

# CGEMS 1.0

## *Technical Guide*



Center for Bioinformatics

December 13, 2006



# TABLE OF CONTENTS

<b>About This Guide .....</b>	<b>1</b>
Purpose .....	1
Release Schedule .....	1
Audience .....	1
Topics Covered .....	2
Additional CGEMS Documentation .....	2
Conventions Used .....	3
Credits and Resources .....	3
<b>Chapter 1</b>	
<b>Introduction to CGEMS .....</b>	<b>5</b>
About CGEMS .....	5
Additional CGEMS Resources .....	6
About caIntegrator .....	6
About caBIG .....	7
About caCORE .....	7
<b>Chapter 2</b>	
<b>CGEMS Architecture .....</b>	<b>11</b>
Clinical Genomic Object Model .....	11
CGEMS API Classes .....	14
Main CGEMS System Components .....	16
<b>Chapter 3</b>	
<b>Understanding the Object Query Service API .....</b>	<b>17</b>
Querying CGEMS Objects .....	17
About the Service Layer .....	17
Accessing the Object Query Service .....	18
Installing and Configuring the Object Query Service API .....	18
Downloading and Installing the Client Package .....	19
Testing the System .....	20
Using the Object Query Service API .....	21

TestClient Example .....	21
Service Methods .....	23
Scenario One: Retrieve All SNPPanels .....	26
Scenario Two: Simple Search (Criteria Object Collection) to retrieve SNPFrequencyFinding for the Gene "WT1" .....	27
Scenario Three: Nested Search to retrieve SNPAssays based on dbSnpId 29	
Scenario Four: Detached Criteria Search .....	30
Scenario Five: HQL Search .....	32

## **Appendix A**

<b>UML Modeling .....</b>	<b>33</b>
UML Modeling .....	33
Use Case Documents and Diagrams .....	34
Class Diagrams .....	37
Relationships Between Classes .....	38
Sequence Diagrams .....	40

## **Appendix B**

<b>CGEMS Glossary .....</b>	<b>43</b>
<b>Index .....</b>	<b>45</b>

# ABOUT THIS GUIDE

This section introduces you to the *CGEMS Technical Guide*. It includes the following topics:

- *Purpose* on this page
- *Release Schedule* on this page
- *Audience* on this page
- *Topics Covered* on page 2
- *Additional CGEMS Documentation* on page 2
- *Conventions Used* on page 3
- *Credits and Resources* on page 3

## Purpose

---

This guide provides an overview of the CGEMS architecture and explains how to use the CGEMS Application Programming Interface (API).

## Release Schedule

---

This guide is updated for each CGEMS release. It may be updated between releases if errors and omissions are found. The current document refers to the 1.0 version of CGEMS, which NCICB released in November 2006.

## Audience

---

This guide is designed for experienced Java developers who are familiar with the following J2EE technologies:

- Unix/Linux environment (Configuring environment variables; Installing Ant, JDK, and JBOSS server)
- Ant build scripts
- J2EE web application development using the Struts framework, Servlet/JSPs, JavaScript, AJAX, and XML/XSLT.
- J2EE middle-ware technologies such as n-tier service oriented architecture and software design patterns.

In addition, you will need assistance / access from an Oracle 9i database administrator to properly configure the database.

## Topics Covered

---

If you are new to CGEMS, please read this brief overview, which explains what you will find in each chapter and appendix.

This chapter provides an overview of the guide.

*Chapter 1* introduces the CGEMS study and provides an overview of calIntegrator, caBIG, and caCORE.

*Chapter 2* describes the CGEMS architectural model and components.

*Chapter 3* explains how to install, configure, and test the Object Query Service API and provides examples of use.

*Appendix A* provides general information about the Unified Modeling Language (UML).

*Appendix B* is a glossary of terms related to CGEMS.

## Additional CGEMS Documentation

---

The [calIntegrator-CGOM API Software Design Description](#) describes the design decisions, architectural design, and the detailed design needed to implement the calIntegrator's Clinical Genomic Object Model (CGOM) Application Programming Interface (API).

The [CGEMS Requirements Specification](#) includes the use cases that CGEMS supports.

The CGEMS JavaDocs, which are included in the client package on the NCICB Web site, contain the current CGEMS API specification.

## Conventions Used

This section explains conventions used in this document. The various typefaces represent interface components, keyboard shortcuts, toolbar buttons, dialog box options, and text that you type.

<b>Convention</b>	<b>Description</b>	<b>Example</b>
<b>Bold</b>	Highlights names of option buttons, check boxes, drop-down menus, menu commands, command buttons, or icons.	Click <b>Search</b> .
<u>URL</u>	Indicates a Web address.	<a href="http://domain.com">http://domain.com</a>
text in small caps	Indicates a keyboard shortcut.	Press ENTER.
text in small caps + text in small caps	Indicates keys that are pressed simultaneously.	Press SHIFT + CTRL.
<i>Italics</i>	Highlights references to other documents, sections, figures, and tables.	See <i>Figure 4.5</i> .
<i>Italic boldface monospace type</i>	Represents text that you type.	In the <b>New Subset</b> text box, enter <i>Proprietary Proteins</i> .
<b>Note:</b>	Highlights information of particular importance	<b>Note:</b> This concept is used throughout the document.
{ }	Surrounds replaceable items.	Replace {last name, first name} with the Principal Investigator's name.

## Credits and Resources

The following individuals contributed to the CGEMS project.

<b><i>Clinical Genetic Markers of Susceptibility (CGEMS) Development and Management Teams</i></b>			
<b><i>Product and Program Management</i></b>	<b><i>Development</i></b>	<b><i>Quality Assurance</i></b>	<b><i>Documentation</i></b>
Liming Yang <sup>2</sup>	Himanso Sahni <sup>1</sup>	Jenny Glenn <sup>3</sup>	Carolyn Kelley Klinger <sup>4</sup>
Subhashree Madhavan <sup>2</sup>	Ram Bhattaru <sup>1</sup>	Ying Long <sup>1</sup>	Eddie VanArsdall <sup>4</sup>
Carl Schaeffer <sup>2</sup>	Michael Holck <sup>5</sup>	We Yu <sup>1</sup>	Jill Hadfield <sup>2</sup>
	Dana Zhang <sup>1</sup>		
	Ryan Landy <sup>1</sup>		

<b>Clinical Genetic Markers of Susceptibility (CGEMS) Development and Management Teams</b>			
<b>Product and Program Management</b>	<b>Development</b>	<b>Quality Assurance</b>	<b>Documentation</b>
<sup>2</sup> National Cancer Institute Center for Bioinformatics (NCICB)	<sup>1</sup> Science Application International Corporation (SAIC)	<sup>3</sup> NARTech, Inc,	
	<sup>4</sup> Management System Designers, Inc.		
	<sup>5</sup> ScenPro		

<b>Contacts and Support</b>	
NCICB Application Support	<a href="http://ncicb.nci.nih.gov/NCICB/support">http://ncicb.nci.nih.gov/NCICB/support</a> Telephone: 301-451-4384 Toll free: 888-478-4423

# CHAPTER 1

## INTRODUCTION TO CGEMS

This chapter introduces you to the CGEMS study. It includes the following topics:

- *About CGEMS* on this page
- *About caIntegrator* on page 6
- *About caBIG* on page 7
- *About caCORE* on page 7

### About CGEMS

---

Cancer Genetic Markers of Susceptibility (CGEMS) is a three-year initiative of the National Cancer Institute that will conduct scans of the entire human genome (genotyping) to identify common, inherited gene mutations that increase the risks for breast and prostate cancer. To access data from this initiative, visit the [CGEMS data access portal](#).

The CGEMS study uses cases and controls from well-designed epidemiological studies to generate genotypes on over 500,000 genetic variants. As such, CGEMS is a Genome-wide Association Study, or GWAS. The two cancers being studied by CGEMS are prostate cancer and breast cancer.

For the prostate cancer study, the GWAS has been conducted in a large, national study in the [Prostate, Lung, Colorectal, and Ovary study \(PLCO\)](#). The analysis includes 1,177 individuals who developed prostate cancer during the observational period and 1,105 individuals who did not develop prostate cancer during the same time period. The prostate scan has been conducted in two parts, Phase 1A and Phase 1B.

The data generated by this CGEMS study can be accessed through this portal. The first posting includes Phase 1A of the prostate cancer scan and includes over 300,000 SNPs. The results of Phase 1B will be available in 2007. The project team has developed analytical tools that provide easy access to the data. The raw genotype data will be available to accredited investigators who register individually and provide

institutional confirmation of research intent. The process to obtain approval for access is under review and details will be posted by the end of November at this Web site.

The CGEMS study will test markers identified as promising in this scan of prostate cancer in follow-up epidemiologic studies, including case-control studies and studies that are members of the [NCI Breast & Prostate Cancer Cohort Consortium](#), a multi-center network of large prospective studies. Executive summaries of the results of the follow-up studies will be posted on this Web site.

