**caCORE SDK Developers Guide 4.0**

**Credits and Resources**

|  |  |  |  |
| --- | --- | --- | --- |
| caCORE SDK Contributors | | | |
| SDK Development Team | Other Development Teams | Guide | Program Management |
| Satish Patel 1 | Kunal Modi 1 | Satish Patel 1 | Denise Warzel 4 |
| Dan Dumitru 1 | Vijay Parmar 1 | Dan Dumitru 1 | Avinash Shanbhag 4 |
| Aynur Abdurazik 2 | Shaziya Muhsin 2 | Charles Griffin 1 | George Komatsoulis 4 |
|  | Konrad Rokicki 2 | Wendy Erickson-Hirons5 | Charles Griffin 1 |
|  | Ye Wu 2 |  |  |
|  | Christophe Ludet 3 |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| 1 Ekagra Software Technologies | 2 Science Applications International Corporation (SAIC) | 3 Oracle Corporation | 4 National Cancer Institute Center for Bioinformatics |
| 5 Northern Taiga Ventures, Inc. |  |  |  |

**SDK Resources**

|  |  |
| --- | --- |
| Resource | Location |
| Mailing List | [CACORESDK\_USERS-L@mail.nih.gov](mailto:CACORESDK_USERS-L@mail.nih.gov) |
| Mailing List Archive | [https://list.nih.gov/archives/ cacore\_sdk\_users-l.html](https://list.nih.gov/archives/%20cacore_sdk_users-l.html) |
| Project Home (GForge) | <https://gforge.nci.nih.gov/projects/cacoresdk/> |
| SDK Support Tracker (GForge) | <https://gforge.nci.nih.gov/tracker/?group_id=148&atid=731> |

**Submitting a Support Issue**

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

**Contacting Technical Support**

Technical support is available by contacting the **NCICB Application Support** group. There contact information is provided below:

|  |  |
| --- | --- |
| **NCICB Application Support** | <http://ncicb.nci.nih.gov/NCICB/support>  Telephone: 301-451-4384  Toll free: 888-478-4423 |

Contents

[1 Introduction 8](#_Toc177554208)

[1.1 SDK within caCORE environment 9](#_Toc177554209)

[1.2 How can I get benefited by using caCORE SDK<<?>> 9](#_Toc177554210)

[1.3 How to Obtain caCORE SDK 10](#_Toc177554211)

[1.4 System requirement 10](#_Toc177554212)

[1.4.1 caCORE SDK Minimal System Requirements 10](#_Toc177554213)

[1.4.2 caCORE SDK Software and Technology Requirements 11](#_Toc177554214)

[1.5 Contributing Towards Development of caCORE SDK? 11](#_Toc177554215)

[2 Code Generation - Technical Overview 12](#_Toc177554216)

[2.1 Code Generation 12](#_Toc177554217)

[2.1.1 What is Code Generation? 12](#_Toc177554218)

[2.1.2 Code Generation’s Role in caCORE SDK 12](#_Toc177554219)

[2.1.3 Features and Limitations of Code Generation 13](#_Toc177554220)

[2.2 Code Generation Process in SDK 13](#_Toc177554221)

[2.2.1 SDK Code Generation Process 13](#_Toc177554222)

[2.2.2 Read UML Model 14](#_Toc177554223)

[2.2.3 Artifact Generation (Model Transformation) 15](#_Toc177554224)

[2.2.4 Output Management 15](#_Toc177554225)

[2.2.5 Code Generation Framework 15](#_Toc177554226)

[2.2.6 Reusable Components of the Code Generation Workflow 17](#_Toc177554227)

[2.3 Overview of SDK Generated Artifacts<<?>> 17](#_Toc177554228)

[3 Runtime System - Technical Overview 19](#_Toc177554229)

[3.1 High-level Architecture 19](#_Toc177554230)

[3.2 n-tier system 21](#_Toc177554231)

[3.2.1 Persistence Tier 21](#_Toc177554232)

[3.2.1.1 Object Relational Mapping 22](#_Toc177554233)

[3.2.1.2 Non-Object Relational Mapping 22](#_Toc177554234)

[3.2.2 Application Service Tier 22](#_Toc177554235)

[3.2.2.1 Extending Application Service 23](#_Toc177554236)

[3.2.3 Security Interception Tier 24](#_Toc177554237)

[3.2.4 Client Interface Tier 24](#_Toc177554238)

[3.2.4.1 XML-HTTP Client 24](#_Toc177554239)

[3.2.4.2 Web Services Client 24](#_Toc177554240)

[3.2.4.3 Java API Client (Local Client & Remote Client) 25](#_Toc177554241)

[3.2.4.3.1 Technical Challenges 26](#_Toc177554242)

[3.2.4.3.2 Dynamic Proxy Based SDK Generated Client API 27](#_Toc177554243)

[3.2.4.3.3 Connecting to Multiple Remote Application Services 29](#_Toc177554244)

[3.2.5 Security Filters 30](#_Toc177554245)

[3.3 Security 30](#_Toc177554246)

[3.3.1 Authentication 31](#_Toc177554247)

[3.3.2 Authorization 31](#_Toc177554248)

[3.3.3 Instance and Attribute Level Security 32](#_Toc177554249)

[4 System Usage 33](#_Toc177554250)

[4.1 XML-HTTP Interface 33](#_Toc177554251)

[4.1.1 Accessing Data via the Browser 33](#_Toc177554252)

[4.1.2 Accessing Data via the REST Interface 36](#_Toc177554253)

[4.2 Java API Interface 38](#_Toc177554254)

[4.2.1 Obtaining ApplicationService 38](#_Toc177554255)

[4.2.2 ApplicationService API Methods 40](#_Toc177554256)

[4.2.2.1 Convenience Query 41](#_Toc177554257)

[4.2.2.2 HQL Query 41](#_Toc177554258)

[4.2.2.3 Detached Criteria Query 42](#_Toc177554259)

[4.2.2.4 CQL Query 44](#_Toc177554260)

[4.2.2.5 Nested Search Criteria Query 49](#_Toc177554261)

[4.3 Web Service Interface 52](#_Toc177554262)

[4.3.1 SDK WSDL Directives - Schema Imports 53](#_Toc177554263)

[4.3.2 WSDL Service Definition 54](#_Toc177554264)

[4.3.3 WSDL Port Types (Network Endpoints) 55](#_Toc177554265)

[4.3.4 Messages, Elements, and Types 56](#_Toc177554266)

[4.3.5 Web Service Error Handling 58](#_Toc177554267)

[4.3.6 SOAP Fault Structure 58](#_Toc177554268)

[5 System Usage – Secured System 59](#_Toc177554269)

[5.1 XML-HTTP Interface 59](#_Toc177554270)

[5.2 Web Services Client 61](#_Toc177554271)

[5.3 Java API 61](#_Toc177554272)

[6 Performance Tuning – Java API 63](#_Toc177554273)

[7 Utilities 65](#_Toc177554274)

[7.1 XML Utility (Marshalling and Unmarshalling) 65](#_Toc177554275)

[7.1.1 Marshaling Java Objects to XML 67](#_Toc177554276)

[7.1.2 Unmarshaling XML to Java Objects 68](#_Toc177554277)

[8 Configuring and Running SDK 70](#_Toc177554278)

[8.1 SDK Configuration Properties 70](#_Toc177554279)

[8.2 Generating the SDK System 74](#_Toc177554280)

[8.2.1 Ant Build Script Targets 74](#_Toc177554281)

[8.2.2 Selectively Generating Components 76](#_Toc177554282)

[8.3 Overview of Generated Packages 76](#_Toc177554283)

[8.4 Deploying the Generated System 77](#_Toc177554284)

[8.4.1 Deploying to JBoss Server 77](#_Toc177554285)

[8.4.2 Deploying to Apache Tomcat Server 78](#_Toc177554286)

[8.5 Testing the caCORE SDK Generated System 78](#_Toc177554287)

[8.5.1 Testing the Web Interface 78](#_Toc177554288)

[8.5.2 Testing the Java API 79](#_Toc177554289)

[8.5.3 Testing the XML Utility 80](#_Toc177554290)

[8.5.4 Testing the Web Service Interface 82](#_Toc177554291)

[8.5.4.1 Testing the Web Service URL 82](#_Toc177554292)

[8.5.4.2 Obtaining the WSDL for Deployed Services: ?WSDL 83](#_Toc177554293)

[8.5.4.3 Testing Web Services via the Client Program 84](#_Toc177554294)

[Configuring Security 85](#_Toc177554295)

[CSM Integration 85](#_Toc177554296)

[ACEGI Security Integration 85](#_Toc177554297)

[Transport Layer Security 85](#_Toc177554298)

[9 FAQs 86](#_Toc177554299)

[10 Roadmap 87](#_Toc177554300)

[Appendix A Example Model and Mapping 88](#_Toc177554301)

[Appendix B Glossary 89](#_Toc177554302)

# Introduction

NCICB provides biomedical informatics support and integration capabilities to the cancer research community. NCICB has created caCORE Software Development Kit or caCORE SDK, a data management framework designed for researchers who need to be able to navigate through a large number of data sources. caCORE SDK is NCICB's platform for data management and semantic integration, built using formal techniques from the software engineering and computer science communities. By providing a common data management framework, caCORE SDK helps streamline the informatics development throughout academic, government and private research labs and clinics. The SDK generated system is built on the principles of Model Driven Architecture (MDA) and n-tier architecture & consistent API. The use of MDA[[1]](#footnote-2) and *n*-tier architecture, both standard software engineering practices, allows for easy access to data, particularly by other applications.

Figure 1: SDK System Generation Process

***caCORE SDK Sub-Systems***

The caCORE SDK primarily comprises of two major sub-systems. First sub-system is the code generation module which accepts UML model as input and produces various artifacts corresponding to the model. Second sub-system is the runtime system which is a pre-built system. This runtime system utilizes the artifacts generated by the code generation module in order to serve the data to the client application. The runtime system can deliver the data to the client in variety of ways and details of that can be found in section <<?>>. Section<<?>> describes the architecture of the code generation sub-system. This section also provides an overview of the artifacts that the caCORE SDK generates. Section<<?>> provides an overview of the architecture of runtime system.

***caCORE SDK Users***

Corresponding to the two sub-systems, the users of the caCORE SDK will also fall into one of the two categories viz 1) user of the code generation sub-system and 2) users of the generated system. Users of the code generation sub-system will primarily be focused on preparing the UML model and running it through the caCORE SDK with appropriate settings to generate the runtime system. Users of the runtime system will primarily be focused on writing queries against the runtime system to retrieve the data from the data source. Section<<?>> provides an overview of how the code generation sub-system can be used by the users. Section<<?>> provides an overview of runtime system usage scenarios.

## SDK within caCORE environment

caCORE SDK can be utilized to quickly generate the silver level compatible systems from the UML model. However, the use of the SDK is not limited to generating the silver level compatible systems. caCORE SDK can be utilized outside the caCORE environment also to create a system that is generated from the UML model and runs on the standardized query languages. Within the caCORE application development process, the caCORE SDK serves the purpose of generating the system from the model. More details on the caCORE application development process can be found at <<?>>

## How can I get benefited by using caCORE SDK<<?>>

Users of the caCORE SDK can get benefited in numerous ways. The primary benefit of using caCORE SDK includes

* **Consistent UML representation of the data** – Users of the caCORE SDK are forced to represent the data in the UML format. As a user of the SDK, the user is likely to maintain their UML model through the life cycle of the application. The same UML model can be used to quickly learn about the organization of the data at various levels in the application.
* **Rapid data service generation** – SDK can generate caBIG’s Silver Level compatible APIs quickly from the UML model. Once the UML model and the database is ready, the user can generate the data service in matter of hours. Manually building the application from the ground up can take several months to achieve the same functionality.
* **Uniform way to access data** – SDK generated systems provide uniform access to the data stores. Other applications developed using the caCORE SDK have similar mechanism to retrieve the data hence the user gets benefited from the knowledge overflow to and from other applications.
* **Query using information model** – SDK generated system allows the users to write queries in various ways including Query By Example. Since the query is independent of the implementation of system, the changes in the runtime systems will not affect the client application
* **Integration with caGrid** – SDK generated systems can be easily integrated with the caGrid using the caGrid’s Introduce toolkit. Developing the caGrid compatible data service without using caCORE SDK can result in error prone and lengthy process

## How to Obtain caCORE SDK

caCORE SDK is released periodically in .zip file format and .tar file format. The updates are released frequently on the NCICB’s GForge website. Latest releases and archives can be accessed at <https://gforge.nci.nih.gov/frs/?group_id=148>

## System requirement

In addition to the caCORE SDK bundle mentioned in section above, one will also need additional hardware and software mentioned below.

### caCORE SDK Minimal System Requirements

The caCORE SDK 3.2.1 has been built and tested on the platforms shown in table below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | *Linux Server* | *Solaris* | *Windows* |
| Model | HP Proliant ML 330 | Sunfire 480R | Dell GX 270 |
| CPU | 1 x Intel® Xeon™ Processor 2.80GHz | 2 x 1050MHz | 1 x Intel® Pentium™ Processor 2.80GHz |
| Memory | 4 GB | 4 GB | 1 GB |
| Local Disk | System 2 x 36GB | (RAID 1)  Data = 2 x 146 (RAID 1)  System 2 x 72GB | System 1 x 36GB |
| Operating System | Red Hat Linux ES 3 (RPM 2.4.21-20.0.1) | Solaris 8 | Windows XP/2000 Professional |

**Note:** The user of the caCORE SDK will need computer system for two purposes; first, to generate a system using caCORE SDK and second, to host the generated system in production environment.

**Note:** User of the caCORE SDK should choose the appropriate hardware configuration based on the amount of data that the system is expected to handle. The configurations mentioned above are for illustration purpose only.

### caCORE SDK Software and Technology Requirements

**Software Requirements**

You must download and install the required software that is listed in table below. This software is not included with the caCORE SDK. The software name, version, description, and URL hyperlinks (for download) are indicated in the table.

|  |  |  |  |
| --- | --- | --- | --- |
| *Description* | *Software* | *Version* | *URL* |
| The J2SE Software Development Kit (SDK) supports creating J2SE applications | JDK | 1.5.0\_11 or higher | <http://java.sun.com/j2se/1.5.0/download.html> |
| UML Modeling Tool† | Enterprise Architect | 6.0 or higher | <http://www.sparxsystems.com.au/> |
| ArgoUML | 0.24 or higher | <http://argouml.tigris.org/> |
| Database Server† | Oracle | 9i | <http://www.oracle.com/technology/products/oracle9i/index.html> |
| MySQL | 5.0.27 | <http://dev.mysql.com/downloads/mysql/5.0.html> |
| Application Server† | JBoss | 4.0.5 | <http://labs.jboss.com/jbossas/downloads> |
| Tomcat | 5.5.20 | <http://tomcat.apache.org/download-55.cgi> |
| Build Tool | Ant | 1.6.5 or higher | <http://ant.apache.org/bindownload.cgi> |

† Only one of the software is required

## Contributing Towards Development of caCORE SDK?

caCORE SDK project is managed by NCICB’s project manager with technical help from Ekagra Software technologies. IF any user would like to make contribution by providing a patch for the defect then they can do so by sending an email to the caCORE SDK Users’ mailing list ([CACORE\_SDK\_USERS-L@list.nih.gov](mailto:CACORE_SDK_USERS-L@list.nih.gov)). Users interested in participating in the development process can contact NCICB management for more details on getting approval on

# Code Generation - Technical Overview

This chapter describes the code generation process as perceived by the developers community and in addition to that how caCORE SDK’s code generation sub-system works.

## Code Generation

### What is Code Generation?

Code generation is a systematic process of converting a model into a series of instructions or programs that can be executed by a machine. Code Generation principle is primarily popular in the world of programming language compilers (e.g. C compiler or Java compiler) in which code generation stage is responsible for generating machine specific instructions or assembly language instructions. The input to the code generation stage typically consists of parsed source code or abstract syntax tree which is prepared by the source code parser. Here the caCORE SDK’s code generator generates the artifacts which are consumed by the SDK’s runtime system.

### Code Generation’s Role in caCORE SDK

Figure 2: Code Generation

As other tools and programming language compilers use the code that the SDK generates, the SDK itself can be viewed as a level above the other compilers and tools. The code generation module is responsible for generating various artifacts from the UML model. Like output from the code generation stage of compilers, the output from the code generation stage of SDK is specific; the output of the caCORE SDK consist of artifacts like Java source code, O/R mapping files etc. In other words, the caCORE SDK transforms the UML model into system specific artifacts and the code generation engine is nothing but a complex transformer for the UML model.

The primary purpose of the caCORE SDK is to allow the users to quickly build the data services. One of the ways in which this requirement can be implemented is to generate the application for the user based on certain settings. caCORE SDK achieves this purpose by taking input as UML model which consist of object model and data model for the user. With application generation settings and UML model, the caCORE SDK generates a complete application with many different features as mentioned in section<<?>>.

### Features and Limitations of Code Generation

UML provides a generic mechanism to represent various parts of the software systems and its lifecycle. However, UML by itself is unable to describe how the complete system works after implementation. To efficiently generate code from the UML model, the SDK specifies additional information (in form of tag values) that needs to be embedded inside the model. This additional information allows the SDK to determine how the code generation should proceed.

The code generation module of SDK can generate only a set of well known features from the model. It cannot read and interpret new features that a user wants to generate. If user wants the SDK to interpret the new UML features then the SDK code generator needs to be modified. In addition to the code generator’s modification, runtime system also requires modification so that it can consume the modified artifacts from code generator.

## Code Generation Process in SDK

SDK code generation process can be viewed as a layer of different processes. In order to generate code from the UML model, one first needs to export the constructed model from a UML modeling tool. Subsequently, this exported model can be used by the SDK to generate the code.

### SDK Code Generation Process

The code generation process involves the following high-level steps to generate the artifacts required.

1. Read UML model
2. Artifact Generation
3. Output management

caCORE SDK Code Generator

UML Modeling Tool

Figure 3: Details of Code Generation Process in SDK

### Read UML Model

When the UML model is constructed in the UML modeling tool, the information about the model is stored in a proprietary formatted file. In order for SDK to read the model, the model information needs to be translated into a standard format that can be interpreted by other modeling tools. Once the model information is exported in the standard format, the SDK can read the model information and convert it into internal data structures. These internal data structures can then be used by other stages of the SDK to generate the desired code. Having the internal data structures also gives additional flexibility to the SDK. If the modeling tools adopt a new standard or starts exporting information in some format other than the one recognized by SDK, a new model reader can be developed without affecting other stages of code generation.

Currently SDK uses the UML Model reader developed as a separate project at NCICB. The UML Model Reader (aka XMI Handler) can interpret model information created by the Enterprise Architect or ArgoUML modeling tool. If a new UML modeling tool is to be supported by SDK’s code generator then in that case the UML Model Reader needs to be modified; however, the code generator does not need to be changed.

### Artifact Generation (Model Transformation)

The SDK generates various artifacts based on the information that it obtains from the UML model. The artifact can be a Java Bean, O/R Mapping file, or web service deployment descriptor. Most artifacts are generated from the information obtained from the UML model. Other artifacts are generated from the property files and configuration files supplied at the time of running the SDK. This section only addresses artifact generation from the UML model only.

The UML model contains various complex elements. The artifact generation stage reads all the elements of the UML model then construct a collection of relevant elements from which a particular artifact can be generated. The artifact generation process needs to be repeated for each type of artifact.

### Output Management

When the artifact is generated, the output needs to be written into the file. The file content can be Java source code or XML. If the artifact is a Java program then it needs to be written in a particular folder hierarchy to preserve namespace. Also, all the Java program files require “.java” for their file extension. Similarly, the generated XML documents need to be placed in appropriate folders and assigned appropriate file names and extensions.

### Code Generation Framework

As explained in the earlier section, the code generation process involves various steps in order to generate an artifact. If there are many different types of artifacts to be generated, the model transformation process needs to be executed for each type of artifact to be generated. In that case, it becomes necessary to automate these steps so that the artifact generation process can be handled efficiently and effectively.

In the given design, the artifact generation process is controlled by means of a control or configuration file. The control file specifies what combination of components will be used to generate a particular type of artifact. The execution engine (Generator), which understands the information specified in the control file, can then read the control file and orchestrate the workflow as desired.



Figure 4: Code Generation Workflow Automation Sequence Diagram

When the code generation execution engine initializes, it reads the control file and configures itself with the information obtained from the control file. The execution engine configuration involves initializing the components defined in the controller file as sub-elements and configuring them one at a time. Once the configuration of the execution engine and components finish, the code generation execution engine executes the workflow as described by the pseudo-code below.

