a previously defined node, the name of the new selection node must reflect the relation that exists between the two corresponding search criteria objects. In this case, the name must define the relation that exists between genes and pathways. Because the root of the tree holds a *GeneSearchCriteria*, the name must correspond to that attribute of the *Gene* class that stores its associated pathways.

Consulting the field summary in the JavaDocs for the *Gene* class, we find the attribute "pathways" listed, so this is what we use for the node name. Because we are not using the pathways as a search filter however, we do not need to specify any values for the attributes. The attributes stored with the selection node are used in this case to define features we wish to be included in the results.

A portion of the output generated from running this example is shown in Figure 10.4-2; a complete listing of the executable source code (which you can compile and run) is included in Appendix G.

6 results for PTEN

PTEN: mtorPathway( mTOR Signaling Pathway )  
PTEN: ptenPathway( PTEN Dependent Cell Cycle Arrest and Apoptosis )  
PTEN: eif4Pathway( Regulation of eIF4e and p70 S6 Kinase )  
Pten: m\_eif4Pathway( Regulation of eIF4e and p70 S6 Kinase )  
Pten: m\_mtorPathway( mTOR Signaling Pathway )  
Pten: m\_ptenPathway( PTEN dependent Cell Cycle Arrest and Apoptosis )

15 results for TP53

TP53: g1Pathway( Cell Cycle: G1/S Check Point )  
TP53: p53Pathway( p53 Signaling Pathway )  
TP53: maPathway( Double Stranded RNA Induced Gene Expression )  
TP53: tidPathway( Chaperones Modulate Interferon Signaling Pathway )  
TP53: atmPathway( ATM Signaling Pathway )  
TP53: pmlPathway( Regulation of Transcriptional Activity by PML )  
TP53: efpPathway( Estrogen-responsive protein Efp controls cell cycle and breast tumors growth )  
TP53: arfPathway( Tumor Suppressor Arf Inhibits Ribosomal Biogenesis )  
TP53: g2Pathway( Cell Cycle: G2/M Checkpoint )  
TP53: telPathway( Telomeres, Telomerase, Cellular Aging and Immortality )  
TP53: plk3Pathway( Regulation of Cell Cycle Progression by Plk3 )  
TP53: p53hypoxiaPathway( Hypoxia and p53 in the Cardiovascular System )  
TP53: rbPathway( RB Tumor Suppressor/Checkpoint Signaling in Response to DNA Damage )  
TP53: atrbrcaPathway( Role of Brac1, Brac2 and Atr )  
TP53: tertPathway( Overview of telomerase protein component gene hTert Transcriptional Regulation )

(additional output continues)

Figure 10.5-2 Sample Output from Example 2.

The capabilities defined in the search packages were developed in concert with the caBIO web interface, BIOgopher. The source code for BIOgopher, which was implemented using the search methods described here, is also available for download at the [NCICB Download](#) page.

For a comparison of the basic and advanced search methods, Appendix G also includes an example demonstrating how the same search might be implemented using the basic search methods. The BIOgopher user interface itself is described fully in the NCICB Applications User Manual. For more complete specifications of the search packages, consult the JavaDoc pages for those packages.

## 10.6 Roles and Attributes

A common theme in all of the search methods—throughout all of the APIs—is the use of object attributes to selectively retrieve information from the database. For example, the following pseudo-SQL statement would retrieve all human genes whose symbol is BRCA1:

```
Select genes from geneTable where symbol = 'BRCA1' and taxonId in
(select taxonId from taxonTable where scientific_name = 'homo sapiens')
```

Using the Java API's basic search method we could formulate this query as:

```
Gene myGene = new Gene();
GeneSearchCriteria gsc = new GeneSearchCriteria();
gsc.setSymbol("BRCA1");
gsc.setTaxonId(new Long("5")); // homo sapiens taxon Id is 5
myGene.search(gsc);
```

An equivalent expression using the HTTP “operation=” syntax (see [Chapter 14](#)) would then be:

<http://cabio.nci.nih.gov/servlet/GetXML?operation=Gene&Symbol=BRCA1&taxonId=5>

Finally, using the SOAP web services (see [Chapter 13](#)), we might use the *GeneService* to call *getGenes* with the argument: symbol=BRCA1 and taxonId=5 .

In each case, we begin by defining the type of objects we would like to retrieve, and then specify the values those objects must exhibit for selected attributes. In the case of simple attributes such as name or id, we can directly specify the values we are seeking. But when the attribute involves a second object that is in some kind of relation to the first, the situation becomes more complex. The complexity arises when we wish to retrieve instances of the first object that are in relation to some second object whose attributes are the ones we wish to constrain.

The previous caBIO releases (versions 1.x) had some convenient methods such as *setTaxonId()* in the *GeneSearchCriteria* to accomplish such queries., but these methods assumed that you knew the proper associations beforehand. A clear draw back of the *setTaxonId* method is that in order to use it, one must know that “5” is the taxonId for “homo sapiens” and “6” is the taxonId for “Mus musculus”, etc.

In the initial release of caBIO, there were several objects—such as Gene, Sequence, Taxon, Clone, Library, Protocol, and Pathway—that had settable attributes in the search criteria classes

that did not map 1:1 with the properties of the domain objects. The *taxonId* attribute is one such example. One could constrain the search for Genes by specifying the *taxonId*, but the *taxonId* could not be accessed as a local property of the Genes in the result set.

The criterion “*taxonId*” literally means “Gene.taxon.id”. That is, “id” refers to a property that is local to a Taxon *associated* with the Gene. In caBIO 2.0, we refer to these types of criteria as “non-normalized.” There are currently many non-normalized criteria in the subclasses of *gov.nih.nci.caBIO.bean.SearchCriteria*, all of which will eventually be phased out in future releases. The new object models (caMOD, caDSR, and EVS) do not support such non-normalized criteria.

Previous releases of caBIO addressed this issue by providing the *putSearchCriteria()* method, which allows the search to be constrained by attributes defined on the “nested” search criteria objects. Using this approach, the following code snippet reproduces the results obtainable from the previous example:

```
Gene myGene = new Gene();
GeneSearchCriteria gsc = new GeneSearchCriteria();
gsc.setSymbol("BRCA1");
TaxonSearchCriteria tsc = new TaxonSearchCriteria();
tsc.setScientificName("homo sapiens");
gsc.putSearchCriteria(tsc);
myGene.search(gsc);
```

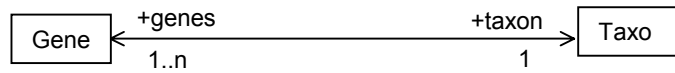
This solution however, was still limited by the need to have predefined “setXxx()” methods for all of the attributes users might wish to use as search criteria. What was needed was a dynamic query mechanism that would allow caBIO users to supply not only the criteria values at runtime, but the attribute tags as well. In this way, any property that was defined for the domain object would immediately become “settable” as an attribute for the associated search criteria object.

This led to development of the *putCriteria()* method in the Java API, and to the new *query=* syntax in the HTTP interface (see [Section 14.2.2](#)). For example, to retrieve all *human* genes whose symbols match “BRCA1” using the Java API, we can replace the previous sample code with the following:

```
Gene myGene = new Gene();
GeneSearchCriteria gsc = new GeneSearchCriteria();
gsc.putCriteria("name", "BRCA1");
TaxonSearchCriteria tsc = new TaxonSearchCriteria();
tsc.putCriteria("scientificName", "homo sapiens");
gsc.putCriteria("taxon", tsc);
myGene.search(gsc);
```

While this solved one problem, it introduced another. In order to use the *putCriteria()* method to define simple (non-nested) attributes you must know the value (e.g. “BRCA1”) as well as the caBIO bean attribute name (e.g. “Gene.name”). Moreover, for “nested” attributes, you must know the *association role* name that signifies a relationship between the two objects. Figure 10.6-1 illustrates the UML model and the association role names for this example.





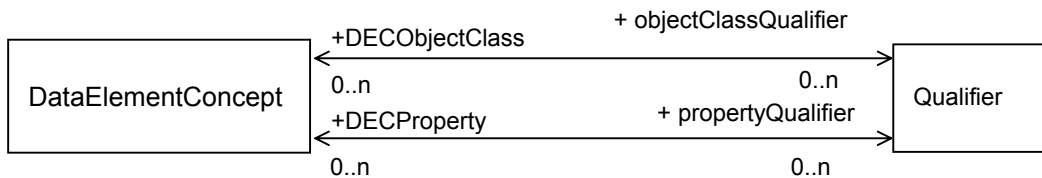
**Figure 10.6-1 The association roles between Gene and Taxon**

In this example, “taxon” is the association role name from the Gene to the Taxon, and “genes” is the association role name from the Taxon to the Gene. Note that the plurality of the role name captures the cardinality of the relation as it is diagrammed in the UML model.

In many cases the association role name can be generated by these guidelines:

1. Begin with the class name of the related object and convert only the first letter to lower case, leaving the rest of the name unchanged.
2. If the cardinality of the relation is greater than 1, pluralize the name.

These simple rules break down however, when the two objects have *multiple* relations to one another, as in the following example from the caDSR API.



**Figure 10.6-2 Objects with multiple association roles**

The above rules are just guidelines; the precise nomenclature for these attributes and association role names can be found in the caBIO [JavaDocs](#), as the Field names for each bean object, and, in the [caBIO object models](#), as attribute names for the individual objects and as role names for associations between objects.

## **11.0 THE caBIO Java API**

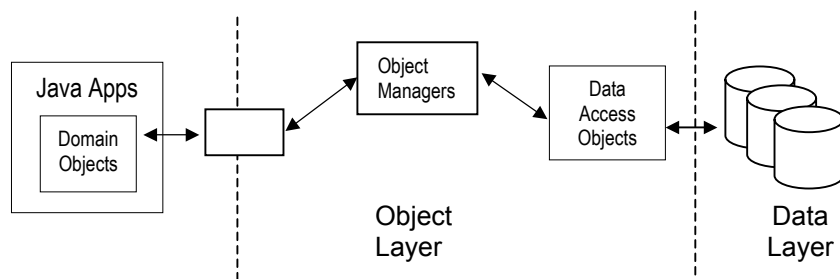
## 11.1 Installing a caBIO Client

The caBIO Java API provides an enhanced object-oriented development environment for bioinformatics researchers, along with access to customized data sources for plumbing the molecular basis of cancer and manipulating clinical data. The data sources include many of the NCICB databases specially curated for cancer research, as well as several of the public databases at NCBI and the Distributed Annotation Server at UCSC. These data sources are described in more detail in [Chapter 15](#).

The caBIO objects in the *bean* packages—also referred to as the domain objects—simulate the behavior and relationships of actual bioinformatic and biomedical components such as genes, chromosomes, sequences, animal models, clinical trials, protocols, etc. They provide access to a variety of data sources including Unigene, LocusLink, Homologene, GoldenPath, and NCICB's CGAP (Cancer Genome Anatomy Project) data repositories. Hence, a gene can get its ESTs, SNPs, or clones; a SNP can provide access to the TraceFiles that were used to identify it; and a chromosome can report the taxon in which it is defined.

Java applications can access the caBIO data sources directly through these domain objects; the network details of communication to the data servers are abstracted away from the developer by the supporting packages described in [Chapter 9](#). Thus, developers need not deal with issues such as RMI and can instead concentrate on the biological problems at hand.

The implementation of a separate data layer allows the domain objects to act independently of the actual data storage facilities. This allows the data layer to migrate as necessary to enhance performance or provide new data stores, without impact on the application programs. In particular, the Data Access Objects (DAOs) in the *db* packages enable platform-independent persistence of these domain objects, and the relational mappings provided by the DAOs are optimized for the data queries presented by the domain objects.



**Figure 11.1-1 The caBIO Java API**

Another important feature of the caBIO data access objects is their ability to cache and manage large amounts of data. Coupled with the throttling mechanisms deployed to control the flow of data through the system, this design provides optimal response time to all users of the system.

The caBIO domain object classes are what most developers will use to access the information available from the caBIO servers. These domain objects are available as Java beans in the caBIO jar file that is downloaded with the caBIO installation. In most cases, the developer need not look beyond the caBIO *bean* package to accomplish his or her goals.

More ambitious applications may require adapting and extending the basic caBIO platform, and/or installing the data sources as a local resource. The entire caBIO source code is available for download, and the complete installation of the caBIO server is described in the next section. As an example of the types of applications that can be built using these development tools, visit the [CMAP](#) web site, which is implemented using the caBIO objects described in this guide.

### 11.1.1 Requirements for Installing the caBIO Java Client

The caBIO Java API was developed and tested using JDK 1.3.1. Later versions of the JDK (e.g., 1.4.1) may have deprecated certain methods used by caBIO, and warnings may be generated at compile time. If JDK is not already installed in your system, follow the instructions from the [Java installation and tutorial](#) web site for details on installing JDK.

The caBIO Java API can be downloaded from the [NCICB Download](#) web site. After filling in your user name, institution, and email address, you are given the option of downloading the caBIO distribution, which contains the caBIO.jar file, code examples written in Java and Perl, and a PDF of the UML model.<sup>8</sup> The caBIO.jar file is a Java archive file that defines all of the caBIO domain objects, as well as the protocols and server information required to issue RMI requests to the caBIO servers. This discussion assumes you have downloaded this package.

Unzip these files into a working directory of your choice; for the purposes of this discussion it is assumed you are using c:\caBIO. Examine the file structure in your newly created directory. In addition to the top-level files, you will find several subdirectories. The directory named *jars* contains the caBIO Java archive, *caBIO.jar*, along with the other following jar files:

- [xercesImpl.jar](#) (the Xerces Java Parser)
- [xml-apis.jar](#) (XSLT processor)
- [jaxp.jar](#) (the Sun API for XML processing)
- [jaxb-rt-1.0-ea.jar](#) (the Sun architecture for XML binding)
- [soap.jar](#) (Apache SOAP)

### 11.1.2 Defining the ClassPath

In order to compile and/or execute caBIO applications, the Java compiler and runtime environments must be able to locate the caBIO class definitions as well as those for the classes in the additional jar files. This can be accomplished in three ways: (1) you can use a compile tool such as *ant*; (2) you can use the scripts provided with the caBIO download (*.bat* for Windows and *.sh* for Unix); or (3) you can set the CLASSPATH environment variable directly.

The first two methods are preferable, as hardcoding the locations of Java archive files in your environment can create problems with versioning. Instructions for using the *ant* utility are included in the *readme* file that accompanies the distribution.

The two script files contained in the caBIO distribution are *compile\_caBIO.\** and *run\_caBIO.\**. The compilation script explicitly specifies the classpath as an argument to the *javac* compiler. The execution script also specifies the classpath and, in addition, specifies the java.security file using the *-D* define flag. If you plan to use either of these batch files, ensure that the classpaths they specify concur with your installation of the corresponding jar files.

---

<sup>8</sup> This guide can also be downloaded from that site.

To set the `CLASSPATH` environment variable directly, Windows 98 users should modify the `autoexec.bat` file by adding the following (single) line:

```
CLASSPATH=%CLASSPATH%;c:\cabio\jars\xercesImpl.jar; c:\cabio\jars\xml-apis.jar;c:\cabio\jars\caBIO.jar;c:\cabio\jars\soap.jar;c:\cabio\jars\jaxp.jar;c:\cabio\jars\jaxb-rt-1.0-ea.jar;.
```

Note that this works only if you have previously defined the `CLASSPATH`; if not, you must use:

```
Set CLASSPATH=c:\cabio\jars\... instead of CLASSPATH=%CLASSPATH%;c:\cabio\jars\...
```

Users of Windows NT and Windows 2000 can enter this information directly, by clicking on *My Computer* → *Properties*, and selecting the *Advanced* tab, which brings up a dialog box for editing your environment variables.

### 11.1.3 Compiling and Running the GeneDemo Program

The *JavaDemos* directory contains both the source code and the executable class for a Java program called `GeneDemo.java`. Using the *.bat* file on Windows machines, you can now run the demo by typing:

```
run_caBIO.bat GeneDemo
```

The screen shot in Figure 11.1-2 captures the first page of output that results from executing the *GeneDemo.class* file. Additional lines of output are also listed in Appendix A of this guide. Using the *compile\_caBIO.bat* file on Windows machines, you can compile the demo by typing:

```
compile_caBIO.bat JavaDemos\GeneDemo.java
```

at the command line in a DOS shell. Alternatively, if you have defined the classpath environment variable, you can just use

```
javac GeneDemo.java
```

This will reproduce the java class file named `GeneDemo.class`, which you can then execute by again using the `run_caBIO.bat` script or typing directly:

```
java -Djava.security.policy=java.policy GeneDemo
```

The `-D` flag defining the security policy is provided to the `RMISecurityManager` class, which requires that you specify a security policy at runtime. The policies defined in the *java.policy* file protect your system — not the caBIO server — and you are free to edit these as you see fit. For example, a policy file granting full access permissions to everyone would contain the text:

```
grant {
    permission java.security.AllPermission;
};
```

By default, the policy file that you downloaded grants all permissions. The commented out section is an example of alternative settings you may wish to use.

### 11.1.4 Troubleshooting

The screen shot in Figure 11.1-2 captures the first page of output that results from executing the *GeneDemo.class* file. If you do not see this, the first place to look for errors is in the `CLASSPATH` definition; verify that all of the required jar files are present and under the correct

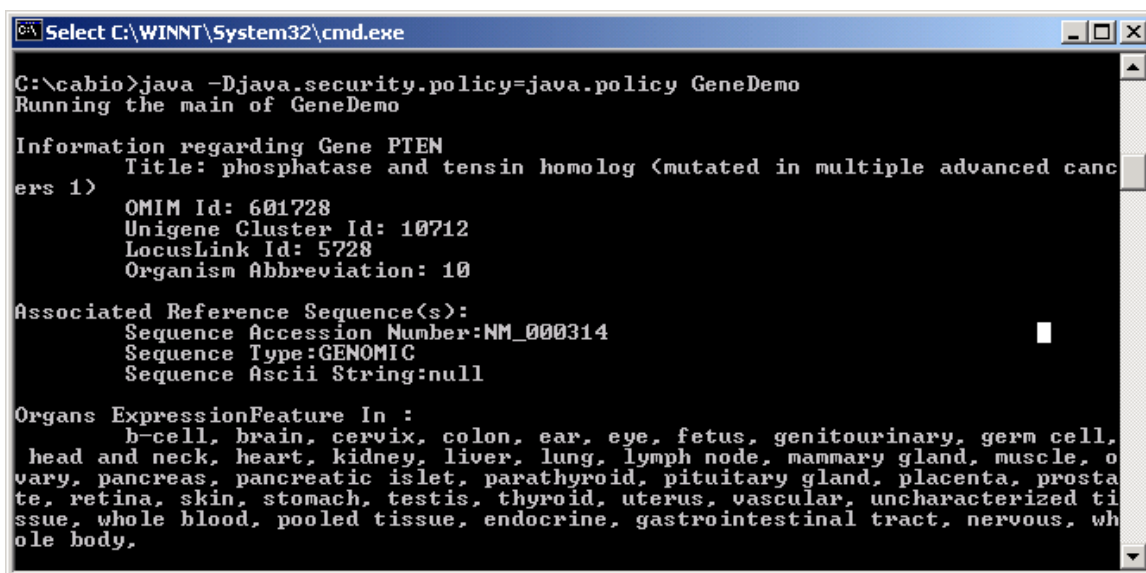
subdirectories. It is also possible that you have installed everything correctly but that the executable has hit a firewall in trying to access the caBIO data services. For example, if you see the first line of output:

```
Running the main of GeneDemo
```

followed by the error message:

```
Proxy unable to contact Gene manager! Connection refused to host:...
```

then you have hit a firewall on your system and need to ask your system administrator to open the ports that the caBIO data services are using. Open another shell window and run the `netstat` system utility (while simultaneously running *GeneDemo*) to identify these ports.



```
Select C:\WINNT\System32\cmd.exe
C:\cabio>java -Djava.security.policy=java.policy GeneDemo
Running the main of GeneDemo

Information regarding Gene PTEN
  Title: phosphatase and tensin homolog (mutated in multiple advanced cancers 1)
  OMIM Id: 601728
  Unigene Cluster Id: 10712
  LocusLink Id: 5728
  Organism Abbreviation: 10

Associated Reference Sequence(s):
  Sequence Accession Number: NM_000314
  Sequence Type: GENOMIC
  Sequence Ascii String: null

Organs Expression Feature In :
  b-cell, brain, cervix, colon, ear, eye, fetus, genitourinary, germ cell,
  head and neck, heart, kidney, liver, lung, lymph node, mammary gland, muscle,
  ovary, pancreas, pancreatic islet, parathyroid, pituitary gland, placenta,
  prostate, retina, skin, stomach, testis, thyroid, uterus, vascular,
  uncharacterized tissue, whole blood, pooled tissue, endocrine,
  gastrointestinal tract, nervous, whole body.
```

Figure 11.1-2 Screen shot of GeneDemo output

A second problem, which can produce similar error messages like:

```
Proxy unable to contact Core manager! Connection refused to host:...,
```

can arise if you have:

- (1) redefined the *java.policy* file, and/or
- (2) installed the *cabio.jar* file in some location other than the default configuration.

In this case, the program will get past the Gene manager and produce the first few lines of output before running into trouble. To test this, redefine the *java.policy* file to grant all permissions (as outlined in the previous subsection), and rerun the program. If this solves the problem, then you will have established that the Core manager class (defined in *cabio.jar*) was not able to locate the *java.policy* file. If you are not comfortable with using the open security policy, then you will need to reconfigure your setup so that the policy file and *cabio.jar* file have the following organization:

```
..\caBIO\  
  java.policy
```

```
jars\  
cabio.jar
```

For a discussion of the GeneDemo program itself, see [Chapter 12](#). In addition to a description of the GeneDemo code that accesses the caBIO domain objects, other short example programs (listed in the Appendices) are discussed which demonstrate the use of the Java API to access the caDSR, EVS, caMOD, and MAGE-OM API domain objects.

## 11.2 Installing the caBIO Server

The complete source code for installing the caBIO server and caBIO database, as well as the BIOgopher web interface, is available for download at:

<http://ncicb.nci.nih.gov/download/>

Select the file named “caBIO\_source\_ver2-0.zip” and follow the detailed step-by-step instructions that are provided in the “Readme.txt” file. In addition to the materials provided at the caBIO web site, Windows users will also need the following open source software:

- [J2EE](#) (Java 2 Platform Enterprise Edition) or [J2SE](#) (Standard Edition)
- [Ant](#) (Apache’s Java-based build tool);
- [Cygwin](#) (provides Linux-like environment for Windows); and
- [Tomcat](#) (for implementing Java Servlets and JSPs).

The complete installation is not difficult but can get quite complicated if the steps outlined in the Readme file are not followed in the exact order specified there. Help is also available at the [NCICB Application Support](#) web site, which provides phone numbers, email addresses, and a knowledgebase of Frequently Asked Questions (FAQs). Two email discussion forums are also accessible from the [caBIO home](#) page:

- A Users' Discussion Forum—for users of the caBIO Java, SOAP, and/or HTTP APIs can be found at:

[http://list.nih.gov/archives/cabio\\_users.html](http://list.nih.gov/archives/cabio_users.html)

- A Developers' Discussion Forum, for developers who are extending the API, adding additional data sources, and/or enhancing the existing source code, can be found at:

[http://list.nih.gov/archives/cabio\\_developers.html](http://list.nih.gov/archives/cabio_developers.html)

## **12.0 CODE EXAMPLES**



Several small Java demonstration programs are included with the download package. The source listings for these programs are also included here as appendices, along with the sample output you should see after running the programs. The sections that follow comment on these appended source listings and their output.

## 12.1 The caBIO GeneDemo program

The *GeneDemo.java* program is in [Appendix A](#). The most important method—indeed, the paradigmatic operation—on a caBIO object is the *search* method. Corresponding to each domain object is a *SearchCriteria* object that can be deployed to retrieve objects of the type that satisfy user-specified criteria. For example, to obtain information about a particular gene:

1. Instantiate a new *Gene* object (e.g., *myGene*).
2. Instantiate a new *GeneSearchCriteria* object (e.g., *criteria*).
3. Set the attributes of the *GeneSearchCriteria* to limit the search.
4. Call the *Gene* object's search method with the *GeneSearchCriteria* as the argument.

This approach is used in *GeneDemo.java* to retrieve all instances of *Gene* objects whose symbols match the string "PTEN." Specifically, the *setSymbol()* method of the *GeneSearchCriteria* is first applied, and a subsequent call to *myGene.search(criteria)* is then executed. Note that the return result needs to be typecast to *Gene[]*, as the *SearchResult* object's method *getResultSet()* returns a generic container.

The *Gene* object in this example and, more generally, every caBIO domain object, should be viewed as a "factory" that enables the procurement of a collection of caBIO objects of that same type. Reviewing the steps outlined above, this manufacturing process can be generalized to:

1. Instantiate a *new* domain object of the desired type.
2. Instantiate a *new SearchCriteria* to be associated with that domain object, and set the attributes of that search criteria object so as to limit the search.
3. Execute the domain object's *search()* method on that *SearchCriteria*, and store the results in a generic *SearchResult* object.
4. Invoke the *getResultSet()* method on the *SearchResult* object and typecast its return value to an array of the same type as the original domain object.

In this example, the search finds an array of genes, and the code then explores the features of each one in turn. Simple features whose values are just strings or numbers are printed directly to the screen. These include attributes like the gene's name, title, OMIM id, Unigene cluster id, LocusLink id, and organism abbreviation. More complex features represent embedded objects or arrays of simple elements or objects, and must be explored recursively.

For example, the gene's *getReferenceSequences()* method returns an array of sequences, and the features of each *Sequence* object are explored in turn. Similarly, the gene's *getDbCrossRefs()* method returns a hashtable or associative array of key/value pairs. The keys are stored in a Java enumeration variable and used to access successive element values. In contrast, the gene's *getExpressionFeature()* method does not return an array, but an *ExpressionFeature* object. In this case, the object's *getExpressedInOrgans()* method produces an array of string values that is iterated over.

The previous section described the caBIO architecture and the underlying design that drives the logistics of the *search()* methods and interactions between the domain objects, object managers, *SearchCriteria*, and *SearchResult* objects. The GeneDemo.java program demonstrates how these objects and devices can be deployed to extract information from the caBIO data sources

## 12.2 The EVSDemo Program

The EVSDemo program in [Appendix B](#) begins by instantiating a *DescLogicConcept* and a *MetaThesaurusConcept* along with search criteria objects that can be used in conjunction with these domain objects. The first demonstration is the *DescLogicConcept*'s generic *search()* method. In this example, a "\*" is used to force wildcard matching—all concepts whose names begin with the string "Gen" will be returned. The search is invoked with the search criteria object as the single argument:

```
Concept[] conceptArray = dlc.search(dlcsc);
```

As described in [Section 4.6](#), the values returned by this statement will be a collection of lightweight *Concept* objects whose only defined attributes are the concept names. This method is useful when all that is required is the concept names, or, when the complete *DescLogicConcept* is desired, but the precise name is not known. Using the generic *search()* method to first obtain the exact name enables the user to subsequently invoke the *getConceptByName()* method—as demonstrated in the next several lines.

Although this method returns a fully defined *DescLogicConcept* that can they be used to directly access all of the attributes for that concept, it is cast as a simple concept here. Instead, the example goes on to demonstrate how convenience methods can be applied to get this same information without requiring the description logic concept itself. The methods demonstrated are *GetPropertiesByConceptName()* and *getRolesByConceptName()*.

Next, the *History* class is deployed to obtain information about the ancestors and descendants of the concept as well as the editing that has been performed on it. Finally, the methods that access information about the subconcept and superconcept relations in the hierarchy are exercised. As described in [Chapter 4](#), this information can also be accessed from the description logic concept itself. This ability to access hierarchical and historical information without accessing the concept itself however, is a great time-saving device for NCI Thesaurus concepts as the proprietary API to the Thesaurus entails significant overhead.

The retrieval of *MetaThesaurusConcept* objects is quite straightforward, as the *search()* method for this class returns these objects directly. The section of the code beginning with the comment "MetaThesaurus Search" demonstrates this, using the *printConceptArray()* method.

## 12.3 The caDSR Demo Program

The caDSR example in [Appendix C](#) also uses the domain object/search criteria paradigm to selectively retrieve information from the caDSR registry. The domain object is a *CaseReportForm*, and the attribute used as a filter is the *id* attribute. The program will retrieve all instances of *CaseReportForm* whose *id* attribute is equal to the value specified in the *setId()* method. The result set is then passed to the *printOutResults()* method, which demonstrates some of the features of the domain object and how they can be accessed.

As described in [Chapter 8](#), the settable attributes of a search criteria object can be obtained from the corresponding catalog entry in that chapter or, alternatively, from the [JavaDocs](#), by examining the *set* methods of that object. Note that in this example the method used is actually inherited from the parent object, *AdministeredComponentSearchCriteria*.

Once the results have been obtained, the loop visits each result and prints information to the screen about that *CaseReportForm*. As noted in the comments, the program does not expose all attributes of the form, but only a few, for the purposes of demonstrating how this can be done. The simple features displayed by the program include the object's *PreferredName*, *Id*, *PreferredDefinition*, *LongName*, *Version*, and *WorkflowStatusName*. As all of these are simple strings, they are displayed directly.

The attributes of several embedded objects contained in the *CaseReportForm* are also explored, using nested loops. In all cases, the *get* methods applied to both the *CaseReportForm* objects, as well as to their embedded objects, are available on the JavaDocs pages.

## 12.4 The caMOD Demo Program

The caMOD demo program in [Appendix D](#) repeats a similar sequence of steps over the caMOD domain objects *TreatmentSchedule*, *Publication*, *MicroarrayData*, and *Phenotype*. Again, the domain object/search criteria paradigm is used to selectively retrieve information from the caMOD database. In each case, the domain object and an associated search criteria object are instantiated; some attribute is set for the search criteria object; and the domain object's *search()* method is invoked with the search criteria object as its single argument. The result set is captured in an array of domain objects, which is then iterated over to expose its features.

## 12.5 The MAGE-OM Demo Program

The MAGE-OM API represents a significant departure from the other caBIO domain-specific APIs. Unlike these other applications, the MAGE-OM API is not integrated with the caBIO objects at the package level. As described in [Chapter 7](#), MAGE-OM-compliant objects are implemented as Java interfaces, which the custom MAGE-OM *Impl* objects implement as concrete java classes.

The example program included with the MAGE API distribution (*MageTest.java*) is also listed in [Appendix E](#), along with a sample of the output from this program. *MageTest.java* demonstrates how to access the microarray experiments stored in the GEDP database and the bioassay data associated with those experiments.

A single class, *MageTest*, is defined, with four methods, two of which can be used to extract experiments from the database. The first of these, *getExperimentWithId()*, uses the experiment Id provided with the top-level arguments at the command line to retrieve the corresponding experiment. Note that in this case, a new *ExperimentImpl* object is simply instantiated with the desired id. The object resulting from this operation is a GEDP experiment with that Id (assuming one exists).

If no arguments were provided, the test program calls the second method, *getExperiments()*. In this case, an “empty” *ExperimentImpl* object is first instantiated, and a generic *SearchCriteria* object is used to generate results. The *ExperimentImpl* objects contained in these results are then collected and returned by the method as an array of type *ExperimentImpl[]*. Because no selection criteria were set, the entire collection of GEDP experiments will be returned. In both cases, the

*dumpExperimentBioAssays()* method is then called to output the bioassay data, which in turn, calls *dumpBioDataValues()*.