Finally, CGEMS is performing genome scan in a total of 1,200 breast cancer cases and 1,200 controls. The samples are from the [Nurse's Health Study](#). The genotyping of these samples has been initiated and the data will be available in the 2007.

## Additional CGEMS Resources

The following CGEMS resources are available online.

<b>Resource</b>	<b>Description</b>
CGEMS Public Web site	<a href="#">Information about the CGEMS project and initiatives</a>
CGEMS Investigator Portal	<a href="#">Web portal</a> for researchers
Related system documents	Documents available on GForge: <ul style="list-style-type: none"> <li>• <a href="#">caIntegrator-CGOM API Software Design Document</a></li> <li>• <a href="#">CGEMS Requirements Specification</a></li> <li>• <a href="#">Clinical Genomic Object Model (CGOM)</a></li> </ul>

*Table 1.1 CGEMS Resource List*

## About caIntegrator

The caIntegrator knowledge framework provides researchers with the ability to perform ad hoc querying and reporting across multiple domains. This application framework comprises an n-tier service oriented architecture that allows pluggable web-based graphical user interfaces, a business object layer, server components that process the queries and result sets, a data access layer and a robust data warehouse.

The following principles guided the development of the caIntegrator framework:

- User requirements
- Design of a user-friendly interface for a wide-ranging audience (i.e., physician scientists, programmers, and statisticians)
- Standards-based and pattern-driven development
- Extensibility and scalability
- Reuse and extension of open source technologies

At the heart of caIntegrator is the Clinical Genomics Object Model (CGOM) that provides standardized programmatic access to the integrated biomedical data collected in the caIntegrator data system. Design of the CGOM is driven by use cases from two critical NCI-sponsored studies, a brain tumor trial called GMDI (Glioma Molecular Diagnostic Initiative) and a breast cancer study called I-SPY TRIAL (Investigation of Serial Studies to Predict Your Therapeutic Response with Imaging And moLecular

analysis). The model represents data from clinical trials, microarray-based gene expression, SNP genotyping and copy number experiments, and Immunohistochemistry-based protein assays.

Clinical domain objects in CGOM allow access to clinical trial protocol, treatment arms, patient information, sample histology, clinical observations and assessments. Genomic domain objects allow access to biospecimen information, raw experimental data, in-silico transformation and analyses performed on the raw experimental datasets and biomarker findings. The clinical and genomic findings domain objects have relationships to the FindingsOntology object, as the findings can be complex concepts which, in turn, can be generically represented as items occurring in an ontology (for example, WHO histopathological classification for brain tumor histology findings).

caIntegrator is envisioned to be the foundation for a number of translational applications. One such reference implementation at NCICB is called Rembrandt (Repository of Molecular BRAIn Neoplasia DaTa) – <http://rembrandt.nci.nih.gov>. This knowledge framework offers a paradigm for rapid sharing of information and accelerates the process of analyzing results from various biomedical studies with the ultimate goal to rapidly change routine patient care.

For more information about caIntegrator and CGOM, see the [caIntegrator-CGOM API Software Design Description](#).

## About caBIG

---

The Cancer Biomedical Informatics Grid (caBIG)<sup>™</sup> delivers CGEMS data to researchers and the public. caBIG<sup>™</sup> is a voluntary network or grid of individuals and institutions that are working to create a better environment for the sharing of cancer research data and software tools. The goal of the network is to speed the delivery of innovative approaches for the prevention, detection, and treatment of cancer.

Since its launch in February 2004, caBIG<sup>™</sup> has delivered a variety of cancer and biomedical research products, including software tools, data sets, infrastructure, standards and policy papers. All are freely available to the community and other interested stakeholders.

caBIG<sup>™</sup> is being developed under the leadership of the National Cancer Institute, the NCI Center for Bioinformatics (NCICB), and other caBIG participants.

For more information about caBIG, see the caBIG<sup>™</sup> web site at <https://cabig.nci.nih.gov>.

## About caCORE

---

Cancer Common Ontologic Representation Environment (caCORE) is a data management framework that is compatible with caBIG. It was designed for researchers who need to be able to navigate through a large number of data sources. The components of caCORE support the semantic consistency, clarity, and comparability of biomedical research data and information.

caCORE is an open-source, enterprise architecture for NCI-supported research information systems. It was built using formal techniques from the software engineering and computer science communities.

caCORE uses the following four development principles:

- Model Driven Architecture (MDA)
- *n*-tier architecture with open Application Programming Interfaces (APIs)
- Use of controlled vocabularies, wherever possible
- Registered metadata

The following domain models comprise caCORE:

- **Enterprise Vocabulary Services (EVS)**  
EVS provides controlled vocabulary resources for the life sciences domain. EVS products include the NCI Thesaurus (a biomedical thesaurus), and the NCI Metathesaurus, which is based on the National Library of Medicine's Unified Medical Language System.
- **Cancer Bioinformatics Infrastructure Objects (caBIO)**  
The caBIO model and architecture are the primary programmatic interface to caCORE. Each of the caBIO domain objects represents an entity found in biomedical research.
- **Cancer Data Standards Repository (caDSR)**  
caDSR is a metadata registry based on the ISO/IEC 11179 standard. It is used to register the descriptive information needed to render cancer research data reusable and interoperable.

The caCORE infrastructure exhibits an *n*-tiered architecture with client interfaces, server components, backend objects, data sources, and additional backend systems (*Figure 1.1*). This *n*-tiered system divides tasks or requests among different servers and data stores. This isolates the client from the details of where and how data is retrieved from different data stores. The system also performs common tasks such as logging and provides a level of security.

Clients (browsers, applications) receive information from backend objects. Java applications also communicate with backend objects via domain objects packaged within the client.jar. Non-Java applications can communicate via SOAP (Simple Object Access Protocol). Back-end objects communicate directly with data sources, either relational databases (using Hibernate) or non-relational systems (using, for example, the Java RMI API).

# caCORE Architecture

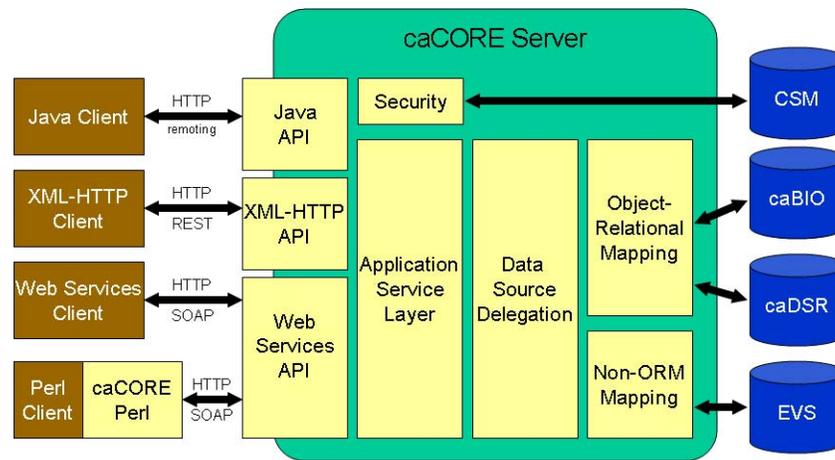


Figure 1.1 caCORE Architecture

Most of the caCORE infrastructure is written in the Java programming language and leverages reusable, third-party components.

The infrastructure is composed of the following layers:

**The Application Service layer** — consolidates incoming requests from the various interfaces and translates them to native query requests that are then passed to the data layers. This layer is also responsible for handling client authentication and access control using the Java API. (This feature is currently disabled for the caCORE system running at NCICB; all interfaces provide full, anonymous read-only access to all data.)

**The Data Source Delegation layer** — is responsible for conveying each query that it receives to the respective data source that can perform the query. The presence of this layer enables multiple data sources to be exposed by a single running instance of a caCORE server.

**Object-Relational Mapping (ORM)** — is implemented using Hibernate. Hibernate is a high performance object/relational persistence and query service for Java. Hibernate provides the ability to develop persistent classes following common object-oriented (OO) design methodologies such as association, inheritance, polymorphism, and composition.

The *Hibernate Query Language (hql)*, designed as a "minimal" object-oriented extension to SQL, provides a bridge between the object and relational databases. Hibernate allows for real world modeling of biological entities without creating complete SQL-based queries to represent them.

Access to non-relational (non-ORM data sources), such as Enterprise Vocabulary Services (EVS), is performed by objects that follow the façade design pattern. These objects make the task of accessing a large number of modules/functions much simpler

by providing an additional interface layer which allows it to interact with the rest of the caCORE system.

Security is provided by the Common Security Module (CSM). The CSM provides highly granular access control and authorization schemes.

Enterprise logging is provided by the Common Logging Module (CLM). The CLM provides a separate service under caCORE for audit and logging capabilities. This is similar to the output generated by Apache log4j, but includes information for auditing.