1. Open UML Model file
2. Read UML Model file containing various UML models
3. Set the UML Model in the Generator
4. Set the Validators in the Generator
5. Set the Transformers in the Generator
6. Execute the Generator
   1. Execute all the registered Validators
   2. If the errors are present during the previous validation then stop executing and log the errors
   3. Execute the validate method of all the registered Transformers
   4. If the errors are present during the previous validation then stop executing and log the errors
   5. Execute all the registered Transformers
      1. Generate Artifact from the UML Model
      2. If errors are discovered during code generation then return the errors
      3. Pass the generated Artifact to the registered ArtifactHandler
         1. Write the artifact to the respective file

### Reusable Components of the Code Generation Workflow

In order to complete the code generation process, the components used in the code generation workflow need to implement specific behaviors. As the code generation engine executes the workflow, it instantiates and configures components required for the workflow (as specified in the control file). Each type of component will need some information to configure itself. This information is supplied from the control file to the component. The class which implements the interface (Validator or Transformer) recognized by the code generation execution engine is responsible for following certain behaviors expected by the engine. This mechanism allows the new implementation of the components to be plugged into the workflow just by modifying the control file.

Since SDK code generator uses a Spring Framework’s bean configuration file, configuring each component becomes easy. It is up to the developer of a component to specify what information the component would need to execute itself.

## Overview of SDK Generated Artifacts<<?>>

As part of the code generation process, the caCORE SDK generated following artifacts with help of different transformers.

* **Beans** – For each object defined in the object model, a Java bean (POJO) is generated. The generated bean follows the same package structure as the folder structure in the object model. The generated Java beans are compiled and packaged in a JAR file. The JAR file is named as *project\_name-beans.jar*
* **Hibernate files -** Following hibernate files are packaged in the separate JAR file after the generation. The JAR file is named as *project\_name-orm.jar*
  + **Hibernate mapping files** - For each object defined in the object model, the caCORE SDK generated a Hibernate mapping file (Object Relational mapping file). In case of inheritance in the object model, the mapping file is created for the root level class in the inheritance hierarchy. The generated files follow the same package structure as the folder structure in the object model.
  + **Hibernate configuration file** – A configuration file named hibernate.cfg.xml is generated for hibernate which contains a list of all the generated hibernate mapping files in addition to the database connection settings.
  + **EHCache configuration file** – A cache configuration file for hibernate
* **XSD and XML Mapping files** – For each package defined in the object model, the caCORE SDK generates a XSD file. The XSD file is named after the fully qualified name of the package name for which the file was generated. caCORE SDK also generates XML mapping files (castor mapping file) for the entire object model. There are two mapping files that are being generated viz xml-mapping.xml and xml-unmapping.xml. These files are primarily used by the caGrid project to create a grid service from the SDK generated system
* **Web Service deployment descriptor file** – A deployment configuration file for the AXIS based web service is generated for the entire object model

# Runtime System - Technical Overview

This section describes the architecture of the caCORE system. It includes information about the major components, such as security, logging, database object-relational mappings (ORM), client-server communication, and how the system connects to non- ORM systems.

## High-level Architecture

The caCORE SDK generated runtime system’s infrastructure exhibits an n-tiered architecture with client interfaces, server components, backend objects, data sources, and additional backend systems. 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 different levels 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).

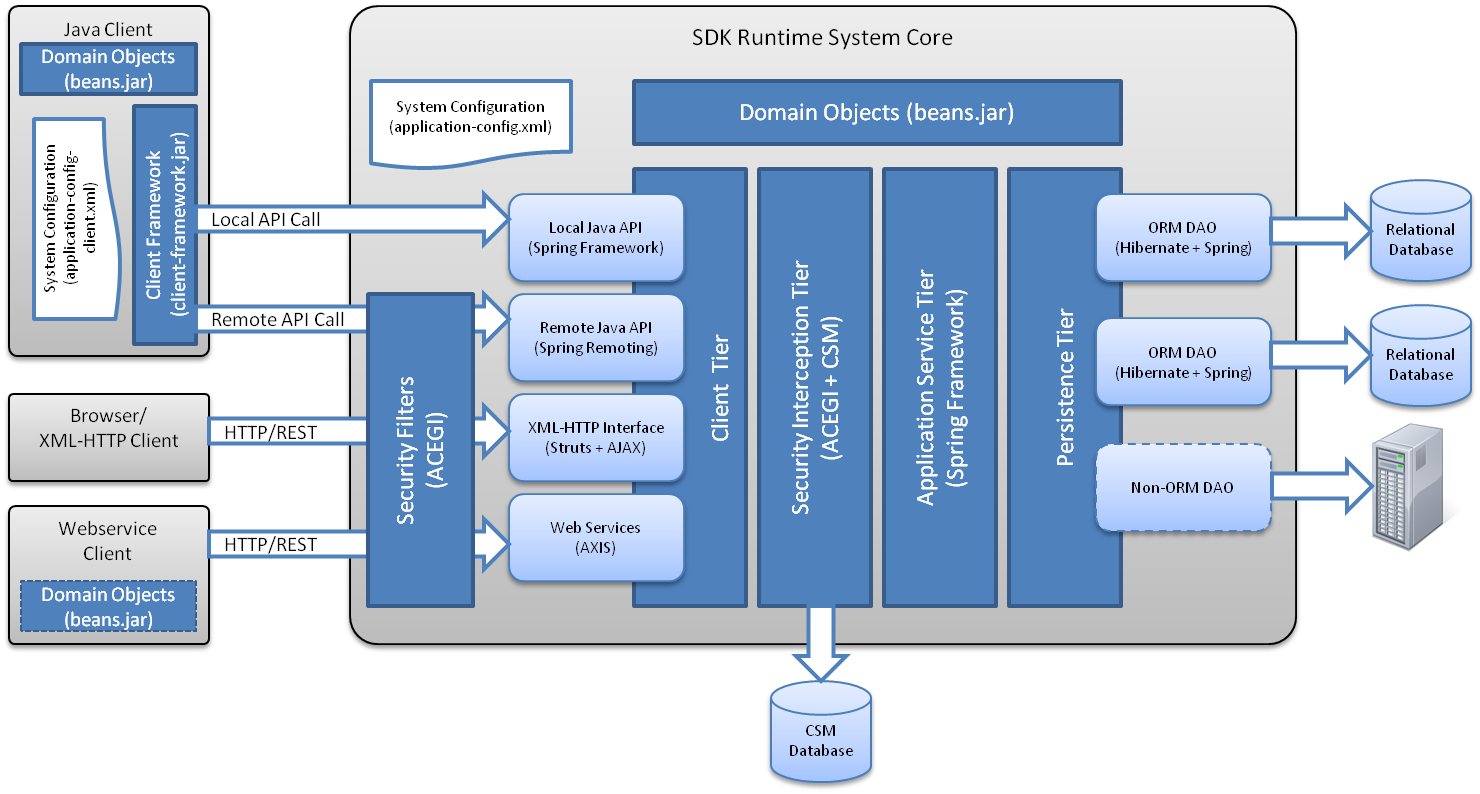


Figure 5: SDK Generated Runtime System Architecture

## 

## n-tier system

The SDK generated system is just like a typical n-tier architecture system. Each tier in the n-tier system is responsible for a set of defined activities. In SDK generated system, the layers starting from the lowest layer are as following

### Persistence Tier



Figure 6: Persistence Tier Classes

The persistence tier is responsible for understanding the query that user has sent and fetching the results corresponding to the query. SDK currently supports persistence tiers created in two ways; object-relational mapping (ORM) based persistence tier and non-object-relational mapping (Non ORM) based persistence tier. For the user of the object-relational mapping based mechanism to access the data stored in the persistence tier, SDK provides a pre-constructed DAO (ORMDAOImpl). The ORMDAOImpl is written specifically for hibernate based object relational mapping. This DAO converts the user query in the hibernate specific query and executes it using hibernate APIs. Each DAO also provides a list of the domain objects when the Application Service tier asks for it by using the getAllClassNames() method of the DAO. If the Application Service tier determines that there is an overlap between the lists of domain objects provided by the DAOs then the application will not be loaded. More details on how each DAO works is in the subsections and .

### Object Relational Mapping

SDK code generator performs the object relational mapping for Hibernate[[2]](#footnote-3) as underlying technology. Hibernate allows the objects to be mapped to the relational database by means of object relational mapping (ORM) files. These ORM files are generated by the caCORE SDK during the code generation process. If the user intends to not to use one or all of the mapping files and provide the mapping files developed by himself then he can do so by altering the code generation process. Alteration of the code generation process can be done through the configuration files and is explained in the later section.

### Non-Object Relational Mapping

SDK users can choose to not to use the ORM as a way to map the relational database to the objects or the data for the objects reside on the remote server. In this scenario, the onus of populating the objects based on the query is on the user of the SDK. The user will have to develop a custom Non ORM DAO which can perform the task of retrieving the data. The custom Non ORM DAO is required to implement the interface expected by the caCORE SDK. Other than supporting the method to retrieve results using query, the Non-ORM DAO needs to implements another required method (getAllClassNames()) which can return a list of the domain objects supported by that DAO.

### Application Service Tier



Figure 7: Application Service

Application service tier consolidates incoming requests from the various interfaces and forwards them to the appropriate persistence tier implementation. This is the main tier which facilitates the operations within the SDK generated system. The methods in the ApplicationService interface are the one that are exposed to the Java Clients. ApplicationServiceImpl class has the concrete implementation of the ApplicationService interface. When any of the client interfaces in the SDK code requests a handle to the ApplicationService, the default implementation of ApplicationServiceImpl is returned. When the remote Java client requests a handle to the ApplicationService, a remote handle to the ApplicationService is wrapped inside the ApplicationServiceProxy and returned to the client. The default Application Service tier has methods to support ORM based system. The methods are sufficient to support requirements for most applications. If the user intends to add additional methods in the Application Service tier then details can be found in the following subsections.

### Extending Application Service



Figure 8: Extending Application Service

The default Application Service tier has methods to support ORM based system If the user intends to add additional methods in the Application Service tier then one of the possible approaches is to modify the source code to add additional methods; another option is to extend the Application Service and modify the configuration files to work with the extended Application Service. As shown in the , a CustomApplicationService can extend the ApplicationService interface and CustomApplicationServiceImpl class provides a concrete implementation of the method inside the CustomApplicationService interface. As the new methods are added to the Application Service, client tiers will also be needed to be modified to expose the additional methods to their respective clients. The configuration files on the client and the server side will need to be modified to reflect the extension of the ApplicationService.

### Security Interception Tier

This tier ensures that only the authorized users are allowed to access the system. The security configuration in the SDK is done using the ACEGI and Common Security Module (CSM) developed by NCICB. In the case of the unsecured system, this layer is disabled through the configuration files. More details on the security implementation in SDK can be found in section<<?>>

### Client Interface Tier

SDK provides four distinct ways to reach the Application Service Tier. Each of these four methods to retrieving data involves preparing the query in the format that the interface understands, sending the request to the corresponding interface, and retrieving the results from the interface to which the query was submitted earlier. Details about the system usage can be found in the next sections.

1. XML-HTTP Interface (browsers, thin clients)
2. Web Services Client
3. Local Java API Client
4. Remote Java API Client

### XML-HTTP Client

The user of the XML-HTTP interface can access the data using two types of the clients 1) Browser to see data in the form of a web page and 2) Thin client to get XML data. The clients can form a query using Query By Example (QBE) syntax and they are provided data for the result object. If client intends to fetch data for the associated object then the client is required to make a second query. In a web browser, the client can click on the link to fetch the associated object while using the thin client the client application is required to form the query to fetch the associated object and send it to the server. If the query executed by the client returns a large number of records then in that case, the server will return only the results allowed per page size. The client is required to make a second call to fetch the next page from the server. (The page size can be configured in the configuration file application-config.xml). More details on how to form the query and send it to the server using a web browser or using a thin client can be found in section<<?>>

### Web Services Client

The clients of the web service use the RPC/SOAP?? protocol to fetch the data. In case of the clients query returning a large amount of data, the client of the web service will only receive maximum number of allowed records per call. The client application is required to make an additional call to the server to fetch next chunk of data. The server also does not return the association to the client application. If the client need to fetch the association then the client application has to make an additional call with specific details on which association the client application would like to fetch. More Details on how to make a call to the caCORE SDK generated web service can be found in section<<?>>

### Java API Client (Local Client & Remote Client)

The user of the Java API interface can access the SDK generated application using two different mechanisms. 1) Remote client server API call and 2) Local API call. Regardless of what type of communication strategy the Java client application chooses, the interaction of the client application remains the same. Typical client communication with the SDK generated application is illustrated below.



Figure 9: Expected Behavior of SDK Generated Application

The client of the generated application intends to fetch data from the database and use the data in the desired manner in their respective application. The client application intends to achieve this behavior by following the steps below.

1. Obtain a reference to the service which can deliver the data
2. Form the query and search the database using the prepared query on the service obtained in step 1
3. Iterate through the result and obtain attributes/associations of the result object

If the client application is using the generated system in remote client server mode then in that case the generated client will need to connect to the remote service using the remote client. On the other hand, if the client application intends to use the generated system locally then it needs the service present in the local environment and remote calls should be avoided. Since the client application is developed in a different environment, it is better to isolate the client from knowledge about what type of client is used to fetch the data i.e. local client vs. remote client.

### Technical Challenges

There are many ways to implement the expected behavior. The choices from the technologies front includes 1) Java RMI 2) Web Services 3) EJB 4) CORBA 5) Remoting etc. From the client perspective it is least relevant which technology is adopted to solve the problem of the client server communication. Another problem is how to fetch large result set and associated objects. Regardless of which technology is used to implement the application framework, the problem of loading the large result set and associated object remains. In order to resolve these problems, the data is required to be loaded on demand (lazy-loading). In order to lazy-load the objects, the developed application framework has to recognize the event when the remaining objects are to be loaded from the database. The events which require lazy loading are 1) Iterating through the large result set and, 2) Accessing attributes/associations of the retrieved objects.

The retrieved objects are required to trigger the event whenever the client application makes an attempt to access the attribute/association of that object. One possible way to achieve this functionality is to hardwire the event triggers in the result objects. This approach makes the result objects tied to the SDK generated application. Another way to achieve the same functionality is to dynamically inject the event triggers in the result objects. Following subsection describes how lazy-loading behavior is achieved in SDK.

### Dynamic Proxy Based SDK Generated Client API



Figure 10: Actual Behavior of the SDK Generated Application - 1

In order for the client application to obtain the handle to the service tier (Application Service) of the generated application, SDK provides a helper class called ApplicationServiceProvider (ASP). ASP instantiates the service based on the settings in the configuration file (application-config.xml). The sequence diagram () shown above demonstrates how ASP retrieves the service. When the client application requests a handle to the service, ASP retrieves the handle to the service using the configuration file and adds an interceptor to the service resulting in ApplicationServiceProxy. ApplicationServiceProxy is a dynamic proxy generated using the AOP feature of the Spring Framework[[3]](#footnote-4). ApplicationServiceProxy intercepts all the calls to the actual ApplicationService and takes action to facilitate the lazy-loading mechanism described earlier. At this time, the client application is expecting a handle to the ApplicationService to be received from ASP but they will receive ApplicationServiceProxy.

When client application makes a call to the ApplicationService, at that time, ApplicationServiceProxy will intercept the call and execute it. After the invocation made by the client on the ApplicationService, the ApplicationServiceProxy obtains the result from the actual ApplicationService. The obtained result set can be primitive objects of Java, domain objects or Java collections. Since domain objects or collection of domain objects can require to lazily load its associated objects, ApplicationServiceProxy is required to add an interceptor on the domain objects. After obtaining the results from the ApplicationService, the ApplicationServiceProxy uses ProxyHelper class to add appropriate interceptor (BeanProxy) to the domain objects so that the domain objects can trigger the event to lazily load attribute/associated objects.



Figure 11: Actual Behavior of the SDK Generated Application - 2

The results returned from the ApplicationServiceProxy to the client application the result objects will have an added interceptor (BeanProxy). The interceptor holds a reference to the ApplicationService where the result objects were loaded from. When the client application invokes on any of the methods on the result objects to retrieve attribute/association then the interceptor (BeanProxy) of the domain object triggers an event. In wake of this event, the ProxyHelper class is being used as a decision maker to determine if the attribute/association should be lazily loaded. If the ProxyHelper class indicates that the method should be lazily loaded (i.e. the method should not be executed locally and ApplicationService should be used to obtain the return value) then in that case BeanProxy again uses ProxyHelper to execute the method and load the result from the correct Application Service.

For the ORM based applications, the ProxyHelper class will always check for the presence of HibernateProxy for the associations. If the HibernateProxy is present instead of actual associated object then in that case the ApplicationService is being called (via ApplicationServiceProxy) to fetch the associated object. The ProxyHelper is responsible for preparing the query and call the ApplicationService with appropriate parameters. For a Non-ORM system, the ApplicationService may have been extended to support additional query methods and these methods can return the domain objects that are not same as regular POJOs. In that case, the implementer of the Non-ORM system will need to intercept all the method calls to the result objects and resolve the lazy initialization routine. The Non-ORM system can configure the custom ProxyHelper through the configuration file (application-config-client.xml)

### Connecting to Multiple Remote Application Services

The client application framework can be used to connect to multiple application services at the same time. i.e. the client can connect to various SDK generated services at the same time using the same framework. Note that this feature can be used only with the remote client and not with the local client API. In order to facilitate this feature, the ApplicationServiceProvider (ASP) class is used in conjunction with the proxy framework mentioned earlier. ASP reads information from the application-config.xml to create a new instance of the Application Service. If the client application does not mention the service it needs to connect to, then in that case the ASP initializes the service described under the “ServiceInfo” bean in the configuration file. However, if the client application mentions the name of the service then ASP locates the configuration entry for that service, instantiates the service handler and returns it to the client after adding the interceptor. When using the client framework in the multiple services mode, the developer of the client application has to make sure of the following

1. Domain objects corresponding to all the services it is trying to connect must be present in the local environment.
2. The services to which the client application intends to connect should be based on the ApplicationService interface of the SDK core.
3. The remote services can be an extension of the ApplicationService interface provided by the SDK. If one or more services have the same extension interface name (e.g. com.xyz.CustomService) then they should have the same method signatures as well.
4. All the extensions of the ApplicationService interface corresponding to different remote services should be present in the local environment
5. If any of the remote service has modified the ApplicationService interface then the client framework will fail to operate.
6. Appropriate entries should be made in the application-config.xml for each of the remote services

### Security Filters

The security filters are the HTTP servlet filters configured through ACEGI[[4]](#footnote-5) in application-config-web-security.xml file. The filters are used in a chained fashion to ensure reusability of the filters. For different client interfaces, the purpose of the filter is to 1) retrieve user’s security credentials from the HTTP message and 2) log user in the application by putting information in the ThreadLocal[[5]](#footnote-6) variable 3) clear user’s security information from ThreadLocal at the end of the request. In case of a web interface, the user’s security information is stored in the HTTPSession so that it can be retrieved on the subsequent call. For all other interfaces, the user is required to resubmit login information for each request to be processed.

## Security

SDK provides an integrated security by using ACEGI and CSM as underlying technologies. ACEGI is being used at the security interceptor layer. After the calls from the user are intercepted with the help of ACEGI, CSM is used to provide security at the core level. By default, the security is disabled in the SDK configuration. When the user intends to enable the security, he/she can do so by enabling security flag in the configuration file and generating the system. User can also enable the security in already generated system by modifying multiple configuration files; since this process is error prone, it is not recommended to all the users of the SDK.

Figure 12: Security Levels in caCORE SDK

SDK can provide security at various levels viz Class level security, Instance level security, Attribute level security or no security.

* Unsecured system/No Security – All the users of the SDK has equal access to the data that the runtime system serves.
* Class level security – Only users who have access to certain objects in the system can query for the data. For e.g. Doctors can view the data for the patients whereas administrator can not
* Instance level security – Users are allowed to access data for only the records for which they have access to. For e.g. Doctors can view data for only their patients and no other patients in the system
* Attribute level security – Users are allowed to see data for which they have authorization. For e.g. Doctors are allowed to see patient’s medical record number but they cannot see patient’s social security number.

The user of the SDK can choose the appropriate level of security for his/her system. Configuration of which level of security is to be enabled can be managed through the configuration file at the system generation time. When the security is enabled, the system achieves the class level security by default. If the user intends to provide instance and/or attribute level security then he can choose to do so. More details on how to configure the security in CSM for SDK generated system can be found in section<<?>>

Figure 13: Security Layers in caCORE SDK

### Authentication

SDK generated application provides a mechanism to log user in the application. It takes the user credentials from the client and supplies it to ACEGI framework so that ACEGI framework can validate the user credentials and decide if user can proceed with the operation or not. Since security policy is managed at the CSM level, a bridge is prepared between CSM and ACEGI. The ACEGI-CSM bridge retrieves the user information from the security database and logs the user in the application. If the user is successfully logged in the system, his/her security policy is cached at the application level. In case of unsuccessful login, an exception is thrown back to the user indicating the cause of the error.

### Authorization

Authorization in SDK refers to the class level security. Whenever user is trying to execute any operation on the service layer, ACEGI framework intercepts the call to the operation and with the help of a SecurityHelper class it determines what classes the user is trying to access. Subsequent to that, ACEGI framework decides if the user has access to that class or not by checking against the security policy of the user. If the user does not have access then access denied exception is thrown back to the user.