## **13.0 THE SOAP API AND WEB SERVICES**

The Simple Object Access Protocol ([SOAP](#)) is a bridging technology that allows heterogeneous peers on diverse platforms to exchange structured data over the Internet via XML and HTTP. The caBIO project provides a SOAP interface for non-Java applications. In this model the client issues XML-encoded requests specifying the desired data services to the appropriate host address and port and, in exchange, receives XML-encoded responses.

The SOAP engine operates on three types of specifications:

- The SOAP message's *envelope* specifications, which define the content type, intended recipient of the message, and whether it is optional or mandatory;
- The encoding rules, which specify the serialization method to be used in the exchange of application-specific data; and
- The RPC (remote procedure call), which defines the conventions used in remote procedure calls and responses.

The simple example of a SOAP message below requests the price of apples from a host, [www.foodprices.com](http://www.foodprices.com):

```
POST /InPrices HTTP/1.1
Host: www.dictionary.com
Content-Type: application/soap; charset=utf-8
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.dictionary.com " />
    <m:GetDefinition>
      <m:Item>Aperture</m:Item>
    </m:GetDefinition>
  </soap:Body>
</soap:Envelope>
```

In this example, the SOAP envelope specifications include only the content type (application/soap) and the recipient (www.dictionary.com). There is no mention of whether the message is mandatory or optional. The encoding rules are defined as

<http://www.w3.org/2001/12/soap-encoding>

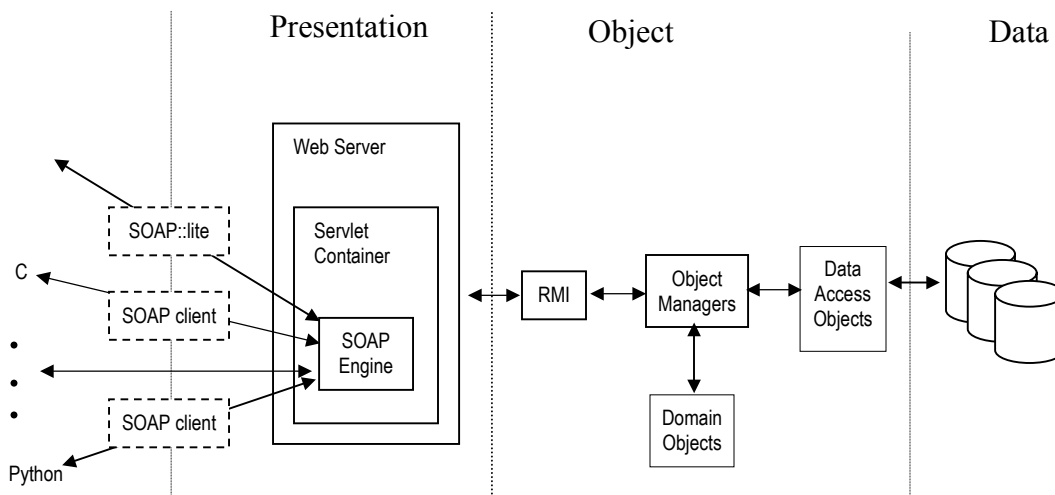
The RPC call specification is defined in the <soap:Body > element, where it requests the price of apples from the server.

### 13.1 The SOAP API and caBIO

As depicted in Figure 13.1-1, all of the caBIO data sources—including the internal databases as well as other NCI resources and external web sites—are at the backend of the caBIO infrastructure. The Object Layer consists of a set of object managers, data access objects, and a collection of classes representing biological and bioinformatic entities. All of the objects in this layer are implemented as Java bean classes and, as such, can be accessed by a Java program using remote method invocation.

The Presentation Layer provides a more generic interface to these same data for non-Java applications written in languages such as Perl, C/C++, Python, etc. caBIO uses the open-source

Apache SOAP package, in combination with appropriate serialization methods for the Java beans, to achieve an application-independent interface. As described in the general discussion of the caBIO APIs in [Chapter 8](#), all of the domain objects are “XML Aware” and are capable of serializing themselves to XML for transport.



**Figure 13.1-1 The caBIO architecture and the SOAP interface**

caBIO’s SOAP API can be used as an interface to *any* language-specific application. In theory, the remote application could in fact communicate directly with the SOAP server without any additional layers of interfacing. In practice, however, this would involve a good deal of effort, as it requires explicitly wrapping each request in a SOAP envelope, parsing the return message types, and network programming to establish and maintain reliable connections.

Most developers instead prefer to install a SOAP client package to handle the implementation of the envelope and the resolution of the SOAP types. A number of SOAP packages catering to different programming languages are available at <http://www.w3.org/TR/SOAP/> and at [Soapware](#).

One such package is SOAP::LITE for Perl, which can be freely downloaded from the [ActiveState](#) web site . The PERL example discussed below utilizes SOAP::LITE, which provides both a client- and a server-side SOAP implementation. Win 32 machines can download the .exe file from ActiveState and simply follow the accompanying installation instructions. The installer will automatically add the <perl soap:lite installation-directory>/bin to the system PATH.

## 13.2 Using the SOAP API with Perl and SOAP::LITE

### 13.2.1 Accessing the caBIO SOAP Services

The first thing you will need to determine before connecting to a SOAP server is the set of callable services it provides. To see the list of SOAP services provided by the NCICB server, point your browser to:

<http://cabio.nci.nih.gov/soap/services/index.html>

The links displayed on the deployed services page are known as the Uniform Resource Name (URN) identifiers for the information resources. Additional information about a deployed service can be obtained by clicking on the URN for that service. This will open a *Deployed Service Information* page, providing details on its properties. For example, clicking on the *urn:nci-gene-service* link displays the *ID*, *Provider Type*, *Provider Class*, and *Methods* properties.

The most important properties are *ID* and *Methods*. *ID* is the Uniform resource Identifier (URI) for the SOAP service; *Methods* enumerates the methods available from the service. For example, some of the methods provided by *GeneService* are *getGenes*, *getTaxons*, *getClones*, *getSequences*, and *getPathways*.

The caBIO SOAP services are implemented by the classes defined in the caBIO *webservices* package. You can get the details on these classes from the [JavaDoc](#) pages for that package. The caBIO architecture includes about 30 “service” classes, which implement communication between the caBIO domain objects and SOAP client applications via XML documents. Table 13.2-1 lists several of the most frequently used services and the java bean domain objects they provide access to. The sample Perl application that follows demonstrates how information about a specific *Gene* object can be selectively extracted using SOAP::Lite.

<u>SOAP service name</u>	<u>caBIO bean class</u>
<i>GeneService</i>	<i>gov.nih.nci.caBIO.bean.Gene</i>
<i>LibraryService</i>	<i>gov.nih.nci.caBIO.bean.Library</i>
<i>TargetService</i>	<i>gov.nih.nci.caBIO.bean.Target</i>
<i>AgentService</i>	<i>gov.nih.nci.caBIO.bean.Agent</i>
<i>PathwayService</i>	<i>gov.nih.nci.caBIO.bean.Pathway</i>
<i>ClinicalTrialProtocolService</i>	<i>gov.nih.nci.caBIO.bean.ClinicalTrialProtocol</i>
<i>GenericObjectService</i>	<i>gov.nih.nci.caBIO.bean</i>

**Table 13.2-1** Frequently used caBIO SOAP services.

### 13.2.2 Accessing the GeneService using SOAP::Lite

SOAP::LITE is a collection of Perl modules that provide a simple interface to both the client and the server. Each SOAP::LITE method can be used for both setting and retrieving values. In the absence of any arguments, the current value is retrieved. When parameters are provided, the new value specified in the arguments will be assigned to the object referred to in the method. The example that follows retrieves information about a specific gene, using the gene’s symbol (PTEN) to select it.

Three mandatory arguments for accessing any SOAP service are:

- *server* – <the NCICB server URL>
- *port* – < the NCICB server listening port> e.g., 80
- *method* – the requested SOAP service, e.g., *GeneService*

These arguments can be specified programmatically inside the Perl script, or at runtime on the command line. For example, given a Perl script called *geneClient.pl*, we can invoke it as follows:

```
geneClient.pl cabio.nci.nih.gov 80 getGenes -symbol PTEN
```



Here, *cabio.nci.nih.gov* is the IP address of the SOAP server, 80 is the listening port, and *getGenes* is the method of the *GeneService* we wish to access. The last argument is a specific parameter to *getGenes*, specifying that we would like to retrieve a *Gene* object whose symbol matches the string PTEN. Alternatively, we might specify these parameters in the Perl script itself, as:

```
$server = "cabio.nci.nih.gov"; # set the server variable
$port = "80"; # set the port variable
$method = "getGenes"; # set the method variable
my %searchRec=(); # declare a hashtable to store the search options
$searchRec{"symbol"} = "pTEN"; # initialize the symbol field in searchRec
```

Using either approach, our Perl script will still need to include additional variables specifying the URI for the SOAP service we wish to access, and a proxy path for message routing:

```
$URI='urn:nci-gene-service'; # set the URI variable to GeneService
$PROXY_PATH='/soap/servlet/rpcrouter'; # set the PROXY_PATH to (RPC|Message) Router
```

The proxy path specifies the endpoint service address and loads the required module. This path is required for dispatching SOAP calls. SOAP::Lite provides explicit functions for these specifications: *uri()* and *proxy()*. Given the above \$variable definitions, we can apply these as:

```
$s = SOAP::Lite; # declare a SOAP::Lite variable
-> uri($URI); # set the SOAP::Lite uri()
-> proxy("http://$server:$port$PROXY_PATH"); # set the SOAP::Lite proxy()
```

### 13.2.3 Issuing a SOAP::Lite Service Request

Thus far, we have set up everything we need for the connection: the IP address for the SOAP server, the listening port for results, the URI for the SOAP service we wish to access, and a proxy path for message routing. We have also created an internal hashtable to store <tag/value> pairs for the search fields we will use; we have stored the pair <“symbol,” “PTEN”> in that table, and we have assigned the string *getGenes* to the *method* variable. All that is left now is to declare a variable to store results in, and a way of actually invoking the method. We can do this as follows:

```
$som=$s->$method(SOAP::Data->type(map => \%searchRec));
```

In general *SOAP::Data* is used to specify a value, a name, a type, a URI, or attributes for SOAP elements. In this example we have used it to specify that the argument, *searchRec*, is a *map* type, since it is essentially a two-dimensional array. Alternatively, we might have specified *value()*, *name()*, *uri()*, or *attr()* in place of *type()*.

The return object *\$som* is a SOAP::SOM object and can be used to access the returned values. If a fault element is in the message, *\$som->fault* will be defined. Additional information, including *faultdetail*, *faultcode*, and *faultstring*, is also available from the *\$som* object. If the request was successful, the response XML can be retrieved and saved by calling *\$som->result* as follows:

```

$xmlldoc = $som->result;           # get the result
open (OUT, ">pTEN.XML");          # open a file for output
print OUT $xmlldoc;              # write output to the file
print $xmlldoc;                  # write to standard output

```

### 13.2.4 The Complete *geneClient.pl* Perl Script

```

use SOAP::Lite;
use HTML::Entities;
$URI='urn:nci-gene-service';
$PROXY_PATH='/soap/servlet/rpcrouter';
my %searchRec=();
$server = "cabio.nci.nih.gov";
$port = "80";
$method = "getGenes";
$searchRec{"symbol"} = "pTEN";
$s = SOAP::Lite
    -> uri($URI)
    -> proxy("http://$server:$port$PROXY_PATH");

# make service request
$som=$s->$method(SOAP::Data->type(map => \%searchRec));

# interpret result
if ($som->fault) {
    print "FAULT ENCOUNTERED!\nfaultcode:\t" . $som->faultcode .
"\nfaultstring:\t" .
    $som->faultstring . "\n";
} else {
    $xmlldoc = $som->result;
    open (OUT, ">pTEN.xml");
    print OUT $xmlldoc;
    print $xmlldoc;
    close OUT;
}

```

### 13.2.5 The XML Output and the Additional Arguments

The resulting XML output of this script contains information relating to the selected gene whose symbol was specified as “PTEN.” Simple features like the gene’s name, title, and cross-referencing IDs into other databases are represented directly, as they are simple text strings. But by default, more complex return values that reference other caBIO objects such as *Chromosome*, *ExpressionFeature*, and *Taxon* are encoded as *XLinks* only.

There are two ways to retrieve further information about these embedded *XLinks*. The simple approach of editing your perl scripts to recursively embed the *Xlink:Href* URIs and retrieving the output may become tedious when those queries in turn return additional *XLinks*.

Alternatively, if you know in advance which *XLinks* you will need to expand, you can do so on the first pass by adding two more arguments: *fillInObjects* and *returnHeavyXML*. The *fillInObjects* option accepts comma-separated arguments, which specify which tags are to be opened up further. The corresponding *XLinks* are then “filled in” with their XML content one level deep. The *returnHeavyXML* option opens *all* of the embedded *XLinks* one level deep.

For example, suppose you want to open up the XLinks corresponding only to *ExpressionFeature* and *MapLocation*. The syntax for this would be:

```
$searchRec{"fillInObjects"} = "ExpressionFeature,MapLocation";
```

where *\$searchRec* is the hashtable variable used to store different arguments to map to the SOAP service. Alternatively, to “fill up” all of the top-level XLinks in the resulting XML, use:

```
$searchRec{"returnHeavyXML"}="true";
```

These options can also be specified at the command line, e.g.:

```
geneClient.pl cabio.nci.nih.gov 80 getGenes -symbol PTEN -fillInObjects  
ExpressionFeature,MapLocation
```

The example provided here is hard-coded with respect to the server, port number, method, and attributes that will be applied in the service request. [Appendix H](#) provides a more generic way of encoding this, where all of the parameters can be entered on the command line.

A number of additional Perl script examples are included in the caBIO distribution file, available at the [NCICB Download](#) web site. Download this demonstration file, and explore the PerlSOAP subdirectory. To get a complete list of the options available for the various services and methods, refer to APIs for the classes implementing these services, on the [NCICB Webservices Java Doc](#) pages.

### 13.3 The caBIO SOAP Services Catalog

The caBIO architecture includes 36 web service classes, which implement communication between the caBIO domain objects and SOAP client applications via XML documents. These services are implemented by classes defined in the caBIO *webservices* package, and are fully documented on the [JavaDoc](#) pages for that package. The tables below summarize the information available from the JavaDocs, and provide additional descriptive information as well as explicit specifications of the methods and parameters that can be used with these services.

We begin by defining each web service and enumerating the methods that service provides. Each web service class encapsulates methods callable by a SOAP client to perform a search on the associated domain object. The method is invoked with a list of tag/value pairs, referred to in SOAP as a hashtable, as in the above example from the geneClient Perl script. In that example, the method is *getGenes*, the tag is *symbol*, and the value is PTEN.

The specific web service being accessed in the Perl script was identified via the Uniform Resource Identifier which defined the *GeneService*. Each web service has a URI, which is analogous to a postal address for receiving messages. The caBIO web services use a simple pattern for URI addressing:

```
urn:nci-[DomainObjectName]-service.
```

For example, to call the *GeneService*, the client would use:

```
urn:nci-gene-service
```

It is important to note that methods with the same name may be owned by *many* web service classes, and that these methods will have different signatures and behaviors depending upon the web service being used to access the method. For example, many of the domain objects have associations with the *Gene* object, and each of their associated web services provides a *GetGenes* method. The allowed arguments to *GetGenes* when invoked through the *ChromosomeService* will be different, however, from those allowed when the method is accessed via the *GeneService*.

More specifically, each web service has a fixed set of parameters that can be applied to *all* of its method invocations. This derives from the underlying association of the web service with a domain object and, in turn, with that domain object's *SearchCriteria* object. Thus, since the *ChromosomeSearchCriteria* accepts different parameters than the *GeneSearchCriteria*, the methods provided by the corresponding web services—even those of the same name—also have different signatures. The URI is used by the server to identify which service owns the method the client is invoking.

Each web service is summarized below by the definition of its associated domain object, the list of parameters accepted by that service, and a list of methods provided. The parameters are defined at the service level and are therefore valid for any method in that service. All of the methods return XML-encoded representations; the return values summarized below reflect the content.

### 13.3.1 AgentService

*Definition:* Agent – a therapeutic agent (drug, intervention therapy) used in a clinical trial

*Parameters:* agentNSCNumber, clinicalTrialProtocolId, comment, evsId, isCMAPAgent, name, source, targetId, therapyId

<u>Method</u>	<u>Description</u>
getAgents	The set of Agents satisfying the search criteria
getClinicalTrialProtocols	The ClinicalTrialProtocols for these Agents
getTargets	The Targets associated with these Agents

### 13.3.2 AnomalyService

*Definition:* Anomaly – an irregularity in either the expression of a gene or its structure (i.e., a mutation).

*Parameters:* anomalyDescription, contextCode, histopathologyId, organId, targetId

<u>Method</u>	<u>Description</u>
getAnomalys	The set of Anomalies satisfying the search criteria
getHistopathologys	The Histopathologies associated with these Anomalies
getTargets	The Targets associated with these Anomalies

### 13.3.3 ChromosomeService

*Definition:* Chromosome – an object representing a specific chromosome for a specific taxon; provides access to all known genes contained in the chromosome and to the taxon.

*Parameters:* name

<u>Method</u>	<u>Description</u>
getChromosomes	The set of Chromosomes satisfying the search criteria
getGenes	The Genes associated with these Chromosomes

### 13.3.4 ClinicalTrialProtocolService

*Definition:* ClinicalTrialProtocol – the protocol associated with a clinical trial; organizes administrative information about the trial such as Organization ID, participants, phase, etc., and provides access to the administered Agents.

*Parameters:* agent, agentId, conceptId, ctepName, diseaseCategory, diseaseId, diseaseName, documentNumber, imtCode, leadOrganizationId, leadOrganizationName, nihAdminCode, pdqIdentifier, phase, piName, protocolAssociationId, title, treatmentFlag

<u>Method</u>	<u>Description</u>
getClinicalTrialProtocols	The set of ClinicalTrialProtocols satisfying the search criteria
getAgents	The Agents associated with these ClinicalTrialProtocols
getProtocolAssociations	The ProtocolAssociations for these ClinicalTrialProtocols

### 13.3.5 CloneService

*Definition:* Clone – an object used to hold information pertaining to I.M.A.G.E. clones; provides access to sequence information, associated trace files, and the clone’s library.

*Parameters:* geneId, name, sequenceId, snpId, verified

<u>Method</u>	<u>Description</u>
getClones	The set of Clones satisfying the search criteria
getSequences	The Sequences associated with these Clones

### 13.3.6 CMAPOntologyService

*Definition:* an object providing entry to the CMAP gene ontology, which categorizes genes by function; provides access to *Gene* objects corresponding to the ontological term, as well as to ancestor and descendant terms in the ontology tree. **Note:** the *CMAPOntologySearchCriteria* class inherits attributes from its parent class, *OntologySearchCriteria*.

*Parameters:* (*direct*) cMAPChildId, cMAPGeneId, cMAPName, cMAPParentId, cMAPOntologyId; (*inherited*) diseaseId, geneId, histopathologyId, name, includeBoth, includeParents, includeChildren, relationshipParentId, relationshipChildId, relationshipType

<u>Method</u>	<u>Description</u>
getCMAPOntologys	The set of CMAPOntology terms satisfying the search criteria
getChildren	The descendant terms of these CMAPOntology terms
getGenes	The Genes associated with these CMAPOntology terms
getParents	The parent terms of these CMAPOntology terms

### 13.3.7 ConsensusSequenceService

*Definition:* ConsensusSequence – a specialization of the *Sequence* class; represents the consensus of a set of contigs, which it also provides access to.

*Parameters:* consensusSequenceType, contigId, geneId, proteinId, refGeneId

<u>Method</u>	<u>Description</u>
getConsensusSequences	The set of ConsensusSequences satisfying the search criteria

### 13.3.8 ContigService

*Definition:* Contig – one of the set of overlapping sequence fragments used to assemble a consensus sequence, which it also provides access to.

*Parameters:* sequenceId, name

<u>Method</u>	<u>Description</u>
getContigs	The set of Contigs satisfying the search criteria
getSequences	The known Sequences for these Contigs

### 13.3.9 DiseaseRelationshipService

*Definition:* DiseaseRelationship – specifies a child or parent relationship between Disease objects. **Note:** the *DiseaseRelationshipSearchCriteria* class inherits attributes from its parent class *RelationshipSearchCriteria*.

*Parameters:* (*inherited*) relationshipChildId, relationshipParentId, relationshipType

<u>Method</u>	<u>Description</u>
getDiseaseRelationships	The set of DiseaseRelationships satisfying the search criteria
getDiseases	The Diseases indicated in these DiseaseRelationships

### 13.3.10 DiseaseService

*Definition:* Disease – an object that specifies a disease name and ID; *Disease* objects also provide access to: ontological relations to other diseases; clinical trial protocols treating the disease; and specific histologies associated with instances of the disease. **Note:** the *DiseaseSearchCriteria* class inherits attributes from its parent class, *OntologySearchCriteria*.

*Parameters:* (*inherited*) diseaseId, histopathologyId, geneId, includeBoth, includeChildren, includeParents, name, relationshipChildId, relationshipParentId, relationshipType

<u>Method</u>	<u>Description</u>
getDiseases	The set of Diseases satisfying the search criteria
getChildRelationships	The DiseaseRelationships to descendants of these Diseases
getHistopathologys	The Histopathologies associated with these Diseases
getParentRelationships	The DiseaseRelationships to parents of these Diseases

### 13.3.11 EstExperimentService

*Definition:* EstExperiment – an object that represents data from an expression experiment using expressed sequence tags. **Note:** the *EstExperimentSearchCriteria* class inherits attributes from its parent class, *ExpressionExperimentSearchCriteria*.

*Parameters:* (*direct*) contextId; (*inherited*) expressionFeatureId, gene, geneId, organ, proteinId, taxonId, threshold, type

<u>Method</u>	<u>Description</u>
getEstExperiments	The set of ESTExperiments satisfying the search criteria

### 13.3.12 ExpressionFeatureService

*Definition:* ExpressionFeature – an object that is associated with a gene and provides access to the list of organs where the gene is known to be expressed.

*Parameters:* geneId, expressionLevelDescId

<u>Method</u>	<u>Description</u>
getExpressionFeatures	The set of ExpressionFeatures satisfying the search criteria
getOrgans	The set of Organs associated with these ExpressionFeatures

### 13.3.13 ExpressionMeasurementArrayService

*Definition:* ExpressionMeasurementArray – an array of ExpressionMeasurements.

*Parameters:* accessionNumber, expressionMeasurementId, name

<u>Method</u>	<u>Description</u>
getExpressionMeasurementArrays	ExpressionMeasurementArrays satisfying the search criteria
getExpressionMeasurements	ExpressionMeasurements contained in these ExpressionMeasurementArrays

### 13.3.14 ExpressionMeasurementService

*Definition:* ExpressionMeasurement – an object that represents a structure capable of measuring the absolute or relative amount of an expressed compound.

*Parameters:* accessionNumber, expressionMeasurementArrayId, geneId, name, sequenceId

<u>Method</u>	<u>Description</u>
getExpressionMeasurements	The ExpressionMeasurements satisfying the search criteria
getExpressionMeasurementArrays	The associated ExpressionMeasurementArrays
getGenes	The Genes associated with these ExpressionMeasurements
getSequences	The associated Sequences

### 13.3.15 GeneAliasService

*Definition:* GeneAlias – an alternative name for a gene; provides descriptive information about the gene (as it is known by this alias), as well as access to the Gene it refers to.

*Parameters:* description, geneId, type

<u>Method</u>	<u>Description</u>
getGeneAliases	The set of GeneAliases satisfying the search criteria

### 13.3.16 GeneHomologService

*Definition:* Defined only in relation to another Gene, the *GeneHomolog* in caBIO is the functional equivalent of that gene in another taxon (i.e., its ortholog). The *GeneHomolog* object is a specialization of the parent *Gene* object; in addition to all of the methods provided by the gene interface, the homolog provides its percent of sequence similarity to the related gene of interest.

*Parameters:* geneId

<u>Method</u>	<u>Description</u>
getGeneHomologs	The set of GeneHomologs satisfying the search criteria

### 13.3.17 GeneService

*Definition:* Gene – the effective portal to most of the genomic information provided by the caBIO data services; organs, diseases, chromosomes, pathways, sequence data, and expression experiments are among the many objects accessible via a gene.

*Parameters:* allPathwayId, bcId, chromosomeId, cloneName, cMAPOntologyId, cytogenicLocation, expressedPathwayId, expressionMeasurementId, functionalPathway, genBankAccessionNumber, GeneKeyword, geneNameKeyword, goOntologyHomoSapienId, goOntologyId, goOntologyMouseId, keyword, mutatedGenePathwayId, organism, overExpressedPathwayId, pathwayId, symbol, targetId, taxonId, tissueType, underExpressedPathwayId, unigeneClusterId, uniqueIdentifier

<u>Methods:</u>	<u>Description</u>
getGenes	The set of Genes satisfying the search criteria
getGeneAliases	GeneAliases for these Genes
getGeneHomologs	GeneHomologs for these Genes
getGenomicSequences	Sequences for these Genes
getGoOntologies	GoOntology entries for these Genes
getMapLocations	MapLocations for these Genes
getPathways	Pathways for these Genes
getProteins	Proteins expressed by these Genes
getReferenceSequences	RefSeq Sequences for these Genes
getSequences	Sequences for these Genes
getSNPs	SNPs for these Genes

### 13.3.18 GoOntologyRelationshipService

*Definition:* GoOntologyRelationship – an object that specifies a child or parent relationship between *GoOntology* objects. **Note:** the *GoOntologyRelationshipSearchCriteria* class inherits attributes from its parent class, *RelationshipSearchCriteria*.

*Parameters:* (*inherited*) relationshipChildId, relationshipParentId, relationshipType

<u>Method</u>	<u>Description</u>
getGoOntologyRelationships	The GoOntologyRelationships satisfying the search criteria
getGoOntologies	The GoOntology entries in these GoOntologyRelationships



### 13.3.19 GoOntologyService

*Definition:* GoOntology – an object that provides entry to a *Gene* object’s position in the Gene Ontology Consortium’s controlled vocabularies, as recorded by LocusLink; provides access to *Gene* objects corresponding to the ontological term, as well as to ancestor and descendant terms in the ontology tree. **Note:** the *GoOntologySearchCriteria* class inherits attributes from its parent class, *OntologySearchCriteria*.

*Parameters:* (*direct*) geneId; (*inherited*) diseaseId, geneId, histopathologyId, includeBoth, includeParents, includeChildren, name, relationshipParentId, relationshipChildId, relationshipType

<u>Method</u>	<u>Description</u>
getGoOntologys	The set of GoOntology entries satisfying the search criteria
getChildRelationships	The child relationships associated with these GoOntology entries
getHomoSapienGenes	The Homo Sapien Genes associated with these GoOntology entries
getMouseGenes	The mouse Genes associated with these GoOntology entries
getParentRelationships	The parent relationships associated with these GoOntology entries

### 13.3.20 HistopathologyService

*Definition:* Histopathology – an object that represents anatomical changes in a diseased tissue sample associated with an expression experiment; also captures the relationship between organ and disease.

*Parameters:* diseaseId, expressionExperimentId, name, organId

<u>Method</u>	<u>Description</u>
getHistopathologys	The set of Histopathology items satisfying the search criteria
getAnomalys	The Anomalies associated with these Histopathologies
getDiseases	The Diseases associated with these Histopathologies
getOrgans	The Organs associated with these Histopathologies

### 13.3.21 LibraryService

*Definition:* Library – an object that provides access to CGAP library information about the tissue sample and its method of preparation, the library protocol that was used, the clones contained in the library, and the sequence information derived from the library.

*Parameters:* geneId, libraryGroup, libraryName, libraryProtocol, organism, sortOrder, tissueHistology, tissueName, tissuePreparation, tissueType

<u>Method</u>	<u>Description</u>
getLibraries	The set of Library objects satisfying the search criteria
getProtocols	The Protocols for these Libraries
getTissues	The Tissues associated with these Libraries

### 13.3.22 MapLocationService

*Definition:* MapLocation – an object that represents the physical map location of a gene.

*Parameters:* type, location, geneId

<u>Method</u>	<u>Description</u>
getMapLocations	The set of MapLocations satisfying the search criteria

### 13.3.23 OrganRelationshipService

*Definition:* OrganRelationship – an object that specifies a child or parent relationship between Organ objects. **Note:** the *GoOrganRelationshipSearchCriteria* class inherits attributes from its parent class, *RelationshipSearchCriteria*.

*Parameters:* (*inherited*) proteinId, relationshipChildId, relationshipParentId, relationshipType

<u>Method</u>	<u>Description</u>
getOrganRelationships	The set of OrganRelationships satisfying the search criteria
getOrgans	The Organs associated with these OrganRelationships

### 13.3.24 OrganService

*Definition:* Organ – a representation of an organ whose name occurs in a controlled vocabulary; provides access to any *Histopathology* objects for the organ, and, because it is “ontolog-able,” provides access to its ancestral and descendant terms in the vocabulary. **Note:** the *OrganSearchCriteria* class inherits attributes from its parent class, *OntologySearchCriteria*.

*Parameters:* (*direct*) anomaly\_id, expressionFeatureId, histopathologyId; (*inherited*) diseaseId, geneId, includeBoth, includeChildren, includeParents, name, relationshipChildId, relationshipParentId, relationshipType

<u>Method</u>	<u>Description</u>
getOrgans	The Organs satisfying the search criteria
getChildRelationships	The relationships of these organs to their descendants in the EVS Organ taxonomy
getHistopathologies	The Histopathologies associated with these organs
getParentRelationships	The relationships of these organs to their parent organs in the EVS Organ taxonomy

### 13.3.25 PathwayService

*Definition:* Pathway – an object that represents a molecular/cellular pathway compiled by [BioCarta](#). Pathways are associated with specific Taxa, and contain multiple Genes, which may be Targets for treatment.

*Parameters:* bioProcessId, context, displayValue, geneId, name, pathwayDiagram, taxonId

<u>Method</u>	<u>Description</u>
getPathways	The set of Pathway objects satisfying the search criteria
getGenes	The Genes associated with these Pathways

### 13.3.26 ProteinHomologService

*Definition:* Defined only in relation to another Protein of interest, the ProteinHomolog in caBIO is the functional equivalent of that protein in another taxon (i.e., its ortholog). The ProteinHomolog is a specialization of the parent Protein object; in addition to the methods

provided by the protein interface, the Homolog object provides the percent of sequence similarity of the homolog to the related protein of interest.

*Parameters:* proteinId

<u>Method</u>	<u>Description</u>
getProteinHomologs	The set of ProteinHomologs satisfying the search criteria

### 13.3.27 ProteinService

*Definition:* Protein – an object representation of a protein; provides access to the encoding gene via its [GenBank](#) ID, the taxon in which this instance of the protein occurs, and references to homologous proteins in other species.

*Parameters:* accessionNumber, description, geneId

<u>Method</u>	<u>Description</u>
getProteins	The set of Proteins satisfying the search criteria
getProteinHomologs	The ProteinHomologs that are orthologs of these Proteins

### 13.3.28 ProtocolAssociationService

*Definition:* ProtocolAssociation – an object that associates ClinicalTrialProtocols to Diseases.

*Parameters:* clinicalTrialProtocolId, protocolId

<u>Method</u>	<u>Description</u>
getProtocolAssociations	The set of ProtocolAssociations satisfying the search criteria
getClinicalTrialProtocols	The ClinicalTrialProtocols associated with these ProtocolAssociations

### 13.3.29 ProtocolService

*Definition:* Protocol – an object used to represent the protocol used in assembling a clone library.