For more information about caCORE, see the caCORE documentation available at <http://ncicb.nci.nih.gov/infrastructure>.

# CHAPTER 2

## CGEMS ARCHITECTURE

This chapter describes the CGEMS architectural model and components. It includes the following topics:

- *Clinical Genomic Object Model* on this page
- *CGEMS API Classes* on page 14
- *Main CGEMS System Components* on page 16

### Clinical Genomic Object Model

---

The Clinical Genomic Object Model (CGOM) is a domain model based on a common set of use cases that were derived from various translational studies such as CGEMS. The purpose of the CGOM is to model the translation space that highlights the integration of the clinical domain with the genomic domain within a context of a clinical study.

Design of the CGOM is driven by use cases from three critical NCI-sponsored studies: a brain tumor trial called the Glioma Molecular Diagnostic Initiative (GMDI), a breast cancer study called I-SPY TRIAL, and CGEMS. The model represents data from clinical trials, micro array-based gene expression, SNP genotyping and copy number experiments, Fluorescent *in situ* Hybridization (FISH), Somatic Mutation, Cell Lycate, and Immunohistochemistry-based protein assays.

Study domain objects in CGOM allow access to the study, treatment arms, patient information, specimen histology, and information on the biospecimen. The *Finding objects* model the in-silico transformation and analyses performed on the raw experimental datasets. The clinical findings domain objects provide clinical observations and assessments. Annotation objects such as GeneBiomarker, ProteinBiomarker, and SNPAnnotation help provide context to the various Findings.

CGEMS domain objects are a subset of the calIntegrator domain. See this subset in [Figure 2.1](#) on page 13.



## CGEMS API Classes

The Object Query Service enables API users to initiate a search from any object within the CGOM and retrieve the query results as a domain object graph. caIntegrator uses the caCORE SDK tool kit to implement the Object Query Service. For more information, see *Understanding the Object Query Service API* on page 17.

The CGEMS UML model is published as an EA (Enterprise Architect) diagram at [http://cabigcvs.nci.nih.gov/viewcvs/viewcvs.cgi/caIntegrator-spec/model/CGOM\\_v2\\_1.EAP?cvsroot=opendevelopment](http://cabigcvs.nci.nih.gov/viewcvs/viewcvs.cgi/caIntegrator-spec/model/CGOM_v2_1.EAP?cvsroot=opendevelopment). Table 2.1 lists each class and a description. Detailed descriptions about each class and its methods are available in the CGEMS JavaDocs, which are included in the client package on the NCICB Web site.

<b>Class Name</b>	<b>Description</b>
<b>DNASpecimen</b>	A class containing information on the collection and processing of a DNA sample from one of the CGEMS subjects. <b>Note:</b> Currently the CGOM-CGEMS API does not return any data for this object.
<b>Finding</b>	Results obtained from an analysis or discovery (finding) gathered through experimental assays or evaluations. <b>Note:</b> Finding is an abstract class.
<b>GeneBiomarker</b>	A gene-based biological parameter that is indicative of a physiological or pathological state. For example, EBBR2 is a biomarker used to identify risk of breast cancer.
<b>GenotypeFinding</b>	A set of observable characteristics of an individual related to the CGEMS project. <b>Note:</b> Currently the CGOM-CGEMS API does not return any data for this object.
<b>Histology</b>	The result of examination of tissues under the microscope to assist diagnosis of tumors. For example, after a biopsy is performed, a pathologist will perform a “histological” evaluation in which the tissue collected will be analyzed for any abnormalities. <b>Note:</b> Currently the CGOM-CGEMS API does not return any data for this object.
<b>Population</b>	Groups of subjects based on self-described ethnic groupings and phenotypic ascertainment schemes.
<b>SNPAnalysisGroup</b>	Representation of analysis groups such as “CEPH Population” or “Non-Tumor Samples”. <b>Note:</b> Currently the CGOM-CGEMS API does not return any data for this object.
<b>SNPAnnotation</b>	Annotations associated with single nucleotide polymorphisms (SNPs)—places in the genomic sequence where one fraction of the human population has one nucleotide or allele, while another fraction has another.

Table 2.1 CGEMS API classes

<b>Class Name</b>	<b>Description</b>
<b>SNPAssay</b>	Information on the design characteristics of a molecular test for the presence of one or both alleles at a specific SNP locus.
<b>SNPAssociationAnalysis</b>	A set of univariate genetic analyses to detect the association between phenotypic characteristics shared by groups of subjects and their genotypes at a series of SNP loci.
<b>SNPAssociationFinding</b>	Statistical results of evidence for or against genetic association between the phenotypes analyzed at a specific SNP locus.
<b>SNPFrequencyFinding</b>	A class describing counts and characteristics of alleles and genotypes for SNP polymorphisms observed in a CGEMS population.
<b>SNPPanel</b>	A set of SNP genotype assays, typically packaged and performed in a multiplex assay.
<b>Specimen</b>	A part of a thing, or of several things, removed to demonstrate or to determine the character of the whole. For example, a specimen could be a substance or portion of material obtained for use in testing, examination, or study, particularly a preparation of tissue or bodily fluid taken for observation, examination, or diagnosis. <b>Note:</b> Currently the CGOM-CGEMS API does not return any data for this object.
<b>SpecimenBasedMolecularFinding</b>	Results obtained from an analysis or discovery (finding) gathered through experimental assays or evaluations performed on a specimen. <b>Note:</b> SpecimenBasedMolecularFinding is an abstract class.
<b>Study</b>	A type of research activity that tests how well new medical treatments or other interventions work in subjects. Studies test new methods of screening, prevention, diagnosis, or treatment of a disease. They are fully defined in the protocol and may be carried out in a clinic or other medical facility.
<b>StudyParticipant</b>	The treatment arm and other specifics regarding the participation of the subject in a particular study. <b>Note:</b> Currently the CGOM-CGEMS API does not return any data for this object.
<b>TimeCourse</b>	An ordered list of times at which events and activities are planned to occur during a clinical trial. <b>Note:</b> Currently the CGOM-CGEMS API does not return any data for this object.

Table 2.1 CGEMS API classes

<b>Class Name</b>	<b>Description</b>
<b>VariationFinding</b>	The change (variation)—alteration, deletion, or rearrangement—in the DNA sequence that may lead to the synthesis of an altered inactive protein and the loss of the ability to produce the protein. If a mutation occurs in a germ cell, then it is a heritable change; it can be transmitted from generation to generation. Mutations may also be in somatic cells and are not heritable in the traditional sense of the word, but are transmitted to all daughter cells. <b>Note:</b> VariationFinding is an abstract class.

Table 2.1 CGEMS API classes

## Main CGEMS System Components

Table 2.2 provides an overview of the main CGEMS system components

<b>Component</b>	<b>Description</b>
Presentation Layer	Provides a <a href="#">web interface</a> to access the CGEMS API. Using this layer, CGEMS Credentialed and Public users can perform queries and retrieve CGEMS data.
System	Refers to the caIntegrator API that enables search and retrieval of CGEMS data.
Data Repository	Stores all CGEMS data.
Metadata Repository	Used to edit and deploy common data elements (CDEs). The NCI and its partners create, edit, and deploy CDEs using caDSR, the metadata repository for caBIG. These CDEs are used as metadata descriptors for domain objects related to caIntegrator and CGEMS.

Table 2.1 CGEMS system components

# CHAPTER 3

## UNDERSTANDING THE OBJECT QUERY SERVICE API

This chapter introduces you to the Object Query Service API, one of the two CGEMS APIs. The Study Query Service API will be documented in a future chapter of this guide. This chapter includes the following topics:

- [Querying CGEMS Objects](#) on this page
- [Installing and Configuring the Object Query Service API](#) on page 18
- [Using the Object Query Service API](#) on page 21

### Querying CGEMS Objects

---

#### About the Service Layer

The caCORE-SDK architecture that the Object Query Service shares includes a service layer that provides a single, common access paradigm to clients using any of the provided interfaces. As an object-oriented middleware layer designed for flexible data access, caCORE-SDK generated API relies heavily on strongly typed objects and an object-in/object-out mechanism. The methodology used for obtaining data from caCORE-SDK generated systems such as the CGEMS Object Query Service is often referred to as *query by example*, meaning that the inputs to the query methods are themselves domain objects that provide the criteria for the returned data. The major benefit of this approach is that it allows for run-time semantic interoperability and provides shared vocabularies and a metadata registry.

## Accessing the Object Query Service

To access the Object Query Service, follow these steps:

1. Ensure that the client application has knowledge of the objects in the domain space.
2. Build the query using the domain objects.
3. Establish a connection to the server.
4. Submit the query objects and specify the desired class of objects to be returned.
5. Use and manipulate the result set as desired.

## Installing and Configuring the Object Query Service API

---

The Object Query Service API provides direct access to domain objects and all service methods.