### Instance and Attribute Level Security

If the system is a secured system and instance and/or attribute level security is enabled then SDK utilizes the CSM’s services to provide the instance and attribute level security. CSM provides instance level security by altering the query using Hibernate filters. The modified query has additional criteria in the where clause which goes against the CSM security configuration which restricts user to retrieve only the records to which he/she has access to. In order to use this feature, the CSM security configuration has to be available on the same database schema.

# System Usage

This section provides an overview of examples on how the generated system’s client interfaces can be accessed by the client application or the user.

## XML-HTTP Interface

XML-HTTP interface can be accessed in two ways. User can access the web interface from the internet browser or user can write a thin client application which can fetch the data in XML format from the server using the REST like syntax.

### Accessing Data via the Browser

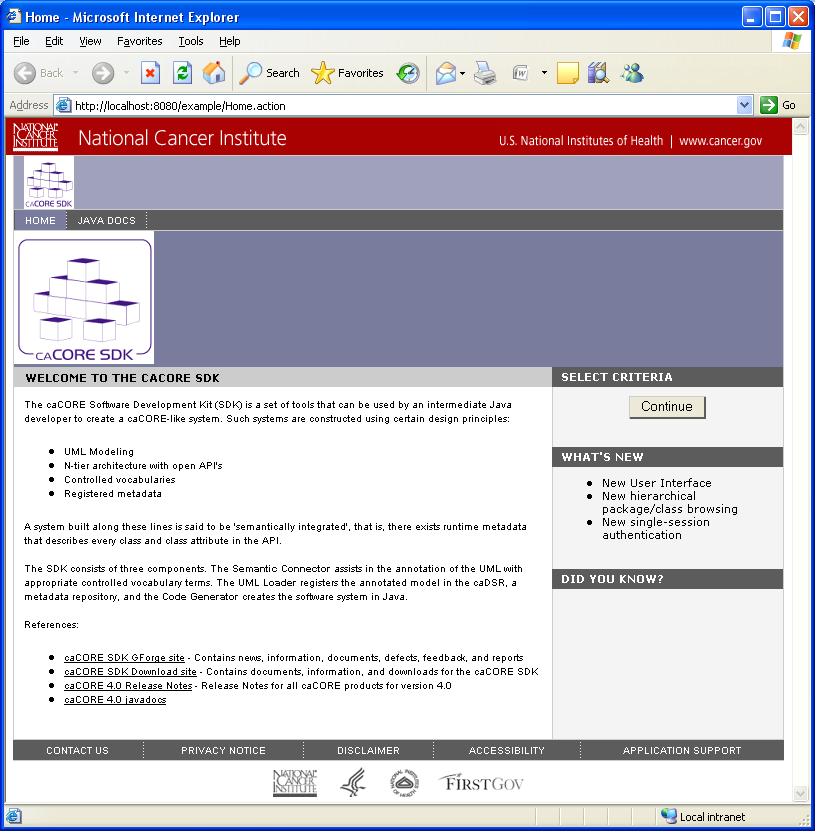


Figure 19: SDK Home Page

|  |  |
| --- | --- |
| **NOTE:** | Whenever security is disabled (which is the default), a Continue button is shown on the Home page. If security is enabled, a Login form requesting a User ID and Password will be shown instead. More about enabling security is provided in section ???? |

The SDK generated web interface consists of several web pages that facilitate access to domain data. The *Home* page can be accessed via the following URL pattern:

|  |  |
| --- | --- |
| **SDK GUI URL Pattern** | http://<server\_name>:<server\_port>/<project\_name> |
| **Sample SDK GUI URL** | http://localhost:8080/example |

The *Home* page contains various links to SDK related sites and documentation, such as:

* The SDK GForge Site;
* The SDK Download Site; and,
* The SDK Release Notes
* Javadocs for the domain objects of the generated system

Clicking on the *Continue* button will take the user to a hierarchical domain package/class browser tree known as the *Content* page, which contains both a domain class browser, as well as a *Criteria* form used to search for records. The *Content* page for the sample SDK model is shown below:

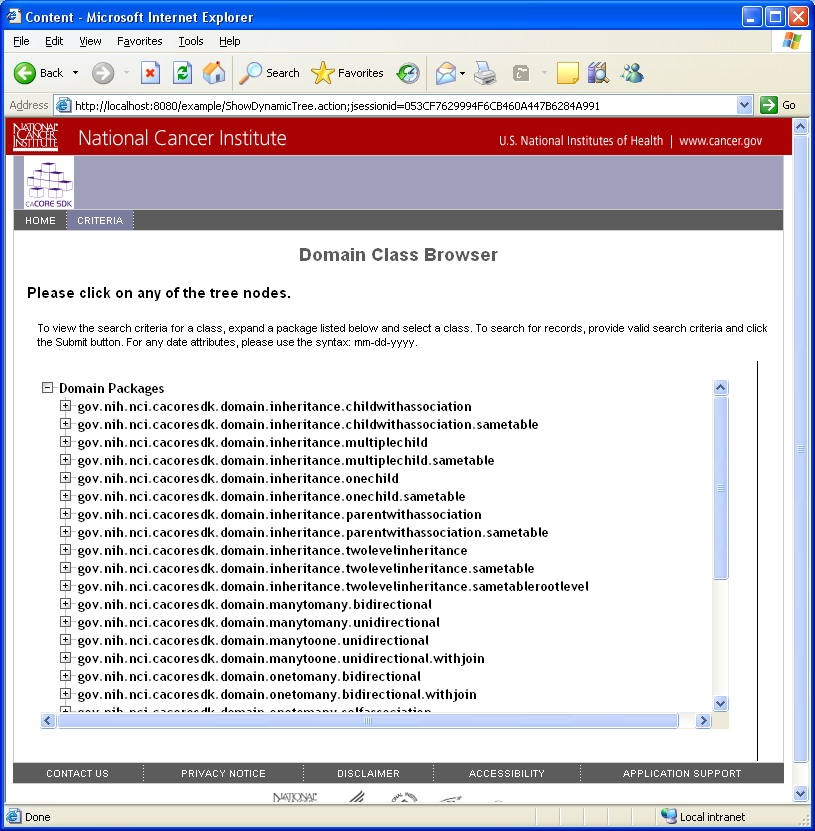


Figure 20: Hierarchical Domain Class Browser Page

The *Domain Class Browser* tree can be expanded or collapsed by clicking on the ‘*+’* or *‘-‘* symbol to the left of a domain package name. To view the *Search Criteria* for a particular class, expand a domain package so that its classes are listed, then select the desired class name node. A *Search Criteria* form listing the searchable class fields will be displayed to the right of the browser tree.

The following diagram illustrates the *Search Criteria* form for the *Credit* class of the sample SDK model:

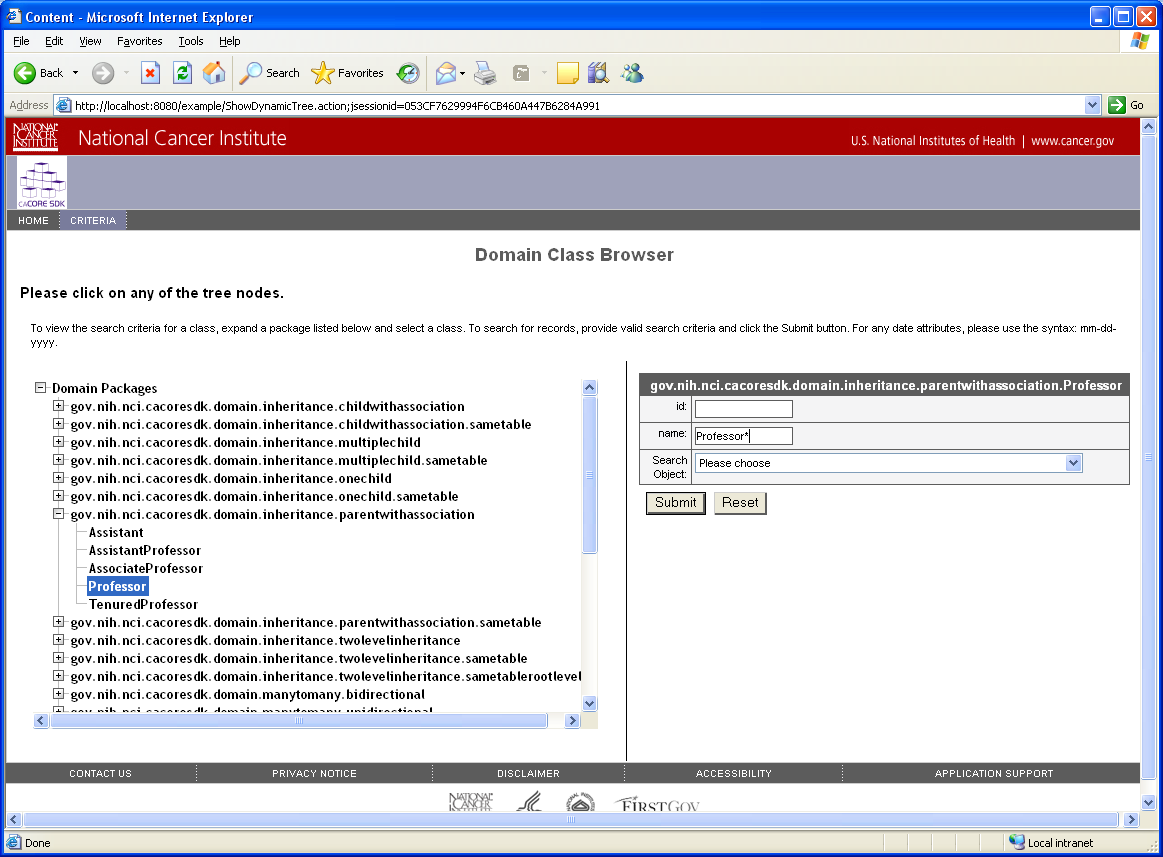


Figure 21: Search Criteria Form

|  |  |
| --- | --- |
| **NOTE:** | For any date attributes, use the syntax: mm-dd-yyyy.  The Search Criteria form accepts the asterisk (‘\*’) as a wildcard character.  The Search Criteria form also contains a drop-down list containing Search Objects (domain classes) that are associated to the current domain class. Selecting a Search Object from the drop-down list will cause the query to return records of the type represented by the Search Object, and not records of the type represented by the selected class, which is the default if no Search Object is selected. |

Once the *Submit* button on the *Search Criteria* form is clicked, the SDK system responds by displaying any matching records on the *Result Data Table* page in new window. Fields from the resulting domain class type are displayed as table columns within the table header on the result page. Collection of wrapper of primitive object type and field values and are displayed as strings within corresponding table cells. Fields that represent an association to another domain class are displayed as links which can be clicked to retrieve any associated domain object records.

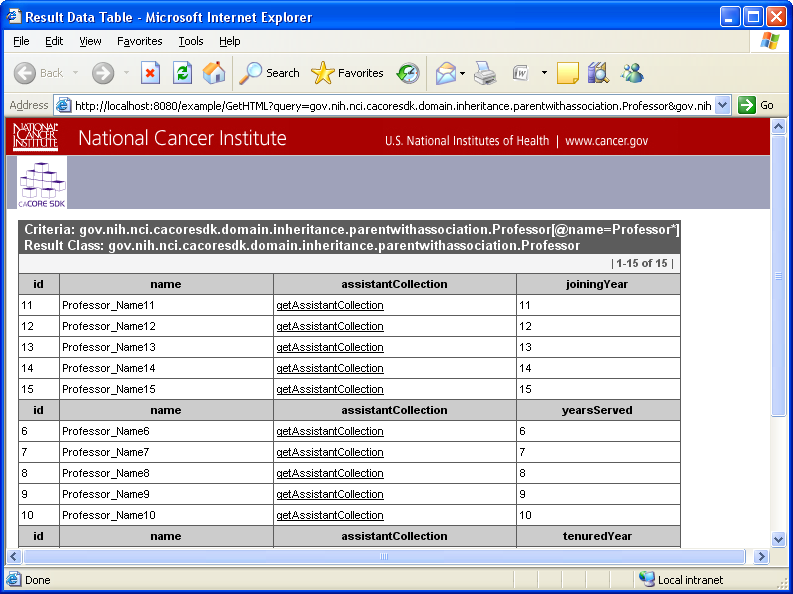


Figure 22: Result Data Table Page

### Accessing Data via the REST Interface

The Representational State Transfer (REST) interface provided by the SDK is a simple URL interface that transmits domain-specific data over HTTP without any additional messaging layer, such as SOAP, or session tracking via HTTP cookies.[[6]](#footnote-7)

The URL used by this interface adheres to the following pattern:

|  |  |
| --- | --- |
| **REST Interface URL Pattern** | http://<server\_name>:<server\_port>/<project\_name>/  GetXML?query=<target>&<criteria>[&rolename=<rolename>] |

The following table describes each of the variable properties of the REST URL:

|  |  |
| --- | --- |
| Parameter | Description |
| server\_name | A string identifying the server, or host, name. Examples include localhost and 127.0.0.1. |
| server\_port | A string identifying port number the SDK server is listening on. Examples include 80 or 8080. |
| project\_name | A string identifying the project name used when building and deploying the SDK application. Examples include example and myproject.  Note: this value coincides with the PROJECT\_NAME property found within the deploy.properties file. |
| target | A string identifying the qualified or non-qualified query target/result class name. Examples include: gov.nih.nci.cacoresdk.domain.inheritance.childwithassociation.Bank |
| criteria | A string identifying the qualified or non-qualified criteria class name to be used as a filter/constraint on the result set. An example would be the SDK sample model Credit class that has an association to the Bank class via its issuingBank attribute.  If desired, the value of the id attribute of the criteria class instance can also be supplied in order to further constrain the result set. The pattern for such a criteria string is <criteria\_class\_name>[@id=<id\_value>]. An example might be Credit[@id=3], which indicates that only target/result class instances are returned that are associated to the Credit record with an id value of 3. |
| rolename | The name of the attribute within the criteria class that identifies the association to be traversed when retrieving the target/result class(es). An example would be the issuingBank attribute of the Credit class found within the sample SDK model.  The rolename property must be specified whenever the Criteria class has two or more attributes representing associations to the same target/result class type. One example would be the Child class within the sample SDK model that contains two attributes, mother and father, that both represent instances of the Parent class. In this scenario, specifying a value of rolename=mother or rolename=father within the REST URL would ensure that the correct Parent instance would be returned. |

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

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

While such a URL can be invoked directly from a browser, it is most frequently done so programmatically via a remote client program. An example of such a program, *TestGetXMLClient.java*, is provided in the *\output\example\package\remote-client\src* folder created by the SDK Code Generator. The diagram below provides a sample of the XML output produced from invoking the *Sample REST URL* above:

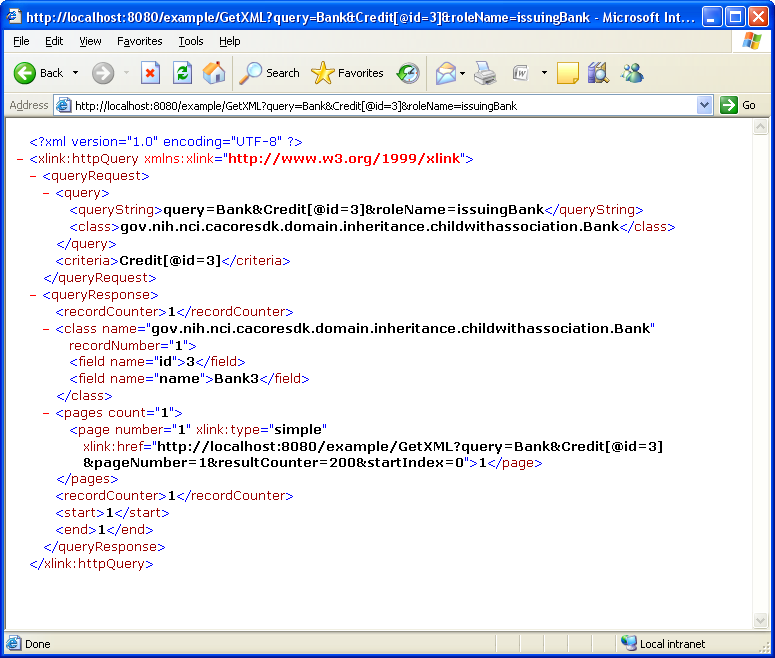


Figure 23: REST Call - Sample Output XML Data

## Java API Interface

At a very high-level, use of the Java API client involves two primary steps. The first step involves obtaining a reference to the instance of ApplicationService from ApplicationServiceProvider class. The second step involves invoking one of the interface methods in order to fetch the results from the SDK generated server component (local in case of local client).

Sample test programs illustrating how the SDK Java API can be used are provided as follows:

* *TestClient*.java: A sample *local* client located in the \*output\example\package\local-client\src\*folder
* *TestClient.java:* A sample *remote* client located in the \*output\example\package\remote-client\src\*folder.

More information about these test programs are provided in section .

### Obtaining ApplicationService

Access to the *ApplicationService* interface is provided via the *ApplicationServiceProvider* class. The *ApplicationServiceProvider* class provides several variations of a single method, shown below:

|  |  |
| --- | --- |
| **Primary Application Service Provider Method** | getApplicationService(service, url, username, password) |

 Figure 24: ApplicationServiceProvider Methods

The four required parameters required by the methods of ApplicationServiceProvider class are detailed in the following table:

|  |  |
| --- | --- |
| ApplicationService Parameter | Description |
| service | A string identifying the name of the Spring bean to use when configuring the ApplicationService instance. The bean represents a hash map and it is defined within the configuration file, application-config-client.xml, located within the /output/<project\_name>/package/[local|remote]-client/conf/ folder.  The default bean name (for those methods that do not require the service parameter) is ServiceInfo. This default hash map defines the following configuration properties:   * APPLICATION\_SERVICE\_BEAN: A reference to another Spring bean within the same configuration file that identifies the ApplicationService class to instantiate. * AUTHENTICATION\_SERVICE\_BEAN: A reference to another Spring bean within the same configuration file that identifies the authentication provider class to use when security is enabled. * APPLICATION\_SERVICE\_URL: The URL to the Spring DispatcherServlet configured within the SDK to handle remote Java API calls. The URL must conform to the following pattern : http://<server\_name>:<server\_port>/<project\_name> * APPLICATION\_SERVICE\_CONFIG: A reference to another Spring bean within the same configuration file that identifies a configuration string used when instantiating the ApplicationService instance. Note: This is an advance property setting, and should rarely, if ever, need to be changed. |
| url | A string identifying the URL to the remote service configured within the SDK to handle remote Java API calls. The URL must conform to the following pattern: http://<server\_name>:<server\_port>/<project\_name>. |
| username | A string identifying the username to use for both authentication and authorization purposes. Only required and valid when security is enabled. |
| password | A string identifying the password to use for authentication purposes. Only required and valid when security is enabled |

**Table 1 Primary ApplicationServiceProvider Method Parameters**

The *ApplicationServiceProvider* methods can be classified into two method groupings. The first group of methods returns an *ApplicationService* instance without requiring an Application Service URL. The second group, in contrast, all require that an Application Service URL be provided.

|  |  |
| --- | --- |
| **NOTE:** | The ApplicationServiceProvider methods requiring a URL are useful when overriding the default URL. These methods are also useful when multiple ApplicationService instances to differing SDK applications are desired. |

### ApplicationService API Methods

The SDK Java API consists of several query/search methods and a few other convenience methods that facilitate *read-only* access to domain data. A class diagram which highlights these methods is shown below:

******Figure 25: ApplicationService Interface Methods**

The *ApplicationService* methods are grouped into different categories and are discussed below.