*Parameters:* name

<u>Method</u>	<u>Description</u>
getProtocols	The set of Protocols satisfying the search criteria

### 13.3.30 ReadSequenceService

*Definition:* ReadSequence – an object representing the output of a TraceFile, an ASCII representation of the nucleotide sequence; a read sequence is created by running PHRED.

*Parameters:* cloneId, geneId, proteinId, readSequenceId, refGeneId, traceFileId

<u>Method</u>	<u>Description</u>
getReadSequences	The set of ReadSequences satisfying the search criteria

### 13.3.31 SageExperimentService

*Definition:* SageExperiment – a subclass of the ExpressionExperiment class, used to represent serial analysis of gene expression (SAGE) data. **Note:** the *SageExperimentSearchCriteria* class inherits attributes from its parent class *ExpressionExperimentSearchCriteria*.

*Parameters:* (*direct*) contextId; (*inherited*) expressionFeatureId, gene, geneId, organ, proteinId, taxonId, threshold, type

<u>Method</u>	<u>Description</u>
getSageExperiments	The set of SAGEExperiments satisfying the search criteria

### 13.3.32 SequenceService

*Definition:* Sequence – an object representing a gene sequence and providing access to the clones from which it was derived, the ASCII representation of the residues it contains, and the sequence ID.

*Parameters:* accessionNumber, cloneId, contigId, expressionMeasurementId, geneId, isRefSeq, refGeneId, returnDna, sequenceType

<u>Method</u>	<u>Description</u>
getSequences	The set of Sequences satisfying the search criteria

### 13.3.33 SNPService

*Definition:* SNP – an object that represents a Single Nucleotide Polymorphism; provides access to the clones and the trace files from which it was identified, the two most common substitutions at that position, the offset of the SNP in the parent sequence, and a confidence score.

*Parameters:* geneId

<u>Method</u>	<u>Description</u>
getSNPs	The set of SNPs satisfying the search criteria
getClones	The Clones of these SNPs
getTraceFiles	The TraceFiles of these SNPs

### 13.3.34 TargetService

*Definition:* Target – an object that represents a gene thought to be at the root of a disease etiology and which is targeted for therapeutic intervention in a clinical trial.

*Parameters:* agentId, anomalyDescription, anomalyId, cancerType, conceptId, geneId

<u>Method</u>	<u>Description</u>
getTargets	The set of Targets satisfying the search criteria
getAgents	The Agents associated with these Targets
getAnomalys	The Anomalies associated with these Targets
getGenes	The Genes associated with these Targets

### 13.3.35 TaxonService

*Definition:* Taxon – an object representing the various names (scientific, common, abbreviated, etc.) for a species associated with a specific Gene, Chromosome, Pathway, Protein, or Tissue.

*Parameters:* abbreviation, animalModelId, chromosomeId, id, isPreferred, name, regulatoryElementId, scientificName, strainId, xenograftId

<u>Method</u>	<u>Description</u>
getTaxons	The set of Taxa satisfying the search criteria

getGenes                    The Genes associated with these Taxa

### **13.3.36 TissueService**

*Definition:* Tissue – defined by any group of similar cells united to perform a specific function.

*Parameters:* libraryId

<u>Method</u>	<u>Description</u>
getTissues	The set of Tissues satisfying the search criteria

### **13.3.37 TraceFileService**

*Definition:* TraceFile – an object that represents the recorded trace file used to identify a SNP, based on the observed intensities for the four possible bases at each position in the sequence.

*Parameters:* cloneId, name, snpId

<u>Method</u>	<u>Description</u>
getTraceFiles	The set of TraceFiles satisfying the search criteria
getReadSequences	The ReadSequences of these TraceFiles

### **13.3.38 Map of Web Services to Methods**

This section concludes with a quick reference chart that maps caBIO web services to methods.

Table 13.3-1 Mapping web services to methods

	AgentService	AnomalyService	ChromosomeService	ClinicalTrialProtocolService	CloneService	CMAPOntologyService	ConsensusSequenceService	ContigService	DiseaseRelationshipService	DiseaseService	ESTExperimentService	ExpressionFeatureService	ExpressionMeasurementArrayService
<i>getAgents</i>	x			x									
<i>getAnomalys</i>		x											
<i>getBioCartIds</i>													
<i>getChildren</i>						x							
<i>getChildRelationships</i>										x			
<i>getChromosomes</i>			x										
<i>getClinicalTrialProtocols</i>	x			x									
<i>getClones</i>					x								
<i>getCMAPOntologys</i>						x							
<i>getConsensusSequences</i>							x						
<i>getContigs</i>								x					
<i>getDiseaseRelationships</i>									x				
<i>getDiseases</i>									x	x			
<i>getEstExperiments</i>											x		
<i>getExpressionExperiments</i>													
<i>getExpressionFeatures</i>												x	
<i>getExpressionMeasurementArrays</i>													x
<i>getExpressionMeasurements</i>													x
<i>getGeneAliases</i>													
<i>getGeneHomologs</i>													
<i>getGenes</i>			x			x							
<i>getGenomicSequences</i>													
<i>getGoOntologys</i>													
<i>getHistopathologys</i>		x								x			
<i>getGoOntologyRelationships</i>													
<i>getHomoSapienGenes</i>													
<i>getLibraries</i>													
<i>getMapLocations</i>													
<i>getMouseGenes</i>													
<i>getOrgans</i>												x	
<i>getOrganRelationships</i>													
<i>getParents</i>						x							
<i>getPathways</i>													
<i>getParentRelationships</i>										x			
<i>getProteins</i>													
<i>getProteinHomologs</i>													
<i>getProtocolAssociations</i>				x									
<i>getProtocols</i>													
<i>getReadSequences</i>													
<i>getReferenceSequences</i>													
<i>getSageExperiments</i>													
<i>getSequences</i>					x			x					
<i>getSNPs</i>													
<i>getTargets</i>	x	x											
<i>getTaxons</i>													
<i>getTissues</i>													
<i>getTraceFiles</i>													

	ExpressionMeasurementService	GeneAliasService	GeneHomologService	GeneService	GoOntologyRelationshipService	GoOntologyService	HistopathologyService		MapLocationService	OrganRelationshipService	OrganService	PathwayService	ProteinHomologService
<i>getAgents</i>													
<i>getAnomalys</i>							x						
<i>getBioCartIds</i>													
<i>getChildren</i>													
<i>getChildRelationships</i>									x				
<i>getChromosomes</i>													
<i>getClinicalTrialProtocols</i>													
<i>getClones</i>													
<i>getCMAPOntologys</i>													
<i>getConsensusSequences</i>													
<i>getContigs</i>													
<i>getDiseaseRelationships</i>													
<i>getDiseases</i>							x						
<i>getEstExperiments</i>													
<i>getExpressionExperiments</i>													
<i>getExpressionFeatures</i>													
<i>getExpressionMeasurementArrays</i>	x												
<i>getExpressionMeasurements</i>	x												
<i>getGeneAliases</i>		x		x									
<i>getGeneHomologs</i>			x	x									
<i>getGenes</i>	x			x						x	x		
<i>getGenomicSequences</i>				x									
<i>getGoOntologys</i>				x	x	x							
<i>getGoOntologyRelationships</i>					x								
<i>getHistopathologys</i>							x			x			
<i>getHomoSapienGenes</i>						x							
<i>getLibraries</i>								x					
<i>getMapLocations</i>				x				x					
<i>getMouseGenes</i>						x							
<i>getOrgans</i>							x			x	x		
<i>getOrganRelationships</i>									x				
<i>getParents</i>													
<i>getPathways</i>				x								x	
<i>getParentRelationships</i>						x				x			
<i>getProteins</i>				x									
<i>getProteinHomologs</i>													x
<i>getProtocolAssociations</i>													
<i>getProtocols</i>								x					
<i>getReadSequences</i>													
<i>getReferenceSequences</i>				x									
<i>getSageExperiments</i>													
<i>getSequences</i>	x			x									
<i>getSNPs</i>				x									
<i>getTargets</i>													
<i>getTaxons</i>													
<i>getTissues</i>								x					
<i>getTraceFiles</i>													

	ProteinService	ProtocolAssociationService	ProtocolService	ReadSequenceService	SageExperimentService	SequenceService	SNPService	TargetService	TaxonService	TissueService	TraceFileService
<i>getAgents</i>								x			
<i>getAnomalys</i>								x			
<i>getBioCartIds</i>											
<i>getChildren</i>											
<i>getChildRelationships</i>											
<i>getChromosomes</i>											
<i>getClinicalTrialProtocols</i>		x									
<i>getClones</i>							x				
<i>getCMAPOntologys</i>											
<i>getConsensusSequences</i>											
<i>getContigs</i>											
<i>getDiseaseRelationships</i>											
<i>getDiseases</i>											
<i>getEstExperiments</i>											
<i>getExpressionExperiments</i>											
<i>getExpressionFeatures</i>											
<i>getExpressionMeasurementArrays</i>											
<i>getExpressionMeasurements</i>											
<i>getGeneAliases</i>											
<i>getGeneHomologs</i>											
<i>getGenes</i>								x	x		
<i>getGenomicSequences</i>											
<i>getGoOntologys</i>											
<i>getGoOntologyRelationships</i>											
<i>getHistopathologies</i>											
<i>getHomoSapienGenes</i>											
<i>getLibraris</i>											
<i>getMapLocations</i>											
<i>getMouseGenes</i>											
<i>getOrgans</i>											
<i>getOrganRelationships</i>											
<i>getParents</i>											
<i>getPathways</i>											
<i>getParentRelationships</i>											
<i>getProteins</i>	x										
<i>getProteinHomologs</i>	x										
<i>getProtocolAssociations</i>		x									
<i>getProtocols</i>			x								
<i>getReadSequences</i>				x							x
<i>getReferenceSequences</i>											
<i>getSageExperiments</i>					x						
<i>getSequences</i>						x					
<i>getSNPs</i>							x				
<i>getTargets</i>								x			
<i>getTaxons</i>									x		
<i>getTissues</i>										x	
<i>getTraceFiles</i>							x				x



## 13.4 The EVS SOAP Services Catalog

The EVS search package contains just two web services: *DescLogicConceptService* and *MetathesaurusConceptService*. Both of the associated search criteria objects for these services descend from the *ConceptSearchCriteria* object. Thus the web service classes inherit additional “settable” attributes from the parent search criteria object. Each service is invoked using the Uniform Resource Identifier (URI) along with a list of tag/value pairs. The caCORE web services use a simple pattern for URI addressing:

urn:nci-[*DomainObjectName*]-service.

For example, to call the *DescLogicConceptService*, the client would use:

urn:nci-DescLogicConcept-service

All of the methods provided by these two services require a *searchTerm* parameter. In addition, the *DescLogicConceptService*'s search method requires a *vocabularyName* and accepts an optional *limit* parameter. The *MetathesaurusConceptService*'s search method also accepts an optional *limit* parameter.

### 13.4.1 DescLogicConceptService

*Definition:* A description logic concept represented in the NCI Thesaurus.

*Parameters:* limit (optional), searchTerm (required), vocabularyName (required)

<u>Method</u>	<u>Description</u>
search	The names of concepts that matched the search term
getAllSubConceptNames	The names of subconcepts whose superconcepts' names matched the search term.
getAllSubConceptCodes	The conceptCodes for subconcepts whose superconcepts' names matched the search term

### 13.4.2 MetathesaurusConceptService

*Definition:* A concept in the NCI Metathesaurus.

*Parameters:* limit (optional), searchTerm (required)

<u>Method</u>	<u>Description</u>
search	The names of concepts that matched the search term

## 13.5 The caDSR SOAP Services Catalog

The caDSR module also contains a *webservices* package listing the available SOAP services. Each service is invoked using the URI along with a list of tag/value pairs. The web services use a simple pattern for URI addressing:

urn:nci-[*DomainObjectName*]-service.

For example, to call the *CaseReportFormService*, the client would use:

urn:nci-CaseReportForm-service

### 13.5.1 CaseReportFormService

*Definition:* A questionnaire that is a collection of data elements used to document patient information stipulated in a protocol.

*Parameters:* displayName; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

<u>Method</u>	<u>Description</u>
GetCaseReportForms	The set of CRFs satisfying the search criteria
getClassificationSchemeItems	The ClassificationSchemeItems for these CRFs
GetContexts	The set of Contexts in which these CRFs occur
GetDesignations	The set of Designations for these CRFs
GetModules	The set of Modules contained in these CRFs
GetProtocolFormsSets	The set of ProtocolFormsSet in which each CRF occurs
GetReferenceDocuments	The set of ReferenceDocuments associated with these CRFs

### 13.5.2 ClassificationSchemeItemService

*Definition:* An item or category in a classification scheme used to classify other components; for example, a node in a taxonomy.

*Parameters:* comments, dateCreated, dateModified, description, name, type

<u>Method</u>	<u>Description</u>
getClassificationSchemeItems	The ClassificationSchemeItems (CSIs) satisfying the search criteria
GetCaseReportForms	The CRFs for these CSIs
GetClassificationSchemes	The ClassificationSchemes in which these CSIs occur
getClassSchemeClassSchemeItems	The associated ClassSchemeClassSchemeItems
GetConceptualDomains	The ConceptualDomains in which these CSIs occur
GetDataElementConcepts	The DataElementConcepts associated with these CSIs
GetDataElements	The DataElements associated with these CSIs
getEnumeratedValueDomains	The associated EumeratedValueDomains
getModules	The Modules associated with these CSIs
getNonEnumeratedValueDomains	The associated NonEumeratedValueDomains
getObjectClasses	The ObjectClasses associated with these CSIs
getPropertyS	The Properties associated with these CSIs
getProtocolFormsSets	The ProtocolFormsSets associated with these CSIs
getProtocolFormsTemplates	The ProtocolFormsTemplates associated with these CSIs
getQuestions	The Questions associated with these CSIs
getRepresentations	The Representations associated with these CSIs
getValidValues	The ValidValues associated with the CSIs

### 13.5.3 ClassificationSchemeService

*Definition:* Any set of organizing principles or dimensions along which data can be organized. A ClassificationScheme may be a simple collection of keywords or a complex ontology.

*Parameters:* labelTypeFlag, type; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

<u>Method</u>	<u>Description</u>
getClassificationSchemes	The ClassificationSchemes satisfying the search criteria
getClassificationSchemeItems	The CSIs occurring in these ClassificationSchemes
getClassSchemeClassSchemeItems	The associated ClassSchemeClassSchemeItems
getContexts	The Contexts in which these ClassificationSchemes occur
getDesignations	The Designations for these ClassificationSchemes
getReferenceDocuments	The associated ReferenceDocuments

### 13.5.4 ClassSchemeClassSchemeItemService

*Definition:* A component used to associate a set of ClassificationSchemeItems with a particular ClassificationScheme, and to store details of that association such as the display order of the items within that scheme.

*Parameters:* dateCreated, dateModified, displayOrder, label

<u>Method</u>	<u>Description</u>
getClassSchemeClassSchemeItems	The ClassSchemeClassSchemeItems (CSCSIs) satisfying the search criteria
getChildClassSchemeClassSchemeItems	The descendants of these CSCSIs
getClassificationSchemes	The associated ClassificationSchemes
getClassificationSchemeItems	The CSIs associated with these CSCSIs
getParentClassSchemeClassSchemeItems	The ancestors of these CSCSIs

### 13.5.5 ConceptualDomainService

*Definition:* The set of all possible ValidValue meanings of a DataElementConcept expressed without representation.

*Parameters:* dimensionality; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

<u>Method</u>	<u>Description</u>
getConceptualDomains	The ConceptualDomains satisfying the search criteria
getClassificationSchemeItems	The associated ClassificationSchemeItems
getContexts	The contexts in which these ConceptualDomains are used
getDataElementConcepts	The associated DataElementConcepts
getDesignations	The Designations for these ConceptualDomains
getEnumeratedValueDomains	The associated EnumeratedValueDomains
getNonEnumeratedValueDomains	The associated NonEnumeratedValueDomain
getReferenceDocuments	The associated ReferenceDocuments
getValueMeanings	The associated ValueMeanings

### 13.5.6 ContextService

*Definition:* A designation or description of the application environment or discipline in which a name is applied or from which it originates.

*Parameters:* dateCreated, dateModified, description, designationId, languageName, name, version

<u>Method</u>	<u>Description</u>
getContext	The Contexts satisfying the search criteria
getCaseReportForms	The CRFs occurring in these Contexts
getClassificationSchemes	The ClassificationSchemes occurring in these Contexts
getConceptualDomains	The associated ConceptualDomains
getDataElementConcepts	The associated DataElementConcepts
getDataElements	The associated DataElements
getDesignations	The associated Designations
getEnumeratedValueDomains	The associated EumeratedValueDomains
getModules	The associated Modules
getNonEnumeratedValueDomains	The associated NonEumeratedValueDomains
getObjectClasses	The associated ObjectClasses
getPropertyS	The associated Properties
getProtocolFormsSets	The associated ProtocolFormsSets
getProtocolFormsTemplates	The associated ProtocolFormsTemplates
getQuestions	The associated Questions
getRepresentations	The associated Representations
getValidValues	The associated ValidValues

### 13.5.7 DataElementConceptRelationshipsService

*Definition:* A description of the affiliation between two occurrences of DataElementConcepts.

*Parameters:* dateCreated, dateModified, description, name

<u>Method</u>	<u>Description</u>
getDataElementConceptRelationships	The DataElementConceptRelationships satisfying the criteria
getChildDataElementConcepts	The Child DEC's encoded by these relationships
getParentDataElementConcepts	The Parent DEC's encoded by these relationships

### 13.5.8 DataElementConceptService

*Definition:* A concept that can be represented in the form of a DataElement and described independent of any particular representation.

*Parameters:* (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

<u>Method</u>	<u>Description</u>
getDataElementConcepts	The DataElementConcepts (DEC's) satisfying the search criteria

getChildDataElementConceptRelationships	The relations to child DECs
getClassificationSchemeItems	The associated CSIs
getConceptualDomains	The associated ConceptualDomains
getContexts	The Contexts in which these DECs occur
getDataElements	The associated DataElements
getDesignations	The associated Designations
getObjectClasses	The associated ObjectClasses
getObjectClassQualifiers	The Qualifiers for these associated ObjectClasses
getParentDataElementConceptRelationships	The relations to parent DECs
getPropertyQualifiers	The Qualifiers for the any associated Properties
getPropertyys	The associated Properties
getReferenceDocuments	The associated ReferenceDocuments

### 13.5.9 DataElementService

*Definition:* A unit of data for which the definition, identification, representation, and permissible values are specified by means of a set of attributes.

*Parameters:* (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

<u>Method</u>	<u>Description</u>
getDataElements	The DataElements satisfying the search criteria
getClassificationSchemeItems	The associated CSIs
getContexts	The Contexts in which these elements occur
getDataElementConcepts	The associated DataElementConcepts
getDesignations	The associated Designations for the data elements
getEnumeratedValueDomains	The associated EumeratedValueDomains
getNonEnumeratedValueDomains	The associated NonEumeratedValueDomains
getQuestions	The Questions associated with elements on CRFs
getReferenceDocuments	The associated ReferenceDocuments

### 13.5.10 DesignationService

*Definition:* A name by which an Administered Component is known in a specific Context. Also, a placeholder to track the usage of Administered Components by different Contexts.

*Parameters:* dateCreated, dateModified, languageName, name, type

<u>Method</u>	<u>Description</u>
getDesignations	The Designations satisfying the search criteria
getCaseReportForms	The CRFs that use these designations
getClassificationSchemes	The ClassificationSchemes using these designations
getConceptualDomains	The associated ConceptualDomains
getContexts	The Contexts in which these designations are used
getDataElementConcepts	The associated DataElementConcepts
getDataElements	The associated DataElements

getEnumeratedValueDomains	The associated EumeratedValueDomains
getModules	The associated Modules
getNonEnumeratedValueDomains	The associated NonEumeratedValueDomains
getObjectClasses	The associated ObjectClasses
getPropertyS	The associated Properties
getProtocolFormsSets	The ProtocolFormsSets that use these designations.
getProtocolFormsTemplates	The associated ProtocolFormsTemplates
getQuestions	The Questions included in the templates
getRepresentations	The associated Representations
getValidValues	The associated ValidValues

### 13.5.11 EnumeratedValueDomainService

*Definition:* A ValueDomain expressed as a list of all PermissibleValues.

*Parameters:* (*inherited*) characterSetName, dataTypeName, decimalPlace, formatName, highValueNumber, lowValueNumber, maximumLengthNumber, minimumLengthNumber, uomNamebeginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

<u>Method</u>	<u>Description</u>
getEnumeratedValueDomains	The EumeratedValueDomains satisfying the criteria
getClassificationSchemeItems	The associated ClassificationSchemeItems
getConceptualDomains	The associated ConceptualDomains
getContexts	The associated Contexts
getDataElements	The associated DataElements
getDesignations	The associated Designations
getQualifiers	The Qualifiers that modify the values
getReferenceDocuments	The associated Modules
getRepresentations	The associated Representations
getValueDomainPermissibleValues	The associated PermissibleValues

### 13.5.12 ModuleService

*Definition:* A collection of DataElements logically grouped on a CaseReportForm.

*Parameters:* displayOrder; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

<u>Method</u>	<u>Description</u>
getModules	The Modules satisfying the search criteria
getCaseReportForms	The CRFs that include these modules
getClassificationSchemeItems	The associated CSIs
getContexts	The Contexts in which these modules are used
getDesignations	The associated Designations
getProtocolFormsTemplates	The ProtocolFormsTemplates that include these modules
getQuestions	The Questions included in these modules

getReferenceDocuments                      The associated ReferenceDocuments

### 13.5.13 NonenumeratedValueDomainService

*Definition:* A ValueDomain expressed by a generative formula or range of allowed values.

*Parameters:* (*inherited*) characterSetName, dataTypeName, decimalPlace, formatName, highValueNumber, lowValueNumber, maximumLengthNumber, minimumLengthNumber, uomNamebeginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

<u>Method</u>	<u>Description</u>
getNonEnumeratedValueDomains	The NonEnumeratedValueDomains satisfying the criteria
getClassificationSchemeItems	The associated ClassificationSchemeItems
getConceptualDomains	The associated ConceptualDomains
getContexts	The associated Contexts
getDataElements	The associated DataElements
getDesignations	The associated Designations
getQualifiers	The Qualifiers that modify the possible values
getReferenceDocuments	The associated ReferenceDocuments
getRepresentations	The associated Representations

### 13.5.14 ObjectClassService

*Definition:* A set of ideas, abstractions, or things in the real world that can be identified with explicit boundaries and meaning and whose properties and behavior follow the same rules.

*Parameters:* definitionSource; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

<u>Method</u>	<u>Description</u>
getObjectClasses	The ObjectClasses satisfying the search criteria
getClassificationSchemeItems	The associated ClassificationSchemeItems
getContexts	The associated Contexts
getDataElementConcepts	The associated DECs
getDesignations	The associated Designations
getReferenceDocuments	The associated ReferenceDocuments

### 13.5.15 PermissibleValueService

*Definition:* The exact names, codes, and text that can be stored in a data field.

*Parameters:* beginDate, dateCreated, dateModified, endDate, highValueNumber, lowValueNumber, value

<u>Method</u>	<u>Description</u>
getPermissibleValues	The PermissibleValues satisfying the search criteria
getValueDomainPermissibleValues	The associated ValueDomainPermissibleValues

getValueMeanings

The associated ValueMeanings

### 13.5.16 PropertyService

*Definition:* A characteristic common to all members of an ObjectClass.

*Parameters:* definitionSource; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

<u>Method</u>	<u>Description</u>
getPropertyS	The Properties satisfying the search criteria
getClassificationSchemeItems	The associated ClassificationSchemeItems
getContexts	The associated Contexts
getDataElementConcepts	The associated DECs
getDesignations	The associated Designations
getReferenceDocuments	The associated ReferenceDocuments

### 13.5.17 ProtocolFormsSetService

*Definition:* A specific clinical trial protocol document and its collection of associated CRFs.

*Parameters:* approvedBy, approvedDate, changeNumber, changeType, leadOrganizationName, phase, protocolId, reviewedBy, reviewedDate, type; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

<u>Method</u>	<u>Description</u>
getProtocolFormsSets	The ProtocolFormsSets satisfying the search criteria
getCaseReportForms	The CRFs included in the ProtocolFormsSets
getClassificationSchemeItems	The associated ClassificationSchemeItems
getContexts	The associated Contexts
getDesignations	The associated Designations
getReferenceDocuments	The associated ReferenceDocuments

### 13.5.18 ProtocolFormsTemplateService

*Definition:* A collection of components (modules, questions) to be included in a CRF.

*Parameters:* (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

<u>Method</u>	<u>Description</u>
getProtocolFormsTemplates	The ProtocolFormsTemplates satisfying the search criteria
getClassificationSchemeItems	The associated ClassificationSchemeItems
getContexts	The associated Contexts
getDesignations	The associated Designations



getModules	The Modules included in these ProtocolFormsTemplates
getReferenceDocuments	The associated ReferenceDocuments

### 13.5.19 QualifierService

*Definition:* A term that helps define and render a concept unique.

*Parameters:* comments, dateCreated, dateModified, description, name

<u>Method</u>	<u>Description</u>
getQualifiers	The Qualifiers satisfying the search criteria
getEnumeratedValueDomains	The associated EnumeratedValueDomains
getNonEnumeratedValueDomains	The associated NonEnumeratedValueDomains
getDECOBJECTCLASSDataElementConcepts	The DECOBJECTCLASSDataElementConcepts associated with these Qualifiers
getDECPropertyDataElementConcepts	The DECPropertyDataElementConcepts associated with these Qualifiers

### 13.5.20 QuestionService

*Definition:* The actual text of the DataElement as specified on a CaseReportForm of a Protocol.

*Parameters:* displayOrder; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

<u>Method</u>	<u>Description</u>
getQuestions	The Questions satisfying the search criteria
getClassificationSchemeItems	The associated ClassificationSchemeItems
getContexts	The associated Contexts
getDataElements	The associated DataElements on CRFs
getDesignations	The associated Designations
getModules	The Modules that include these Questions
getReferenceDocuments	The associated ReferenceDocuments
getValidValues	The ValidValues for these Questions

### 13.5.21 ReferenceDocumentService

*Definition:* A place to document additional information about AdministeredComponents.

*Parameters:* dateCreated, dateModified, displayOrder, docText, languageName, name, organizationId, rdTitleName, type, url

<u>Method</u>	<u>Description</u>
getReferenceDocuments	The ReferenceDocuments satisfying the search criteria
getCaseReportForms	The CRFs that refer to these ReferenceDocuments
getClassificationSchemes	The associated ClassificationSchemes
getConceptualDomains	The associated ConceptualDomains
getDataElementConcepts	The associated DataElementConcepts
getDataElements	The associated DataElements

getEnumeratedValueDomains	The associated EumeratedValueDomains
getModules	The associated Modules
getNonEnumeratedValueDomains	The associated NonEumeratedValueDomains
getObjectClasses	The associated ObjectClasses
getPropertyS	The associated PropertyS
getProtocolFormsSets	The associated ProtocolFormsSets
getProtocolFormsTemplates	The associated ProtocolFormsTemplates
getQuestions	The associated Questions
getRepresentations	The associated Representations
getValidValues	The associated ValidValues

### 13.5.22 RepresentationService

*Definition:* Mechanism by which the functional and/or presentational category of an item maybe conveyed to a user.

*Parameters:* definitionSource; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

<u>Method</u>	<u>Description</u>
getRepresentations	The Representations satisfying the search criteria
getClassificationSchemeItems	The associated ClassificationSchemeItems
getContexts	The associated Contexts
getDesignations	The associated Designations
getEnumeratedValueDomains	The associated EnumeratedValueDomains
getNonEnumeratedValueDomains	The associated NonEnumeratedValueDomains
getReferenceDocuments	The associated ReferenceDocuments

### 13.5.23 ValidValueService

*Definition:* The allowable values for a given DataElement (Question) on a CaseReportForm

*Parameters:* displayOrder; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

<u>Method</u>	<u>Description</u>
getValidValues	The ValidValues satisfying the search criteria
getClassificationSchemeItems	The associated ClassificationSchemeItems
getContexts	The associated Contexts
getDesignations	The associated Designations
getQuestions	The Questions satisfied by these ValidValues
getReferenceDocuments	The associated ReferenceDocuments
getValueDomainPermissibleValues	The associated ValueDomainPermissibleValues

### 13.5.24 ValueDomainPermissibleValueService

*Definition:* An object that stores the many to many relationships between ValueDomains and PermissibleValues.

*Parameters:* dateCreated, dateModified

<u>Method</u>	<u>Description</u>
getValueDomainPermissibleValues	The ValueDomainPermissibleValues satisfying the criteria
getEnumeratedValueDomains	The associated EnumeratedValueDomains
getPermissibleValues	The associated PermissibleValues
getValidValues	The associated ValidValues

### 13.5.25 ValueMeaningService

*Definition:* The significance associated with an allowable/permissible value.

*Parameters:* beginDate, comments, dateCreated, dateModified, description, endDate, shortMeaning

<u>Method</u>	<u>Description</u>
getValidValues	The ValueMeanings satisfying the search criteria
getConceptualDomains	The associated ConceptualDomains
getPermissibleValues	The associated PermissibleValues

## 13.6 The caMOD Soap Services Catalog

### 13.6.1 AnimalModelService

*Definition:* A strain of genetically modified animals (rat or mouse) used to study the molecular etiology of various types of cancer.

*Parameters:* agentId, availabilityId, diseaseId, experimentDescription, modelDescriptor, partyRoleId, phenotypeId, taxonId

<u>Method</u>	<u>Description</u>
getAnimalModels	The set of AnimalModels satisfying the search criteria
getAvailability	The Availability information for these AnimalModels
getCarcinogenicInterventions	The CarcinogenicInterventions applied to these AnimalModels
getCellLines	The CellLines associated with these AnimalModels
getGenomicSegments	The GenomicSegments extracted from the AnimalModels
getHistopathologies	The Histopathologies taken from these AnimalModels
getImages	The Images associated with these AnimalModels
getInducedMutations	The InducedMutations on these AnimalModels
getMicroArrayData	The MicroarrayData associated with the AnimalModels
getPhenotypes	The Phenotypes of these AnimalModels
getPrincipalInvestigator	The PrincipalInvestigators for these AnimalModels
getPublications	The Publications associated with these AnimalModels
getSpecies	The Species of these AnimalModels

getSubmitter	The Submitters of the data for these AnimalModels
getTargetedModifications	The TargetedModifications in these AnimalModels
getTherapies	The Therapies associated with these AnimalModels
getTransgenes	The Transgenes used in these AnimalModels
getXenografts	The Xenografts applied in these AnimalModels

### 13.6.2 AvailabilityService

*Definition:* The availability status of a developed strain from the MMHCC, from the Jackson Laboratory, or directly from the principal investigator.

*Parameters:* animalModelId, enteredDate, modifiedDate, releaseDate, visibleTo

<u>Method</u>	<u>Description</u>
getAvailability	The Availability objects satisfying the search criteria

### 13.6.3 CarcinogenicInterventionService

*Definition:* A treatment (chemical or drug administration, radiation, genetic modification, knockout, etc) applied to an animal model to induce a disease state.

*Parameters:* animalModelId, environmentalFactorId, geneDeliveryId, treatmentScheduleId

<u>Method</u>	<u>Description</u>
getCarcinogenicInterventions	The CarcinogenicInterventions satisfying the search criteria
getEnvironmentalFactors	The EnvironmentalFactors complicating these interventions
getGeneDelivery	The methods of GeneDelivery used in these interventions
getTreatmentSchedule	The TreatmentSchedules used in applying these interventions

### 13.6.4 CellLineService

*Definition:* A CellLine associated with a particular strain of animal model.