To use the Object Query Service API, you should have the software listed in [Table 3.1](#) installed on the client machine.

<b>Software</b>	<b>Version</b>	<b>Required?</b>
<a href="#">Java 2 Platform Standard Edition Software 5.0 Development Kit (JDK 5.0)</a>	1.5.04	Yes
<a href="#">Apache Ant</a>	1.6.2	Yes

*Table 3.1 CGEMS Object Query Service API Client software*

**Note:** You must also have an Internet connection to access the API.

Please acquire each of these and follow the installation instructions provided with each respective product for your environment.

## Downloading and Installing the Client Package

To download the client package from NCICB Web site, follow these steps:

1. Open your browser and navigate to <http://ncicb.nci.nih.gov>.

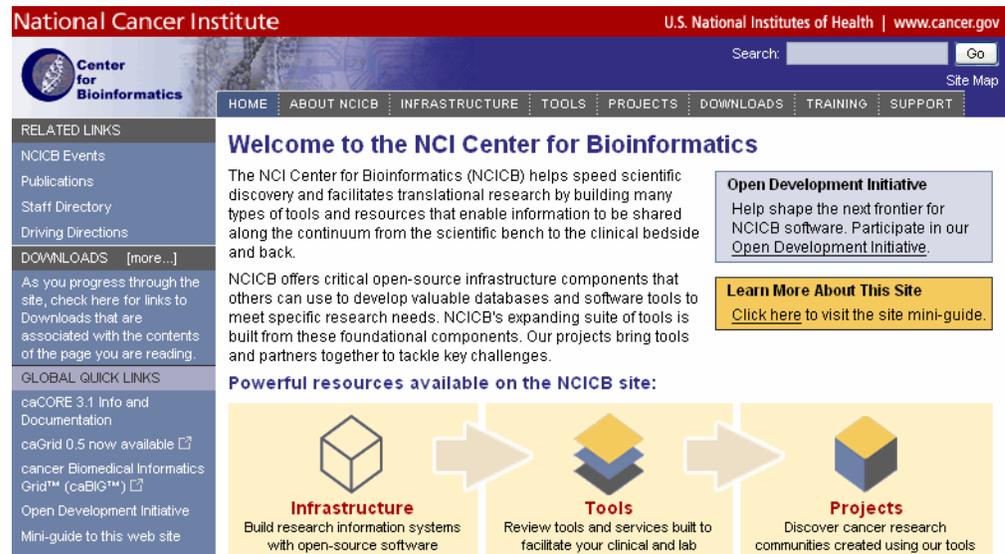


Figure 3.1 Downloads section on the NCICB Web site

2. Click the **Downloads** tab at the top of the page. The Downloads page appears.
3. Click the letter C to jump to the sections with names that start with C.
4. Locate the *CGEMS* section by scrolling, then click the **Download** link. A welcome page appears.
5. Enter your name, e-mail address, and institution name, then click the **Enter the Download Area** button. The license agreement page appears.
6. Accept the license agreement.
7. On the *CGEMS* downloads page, download **cgom-cgems-client.zip** from the Primary Distribution section.
8. Extract the contents of the downloadable archive to a directory on your hard drive (for example, `c:\cgems` on Windows or `/usr/local/cgems` on Linux). The extracted directories and files include the following:

Directories and Files	Description	Component
TestClient.java	Java API client sample	Sample code
build.xml	Ant build file	Build file
log directory	Location of client.log	
lib directory	contains required jar files	
conf directory		

Table 3.2 Extracted directories and files in *CGEMS* client package

All of the *jar* files provided in the *lib* and the *conf* directories of the CGEMS client package are required for using the Object Query Service API. Include these files in the Java classpath when building applications. The *build.xml* file that is included demonstrates how to do this when you are using Ant for command-line builds.

If you are using an integrated development environment (IDE) such as Eclipse, refer to the tool's documentation for information on how to set the classpath.

## Testing the System

To test the system, enter the following URL in your browser to verify all your required system resources are available: <http://cainegrator.nci.nih.gov/cgom-cgems/Happy.jsp>.

The following figure displays the browser window that opens when the system has been properly built.



Figure 3.2 Happy.jsp introductory window

The Happy.jsp page provides a simple query interface that can be used to test the system and ensure that data has been correctly loaded. Perform the following steps to test the system:

Step	Action
1	In the lower-left window, select the Population link. A query page appears in the main window.
2	Enter CASE* in the Name field and click Submit.  A new window appears that displays 3 objects that match the query you submitted. In addition to displaying the attributes of each of these objects, you can also navigate to associated objects by clicking the links in each row.

Table 3.3 How to use Happy.jsp to test the system

## Using the Object Query Service API

---

This section includes a number of examples that demonstrate the use of the caCORE APIs. Included with each example is a brief description of the type of search being performed and the example code accompanied by explanatory text.

### TestClient Example

To run the example program after installing the CGEMS client, open a command prompt or terminal window from the directory where you extracted the downloaded archive and enter `ant rundemo`. This will compile and run the TestClient class; successfully running this example indicates that you have properly installed and configured the caCORE client. The following is a short segment of code from the TestClient class along with an explanation of its functioning.

```

384 @SuppressWarnings("unchecked")
385 private static void searchSNPAssociationFinding() {
386     Collection geneBiomarkerCollection = new ArrayList();
387     GeneBiomarker wt1 = new GeneBiomarker();
388     wt1.setHugoGeneSymbol("WT1");
389     geneBiomarkerCollection.add(wt1);
390
391     SNPAnnotation snpAnnotation = new SNPAnnotation();
392     snpAnnotation.setGeneBiomarkerCollection(geneBiomarkerCollection);
393     try {
394         System.out
395             .println("_____");
396         System.out.println("Retrieving all SNPAssociationFindings for WT1");
397         ApplicationService appService = ApplicationServiceProvider
398             .getApplicationService();
399
400         List resultList = appService.search(SNPAssociationFinding.class,
401             snpAnnotation);
402         if (resultList != null) {
403             System.out.println("Number of results returned: "
404                 + resultList.size());
405             System.out.println("DbsnpId" + "\t" + "ChromosomeName" + "\t"
406                 + "ChromosomeLocation" + "\t" + "GenomeBuild" + "\t"
407                 + "ReferenceSequence" + "\t" + "ReferenceStrand" + "\t"
408                 + "GeneBiomarker(s)" + "\t" + "Analysis Name" + "\t"
409                 + "p-Value" + "\t" + "rank" + "\n");
410             for (Iterator resultsIterator = resultList.iterator(); resultsIterator
411                 .hasNext();) {
412                 SNPAssociationFinding returnedObj = (SNPAssociationFinding) resultsIterator
413                     .next();
414                 System.out.println(returnedObj.getSnpAnnotation()
415                     .getDbSnpId()
416                     + "\t"
417                     + returnedObj.getSnpAnnotation()
418                     .getChromosomeName()
419                     + "\t"
420                     + returnedObj.getSnpAnnotation()
421                     .getChromosomeLocation()
422                     + "\t"
423                     + pipeGeneBiomarkers(returnedObj.getSnpAnnotation()
424                     .getGeneBiomarkerCollection())
425                     + "\t"
426                     + returnedObj.getSnpAssociationAnalysis().getName()
427                     + "\t"
428                     + returnedObj.getPvalue()
429                     + "\t"
430                     + returnedObj.getRank() + "\n");
431             }
432         }
433     } catch (Exception e) {
434         e.printStackTrace();
435     }
436 }

```

This code snippet creates an instance of a class that implements the `ApplicationService` interface. This interface defines the service methods used to access data objects. A criterion object is then created that defines the attribute values for which to search. The search method of the `ApplicationService` implementation is called with parameters that indicate the type of objects to return; for example, `SNPAssociationFinding.class`, and the criteria that returned objects must meet, defined by that object. The search method returns objects in a `List` collection, which is iterated through to print some basic information about the objects.

Although this is a fairly simple example of the use of the Java API, a similar sequence can be followed with more complex criteria to perform sophisticated manipulation of the data provided by CGEMS. Additional information and examples are provided in the sections that follow.

## Service Methods

The methods that provide programmatic access to running the CGEMS caCORE Object Query API server are located in the `gov.nih.nci.system.application-service` package. The `ApplicationServiceProvider` class uses the factory design pattern to return an implementation of the `ApplicationService` interface. The provider class determines whether there is a locally running instance of the caCORE system or whether it should use a remote instance. The returned `ApplicationService` implementation exposes the service methods that enable read/write operations on the domain objects

The separation of the service methods from the domain classes is an important architectural decision that insulates the domain object space from the underlying service framework. As a result, new business methods can be added without needing to update any of the domain model or the associated metadata information from the object model. (This is critical for ensuring semantic interoperability over multiple iterations of architectural changes.) Within the `ApplicationService` implementation, a variety of methods are provided allowing users to query data based on the specific needs and types of queries to be performed. In general, there are four types of searches:

- **Simple searches** are those that take one or more objects from the domain models as inputs and return a collection of objects from the data repositories that meet the criteria specified by the input objects.
- **Nested searches** also take domain objects as inputs but determine the type of objects in the result set by traversing a known path of associations from the domain model.
- **Detached criteria searches** use Hibernate detached criteria objects to provide a greater level of control over the results of a search (such as boolean operations, ranges of values, etc.)
- **HQL searches** provide the ability to use the Hibernate Query Language for the greatest flexibility in forming search criteria.