### Convenience Query

The *ApplicationService* interface provides various convenience query methods, *which can be SDK users, but which are typically used by the SDK infrastructure*. The following table highlights these methods:

|  |  |
| --- | --- |
| ApplicationService Method | Description |
| getMaxRecordsCount() | Returns the maximum number of records the ApplicationService interface has been configured to return at one time. |
| getQueryRowCount(Object criteria, String targetClassName) | Returns the number of records that meet the search criteria. This method is used by the client framework to determine the number of list chunks in the result set. SDK users can also invoke this method in conjunction with the getMaxRecordsCount() method; however, this is not typical. |
| getAssociation(Object source, String associationName) | Retrieves an associated object for the example object specified by the source parameter. |

### HQL Query

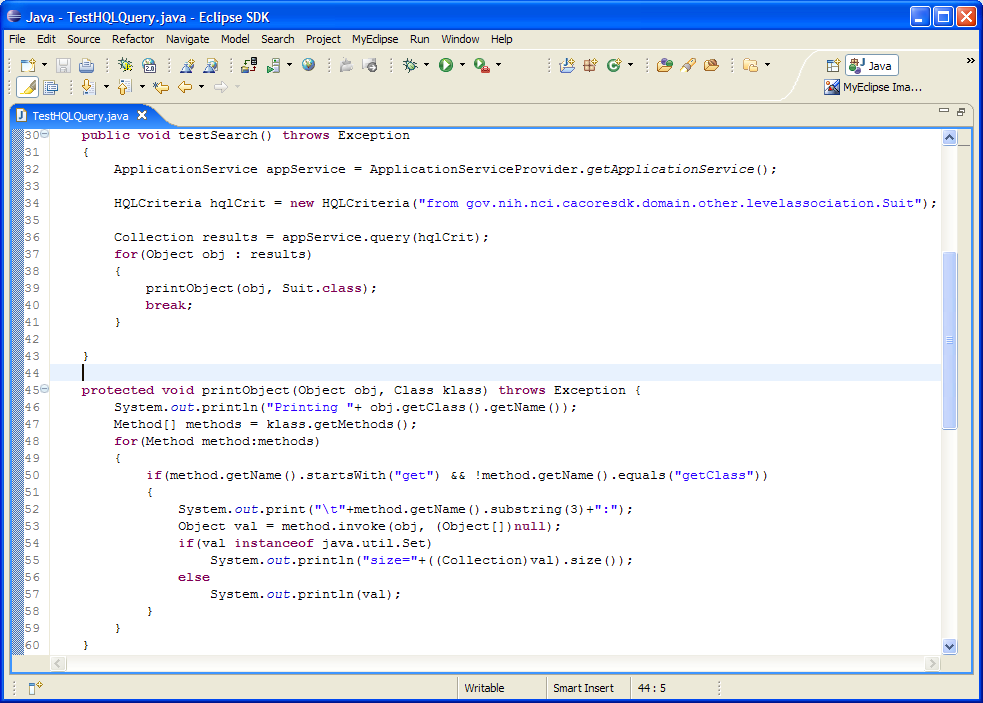
Hibernate is equipped with a powerful query language, called Hibernate Query Language (HQL), that looks very much like SQL. However, though the syntax is SQL-like, HQL is still fully object-oriented, and understands concepts like inheritance, polymorphism and association[[7]](#footnote-8). The SDK contains a wrapper class called *HQLCriteria*, which is used for submitting HQL queries. A diagram of this class is shown below:

******Figure 26: HQLCriteria Class Diagram**

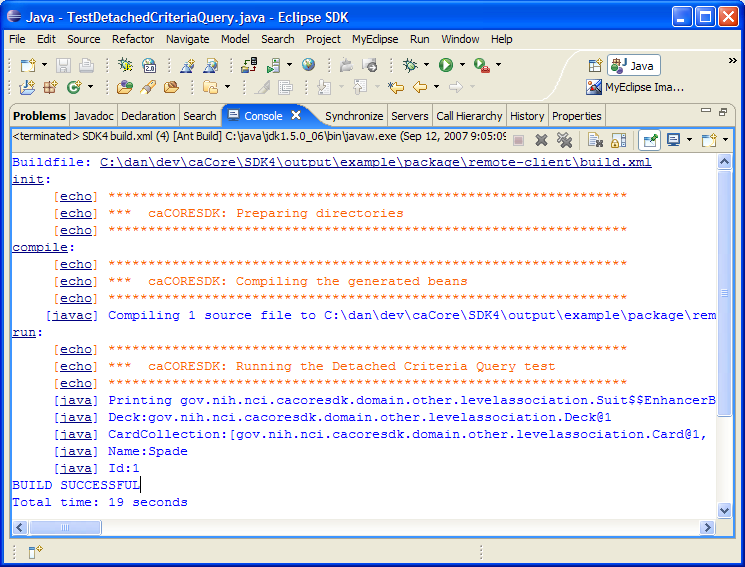
The following table highlights the HQL related *ApplicationService* methods:

|  |  |
| --- | --- |
| *ApplicationService* Method | Description |
| query(HQLCriteria hqlCriteria) | This method retrieves the results obtained by querying the data source using the Hibernate Query Language (HQL). As such, the data source must use Hibernate at the persistence tier. Internally, Hibernate executes the HQL query against the relational database and fetches the results.  Note: The retrieved results are converted into a list which may not be completely loaded. If the number of retrieved records is more than the maximum number of supported records as indicated by the getMaxRecordsCount() method, then the result set will only contain a subset of the total records. The client framework will execute a subsequent query (transparent to the client application) against the ApplicationService to load the remaining results in the list chunk. |
| query(HQLCriteria hqlCriteria, String targetClassName) | Deprecated. Internally calls the query(HQLCriteria hqlCriteria) method without the targetClassName parameter. |

The following figure shows how an SDK *HQLCriteria* object representing an HQL query might be instantiated and submitted:

****

**Figure 26: Sample HQL Query**

 **Figure 26: Sample HQL Query Results**

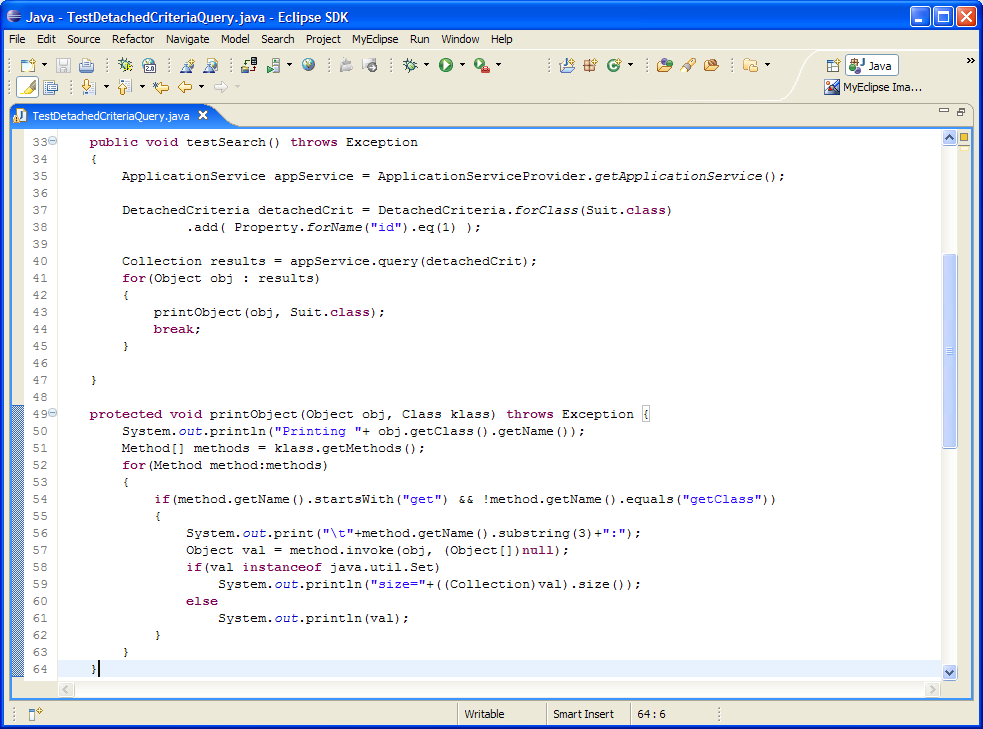
### Detached Criteria Query

While HQL is extremely powerful, some developers prefer to build queries dynamically, using an object-oriented API, rather than building query strings. To this end, Hibernate provides an intuitive *Criteria query API*[[8]](#footnote-9). The Hibernate *Detached Criteria* extends the Criteria concept, allowing Criteria queries to be created outside the scope of a session to be executed later using some arbitrary Hibernate Session.

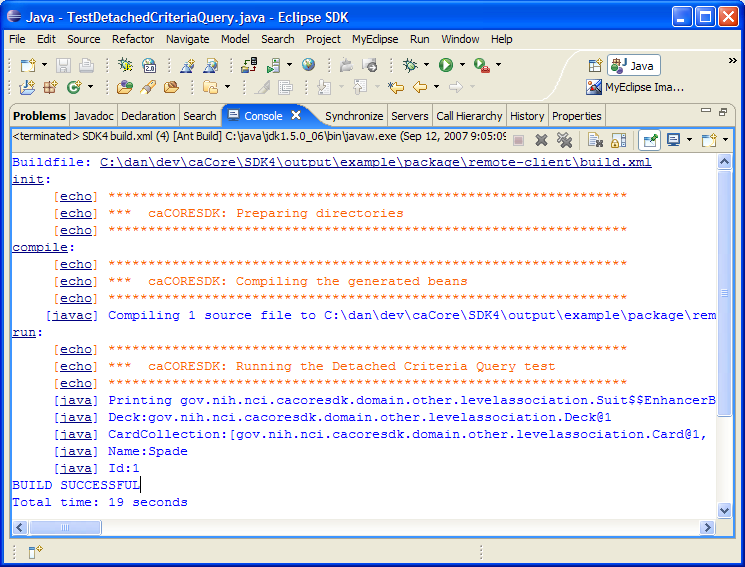
The following table highlights the *Detached Criteria* related *ApplicationService* methods:

|  |  |
| --- | --- |
| ApplicationService Method | Description |
| query(DetachedCriteria detachedCriteria) | Retrieves the result from the data source using the DetachedCriteria query object. The DetachedCriteria query structure can be used only by the Object Relational Mapping based persistence tier. Hibernate executes it against the relational database and fetches the results.  Note: The retrieved results are converted into a list which may not be completely loaded. If the number of retrieved records is more than the maximum number of supported records as indicated by the getMaxRecordsCount() method, then the result set will only contain a subset of the total records. The client framework will execute a subsequent query (transparent to the client application) against the ApplicationService to load the remaining results in the list chunk. |
| query(DetachedCriteria detachedCriteria, String targetClassName) | Deprecated. Internally calls the query(DetachedCriteria detachedCriteria)method without the targetClassName parameter. |

The following figure shows how a Hibernate *DetachedCriteria* object might be instantiated and the query submitted:



**Figure 26: Sample DetachedCriteria Query**

 **Figure 30: Sample DetachedCriteria Query Results**

### CQL Query

In addition to providing access to hibernate specific queries, SDK also provides language neutral SDK specific queries. CQL is one of such two query mechanisms. SDK CQL queries are modeled similarly to the object representation of the caBIG Query Language (CQL), which uses syntax similar to the Query-by-Example (QBE)[[9]](#footnote-10) query language to specify the way results are to be retrieved. The system formulates the query based on the navigation path specified in the query search criteria. The query mechanism allows the user to search for the objects using platform-independent query syntax.

The CQL query is represented by a complex object structure as shown in the diagram below. Starting object for CQL query is always CQLQuery object in which the user has to specify which object (target object) user would like to fetch from the database. The target object (CQLObject) is an example of the object that user intends to search. The example query object has space for 1)an attribute (CQLAttribute) 2)an association (CQLAssociation) and 3) a group (CQLGroup) of association collection and attributes collection.

If user intends to search for an object with one of its attributes called zipcode with value equal to 20852 then the user has to create a CQLObject with a CQLAttribute object populated inside it. CQLAttribute object will have its name attribute’s value as zipcode and value attribute’s value as 20852. CQLAttribute will also require a CQLPredicate for comparison between CQLAttribute and the database value. In this example the CQLPredicate of EQUAL\_TO will be selected. This example is equivalent to “where zipcode=20852”. CQLGroup allows logical grouping of other groups, attributes or associations. If user intends to create queries like “where zipcode=20852 and name like ‘%Dav%’ ” then in that case CQLGroup can be utilized.

****

**Figure 30:** **CQL Query Association Diagram**

The following table highlights the *CQLQuery* related *ApplicationService* query methods:

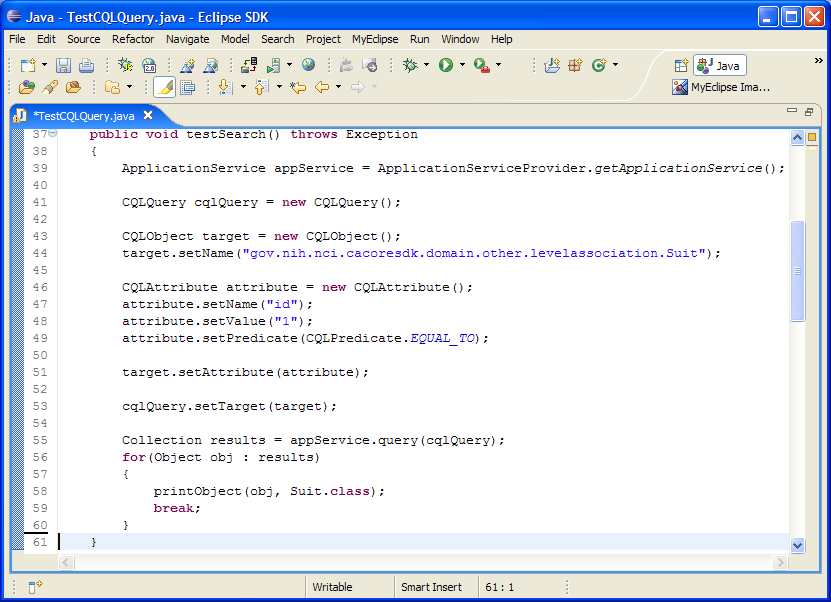
|  |  |
| --- | --- |
| ApplicationService Method | Description |
| query(CQLQuery cqlQuery) | Retrieves the query result from the data source using the CQL query syntax. Internally, CQL query structure is converted into Hibernate Query Language (HQL). Hibernate in turn converts the HQL into SQL and executes it against the relational database.  Note: The retrieved results are converted into a list which may not be completely loaded. If the number of retrieved records is more than the maximum number of supported records as indicated by the getMaxRecordsCount() method, then the result set will only contain a subset of the total records. The client framework will execute a subsequent query (transparent to the client application) against the ApplicationService to load the remaining results in the list chunk. |
| query(CQLQuery cqlQuery, String targetClassName) | Deprecated. Internally calls the query(CQLQuery cqlQuery) method without the targetClassName parameter. |

The following paragraphs provide an example of how to create and execute a CQL query using the *ApplicationService* interface. The first diagram shows classes from the sample SDK model package *gov.nih.nci.cacoresdk.domain.other.levelassociation*, and is provided as a point of reference:

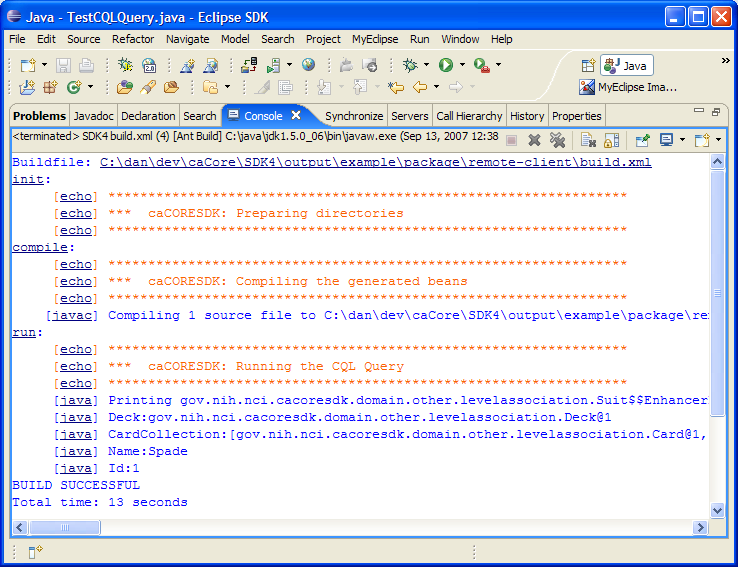


**Figure 32: Sample Domain Class Diagram**

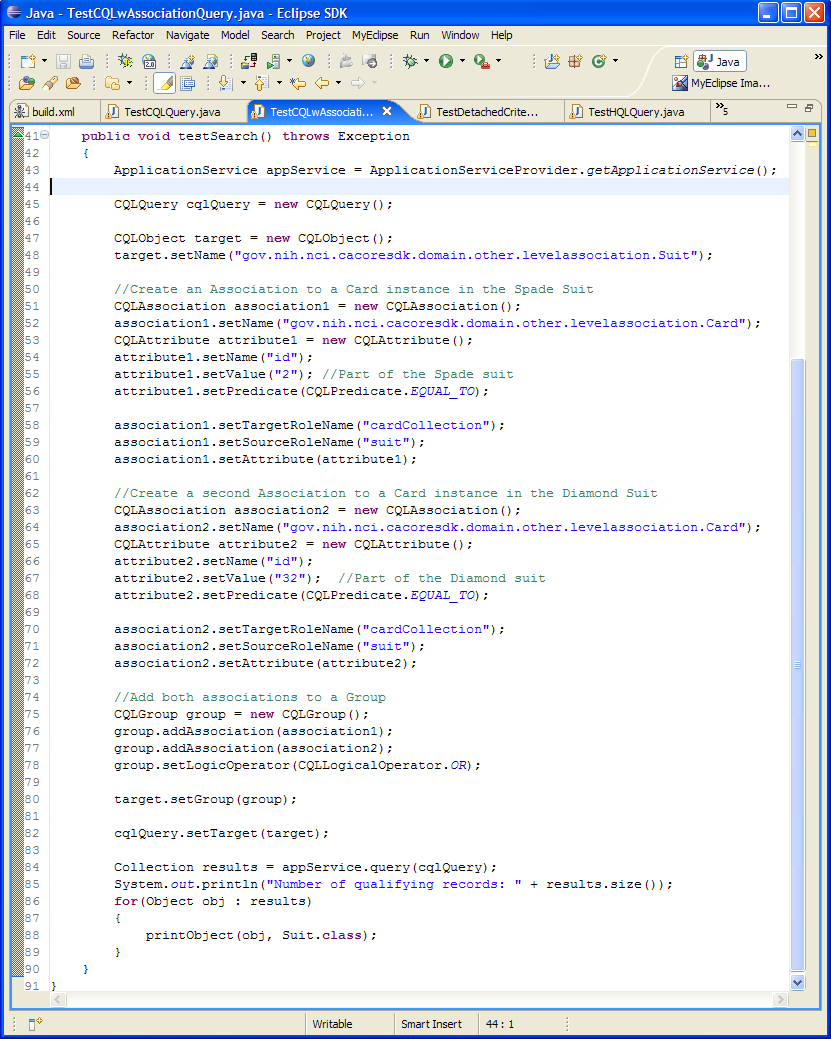
The following figure shows how an SDK CQL query object might be instantiated and the query submitted as “select \* from Suit where id=1”:

****

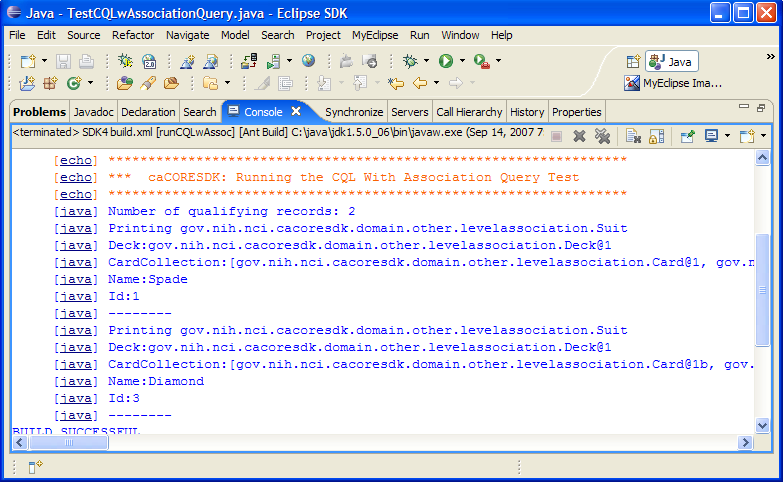
**Figure 32: Sample CQL Query without Association**

** Figure 34: Sample CQL Query without Association Results**

The following figure shows how an SDK CQL query object might be instantiated and the query submitted as “select \* from Suit where suit in (select suit from card where id=2 or id=32)”:

****

**Figure 34: Sample CQL Query with Association**



**Figure 34: Sample CQL Query with Association Results**

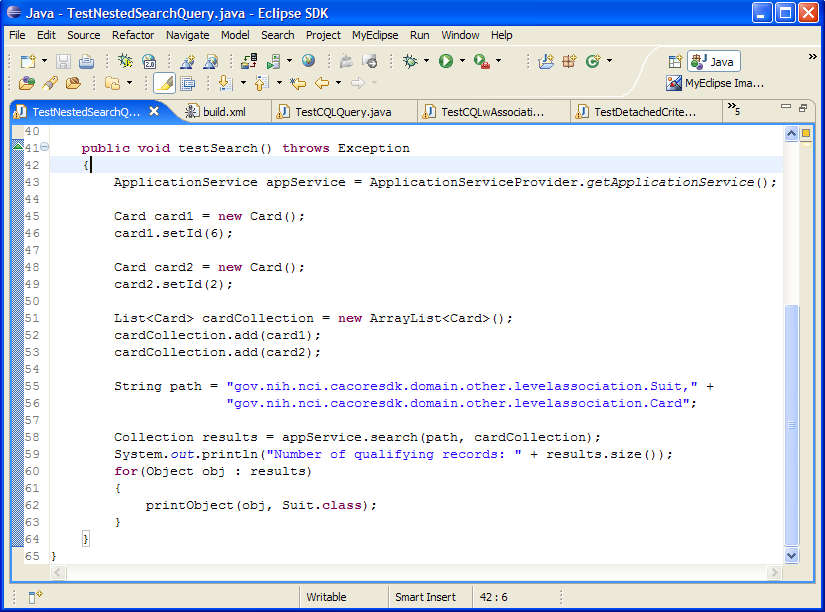
### Nested Search Criteria Query

SDK Nested Search Criteria queries are developed specifically for SDK. Nested Search Criteria query has two parts 1) Comma separated path to target search object and 2) example of the source object. Comma separated path starts with the target object (fully qualified name of the class) that the user intends to retrieve from the database. Next item in the comma separated path is a link in the chain is an element (fully qualified name of the class) which helps to reach the element to its left from the element to its right. The element in the right could be the example object or another element in the chain. In other words, it provides a mechanism to traverse from example object to the object that is desired using comma separated path.