*Parameters:* animalModelId, comments, experiment, name, organId

<u>Method</u>	<u>Description</u>
getCellLines	The CellLines satisfying the search criteria
getOrgan	The Organs from which these CellLines were extracted from
getPublications	The Publications describing these CellLines

### 13.6.5 ConditionalityService

*Definition:* Conditionality is used to indicate whether or not the administration of a transgene or targeted modification depends on time- or tissue-specific conditions.

*Parameters:* conditionedBy, Desc

<u>Method</u>	<u>Description</u>
getConditionalitys	The Conditionality objects satisfying the search criteria

### 13.6.6 ContactInfoService

*Definition:* Contact information for the principal investigator of the selected model.

*Parameters:* city, email, fax, institute, labName, partyId, phoneNumber, state, street, zip

<u>Method</u>	<u>Description</u>
getContactInfo	The contact information satisfying the search criteria

### **13.6.7 EngineeredGeneService**

*Definition:* A gene sequence genetically modified to induce a desired state in an animal model.

*Parameters:* caBioId, conditionalityId, dbCrossRefs, genomicSegmentId, genotypeSummaryId, imageId, inducedMutationId, locusLinkSummary, name, targetedModificationId, title

<u>Method</u>	<u>Description</u>
getEngineeredGenes	The EngineeredGenes which satisfy the search criteria
getConditionality	The Conditionality information for these genes
getExpressionFeatures	The ExpressionFeatures associated with these genes
getGenes	The Locus link Genes associated with these EngineeredGenes
getGenotypeSummary	The GenotypeSummaries for these genes
getImage	The Images associated with these genes

### **13.6.8 EnvironmentalFactorService**

*Definition:* A contributing factor to the disease state of an animal model.

*Parameters:* carcinogenicId, name, type

<u>Method</u>	<u>Description</u>
getEnvironmentalFactors	The EnvironmentalFactors satisfying the search criteria

### **13.6.9 GeneDeliveryService**

*Definition:* The method of introducing a modified gene to the recipient animal.

*Parameters:* engineeredGeneId, geneId, organId, viralVector

<u>Method</u>	<u>Description</u>
getGeneDeliverys	The methods of GeneDelivery satisfying the search criteria
getEngineeredGene	The Genes engineered by these methods of delivery

### **13.6.10 GeneticAlterationService**

*Definition:* An intentionally induced change in a gene sequence or in the number of gene copies.

*Parameters:* histopathologyId, methodOfObservation, observation,

<u>Method</u>	<u>Description</u>
getGeneticAlterations	The GeneticAlterations satisfying the search criteria

### **13.6.11 GenomicSegmentService**

*Definition:* A region or set of regions of a genome including chromosome, gene, breakpoint etc.

*Parameters:* animalModelId, cloneDesignator, integrationTypeId, LocationOfIntegration, SegmentSize, segmentTypeId

<u>Method</u>	<u>Description</u>
getGenomicSegments	The GenomicSegments satisfying the search criteria
getEngineeredGenes	The EngineeredGenes associated with these GenomicSegments
getSegmentType	The SegmentTypes of these GenomicSegments

### 13.6.12 GenotypeSummaryService

*Definition:* Genotype Summary information for a particular model.

*Parameters:* genotype, nomenclatureID, summary

<u>Method</u>	<u>Description</u>
getGenotypeSummarys	The GenotypeSummarys satisfying the search criteria
getNomenclature	The Nomenclature used in these summaries

### 13.6.13 ImageService

*Definition:* An image of the diseased tissues or organs from an animal model.

*Parameters:* animalModelId, description, image, staining, title

<u>Method</u>	<u>Description</u>
getImages	The Images satisfying the search criteria
getAvailaility	The Availability of these images

### 13.6.14 InducedMutationService

*Definition:* A mutation in a gene caused by exposure to chemicals, radiation or other mutagens.

*Parameters:* animalModelId, environmentalFactorId

<u>Method</u>	<u>Description</u>
GetInducedMutations	The InducedMutations satisfying the search criteria
getCarcinogenicInterventions	The CarcinogenicInterventions applied to induce the mutations
GetEngineeredGenes	The Genes engineered by the InducedMutations

### 13.6.15 MicroArrayDataService

*Definition:* Expression data from microarray experiments with samples from animal models.

*Parameters:* animalModelId, experimentId, experimentName

<u>Method</u>	<u>Description</u>
getMicroarrayDatas	The MicroarrayData sets satisfying the search criteria
getAvailaility	The Availability of these microarray data sets

### 13.6.16 ModificationTypeService

*Definition:* The type of gene modification in the target gene, such as the *Null* modification, change in amino acid, deletion, insertion, misense, nonsense, point mutation, etc.

*Parameters:* modificationTypeName, targetModificationId,

<u>Method</u>	<u>Description</u>
getModificationTypes	The ModificationTypes satisfying the search criteria

### **13.6.17 NomenclatureService**

*Definition:* Controlled vocabulary terms used for the MMHCC.

*Parameters:* name

<u>Method</u>	<u>Description</u>
getNomenclatures	The Nomenclatures satisfying the search criteria

### **13.6.18 PersonService**

*Definition:* An individual involved in the deposition, review, and/or approval of MMHCC data.

*Parameters:* firstName, lastName

<u>Method</u>	<u>Description</u>
getPersons	The Persons satisfying the search criteria
getContactInfo	The contact information for these persons
getRoles	The Roles served by these persons

### **13.6.19 PhenotypeService**

*Definition:* The physical appearance or otherwise observable characteristics of a model animal.

*Parameters:* animalModelId, breedingNotes, desc, sexDistributionId

<u>Method</u>	<u>Description</u>
getPhenotypes	The Phenotypes satisfying the search criteria
getSexDistribution	The observed SexDistributions for these Phenotypes

### **13.6.20 PromoterService**

*Definition:* A region of DNA sequence upstream of the coding region to which RNA polymerase will bind and initiate replication.

*Parameters:* name, regulatoryElementTypeId, speciesId, transGeneId

<u>Method</u>	<u>Description</u>
getPromoters	The Promoters satisfying the search criteria
getGenes	The Genes regulated by these promoters

### **13.6.21 PublicationService**

*Definition:* A paper or article associated with MMHCC data published in a scientific journal.

*Parameters:* animalmodelId, authors, celllineId, endPage, journal, pmId, publicationStatusId, startPage, status, therapyId, title, volume, year

<u>Method</u>	<u>Description</u>
getPublications	The Publications satisfying the search criteria
getPublicationStatus	The statuses of these publications

### 13.6.22 RegulatoryElementService

*Definition:* A region of DNA sequence controlling the transcription/expression of a gene.

*Parameters:* name, regulatoryElementTypeId, speciesId, transGeneId

<u>Method</u>	<u>Description</u>
getRegulatoryElements	The RegulatoryElements satisfying the search criteria
getRegulatoryElementTypes	The regulatory types of these elements

### 13.6.23 RegulatoryElementTypeService

*Definition:* The type of regulation imposed by the element, e.g., suppressor, promoter, etc.

*Parameters:* regulatoryElementTypeName

<u>Method</u>	<u>Description</u>
getRegulatoryElementTypes	The RegulatoryElementTypes satisfying the search criteria

### 13.6.24 RoleService

*Definition:* Role that a person or organization plays; e.g., submitter, reviewer, etc.

*Parameters:* roleName

<u>Method</u>	<u>Description</u>
getRoles	Roles satisfying the search criteria

### 13.6.25 SegmentTypeService

*Definition:* Genetic segment type such as chromosome, contig, CpG islands, repetitive DNA (e.g. Alu, LINE, SINE etc.), gene, breakpoint, etc.

*Parameters:* segmentTypeName

<u>Method</u>	<u>Description</u>
getSegmentTypes	The SegmentTypes satisfying the search criteria

### 13.6.26 SexDistributionService

*Definition:* The observable distribution of phenotypes between sexes.

*Parameters:* SexDistributionTypeName

<u>Method</u>	<u>Description</u>
getSexDistributions	The SexDistributions satisfying the search criteria

### 13.6.27 TargetedModificationService

*Definition:* Modification of a specific gene (versus one randomly selected) by a technology called gene targeting through homologous recombination.

*Parameters:* animalModelId, blastocystName, engineeredGeneId, escellLineName, geneId, modificationTypeId,

<u>Method</u>	<u>Description</u>
getTargetedModifications	The TargetedModifications satisfying the search criteria



getEngineeredGenes	The Genes engineered by these TargetedModifications
getModificationType	The types of identified modifications

### 13.6.28 TherapyService

*Definition:* A defined treatment protocol for testing the efficacy of the treatment on an engineered animal model.

*Parameters:* agentId, animalModelId, comments, experiment, treatmentScheduleId

<u>Method</u>	<u>Description</u>
getTherapys	The Therapies satisfying the search criteria
getAgent	The Agents used to deliver these Therapies
getPublications	The Publications describing these Therapies
getTreatmentSchedule	The TreatmentSchedules used in these Therapies

### 13.6.29 TransgeneService

*Definition:* A gene that has been integrated into the germ line of a transgenic animal by gene targeting technology.

*Parameters:* animalModelId, integrationTypeId, locationOfIntegration, speciesId

<u>Method</u>	<u>Description</u>
getTransgenes	The Transgenes satisfying the search criteria
getSpecies	The Species from which these transgenes were taken

### 13.6.30 TreatmentScheduleService

*Definition:* The dosage and regimen for treating cancer in an animal model.

*Parameters:* carcinogenicInterventionId, dosage, regimen, therapyId

<u>Method</u>	<u>Description</u>
getTreatmentSchedules	The TreatmentSchedules satisfying the search criteria

### 13.6.31 XenograftService

*Definition:* A surgical graft of tissue from one species onto or into individuals of unlike species, genus, or family.

*Parameters:* administrativeSite, animalModelId, geneticManipulation, hostSpeciesId, modificationDescription, name, organId, originSpeciesId, parentCellLineName, type

<u>Method</u>	<u>Description</u>
getXenografts	The Xenografts satisfying the search criteria
getOrgan	The Organ from which these Xenografts were taken
getSpecies	The Species from which these Xenografts were taken

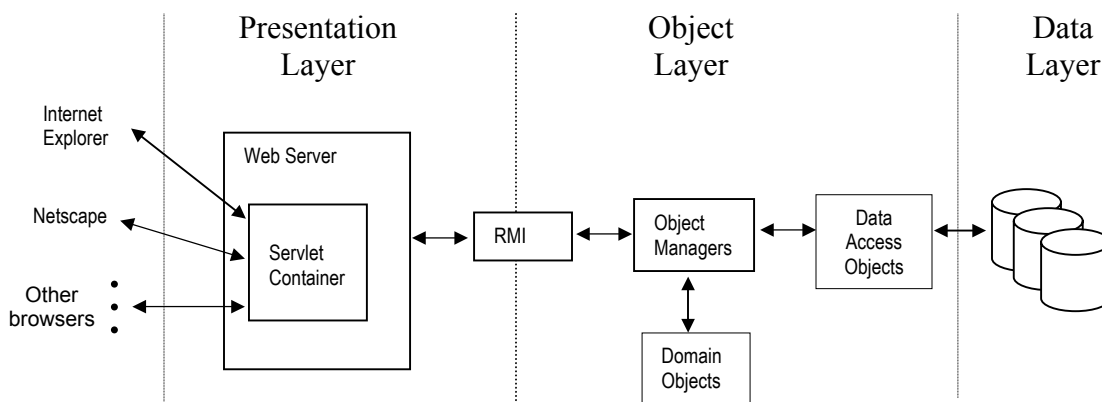
## **14.0 THE HTTP INTERFACE**

## 14.1 Overview

The [Hypertext Transfer Protocol](#) is a generic, stateless application protocol for distributed, collaborative information systems. The HTTP uses the concept of reference provided by the Uniform Resource Identifier as a location (URL) or name (URN) for indicating the resource on which a method is to be applied. The HTTP is said to be *connectionless* because once the server has responded to the single request, the connection is dropped. Because an HTTP server treats each request as unprecedented, it is also called a *stateless* protocol.

The *n*-tier model of the caBIO architecture in Figure 14.1-1 emphasizes those components in the Presentation Layer that implement the HTTP interface. The HTTP interface utilizes web browsers such as Netscape™ and Internet Explorer™. HTTP requests, which are issued as URLs on the client browsers, are processed by Java servlets on the caBIO web server, and forwarded as messages via RMI to the Object Layer.

Inside the Object Layer, the domain objects and their associated infrastructure classes (*SearchCriteria* and *SearchResult* objects) are used to register the requests and hold the data as they are fetched from the Data Layer via the data access objects. These results are then XML-encoded by the domain objects' *toXML()* methods and returned to the servlets in the Presentation Layer via RMI.



**Figure 14.1-1 The caBIO HTTP interface**

Nonprogrammers can transform the XML-encoded HTTP response using [XSL/XSLT](#). XSL (extensible style sheet language) is a language for expressing style sheets; XSL Transformations (XSLT) is a language for transforming XML documents using XSL. An example of how to use the XSL to transform an XML document is provided at the end of this section.

The caBIO objects use two devices to limit the amount of information that will be returned on a single HTTP request. The first of these is a *throttling* mechanism that limits the *number* of items returned. For example, a request to retrieve all known genes on the human genome could potentially retrieve over 30,000 gene records. To protect naïve users as well as the system from the onslaught of data that would ensue, a (re-settable) default maximum of 1,000 records per data request is enforced.

The second device limits the extent of data that will be contained in a single record, and serves as a safeguard against infinite recursion. Many of the data objects contain or otherwise entail “embedded” objects. For example, a *Gene* references the set of *Sequences* it encodes, and each of those sequences in turn references the *Gene*. Clearly, allowing each of these objects to return a full encoding of their nested references would be disastrous.

Generally, it is unlikely that the user will want access to *all* of the detailed information associated with each retrieved object, but would rather selectively specify where to drill down. The XML linking language (XLink) provides just the type of mechanism that is needed to address this concern.

The caBIO web server returns only those features that can be expressed as simple data types (i.e., strings, numbers) in the top-level encoding. Other features, such as embedded objects and arrays or other data structures, are returned as XLinks, which specify the URL to use in a subsequent request in order to retrieve that information. As described below, however, it is also possible to selectively expand these XLinks in the initial HTTP request using additional arguments.

## 14.2 Using the HTTP Interface

Requests sent by the client via HTTP are processed by a servlet called *getXML* residing on the caBIO server. *getXML* anticipates a list of parameters, which specify the type of object to retrieve along with additional search criteria to narrow the search. Currently there are two ways of invoking the *getXML* servlet.

### 14.2.1 The *operation=* syntax

The first way was introduced in caCORE 1.0, and uses the “*operation=*” syntax. For example, the following HTTP request retrieves all genes having “PTEN” as their symbol:

<http://cabio.nci.nih.gov/servlet/GetXML?operation=Gene&Symbol=PTEN>

The most important parameter here is *operation*, which specifies the class name of the type of domain object being requested. All other parameters are defined according to this first term. In particular, only the *Gene* objects have a *Symbol* field. Thus, if the parameters had been:

```
operation=Chromosome&Symbol=PTEN
```

an error would be produced stating that the search method for “symbol” does not exist. Referring back to Figure 14.1-1, recall that each HTTP request is effectively forwarded to the appropriate objects in the Object Layer. Accordingly, any additional criteria the user may wish to specify to narrow the search must be understandable by the receiving objects in the Object Layer. For convenience, the last section of this chapter summarizes the operations that can be invoked and the parameters these operations accept.

Although these definitions are relatively stable, the applications at NCICB are evolving over time, so it is also useful to understand how to obtain this information directly from the dynamically updated JavaDoc pages:

- To see the list of class names that can serve as values for the *operation* parameter, refer to the [JavaDoc](#) pages for the *bean* packages. Example class names are: *Gene*, *Library*, *Chromosome*, *Sequence*, etc. While the arguments following the operation parameter are

not case sensitive, this one *is*, so be sure to use the same case as appears in the class name.

- To get a list of the additional parameters that can follow a given *operation* request, visit the associated search criteria object's *set* methods in the *bean* or *search* packages. Each domain object (*Xxx*) in the *bean* package has an associated search criteria object, named *XxxSearchCriteria*.<sup>9</sup>

For example, the *Chromosome* domain object has an associated *ChromosomeSearchCriteria* object. This search criteria object is designed for use as the single argument to the domain object's *search()* method. The idea is to set a few attributes in the search criteria so as to limit the search, and to subsequently invoke the domain object's search method on that search criteria object.

Thus, we must consult the search criteria's *set* methods to determine what attributes are settable. The *ChromosomeSearchCriteria* has only two such named methods: *SetName()* and *SetId()*. Accordingly, if we are using *Chromosome* as the value for *operation*, then the only additional legitimate parameters are *name=* and *id=*. In other words, the additional allowed parameters for the HTTP request are defined by removing the "set" prefix from the search criteria's methods.

In summary, the "operation=" expression must always be followed by the name of a domain object specifying the type of result that will be returned. Any additional parameters which follow this expression must occur as settable attributes of the domain object's associated search criteria object. The *query* syntax, which is described next, allows for more complex queries to be formulated. The parameters which follow the "query=" expression are limited only by the terms which immediately precede them.

### 14.2.2 The *query=* syntax

The alternative syntax substitutes "query=" for "operation=" in the URL, and explicitly tags the parameters with the string "crit\_". The example below contrasts the syntax for these two forms:

Old way: <http://cabio.nci.nih.gov/servlet/GetXML?operation=Target&AgentId=10412>

New way: [http://cabio.nci.nih.gov/servlet/GetXML?query=Target&crit\\_agents\\_id=10412](http://cabio.nci.nih.gov/servlet/GetXML?query=Target&crit_agents_id=10412)

The first of these two expressions uses the operation syntax. Effectively, this expression states that a collection of *Target* objects should be searched for using a *TargetSearchCriteria* object whose *agentId* has been set to 10412. In other words we are looking for those targets whose associated agents include at least one whose Id is equal to 10412.

The second expression should be interpreted as follows. Again, we have specified that the type of objects to be returned should be *Target* objects. But in this case, the parameters that follow are no longer limited to the selection attributes defined for *TargetSearchCriteria*. Instead, each *crit\_* tag signifies that some other object, which is in relation to the primary object, should be used to define additional selection criteria. More specifically, the second expression states

---

<sup>9</sup> As explained in [Chapter 9](#), the caBIO package structure includes the search criteria in the *bean* package, along with the domain objects. The other modules define the search criteria objects in the *search* packages.

that an *AgentSearchCriteria* (with its *id* set to 10412) should be embedded (“put”) inside the *TargetSearchCriteria*. This is analogous to the *putSearchCriteria()* methods which were described in [Section 8.1](#).

As illustrated here, immediately following the “crit\_” tag is the name of the type of object you wish to use as a filter. The name of the object is in lowercase and in plural form, to indicate that the objects are in relation to each other.

As demonstrated by this example, the new syntax is equivalent to the old syntax when the associated search criteria object defines the “foreign attribute” you wish to search on. Because the *TargetSearchCriteria* object has a *setAgentId()* method, we can achieve the same results using either approach. More complex queries however, which can be expressed using the new *query* syntax, have no equivalents in the deprecated syntax.

For example, suppose that you would like to retrieve information on the human chromosome containing the BRCA1 gene. Unfortunately, the *ChromosomeSearchCriteria* object provides no direct method for filtering its result set by gene name. Thus, using the “operation=” syntax, the only solution is to first fetch the gene, and subsequently, the chromosome on which it occurs.

But in this example we would also like to filter by taxon—that is, we want to retain only human chromosomes. Although the *GeneSearchCriteria* does allow you to filter by taxon, the only available method is *setTaxonId()*. So unless we already know that the taxon Id for homo sapiens is 5, the “operation=” syntax will have to begin by retrieving taxons. In summary, the “operation=” syntax will require three invocations to obtain the desired results:

- (1) <http://cabio.nci.nih.gov/servlet/GetXML?operation=Taxon&scientificName=homo+sapiens>
- (2) <http://cabio.nci.nih.gov/servlet/GetXML?operation=Gene&symbol=BRCA1&taxonID=5>
- (3) <http://cabio.nci.nih.gov/servlet/GetXML?operation=Chromosome&id=19>

From our first query we can extract the appropriate taxon id (5); from our second query we extract the chromosome Id (19); and finally, the third query retrieves the desired information. Using the new syntax however, this can be accomplished in a single statement, as:

[http://cabio.nci.nih.gov/servlet/GetXML?query=Chromosome&crit\\_genes\\_name=BRCA1&crit\\_taxon\\_scientificName=homo+sapiens](http://cabio.nci.nih.gov/servlet/GetXML?query=Chromosome&crit_genes_name=BRCA1&crit_taxon_scientificName=homo+sapiens)

Using the query syntax, it is also possible to mix local attributes with the object relations described above. And in actuality, these relations *are* local attributes of the primary object. For example, the following two expressions are effectively equivalent to (2) above:

[http://cabio.nci.nih.gov/servlet/GetXML?query=Gene&crit\\_name=BRCA1&crit\\_taxonId=5](http://cabio.nci.nih.gov/servlet/GetXML?query=Gene&crit_name=BRCA1&crit_taxonId=5)

[http://cabio.nci.nih.gov/servlet/GetXML?query=Gene&crit\\_name=BRCA1&crit\\_taxon\\_id=5](http://cabio.nci.nih.gov/servlet/GetXML?query=Gene&crit_name=BRCA1&crit_taxon_id=5)

In the first case we are accessing the taxon information as an attribute of the *Gene* object. In the second case we are accessing the id as an attribute of the *Taxon* object associated with the *Gene*. Looking at this second statement, we see that the general syntax for accessing a “foreign” object’s attribute indirectly is “role\_tag=value”. Here the “role” is *taxon*—which is in relation to the *Gene* object, and the “tag” is *id*.

### 14.2.3 Syntax Summary

Each HTTP request must include *at least one* search criterion that can be associated with the requested object type. In both protocols, the first two terms in the *GetXML* expression are case-sensitive — using “Operation” instead of “operation” or “Query” instead of “query” will produce an error. The class names which provide values to these operators are also case-sensitive. The case sensitivity of the HTTP requests are summarized in Table 14.2-1

**Table 14.2-1 Summary of the HTTP syntax**

<i>operation</i>	=<class name>	Specifies the class name of the type of domain object to search for. The class name is case sensitive, as is the expression “operation=”.
	&<tag=value>	Specifies a settable attribute of the search criteria associated with the domain object identified in the class name. The tag and value are case <i>insensitive</i> .
<i>query</i>	=<class name>	Specifies the class name of the type of domain object to search for. The class name is case sensitive, as is the expression “query=”.
	&crit_<tag=value>	The expression “&crit_” is case sensitive. The tag specifies the name of an attribute local to the domain object immediately preceding &crit_. Neither the tag nor its value is case sensitive.
	&crit_<role_tag=value>	The “role” refers to a domain object which is in relation to the domain object immediately preceding &crit_. The attribute is local to the domain object immediately following &crit_. The role, tag, and value are all case <i>insensitive</i> .

For convenience, a catalog of the current HTTP operations and their allowed parameters are included at the end of this chapter. The available operations and parameters will be evolving over time however, and it is useful to understand how these catalogs are constructed:

- For the *operation=* syntax: the allowed arguments to *operation=* are the domain objects defined in the various bean packages. Given the selected domain object, the allowed parameters are determined by the associated *SearchCriteria* object’s *set* methods. Refer to the [Java Docs](#) to find the domain object class names and their corresponding search criteria’s *set* methods.
- For the *query=* syntax: the allowed arguments to *query=* are the domain objects defined in the various bean packages. Given the selected domain object, the arguments which can follow the *crit\_* tag are defined by that domain object’s *get* methods. Simple local attributes which return native Java types (String, float, etc.) are expressed with a simple “tag=value” statement, where the tag is that attribute’s variable name. Those *get* methods which return domain objects allow the user to insert additional selection criteria using an underscore, as in

“role\_tag =value.” The attribute in this case constrains the domain object identified by the role name however – *not* the domain object identified in the *query=expression*.

### 14.3 Drilling Down Through XLinks

The output returned by the caBIO server in response to an HTTP request is formatted XML with embedded XLinks. Using either syntax, there are two special request parameters that can further expand these XLinks in the XML output. The *returnHeavyXML* parameter will open up all of the embedded XLinks one level deep. For example, to search for genes whose symbol match “PTEN” and open up all XLinks, you would use:

```
operation=Gene&Symbol=PTEN&returnHeavyXML=1
```

An equivalent expression in the *query=* syntax is:

```
query=Gene&crit_Symbol=PTEN&returnHeavyXML=1
```

The other parameter, *fillInObjects*, “fills” in only those XLinks whose tags are specified in a comma-separated list. Thus, to open up only the *GoOntology* and *ExpressionFeature* tags in the XML output, you would use:

```
operation=Gene&Symbol=pTEN&fillInObjects=GoOntology,ExpressionFeature
```

### 14.4 Controlling the Number of Items Returned

It is also possible to fine-tune the default “throttling” mechanism defining the number of results returned on any single request. For example, assuming the search request yields, say, 500 results, specifying *resultStart=450* will return only the last 50. Similarly, one can use *resultCount=50* to get back only the first 50:

```
operation=Gene&symbol=pTEN&resultCount=50
```

Alternatively, you can use the parameter *ReturnCount* to specify the number of results to return, without concern for starting or ending indices. By default, the return results start at index 1 and the maximum number returned is 1,000.

### 14.5 Specifying the IP Address and Port in the URL

To use the HTTP API you need additional information such as the NCICB server and listening port for HTTP requests. The complete syntax of the HTTP request is:

```
http://<server:port>/servlet/GetXML?<Argument list>
```

where *<server:port>* is the caBIO web server and port number reserved for HTTP requests. As of the time of this writing, the port number is 80. These server addresses and port numbers may change; check with NCICB Application Support for the current specifications.

### 14.6 Applying XSL to XML Output

Some browsers (e.g., Netscape) cannot process XML-formatted documents, and on these platforms you will need to transform the XML response to an HTML document. As mentioned in the foregoing section, XSL/XSLT can be used for these purposes.

One option is to save the XML output you receive to a file and subsequently apply an appropriate XSL style sheet. Alternatively, you can use the *ApplyXSLT* servlet running on the caBIO web server to transform the XML output in real time.



*ApplyXSLT* requires two parameters: *mURL* and *xslURL*. *mURL* is a “modified” URL in which the original HTTP request is modified such that all instances of “?” are replaced with “\$” and all ampersands (“&”) are replaced with “@.” For example, the original HTTP request

<http://cabio.nci.nih.gov/servlet/GetXML?operation=Gene&Symbol=vegf>

would now become

```
http://cabio.nci.nih.gov/servlet/GetXML?$operation=Gene@Symbol=vegf
```

These modifications allow the new URL to be embedded inside a second URL. This is needed because the HTTP request will now be sent directly to the *ApplyXSLT* servlet, with the original request as an argument to that servlet. *ApplyXSLT* will then issue the original HTTP request, receive the results on behalf of the client, and transform the results using the style sheet specified in the *xslURL* parameter. The complete syntax of using *ApplyXSLT* is:

```
http://<server:port>/servlet/ApplyXSLT?mURL=<mURL>&xslURL=<xslURL>
```

For example:

[http://cabio.nci.nih.gov/servlet/ApplyXSLT?mURL=http://cabio.nci.nih.gov/servlet/GetXML\\$operation=Gene@Symbol=pten&xslURL=http://cabio.nci.nih.gov/servlet/xsl/cabio-beans.xsl](http://cabio.nci.nih.gov/servlet/ApplyXSLT?mURL=http://cabio.nci.nih.gov/servlet/GetXML$operation=Gene@Symbol=pten&xslURL=http://cabio.nci.nih.gov/servlet/xsl/cabio-beans.xsl)

## 14.7 The HTTP Operation Catalog

The next section catalogs the operations that can be invoked and the parameters these operations accept. As described above, the *operation* parameter in the HTTP URL is the name of the object class you wish to search on. Thus, if you are looking for genes, the URL should contain the string:

```
operation=Gene
```

The operation name must immediately follow the `GetXML?` request. As noted, all operations require that at least one attribute is included in the search criteria in order to limit the search. The following list of operations is taken directly from the list of caBIO domain object names. The parameters are defined as the settable attributes of the associated *SearchCriteria* objects. These attributes can also be gleaned from the embedded XLinks in the XML responses.

Figure 14.7-1 shows an excerpt of the XML document generated in response to:

<http://cabio.nci.nih.gov/servlet/GetXML?operation=Gene&Symbol=PTEN>

```

<?xml version="1.0" encoding="UTF-8" ?>
= <nci-core xmlns="http://ncicb.nih.gov/caBIO">
  = <gov.nih.nci.caBIO.bean.Gene xmlns="" id="2008"
    xmlns:xlink="http://www.w3.org/1999/xlink/">
    <name>PTEN</name>
    <title>phosphatase and tensin homolog (mutated in multiple
      advanced cancers 1)</title>
    <dbCrossRefs>{LOCUS_LINK=5728, OMIM=601728,
      UNIGENE=10712}</dbCrossRefs>
  = <gov.nih.nci.caBIO.bean.Chromosome id="21"
    xmlns:xlink="http://www.w3.org/1999/xlink/">
    <name>10</name>
  = <gov.nih.nci.caBIO.bean.Taxon id="5"
    xmlns:xlink="http://www.w3.org/1999/xlink/">
    <scientificName>Homo sapiens</scientificName>
    <abbreviation>Hs</abbreviation>
    <Gene
      xlink:href="http://cabio.nci.nih.gov:80/servlet/Get
        XML?operation=Gene&TaxonId=5" />
    </gov.nih.nci.caBIO.bean.Taxon>
  ...

```

Figure 14.7-1 XML excerpt in response to the Gene operation

In this example, the next to the last line in the XML excerpt shows an Xlink that contains a URL for invoking the *Gene* operation with the TaxonId attribute used as a filter.

## 14.8 The caBIO HTTP Catalog

### Agent Operation

*Definition:* Agent – a therapeutic agent (drug, intervention therapy) used in a clinical trial.

*Parameters:* agentNSCNumber, clinicalTrialProtocolId, comment, evsId, isCMAPAgent, name, source, targetId, therapyId

### Anomaly Operation

*Definition:* Anomaly – an irregularity in either the expression of a gene or its structure (i.e., a mutation).

*Parameters:* anomalyDescription, contextCode, histopathologyId, organId, targetId

### Chromosome Operation

*Definition:* Chromosome – an object representing a specific chromosome for a specific taxon; provides access to all known genes contained in the chromosome and to the taxon.

*Parameters:* name

### ClinicalTrialProtocol Operation

*Definition:* ClinicalTrialProtocol – the protocol associated with a clinical trial; organizes administrative information about the trial such as Organization ID, participants, phase, etc., and provides access to the administered Agents.

*Parameters:* agent, agentId, conceptId, cstepName, diseaseCategory, diseaseId, diseaseName, documentNumber, imtCode, leadOrganizationId, leadOrganizationName, nihAdminCode, pdqIdentifier, phase, piName, protocolAssociationId, title, treatmentFlag

### **Clone Operation**

*Definition:* Clone – an object used to hold information pertaining to I.M.A.G.E. clones; provides access to sequence information, associated trace files, and the clone’s library.

*Parameters:* geneId, name, sequenceId, snpId, verified

### **CMAPOntology Operation**

*Definition:* An object providing entry to the CMAP gene ontology, which categorizes genes by function; provides access to Gene objects corresponding to the ontological term, as well as to ancestor and descendant terms in the ontology tree. **Note:** the *CMAPOntologySearchCriteria* class inherits attributes from its parent class, *OntologySearchCriteria*.