<b>Method Signature</b>	<code>List search(     Class targetClass,     Object obj)</code>
<b>Search Type</b>	Simple (One criteria object)
<b>Description</b>	Returns a List collection containing objects of type targetClass that conform to the criteria defined by obj
<b>Example</b>	<code>search(Study.class, study);</code>

<b>Method Signature</b>	<code>List search(     Class targetClass,     List objList)</code>
<b>Search Type</b>	Simple (Criteria object collection)
<b>Description</b>	Returns a List collection containing objects of type targetClass that conform to the criteria defined by a collection of objects in objList. The returned objects must meet ANY criteria in objList (i.e. a logical OR is performed).
<b>Example</b>	<code>search(GeneBiomarker.class)</code>

<b>Method Signature</b>	<code>List search(     String path,     Object obj)</code>
<b>Search Type</b>	Nested
<b>Description</b>	Returns a List collection containing objects conforming to the criteria defined by obj and whose resulting objects are of the type reached by traversing the node graph specified by path
<b>Example</b>	<code>search("gov.nih.nci.caintegrator.domain.annotation.snp.SNPAssay", snpAnnotation)</code>

<b>Method Signature</b>	<code>List search(     String path,     List objList)</code>
<b>Search Type</b>	Nested
<b>Description</b>	Returns a List collection containing objects conforming to the criteria defined by the objects in objList and whose resulting objects are of the type reached by traversing the node graph specified by path
<b>Example</b>	<code>search("geneBiomarkerCollection", gov.nih.nci.caintegrator.domain.annotation.snp.SNP Assay+gov.nih.nci.caintegrator.domain.annotation.s np.SNPAnnotation)</code>

<b>Method Signature</b>	List query( DetachedCriteria detachedCriteria, String targetClassName)
<b>Search Type</b>	Detached criteria
<b>Description</b>	Returns a List collection conforming to the criteria specified by detachedCriteria and whose resulting objects are of the type specified by targetClassName
<b>Example</b>	query(criteria, "SNPAnnotation.class.getName()")

<b>Method Signature</b>	List query( Object criteria, int firstRow, int resultsPerQuery, String targetClassName)
<b>Search Type</b>	Detached criteria
<b>Description</b>	Identical to the previous query method, but allows for control over the size of the result set by specifying the row number of the first row and the maximum number of objects to return
<b>Example</b>	query(criteria, 101, 100, targetClassName)

<b>Method Signature</b>	List query( HQLCriteria hqlCriteria, String targetClassName)  Search Type HQL
<b>Description</b>	Returns a List collection of objects of the type specified by targetClassName that conform to the query in HQL syntax contained in hqlCriteria
<b>Example</b>	query(hqlCriteria, SNPAnnotation.class .getName() )

In addition to the data access methods, several helper methods are available via the ApplicationService class that provide flexibility in controlling queries and result sets.

## Scenario One: Retrieve All SNPPanels

In this example, an unrestricted search is performed for all SNPPanels.

```

089 private static void searchSNPPanel() {
090     SNPPanel snpPanel = new SNPPanel();
091     try {
092         System.out
093             .println("_____");
094         System.out.println("Retrieving all SNPPanels...");
095         ApplicationService appService = ApplicationServiceProvider
096             .getApplicationService();
097
098         List resultList = appService.search(SNPPanel.class, snpPanel);
099         if (resultList != null) {
100             System.out.println("Number of results returned: "
101                 + resultList.size());
102             for (Iterator resultsIterator = resultList.iterator(); resultsIterator
103                 .hasNext();) {
104                 SNPPanel returnedObj = (SNPPanel) resultsIterator.next();
105                 System.out.println("Panel Name: " + returnedObj.getName()
106                     + "\n" + "Description: "
107                     + returnedObj.getDescription() + "\n"
108                     + "Technology: " + returnedObj.getTechnology()
109                     + "\n" + "Vendor: " + returnedObj.getVendor()
110                     + "\n" + "Vendor PanelId: "
111                     + returnedObj.getVendorPanelId() + "\n"
112                     + "Version: " + returnedObj.getVersion() + "\n");
113             }
114         }
115     } catch (Exception e) {
116         e.printStackTrace();
117     }
118 }

```

<b>Lines</b>	<b>Description</b>
95	Creates an instance of a class that implements the ApplicationService interface; this interface defines the service methods used to access data objects
98	Calls the search method of the ApplicationService implementation and passes it the type of objects to return, SNPPanel.class, and the criteria that returned objects must meet, defined by the SNPPanel object; the search method returns objects in a List collection
104	Casts an object from the result List and creates a variable reference to it of type SNPPanel.
105	Prints the SNPPanel attribute
106	Prints the Description attribute
109	Prints the Technology attribute
110	Prints the Vendor Panel ID attribute
111	Prints the Vendor attribute
112	Prints the Version attribute

## Scenario Two: Simple Search (Criteria Object Collection) to retrieve SNPFrequencyFinding for the Gene “WT1”

In this example, a search is performed for WT1 genes to retrieve the SNPFrequencyFinding. The code iterates through the returned objects and prints out the several properties of each of the object, as shown in the code listing.

```

245 @SuppressWarnings( { "unused", "unchecked" })
246 private static void searchSNPFrequencyFinding() {
247     Collection geneBiomarkerCollection = new ArrayList();
248     GeneBiomarker wt1 = new GeneBiomarker();
249     wt1.setHugoGeneSymbol("WT1");
250     geneBiomarkerCollection.add(wt1);
251
252     SNPAnnotation snpAnnotation = new SNPAnnotation();
253     snpAnnotation.setGeneBiomarkerCollection(geneBiomarkerCollection);
254
255     SNPFrequencyFinding snpFrequencyFinding = new SNPFrequencyFinding();
256     snpFrequencyFinding.setSnpAnnotation(snpAnnotation);
257     try {
258         System.out
259             .println("_____");
260         System.out
261             .println("Retrieving all SNPFrequencyFinding objects for WT1");
262         ApplicationService appService = ApplicationServiceProvider
263             .getApplicationService();
264
265         List resultList = appService.search(SNPFrequencyFinding.class,
266             snpAnnotation);
267         if (resultList != null) {
268             System.out.println("Number of results returned: "
269                 + resultList.size());
270             System.out.println("DbsnpId" + "\t" + "ChromosomeName" + "\t"
271                 + "ChromosomeLocation" + "\t" + "MinorAlleleFrequency"
272                 + "\t" + "HardyWeinbergPValue" + "\t"
273                 + "ReferenceAllele" + "\t" + "OtherAllele" + "\t"
274                 + "Population" + "\n");
275             for (Iterator resultsIterator = resultList.iterator(); resultsIterator
276                 .hasNext();) {
277                 SNPFrequencyFinding returnedObj = (SNPFrequencyFinding) resultsIterator
278                     .next();
279                 System.out.println(returnedObj.getSnpAnnotation()
280                     .getDbSnpId()
281                     + "\t"
282                     + returnedObj.getSnpAnnotation()
283                     .getChromosomeName()
284                     + "\t"
285                     + returnedObj.getSnpAnnotation()
286                     .getChromosomeLocation()
287                     + "\t"
288                     + returnedObj.getMinorAlleleFrequency()
289                     + "\t"
290                     + returnedObj.getHardyWeinbergPValue()
291                     + "\t"
292                     + returnedObj.getReferenceAllele()
293                     + "\t"
294                     + returnedObj.getOtherAllele()
295                     + "\t"
296                     + returnedObj.getPopulation().getName() + "\n");
297             }
298         }
299     } catch (Exception e) {
300         e.printStackTrace();
301     }
302 }

```

<b><i>Lines</i></b>	<b><i>Description</i></b>
247-250	Creates a GeneBiomarker object and sets the hugoGeneSymbol to "WT1"
250-253	Because the SNPAnnotation and GeneBiomarker classes are related by a many-to-many association, it is necessary to create a collection to contain the GeneBiomarker object that will act as part of the compound criteria; multiple GeneBiomarker objects could be added to this collection as needed
255-256	Creates a SNPAnnotation object and sets the value of its setGeneBiomarkerCollection method to the geneBiomarkerCollection object just created
265	Searches for all SNPAnnotation objects whose geneBiomarkerCollection contains objects that match the set criteria (i.e. the symbol is "WT1")