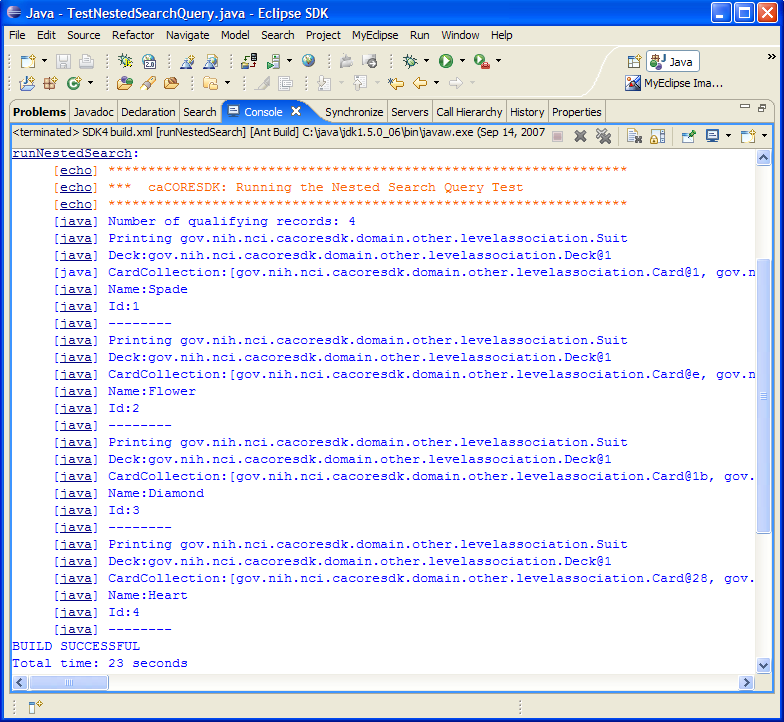
The following table highlights the *Nested Search Criteria* related *ApplicationService* methods:

|  |  |
| --- | --- |
| ApplicationService Method | Description |
| search(String path, List<?> objList) | Retrieves the result from the data source using a Nested Search Criteria. The path specifies the list of objects (separated by commas), which should be used to reach the target object from the example objects passed in the objList, or the associated object for the example object. Internally, the Nested Search Criteria is converted into the data source specific query language. For the Object Relational Mapping based persistence tier, the query structure is first converted into the Hibernate Query Language (HQL). Hibernate then converts the HQL into SQL and executes it against the relational database.  Note: The retrieved results are converted into a list which may not be completely loaded. If the number of retrieved records is more than the maximum number of supported records as indicated by the getMaxRecordsCount() method, then the result set will only contain a subset of the total records. The client framework will execute a subsequent query (transparent to the client application) against the ApplicationService to load the remaining results in the list chunk. |
| search(Class targetClass, List<?> objList) | Retrieves the result from the data source using the Query by Example query language. The targetClass specifies the object that the user intends to fetch after executing the query. The targetClass should be the same as the object specified in the objList or associated object for the example object. All the objects in the objList have to be of the same type. The example query is converted into the data source specific query language. For the Object Relational Mapping based persistence tier, the example query structure is first converted to a Nested Search Criteria, and then to Hibernate Query Language (HQL). Hibernate then converts the HQL into SQL and executes it against the relational database.  Note: The retrieved results are converted into a list which may not be completely loaded. If the number of retrieved records is more than the maximum number of supported records as indicated by the getMaxRecordsCount() method, then the result set will only contain a subset of the total records. The client framework will execute a subsequent query (transparent to the client application) against the ApplicationService to load the remaining results in the list chunk. |
| search(Class targetClass, Object obj) | Retrieves the result from the data source using the Query by Example query language. The targetClass specifies the object that the user intends to fetch after executing the query. The targetClass should be same as the example object or associated object for the example object. The example query is first converted into the data source specific query language. For the Object Relational Mapping based persistence tier, the example query structure is first converted to a Nested Search Criteria, and then to Hibernate Query Language (HQL). Hibernate finally converts the HQL into SQL and executes it against the relational database.  Note: The retrieved results are converted into a list which may not be completely loaded. If the number of retrieved records is more than the maximum number of supported records as indicated by the getMaxRecordsCount() method, then the result set will only contain a subset of the total records. The client framework will execute a subsequent query (transparent to the client application) against the ApplicationService to load the remaining results in the list chunk. |
| search(String path, Object obj) | Retrieves the result from the data source using the Nested Search Criteria. The path specifies the list of objects (separated by commas) which should be used to reach the target object from the example object passed as obj, or the associated object for the example object. Internally, the Nested Search Criteria is converted into the data source specific query language. For the Object Relational Mapping based persistence tier, the query structure is first converted into the Hibernate Query Language (HQL). Hibernate then converts the HQL into SQL and executes it against the relational database.  Note: The retrieved results are converted into a list which may not be completely loaded. If the number of retrieved records is more than the maximum number of supported records as indicated by the getMaxRecordsCount() method, then the result set will only contain a subset of the total records. The client framework will execute a subsequent query (transparent to the client application) against the ApplicationService to load the remaining results in the list chunk. |

Following figure demonstrates how to use the nested search criteria. Here we want to retrieve Suit class from the database from the Card as example. We have two different instances of the Card object inside the cardCollection which will be ORed and their corresponding Suit will be retrieved. The resulting query will be as “select \* from Suit where suit in (select suit from card where id=2 or id=6)”:

****

**Figure 37: Sample Nested Search Criteria Query**



**Figure 38: Sample Nested Search Criteria Query Results**

## Web Service Interface

The SDK 4.0 Web Services is based on the Axis 1.4 framework, which adheres to the J2EE 1.4 server programming model described by JAX-RPC and JSR 109[[10]](#footnote-11); i.e., the SDK 4.0 Web Services uses the Remote Procedure Call (RPC) Web Service style[[11]](#footnote-12).

|  |  |
| --- | --- |
| **NOTE:** | While the SDK Web Service continues to be based on the Axis 1.4 framework, the extraneous .ws layer found in previous SDK versions has been eliminated.  Also, the SDK Web Service Deployment Descriptor (WSDD) is now packaged along with the rest of the SDK generated system, thus allowing for automatic deployment of the Web Service; i.e., manual deployment of the Web Service is no longer required. |

A sample test program illustrating how the SDK generated Web Service can be consumed, *TestClient.java*, is provided in the *\output\example\package\ws-client\src* folder. More information about this test program is provided in section .

The remainder of this section provides specifications for the SDK generated Web Service via the use of the Web Services Description Language (WSDL), including an overview of the schema imports, service, port types, and messages found within the WSDL.[[12]](#footnote-13)

### SDK WSDL Directives - Schema Imports

The diagram below provides a listing of the schema imports found within the WSDL for the *sample SDK model.* It is provided here in order to emphasize that a schema import statement is added to the WSDL for each of the distinct domain package(s) found within the model provided to the SDK Code Generator:

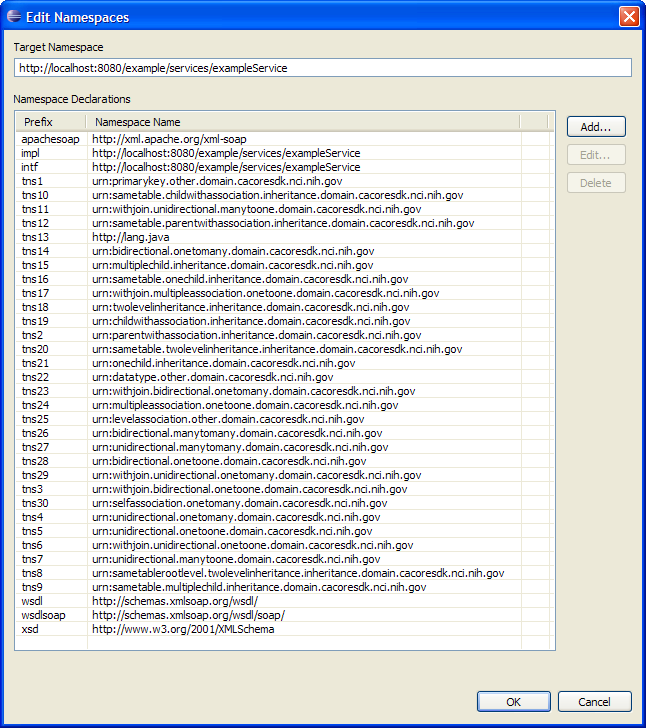
****

Figure 22: Sample WSDL Directives - Schema Imports

### WSDL Service Definition

The WSDL defines a Web Service as a collection of network endpoints, or *ports*. The following diagrams provide details for the SDK generated Web Service defined within the WSDL, which includes it’s:

* Name
* Prefix
* Target Namespace, and
* Port Information

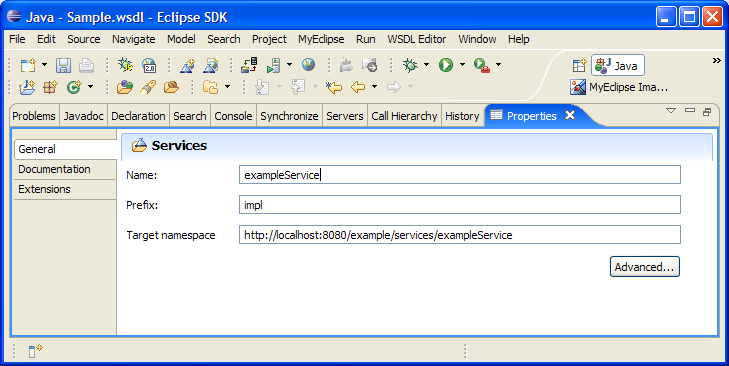
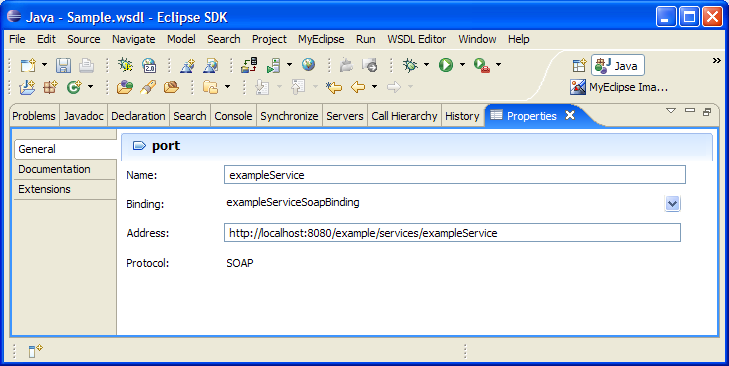
****

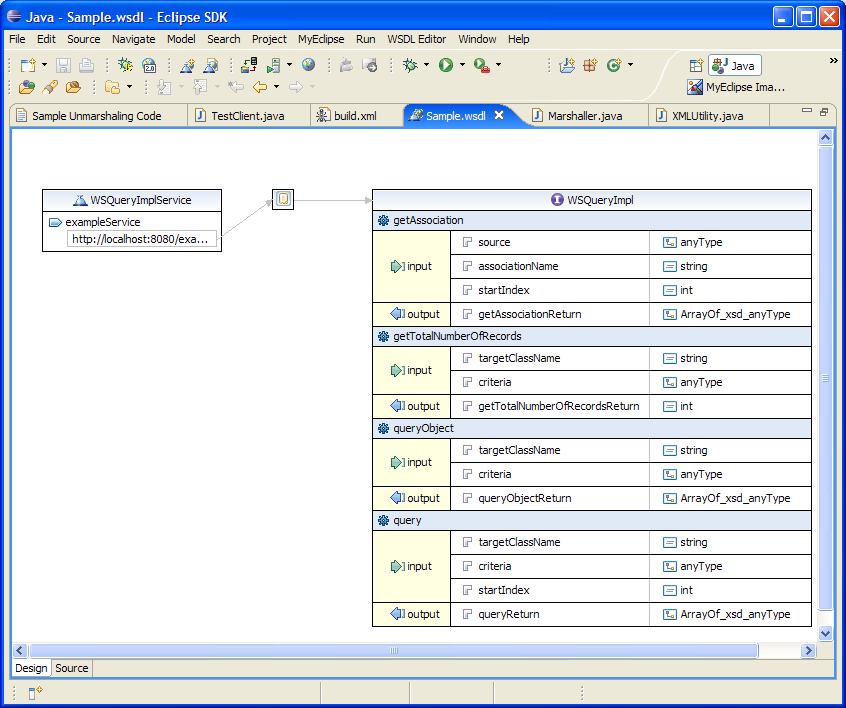
Figure 22: Sample WSDL Service Definition

 Figure 22: Sample WSDL Service Definition – Port

|  |  |
| --- | --- |
| **NOTE:** | The SDK Code Generator uses the value of the PROJECT\_NAME property provided within the deploy.properties file (in this case, “example”) while generating the WSDL. So, while the information displayed above is specific to the sample SDK model, the same pattern is followed in the generation of the WSDL Service and Port definitions for other models. |

### WSDL Port Types (Network Endpoints)

The WSDL defines a port as an association of a network address with a reusable binding. Port types, in turn, are abstract collections of supported operations. The diagram below provides a summary of the collection of network endpoints (and their messages) that composes any SDK generated Web Service:

  
Figure 37: WSDL Port Types (Network Endpoints)

### Messages, Elements, and Types

The WSDL defines messages as abstract descriptions of the data being exchanged. The concrete protocol and data format specifications for a particular port type constitutes a reusable binding, where the messages and operations are then bound to a concrete network protocol and message format. The following table provides a summary of the messages and elements (including parameters and data types) that make up the Web Service defined in the WSDL for the sample SDK model.

|  |  |
| --- | --- |
| Message | Description |
| getAssociationRequest | The getAssociationRequest message is used by a Web Service client to request object(s) associated to a given Java domain object instance. Required parameters include:   * source: An instance of the Java domain object containing the association (rolename) method to be invoked; * associationName: The name of the method (rolename) that represents the associated object(s) to be returned; * startIndex: The starting index into the resulting dataset. Useful during subsequent calls when “scrolling” through a large result dataset. Initial requests should set the value of this parameter to zero (0). |
| getAssociationResponse | The getAssociationResponse message is used by the SDK server to provide any qualifying objects associated to the source Java domain object. The response is an array of qualifying objects. |
| getTotalNumberOf RecordsRequest | The getTotalNumberOfRecordsRequest message is used by a Web Service client to request a count of total number of records that would be returned for a given search criteria. Required parameters include:   * targetClassName: The fully qualified class name of the search object type to be returned. This may represent the name of the criteria object class itself, or the name of a class associated to the criteria object. * criteria: a sample instance of the criteria search object, containing values for any desired field(s) (attributes) that should act as a filter (constraint) on the resulting dataset; |
| getTotalNumberOf RecordsResponse | The getTotalNumberOfRecordsResponse message is used by the SDK server to provide a count of the total number of records that would be returned for a given search criteria. The response type is a positive integer (int), or zero, if no qualifying records are found. |
| queryRequest | The queryRequest message is used by a Web Service client to request to request object(s) that meet the supplied search criteria. Internally, a nested search criteria is performed. Required parameters include:   * targetClassName: The fully qualified class name of the search object type to be returned. This may represent the name of the criteria object class itself, or the name of a class associated to the criteria object. * criteria: a sample instance of the criteria search object, containing values for any desired field(s) (attributes) that should act as a filter (constraint) on the resulting dataset * startIndex: The starting index into the resulting dataset. Useful during subsequent calls when “scrolling” through a large result dataset. Initial requests should set the value of this parameter to zero (0). |
| queryResponse | The queryResponse message is used by the SDK server to return any objects that meet the search criteria passed via the queryRequest message. The response is an array of qualifying objects. |
| queryObjectRequest | The queryObjectRequest message is used by a Web Service client to request object(s) that meet the supplied search criteria. Required parameters include:   * targetClassName: The fully qualified class name of the search object type to be returned. This may represent the name of the criteria object class itself, or the name of a class associated to the criteria object. * criteria: a sample instance of the criteria search object, containing values for any desired field(s) (attributes) that should act as a filter (constraint) on the resulting dataset.   Note: The queryObjectRequest operation has the same effect as invoking the queryRequest message with a startIndex of zero (0). A different operation/message name had to be used as the Axis 1.4 framework does not seem to allow the “overloading” of method signatures. |
| queryObjectResponse | The queryObjectResponse message is used by the SDK server to return any objects that meet the search criteria passed via the queryObjectRequest message. The response is an array of qualifying objects. |

### Web Service Error Handling

The errors that may be generated during a message exchange between a Web Service client and a generated SDK system Web Service fall into one of the two following categories:

* Those that would be generated by the generated SDK application, and
* Those that would be generated by any of the framework API’s used during the message exchange between systems.

In both instances, a SOAP Fault[[13]](#footnote-14) element will handle the transport of error messages. The application-related errors occur when the SDK generated cannot fulfill a request from a Web Service client. For example, when a Web Service client sends a *getAssociationRequest*  message and supplies an invalid *associationName* value. In the case of the Web Services framework API, an error could occur when a message cannot reach its destination. An example of that would be interruption in network, issue with the message structure, or message load, etc. In these instances, the Web Services framework will generate an error relevant to the incident and a SOAP Fault element will transport the message to the client, if it can be delivered.

### SOAP Fault Structure

As stated above, the SOAP Fault element is used to carry error and/or status information within a SOAP message. If a Fault element is present, it must appear as a child element of the Body element. A Fault element can only appear once in a SOAP message.

The SOAP Fault element has the following sub elements:

|  |  |
| --- | --- |
| Sub Element | Description |
| <faultcode> | A code for identifying the fault |
| <faultstring> | A human readable explanation of the fault |
| <faultactor> | Information about who caused the fault to happen |
| <detail> | Holds application specific error information related to the Body element |

**Table 2 SOAP Fault Structure Element Descriptions**

# System Usage – Secured System

This section provides details on how to use the SDK generated runtime system when security is enabled. When the generated system is secured, the user of the system is required to perform the login operation before making any query to the system. Depending on the client interface the user decides to use, the login operation varies. Given below is the information on logging in the individual client interface. Once the login operation is complete, querying the system can be done in the same fashion as explained in the previous chapter.

## XML-HTTP Interface

The user of the XML-HTTP interface can access the data using two types of the clients 1) Browser to see data in the form of a web page and 2) Thin client to get XML data. Browser based clients have security configured through the Form based authentication. User has to type in the username and the password in the login form provided by the application. Note that the login form appears on home page whenever the secured system is generated.