*Parameters:* (*direct*) cMAPChildId, cMAPGeneId, cMAPName, cMAPParentId, cMAPOntologyId; (*inherited*) diseaseId, geneId, histopathologyId, name includeBoth, includeParents, includeChildren, relationshipParentId, relationshipChildId, relationshipType

### **ConsensusSequence Operation**

*Definition:* ConsensusSequence – a specialization of the Sequence class; represents the consensus of a set of Contigs, which it also provides access to.

*Parameters:* consensusSequenceType, contigId, geneId, proteinId, refGeneId

### **Contig Operation**

*Definition:* Contig – one of the set of overlapping sequence fragments used to assemble a ConsensusSequence, which it also provides access to.

*Parameters:* sequenceId, name

### **DiseaseRelationship Operation**

*Definition:* DiseaseRelationship – specifies a child or parent relationship between Disease objects. **Note:** the *DiseaseRelationshipSearchCriteria* class inherits attributes from its parent class *RelationshipSearchCriteria*.

*Parameters:* (*inherited*) relationshipChildId, relationshipParentId, relationshipType

### **Disease Operation**

*Definition:* Disease – an object that specifies a disease name and ID; also provides access to: ontological relations to other diseases; clinical trial protocols treating the disease; and specific histologies associated with instances of the disease. **Note:** the *DiseaseSearchCriteria* class inherits attributes from its parent class, *OntologySearchCriteria*.

*Parameters:* (*inherited*) diseaseId, histopathologyId, geneId, includeBoth, includeChildren, includeParents, name, relationshipChildId, relationshipParentId, relationshipType

### **EstExperiment Operation**

*Definition:* EstExperiment – an object that represents data from an expression experiment using expressed sequence tags. **Note:** the *EstExperimentSearchCriteria* class inherits attributes from its parent class, *ExpressionExperimentSearchCriteria*.

*Parameters:* (direct) contextId; (inherited) expressionFeatureId, gene, geneId, organ, proteinId, taxonId, threshold, type

### **ExpressionFeature Operation**

*Definition:* ExpressionFeature – an object associated with a Gene that provides access to the list of Organs where the Gene is expressed.

*Parameters:* geneId, expressionLevelDescId

### **ExpressionMeasurementArray Operation**

*Definition:* ExpressionMeasurementArray – an array of ExpressionMeasurements.

*Parameters:* AccessionNumber, expressionMeasurementId, name

### **ExpressionMeasurement Operation**

*Definition:* ExpressionMeasurement – an object representing a structure capable of measuring the absolute or relative amount of an expressed compound.

*Parameters:* accessionNumber, expressionMeasurementArrayId, geneId, name, sequenceId

### **GeneAlias Operation**

*Definition:* GeneAlias – an alternative name for a gene; provides descriptive information about the gene (as it is known by this alias), as well as access to the Gene object it refers to.

*Parameters:* description, geneId, type

### **GeneHomolog Operation**

*Definition:* Defined only in relation to another Gene, the GeneHomolog in caBIO is the functional equivalent of that gene in another taxon (i.e., its ortholog). The GeneHomolog object is a specialization of the parent Gene object; in addition to all of the methods provided by the gene interface, the homolog provides its percent of sequence similarity to the related gene of interest.

*Parameters:* geneId

### **Gene Operation**

*Definition:* Gene – the effective portal to most of the genomic information provided by the caBIO data Operations; organs, diseases, chromosomes, pathways, sequence data, and expression experiments are among the many objects accessible via a gene.

*Parameters:* allPathwayId, bcId, chromosomeId, cloneName, cMAPOntologyId, cytogenicLocation, expressedPathwayId, expressionMeasurementId, functionalPathway, genBankAccessionNumber, GeneKeyword, geneNameKeyword, goOntologyHomoSapienId, goOntologyId, goOntologyMouseId, keyword, mutatedGenePathwayId, organism, overExpressedPathwayId, pathwayId, symbol, targetId, taxonId, tissueType, underExpressedPathwayId, unigeneClusterId, uniqueIdentifier,

### **GoOntologyRelationship Operation**

*Definition:* GoOntologyRelationship – an object that specifies a child or parent relationship between *GoOntology* objects. **Note:** the *GoOntologyRelationshipSearchCriteria* class inherits attributes from its parent class, *RelationshipSearchCriteria*.

*Parameters:* (*inherited*) relationshipChildId, relationshipParentId, relationshipType

### **GoOntology Operation**

*Definition:* GoOntology – an object that provides entry to a Gene object’s position in the Gene Ontology Consortium’s controlled vocabularies, as recorded by LocusLink; provides access to Gene objects corresponding to the ontological term, as well as to ancestor and descendant terms in the ontology tree. **Note:** the *GoOntologySearchCriteria* class inherits attributes from its parent class, *OntologySearchCriteria*.

*Parameters:* (*direct*) geneId; (*inherited*) diseaseId, geneId, histopathologyId, includeBoth, includeParents, includeChildren, name, relationshipParentId, relationshipChildId, relationshipType

### **Histopathology Operation**

*Definition:* Histopathology – an object that represents anatomical changes in a diseased tissue sample associated with an expression experiment; also captures the relationship between organ and disease.

*Parameters:* diseaseId, expressionExperimentId, name, organId

### **Library Operation**

*Definition:* Library – an object that provides access to CGAP library information about the tissue sample and its method of preparation, the library protocol that was used, the clones contained in the library, and the sequence information derived from the library.

*Parameters:* geneId, libraryGroup, libraryName, libraryProtocol, organism, sortOrder, tissueHistology, tissueName, tissuePreparation, tissueType

### **MapLocation Operation**

*Definition:* MapLocation – an object that represents the physical map location of a gene.

*Parameters:* type, location, geneId

### **OrganRelationship Operation**

*Definition:* OrganRelationship – an object that specifies a child or parent relationship between Organ objects. **Note:** the *GoOrganRelationshipSearchCriteria* class inherits attributes from its parent class, *RelationshipSearchCriteria*.

*Parameters:* (*inherited*) proteinId, relationshipChildId, relationshipParentId, relationshipType

### **Organ Operation**

*Definition:* Organ – a representation of an organ whose name occurs in a controlled vocabulary; provides access to any Histopathology objects for the organ, and, because it is “ontolog-able,” provides access to its ancestral and descendant terms in the vocabulary. **Note:** the *OrganSearchCriteria* class inherits attributes from its parent class, *OntologySearchCriteria*.

*Parameters:* (*direct*) anomaly\_id, expressionFeatureId, histopathologyId; (*inherited*) diseaseId, geneId, includeBoth, includeChildren, includeParents, name, relationshipChildId, relationshipParentId, relationshipType

### **Pathway Operation**

*Definition:* Pathway – an object representing a molecular/cellular pathway compiled by [BioCarta](#). Pathways are associated with specific Taxa, and contain multiple Genes, which may be Targets for treatment.

*Parameters:* bioProcessId, context, displayValue, geneId, name, pathwayDiagram, taxonId

### **ProteinHomolog Operation**

*Definition:* Defined only in relation to another Protein of interest, the ProteinHomolog in caBIO is the functional equivalent of that protein in another taxon (i.e., its ortholog). The ProteinHomolog is a specialization of the parent Protein object; in addition to the methods inherited from Protein, the homolog provides its percent of sequence similarity to the related protein of interest.

*Parameters:* proteinId

### **Protein Operation**

*Definition:* Protein – an object representation of a protein; provides access to the encoding gene via its [GenBank](#) ID, the taxon in which this instance of the protein occurs, and references to homologous proteins in other species.

*Parameters:* accessionNumber, description, geneId

### **ProtocolAssociation Operation**

*Definition:* ProtocolAssociation – an object that associates ClinicalTrialProtocols to Diseases.

*Parameters:* clinicalTrialProtocolId, protocolId

### **Protocol Operation**

*Definition:* Protocol – an object that represents the protocol used in assembling a clone library.

*Parameters:* name

### **ReadSequence Operation**

*Definition:* ReadSequence – an object representing the output of a TraceFile, an ASCII representation of the nucleotide sequence; a read sequence is created by running PHRED.

*Parameters:* cloneId, geneId, proteinId, readSequenceId, refGeneId, traceFileId

### **SageExperiment Operation**

*Definition:* SageExperiment – a subclass of the ExpressionExperiment class, used to represent serial analysis of gene expression (SAGE) data. **Note:** the *SageExperimentSearchCriteria* class inherits attributes from its parent class *ExpressionExperimentSearchCriteria*.

*Parameters:* (*direct*) contextId; (*inherited*) expressionFeatureId, gene, geneId, organ, proteinId, taxonId, threshold, type

### **Sequence Operation**

*Definition:* Sequence – an object representing a gene sequence and providing access to the clones from which it was derived, the ASCII representation of the residues it contains, and the sequence ID.

*Parameters:* accessionNumber, cloneId, contigId, expressionMeasurementId, geneId, isRefSeq, refGeneId, returnDna, sequenceType

### **SNP Operation**

*Definition:* SNP – an object that represents a Single Nucleotide Polymorphism; provides access to the clones and the trace files from which it was identified, the two most common substitutions at that position, the offset of the SNP in the parent sequence, and a confidence score.

*Parameters:* geneId

### **Target Operation**

*Definition:* Target – an object that represents a gene thought to be at the root of a disease etiology and which is targeted for therapeutic intervention in a clinical trial.

*Parameters:* agentId, anomalyDescription, anomalyId, cancerType, conceptId, geneId

### **Taxon Operation**

*Definition:* Taxon – an object representing the various names (scientific, common, abbreviated, etc.) for a species associated with a specific Gene, Chromosome, Pathway, Protein, or Tissue.

*Parameters:* abbreviation, animalModelId, chromosomeId, id, isPreferred, name, regulatoryElementId, scientificName, strainId, xenograftId

### **Tissue Operation**

*Definition:* Tissue – defined by any group of similar cells united to perform a specific function.

*Parameters:* libraryId

### **TraceFile Operation**

*Definition:* TraceFile – an object that represents the recorded trace file used to identify a SNP, based on the observed intensities for the four possible bases at each position in the sequence.

*Parameters:* cloneId, name, snpId

## **14.9 The EVS HTTP Catalog**

### **DescLogicConceptService**

*Definition:* A description logic concept represented in the NCI Thesaurus.

*Parameters:* conceptCode, initialDate, property, role, vocabularyName; (*inherited*) allSource, limit, searchTerm, source

### **MetathesaurusConceptService**

*Definition:* A concept in the NCI Metathesaurus.

*Parameters:* code, score, semanticType, shortResult; (*inherited*) allSource, limit, searchTerm, source

## 14.10 The caDSR HTTP Catalog

### CaseReportForm Operation

*Definition:* A questionnaire that is a collection of DataElements used to document patient information stipulated in a protocol.

*Parameters:* displayName; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

### ClassificationSchemeItem Operation

*Definition:* An item or category in a ClassificationScheme used to classify other components; for example, a node in a taxonomy.

*Parameters:* comments, dateCreated, dateModified, description, name, type

### ClassificationScheme Operation

*Definition:* Any set of organizing principles or dimensions along which data can be organized. A ClassificationScheme may be a simple collection of keywords or a complex ontology.

*Parameters:* labelTypeFlag, type; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

### ClassSchemeClassSchemeItem Operation

*Definition:* A component used to associate a set of ClassificationSchemeItems with a particular ClassificationScheme, and to store details of that association such as the display order of the items within that scheme.

*Parameters:* dateCreated, dateModified, displayOrder, label

### ConceptualDomain Operation

*Definition:* The set of all possible ValidValue meanings of a DataElementConcept expressed without representation.

*Parameters:* dimensionality; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

### Context Operation

*Definition:* A designation or description of the application environment or discipline in which a name is applied or from which it originates.

*Parameters:* dateCreated, dateModified, description, designationId, languageName, name, version

### DataElementConceptRelationships Operation

*Definition:* A description of the affiliation between two occurrences of DataElementConcepts.



*Parameters:* dateCreated, dateModified, description, name

### **DataElementConcept Operation**

*Definition:* A concept that can be represented in the form of a DataElement and described independent of any particular representation.

*Parameters:* (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

### **DataElement Operation**

*Definition:* A unit of data for which the definition, identification, representation, and permissible values are specified by means of a set of attributes.

*Parameters:* (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

### **Designation Operation**

*Definition:* A name by which an Administered Component is known in a specific Context. Also a placeholder to track the usage of Administered Components by different Contexts.

*Parameters:* dateCreated, dateModified, languageName, name, type

### **EnumeratedValueDomain Operation**

*Definition:* A ValueDomain expressed as a list of all PermissibleValues.

*Parameters:* (*inherited*) characterSetName, dataTypeName, decimalPlace, formatName, highValueNumber, lowValueNumber, maximumLengthNumber, minimumLengthNumber, uomNamebeginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

### **Module Operation**

*Definition:* A collection of DataElements logically grouped on a case report form.

*Parameters:* displayOrder; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

### **NonenumeratedValueDomain Operation**

*Definition:* A ValueDomain expressed by a generative formula or range of allowed values.

*Parameters:* (*inherited*) characterSetName, dataTypeName, decimalPlace, formatName, highValueNumber, lowValueNumber, maximumLengthNumber, minimumLengthNumber, uomNamebeginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

### **ObjectClass Operation**

*Definition:* A set of ideas, abstractions, or things in the real world that can be identified with explicit boundaries and meaning and whose properties and behavior follow the same rules.

*Parameters:* definitionSource; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

### **PermissibleValue Operation**

*Definition:* The exact names, codes and text that can be stored in a data field.

*Parameters:* beginDate, dateCreated, dateModified, endDate, highValueNumber, lowValueNumber, value

### **Property Operation**

*Definition:* A characteristic common to all members of an ObjectClass.

*Parameters:* definitionSource; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

### **ProtocolFormsSet Operation**

*Definition:* A specific clinical trial protocol document and its collection of associated CRFs.

*Parameters:* approvedBy, approvedDate, changeNumber, changeType, leadOrganizationName, phase, protocolId, reviewedBy, reviewedDate, type; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

### **ProtocolFormsTemplate Operation**

*Definition:* A collection of components (Modules, Questions) to be included in a CRF.

*Parameters:* (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

### **Qualifier Operation**

*Definition:* A term that helps define and render a concept unique.

*Parameters:* comments, dateCreated, dateModified, description, name

### **Question Operation**

*Definition:* The actual text of the DataElement as specified on a CaseReportForm of a protocol

*Parameters:* displayOrder; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

### **ReferenceDocument Operation**

*Definition:* A place to document additional information about AdministeredComponents.

*Parameters:* dateCreated, dateModified, displayOrder, docText, languageName, name, organizationId, rdttlName, type, url

### **Representation Operation**

*Definition:* Mechanism by which the functional and/or presentational category of an item may be conveyed to a user.

*Parameters:* definitionSource; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

### **ValidValue Operation**

*Definition:* The allowable values for a given DataElement (Question) on a CaseReportForm

*Parameters:* displayOrder; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

### **ValueDomainPermissibleValue Operation**

*Definition:* An object that stores the many to many relationships between ValueDomains and PermissibleValues.

*Parameters:* dateCreated, dateModified

### **ValueDomain Operation**

*Definition:* A set of PermissibleValues for a DataElement

*Parameters:* characterSetName, dataTypeName, decimalPlace, formatName, highValueNumber, lowValueNumber, maximumLengthNumber, minimumLengthNumber, uomName; (*inherited*) beginDate, changeNote, dateCreated, dateModified, deletedIndicator, endDate, latestVersionIndicator, longName, origin, preferredDefinition, preferredName, publicId, unresolvedIssue, version, workflowStatusDescription, workflowStatusName

### **ValueMeaning Operation**

*Definition:* The significance associated with an allowable/permissible value.

*Parameters:* beginDate, comments, dateCreated, dateModified, description, endDate, shortMeaning

## **14.11 The caMOD HTTP Catalog**

### **AnimalModel Operation**

*Definition:* A strain of genetically modified animals (rat or mouse) used to study the molecular etiology of various types of cancer.

*Parameters:* agentId, availabilityId, diseaseId, experimentDescription, modelDescriptor, partyRoleId, phenotypeId, taxonId

### **Availability Operation**

*Definition:* The availability status of a developed strain from the MMHCC, from the Jackson Laboratory, or directly from the principal investigator.

*Parameters:* animalModelId, enteredDate, modifiedDate, releaseDate, visibleTo

### **CarcinogenicIntervention Operation**

*Definition:* A treatment (chemical or drug administration, radiation, genetic modification, knockout, etc.) applied to an AnimalModel to induce a disease state.

*Parameters:* animalModelId, environmentalFactorId, geneDeliveryId, treatmentScheduleId

### **CellLine Operation**

*Definition:* A CellLine associated with a particular strain of AnimalModel.

*Parameters:* animalModelId, comments, experiment, name, organId

### **Conditionality Operation**

*Definition:* Conditionality is used to indicate whether or not the administration of a transgene or targeted modification depends on time- or tissue-specific conditions.

*Parameters:* conditionedBy, Desc

### **ContactInfo Operation**

*Definition:* Contact information for the principal investigator of the selected model.

*Parameters:* city, email, fax, institute, labName, partyId, phoneNumber, state, street, zip

### **EngineeredGene Operation**

*Definition:* A gene sequence genetically modified to induce a desired state in an AnimalModel.

*Parameters:* caBioId, conditionalityId, dbCrossRefs, genomicSegmentId, genotypeSummaryId, imageId, inducedMutationId, locusLinkSummary, name, targetedModificationId, title

### **EnvironmentalFactor Operation**

*Definition:* A contributing factor to the disease state of an AnimalModel.

*Parameters:* carcinogenicId, name, type

### **GeneDelivery Operation**

*Definition:* The method of introducing a modified gene to the recipient animal.

*Parameters:* engineeredGeneId, geneId, organId, viralVector

### **GeneticAlteration Operation**

*Definition:* An intentionally induced change in a gene sequence or in the number of gene copies.

*Parameters:* histopathologyId, methodOfObservation, observation

### **GenomicSegment Operation**

*Definition:* A region or set of regions of a genome including chromosome, gene, breakpoint, etc.

*Parameters:* animalModelId, cloneDesignator, integrationTypeId, LocationOfIntegration,

SegmentSize, segmentTypeId

### **GenotypeSummary Operation**

*Definition:* GenotypeSummary information for a particular model.

*Parameters:* genotype, nomenclatureID, summary

### **Image Operation**

*Definition:* An image of the diseased tissues or organs from an AnimalModel.

*Parameters:* animalModelId, description, image, staining, title

### **InducedMutation Operation**

*Definition:* A mutation in a Gene caused by exposure to chemicals, radiation or other mutagens.

*Parameters:* animalModelId, environmentalFactorId

### **MicroArrayData Operation**

*Definition:* Expression data from microarray experiments with samples from AnimalModels.

*Parameters:* animalModelId, experimentId, experimentName

### **ModificationType Operation**

*Definition:* The type of gene modification in the target gene, such as the *Null* modification, change in amino acid, deletion, insertion, misense, nonsense, point mutation, etc.

*Parameters:* modificationTypeName, targetModificationId,

### **Nomenclature Operation**

*Definition:* Controlled vocabulary terms used for the MMHCC.

*Parameters:* name

### **Person Operation**

*Definition:* An individual involved in the deposition, review, and/or approval of MMHCC data.

*Parameters:* firstName, lastName

### **Phenotype Operation**

*Definition:* The physical appearance or otherwise observable characteristics of a model animal.

*Parameters:* animalModelId, breedingNotes, desc, sexDistributionId

### **Promoter Operation**

*Definition:* A region of DNA sequence upstream of the coding region to which RNA polymerase will bind and initiate replication.

*Parameters:* name, regulatoryElementTypeId, speciesId, transGeneId

### **Publication Operation**

*Definition:* A paper or article associated with MMHCC data published in a scientific journal.

*Parameters:* animalmodelId, authors, cellLineId, endPage, journal, pmId, publicationStatusId, startPage, status, therapyId, title, volume, year

### **RegulatoryElement Operation**

*Definition:* A region of DNA sequence controlling the transcription/expression of a gene.

*Parameters:* name, regulatoryElementTypeId, speciesId, transGeneId

### **RegulatoryElementType Operation**

*Definition:* The type of regulation imposed by the element, e.g., suppressor, promoter, etc.

*Parameters:* regulatoryElementTypeName

### **Role Operation**

*Definition:* The Role that a person or organization plays; for example, submitter, reviewer, etc.

*Parameters:* roleName

### **SegmentType Operation**

*Definition:* Genetic segment type such as chromosome, contig, CpG islands, repetitive DNA (e.g. Alu, LINE, SINE etc.), gene, breakpoint, etc.

*Parameters:* segmentTypeName

### **SexDistribution Operation**

*Definition:* The observable distribution of phenotypes between sexes.

*Parameters:* SexDistributionTypeName

### **TargetedModification Operation**

*Definition:* Modification of a specific gene (versus one randomly selected) by a technology called gene targeting through homologous recombination.

*Parameters:* animalModelId, blastocystName, engineeredGeneId, escellLineName, geneId, modificationTypeId,

### **Therapy Operation**

*Definition:* A defined treatment protocol for testing the efficacy of the treatment on an engineered AnimalModel.

*Parameters:* agentId, animalModelId, comments, experiment, treatmentScheduleId

### **Transgene Operation**

*Definition:* A gene that has been integrated into the germ line of a transgenic animal by gene targeting technology.

*Parameters:* animalModelId, integrationTypeId, locationOfIntegration, speciesId

### **TreatmentSchedule Operation**

*Definition:* The dosage and regimen for treating cancer in an AnimalModel.

*Parameters:* carcinogenicInterventionId, dosage, regimen, therapyId

### **Xenograft Operation**

*Definition:* A surgical graft of tissue from one species onto or into individuals of unlike species, genus, or family.

*Parameters:* administrativeSite, animalModelId, geneticManipulation, hostSpeciesId, modificationDescription, name, organId, originSpeciesId, parentCellLineName, type

## **15.0 THE caCORE DATA SOURCES**



## 15.1 Data Sources in the caBIO Database

The caCORE application programming interfaces were developed primarily in response to the need for programmatic access to the information at several NCI web sites, including:

- the Cancer Genome Anatomy Project ([CGAP](#))
- the CGAP Genetic Annotation Initiative ([GAI](#))
- the Enterprise Vocabulary Services ([EVS](#))
- the Cancer Data Standards Repository ([caDSR](#))
- the Mouse Models of Human Cancers Consortium ([MMHCC](#))
- the Cancer Molecular Analysis Project ([CMAP](#))
- the Gene Expression Data Portal ([GEDP](#))

While all of these sites provide information and search tools relevant to the molecular analysis of cancer, each organizes its information somewhat differently, emphasizing for example, gene sequences, clone libraries, chromosome maps, cancer terminologies, DNA microarray data, or clinical trials data. The primary operation in the caCORE APIs involves defining an object type of interest along with a set of search criteria for that object type, and retrieving all instances of that object that satisfy the defined search criteria.

For example, the goal might be to find all genes that are expressed in bone marrow cells, where those genes are also known to participate in apoptosis. Using the Java API, the user would first instantiate a *Gene* object and a *GeneSearchCriteria* object. The methods *setFunctionalPathway()* and *setTissueType()*, associated with *GeneSearchCriteria* objects, could then be applied to define the search criteria. A subsequent call to *myGene.search(myCriteria)* would then retrieve all genes known to satisfy these criteria.

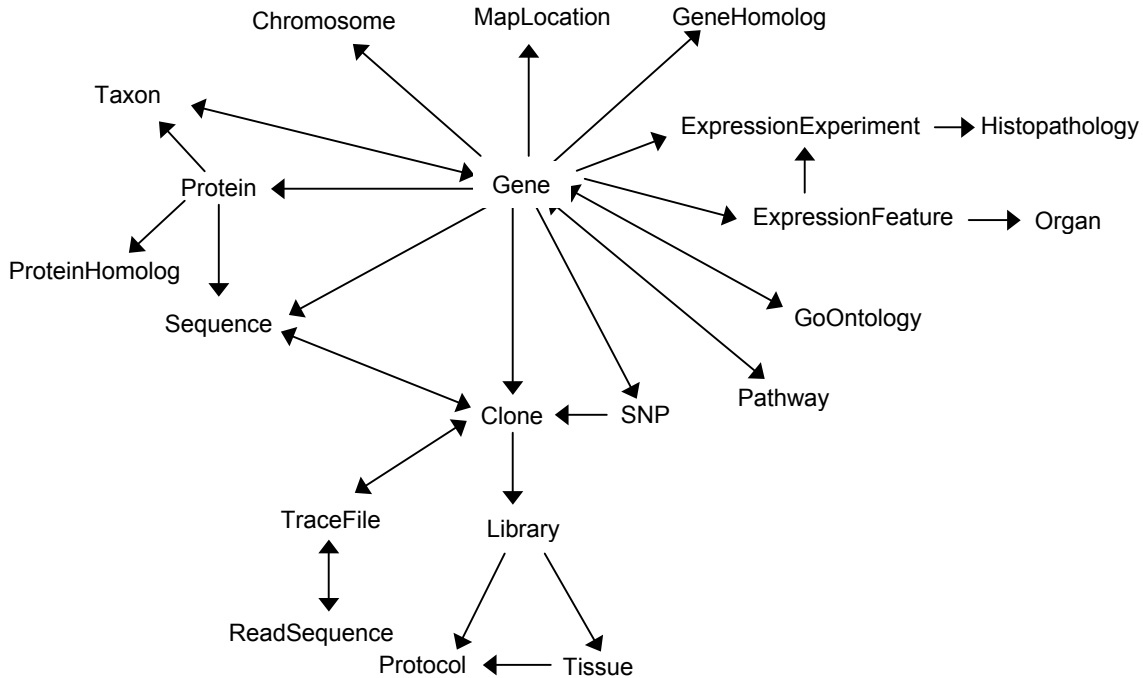
While this information is in theory available from multiple public sites, the number of links to traverse and the extent of collation that would have to be performed is daunting. The CGAP, CMAP, and GAI web sites have distilled this information from both internal and public databases, and the caBIO data warehouses have optimized it for access with respect to the types of queries defined in the APIs. This section discusses the external and internal data sources for caBIO and how the information these sources provide can be accessed via caBIO objects.

The caBIO objects fall roughly into two categories: those that pertain to clinical trials data, and those that are relevant to basic research. The two groups are not mutually exclusive, as some objects such as *Gene*, *Organ*, and *Histopathology* occur in both. In particular, the *Gene* object functions as a central hub of the basic research objects and, accordingly, serves as a portal between the two relatively disjunct groups.

Before discussing the specific data sources whose information is made available via the caBIO objects, it is useful to consider the types of data that might be needed to investigate the molecular basis of cancer. The challenges are to discover which chromosome aberrations, DNA mutations, and single nucleotide polymorphisms may lead to or be associated with neoplasm formations and/or cancerous preconditions, as well as what genetic idiosyncracies may affect variable responses to treatment.

Clearly, sequence information must be available, including whole genomic sequences, expressed mRNA sequences, expressed sequence tags, and single nucleotide polymorphisms. Moreover, this sequence information must be available from multiple sources, including both

normal and diseased tissue, so as to allow statistical analysis and identification of significant correlations. But it is not enough to provide the sequence data alone, devoid of any source information. In particular, it must be possible to identify the tissue types, histological states, and preparation methods of the samples, as well as the protocols used in generating the libraries from which the sequences were extracted. As depicted in Figure 15-1, *Clone* objects can be accessed directly from either a *Gene*, *Sequence*, or *SNP* object. The *Clone* object, in turn, provides access to information about the protocol and preparation methods for its associated *Library*, as well as access to *TraceFile* objects.



**Figure 15.1-1 caBIO objects supporting basic research**

Figure 15.1-1 is a very reduced view of the entire collection of caBIO objects, showing only those objects that are most relevant to basic research. The links between objects in Figure 15.1-1 reflect only the *get* methods defined for those objects. For example, a *Library* object has *getTissue()* and *getProtocol()* methods; neither the *Tissue* nor *Protocol* objects have a *getLibrary()* method, however, so the links are unidirectional. Each object also provides access to a wealth of additional information not shown in Figure 15.1-1.

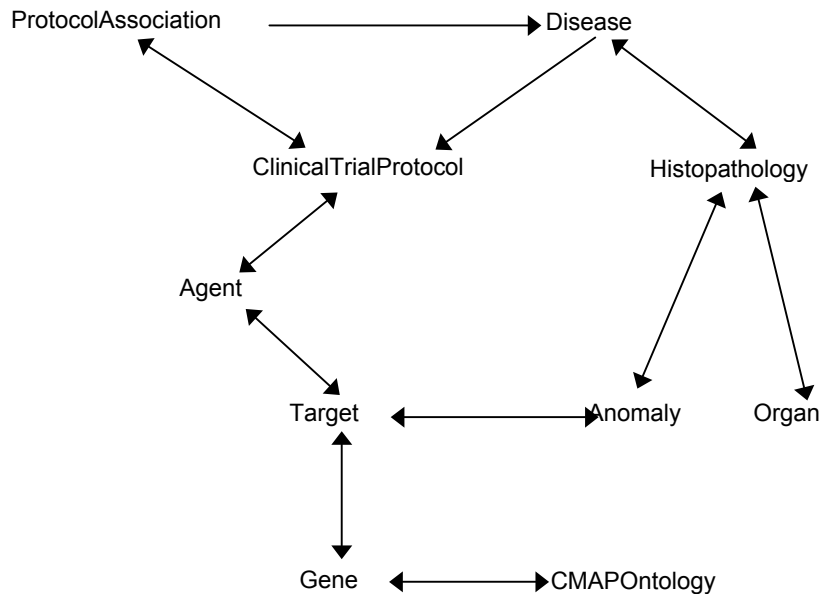
Thus, an application that attempts to identify new SNPs might use the *Clone* object to gain access to *TraceFiles*. Alternatively, an application that attempts to correlate known SNPs with disease states might use the *Clone* objects to filter SNPs according to tissue type, preparation method, and library protocol.

Chromosome and map location information are important to studies that focus on oncology at the cytogenetic level. Given a chromosome that is known to have aberrations associated with cancer, this information can be used to drill down to the molecular level using the caBIO objects and appropriate search criteria on the chromosomes and map locations of genes and sequences.

Another common focus is on proteomic pathways, for example cell cycle control. The caBIO *Pathway* objects provide methods to selectively retrieve the genes occurring on that pathway, according to whether they are mutated, overexpressed, or underexpressed, etc. Thus, starting from a *Pathway* that is hypothesized to be involved in some disease etiology, it is possible to first retrieve the associated *Gene* objects and, subsequently, explore the features of each *Gene*, including its chromosome and map locations, variable expression levels (via *ExpressionExperiment* objects), and its position in the [Gene Ontology Consortium](#) hierarchies (via its *GoOntology* objects).

With the wealth of bioinformatic data that has emerged over the past decade, the need for translational research that can deliver these advances in knowledge and understanding to the clinical setting has become increasingly critical. Thus, another important type of information that must be available is clinical data.

The caBIO objects that are geared to clinical research form a clique or subgrouping among the larger set of objects and are displayed separately in Figure 15.1-2. Objects that appear in both “sub-networks” include the *Gene*, *Organ*, and *Histopathology* objects.