## Scenario Three: Nested Search to retrieve SNPAssays based on dbSnpId

A nested search is one where a traversal of more than one class-class association is required to obtain a set of result objects given the criteria object. This example demonstrates one such search in which the criteria object passed to the search method is of type SNPAnnotation, and the desired objects are of type SNPAssay.

```

312 @SuppressWarnings( { "unused", "unchecked" })
313 private static void searchSNPAssay() {
314     SNPAnnotation snpAnnotation = new SNPAnnotation();
315     snpAnnotation.setDbsnpId("rs5030335");
316     SNPAssay snpAssay = new SNPAssay();
317     snpAssay.setSnpAnnotation(snpAnnotation);
318     try {
319         System.out
320             .println("_____");
321         System.out.println("Retrieving all SNPAssay objects for rs5030335");
322         ApplicationService appService = ApplicationServiceProvider
323             .getApplicationService();
324
325         List resultList = appService.search(SNPAssay.class, snpAnnotation);
326         if (resultList != null) {
327             System.out.println("Number of results returned: "
328                 + resultList.size());
329             System.out.println("Vendor Assay ID" + "\t" + "DbsnpId" + "\t"
330                 + "ChromosomeName" + "\t" + "ChromosomeLocation" + "\t"
331                 + "SNP Panel" + "\t" + "Version" + "\t"
332                 + "DesignAlleles" + "\t" + "Status" + "\n");
333             for (Iterator resultsIterator = resultList.iterator(); resultsIterator
334                 .hasNext();) {
335                 SNPAssay returnedObj = (SNPAssay) resultsIterator.next();
336                 System.out.println(returnedObj.getVendorAssayId()
337                     + "\t"
338                     + returnedObj.getSnpAnnotation().getDbsnpId()
339                     + "\t"
340                     + returnedObj.getSnpAnnotation()
341                         .getChromosomeName()
342                     + "\t"
343                     + returnedObj.getSnpAnnotation()
344                         .getChromosomeLocation() + "\t"
345                     + returnedObj.getSnpPanel().getName() + "\t"
346                     + returnedObj.getVersion() + "\t"
347                     + returnedObj.getDesignAlleles() + "\t"
348                     + returnedObj.getStatus() + "\n");
349             }
350         }
351     } catch (Exception e) {
352         e.printStackTrace();
353     }
354 }
355

```

Lines	Description
314-317	Creates a SNPAnnotation object and sets the dbsnpId to "rs5030335"
325	Defines search path as traversing from the criteria object of type SNPAnnotation to SNPAssay; note that the first element in the path is the desired class of objects to be returned, and that subsequent elements traverse back to the criteria object
325	Sets the criteria object to the previously-created SNPAnnotation

## Scenario Four: Detached Criteria Search

This example demonstrates the use of Hibernate detached criteria objects to formulate and perform more sophisticated searches. A detailed description of detached criteria is beyond the scope of this document; for more information, please consult the Hibernate documentation at [http://www.hibernate.org/hib\\_docs/v3/api/org/hibernate/criterion/DetachedCriteria.html](http://www.hibernate.org/hib_docs/v3/api/org/hibernate/criterion/DetachedCriteria.html).

```

444 @SuppressWarnings("unused")
445 private static void searchSNPAnnotation() {
446     DetachedCriteria criteria = DetachedCriteria
447         .forClass(SNPAnnotation.class);
448     criteria.add(Restrictions
449         .ge("chromosomeLocation", new Integer(400000)));
450     criteria.add(Restrictions
451         .le("chromosomeLocation", new Integer(420000)));
452     criteria.add(Restrictions.eq("chromosomeName", "1"));
453     try {
454         System.out
455             .println("_____");
456         System.out
457             .println("Retrieving all SNPAnnotations for Chr 1,400000 - 420000");
458         ApplicationService appService = ApplicationServiceProvider
459             .getApplicationService();
460
461         List resultList = appService.query(criteria, SNPAnnotation.class
462             .getName());
463         if (resultList != null) {
464             System.out.println("Number of results returned: "
465                 + resultList.size());
466             System.out.println("DbsnpId" + "\t" + "ChromosomeName" + "\t"
467                 + "ChromosomeLocation" + "\t" + "GenomeBuild" + "\t"
468                 + "ReferenceSequence" + "\t" + "ReferenceStrand" + "\t"
469                 + "GeneBiomarker(s)" + "\n");
470             for (Iterator resultsIterator = resultList.iterator(); resultsIterator
471                 .hasNext();) {
472                 SNPAnnotation returnedObj = (SNPAnnotation) resultsIterator
473                     .next();
474                 System.out.println(returnedObj.getDbsnpId()
475                     + "\t"
476                     + returnedObj.getChromosomeName()
477                     + "\t"
478                     + returnedObj.getChromosomeLocation()
479                     + "\t"
480                     + returnedObj.getGenomeBuild()
481                     + "\t"
482                     + returnedObj.getReferenceSequence()
483                     + "\t"
484                     + returnedObj.getReferenceStrand()
485                     + "\t"
486                     + pipeGeneBiomarkers(returnedObj
487                         .getGeneBiomarkerCollection()) + "\n");
488             }
489         }
490     } catch (Exception e) {
491         e.printStackTrace();
492     }
493 }

```

<b><i>Lines</i></b>	<b><i>Description</i></b>
446	Creates an DetachedCriteria object and sets the class on which the criteria will operate to SNPAnnotation
448	Sets a restriction on the objects that states that the attribute chromosomeLocation must be greater than or equal to ("ge") the value 4000000
450	Sets a restriction on the objects that states that the attribute chromosomeLocation must be less than or equal to ("le") the value 4200000
452	Sets a restriction on the objects that states that the attribute chromosomeName must be equal to ("eq") the value 1
461	Calls the query method of the ApplicationService implementation, specifying the desired object type to return, SNPAnnotation, and passing the detached criteria object

## Scenario Five: HQL Search

This example demonstrates the use of HQL to retrieve SNPAssay, whose ID is less than 100. It uses a Hibernate Query Language (HQL) search string to form the query. For more information on HQL syntax, consult the Hibernate documentation at [http://www.hibernate.org/hib\\_docs/v3/reference/en/html/queryhql.html](http://www.hibernate.org/hib_docs/v3/reference/en/html/queryhql.html).

```

502 private static void searchSNPAssayHQL() {
503     String hqlString = "FROM SNPAssay a WHERE a.id < 100";
504     HQLCriteria hqlC = new HQLCriteria(hqlString);
505     try {
506         System.out
507             .println("_____");
508         System.out.println("Retrieving all SNPAssay objects, id < 100");
509         ApplicationService appService = ApplicationServiceProvider
510             .getApplicationService();
511         List resultList = appService.query(hqlC, SNPAnnotation.class
512             .getName());
513         if (resultList != null) {
514             if (resultList != null) {
515                 System.out.println("Number of results returned: "
516                     + resultList.size());
517                 System.out.println("Id\t" + "Vendor Assay ID" + "\t"
518                     + "SNP Panel" + "\t" + "Version" + "\t"
519                     + "DesignAlleles" + "\t" + "Status" + "\n");
520                 for (Iterator resultsIterator = resultList.iterator(); resultsIterator
521                     .hasNext();) {
522                     SNPAssay returnedObj = (SNPAssay) resultsIterator
523                         .next();
524                     System.out.println(returnedObj.getId() + "\t"
525                         + returnedObj.getVendorAssayId() + "\t"
526                         + returnedObj.getSnpPanel().getName() + "\t"
527                         + returnedObj.getVersion() + "\t"
528                         + returnedObj.getDesignAlleles() + "\t"
529                         + returnedObj.getStatus() + "\n");
530                 }
531             }
532         }
533     } catch (Exception e) {
534         e.printStackTrace();
535     }
536 }

```

<i>Lines</i>	<i>Description</i>
503	Creates a string that contains the query in HQL syntax
504	Instantiates an HQLCriteria object and sets the query string
511	Calls the query method of the ApplicationService implementation and passes it the HQLCriteria object and the type of objects to return

## APPENDIX

# A

## UML MODELING

The CGEMS team bases its software development primarily on the Unified Modeling Language (UML). In case you have not worked with UML, this appendix will familiarize you with UML background and notation.

The following topics are included in this appendix:

- *UML Modeling* on this page
- *Use Case Documents and Diagrams* on page 34
- *Class Diagrams* on page 37
- *Relationships Between Classes* on page 38
- *Sequence Diagrams* on page 40

### UML Modeling

---

The UML is an international standard notation for specifying, visualizing, and documenting the artifacts of an object-oriented software development system. Defined by the Object Management Group, <http://www.omg.org/>, the UML emerged as the result of several complementary systems of software notation and has now become the *de facto* standard for visual modeling.

For a brief tutorial on UML, refer to <http://bdn.borland.com/article/0.1410.31863.00.html>.

The underlying tenet of any object-oriented programming begins with the construction of a model. The UML comprises nine different types of modeling diagrams that form a software blueprint.