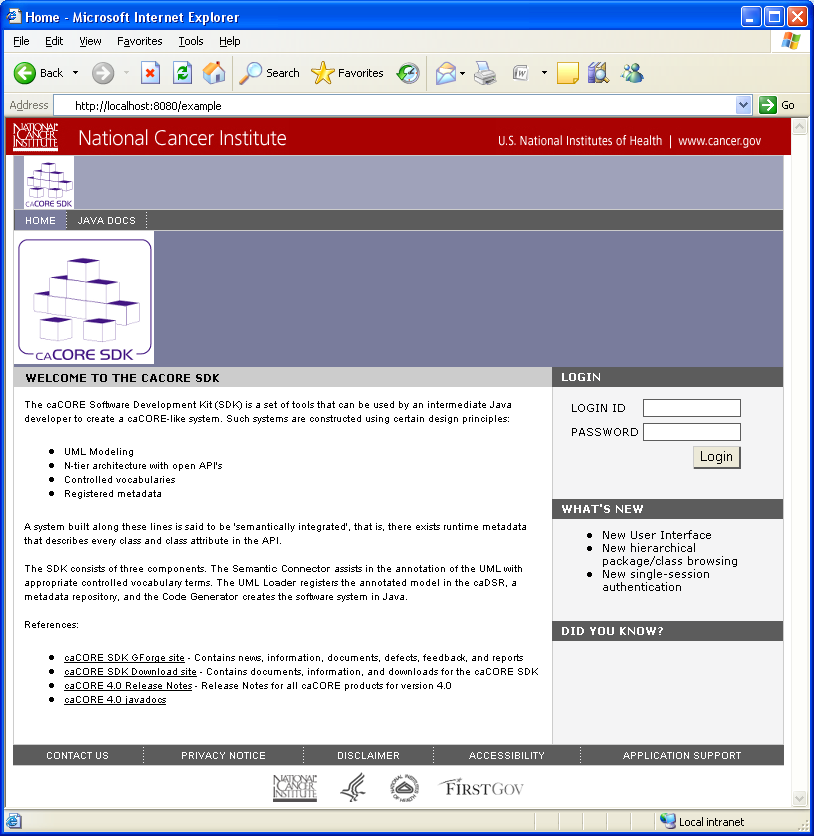


Figure 14: Security Login in the Web System

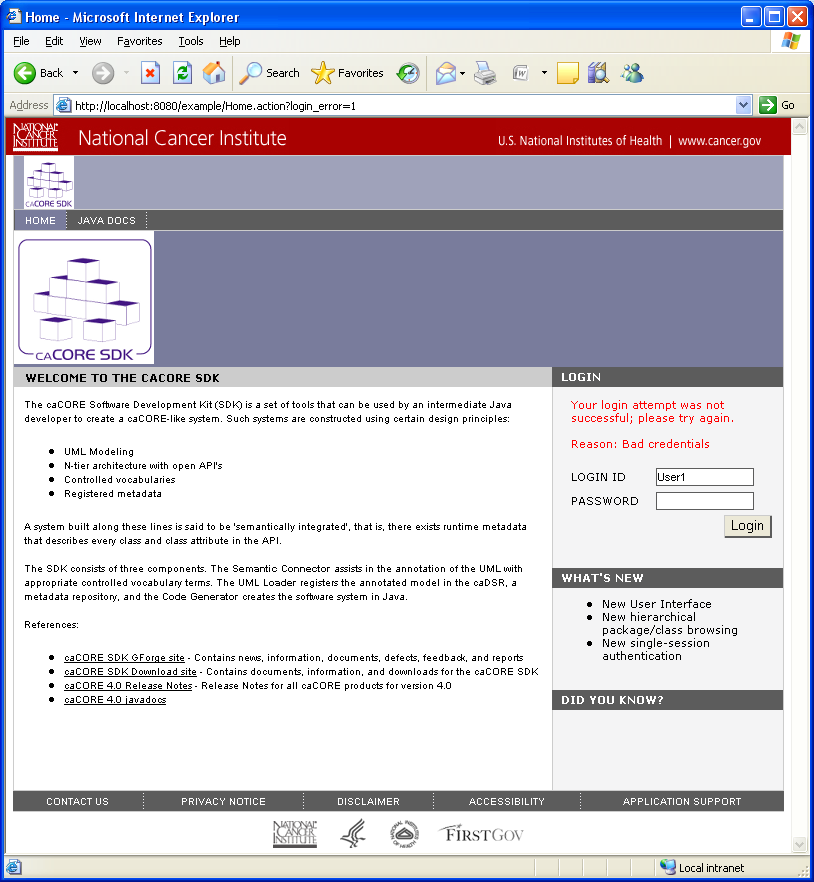


Figure 15: Security Login Failure in the Web System

On unsuccessful login, the client will get error message on the screen as shown in the diagram above. After three consecutive unsuccessful login attempts, the user’s login account will be locked out by CSM for 30 minutes.

User of the XML based REST interface communicates with the SDK generated application using a thin client application. If the SDK generated system is secured then in that case, the thin client application is required to provide the username and the password using BASIC[[14]](#footnote-15) [[15]](#footnote-16) authentication. Under BASIC authentication, the username and password are encrypted using Base64 encryption and are supplied as part of the HTTP header to the server. The server side component will decrypt the headers using the corresponding decryption logic and try logging in the user in the application. The code snippet below demonstrates setup of BASIC authentication using Java API.

|  |
| --- |
| URLConnection conn = url.openConnection();  String base64 = "userId" + ":" + "password";  conn.setRequestProperty("Authorization", "Basic " + new String( org.apache.commons.codec.binary.Base64.encodeBase64(base64.getBytes()))); |

Figure 16: Security Login in Java based REST (XML) client

Note that when the user is using the REST interface, the thin client application will be required to provide the username and password in each and every call that is made to the server.

## Web Services Client

Secured SDK generated web services user is required to supply the user credentials in the form of a web service message header. As part of the web service message, a new header is required to be added to the web service call, called *SecurityHeader*. This header will have a element called security with two child elements named *username* and *password* respectively. Value of these child elements reflect the user’s login username and login password for the underlying application. The code snippet given below demonstrates the usage of *SecurityHeader* in Java.

|  |
| --- |
| SOAPHeaderElement headerElement = new  SOAPHeaderElement(call.getOperationName().getNamespaceURI(),"SecurityHeader");  headerElement.setPrefix("security");  headerElement.setMustUnderstand(false);  SOAPElement usernameElement = headerElement.addChildElement("username");  usernameElement.addTextNode("userId");  SOAPElement passwordElement = headerElement.addChildElement("password");  passwordElement.addTextNode("password");  call.addHeader(headerElement); |

Figure 17: Security Login in Java based web services client

Note that the web service communication is stateless. Hence, the user of the web service is required to provide the login information in the header of the message each time it is making a call to the server.

## Java API

Using security with the Java API client is a simple one step process. The user is required to use the methods which accept the username and password to obtain the reference to the ApplicationService from the ApplicationServiceProvider class. If the username and password are passed as parameters then the ApplicationServiceProvider class validates the username and password against the authentication service and if successful, logs the user in the application.

|  |
| --- |
| ApplicationService appService = ApplicationServiceProvider.getApplicationService(“userId”, “password”) |

Figure 18: Security Login in Java API client

Note that in addition to the example shown above, there are other additional convenience methods in the ApplicationServiceProvider class which allows user to login on a different service or different URL.

# Performance Tuning – Java API

SDK Generated Java API provides a very powerful feature of creating the data service in small amount of time. As the SDK is just a tool to generate the API, it cannot understand all the use cases of the user’s application and hence cannot provide a comprehensive solution to requirements of all the users. The development team of SDK and many of the SDK users have come across many of these problems and they have discovered several solutions to improve performance. This section makes an attempt to suggest some of the solutions discovered by these users.

**Database Indexes** – Missing or corrupt indexes can explain performance problems for most queries. Most database modeling tools provides an option to create indexes for the primary key and foreign keys; however, the database indexes have been found missing or corrupted due to various reasons including batch data load and recreating the records. Fixing the indexes should improve the performance of the queries. Proper indexes on the primary and foreign key columns will definitely improve performance for the database table joins. The user may have to create additional indexes for the columns which are more likely to be hit from the end user search.

**Fine Tuning the Page Size** – SDK user can choose the page size for the SDK generated system at the time of generating system. There are two kinds of pages for the generated system. First is for the maximum number of records (rowCounter) that can be displayed to the user of the web interface. Second is the maximum number of records (resultCountPerQuery) that can be fetched by the Java API per call. Both of these properties can be altered in the file named application-config.xml which is available under /conf/system/web/WEB-INF/classes folder of SDK distribution. By default, maximum number of records shown to the user of the web application is set to 200 and maximum number of records that can be fetched by Java API in one call to the server is set to 1000. Based on the nature of the underlying data, the developer of the application can choose appropriate page size.

**Lazy Loading** – By default, SDK generated application will fetch only the objects demanded by the query. For e.g. a query for person object will result in fetching of only Person object and none of its associated objects. When the client application invokes in the getter method to retrieve the associated object (for e.g getAddress() of a Person object) then at that time, the SDK generated application seamlessly connects to the server to retrieve the associated object. This approach known as lazy loading will result in delayed response from the application due to additional call over the network. User of the SDK can override this default behavior in two ways

* 1. Disabling lazy load for certain associations of the object through the O/R mapping – Based on the use case of the system, user of the SDK code generator can specify a UML tag-value with key as “lazy-load” and value for the key as “no” on the association which the user intends to fetch eagerly (or not lazily). This approach works for unidirectional association only for now.
  2. Using Hibernate’s DetachedCriteria for eager loading of associated objects – This approach is a more flexible approach of two. The user of the SDK generated runtime system can use DetachedCriteria[[16]](#footnote-17) from Hibernate. DetachCriteria allows user to specify eager loading option for some of the associations for the queried object.

**Hibernate Query Language (HQL)[[17]](#footnote-18)** – SDK generated queries from SDK’s Nested Search Criteria and SDK’s CQL search criteria results in fetching of the complete domain object from the database. At the same time, the database queries generated by the SDK specific search criteria can result in poor performance. User of the SDK has an option to use the HQL queries to fetch the domain objects from the data service. User can choose to retrieve selected attributes of the domain object and not the complete object if he wishes to do so by writing more granular HQL query.

# Utilities

## XML Utility (Marshalling and Unmarshalling)

While used primarily by caGRID, the caCORE SDK does provide a class, XMLUtility.java, which can be used to marshal (serialize) the generated domain Java Beans to XML, and unmarshal (deserialize) XML data back to the generated domain Java Beans.

  
Figure 37: XML Utility Class Diagram

As implied by the XMLUtility Constructor method, the XMLUtility class wraps both a SDK Marshaller and Unmarshaller class, which it is dependent upon in order to perform its work. These collaborating classes and their interfaces are discussed in the following sub-sections. For more information, also see section .

### 

**The caCOREMarshaller Class**

The SDK *caCOREMarshaller* class implements the SDK *Marshaller* interface, and is used by the *XMLUtility* class to perform the actual work of marshalling (serializing) domain Java Bean objects to XML.

|  |  |
| --- | --- |
| NOTE: | The caCOREMarshaller class is used internally by the XML Utility infrastructure, and is not typically invoked by the end user. |

  
Figure 37: Marshaller Class Diagram

The *caCOREMarshaller* uses *Castor*[[18]](#footnote-19) technology, and utilizes the SDK generated *xml-mapping.xml* file, which provides Java-to-XML binding settings used by the Castor engine. Mappings are included for value attributes, collections, and associations to other domain Java Beans.

|  |  |
| --- | --- |
| NOTE: | When processing associations and collections, the caCOREMarshaller also uses custom Castor collection and domain object Field Handlers. This is done in order to prevent infinite recursion whenever domain classes have circular references/associations to each other. Consequently, associations and collections are only serialized to their first level. |

**The caCOREUnmarshaller Class**

The SDK *caCOREUnmarshaller* class implements the SDK *Unmarshaller* interface, and is used by the *XMLUtility* class to perform the actual work of unmarshalling (deserializing) XML to domain Java Bean objects.

|  |  |
| --- | --- |
| NOTE: | The caCOREUnmarshaller class is used internally by the XML Utility infrastructure, and is not typically invoked by the end user. |

  
Figure 37: Unmarshaller Class Diagram

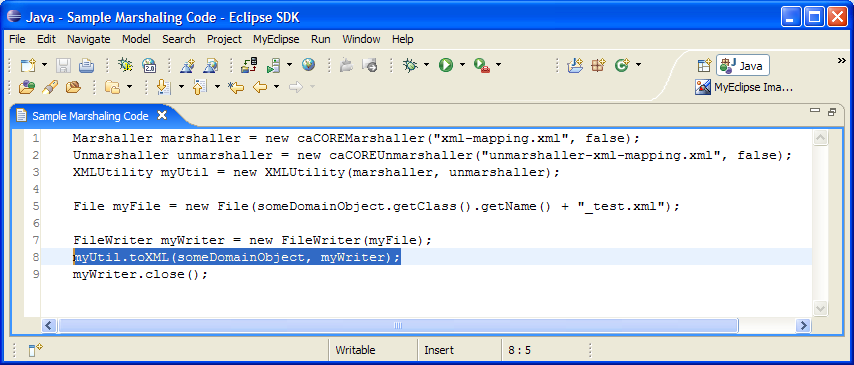
The *caCOREUnmarshaller* uses *Castor*[[19]](#footnote-20) technology, and utilizes the SDK generated *unmarshaller-xml-mapping.xml* file, which provides XML-to-Java binding settings used by the Castor engine. Mappings are included for value attributes, collections, and associations to other domain Java Beans.

### Marshaling Java Objects to XML

The XMLUtility class provides two wrapper methods for marshaling (serializing) domain Java objects to XML, as described in the following table:

|  |  |
| --- | --- |
| XMLUtility Method | Description |
| toXML(Object beanObject) | Accepts a domain Java Bean instance and passes it to the Marshaler instance (caCOREMarshaller, by default), which in turn marshals (serializes) the instance to XML and returns it as an XML string. |
| toXML(Object beanObject, Writer stream) | Accepts a domain Java Bean instance. This object is similarly passed to the Marshaler instance (caCOREMarshaller, by default), which marshals (serializes) it to XML. However, the XMLUtility then writes the serialized XML string to a character stream writer instead. |

The following code snippet demonstrates how one of the XML Utility marshaling methods might be invoked:

Figure Sample Marshaling Code

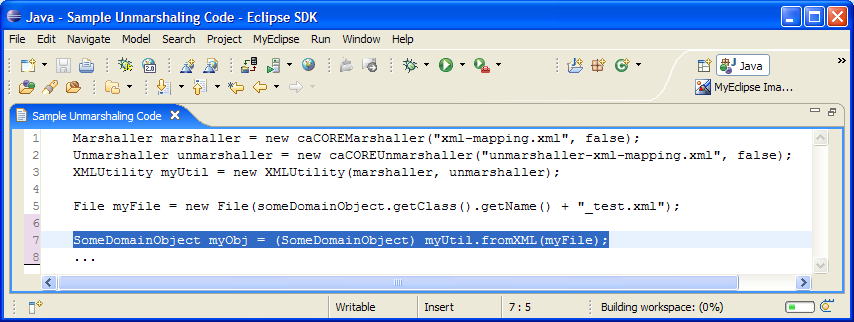
A sample test program, TestXMLClient.java, is also provided in the \output\example\package\remote-client\src folder. More information about this test program is provided in section  *.*

### Unmarshaling XML to Java Objects

The XMLUtility class also provides two wrapper methods for unmarshalling (deserializing) XML to domain Java objects, as described in the following table:

|  |  |
| --- | --- |
| XMLUtility Method | Description |
| fromXML(File xmlFile) | Instantiates a domain Java Bean object from an XML file that contains the serialized output of that object. |
| fromXML(Reader input) | Also instantiates a Java Bean domain object from XML, but reads it instead from a *java.io.Reader* character stream. |

The highlighted portion of the following code snippet demonstrates how one of the XML Utility unmarshaling methods, fromXML(File), can be invoked:

 Figure 37: Sample Unmarshaling Code

A sample test program, TestXMLClient.java, is also provided in the \output\example\package\remote-client\src folder. More information about this test program is provided in section  *.*

# Configuring and Running SDK

## SDK Configuration Properties

The SDK Code Generator is configured, for the most part, by a single file, deploy.properties, which is located within the /conf folder under SDK’s distribution.

The following table describes each of the properties (and their values) found within this file.

|  |  |  |
| --- | --- | --- |
| Property | Default Value | Description |
| PROJECT\_NAME | example | Used in the creation/naming of the following items:   * Output project directory folder name * Beans JAR file name * ORM JAR file name * Client JAR file name * WAR file name * Web Service Namespace * Documentation title in the generated API (Javadocs) * Server URL context value   SDK users should modify this property to reflect their own project name. |
| NAMESPACE\_PREFIX | gme://caCORE.caCORE/3.2/ | Used in the creation/naming of the following code generation artifacts:   * Schemas (XSD’s) * XML Marshalling and Unmarshalling Mapping files   If XSD’s are to be used for the caGRID, the value of the NAMESPACE\_PREFIX is the same as the GME namespace value |
| SECURITY\_ENABLED | false | Used to enable or disable security within the generated system during code generation.  This applies to all of the SDK interfaces, including:   * Web Interface (GUI) * Java API Interface (local and remote clients) * Web Service Interface |
| CSM\_PROJECT\_NAME | sdk | Used as a prefix when creating the CSM security configuration file name. CSM configuration should have the same application name configured |
| INSTANCE\_LEVEL \_SECURITY | false | Used to enable/disable CSM instance level security.  Only relevant if the SECURITY\_ENABLED property is set to ‘true’ |
| ATTRIBUTE\_LEVEL \_SECURITY | false | Used to enable/disable attribute level security.  Only relevant if the SECURITY\_ENABLED property is set to ‘true’ |
| WEBSERVICE\_NAME | ${PROJECT\_NAME}Service | The name of the Web Service. |
| SERVER\_TYPE | other | Used to include/exclude the log4j.jar file during the war file packaging. If set to ‘jboss’ will exclude log4j.jar from the war file, as the JBoss server already has its own instance of the log4j.jar file. Any other value will include the log4j.jar in the war file.  Valid values are ‘jboss’ if deploying to a JBoss server, and ‘other’ if deploying to any other type of Servlet container such as Apache Tomcat. |
| SERVER\_URL | http://localhost:8080/ ${PROJECT\_NAME} | The URL (including the application context) of the deployed application. Used as part of the URL that specifies the location of the deployed Web Service. I.e., the following pattern is used when undeploying the Web Service from the server:  ${SERVER\_URL}/services/${WEBSERVICE\_NAME}Service |
| MODEL\_FILE | sdk.xmi | The name of the file which contains the object/data model be processed.  SDK users should modify this property to reflect their own model file name. The file is to be placed under the \models directory. |
| MODEL\_FILE\_TYPE | EA | The file type of the object/data model file to be processed.  Valid values are ‘EA’ for Enterprise Architect files, and ‘ARGO’ for ArgoUML files . |
| LOGICAL\_MODEL | Logical View.Logical Model | The logical model base (root) package/folder name containing the domain package(s) and class(es) to be processed by the Code Generator. |
| DATA\_MODEL | Logical View.Data Model | The data model base (root) package/folder name containing the data model package(s) and class(es) to be processed by the Code Generator. |
| INCLUDE\_PACKAGE | domain | Used to determine which packages within the model should be included during code generation; i.e., all packages containing this property value as part of their fully qualified name will be included during code processing. |
| EXCLUDE\_PACKAGE | none | Used to determine which packages within the model should be excluded during code generation; i.e., all packages containing this property value as part of their fully qualified name will be excluded during code processing.  NOTE: All packages are first filtered/constrained by the INCLUDE\_PACKAGE property value, and then furthered filtered by the EXCLUDE\_PACKAGE value. |
| EXCLUDE\_NAME | none | Used to determine which classes within the model should be excluded during code generation; i.e., all classes containing this property value as part of their non-fully qualified name will be excluded during code processing. |
| USE\_JNDI\_BASED \_CONNECTION | no | Indicates whether a JNDI DB Connection should be used for the application database.  If USE\_JNDI\_BASED\_CONNECTION=yes, then the DB\_JNDI\_URL property value is used to obtain the DB connection and retrieve data |
| DB\_JNDI\_URL | java:/SDK | The DB JNDI URL value of the application database.  This property is irrelevant/ignored if USE\_JNDI\_BASED\_CONNECTION=no |
| DB\_CONNECTION\_URL  DB\_USERNAME  DB\_PASSWORD | none | The application database connection properties.  A sample DB\_CONNECTION\_URL value: jdbc:oracle:thin:@cbiodb30.nci.nih.gov :1521:CBTEST  These values are purposely blank. SDK users should provide appropriate values for their database within the local.properties file located in the root folder of the SDK distribution. |
| DB\_DIALECT | org.hibernate.dialect.OracleDialect | The Hibernate Database dialect to be used when connecting to the application database.  Typical values include:   * org.hibernate.dialect.OracleDialect * org.hibernate.dialect.MySQLDialect |
| CSM\_USE\_JNDI \_BASED\_CONNECTION | no | Indicates whether a JNDI DB connection should be used for the CSM database.  If USE\_JNDI\_BASED\_CONNECTION=yes, then the DB\_JNDI\_URL property value is used to obtain the DB connection and retrieve data |
| CSM\_DB\_JNDI\_URL | java:/SDK | The DB JNDI URL value for the CSM database.  This property is irrelevant/ignored if CSM\_USE\_JNDI\_BASED\_CONNECTION=no |
| CSM\_DB\_CONNECTION\_URL  CSM\_DB\_USERNAME  CSM\_DB\_PASSWORD | none | The CSM database connection properties.  A sample DB\_CONNECTION\_URL value: jdbc:oracle:thin:@cbiodb30.nci.nih.gov :1521:CBTEST  These values are purposely blank. SDK users should provide appropriate values for their CSM database instance within the local.properties file located in the root folder of the SDK distribution. |
| CSM\_DB\_DIALECT | org.hibernate.dialect.OracleDialect | The Hibernate Database dialect used when connecting to the CSM database.  Typical values include:   * org.hibernate.dialect.OracleDialect * org.hibernate.dialect.MySQLDialect |
| VALIDATE\_LOGICAL \_MODEL | true | Used to enable/disable the validation of the logical object model prior to code generation. |
| VALIDATE\_MODEL \_MAPPING | true | Used to enable/disable the validation of the logical object model to the data model mapping prior to code generation. |
| GENERATE\_ HIBERNATE\_MAPPING | true | Used to enable/disable the generation of the Hibernate Object-Relational Mapping files during code generation. |
| GENERATE\_BEANS | true | Used to enable/disable the generation of the domain object beans (Java Beans) during code generation. |
| GENERATE\_CASTOR\_MAPPING | true | Used to enable/disable the generation of the Castor XML marshalling and unmarshalling mapping files. |
| GENERATE\_XSD | true | Used to enable/disable the generation of the XML Schemas (XSDs). |
| GENERATE\_WSDD | true | Used to enable/disable the generation of the Axis Web Service Deployment Descriptor (WSDD) file. |
| INCLUDE\_SEARCH \_EVENT\_LISTENER | false | Used to toggle whether or not Event Listeners for the Hibernate Search API will be generated in the Hibernate Configuration file (*hibernate.config.xml*). |
| CACHE\_PATH | java.io.tmpdir | An advanced property used by ehcache to store its cache files on disk. A value of ‘java.io.tmpdir’ will create the cache files within the temporary directory.  SDK users may choose to specify any absolute path instead for the cache files. |

Table SDK Configuration Properties

## Generating the SDK System

### Ant Build Script Targets

The SDK provides an Ant[[20]](#footnote-21) script, build.xml, which is located in the root folder of the SDK distribution. This script contains targets for performing various system generation tasks, including building and packaging the system.