**Figure 15.1-2 caBIO objects supporting clinical research**

The remainder of this section discusses the external and internal data sources whose information is used to populate the objects in Figures 15.1-1 and 15.1-2. While the caBIO data are extracted from many sources that include information from a wide variety of species, we emphasize that *only genomic data pertaining to human and mouse are available from caBIO*. caBIO provides access to curated data from multiple sources, including:

- The NCBI [UniGene](#) database [1 – 3]. Unigene provides a nonredundant partitioning of the genetic sequences contained in GenBank into gene clusters. Each such cluster has a unique UniGene ID and a list of the mRNA and EST sequences that are subsumed by that cluster. Related information stored with the cluster includes tissue types in which the gene has been expressed, mapping information, and the associated LocusLink, OMIM, and HomoloGene

IDs, thus providing access to related information in those NCBI databases as well. Because the information in UniGene is centered around genes, access to Unigene is provided via the caBIO *Gene* objects. Specifically, the method *getClusterId()* associated with a *Gene* object can be used to fetch the gene's UniGene ID. Similarly, the database IDs for the NCBI OMIM and LocusLink databases can be obtained using the *getOMIMId()* and *getLocusLinkId()* methods. While there is no explicit caBIO object corresponding to a Unigene cluster, all of the information associated with the cluster is available directly via the caBIO *Gene* object's methods. For example:

- *getGenomicSequences()* returns an array containing the mRNA and EST sequences contained in the Unigene cluster;
- *getExpressionFeature()* returns an *ExpressionFeature* object, which can in turn be queried to obtain a list of the tissues in which the gene is expressed;
- *getGeneHomologs()* returns an array of *GeneHomolog* objects for the gene;
- *getChromosome()* returns the *Chromosome* on which this gene occurs;
- *getMapLocation()* returns an array of *MapLocation* objects associated with the gene.

In all of the above methods, the returned value is itself a caBIO object. Thus, further information associated with the returned object can in turn be accessed using that object's methods.

The information stored with an *ExpressionFeature* object requires a bit more explanation, as it is not actually a copy of what is stored in Unigene. caBIO's expression information is instead derived as the result of passing the Unigene free-text information through a controlled vocabulary that defines only about 55 tissue types. Using an ontology to match the Unigene terms to terms in the caBIO vocabulary, the result is generally a condensed version, as several terms in the Unigene data may map to the same more general term in the vocabulary.

- NCBI's [LocusLink](#) database [4, 5]. LocusLink contains curated sequence and descriptive information associated with a gene. Each entry includes information about the gene's nomenclature, aliases, sequence accession numbers, phenotypes, UniGene cluster IDs, OMIM IDs, gene homologies, associated diseases, map locations, and a list of related terms in the [Gene Ontology Consortium](#)'s ontology. Sequence accessions include a subset of GenBank accessions for a locus, as well as the NCBI Reference Sequence. As mentioned above, a caBIO *Gene* object has explicit methods for retrieving the gene's associated LocusLink, OMIM, and Unigene IDs. The methods to access the gene's reference sequences and aliases are *getReferenceSequences()* and *getAliases()*, respectively. Related terms in the GO ontology are retrieved using the gene's *getGoOntologies()* method (see discussion below). Finally, a *Gene* object's *getLocusLinkSummary()* method returns a free-text paragraph summarizing gene function.
- The [Gene Ontology Consortium](#) [6, 7]. The Gene Ontology Consortium provides a controlled vocabulary for the description of molecular functions, biological processes, and cellular components of gene products. The terms provided by the consortium define the recognized attributes of gene products and facilitate uniform queries across collaborating databases. The caBIO *Gene* object's *getGoOntologies()* method returns a list of *GoOntology* objects for the

gene, which can in turn be queried to examine relationships among genes and other terms in the gene ontology.

In general, each gene is associated with one or more biological processes, and each of these processes may in turn be associated with many genes. In addition, the GO ontologies define many parent/child relationships among terms. For example, a branch of the ontology tree under `biological_process` contains the term `cell cycle control`, which in turn bifurcates into the “child” terms `cell cycle arrest`, `cell cycle checkpoint`, `control of mitosis`, etc. caBIO’s *GoOntology* objects capture these relations via the *getChildRelationships()*, *getParentRelationships()*, *getOntologyHomoSapienGenes()*, and *getOntologyMouseGenes()* methods. Thus, it is possible to start with a *Gene* object and retrieve its *GoOntology* objects, and, from there, traverse a network of related genes via the links deriving from the ontological terms.

As mentioned above, caBIO does not extract ontology terms directly from the Gene Ontology Consortium but, instead, extracts those terms stored with the LocusLink entry for that gene.

- The [HomoloGene](#) database [8]. HomoloGene is an NCBI resource for curated and calculated gene homologs. The caBIO data sources capture only the calculated homologs stored by HomoloGene. These calculated homologs are the result of nucleotide sequence comparisons performed between each pair of organisms represented in UniGene clusters. The caBIO *Gene* method to access the gene’s homologs is *getGeneHomologs()*, and returns an array of *GeneHomolog* objects.
- [BioCarta](#) pathways. BioCarta and its Proteomic Pathway Project (P3) provides detailed graphical renderings of pathway information concerning adhesion, apoptosis, cell activation, cell signalling, cell cycle regulation, cytokines/chemokines, developmental biology, hematopoiesis, immunology, metabolism, and neuroscience. NCI’s CMAP web site captures pathway information from BioCarta, and transforms the downloaded image data into Scalable Vector Graphics ([SVG](#)) representations that support interactive manipulation of the online images. The CMAP web site displays BioCarta pathways selected by the user and provides options for highlighting *anomalies*, which include under- or overexpressed genes as well as mutations.

The caBIO *Pathway* objects make this same information available via their associated methods, which include: *getGenes()*, *getExpressedGenes()*, *getMutatedGenes()*, *getOverExpressedGenes()*, *getUnderExpressedGenes()*, and *getTargetGenes()*. The pathway diagram is also available, as an XML document (*getPathwayDiagram()*) or in SVG format (*getSvgPathwayDiagram()*). The expression and mutation information that is associated with the Pathway object is derived from EST and SAGE expression data that have been culled by the CGAP project. Information about target genes is taken from data stored with CMAP.

- The Distributed Annotation System ([DAS](#)) [9] at UCSC. DAS is a client-server system that allows a single client machine to collect genome annotation information from multiple distant servers, collate the information, and display it in a single view, with little or no coordination among the information providers. DAS/1 servers are currently running at WormBase, FlyBase, Ensembl, TIGR, and UCSC. caBIO provides access to the DAS

information at UCSC, via the caBIO objects defined in the *gov.nih.nci.caBIO.util.das* package.

The starting point for any DAS search is one of the three DAS search criteria objects: *DasTypeSearchCriteria*, *DasDnaSearchCriteria*, and *DasGffFeatureSearchCriteria*. The first of these can be used to obtain an array of annotation types, the second fetches DNA sequences, and the last retrieves annotations satisfying the specified criteria. Further documentation on these search criteria objects and their affiliated domain objects can be found in the [JavaDocs](#) pages for the *das* package.

- The Cancer Genome Anatomy Project [10 – 13] ([CGAP](#)). The NCI CGAP web site provides a collection of gene expression profiles of normal, pre-cancer, and cancer cells taken from various tissues. The CGAP interface allows the user to browse these profiles by various search criteria, including histology type, tissue type, library protocol, and sample preparation methods. The goal at NCI is to exploit such expression profile information for the advancement of improved detection, diagnosis, and treatment for the cancer patient. Researchers have access to all CGAP data and biological resources for human and mouse, including ESTs, gene expression patterns, SNPs, cluster assemblies, and cytogenetic information.

The CGAP web site provides a powerful set of interactive data-mining tools to explore these data, and the caBIO project was initially conceived as a programmatic interface to these tools and data. Accordingly, most of the data that are available from CGAP can also be accessed through the caBIO objects. Exceptions are those data sets having proprietary restrictions, such as the Mittleman Chromosome Aberration database.

The caBIO *ExpressionExperiment* object provides a generic interface to the CGAP expression data, with methods including: *getType()*, *getExpressables()*, *getExpressionLevel()*, and *getHistopathologies()*. The type information returned by the first method is simply a string with the value “SAGE” or “EST.” *Expressable* is a Java interface that is currently implemented only by the caBIO *Gene* objects. Thus, the *getExpressables()* method of an *ExpressionExperiment* returns the set of *Gene* objects whose expression levels were detected. *getExpressionLevel()* returns an array of *ExpressionLevel* objects, where each of these objects in turn provides information about the observed expression ratio for that gene. *getHistopathologies()* returns an array of *Histopathology* objects. A *Histopathology* object provides information about the organ and disease where the pathology was observed, along with information about the type of anomaly (mutation or variation in expression) associated with the histopathology.

Two subclasses, *ESTExperiment* and *SAGEExperiment*, inherit their methods from the *ExpressionExperiment* object and provide access to CGAP’s EST and SAGE data, respectively. The immediate source for EST library metadata (who made it, how many sequences were submitted, tissue, histology, other keywords, etc.) is a custom import from NCBI’s dbEST database. Most of this information is also available to the public through an HTTP request at NCBI’s UniGene pages, e.g.:

<http://www.ncbi.nlm.nih.gov/UniGene/lib.cgi?ORG=Hs&LID=289>

Assignment of the individual ESTs to genes is obtained from the standard UniGene dump. CGAP's SAGE data are derived from a collaboration between NCI and Duke University and are based on new algorithms for mapping sequences to tags [13].

The caBIO *Gene* object provides access to these expression objects via its overloaded *getExpression()* method. With no arguments, this method returns an array of all SAGE and EST expression experiments for the gene. If a *type* argument ("SAGE" or "EST") is provided, then only the experiments of that type are returned. Finally, it is also possible to specify the particular *Organ* and *Disease* of interest.

CGAP also provides access to lists of sequence-verified human and mouse cDNA IMAGE clones supplied by [Invitrogen](#). Starting with a caBIO *Gene* object, you can get the list of *Clones* encoding that gene via the *getSequenceVerifiedClones()* method. From the *Clone* object, one can retrieve the *Library* object that contains that clone using *getLibrary()*. Specific information about the library can then be extracted using the methods *getCloneProducer()*, *getCloneVector()*, *getDescription()*, *getKeyword()*, *getLabHost()*, etc.

- The CGAP Genetic Annotation Initiative [14] ([GAI](#)). GAI is an NCI research program to explore and apply technology for identification and characterization of genetic variation in genes important in cancer. The GAI utilizes data-mining to identify "candidate" variation sites from publicly available DNA sequences, as well as laboratory methods to search for variations in cancer-related genes. All GAI candidate, validated, and confirmed genetic variants are available directly from the GAI web site, and all validated SNPs have been submitted to the NCBI dbSNP database as well.

SNPs identified by the GAI project can be accessed using caBIO *SNP* objects. The *SNP* object provides access to the *Clones* in which the SNP was observed via the *getClones()* method. The offset of this SNP in the parent sequence is available from the *getOffset()* method. The two most common base substitutions occurring at the site are extracted using *getBase1()* and *getBase2()*. The *getScore()* method returns the confidence score for the predicted SNP, and *getTracefiles()* provides access to the trace files used to identify the site as an SNP. The sequencing trace files used by GAI are imported from [Washington University](#).

- The NCI Cancer Therapy Evaluation Program [15] ([CTEP](#)). CTEP funds an extensive national program of basic and clinical research to evaluate new anti-cancer agents, with a particular emphasis on translational research to elucidate molecular targets and drug mechanisms. In response to this emergent need for translational research, there has been a groundswell of translational support tools defining controlled vocabularies and registered terminologies so as to enhance electronic data exchange in areas that have heretofore been relatively non-computational. The caBIO trials data are updated with new CTEP data on a quarterly basis, and many of the objects in Figure 15-2 are designed to support translational research.

For example, a caBIO *Target* object represents a molecule of special diagnostic or therapeutic interest for cancer research, and an *Anomaly* object is an observed deviation in the structure or expression of a *Target*. An *Agent* is a drug or other intervention that is effective in the presence of one or more specific *Targets*. The *ClinicalTrialProtocol* object

organizes administrative information pertaining to that protocol and has a *getAgents()* method for programmatic access to the specific therapies deployed.

- NCI's Cancer Molecular Analysis Project ([CMAP](#)) [16]. The CMAP web site is powered by caBIO, and makes extensive use of the objects in both Figures 15-1 and 15-2. The goal of CMAP is to enable researchers to identify and evaluate molecular targets in cancer. Towards this goal, CMAP provides four interfaces.

The CMAP *Profile Query* tool finds genes with the highest or lowest expression levels (using SAGE and microarray data) for a given tissue and histology. Selecting a gene from the resulting table then leads to a *Gene Info* page, providing information about cytogenetic location, chromosome aberrations, protein similarities, curated and computed orthologs, and sequence-verified as well as full-length MGC clones, along with links to various other databases. The CMAP ontology can be accessed through the caBIO *CMAPOntology* object.

CMAP's *Molecular Targets* interface organizes collections of genes by pathways and by ontology. Two ontologies are available: (1) the GO ontology described above, and (2) the CMAP ontology described here. The CMAP ontology relates functional classifications to molecular targets and agents. For example, selecting "angiogenesis" as the functional term brings up KDR, a type III receptor tyrosine kinase, and a list of agents for KDR. Selecting the target then produces a *Gene Info* page, as described above.

CMAP's *AgentSearch* tool allows the researcher to search for drug therapies by name (with wildcard matching), with the option of restricting the search to agents that are either associated with a term in the CMAPOntology or registered with a CTEP protocol. If the agent is associated with CTEP protocols, a table is presented on the *Agent Info* page, listing the title of each protocol and a link to its associated documentation. Selecting an entry from this table in turn leads to the *Therapeutic Trials Info* page for that CTEP protocol.

With the exception of the Mitelman Chromosome Aberration data, all of the information available through CGAP is also accessible programmatically through the caBIO objects in Figures 15-1 and 15-2.

- The NCI Enterprise Vocabulary Services [17, 18] ([EVS](#)). The EVS provides NCI with services and resources for controlled biomedical vocabularies, and includes both the NCI Thesaurus and the NCI Metathesaurus. The Thesaurus is composed of over 27,000 concepts represented by about 78,000 terms. The Thesaurus is organized into 18 hierarchical trees covering areas such as Neoplasms, Drugs, Anatomy, Genes, Proteins, and Techniques. These terms are deployed by NCI in its automated systems for uses such as keywording and database coding.

The NCI Metathesaurus maps terms from one standard vocabulary to another, facilitating collaboration, data sharing, and data pooling for clinical trials and scientific databases. The Metathesaurus is based on the NLM's Unified Medical Language System (UMLS) and is composed of over 70 biomedical vocabularies.

- The NCI Cancer Data Standards Repository [18] ([caDSR](#)). The Cancer Data Standards Repository is part of a larger effort associated with the ISO/IEC 11179 standard, whose purpose is to regularize the vocabularies used in representing and annotating shared electronic data. The caDSR at NCI is currently being used by a number of clinical trials



research centers for the standardization of case report forms and data collection terminologies. These groups include:

- The Cancer Therapy Evaluation Project ([CTEP](#))
  - Specialized Programs of Research Excellence ([SPOREs](#))
  - The Early Detection Research Network ([EDRN](#))
  - The Division of Cancer Prevention ([DCP](#))
  - The Cancer Imaging Program ([CIP](#))
  - The Division of Cancer Epidemiology and Genetics ([DECG](#))
- 
- The NCI Gene Expression Data Portal ([GEDP](#)). The Gene Expression Data Portal is designed to serve the microarray community as both a public source of microarray research as well as an online resource for data annotation and analysis tools. The massive amount of microarray data being generated today presents a significant challenge for analysis, storage, and exchange of data. The GEDP was developed to address this problem, and is part of the NCICB's cancer array informatics project (caARRAY). GEDP also functions as a depot for the exchange of pre- and post-publication data. The GEDP database is fully compliant with the Minimum Information About a Microarray Experiment (MIAME) specifications and was developed using the MicroArray and GeneExpression Object Model (MAGE-OM).
  - The NCI Cancer Models Database [18] ([caMOD](#)). The NCI Cancer Models Database is provided as a public service to the scientific community to foster the rapid dissemination of information concerning animal models of human cancer. The database currently contains murine models contributed from the Mouse Models of Human Cancers Consortium ([MMHCC](#)) Repository, from the [Jackson Laboratory](#), and directly from principle investigators in the larger research community. The flexible design of the database can accommodate models from a wide range of species—not just murine models—and it is anticipated that such models will be added in the future.

## 15.2 References

1. Schuler (1997). Pieces of the puzzle: expressed sequence tags and the catalog of human genes. *J Mol Med* 75(10):694–8.
2. Schuler et al. (1996). A gene map of the human genome. *Science* 274: 540–6.
3. Boguski & Schuler (1995). ESTablishing a human transcript map. *Nature Genetics* 10: 369–71.
4. Pruitt KD, and Maglott DR (2001). RefSeq and LocusLink: NCBI gene-centered resources. *Nucleic Acids Res* 29(1):137–40.
5. Pruitt KD, Katz KS, Sicotte H, and Maglott DR (2000). Introducing RefSeq and LocusLink: curated human genome resources at the NCBI. *Trends Genet* 16(1):44–7.
6. The Gene Ontology Consortium. (2000). Gene ontology: tool for the unification of biology. *Nature Genetics* 25:25–9.
7. The Gene Ontology Consortium. (2001). Creating the gene ontology resource: design and implementation. *Genome Res* 11:1425–33.
8. Zhang, Schwartz, Wagner, and Miller (2000). A Greedy algorithm for aligning DNA sequences. *J Comp Biol* 7(1-2):203–14.
9. Dowell RD, Jokerst RM, Day A, Eddy SR, Stein L. The Distributed Annotation System. *BMC Bioinformatics* 2(1):7.
10. Strausberg RL (2001). The Cancer Genome Anatomy Project: new resources for reading the molecular signatures of cancer. *J Pathol* 195:31–40.
11. Strausberg RL, Buetow KH, Emmert-Buck M, and Klausner R. (2000). The Cancer Genome Anatomy Project: building an annotated gene index. *Trends Genet* 16:103–106.
12. Strausberg RL (1999). The Cancer Genome Anatomy Project: building a new information and technology platform for cancer research. In: *Molecular Pathology of Early Cancer* (Srivastava S, Henson DE, Gazdar A, eds. IOS Press, 365–70.
13. Boon K, Osorio EC, Greenhut SF, Schaefer CF, Shoemaker J, Polyak K, Morin PJ, Buetow KH, Strausberg RL, De Souza SJ, and Riggins GJ (2002). An anatomy of normal and malignant gene expression. *Proc Natl Acad Sci U S A* 2002 Jul 15.
14. Clifford R, Edmonson M, Hu Y, Nguyen C, Scherpbier T, and Buetow KH (2000). Expression-based genetic/physical maps of single-nucleotide polymorphisms identified by the Cancer Genome Anatomy Project. *Genome Res* 10(8):1259–65.
15. Ansher SS and Scharf R (2001). The Cancer Therapy Evaluation Program (CTEP) at the National Cancer Institute: industry collaborations in new agent development. *Ann N Y Acad Sci* 949:333–40.

16. Buetow KH, Klausner RD, Fine H, Kaplan R, Singer DS, and Strausberg RL (2002). Cancer Molecular Analysis Project: Weaving a rich cancer research tapestry. *Cancer Cell* 1(4):315–8.
17. Hartel FW and de Coronado S (2002). Information standards within NCI. In: *Cancer Informatics: Essential Technologies for Clinical Trials*. Silva JS, Ball MJ, Chute CG, Douglas JV, Langlotz C, Niland J and Scherlis W, eds. Springer-Verlag.
18. Covitz PA., Hartel F, Schaefer C, De Coronado S, Fragoso G, Sahni H, Gustafson S, and Buetow KH. caCORE: A common infrastructure for cancer informatics. *Bioinformatics*, in press.

## Appendix A: The GeneDemo Program

```
import gov.nih.nci.caBIO.bean.*;
import gov.nih.nci.caBIO.util.*;
import java.util.*;

public class GeneDemo {

    public static void main (String args[])
    {
        System.out.println("Running the main of GeneDemo");
        try {

            Gene myGene = new Gene();
            GeneSearchCriteria criteria = new GeneSearchCriteria();
            criteria.setSymbol("pTEN");
            SearchResult result = myGene.search(criteria);
            if (result != null){
                Gene[] genes = (Gene[]) result.getResultSet();
                for(int i = 0; i < genes.length; i++){
                    System.out.println("\nInformation regarding Gene "+genes[i].getName());
                    System.out.println("\tTitle: "+genes[i].getTitle());
                    System.out.println("\tOMIM Id: "+genes[i].getOMIMId());
                    System.out.println("\tUnigene Cluster Id: "+genes[i].getClusterId());
                    System.out.println("\tLocusLink Id: "+genes[i].getLocusLinkId());
                    System.out.println("\tOrganism Abbreviation: "+genes[i].getOrganismAbbreviation());

                    Sequence[] refSeqs = genes[i].getReferenceSequences();
                    if( refSeqs.length > 0 ){
                        System.out.println("\nAssociated Reference Sequence(s) : ");
                        for(int k = 0; k < refSeqs.length; k++){
                            System.out.println("\n\tSequence Id:"+refSeqs[k].getId());
                            System.out.println("\tSequence Acc #:"+refSeqs[k].getAccessionNumber());
                            System.out.println("\tSequence Type:"+refSeqs[k].getType());
                            System.out.println("\tSequence Ascii String:"+refSeqs[k].getAsciiString());
                        }
                    }
                    StringBuffer buf = new StringBuffer();
                    ExpressionFeature feature = genes[i].getExpressionFeature();
                    if(feature != null){
                        Organ[] organs = feature.getExpressedInOrgans();
                        if(organs.length > 0){
                            System.out.println("\nOrgans ExpressionFeature In :");
                            buf.append("\t");
                            for(int j =0; j < organs.length ; j++){
                                organs[j].getName();
                                buf.append(organs[j].getName()+", ");
                            }
                            System.out.println(buf.toString());
                        }
                    }

                    GeneAlias[] geneAliases = genes[i].getGeneAliases();
                    if(geneAliases.length > 0){
                        System.out.println("\nAliase for "+genes[i].getName());
                        for(int j =0; j < geneAliases.length ; j++){
                            System.out.println("\t Type: "+geneAliases[j].getType());
                            System.out.println("\t Name: "+geneAliases[j].getName());
                            System.out.println("\t Description: "+geneAliases[j].getDescription());
                        }
                    }

                    Hashtable hash = genes[i].getDbCrossRefs();
                    if(hash != null){
                        Enumeration dbCrossRefs = hash.keys();
                        System.out.println("\nCross References for "+genes[i].getName());
                        while(dbCrossRefs.hasMoreElements()){
                            String key = (String)dbCrossRefs.nextElement();
                            String value = (String) hash.get(key);
                            System.out.println("\t"+key+" :"+value);
                        }
                    }

                    Taxon taxon = genes[i].getTaxon();
                    System.out.println("\nTaxon: "+taxon.getScientificName()+ " (Scientific Name)");
                    System.out.println("Taxon: "+taxon.getCommonName()+ " (Common Name)");
                }
            }
        }
    }
}
```

```

Chromosome chromosome = genes[i].getChromosome();
System.out.println("\n"+genes[i].getName() +
    " is on Chromosome: "+chromosome.getName());

Clone[] clones = genes[i].getSequenceVerifiedClones();
if(clones.length > 0){
System.out.println("\nAssociated Sequence Verified Clones : ");
for(int k = 0; k < clones.length; k++){
    System.out.println("\tClone Name: "+clones[k].getName());
    System.out.println("\tClone Version: "+clones[k].getVersion());
    System.out.println("\tClone Accession No: "+clones[k].getAccessionNumber());
    System.out.println("\tClone Strain: "+clones[k].getCloneStrain());
    System.out.println("\tCloning Site: "+clones[k].getCloningSite());
    System.out.println("\tClone Insert Size: "+clones[k].getInsertSize());
    System.out.println("\tCloning Site: "+clones[k].getCloningSite());

    Library library = clones[k].getLibrary();
    if(library != null){
        System.out.println("\n\t\tAssociated Library: ");
        System.out.println("\t\tLibrary Id: "+library.getId());
        System.out.println("\t\tLibrary Name: "+library.getName());
        System.out.println("\t\tLibrary Description: "+library.getDescription());
        System.out.println("\t\tLibrary Host: "+library.getLabHost());
        System.out.println("\t\tLibrary Creation Date: "+library.getCreationDate());
        Protocol protocol = library.getProtocol();
        if(protocol != null){
            System.out.println("\t\t\tAssociated Protocol: ");
            System.out.println("\t\t\tProtocol Id: "+protocol.getId());
            System.out.println("\t\t\tProtocol Name: "+protocol.getName());
            System.out.println("\t\t\tProtocol Description: "+protocol.getDescription());
            System.out.println("\t\t\tProtocol Type: "+protocol.getType());
        }
    }
}
}

Pathway[] pathways = genes[i].getPathways();
if(pathways.length > 0){
    System.out.println("\nAssociated Pathway :");
    for(int k = 0; k < pathways.length; k++){
        System.out.println("\t"+pathways[k].getName());
    }
}

Protein[] proteins = genes[i].getProteins();
if(proteins.length > 0 ){
System.out.println("\nAssociated Protein :");
for(int k = 0; k < proteins.length; k++){
    ProteinHomolog[] pHomologs = proteins[k].getProteinHomologs();
    if(pHomologs.length > 0) {
        System.out.println("\nAssociated Protein Homologs:");
        for(int l = 0; l < pHomologs.length; l++){
            Taxon homologTaxon = pHomologs[l].getTaxon();
            System.out.println("\tProtein Homolog Taxon:"+
                homologTaxon.getScientificName());
            System.out.println("\tProtein Homolog Alignment Length:"+
                pHomologs[l].getAlignmentLength());
            System.out.println("\tProtein Homolog Percentage Similarity:"+
                pHomologs[l].getSimilarityPercentage()+"%");
        }
    }
}
}

GeneHomolog[] homologs = genes[i].getGeneHomologs();
if(homologs.length > 0){
    System.out.println("\nGene Homolog :");
    for(int k = 0; k < homologs.length; k++){
        System.out.println("\tGene Homolog Name:"+homologs[k].getName());
        Taxon homologTaxon = homologs[k].getTaxon();
        System.out.println("\tGene Homolog Taxon:"+homologTaxon.getScientificName());
        System.out.println("\tGene Homolog Percentage Similarity:"+
            homologs[k].getSimilarityPercentage()+"%");
    }
}
}

```

```

GoOntology[] goOntologies = genes[i].getGoOntologies();
if (goOntologies.length > 0){
    System.out.println("\nRelated Gene Ontology : ");
    for(int k = 0; k < goOntologies.length; k++){
        System.out.println("\t"+goOntologies[k].getName());
    }
}

System.out.println("\n=====");
}
} catch (Exception exc) {
    System.out.println("Test failed in the main of GeneDemo.java: " + exc.getMessage());
    exc.printStackTrace();
}
}
}

```

## Sample Output from GeneDemo:

Running the main of GeneDemo

Information regarding Gene PTEN

Title: phosphatase and tensin homolog (mutated in multiple advanced cancers 1)  
OMIM Id: 601728  
Unigene Cluster Id: 10712  
LocusLink Id: 5728  
Organism Abbreviation: Hs

Organs ExpressionFeature In :

bone, brain, cervix, colon, ear, endocrine, eye, gastrointestinal tract, genitourinary, germ cell, head and neck, heart, kidney, liver, lung, lymph node, mammary gland, muscle, nervous, ovary, pancreas, pancreatic islet, parathyroid, pituitary gland, placenta, prostate, retina, salivary gland, skin, spleen, stomach, testis, thyroid, uncharacterized tissue, uterus, vascular, whole blood, b-cell, fetus, pooled tissue, whole body,

Aliase for PTEN

Type: BioCarta  
Name: pten  
Description: null  
Type: CGAP  
Name: PTEN  
Description: null

Cross References for PTEN

LOCUS LINK :5728  
OMIM :601728  
UNIGENE :10712

Taxon: Homo sapiens (Scientific Name)

Taxon: null (Common Name)

PTEN is on Chromosome: 10

Associated Sequence Verified Clones :

Clone Name: IMAGE:1535806  
Clone Version: null  
Clone Accession No: AA936678  
Clone Strain: null  
Cloning Site: null  
Clone Insert Size: null  
Cloning Site: null

Associated Library:

Library Id: 49  
Library Name: NCI\_CGAP\_Kid3  
Library Description: 1st strand cDNA was primed with a Not I - oligo(dT) primer, double-stranded cDNA was ligated to Eco RI adaptors (Pharmacia), digested with Not I and cloned into the Not I and Eco RI sites of the modified pT7T3 vector. mRNA source: 2 pooled kidneys. Library went through one round of normalization. Library constructed by Bento Soares and M. Fatima Bonaldo.  
Library Host: DH10B  
Library Creation Date: null  
Associated Protocol:  
Protocol Id: 8  
Protocol Name: normalized  
Protocol Description: null  
Protocol Type: LIBRARY

Associated Pathway :

EIF4Pathway  
PTENPathway  
MTORPathway

Associated Protein :

Associated Protein Homologs:

Protein Homolog Taxon:Arabidopsis thaliana  
Protein Homolog Alignment Length:177  
Protein Homolog Percentage Similarity:51%  
Protein Homolog Taxon:Caenorhabditis elegans  
Protein Homolog Alignment Length:234  
Protein Homolog Percentage Similarity:40%

Protein Homolog Taxon:Drosophila melanogaster  
Protein Homolog Alignment Length:347  
Protein Homolog Percentage Similarity:44%  
Protein Homolog Taxon:Homo sapiens  
Protein Homolog Alignment Length:403  
Protein Homolog Percentage Similarity:100%  
Protein Homolog Taxon:Mus musculus  
Protein Homolog Alignment Length:403  
Protein Homolog Percentage Similarity:99%  
Protein Homolog Taxon:Rattus norvegicus  
Protein Homolog Alignment Length:403  
Protein Homolog Percentage Similarity:99%  
Protein Homolog Taxon:Sacharomyces cerevisiae  
Protein Homolog Alignment Length:179  
Protein Homolog Percentage Similarity:27%

Gene Homolog :  
Gene Homolog Name:Pten  
Gene Homolog Taxon:Mus musculus  
Gene Homolog Percentage Similarity:91.98%

Related Gene Ontology :  
regulation of cell cycle  
regulation of CDK activity  
inositol/phosphatidylinositol phosphatase activity  
protein phosphatase activity  
protein tyrosine phosphatase activity  
cytoplasm  
protein amino acid dephosphorylation  
development  
cell proliferation  
phosphatidylinositol-3,4,5-trisphosphate 3-phosphatase activity  
hydrolase activity  
negative regulation of cell cycle

=====

Information regarding Gene PTEN  
Title: phosphatase and tensin homolog (mutated in multiple advanced cancers 1)  
OMIM Id: 601728  
Unigene Cluster Id: 356062  
LocusLink Id: 5728  
Organism Abbreviation: Hs

Organs ExpressionFeature In :  
brain, colon, endocrine, gastrointestinal tract, genitourinary, germ cell, lung, nervous,  
pineal gland, prostate, testis, uncharacterized tissue, whole blood, b-cell, fetus, pooled  
tissue,

Aliase for PTEN  
Type: BioCarta  
Name: pten  
Description: null  
Type: CGAP  
Name: PTEN  
Description: null