The following subset of UML diagrams is used in CGEMS development:

- Use case diagrams
- Class diagrams
- Sequence diagrams

The CGEMS 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, the team uses class diagrams to define static attributes, functionalities, and relations that must be implemented.

As design progresses, the team uses other types of interaction diagrams to capture the dynamic behaviors and cooperative activities that the objects must execute. Finally, the team uses additional diagrams such as package and sequence diagrams to represent pragmatic information, including the physical locations of source modules and the allocations of resources.

Each type of diagram 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.

While many development tools provide support for generating UML diagrams, the CGEMS development team uses Enterprise Architect (EA).

## Use Case Documents and Diagrams

---

A good starting point for capturing system requirements is to develop a structured textual description, often called a *use case*, of how users will interact with the system. While there is no predefined structure for this artifact, use case documents typically consist of one or more actors, a process, a list of steps, and a set of pre- and post-conditions. In many cases, these documents describe the post-conditions associated with success, as well as failure. An example use case document is represented in [Table A.1](#).

Using the use case document as a model, a use case diagram is created to confirm the requirements.

<b>Use Case Name</b>	Perform SNP Associated Finding Search
<b>Use Case ID</b>	3.1
<b>Primary Actor</b>	Researcher via Presentation Layer
<b>Trigger</b>	Researcher has logged into the system.
<b>Pre-conditions</b>	Presentation Layer has authenticated the user.
<b>Flow of Events</b>	<ol style="list-style-type: none"> <li>1. Presentation Layer allows researcher to search for SNP Associated Finding based on the following: <ol style="list-style-type: none"> <li>a. p-value</li> <li>b. rank</li> <li>c. Analysis Group Names list</li> <li>d. Analysis Method list</li> <li>e. Perform SNP search use case</li> <li>f. Perform Study search use case</li> </ol> </li> <li>2. Researcher completes a list of search fields. Field values are joined using AND to create query criteria.</li> <li>3. The displayed search fields are registered in the caBIG metadata repository as part of caBIG compliance.</li> <li>4. Researcher enters the fields to be searched and the condition for search (if any).</li> <li>5. Researcher clicks the <b>Submit</b> button</li> <li>6. The system does the following: <ol style="list-style-type: none"> <li>a. Populates user selections to formulate the query criteria.</li> <li>b. Validates the data entered.</li> <li>c. If no exceptions occur, displays the search results</li> </ol> </li> </ol>
<b>Post-conditions</b>	<p><b>Success Condition:</b> Researcher sees the search results screen to view or download the results.</p> <p><b>Error Condition:</b> Researcher receives an Invalid Data or Incomplete Data message.</p> <p><b>Error Condition:</b> Researcher receives a system error while processing the search query.</p>

Table A.1 Example Use Case

<p><b>Alternative Flow</b></p>	<ol style="list-style-type: none"> <li>1. If a validation error occurs, the system displays the appropriate error and redisplay the page.</li> <li>2. The actor does either of the following:             <ol style="list-style-type: none"> <li>a. Adds additional data, edits entered data, or clears the screen and re-enters search criteria.</li> <li>b. Logs out of the system and terminates the process.</li> </ol> </li> <li>3. One of the following occurs:             <ol style="list-style-type: none"> <li>a. If system error occurs, the actor receives a message to contact the system administrator to report the error.</li> <li>b. If the query returns no data, the system displays the appropriate error and redisplay the page.</li> </ol> </li> <li>4. The actor does either of the following:             <ol style="list-style-type: none"> <li>a. Adds additional data, changes entered data, or clears the screen and re-enters all data.</li> <li>b. Logs out of the system and terminates the process.</li> </ol> </li> </ol>
<p>Related Use Case</p>	<p>3.2 Perform Study Search 3.3 Perform SNP Search</p>

*Table A.1 Example Use Case*

A use case diagram, which is language independent and graphically described, uses simple ball and stick figures with labeled ellipses and arrows to show how users or other software agents might interact with the system. The emphasis is on what a system does rather than how. Each use case (an ellipse) describes a particular activity that an actor (a stick figure) performs or triggers. The communications between actors and use cases are depicted by connecting lines or arrows.

## Class Diagrams

The system designer uses use case diagrams to identify classes that must be implemented in the system, their attributes and behaviors, and the relationships and co-operative activities that must be realized. A class diagram is used later in the design process to give an overview of the system, showing the hierarchy of classes and their static relationships at varying levels of detail. *Figure A.1* shows an abbreviated version of a UML Class diagram depicting the Apache ObjectRelationalBridge (OBJ) abstraction layer and DAO classes.

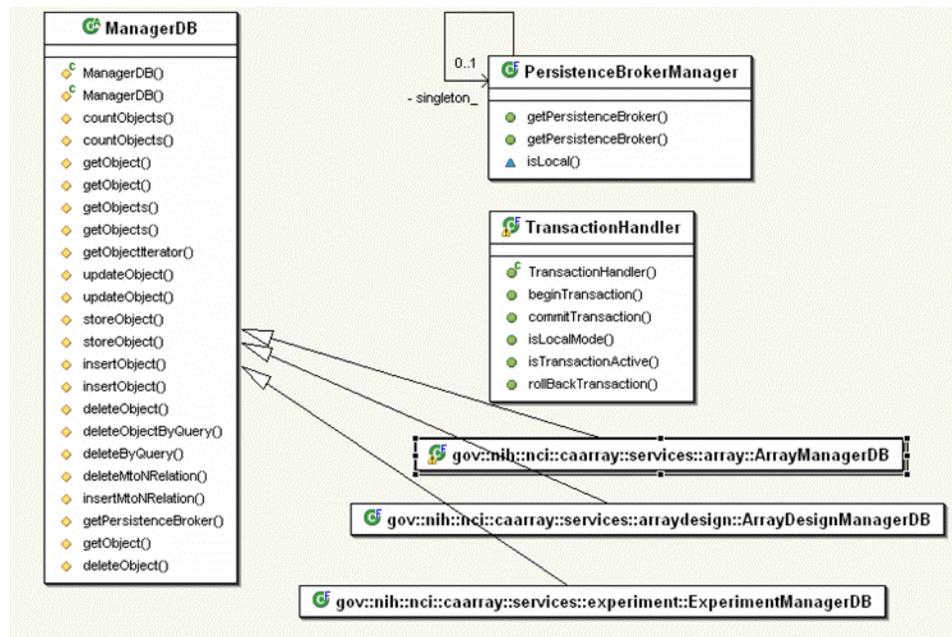


Figure A.1 OBJ Abstraction Layer and DAO Classes

Class objects can have a variety of possible relationships, including *is derived from*, *contains*, *uses*, or *is associated with*. 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 required to interpret new software specifications and to learn how to navigate in a new programming environment.

*Figure A.2 (a)* is a schematic for a UML class representation, the fundamental element of a class diagram. *Figure A.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 (structures) for the class. The attribute in the class diagram maps to a column name in the data model and an attribute within the Java class. The bottom section lists the object's operations

(methods). *Figure A.2* (b) specifies the Gene class as having a single attribute called *sequence* and a single operation called *getSequence()*:



*Figure A.2* (a) Schematic for a UML class (b) Simple Gene class

Naming conventions are very important when you are creating class diagrams. CGEMS follows the formatting convention for Java APIs: a class starts with an uppercase letter and an attribute starts with a lowercase letter. Names contain no underscores. If the name contains two words, then both words are capitalized, with no space between them. If an attribute contains two words, then the second word is capitalized with no space between words. Boolean terms (has, is) are used as prefixes to words for test cases.

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, and UML tools vary with respect to how they represent their private, public, and protected notations for class diagrams.

The classes represented in *Figure A.1* show only class names and attributes. The operations are suppressed in that diagram. This is an example of a UML view. Details are hidden where they might obscure the bigger picture that the diagram is intended to convey. Most UML design tools provide a means for selectively suppressing either or both attributes and operation compartments of the class without removing the information from the underlying design model.

The following notations (as shown in *Figure A.2*) are used to indicate that a feature is public or private:

- A hyphen (-) prefix signifies a private feature.
- A plus sign (+) signifies a public feature.

In *Figure A.2*, for example, the Gene object's *sequence* attribute is private and can only be accessed using the public *getSequence ()* method.

## Relationships Between Classes

*Figure A.3* illustrates the following relationships between classes:

- **Association:** The most primitive of the relationships. Represents the ability of one instance to send a message to another instance. Association is depicted by a simple solid line connecting two classes.
- **Directionality:** Sometimes called navigability. Here, a *Gene* object is uniquely associated with a *Taxon* object, with an arrow denoting bi-directional navigability. Specifically, the *Gene* object has access to the *Taxon* object (i.e., there is a *getTaxon()* method), and the *Taxon* object has access to the *Gene* object (there is a corresponding *getGeneCollection()* method). *Figure A.3*

displays role names, clarifying the nature of the association between the two classes. For example, a *taxon* (role name identified in [Figure A.3](#)) is a line item of each *Gene* object. The (+) indicates public accessibility. The *Bi-directional association* label indicates that the association is bidirectional.