Typically speaking, most SDK users will only need to run the following two targets:

* **build-system:** Executes the SDK Code Generator using the properties configured within the deploy.properties file. See section  for more information.
* **clean-all:** Deletes all files and folders from the previous build process. It is strongly recommended that SDK users run this target prior to running the ‘build-system’ target.

|  |  |
| --- | --- |
| NOTE: | The SDK build process is configured by the properties found within the deploy.properties file, as described in section . Please review and update these properties as needed to reflect your environment prior to generating the system. |

For those interested in the remaining targets, the table below provides a complete list:

|  |  |
| --- | --- |
| Ant Target | Description |
| build-system | Generates the SDK system using properties set within \conf\deploy.properties.  This is the primary [default] target within the build script, and the one SDK users will most typically use when generating the system. SDK users are strongly recommended to run the ‘clean-all’ target prior to running the ‘build-system’ target. |
| clean | Cleans the main generated directories and files (\output) created following the execution of the build-system target. |
| clean-all | Cleans the generated directories and files of both the main and child projects.  SDK users are strongly recommended to run the ‘clean-all’ target prior to running the ‘build-system’ target |
| codegen | Runs the SDK Code Generator. The Generator is capable of selectively generating the system components. The following properties within the deploy.properties file control the behavior of the Code Generator:   * VALIDATE\_LOGICAL\_MODEL * VALIDATE\_MODEL\_MAPPING * GENERATE\_HIBERNATE\_MAPPING * GENERATE\_BEANS * GENERATE\_CASTOR\_MAPPING * GENERATE\_XSD * GENERATE\_WSDD   See section  for more information.  This target is run as part of the process run by the ‘build-system’. SDK users should rarely, if ever need to invoke this target individually. |
| compile-beans | Compiles the generated beans.  This target is run as part of the ‘build-system’ process. SDK users should rarely, if ever need to invoke this target individually. |
| doc | Generates Javadocs for the generated beans.  This target is run as part of the ‘build-system’ process. SDK users should rarely, if ever need to invoke this target individually. |
| init | An internal target that prepares the output directory structure.  This target is run as part of the ‘build-system’ process. SDK users should rarely, if ever need to invoke this target individually. |
| package-codegen-artifacts | Packages (jars) the generated Hibernate ORM and Java bean artifacts.  This target is run as part of the ‘build-system’ process. SDK users should rarely, if ever need to invoke this target individually. |
| package-local-client | An internal target that prepares, packages the local client files.  This target is run as part of the ‘build-system’ process. SDK users should rarely, if ever need to invoke this target individually. |
| package-release-contents | Creates release binary and source packages in a zip file format for distribution.  This target is typically run by the SDK team when creating an SDK release for distribution. |
| package-remote-client | An internal target that prepares, packages the remote client files.  This target is run as part of the ‘build-system’ process. SDK users should rarely, if ever need to invoke this target individually. |
| package-system | Packages the system. Internally runs the following targets:   * package-remote-client * package-local-client * package-webapp * package-ws-client   This target is run as part of the ‘build-system’ process. SDK users should rarely, if ever need to invoke this target individually. |
| package-webapp | An internal target that prepares, packages the Web application Archive (war) file for deployment to a Servlet container such as JBoss or Tomcat.  This target is run as part of the ‘build-system’ process. SDK users should rarely, if ever need to invoke this target individually. |
| package-ws-client | An internal target that prepares, packages the Web Service client.  This target is run as part of the ‘build-system’ process. SDK users should rarely, if ever need to invoke this target individually. |
| prepare prepare-codegen prepare-condition-codegen prepare-condition-system prepare-system | An internal target that runs the following targets:   * prepare-codegen * prepare-system   This target is run as part of the ‘build-system’ process. SDK users should rarely, if ever need to invoke this target individually |
| show-properties | Dumps a list of all currently set properties. |

Table Ant Script Target Descriptions

### Selectively Generating Components

For those SDK users interested in only generating certain SDK components, the SDK Code generator is capable of selectively generating the following components:

* Hibernate O/R Mapping files
* Java Beans (domain Java objects)
* Castor XML Mapping files
* Schema (XSD) files
* Axis Web Service Deployment Descriptor (WSDD) file

To control which components are generated by the Code Generator, toggle the following respective properties within the deploy.properties file:

* GENERATE\_HIBERNATE\_MAPPING
* GENERATE\_BEANS
* GENERATE\_CASTOR\_MAPPING
* GENERATE\_XSD
* GENERATE\_WSDD

Setting the value of a given property to ‘true’ will cause the component to be generated; conversely, setting it to ‘false’ will cause the component to be ignored. See section for more information.

## Overview of Generated Packages

During the code generation process, SDK prepares four different packages. These generates packages are placed under a folder located at \output\<project\_name>\package\. Given below is the summary of the different packages created

* **local-client** – This package contains the complete application which can be used in the local environment. This package corresponds to the local-client interface of the SDK generated application. The generated binaries along with other required libraries are placed under /lib folder. /conf folder contains the configuration file required by the local client to function. /src folder contains a sample test program which can be used to test the generated local-client.
* **remote-client** - This package contains the remote client component of the generated application which can be used in the isolated environment. This package corresponds to the remote-client interface of the SDK generated application. The generated binaries along with other required libraries are placed under /lib folder. /conf folder contains the configuration file required by the local client to function in addition to the generated XSDs and castor mapping files. /src folder contains a sample test programs which can be used to test the generated remote-client. The sample programs test 1) Java API interface 2) XML marshalling and unmarshalling and 3) XML-HTTP interface’s REST capabilities.
* **ws-client** - This package contains the environment to invoke the SDK generated web services with the Java based web services client. This package corresponds to the web service interface of the SDK generated application. The generated binaries along with other required libraries are placed under /lib folder. /src folder contains a sample test program which can be used to test the generated client.
* **webapp** – This package contains a <project\_name>.war file generated by the SDK. This file represents the server component of the SDK generated system. This file has to be deployed to the application server before any of the client interfaces (except local-client) are accessed.

## Deploying the Generated System

The Ant build process packages the generated SDK system Web ARchive (war) file for ease of deployment. This file is named <project\_name>.war, and is located in the \output\<project\_name>\ package\webapp directory. Typically, this file can be copied to the web server deployment folder and the system is automatically deployed when the web server is started.

|  |  |
| --- | --- |
| NOTE: | The generated SDK system has been tested on both JBoss v4.0.5 and Apache Tomcat v5.5.20 servers. The system should also work on other servers such as Weblogic or WebSphere; however, no guarantees are made. |

### Deploying to JBoss Server

If the generated system war file is to be deployed to a JBoss server instance, the SERVER\_TYPE property found in the \conf\deploy.properties file should be set to ‘jboss’. This will ensure that the log4j.jar file is excluded from the packaged war file during the build process. This is required as JBoss already has its own copy of the log4j.jar file, and will report an error if it finds another copy of this file within the war.

To deploy to a JBoss server instance, copy the generated war file to <JBoss installation directory>\ server\default\deploy directory, and then restart the server.

### Deploying to Apache Tomcat Server

If the generated system war file is to be deployed to a JBoss server instance, the SERVER\_TYPE property found in the \conf\deploy.properties file should be set to ‘jboss’. This will ensure that the log4j.jar file is excluded from the packaged war file. This is required as JBoss already has its own copy of the log4j.jar file, and will report an error if it finds another copy of this file within the war.

To deploy to a Tomcat server instance, copy the generated war file to <Tomcat Installation Directory>\ webapps directory, and then restart the server.

|  |  |
| --- | --- |
| NOTE: | When re-deploying the system war file to Tomcat after an initial build, it is strongly recommended that the old war file and corresponding exploded directory be deleted before the new war file is copied to the deployment directory. This will ensure that all files from the previous deployment are properly deleted. |

## Testing the caCORE SDK Generated System

The following sub-sections discuss various tests for determining whether or not the SDK system has been successfully generated and deployed.

### Testing the Web Interface

The SDK generated GUI consists of several web pages that facilitate access to domain data. The Home page can be accessed via the following URL pattern:

|  |  |
| --- | --- |
| **SDK Web Interface Test URL Pattern** | http://<server\_name>:<server\_port>/<project\_name> |

Thus, for the Home page of the sample SDK model, the URL might be: <http://localhost:8080/example>. If the system has been successfully deployed, the following page should be displayed as shown in the diagram below:

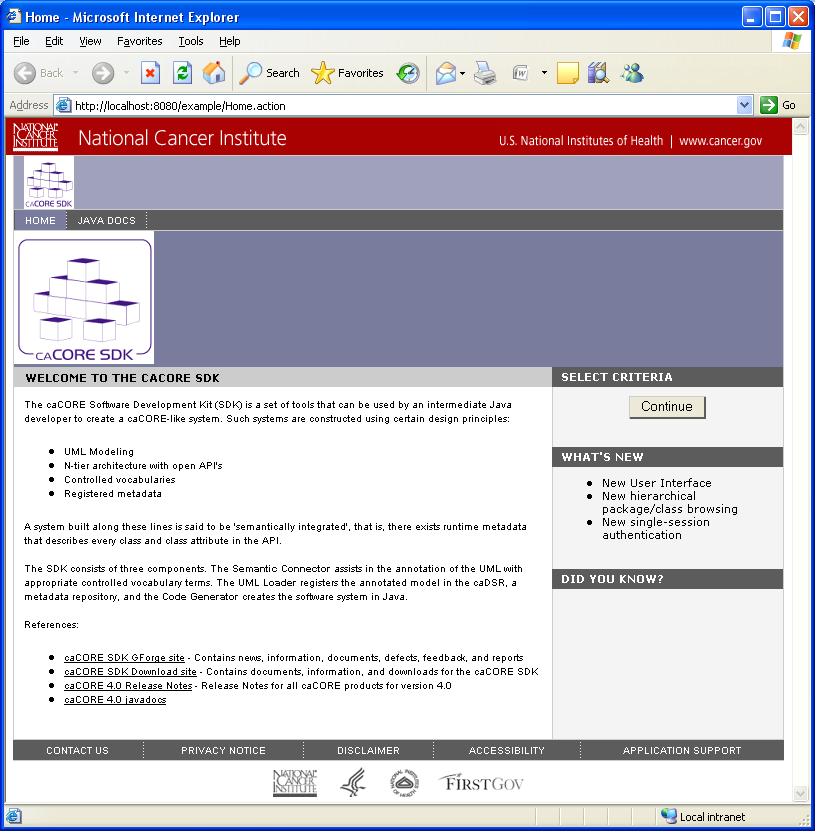


Figure 48: Web Interface Test Page

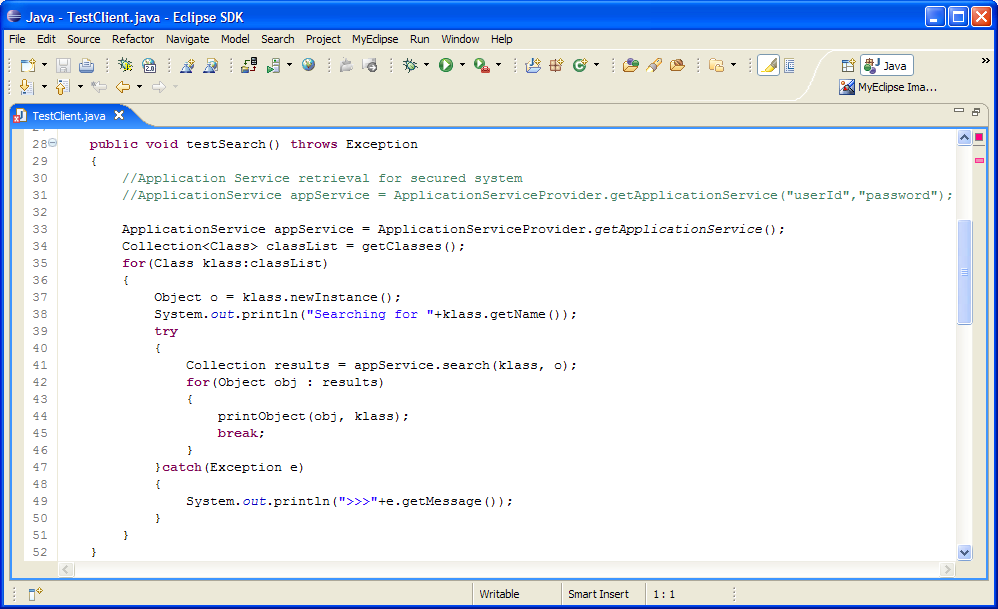
See section for more information on how to use the Web Interface.

### Testing the Java API

A program, TestClient.java, is provided with the SDK distribution for testing the Java API. This program is located within the \output\<project\_name>\package\remote-client\src\ folder. To execute the program, run the default run target of the Ant script, build.xml, located within the \output\ <project\_name>\package\remote-client\ folder.

|  |  |
| --- | --- |
| NOTE: | The generated system must be deployed to the server, and the server must be running before the test is invoked. |

The following diagram shows the main test method algorithm:

 Figure 37: Java API Test Algorithm

As shown, the program systematically loops through all the generated Java Bean classes and searches for each one without any filtering. It then takes the first qualifying record returned from the search and prints out its details to stdout, thus testing whether or not the Java API is working.

|  |  |
| --- | --- |
| NOTE: | The *TestClient.java* program is simply that, a client for testing the Java API. It provides only one example of how the SDK Application Service search API may be invoked. If desired, it can be modified to use a different method within the Application Service API, or the return results filtered by adding criteria data to the search object prior to the search. |

See section  for more information on how to use the Java API Interface.

### Testing the XML Utility

A program, TestXMLClient.java, is provided with the SDK distribution for testing the generated Castor XML Mapping and Schema (XSD) files. This program is located within the \output\<project\_name>\ package\remote-client\src\ folder. To execute the program, run the runXML target of the Ant script, build.xml, located within the \output\<project\_name>\package\remote-client\ folder.

|  |  |
| --- | --- |
| NOTE: | The generated system must be deployed to the server, and the server must be running before the test is invoked. |

The following diagram shows a portion of the main test method:

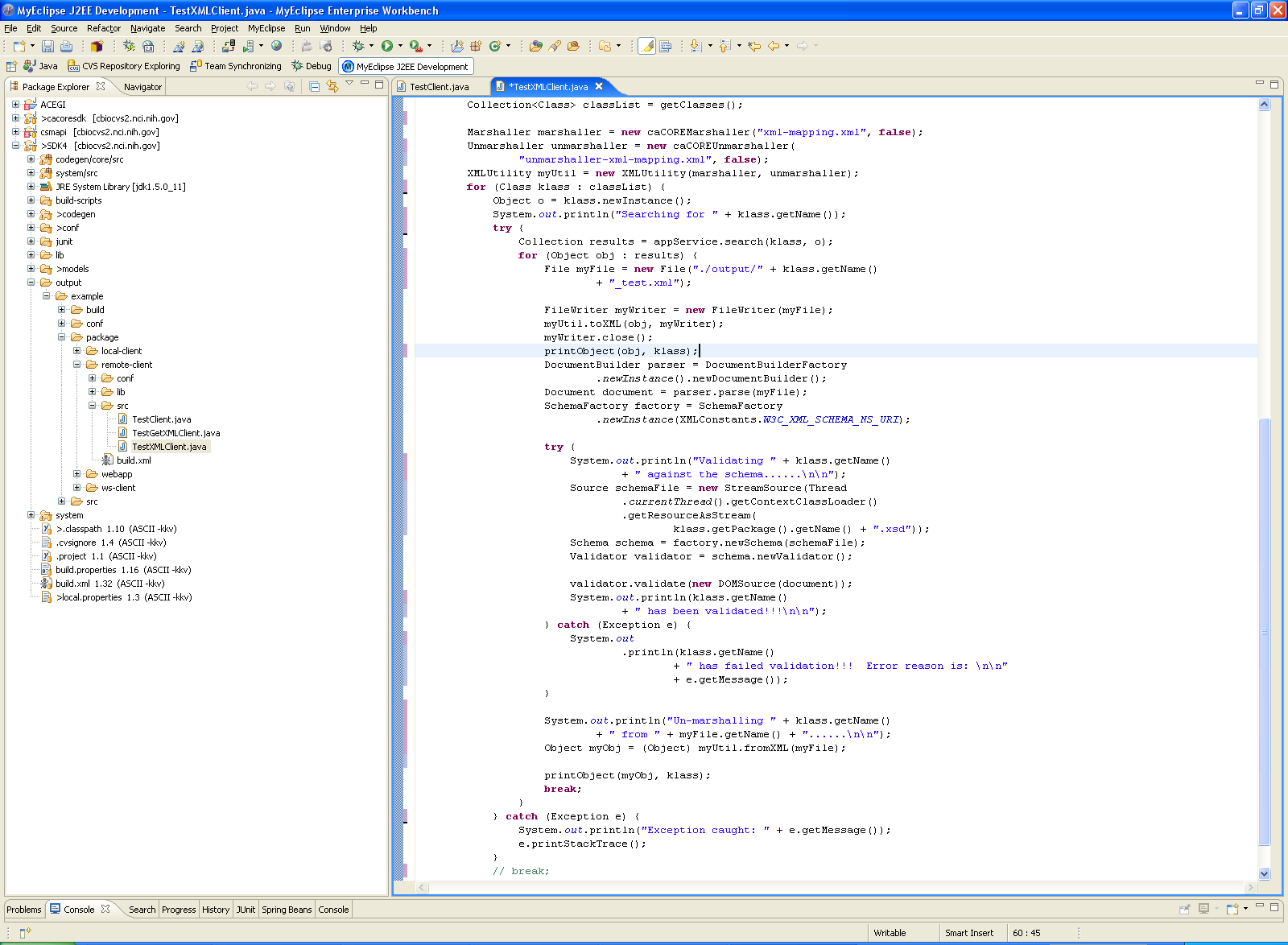


Figure 37: XML Mapping and Schema Test Algorithm

As shown, the program systematically loops through all the generated Java Bean classes and searches for each one without any filtering. It then takes the first qualifying record returned from the search, and marshalls (serializes) it to a file. Next, it reads the XML file back in, parses the XML, and validates it against the generated Schema. Finally, it unmarshalls (deserializes) the XML back to the corresponding domain Java Bean object, thus testing that the generated XML Mapping and Schema files are working properly.

|  |  |
| --- | --- |
| NOTE: | The *TestXMLClient.java* program is simply that, a client for testing the XML Utility. It provides only one example of how the XML Utility marshalling/unmarshalling methods may be invoked. It can be modified to use a different method if so desired.  Also, the same search algorithm used during the testing of the Java API’s is re-used here. See section for more details.  Finally, by its very nature, XML processing can by memory intensive. The TestXMLClient.java program has been successfully run against the sample SDK model, which does not contain a lot of data. When running the test program against a model with a lot of data, the memory specified by the *maxmemory=512m* attribute within the *runXML* target may need to be increased. |

### 

### Testing the Web Service Interface

### Testing the Web Service URL

A successful Web Service deployment can be tested by entering in a browser the Web Service URL that conforms to the following pattern:

|  |  |
| --- | --- |
| **SDK Web Service Test URL Pattern** | http://<server\_name>:<server\_port>/<project\_name>/services/<project\_name>Service |

Thus, a successful Web Service deployment URL for the sample SDK model might be: <http://localhost:8080/example/services/exampleService>.