*[additional pages of output omitted]*



## Appendix B: The EVSDemo Program

```
import gov.nih.nci.EVS.bean.*;
import gov.nih.nci.EVS.search.*;
import gov.nih.nci.EVS.exception.*;
import gov.nih.nci.common.util.*;
import gov.nih.nci.common.exception.*;
import java.util.*;

public class EVSDemo {

    public static void main (String args[])
    {
        System.out.println("Running the NCI Thesaurus ");
        String termStr = "";

        try
        {
            DescLogicConceptSearchCriteria dlcsc = new DescLogicConceptSearchCriteria();
            DescLogicConcept dlc = new DescLogicConcept();
            MetaThesaurusConceptSearchCriteria mtcsc = new MetaThesaurusConceptSearchCriteria();
            MetaThesaurusConcept mtc = new MetaThesaurusConcept();
            String vocabularyName= "NCI_Thesaurus";
            String conceptname = "Gene";

            //-----
            //  NCI Thesaurus
            //  Search()
            //-----

            dlscsc.setLimit(10000);
            dlscsc.setSearchTerm("Gen*");
            dlscsc.setVocabularyName("NCI_Thesaurus");
            Concept[] conceptArray = dlc.search(dlscsc);

            for(int i=0; i<conceptArray.length; i++)
                System.out.println(conceptArray[i].getName());

            //-----
            //  NCI Thesaurus
            //  getConceptByName()
            //-----
            dlscsc.setSearchTerm("Gene");
            dlscsc.setVocabularyName("NCI_Thesaurus");
            Concept concept = dlc.getConceptByName(dlscsc);
            conceptArray = new Concept[1];
            conceptArray[0] = concept;
            printConceptArray (conceptArray);

            //-----
            //  NCI Thesaurus
            //  getProperties()
            //-----
            Property[] propertyArray = dlc.getPropertiesByConceptName("NCI_Thesaurus", "Gene");

            for(int i=0; i<propertyArray.length; i++)
                System.out.println(propertyArray[i].getName() +
                    " ---> "+propertyArray[i].getValue()+"\n");

            //-----
            //  NCI Thesaurus
            //  getRoles()
            //-----
            Role[] roleArray = dlc.getRolesByConceptName("NCI_Thesaurus", "Oncogene MYC");

            for(int i=0; i<roleArray.length; i++)
                System.out.println(roleArray[i].getName() +
                    " ---> "+roleArray[i].getValue()+"\n");

            //-----
            //      NCI Thesaurus
            //      Ancestor
            //-----

            /**
```

```

* Gets the ancestor concepts' codes of the specified concept.
* The search extends to a final baseline date starting from an initial baseline date.
* The search is based on a boolean value. If the value is true, the method only
* searches for the active concepts at the initial baseline date.
* If the value is false, the method searches for all ancestor concepts,
* whether active or retired. The search is only applied to "split/merge"
* actions and the specified concept could be the ancestor concept of itself.
*
* @param vocabularyName. The specified vocabulary name.
* @param code1. The specified concept code.
* @param atBaseline The boolean value to specify the search type.
* @param date1. The initial baseline date.
* @param date2. The final baseline date.
* @return a vector of concept codes belonging to the ancestor concepts of
*         inCode.
*/

boolean atBaseline=false;
String code1="C21434";
String code2 = "";
String date1="5/7/2003";
String date2="12/12/2003";

String[] ancestorCodes =
    new History().getAncestors(vocabularyName, code1, atBaseline, date1, date2);

for (int i=0; i<ancestorCodes.length; i++)
    System.out.println("\t"+ancestorCodes[i]);

//-----
//      NCI Thesaurus
//      Descendants
//-----

/**
* Gets the descendant concepts' codes of the specified concept.
* The search extends to a final baseline date starting from an initial baseline date.
* The search is based on a boolean value. If the value is true, the method only
* searches for the active concepts at the initial baseline date.
* If the value is false, the method searches for all descendant concepts,
* whether active or retired. The search is only applied to "split/merge"
* actions and the specified concept could be the descendant concept of itself.
*
* @param vocabularyName. The specified vocabulary name.
* @param code1. The specified concept code.
* @param atBaseline The boolean value to specify the search type.
* @param date1. The initial baseline date.
* @param date2. The final baseline date.
* @return a vector of concept codes belonging to the descendant concepts of
*         inCode.
*/

atBaseline=false;
code1="C21434";
code2 = "";
date1="5/7/2003";
date2="12/12/2003";

String[] descCodes =
    new History().getDescendants(vocabularyName,code1, atBaseline, date1, date2);

for (int i=0; i<descCodes.length; i++)
    System.out.println("\t"+descCodes[i]);

// -----
//      NCI Thesaurus
//      getConceptEditAction
// -----

try {
String[] actions = new History(). getConceptEditAction(vocabularyName, code1);
for (int i=0; i<actions.length; i++)
    System.out.println("\t"+actions[i]);
}

```

```

    } catch(Exception e){
        e.printStackTrace();
    }

    //-----
    //      NCI Thesaurus
    //      isRetired
    //-----
    try {
        System.out.println( "Concept "+code1+" Retired ? :" +
            new DescLogicConcept().isRetired(vocabularyName, code1) );
    } catch(Exception e){
        e.printStackTrace();
    }

    //-----
    //      is Sub Concept
    //-----

    String name1 = "Autoantigen Gene";
    String name2 = "Paraneoplastic Disease Antigen Gene";
    System.out.println(name2+" is subconcept of "+name1+"? :" +
        new DescLogicConcept().isSubConcept(vocabularyName, name2, name1));

    //-----
    //      Get Sub Concept
    //-----
    // Subconcepts
    System.out.println("Sub Concept of Gene");
    String[] subConcepts =
        (new DescLogicConcept()).getSubConcepts(vocabularyName, conceptname,
            Boolean.FALSE, Boolean.FALSE);
    for (int i=0; i<subConcepts.length; i++)
        System.out.println("\t" + subConcepts[i]);

    //-----
    //      Get Super Concept
    //-----
    // Superconcepts
    conceptname = "Iris";
    System.out.println("Super Concept of Gene");
    String[] superConcepts =
        new DescLogicConcept().getSuperConcepts(vocabularyName, conceptname,
            Boolean.FALSE, Boolean.FALSE);
    for (int i=0; i<superConcepts.length; i++)
        System.out.println("\t" + superConcepts[i]);

    //-----
    //      Meta Thesaurus
    //      search
    //-----
    termStr = "lung";
    mtcsc.setSearchTerm(termStr);
    mtcsc.setLimit(2);
    printConceptArray(mtc.search(mtcsc));

    //-----
    //      MetaThesaurus
    //      Get Semantic Types
    //-----

    SemanticType[] semanticTypes = mtc.getSemanticTypes();
    if(semanticTypes.length>0) printSemanticArray(semanticTypes);

}

catch (Exception exc)
{
    System.out.println("Test failed in the main of EVSDemo.java: " + exc.getMessage());
    exc.printStackTrace();
}
} // main

```

```

public static void printConceptArray(Concept [] conceptArray)
{
    for(int i=0; i<conceptArray.length; i++)
    {
        try
        {
            Concept conceptObj = (Concept)conceptArray[i];
            System.out.println(conceptObj.getName());
            System.out.println(conceptObj.getConceptCode());

            //Source of the concept
            if(conceptObj.getSources() != null )
            {
                Source[] sourceArray = conceptObj.getSources();

                for(int j=0; j<sourceArray.length; j++)
                    System.out.println("Source Abbreviation ==>" +
                        sourceArray[j].getAbbreviation());
            }
            //CUI UMLS & TEMP
            if(conceptObj.getConceptUniqueIdentifier() != null )
            {
                ConceptUniqueIdentifier conceptUID = conceptObj.getConceptUniqueIdentifier();
                //if UMLS is false then TEMP is true
                System.out.println(" Is it a UMLS ? :"+conceptUID.isUMLS());
                //Type of CUI
                System.out.println("CUI : " + conceptUID.getCUI() +
                    " of Type "+conceptUID.getType());
            }
            //Semantic Types
            if(conceptObj.getSemanticTypes() != null )
            {
                SemanticType[] semanticTypeArray = conceptObj.getSemanticTypes();

                if(semanticTypeArray.length>0) printSemanticArray(semanticTypeArray);
            }

            //Definitions
            if(conceptObj.getDefinitions() != null )
            {
                Definition[] definitionArray = conceptObj.getDefinitions();
                if(definitionArray.length>0)
                    for(int j=0; j<definitionArray.length; j++)
                    {
                        System.out.println("Definition \t" +
                            definitionArray[j].getDefinition()+"\n");
                        //Source of the Definition
                        System.out.println(" \t\tAbbreviation:" +
                            definitionArray[j].getSource().getAbbreviation()+" | ");
                        System.out.println(" \t\tDescription: " +
                            definitionArray[j].getSource().getDescription()+" \n");
                    }
            }
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}

public static void printSemanticArray(SemanticType[] semanticTypeArray) throws Exception
{
    for (int i = 0; i < semanticTypeArray.length; ++i)
    {
        System.out.println(semanticTypeArray[i]);
    }
}
}

```

## Sample Output from EVSDemo:

```
Running the NCI Thesaurus
Gender
Gender/Minority Enrollment
Gender/Minority Projection
Gene
Gene Alteration, Environmentally Produced
Gene Amplification
Gene Amplification Technique
Gene Amplification or Deletion Detection
Gene Bank
Gene Cloning
Gene
C16612
  Is it a UMLS ? :true
  CUI :C0017337 of Type UMLS_CUI
  SemanticType => Name: Gene or Genome,   ID:
  Definition    Specific sequences of nucleotides along a molecule of DNA (or, in the case of some
  viruses, RNA) which represent the functional units of heredity. The majority of eukaryotic genes
  contain coding regions (codons) that are interrupted by non-coding regions (introns) and are
  therefore labeled split genes.

      Abbreviation:null |
      Description: null

Definition    The functional and physical unit of heredity passed from parent to offspring. Genes
are pieces of DNA, and most genes contain the information for making a specific protein.

      Abbreviation:null |
      Description: null

Definition    A functional unit of heredity which occupies a specific position (locus) on a
particular chromosome, is capable of reproducing itself exactly at each cell division, and
directs the formation of an enzyme or other protein. The gene as a functional unit consists of a
discrete segment of a giant DNA molecule containing the purine (adenine and guanine) and
pyrimidine (cytosine and thymine) bases in the ordered and correct sequence that encodes a
specific functional product (i.e., a protein or RNA molecule). Protein synthesis is mediated by
molecules of messenger-RNA formed on the chromosome with the gene acting as template. The RNA
then passes into the cytoplasm and becomes oriented on the ribosomes where it in turn acts as
template to organize a chain of amino acids to form a peptide. Genes normally occur in pairs in
all cells except gametes, as a consequence of the fact that all chromosomes are paired except the
sex chromosomes (X and Y) of the male.

      Abbreviation:null |
      Description: null

Preferred_Name ---> Gene
Semantic_Type  ---> Gene or Genome
UMLS_CUI      ---> C0017337

DEFINITION ---> <def-source>MSH2001</def-source><def-definition>Specific sequences of
nucleotides along a molecule of DNA (or, in the case of some viruses, RNA) which represent the
functional units of heredity. The majority of eukaryotic genes contain coding regions (codons)
that are interrupted by non-coding regions (introns) and are therefore labeled split genes.</def-
definition>

DEFINITION ---> <def-source>NCI-GLOSS</def-source><def-definition>The functional and physical
unit of heredity passed from parent to offspring. Genes are pieces of DNA, and most genes contain
the information for making a specific protein.</def-definition>

DEFINITION ---> <def-source>NCI</def-source><def-definition>A functional unit of heredity which
occupies a specific position (locus) on a particular chromosome, is capable of reproducing itself
exactly at each cell division, and directs the formation of an enzyme or other protein. The gene
as a functional unit consists of a discrete segment of a giant DNA molecule containing the purine
(adenine and guanine) and pyrimidine (cytosine and thymine) bases in the ordered and correct
sequence that encodes a specific functional product (i.e., a protein or RNA molecule). Protein
synthesis is mediated by molecules of messenger-RNA formed on the chromosome with the gene acting
as template. The RNA then passes into the cytoplasm and becomes oriented on the ribosomes where
it in turn acts as template to organize a chain of amino acids to form a peptide. Genes normally
occur in pairs in all cells except gametes, as a consequence of the fact that all chromosomes are
paired except the sex chromosomes (X and Y) of the male.</def-definition>
```

FULL\_SYN ---> <term-name>Gene</term-name><term-group>PT</term-group><term-source>NCI</term-source>  
 FULL\_SYN ---> <term-name>Genes</term-name><term-group>SY</term-group><term-source>NCI</term-source>  
 FULL\_SYN ---> <term-name>gene</term-name><term-group>PT</term-group><term-source>NCI-GLOSS</term-source><source-code>CDR0000045693</source-code>  
 Synonym ---> Genes  
 Gene\_Associated\_With\_Disease ---> Burkitt's Lymphoma  
 Gene\_Found\_In\_Organism ---> Human  
 Gene\_Has\_Function ---> Transcription  
 Gene\_Has\_Function ---> Transcriptional Regulation  
 Gene\_Has\_Function ---> Tumorigenesis  
 Gene\_In\_Chromosomal\_Location ---> 8q24  
 modify|08/22/2003  
 Concept C21434 Retired ? :false  
 Paraneoplastic Disease Antigen Gene is subconcept of Autoantigen Gene? :true  
 Sub Concept of Gene  
   Apoptosis Regulation Gene  
   Cancer Gene  
   Candidate Disease Gene  
   Cell Cycle Gene  
   Chaperone Gene  
   Complement Component Gene  
   Cysteine Proteinase Inhibitor Gene  
   DNA Repair Gene  
   Enzyme Gene  
   Fusion Gene  
   Gene with Unknown or Unclassified Function  
   Growth Factor Gene  
   Housekeeping Gene  
   Immunoglobulin Family Gene  
   Immunoprotein Gene  
   Ion Channel Protein Gene  
   Ligand Binding Protein Gene  
   Major Histocompatibility Complex Gene  
   Membrane Protein Gene  
   Non-Human Gene  
   Peptide Hormone Gene  
   Protein Complex Subunit Gene  
   Regulatory Gene  
   Replication Initiation Gene  
   Reporter Gene  
   Signaling Pathway Gene  
   Structural Gene  
   Telomere Maintenance Gene  
   Trafficking Protein Gene  
   Transcription Factor Gene  
   Translation Process Gene  
   Unmodeled Gene  
 Super Concept of Gene  
   Eye Part  
 Lung  
 null  
 Source Abbreviation ==>AOD2000  
 Source Abbreviation ==>CSP2002  
 Source Abbreviation ==>ELC2001  
 Source Abbreviation ==>ICDO3  
 Source Abbreviation ==>LCH90  
 Source Abbreviation ==>LNC205

*[additional pages of output omitted]*

## Appendix C: The CaseReportFormDemo Program

```
import gov.nih.nci.caDSR.bean.*;
import gov.nih.nci.caDSR.util.*;
import gov.nih.nci.caDSR.search.*;
import gov.nih.nci.common.search.*;
import java.util.*;

public class CaseReportFormDemo {

    public static void printOutResults(SearchResult result) {
        CaseReportForm[] casereportforms = null;

        if (result != null) {
            try {
                casereportforms = (CaseReportForm[]) result.getResultSet();
                for(int i = 0; i < casereportforms.length; i++){

                    System.out.println("\n\tCaseReportForm: "+
                        casereportforms[i].getPreferredName());
                    System.out.println("\n\tId: "+casereportforms[i].getId());
                    System.out.println("\tPreferredDefinition:
                        "+casereportforms[i].getPreferredDefinition());
                    System.out.println("\tLong Name: "+casereportforms[i].getLongName());
                    System.out.println("\tVersion: "+casereportforms[i].getVersion());
                    System.out.println("\tWorkflowStatusName: "+
                        casereportforms[i].getWorkflowStatusName());

                    ReferenceDocument[] refDocs = casereportforms[i].getReferenceDocuments();
                    if( refDocs.length > 0 ){
                        System.out.println("\n\tAssociated Reference ReferenceDocument(s): ");
                        for(int k = 0; k < refDocs.length; k++){
                            System.out.println("\tName:"+refDocs[k].getName());
                            System.out.println("\tType:"+refDocs[k].getType());
                            System.out.println("\tDocText:"+refDocs[k].getDoctext());
                        }
                    }

                    Module[] mods = casereportforms[i].getModules();
                    if( mods.length > 0 ){
                        System.out.println("\n\tAssociated Module(s): ");
                        for(int k = 0; k < mods.length; k++){
                            System.out.println("\n\tId:"+mods[k].getId());
                            System.out.println("\tPreferredName:"+mods[k].getPreferredName());
                            System.out.println("\tPreferredDefinition: " +
                                mods[k].getPreferredDefinition());
                            System.out.println("\tLongName:"+mods[k].getLongName());
                        }
                    }

                    Designation[] desigs = casereportforms[i].getDesignations();
                    if( desigs.length > 0 ){
                        System.out.println("\n\tAssociated Designation(s): ");
                        for(int k = 0; k < desigs.length; k++){
                            System.out.println("\n\tId:"+desigs[k].getId());
                            System.out.println("\tName:"+desigs[k].getName());
                            System.out.println("\tType:"+desigs[k].getType());
                            System.out.println("\tLanguageName:"+desigs[k].getLanguageName());
                        }
                    }

                    ClassificationSchemeItem[] csItems =
                        casereportforms[i].getClassificationSchemeItems();
                    if( csItems.length > 0 ){
                        System.out.println("\n\tAssociated ClassificationSchemeItem(s): ");
                        for(int k = 0; k < csItems.length; k++){
                            System.out.println("\n\tId:"+csItems[k].getId());
                            System.out.println("\tName:"+csItems[k].getName());
                            System.out.println("\tType:"+csItems[k].getType());
                            System.out.println("\tDescription:"+csItems[k].getDescription());
                        }
                    }

                    ProtocolFormsSet myProtocolFormsSet =
```

```

        casereportforms[i].getProtocolFormsSet();
        System.out.println("\n\tAssociated ProtocolFormsSet: ");
        System.out.println("\tTitle: " + myProtocolFormsSet.getTitle());
        System.out.println("\tType: " + myProtocolFormsSet.getType());
        System.out.println("\tPhase: " + myProtocolFormsSet.getPhase());
        System.out.println("\n\tAssociated Context:");
        Context myContext = casereportforms[i].getContext();
        System.out.println("\tId:" + myContext.getId());
        System.out.println("\tName:" + myContext.getName());
        System.out.println("\tVersion:" + myContext.getVersion());
        System.out.println("\tDescription:" + myContext.getDescription());
        System.out.println("\tDateCreated:" + myContext.getDateCreated());
        System.out.println("\tDateModified:" + myContext.getDateModified());
        System.out.println(
            "\n=====");
    }
} catch (Exception ex) {
    System.out.println(ex.toString());
}
}

public static void main (String args[]) {

    // This demonstration program searches for CaseReportForm objects using the
    // CRF id as the search criterias. After finding a CaseReportForm the program then
    // displays some of the attributes and associated objects (and their attribtes).

    CaseReportForm myCaseReportForm1 = new CaseReportForm();
    CaseReportFormSearchCriteria criterial = new CaseReportFormSearchCriteria();
    SearchResult result1 = new SearchResult();
    try {
        System.out.println("*** Searching for CaseReportForms based on Id: " +
            "A73D1CA4-8ADF-4761-E034-0003BA0B1A09 **");
        criterial.setId("A73D1CA4-8ADF-4761-E034-0003BA0B1A09");
        result1 = myCaseReportForm1.search(criterial);
        printOutResults(result1);
    } catch (Exception ex) {
    }
}
}

```



## Sample Output from CaseReportFormDemo:

\*\* Searching for CaseReportForms based on Id: A73D1CA4-8ADF-4761-E034-0003BA0B1A09 \*\*

CaseReportForm: CALGB\_49903\_ADV\_TX\_SUMMARY\_FOR

Id: A73D1CA4-8ADF-4761-E034-0003BA0B1A09  
PreferredDefinition: CALGB: 49903 ADVANCED TREATMENT SUMMARY FORM; All Patients  
Long Name: CALGB: 49903 ADVANCED TREATMENT SUMMARY FORM; All Patients  
Version: 3.0  
WorkflowStatusName: APPRVD FOR TRIAL USE

Associated Module(s):

Id:A73D1CA4-8AE3-4761-E034-0003BA0B1A09  
PreferredName:CALGB\_49903\_ADV\_TX\_SUMM2038661  
PreferredDefinition:CCRR MODULE FOR CALGB: 49903 ADVANCED TREATMENT SUMMARY FORM; All  
Patients  
LongName:CCRR MODULE

Id:A73D1CA4-8B6A-4761-E034-0003BA0B1A09  
PreferredName:TREATMENT\_SCHEDULEOTH\_T2038705  
PreferredDefinition:Treatment Schedule - Other Therapy  
LongName:Treatment Schedule - Other Therapy

Id:A73D1CA4-8B9D-4761-E034-0003BA0B1A09  
PreferredName:COMMENTS2038722  
PreferredDefinition:Comments  
LongName:Comments

Id:A73D1CA4-8B4F-4761-E034-0003BA0B1A09  
PreferredName:TREATMENT\_SCHEDULESYSIC2038696  
PreferredDefinition:Treatment Schedule - Systemic Therapy  
LongName:Treatment Schedule - Systemic Therapy

Id:A73D1CA4-8AE9-4761-E034-0003BA0B1A09  
PreferredName:UNNAMED12038663  
PreferredDefinition:Unnamed1  
LongName:Unnamed1

Id:A73D1CA4-8B0A-4761-E034-0003BA0B1A09  
PreferredName:UNNAMED22038674  
PreferredDefinition:Unnamed2  
LongName:Unnamed2

Id:A73D1CA4-8B1F-4761-E034-0003BA0B1A09  
PreferredName:TREATMENT\_CYCLE\_INFORMA2038681  
PreferredDefinition:Treatment Cycle Information  
LongName:Treatment Cycle Information

Id:A73D1CA4-8BA3-4761-E034-0003BA0B1A09  
PreferredName:UNNAMED52038724  
PreferredDefinition:Unnamed5  
LongName:Unnamed5

Associated Designation(s):

Id:CA69A1C9-2495-4B35-E034-0003BA0B1A09  
Name:2547042  
Type:QC\_ID  
LanguageName:ENGLISH

Associated ProtocolFormsSet:

Title: null  
Type: Treatment trials  
Phase: 3

Associated Context:

Id:99BA9DC8-2095-4E69-E034-080020C9C0E0  
Name:CTEP  
Version:2.31  
Description:NCI Cancer Therapy Evaluation Program

DateCreated:Wed Feb 13 00:00:00 EST 2002  
DateModified:Wed Feb 13 00:00:00 EST 2002

=====

## Appendix D: The CancerModelDemo Program

```
import java.io.*;
import java.util.*;
import java.net.*;
import org.w3c.dom.*;
import gov.nih.nci.caMOD.bean.*;
import gov.nih.nci.caMOD.search.*;
import gov.nih.nci.common.exception.*;
import gov.nih.nci.common.search.*;
import gov.nih.nci.common.util.*;

public class CancerModelDemo
{
    // *****
    //          ATTRIBUTES
    // *****
    public MicroArrayData[] myMicroArrays = null;
    public Publication[] myPubs = null;
    public TreatmentSchedule[] mySchedules = null;
    public Phenotype[] myPhenotypes = null;

    public SearchResult microArrayResults = null;
    public SearchResult phenotypeResults = null;
    public SearchResult pubResults = null;
    public SearchResult tsResults = null;

    public CancerModelDemo() {}

    private void runDemo()
    {
        try {
            // =====
            //          TreatmentSchedule Test
            // =====
            MessageLog.printlnInfo("\n<==== TESTING TREATMENT SCHEDULE =====>\n");

            // Instantiate a TreatmentScheduleSearchCriteria
            // and set the regimen value we are using to select treatment schedules.
            TreatmentScheduleSearchCriteria tsCriteria = new TreatmentScheduleSearchCriteria();
            tsCriteria.setRegimen("weekly for 6 weeks");

            // Instantiate a TreatmentSchedule
            TreatmentSchedule ts = new TreatmentSchedule();

            // Perform the search on the TreatmentSchedules
            tsResults = ts.search(tsCriteria);

            // Capture the results and loop through the results, casting each returned object
            // to it's expected type...TreatmentSchedule.
            Object[] theSchedules = tsResults.getResultSet();
            MessageLog.printlnInfo("\nAPI_TEST_CLIENT(search)::TreatmentSchedule Test Results");
            for ( int i=0; i < theSchedules.length; i++ )
            {
                MessageLog.printlnInfo("\nTreatment Schedule => " +
                    ((TreatmentSchedule) theSchedules[i]).toString());
            }

            // =====
            //          Publication Test
            // =====
            MessageLog.printlnInfo("\n<==== TESTING PUBLICATION =====>\n");

            // Instantiate a PublicationSearchCriteria
            // Here, we set the criteria to look for a publication with an id of 1690.
            PublicationSearchCriteria pubCriteria = new PublicationSearchCriteria();
            pubCriteria.setId(new Long(1690));
            String[] pubObjs = {"PublicationStatus"};

            // Instantiate a Publication
            Publication pub = new Publication();

            // Perform the search on the Publications

```

```

pubResults = pub.search(pubCriteria);

// Capture the results and loop through the results, casting each returned object
// to it's expected type...Publication.
Object[] thePublications = pubResults.getResultSet();
MessageLog.printlnInfo("\nAPI_TEST_CLIENT(search)::Publication Test Results");
for ( int i=0; i < thePublications.length; i++ )
{
    MessageLog.printlnInfo("\nPublication => " +
        ((Publication) (thePublications[i])).toString());
}

// =====
//                               MicroArrayData Test
// =====
MessageLog.printlnInfo("\n<==== TESTING MICROARRAYDATA ====>\n");

// Instantiate a TreatmentScheduleSearchCriteria
// Here, we set the criteria to look for micro array data with an id of 1.
//
MicroArrayDataSearchCriteria microArrayDataCriteria = new MicroArrayDataSearchCriteria();
microArrayDataCriteria.setId(new Long(1));

// Instantiate a MicroArrayData
MicroArrayData microArrayData = new MicroArrayData();

// Perform the search on the MicroArrayData
microArrayResults = microArrayData.search(microArrayDataCriteria);

// Capture the results and loop through the results, casting each returned object
// to it's expected type...MicroArrayData.
Object[] theArrayData = microArrayResults.getResultSet();
MessageLog.printlnInfo("\nAPI_TEST_CLIENT(search)::MicroArrayData Test Results");
for ( int i=0; i < theArrayData.length; i++ )
{
    MessageLog.printlnInfo("\nMicroArrayData => " +
        ((MicroArrayData) (theArrayData[i])).toString());
}

// =====
//                               Phenotype Test
// =====
MessageLog.printlnInfo("\n<==== TESTING PHENOTYPE ====>\n");
// Instantiate a PhenotypeSearchCriteria
// Here, we set the criteria to look for all Phenotypes where the breeding notes = "none".
//
PhenotypeSearchCriteria phenotypeCriteria = new PhenotypeSearchCriteria();
phenotypeCriteria.setBreedingNotes("none");

// Instantiate a Phenotype
Phenotype phenotype = new Phenotype();

// Perform the search on the Phenotypes
phenotypeResults = phenotype.search(phenotypeCriteria);

// Capture the results and loop through the results, casting each returned object
// to it's expected type...Phenotype.
Object[] thePhenotypes = phenotypeResults.getResultSet();
MessageLog.printlnInfo("\nAPI_TEST_CLIENT(search)::Phenotype Test Results");
for ( int i=0; i < thePhenotypes.length; i++ )
{
    MessageLog.printlnInfo("\nPhenotype => " + ((Phenotype) (thePhenotypes[i])).toString());
}
} catch (Exception e) {
}
}

public static void main(String args[])
{
    MessageLog.printlnError("Testing the caMOD API ...");
    CancerModelDemo apiTest = new CancerModelDemo();
    apiTest.runDemo();
}
}

```

## Sample Output from CancerModelDemo:

<==== TESTING TREATMENT SCHEDULE =====>

API\_TEST\_CLIENT(search)::TreatmentSchedule Test Results

Treatment Schedule => TreatmentSchedule id: 6  
TreatmentSchedule dosage: 100 mg  
TreatmentSchedule regimen: weekly for 6 weeks

Treatment Schedule => TreatmentSchedule id: 7  
TreatmentSchedule dosage: 100 mg/kg  
TreatmentSchedule regimen: weekly for 6 weeks

<==== TESTING PUBLICATION =====>

API\_TEST\_CLIENT(search)::Publication Test Results

Publication => Publication id: 1690  
Publication journal: Toxicol Pathol  
Publication volume: ;29 Suppl  
Publication pmid: 11695547  
Publication startPage: 117  
Publication year: 2001  
Publication authors: van Kreijl, CF  
Publication title: Xpa and Xpa/p53+/- knockout mice: overview of available data  
Publication endPage: 127  
status: PublicationStatus id: 1  
PublicationStatus publicationStatusName: Published

<==== TESTING MICROARRAYDATA =====>

API\_TEST\_CLIENT(search)::MicroArrayData Test Results

MicroArrayData => MicroArrayData id: 1  
MicroArrayData experimentName: Initiating oncogenic event determines gene-expression patterns of human breast cancer models  
MicroArrayData experimentID: 160  
availability: null

<==== TESTING PHENOTYPE =====>

API\_TEST\_CLIENT(search)::Phenotype Test Results

Phenotype => Phenotype id: 108  
Phenotype desc: Blocked for tumor promoter-induced activation of AP-1 transcription factor and for skin tumor promotion (papillomagenesis that leads to carcinoma formation). Not blocked to tumor promoter induced hyperplasia.  
Phenotype breedingNotes: none  
sexDistribution: null

Phenotype => Phenotype id: 109  
Phenotype desc: No phenotype  
Phenotype breedingNotes: none  
sexDistribution: null

Phenotype => Phenotype id: 96  
Phenotype desc: Cell proliferation, but not cell survival, is increased in Pten+/-/Cdkn1b-/- mice. Moreover, Pten+/-/Cdkn1b-/- mice develop prostate carcinoma at complete penetrance within three months from birth.  
Phenotype breedingNotes: none  
sexDistribution: SexDistribution id: 3  
SexDistribution sexDistributionTypeName: Male Only

Phenotype => Phenotype id: 19  
Phenotype desc: Pten +/Δ mice, despite their young age (Pten+/Δ mice, <3.5 months), spontaneously develop malignant tumours of various histological origins.