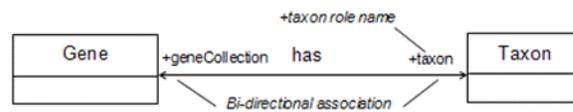


Figure A.3 One-to-one association

- Multiplicity:** 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. [Table A.1](#) displays the most commonly used multiplicities.

Multiplicities	Interpretation
0..1	Zero or one instance. The notation $n..m$ indicates $n$ to $m$ instances.
0..* or *	Zero to many; No limit on the number of instances (including none). An asterisk (*) is used to represent a multiplicity of many.
1	Exactly one instance
1..*	At least one instance to many

Table A.1 Commonly used multiplicities

- Aggregation:** The relationship is between a whole and its parts. 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). Aggregation is represented by a line with a diamond end next to the class representing the whole, as shown in the Clone-to-Library relation of [Figure A.4](#). As illustrated, a Library can contain Clones, but not vice-versa.

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.

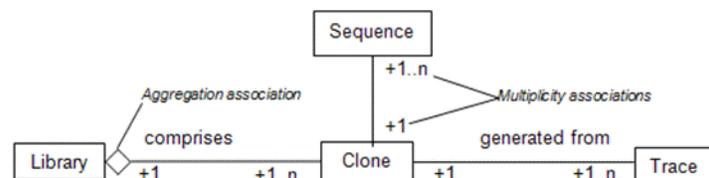


Figure A.4 Aggregation and multiplicity

*Figure A.4* shows a more complex network of relations. This diagram indicates the following:

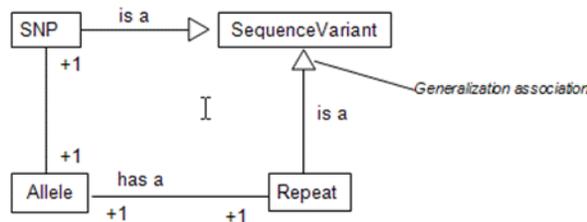
- One or more sequences is associated with a Clone
- The Clone is contained in a Library, which comprises one or more Clones
- The Clone may have one or more Traces.

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

- Generalization:** An inheritance link indicating that one class is a subclass of another. *Figure A.5* 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.

In generalization, 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 represented by a connecting line with an empty arrowhead at its end pointing to the superclass, as shown in the SequenceVariant-to-Repeat and SequenceVariant-to-SNP relations of *Figure A.5*.



*Figure A.5* 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 for 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 is referencing classes, it is representing the static structure.

## Sequence Diagrams

A sequence diagram describes the exchange of messages being passed from object to object. The flow of logic within a system is modeled visually, validating the logic of a usage scenario. In a sequence diagram, bottlenecks can be detected within an object-oriented design, and complex classes can be identified.

*Figure A.6* is an example of a DTO sequence diagram. The vertical lines in the diagram with the boxes along the top row represent instantiated objects. The vertical dimension displays the sequence of messages in the time order that they occur; the horizontal dimension shows the object instances to which the messages are sent. Read the diagram from left to right, top to bottom, following the sequential execution of events.

The DTO sequence diagram (*Figure A.6*) includes the following:

- The application client sets user-entered values in the ProtocolData Transfer Object.
- The client application then invokes the EJB method to add protocol, sending the Transfer Object by value.

The EJB method then retrieves all user-entered values from the Transfer Object and begins business processing.

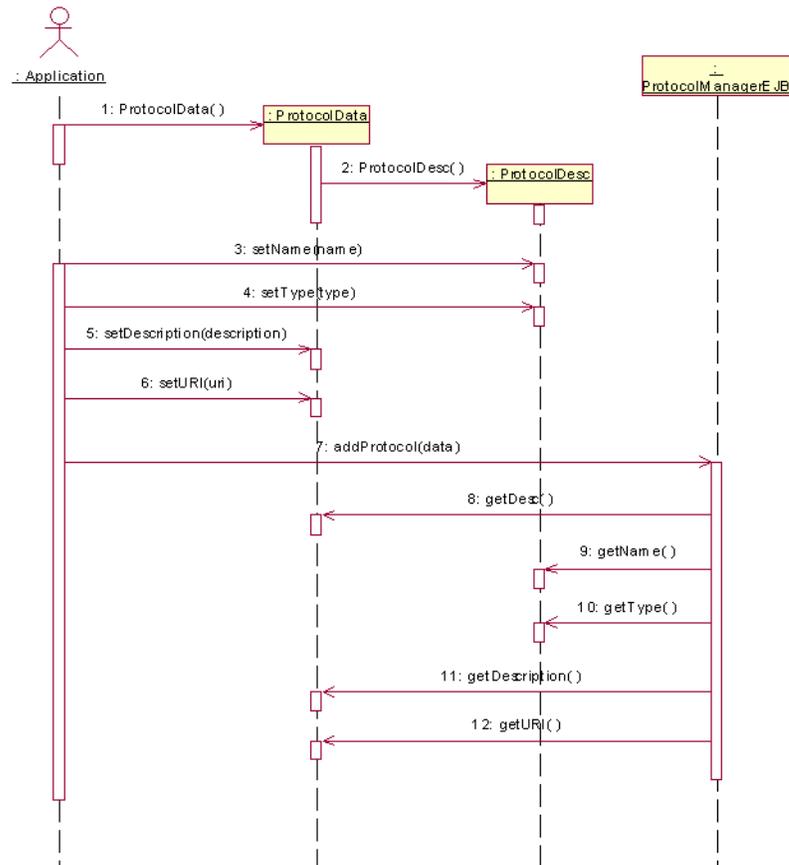


Figure A.6 DTO sequence diagram



# APPENDIX B

## CGEMS GLOSSARY

This glossary describes acronyms, objects, tools, and other terms referenced in the chapters or appendixes of the *CGEMS Technical Guide*.

<b>Term</b>	<b>Definition</b>
API	Application Programming Interface
caBIG	Cancer Biomedical Informatics Grid
caBIO	Cancer Bioinformatics Infrastructure Objects
caCORE	Cancer Common Ontologic Representation Environment
caDSR	Cancer Data Standards Repository
CGEMS	Cancer Genetic Markers of Susceptibility
caMOD	Cancer Models Database
CGF	Core Genotyping Facility
CGH	Comparative Genomic Hybridization
EBI	European Bioinformatics Institute
EVS	Enterprise Vocabulary Services
MAGE 1.1	A widely used microarray data standard or guideline
MAGE-ML software format	XML-based standard for representation of microarray data
MIAME 1.1	A standard or guideline for the minimum amount of information required to make a microarray record useful to others.
MGED Ontology	A controlled vocabulary standard that concisely defines terms as they relate to Microarrays and caArray as a whole
MGED	Microarray Gene Expression Data Society
MMHCC	Mouse Models of Human Cancers Consortium
NCI	National Cancer Institute

<b>Term</b>	<b>Definition</b>
NCICB	National Cancer Institute Center for Bioinformatics
OJB	Apache ObJectRelationalBridge (OJB) is an Object/Relational mapping tool that allows transparent persistence for Java Objects against relational databases.
URI	Uniform Resource Identifier
URL	Uniform Resource Locators
XML	Extensible Markup Language ( <a href="http://www.w3.org/TR/REC-xml/">http://www.w3.org/TR/REC-xml/</a> )  XML is a subset of the Standard Generalized Markup Language (SGML). Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

# INDEX

## A

- Application Service layer 9
- Architecture
  - layers 9
- Association, described 38

## C

- Capturing system requirements 34
- Cardinality 39
- Class diagrams
  - described 37, 38
  - fundamental elements 37
  - naming conventions 38
  - private feature 38
  - public feature 38

## D

- Data Source Delegation layer 9
- Directionality 38

## H

- Happy.jsp 20
- Hibernate 9
- Hibernate Query Language 9

## M

- Multiplicity 39

## N

- Naming conventions, class diagrams 38
- Navigability 38

## O

- Object Query Service API
  - configuration 18
  - description 18
  - installation 18
  - testing 20
- Object-Relational Mapping 9

## P

- Private feature 38
- Public feature 38

## R

- Relationships in class diagrams
  - aggregation 39
  - association 38
- Role names
  - defined 39

## S

- Scenario
  - Detached Criteria Search 30
  - HQL Search 32
  - Nested Search to retrieve SNPAssays based on dbSnid 29
  - Retrieve All SNPPanels 26
  - Simple Search (Criteria Object Collection) to retrieve SNPFrequencyFinding for the Gene "WT1" 27
- Sequence diagrams
  - described 40
  - example 40

## T

- TestClient 21

## U

- UML
  - class diagrams 37
  - introduction 33
  - sequence diagrams 40
  - tutorial 33
  - types of diagrams 33
  - use case, documents and diagrams 35