The following diagram illustrates the result of a successful Web Service deployment test:

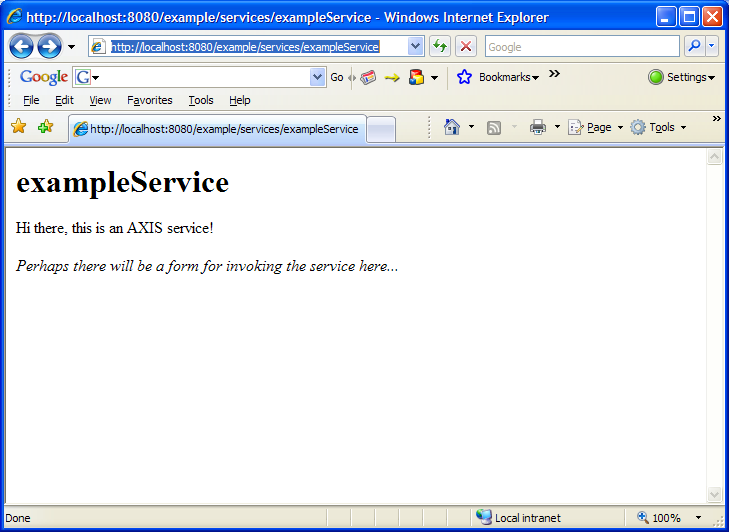


Figure 51 Web Service Test Page

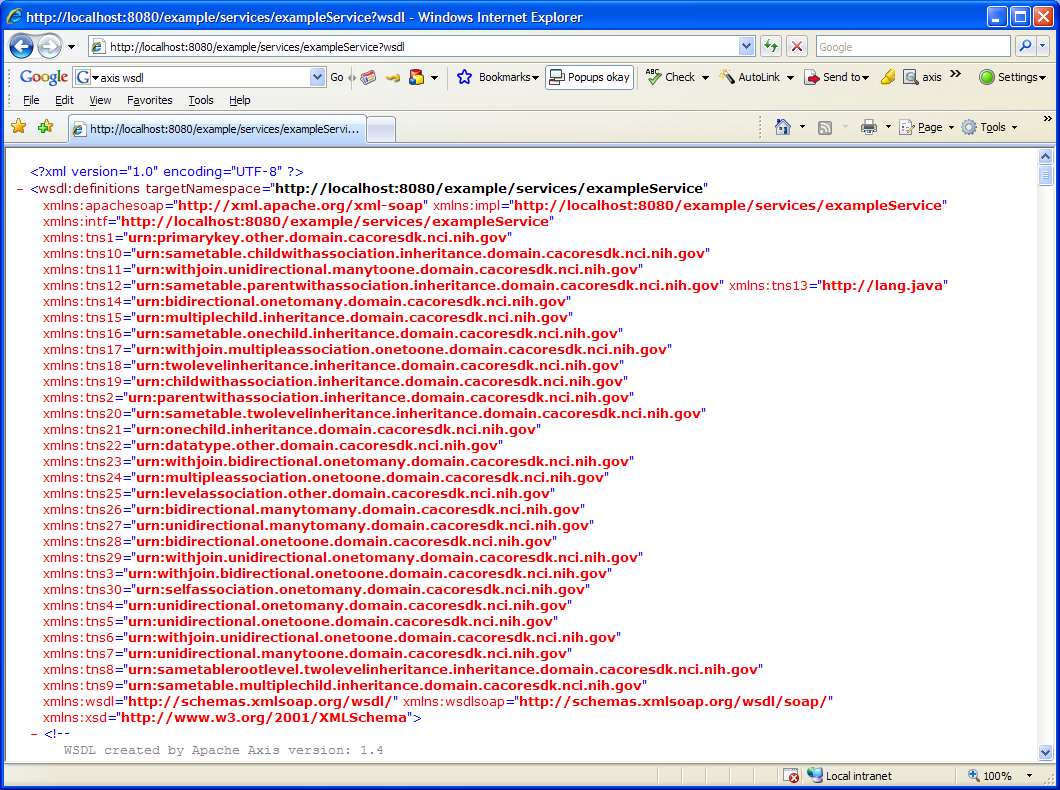
|  |  |
| --- | --- |
| NOTE: | The SDK Web Service Deployment Descriptor (WSDD) is now packaged along with the rest of the SDK generated system, thus allowing for automatic deployment of the SDK Web Service whenever the system is deployed; i.e., manual deployment of the Web Service is no longer required. |

### Obtaining the WSDL for Deployed Services: ?WSDL

As shown in the previous section, entering the Web Service URL in a browser will cause a message to be displayed indicating that the endpoint is an Axis service. However, if the suffix ‘*?wsdl’* is added to the end of the URL, Axis will automatically generate a WSDL service description for the deployed service, and return it as XML in the browser . The URL pattern is shown below:

|  |  |
| --- | --- |
| **SDK Web Service WSDL Pattern** | http://<server\_name>:<server\_port>/<project\_name>/services/<project\_name>Service?wsdl |

The following diagram illustrates a portion of the resulting XML that is generated after invoking the WSDL URL for the SDK sample Web Service:

Figure 52 Obtaining the WSDL for Deployed Services: ?WSDL

### Testing Web Services via the Client Program

The SDK distribution also provides a client program, TestClient.java, for testing the Web Service Interface. This program is located within the \output\<project\_name>\package\ws-client\src\ folder. To execute the program, run the default run target of the Ant script, build.xml, located within the \output\<project\_name>\package\ws-client\ folder.

|  |  |
| --- | --- |
| NOTE: | The generated system must be deployed to the server and the server must be running before the Web Service test is invoked. |

The following diagram shows a portion of the main test method:

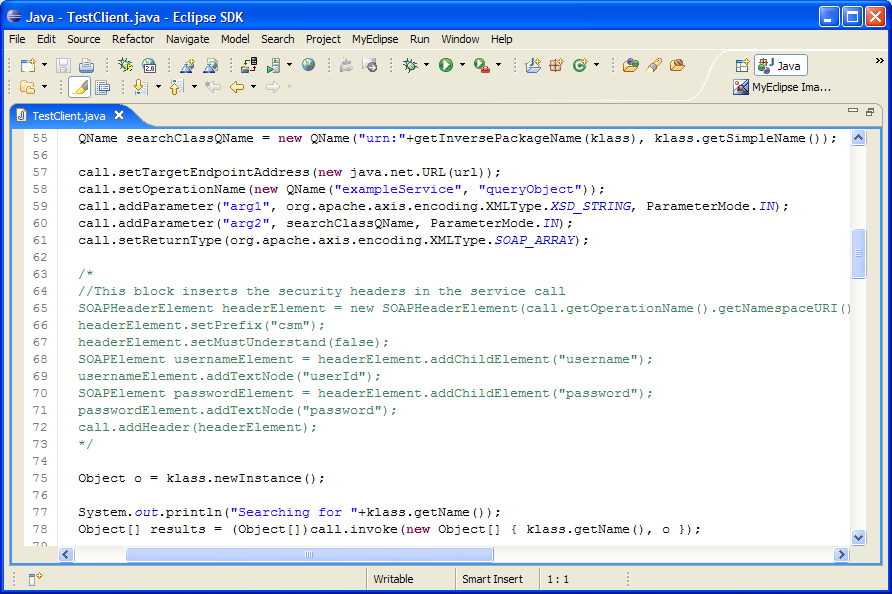


Figure 53 Web Service Test Algorithm

The Web Service test program systematically loops through all the generated Java Bean classes and creates a Web Service queryObject call for each one. It then takes the first qualifying record returned from the call, and checks to see if the returned object has an association to another domain object. If it does, the program then proceeds to create and invoke a Web Service getAssociation call for it, thus testing a couple of the Web Service operations defined within the WSDL.

|  |  |
| --- | --- |
| NOTE: | The Web Service *TestClient.java* program is simply that, a client for testing the generated Web Service. It provides only one example of how the SDK Web Service messages may be created and invoked. It can be modified to use a different operation or algorithm if so desired.  Also, the same search algorithm used during the testing of the Java API’s is re-used within the test program. See section for more details. |

See section  for more information on how to use the Web Service Interface.

## Configuring Security

Security in SDK is provided by ACEGI and CSM. Users of the SDK do not need any specific configuration within SDK to use security. Users of the SDK who intends to enable or disable security in the SDK can do so by altering the configuration parameters in the deploy.properties file before generating the system. Table given below shows the properties that the user has to modify in order to correctly enable the security in SDK.

|  |  |  |
| --- | --- | --- |
| Property Name | Default Value | Description |
| SECURITY\_ENABLED | false | Used to enable or disable security within the generated system during code generation.  This applies to all of the SDK interfaces, including:   * Web Interface (GUI) * Java API Interface (local and remote clients) * Web Service Interface |
| CSM\_PROJECT\_NAME | sdk | Used as a prefix when creating the CSM security configuration file name. CSM configuration should have the same application name configured |
| INSTANCE\_LEVEL \_SECURITY | false | Used to enable/disable CSM instance level security.  Only relevant if the SECURITY\_ENABLED property is set to ‘true’ |
| ATTRIBUTE\_LEVEL \_SECURITY | false | Used to enable/disable attribute level security.  Only relevant if the SECURITY\_ENABLED property is set to ‘true’ |
| CSM\_USE\_JNDI \_BASED\_CONNECTION | no | Indicates whether a JNDI DB connection should be used for the CSM database.  If USE\_JNDI\_BASED\_CONNECTION=yes, then the DB\_JNDI\_URL property value is used to obtain the DB connection and retrieve data |
| CSM\_DB\_JNDI\_URL | java:/SDK | The DB JNDI URL value for the CSM database.  This property is irrelevant/ignored if CSM\_USE\_JNDI\_BASED\_CONNECTION=no |
| CSM\_DB\_CONNECTION\_URL  CSM\_DB\_USERNAME  CSM\_DB\_PASSWORD | none | The CSM database connection properties.  A sample DB\_CONNECTION\_URL value: jdbc:oracle:thin:@cbiodb30.nci.nih.gov :1521:CBTEST  These values are purposely blank. SDK users should provide appropriate values for their CSM database instance within the local.properties file located in the root folder of the SDK distribution. |
| CSM\_DB\_DIALECT | org.hibernate.dialect.OracleDialect | The Hibernate Database dialect used when connecting to the CSM database.  Typical values include:   * org.hibernate.dialect.OracleDialect * org.hibernate.dialect.MySQLDialect |

Login-config

### CSM Integration

#### CSM for authentication

#### Protection Element for authorization

# 

# FAQs

1. ***I tried to use SDK but during code generation, I am getting just the exceptions and not error messages***

Getting just the exceptions indicates that the SDK code generator did not initialize due to either invalid settings in the deploy.properties or invalid UML model file. UML model file can be considered invalid if it is not developed per specification of SDK or it is not exported as specified by SDK.

1. ***I tried to generate an application with SDK but I received validation errors. How do I make sure that model that I have created runs through the code generator***

The validation error messages generated from the SDK indicates specific error conditions under which SDK cannot generate the code. Fixing the UML model and executing the code generator will solve the problem

1. ***When running the generated application (.war file) under JBoss I am getting Log4J exception***

SDK by default includes the log4j.jar and commons-logging.jar file in the generated .war file’s lib directory. JBoss server requires both of these files to be excluded from the .war file before deployment. Developer using the SDK can either remove these two jar files from the .war file before deployment or they can specify SERVER\_TYPE=jboss under deploy.properties file and regenerate the system. Specifying server type as jboss during code generation will exclude the unnecessary jar files from being packaged in the .war file

1. ***I successfully generated the application with SDK. When running the application, I am getting database connection errors***

While generating the application with SDK, user has to specify the database connection parameters in the deploy.properties file. If these settings are incorrect, SDK cannot fetch the data from the database. Make sure that the database settings are valid and database server is running

1. ***When I try to query the generated system, queries for some of the objects are running very slow.***

There can be many different problems associated with the slow searches. Primary problem is with the missing indexes on the primary key filed, foreign key field or search key field. Creating these indexes should stop database from performing full table scans and improve performance. Refer to section<<?>> of the documentation to get details on how to improve the performance of the system.

# Roadmap

SDK development team constantly strives to improve the experience of using SDK by providing new features and enhancing existing features. During the course of the development for the current release, SDK team has come across many new features that will be considered for development immediately following the release of the current version. Below is the short summary of some of the major features under consideration

**Grid Integration –** Latest release of SDK is not compatible with caGrid 1.0 or 1.1. SDK team will be working with the caGrid team to integrate the latest release of SDK with future version of caGrid.

**GUI for installation and build process** – Current SDK build process involves executing the ANT scripts to generate code with the SDK code generation module and preparing the packages for the deployment and release. Although this process is geared towards novice users, many of the users find it difficult to use the command line scripts execution. A new tool is under consideration for development which will allow users to control the execution of code generation process from the graphical interface.

**Robust user interface** – Current user interface for the web application is a major improvement over the user interface provided by the previous release. The current version of the interface is based on the NCICB UI templates and it has better integration of security then previous version. This user interface will be expanded to provide additional features like

* Complex Query By Example (QBE) input forms
* In line documentation for the UML class and attributes in the domain class browser
* Displaying UML diagrams in the domain class browser
* Allow to edit the records

**Writable API –** Current version of SDK provides read only API for the domain model. Previous release of SDK did include primitive version of writable API but it is not being released with the new SDK. SDK team is planning to develop a complete working solution for the writable API which is applicable to the most users’ requirements.

# Appendix A Example Model and Mapping

# Appendix B Glossary

This section contains a list of terms used in this document, with accompanying definitions.

| Term | Definition |
| --- | --- |
| Acegi | Acegi is a security framework that provides a powerful, flexible security solution for enterprise software, with a particular emphasis on applications that use the Spring Framework. Acegi Security provides the SDK with comprehensive authentication, authorization, instance-based access control, channel security and human user detection capabilities. See <http://www.acegisecurity.org/> for more information. |
| Ant | Apache Ant is a Java-based build tool used within the SDK to perform various build related tasks. See section  for more information on how Ant is used within the SDK. See <http://ant.apache.org/> for more information on Ant itself. |
| Castor | Castor is an Open Source data binding framework for Java, and facilitates conversion between Java Beans, XML documents and relational tables. Castor provides Java-to-XML binding, Java-to-SQL persistence, and more. See <http://www.castor.org/> for more information. |
| Ehcache | Ehcache is a simple, fast and thread safe cache for Java that provides memory and disk stores and distributed operation for clusters. The SDK uses ehcache in conjunction with Hibernate. See <http://sourceforge.net/projects/ehcache> for more information. |
| QBE | Query by Example (QBE) is a database query language for relational databases. It was devised by Moshé M. Zloof at IBM Research during the mid 1970s, in parallel to the development of SQL. It is the first graphical query language, using visual tables where the user would enter commands, example elements and conditions. See <http://en.wikipedia.org/wiki/Query_by_Example> for more information. |
| Hibernate | Hibernate is an object-relational mapping (ORM) solution for the Java language, and provides an easy to use framework for mapping an object-oriented domain model to a traditional relational database. Its purpose is to relieve the developer from a significant amount of relational data persistence-related programming tasks. See <http://www.hibernate.org/> for more information. |
| HQL | Hibernate Query Language (HQL) is a powerful query language that looks a lot likie SQL. But though the syntax is SQL-like, HQL is fully object-oriented, and understands concepts like inheritance, polymorphism and association. See <http://www.hibernate.org/hib_docs/v3/reference/en/html/queryhql.html> for more information. |
| Marshaling | The process of producing an XML document from Java Beans; i.e., the process of serializing Java Beans to XML. |
| ORM | An acronym for Object-Relational Mapping, a programming technique for converting data between incompatible type systems in databases and Object-oriented programming languages. This creates, in effect, a "virtual object database" which can be used from within the programming language. See <http://en.wikipedia.org/wiki/Object-relational_mapping> for more information. Hibernate implements this technique within the SDK. |
| REST | “Representational State Transfer (REST) is a style of software architecture for distributed hypermedia systems such as the World Wide Web. The term was introduced in the doctoral dissertation of Roy Fielding in 2000,[1] one of the principal authors of the Hypertext Transfer Protocol (HTTP) specification, and has come into widespread use in the networking community.  “REST strictly refers to a collection of network architecture principles that outline how resources are defined and addressed. The term is often used in a looser sense to describe any simple interface that transmits domain-specific data over HTTP without an additional messaging layer such as SOAP or session tracking via HTTP cookies. These two meanings can conflict as well as overlap. It is possible to design any large software system in accordance with Fielding's REST architectural style without using the HTTP protocol and without interacting with the world wide web. It is also possible to design simple XML+HTTP interfaces that do not conform to REST principles, and instead follow a Remote Procedure Call model. The two different uses of the term "REST" cause some confusion in technical discussions.” See <http://en.wikipedia.org/wiki/REST> for more information. |
| Unmarshaling | The process of populating a generated class object from a corresponding XML document; i.e., the process of deserializing XML to Java Beans. |
| WSDD | An acronym for Web Service Deployment Descriptor, which can be used to specify resources that should be exposed as Web Services. See <http://ws.apache.org/axis/java/user-guide.html#CustomDeploymentIntroducingWSDD> for more information. |
| WSDL | An acronym for Web Services Definition Language, which is an XML-based language that provides a model for describing Web services. See <http://www.w3.org/TR/wsdl.html> or <http://en.wikipedia.org/wiki/WSDL> for more information. |

Table Glossary

1. Model Driven Architecture (MDA) is a software development practice that uses a structured modeling language to describe the requirements, objects, and interactions of a data system prior to its construction. [↑](#footnote-ref-2)
2. <http://www.hibernate.org> [↑](#footnote-ref-3)
3. <http://www.SpringFramework.org> [↑](#footnote-ref-4)
4. <http://www.acegisecurity.org/> [↑](#footnote-ref-5)
5. <http://java.sun.com/j2se/1.4.2/docs/api/java/lang/ThreadLocal.html> [↑](#footnote-ref-6)
6. See <http://en.wikipedia.org/wiki/REST> for more information on REST [↑](#footnote-ref-7)
7. See <http://www.hibernate.org/hib_docs/v3/reference/en/html/queryhql.html> for more information on the Hibernate Query Language. [↑](#footnote-ref-8)
8. See <http://www.hibernate.org/hib_docs/v3/reference/en/html_single/#querycriteria> for more information on Hibernate Criteria queries. See section *15.8. Detached Queries and Subqueries* of the same chapter for details on the Hibernate DetachedCriteria itself. [↑](#footnote-ref-9)
9. Query by Example (QBE) is a database query language for relational databases. It was devised by Moshé M. Zloof at IBM Research during the mid 1970s, in parallel to the development of SQL. It is the first graphical query language, using visual tables where the user would enter commands, example elements and conditions. See <http://en.wikipedia.org/wiki/Query_by_Example> for more information. [↑](#footnote-ref-10)
10. See the Introduction to Web Services Metadata: <http://dev2dev.bea.com/pub/a/2004/10/Anil_WServices.html>. [↑](#footnote-ref-11)
11. There are four "styles" of service in Axis. RPC services use the SOAP RPC conventions, and also the SOAP "section 5" encoding. Document services do not use any encoding (so in particular, you won't see multiref object serialization or SOAP-style arrays on the wire) but DO still do XML<->Java databinding. Wrapped services are just like document services, except that rather than binding the entire SOAP body into one big structure, they "unwrap" it into individual parameters. Message services receive and return arbitrary XML in the SOAP Envelope without any type mapping / data binding. For more information, please refer to <http://ws.apache.org/axis/java/user-guide.html#ServiceStylesRPCDocumentWrappedAndMessage>. [↑](#footnote-ref-12)
12. For more information related to the WSDL format and structure, please refer to <http://www.w3.org/TR/wsdl.html> or <http://en.wikipedia.org/wiki/WSDL>. [↑](#footnote-ref-13)
13. More information related to the SOAP Fault can be found in the Simple Object Access Protocol (SOAP) 1.1 Specification: <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>. [↑](#footnote-ref-14)
14. <http://www.ietf.org/rfc/rfc2617.txt> [↑](#footnote-ref-15)
15. <http://en.wikipedia.org/wiki/Basic_access_authentication> [↑](#footnote-ref-16)
16. <http://www.hibernate.org/hib_docs/reference/en/html/querycriteria.html#querycriteria-detachedqueries> [↑](#footnote-ref-17)
17. <http://www.hibernate.org/hib_docs/reference/en/html/queryhql.html> [↑](#footnote-ref-18)
18. Castor is an Open Source data binding framework for Java, and facilitates conversion between Java Beans, XML documents and relational tables. Castor provides Java-to-XML binding, Java-to-SQL persistence, and more. See <http://www.castor.org/> for more information. [↑](#footnote-ref-19)
19. Castor is an Open Source data binding framework for Java, and facilitates conversion between Java Beans, XML documents and relational tables. Castor provides Java-to-XML binding, Java-to-SQL persistence, and more. See <http://www.castor.org/> for more information. [↑](#footnote-ref-20)
20. Apache Ant is a Java-based build tool used within the SDK to perform various build related tasks. See http://ant.apache.org/ for more information. [↑](#footnote-ref-21)