Moreover, Pten haploinsufficiency results in a lethal autoimmune disorder and Pten+/- mice are impaired in Fas-mediated apoptosis.  
Phenotype breedingNotes: none  
sexDistribution: SexDistribution id: 1  
SexDistribution sexDistributionTypeName: Both Sexes

Phenotype => Phenotype id: 20  
Phenotype desc: Pten inactivation resulted in early embryonic lethality.  
Phenotype breedingNotes: none  
sexDistribution: SexDistribution id: 1  
SexDistribution sexDistributionTypeName: Both Sexes

## Appendix E: The MageTest Program

```
import gov.nih.nci.mageom.bean.Experiment.*;
import gov.nih.nci.mageom.bean.BioAssayData.*;
import gov.nih.nci.mageom.bean.BioAssay.*;
import gov.nih.nci.mageom.bean.Description.*;
import gov.nih.nci.mageom.bean.QuantitationType.*;
import gov.nih.nci.mageom.domain.Experiment.*;
import gov.nih.nci.mageom.domain.BioAssay.*;
import gov.nih.nci.mageom.domain.BioAssayData.*;
import gov.nih.nci.mageom.domain.Description.*;
import gov.nih.nci.mageom.search.*;

import java.util.Vector;

/**
 * This class provides sample code that demonstrates the following.
 * 1) Getting a list of experiments in the database.
 * 2) Getting a list of bioassays for each experiment.
 * 3) Getting the bioassay datum values from each bioassay.
 */
public class MageTest
{
    /**
     * This method will dump the bioassay datum values obtained from the
     * specified BioAssayDataImpl object to stdout.
     *
     * @param bioAssayData - The object to obtain the bioassay datum values
     * from.
     */
    public static void dumpBioDataValues(BioAssayDataImpl bioAssayData)
    {
        try
        {
            long totalMillis = 0;

            //first get all the values into memory... time this
            long startTimeMillis = 0;
            long endTimeMillis = 0;

            startTimeMillis = System.currentTimeMillis();
            BioDataValuesImpl values = bioAssayData.getBioDataValuesImpl();
            endTimeMillis = System.currentTimeMillis();
            totalMillis += (endTimeMillis - startTimeMillis);

            if(values instanceof BioDataTuplesImpl)
            {
                BioDataTuplesImpl tuples = (BioDataTuplesImpl)values;

                startTimeMillis = System.currentTimeMillis();
                BioAssayDatumImpl[] datum = tuples.getBioAssayTupleDataImpl();
                endTimeMillis = System.currentTimeMillis();
                totalMillis += (endTimeMillis - startTimeMillis);

                if(datum != null)
                {
                    System.out.println(" BioDataTuple returned " + datum.length + " BioAssayDatum");
                    Vector outputValues = new Vector();
                    for(int x = 0; x < datum.length; x++)
                    {
                        startTimeMillis = System.currentTimeMillis();
                        String value = datum[x].getValue();
                        endTimeMillis = System.currentTimeMillis();
                        totalMillis += (endTimeMillis - startTimeMillis);

                        String quantTypeDesc = "";

                        startTimeMillis = System.currentTimeMillis();
                        QuantitationTypeImpl qType = datum[x].getQuantitationTypeImpl();
                        DescriptionImpl[] descs = (DescriptionImpl[])qType.getDescriptions();
                        endTimeMillis = System.currentTimeMillis();
                        totalMillis += (endTimeMillis - startTimeMillis);

                        if(descs != null && descs.length > 0)
                        {

```

```

        startTimeMillis = System.currentTimeMillis();
        quantTypeDesc = descs[0].getText();
        endTimeMillis = System.currentTimeMillis();
        totalMillis += (endTimeMillis - startTimeMillis);
    }
    else
    {
        quantTypeDesc = "Unknown";
    }
    System.out.println(" Datum #" + x + ":" + quantTypeDesc + "=" + value);
    outputValues.add(quantTypeDesc + ": " + value);
}

System.out.println("*****");
System.out.println("The preceding " + datum.length +
    " BioAssayDatum were retrieved in " + totalMillis + " milliseconds.");
System.out.println("*****");
outputValues.clear();
outputValues = null;
}
else
{
    System.out.println("No data available for bioassay from impl.");
}
}
else
{
    System.out.println("Returned data was not BioDataTuples: " +
        values.getClass().toString());
}
}
catch(Exception e)
{
    System.out.println("Error getting details of bio data: " + e.toString());
    e.printStackTrace();
}
}

/**
 * Gets the list of bioassays for the experiment and calls a method to print
 * out the bioassay datum for each.
 *
 * @param exp - The experiment object to get bioassays from.
 */
public static void dumpExperimentBioAssays(ExperimentImpl exp)
{
    try
    {
        Long id = exp.getId();
        BioAssayImpl[] bioassays = exp.getBioAssaysImpl();

        if(bioassays.length > 0)
        {
            System.out.println("Experiment: " + id + " returned " +
                bioassays.length + " bioassays.");
        }

        for(int y = 0; y < bioassays.length; y++)
        {
            try
            {
                if(bioassays[y] instanceof MeasuredBioAssay)
                {
                    System.out.println("Measured BioAssay");
                    MeasuredBioAssayDataImpl[] baData =
                        ((MeasuredBioAssayImpl)bioassays[y]).getMeasuredBioAssayDataImpl();
                    for(int z = 0; z < baData.length; z++)
                    {
                        dumpBioDataValues(baData[z]);
                    }
                }
                else if(bioassays[y] instanceof DerivedBioAssay)
                {
                    System.out.println("Derived BioAssay");
                    DerivedBioAssayDataImpl[] baData =

```



```

        ((DerivedBioAssayImpl)bioassays[y]).getDerivedBioAssayDataImpl();
        System.out.println("got " + baData.length + " DerivedBioAssayDataImpl-s back");
        for(int z = 0; z < baData.length; z++)
        {
            dumpBioDataValues(baData[z]);
        }
    }
    else
    {
        System.out.println("Unknown BioAssay type returned.");
    }

    bioassays[y] = null;
}
catch(Exception e)
{
    System.out.println("Error getting bioassaydata: " + e);
    e.printStackTrace();
}
}
}
catch(Exception e)
{
    System.out.println("Error getting experiment details: " + e.getMessage());
    e.printStackTrace();
}
}
}

/**
 * Returns an array of all experiments with the specified ID. Array should have
 * zero or one elements on return, zero if there are no experiments with the
 * specified Id.
 *
 * @param expId - The experiment Id to search on.
 */
public static ExperimentImpl getExperimentWithId(long expId)
{
    ExperimentImpl experiment = null;

    try
    {
        experiment = new ExperimentImpl(new Long(expId));
    }
    catch(Exception e)
    {
        System.out.println("Error creating experiment '" + expId + "': " + e.getMessage());
        e.printStackTrace();
    }

    return experiment;
}

/**
 * Returns a list of all experiments available on the server.
 */
public static ExperimentImpl[] getExperiments()
{
    ExperimentImpl[] experiments = new ExperimentImpl[0];

    Vector temp = new Vector();
    try
    {
        ExperimentImpl emptyExp = new ExperimentImpl();
        //get all experiments by searching experiments with empty search criteria
        SearchResult results = emptyExp.search(new SearchCriteria());
        Object[] objects = results.getResultSet();
        for(int x = 0; x < objects.length; x++)
        {
            if(objects[x] instanceof ExperimentImpl)
            {
                temp.add(objects[x]);
            }
        }
    }
}
}

```

```

catch(Exception e)
{
    System.out.println("Error getting experiments from server: " + e.getMessage());
    e.printStackTrace();
}

//dump the experiment objects into the return array.
experiments = (ExperimentImpl[])temp.toArray(experiments);

return experiments;
}

/**
 * Test driver. Performs simple test that dumps all bioassay datum values from
 * all experiments to stdout. The program also logs performance measurements as
 * it goes.
 *
 * @param args - Not used.
 */
public static void main(String[] args)
{
    if(args.length > 0)
    {
        String expIdStr = args[0];
        try
        {
            long expId = Long.valueOf(expIdStr).longValue();
            ExperimentImpl exp = getExperimentWithId(expId);
            System.out.print(" Experiment #" + expId);
            System.out.print(" name=" + exp.getName());
            System.out.println(" plat=" + exp.getPlatformType());
            dumpExperimentBioAssays(exp);
        }
        catch(Exception e)
        {
        }
    }
    else
    {
        ExperimentImpl[] experiments = getExperiments();
        for(int x = 0; x < experiments.length; x++)
        {
            dumpExperimentBioAssays(experiments[x]);
        }
    }
}
}

```

## Sample Output from MageTest:

Experiment #2915 name=CGH plat=BAC - Spotted  
Experiment: 2915 returned 1 bioassays.  
Derived BioAssay

```
got 1 DerivedBioAssayDataImpl-s back
BioDataTuple returned 1000 BioAssayDatum
Datum #0:Log2(ratio)=.168334
Datum #1:Log2(ratio)=.131524
Datum #2:Log2(ratio)=.159808
Datum #3:Log2(ratio)=.057781
Datum #4:Log2(ratio)=.223079
Datum #5:Log2(ratio)=-.002598
Datum #6:Log2(ratio)=.134221
Datum #7:Log2(ratio)=.135854
Datum #8:Log2(ratio)=.038074
Datum #9:Log2(ratio)=.097623
Datum #10:Log2(ratio)=-.019812
Datum #11:Log2(ratio)=.199289
Datum #12:Log2(ratio)=.139072
Datum #13:Log2(ratio)=.208712
Datum #14:Log2(ratio)=.128289
Datum #15:Log2(ratio)=.081225
Datum #16:Log2(ratio)=.011551
Datum #17:Log2(ratio)=.069545
Datum #18:Log2(ratio)=.150789
Datum #19:Log2(ratio)=-.002607
Datum #20:Log2(ratio)=.18813
Datum #21:Log2(ratio)=.120432
Datum #22:Log2(ratio)=.180525
Datum #23:Log2(ratio)=.132566
Datum #24:Log2(ratio)=.187966
Datum #25:Log2(ratio)=-.013124
Datum #26:Log2(ratio)=.105193
Datum #27:Log2(ratio)=.163016
Datum #28:Log2(ratio)=.076419
Datum #29:Log2(ratio)=.137624
Datum #30:Log2(ratio)=.041027
Datum #31:Log2(ratio)=-.026699
Datum #32:Log2(ratio)=.15916
Datum #33:Log2(ratio)=.15978
Datum #34:Log2(ratio)=.111909
Datum #35:Log2(ratio)=.168225
Datum #36:Log2(ratio)=.173834
Datum #37:Log2(ratio)=.133104
Datum #38:Log2(ratio)=.22337
Datum #39:Log2(ratio)=.09875
Datum #40:Log2(ratio)=.152262
Datum #41:Log2(ratio)=.11133
Datum #42:Log2(ratio)=.179602
Datum #43:Log2(ratio)=.110244
Datum #44:Log2(ratio)=.151796
Datum #45:Log2(ratio)=-.031964
Datum #46:Log2(ratio)=-.094368
Datum #47:Log2(ratio)=.175549
Datum #48:Log2(ratio)=.067724
Datum #49:Log2(ratio)=-.10684
Datum #50:Log2(ratio)=.178781
Datum #51:Log2(ratio)=.004972
Datum #52:Log2(ratio)=.025608
Datum #53:Log2(ratio)=-.100067
Datum #54:Log2(ratio)=-.005743
Datum #55:Log2(ratio)=.138009
Datum #56:Log2(ratio)=.04702
Datum #57:Log2(ratio)=.034106
Datum #58:Log2(ratio)=.186648
Datum #59:Log2(ratio)=.097639
Datum #60:Log2(ratio)=-.072626
Datum #61:Log2(ratio)=.211926
Datum #62:Log2(ratio)=.129871
Datum #63:Log2(ratio)=.093671
```

*[additional pages of output omitted]*

## Appendix F: The caBIO\_MageTest Program

```
import gov.nih.nci.caBIO.bean.Gene;
import gov.nih.nci.caBIO.bean.GeneSearchCriteria;
import gov.nih.nci.mageom.bean.*;
import gov.nih.nci.mageom.bean.BioAssay.*;
import gov.nih.nci.mageom.bean.BioAssayData.*;
import gov.nih.nci.mageom.bean.DesignElement.*;
import gov.nih.nci.mageom.bean.Experiment.*;
import gov.nih.nci.mageom.domain.Description.*;
import gov.nih.nci.mageom.domain.BioAssay.*;

/**
 * This class provides sample code that demonstrates the following.
 * 1) Getting an experiment from the database.
 * 2) Getting a list of bioassays for the experiment.
 * 3) Getting the bioassay datum values from each bioassay.
 * 4) Get the DesignElement/Reporter from the bioassay
 * 5) Get the caBio Gene corresponding to the Reporter
 *
 * Invoke as: MageTest <experiment-id>
 */
public class MageTest {
    /**
     * This method will retrieve the caBIO Gene corresponding to a MAGE-OM Reporter
     *
     * @param theReporter - The Mage-OM ReporterImpl object to obtain the
     *                       corresponding caBIO Gene
     */
    public static void getCabioGeneInfo(ReporterImpl theReporter) {
        try {
            Long theId = theReporter.getId();
            if ( theId == null ) return;

            System.out.println("MAGE-OM Reporter id =" + theId);
            System.out.println("MAGE-OM Reporter name=" + theReporter.getName());

            GeneSearchCriteria caCrit = new GeneSearchCriteria();
            caCrit.setExpressionMeasurementId(theId);
            gov.nih.nci.caBIO.bean.SearchResult caRes =
                (new Gene()).search(caCrit);
            Gene[] caGenes = (Gene[]) caRes.getResultSet();
            System.out.println(" Genes from cabio #=" + caGenes.length);

            for (int ig = 0; ig < caGenes.length; ig++) {
                System.out.println(
                    "caBIO Gene #" + ig + " id = " + caGenes[ig].getId());
                System.out.println(
                    "caBIO Gene #" + ig + " nam= " + caGenes[ig].getName());
            }
        } catch (Exception e) {
            System.out.println(
                "Error getting cabio gene info: " + e.toString());
            e.printStackTrace();
        }
    }

    /**
     * This method will retrieve the bioassay datum values obtained from the
     * specified BioAssayDataImpl object. Get the DesignElement of the bioassay
     * datum. And finally call the getCabioGene
     *
     * @param bioAssayData - The object to obtain the bioassay datum values
     *                       from.
     */
    public static void getBioDataValues(BioAssayDataImpl bioAssayData) {
        try {
            BioDataValuesImpl values = bioAssayData.getBioDataValuesImpl();

            if (values instanceof BioDataTuplesImpl) {
                BioDataTuplesImpl tuples = (BioDataTuplesImpl) values;
                BioAssayDatumImpl[] datum = tuples.getBioAssayTupleDataImpl();

                if (datum != null) {

```

```

        System.out.println(
            datum.length + " MAGE-OM BioAssayDatums returned");

        for (int x = 0; x < datum.length; x++) {
            String value = datum[x].getValue();
            Object theElement = datum[x].getDesignElementImpl();

            if (theElement instanceof ReporterImpl) {
                getCabioGeneInfo((ReporterImpl) theElement);
            } else {
                System.out.println("DesignElement not a Reporter?.");
            }
        }
    } else {
        System.out.println("No data available for bioassay from impl.");
    }
} else {
    System.out.println("Returned data was not BioDataTuples: " +
        values.getClass().toString());
}
} catch (Exception e) {
    System.out.println(
        "Error getting details of bio data: " + e.toString());
    e.printStackTrace();
}
}

/**
 * Test driver. A simple example of getting caBIO objects from MAGE-OM
 * references.
 *
 * @param args - the expIdStr
 */
public static void main(String[] args) {
    if (args.length > 0) {
        String expIdStr = args[0];
        try {
            long expId = Long.valueOf(expIdStr).longValue();
            ExperimentImpl exp = new ExperimentImpl(new Long(expId));
            if (exp == null)
                throw new Exception("Experiment " + expIdStr + " not loaded.");
            Long id = exp.getId();
            System.out.println("MAGE-OM Experiment Id = " + id);
            System.out.println("MAGE-OM Experiment Name = " + exp.getName());
            System.out.println("MAGE-OM Experiment Platform = " +
                exp.getPlatformType());

            Description[] eDescripts=exp.getDescriptions();
            System.out.println("MAGE-OM Experiment # Descripts = " +
                eDescripts.length);
            for (int di=0; di < eDescripts.length; di++){
                System.out.println("MAGE-OM Experiment Desc# " + di + " = " +
                    eDescripts[di].getText());
            }

            BioAssayImpl[] bioassays = exp.getBioAssaysImpl();
            System.out.println("MAGE-OM Experiment: " + id + " returned " +
                bioassays.length + " bioassays.");

            if (bioassays.length > 0) {
                for (int y = 0; y < bioassays.length; y++) {
                    Long bioassayId = bioassays[y].getId();

                    if (bioassays[y] instanceof MeasuredBioAssay) {
                        //String bioAssayName = bioassays[y].
                        MeasuredBioAssayDataImpl[] baData = ((MeasuredBioAssayImpl)
                            bioassays[y]).getMeasuredBioAssayDataImpl();
                        for (int z = 0; z < baData.length; z++) {
                            getBioDataValues(baData[z]);
                        }
                    } else if (bioassays[y] instanceof DerivedBioAssay) {
                        DerivedBioAssayDataImpl[] baData = ((DerivedBioAssayImpl)
                            bioassays[y]).getDerivedBioAssayDataImpl();

                        for (int z = 0; z < baData.length; z++) {

```

```
        getBioDataValues(baData[z]);
    }
} else {
    System.out.println(
        "Unknown BioAssay type returned." + bioassays[y])
    }
} else {
    System.out.println("No BioAssay data.");
}
} catch (Exception e) {
    System.out.println("Failure Except " + e);
    e.printStackTrace();
}
} else {
    System.out.println(
        "Please specify an experiment id as an argument.");
}
}
}
```

## Sample Output from caBIOMageTest:

```
MAGE-OM Experiment Id = 2998
MAGE-OM Experiment Name = test
MAGE-OM Experiment Platform = cDNA - Spotted
MAGE-OM Experiment # Descriptors = 1
MAGE-OM Experiment Desc#0 = test
MAGE-OM Experiment: 2998 returned 1 bioassays.
8 MAGE-OM BioAssayDatums returned
MAGE-OM Reporter id =86181
MAGE-OM Reporter name=92584_at
  Genes from cabio #=1
caBIO Gene #0 id = 111155
caBIO Gene #0 nam= Kbras2-pending
MAGE-OM Reporter id =85511
MAGE-OM Reporter name=98065_at
  Genes from cabio #=1
caBIO Gene #0 id = 147251
caBIO Gene #0 nam= Ormdl3
MAGE-OM Reporter id =85397
MAGE-OM Reporter name=102327_at
  Genes from cabio #=0
MAGE-OM Reporter id =84927
MAGE-OM Reporter name=101011_at
  Genes from cabio #=1
caBIO Gene #0 id = 122396
caBIO Gene #0 nam= Cct4
MAGE-OM Reporter id =86738
MAGE-OM Reporter name=94713_at
  Genes from cabio #=1
caBIO Gene #0 id = 108665
caBIO Gene #0 nam= Myo7a
MAGE-OM Reporter id =85404
MAGE-OM Reporter name=97614_f_at
  Genes from cabio #=1
caBIO Gene #0 id = 111738
caBIO Gene #0 nam= Anxa1
MAGE-OM Reporter id =86184
MAGE-OM Reporter name=95563_at
  Genes from cabio #=1
caBIO Gene #0 id = 151542
caBIO Gene #0 nam= Arih1
MAGE-OM Reporter id =85778
MAGE-OM Reporter name=92689_at
  Genes from cabio #=1
caBIO Gene #0 id = 121992
caBIO Gene #0 nam= I118bp
```

## Appendix G: The SearchPkgExample Program

```
import gov.nih.nci.caBIO.search.*;
import gov.nih.nci.caBIO.bean.*;

import java.util.*;
import java.io.*;

public class SearchPkgExample{

    public static void main( String[] args ){
        try{
            //Build the root search criteria
            List geneNames = new ArrayList();
            geneNames.add( "PTEN" );
            geneNames.add( "TP53" );
            geneNames.add( "BRCA1" );
            geneNames.add( "yadda" );
            geneNames.add( "CEACAM3" );
            geneNames.add( "CEACAM4" );
            GeneSearchCriteria gsc = new GeneSearchCriteria();
            gsc.putCriteria( "name", geneNames );

            // Initialize the query tree
            List geneAtts = new ArrayList();
            geneAtts.add( "name" );
            SelectionNode tree = new SelectionNodeImpl( "Gene", gsc, geneAtts );

            // Insert another selection node
            List pathAtts = new ArrayList();
            pathAtts.add( "name" );
            pathAtts.add("displayValue");

            SelectionNode pathNode =
                new SelectionNodeImpl( "Gene.pathways", new PathwaySearchCriteria(),pathAtts );
            tree.insert( pathNode, 0 );

            // Construct a SearchCriteriaMapping
            ObjectGrid og = new ObjectGridImpl();
            SearchCriteriaMapping[] batchSearch =
                new SearchCriteriaMapping[ geneNames.size() ];
            for( ListIterator i = geneNames.listIterator(); i.hasNext(); ){
                String clientData = (String)i.next();
                GeneSearchCriteria sc = new GeneSearchCriteria();
                sc.putCriteria( "name", clientData );
                SearchCriteriaMapping mapping = new SearchCriteriaMapping( clientData, sc );
                batchSearch[i.previousIndex()] = mapping;
            }
            GridSearchCriteria sc = new GridSearchCriteria( tree, batchSearch );
            GridSearchResultMapping[] results = og.search( sc );

            // Pull out the results
            for( int i = 0; i < results.length; i++ ){
                GridSearchResultMapping resultMapping = results[i];
                String name = (String)resultMapping.getClientData();
                GridRow[] rows = resultMapping.getResult();
                System.out.println( "\n" + rows.length + " results for " + name + "\n");
                for( int j = 0; j < rows.length; j++ ){
                    GridCell cell1 = rows[j].getCell( "Gene.name" );
                    GridCell cell2 = rows[j].getCell( "Gene.pathways.name" );
                    GridCell cell3 = rows[j].getCell( "Gene.pathways.displayValue" );
                    if (cell1 != null && cell2 != null && cell3 != null){
                        System.out.println( cell1.getObject() + ": " +
                            cell2.getObject() + " ( " +
                                cell3.getObject() + " )" );
                    }
                }
            }
        } catch( Exception ex ){
            ex.printStackTrace();
            System.exit( 1 );
        }
        System.exit( 0 );
    }
}
```



## Sample Output from SearchPkgExample:

6 results for PTEN

```
PTEN: mtorPathway( mTOR Signaling Pathway )
PTEN: ptenPathway( PTEN Dependent Cell Cycle Arrest and Apoptosis )
PTEN: eif4Pathway( Regulation of eIF4e and p70 S6 Kinase )
Pten: m_eif4Pathway( Regulation of eIF4e and p70 S6 Kinase )
Pten: m_mtorPathway( mTOR Signaling Pathway )
Pten: m_ptenPathway( PTEN dependent Cell Cycle Arrest and Apoptosis )
```

15 results for TP53

```
TP53: g1Pathway( Cell Cycle: G1/S Check Point )
TP53: p53Pathway( p53 Signaling Pathway )
TP53: rnaPathway( Double Stranded RNA Induced Gene Expression )
TP53: tidPathway( Chaperones Modulate Interferon Signaling Pathway )
TP53: atmPathway( ATM Signaling Pathway )
TP53: pmlPathway( Regulation of Transcriptional Activity by PML )
TP53: efpPathway( Estrogen-responsive protein Efp controls cell cycle and breast tumors growth )
TP53: arfPathway( Tumor Suppressor Arf Inhibits Ribosomal Biogenesis )
TP53: g2Pathway( Cell Cycle: G2/M Checkpoint )
TP53: telPathway( Telomeres, Telomerase, Cellular Aging and Immortality )
TP53: plk3Pathway( Regulation of Cell Cycle Progression by Plk3 )
TP53: p53hypoxiaPathway( Hypoxia and p53 in the Cardiovascular System )
TP53: rbPathway( RB Tumor Suppressor/Checkpoint Signaling in Response to DNA Damage )
TP53: atrbrcaPathway( Role of Bracl, Brac2 and Atr )
TP53: tertPathway( Overview of telomerase protein component gene hTert Transcriptional Regulation )
```

7 results for BRCA1

```
BRCA1: atrbrcaPathway( Role of Bracl, Brac2 and Atr )
BRCA1: atmPathway( ATM Signaling Pathway )
BRCA1: g2Pathway( Cell Cycle: G2/M Checkpoint )
BRCA1: carm-erPathway( CARM1 and Regulation of the Estrogen Receptor )
Brcal: m_atmPathway( ATM Signaling Pathway )
Brcal: m_g2Pathway( Cell Cycle: G2/M Checkpoint )
Brcal: m_atrbrcaPathway( Role of Bracl, Brac2 and Atr )
```

0 results for yadda

1 results for CEACAM3

1 results for CEACAM4

## TypicalCabioSearch Program

```
import gov.nih.nci.caBIO.search.*;
import gov.nih.nci.caBIO.bean.*;
import java.util.*;
import java.io.*;

/*
 * This example is used to demonstrate the same functionality as the previous
 * example - but in this case, using the basic search methods rather than the
 * advanced methods defined in the search packages
 */

public class TypicalCabioSearch {

    public static void main (String args[])
    {
        System.out.println("Running the main of TypicalCabioSearch ");
        try {

            // add the Gene Names to an Array List object

            List geneNames = new ArrayList();
            geneNames.add( "PTEN" );
            geneNames.add( "TP53" );
            geneNames.add( "BRCA1" );
            geneNames.add( "yadda" );
            geneNames.add( "CEACAM3" );
            geneNames.add( "CEACAM4" );

            // Now use a GeneSearchCriteria Object to apply each criteria
            // and iterate over the SearchResults one at a time

            Gene gene = new Gene();
            SearchResult result = null;
            System.out.println("\nInformation regarding Gene ");

            for( ListIterator i = geneNames.listIterator(); i.hasNext(); ){
                String symbol = (String) i.next();
                GeneSearchCriteria sc = new GeneSearchCriteria();
                sc.setSymbol(symbol);
                result = gene.search(sc);
                if (result != null){
                    Gene[] genes = (Gene[]) result.getResultSet();
                    for(int j = 0; j < genes.length; j++){
                        System.out.println("\nGene "+genes[j].getName());

                        // now define a second query to get the Pathways for this gene

                        Pathway[] pathways = genes[j].getPathways();
                        if( pathways.length > 0 ){
                            System.out.println("\nAssociated Pathway(s): ");
                            for(int k = 0; k < pathways.length; k++){
                                System.out.println("\t"+ pathways[k].getDisplayValue()+ " ( " +
                                    pathways[k].getName() +" )");
                            } // end for
                        } // end if
                    } // end for
                } // end if
            } // end for

        } catch (Exception exc) {
            System.out.println("Test failed in the main of TypicalCabioSearch.java: " +
                exc.getMessage());
            exc.printStackTrace();
        }
    } // end main
}
```

## APPENDIX H. geneClient.pl

```
#!/usr/bin/perl -w

#

# Client side SOAP application which tests the urn:nci-gene-service
#

use SOAP::Lite;
use HTML::Entities;

$USAGE=qq`
Usage: geneClient.pl {server} {port} {method} [-organism {organism}]
  [-symbol {symbol}] [-genBankAccessionNumber {genBankAccessionNumber}]
  [-unigeneClusterId {unigeneClusterId}] [-pathwayId {pathwayId}]
  [-allPathwayId {allPathwayId}] [-expressedPathwayId {expressedPathwayId}]
  [-overExpressedPathwayId {overExpressedPathwayId}]
  [-underExpressedPathwayId {underExpressedPathwayId}]
  [-mutatedGenePathwayId {mutatedGenePathwayId}] [-chromosomeId {chromosomeId}]
  [-uniqueIdentifier {uniqueIdentifier}] [-tissueType {tissueType}]
  [-functionalPathway {functionalPathway}]
  [-cytogenicLocation {cytogenicLocation}] [-geneNameKeyword {geneNameKeyword}]
  [-goOntologyHomoSapienId {goOntologyHomoSapienId}]
  [-goOntologyMouseId {goOntologyMouseId}]
  [-resultStart {resultStartNum} ]
  [-resultCount {resultCountNum} ]
  [-fillInObjects {objects comma delimited such as GoOntology, Maplocation } ]
  [-returnHeavyXML {"true"} to return a completely fill in XML document ]
... where {method} must be one of the following:

... where {method} must be one of the following:
getGenes, getTaxons, getClones, getReferenceSequences, getBioCartaIds,
getGenomicSequences, getGoOntologies, getGeneHomologs, getSequences,
getPathways, getChromosome, getMapLocations, getExpressionFeature,
getProteins, getGenes, getTaxons, getClones,
getReferenceSequences, getBioCartaIds,
getGenomicSequences, getGoOntologies, getGeneHomologs,
```

```

getSequences, getPathways, getChromosome,
getMapLocations, getExpressionFeature, getProteins
for complete upto date method list, please see
http://ncicb.nci.nih.gov/content/coreftp/caBIO\_JavaDocs/index.html

e.g.,
\ geneClient.pl cabio.nci.nih.gov 80 getGenes -symbol brca1
`
;

$URI='urn:nci-gene-service';
$PROXY_PATH='/soap/servlet/rpcrouter';

my $argc=@ARGV;

# anonymous hash reference, passed as map to SOAP service
my $searchRec={};

# first three arguments are mandatory
if ($argc < 3) {
    die $USAGE;
}
else {
    $server = shift @ARGV;
    $port = shift @ARGV;
    $method = shift @ARGV;

    $argc=@ARGV;
    while ($argc) {
        my $arg = shift @ARGV;

        if ($arg eq "-organism") {$searchRec->{organism} = shift @ARGV;}
        elsif ($arg eq "-symbol") {$searchRec->{symbol} = shift @ARGV;}
        elsif ($arg eq "-genBankAccessionNumber") {$searchRec->{genBankAccessionNumber} = shift @ARGV;}
        elsif ($arg eq "-unigeneClusterId") {$searchRec->{unigeneClusterId} = shift @ARGV;}
        elsif ($arg eq "-pathwayId") {$searchRec->{pathwayId} = shift @ARGV;}
    }
}

```

```

    elsif ($arg eq "-allPathwayId") {$searchRec->{allPathwayId} = shift @ARGV;}
    elsif ($arg eq "-expressedPathwayId") {$searchRec->{expressedPathwayId} = shift @ARGV;}
    elsif ($arg eq "-overExpressedPathwayId") {$searchRec->{overExpressedPathwayId} = shift
@ARGV;}
    elsif ($arg eq "-underExpressedPathwayId") {$searchRec->{underExpressedPathwayId} = shift
@ARGV;}
    elsif ($arg eq "-mutatedGenePathwayId") {$searchRec->{mutatedGenePathwayId} = shift @ARGV;}
    elsif ($arg eq "-chromosomeId") {$searchRec->{chromosomeId} = shift @ARGV;}
    elsif ($arg eq "-uniqueIdentifier") {$searchRec->{uniqueIdentifier} = shift @ARGV;}
    elsif ($arg eq "-tissueType") {$searchRec->{tissueType} = shift @ARGV;}
    elsif ($arg eq "-functionalPathway") {$searchRec->{functionalPathway} = shift @ARGV;}
    elsif ($arg eq "-cytogenicLocation") {$searchRec->{cytogenicLocation} = shift @ARGV;}
    elsif ($arg eq "-geneNameKeyword") {$searchRec->{geneNameKeyword} = shift @ARGV;}
    elsif ($arg eq "-goOntologyHomoSapienId") {$searchRec->{goOntologyHomoSapienId} = shift
@ARGV;}
    elsif ($arg eq "-goOntologyMouseId") {$searchRec->{goOntologyMouseId} = shift @ARGV;}
    elsif ($arg eq "-resultStart") {$searchRec->{resultStart} = shift @ARGV;}
    elsif ($arg eq "-resultCount") {$searchRec->{resultCount} = shift @ARGV;}
        elsif ($arg eq "-fillInObjects") {$searchRec->{fillInObjects} = shift @ARGV;}
    elsif ($arg eq "-returnHeavyXML") {$searchRec->{returnHeavyXML} = shift @ARGV;}

    $argc=@ARGV;
}
}

$s = SOAP::Lite
-> uri($URI)
-> proxy("http://$server:$port$PROXY_PATH");

# make service request
$som=$s->$method(SOAP::Data->type(map => $searchRec));

# interpret result
if ($som->fault) {
    print "FAULT ENCOUNTERED!\nfaultcode:\t" . $som->faultcode . "\nfaultstring:\t" . $som-
>faultstring . "\n";
} else {
    $xmldoc = $som->result;

```

```
print "\nMETHOD CALLED: $method\n\n" . decode_entities($xmldoc) . "\n\n";  
}
```