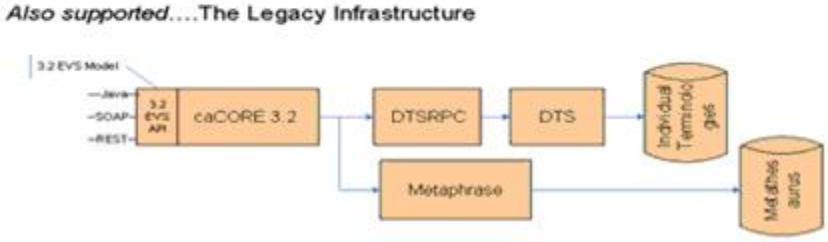
Interacting with caCORE LexEVS 3.x and 4.x

This section describes the components of the caCORE LexEVS and the service interface layer provided by the EVS API architecture. It gives examples of how to use the EVS APIs. It also describes the Distributed LexBIG API and the Distributed LexBIG Adapter.

Contents of this Page

caCORE LexEVS Components

The caCORE LexEVS API is a public domain, open source wrapper that provides full access to the LexBIG Terminology Server. LexBIG hosts the NCI Thesaurus, the NCI Metathesaurus, and several other vocabularies. Java clients accessing the NCI Thesaurus and Metathesaurus vocabularies communicate their requests via the open source caCORE LexEVS APIs, as shown in Overview of the caCORE LexEVS 4.0 release components.



The open source interfaces provided as part of caCORE LexEVS 5.x include Java APIs, a SOAP interface, and an HTTP REST interface. The Java APIs are based on the EVS 3.2 object model and the LexBIG Service object model.

The EVS 3.2 model, exposed as part of caCORE 3.2, has been re-released with LexBIG as the back-end terminology service in place of the proprietary Apelon DTS back end. The SOAP and HTTP REST interfaces are also based on the 3.2 object model. The SDK 4.0 was used to generate the EVS 3.2 Java API, as well as the SOAP and HTTP REST interfaces.

The only difference between the EVS 3.2 API exposed as part of the caCORE LexEVS 5.x and the API exposed as part of caCORE 3.2 is the back-end terminology server used to retrieve the vocabulary data. The interface (API calls) are the same and should only require minor adjustments to user applications.

> **Note**
>
> You cannot integrate caCORE 3.2 components with caCORE LexEVS 5.x. If you used multiple components of caCORE 3.2 (for example, EVS with caDSR), you need to continue to work with the caCORE 3.2 release until the other caCORE 4.0 components are available.

The LexBIG object model was developed by the Mayo Clinic. In its native form, the associated API assumes a local, non-distributed means of access. With caCORE LexEVS 5.x, a proxy layer enables EVS API clients to access the native LexBIG API from anywhere without having to worry about the underlying data sources. This is called the Distributed LexBIG (DLB) API.

The DLB Adapter is another option for caCORE LexEVS 5.x clients who choose to interface directly with the LexBIG API. This is essentially a set of convenience methods intended to simplify the use of the LexBIG API. For example, a series of method calls against the DLB API might equate to a single method call to the DLB Adapter.

> **Note**
>
> The DLB Adapter is not intended to represent a complete set of convenience methods. As part of the caCORE LexEVS 5.x release, the intention is that users will work with the DLB API and suggest useful methods of convenience to the EVS Development Team.

LexEVS Data Sources

The LexEVS data source is the open source LexBIG terminology server. EVS clients interface with the LexBIG API to retrieve desired vocabulary data. The EVS provides the NCI with services and resources for controlled biomedical vocabularies, including the NCI Thesaurus and the NCI Metathesaurus.

NCI Thesaurus

The NCI Thesaurus is composed of over 27,000 concepts represented by about 78,000 terms. The Thesaurus is organized into 18 hierarchical trees covering areas such as Neoplasms, Drugs, Anatomy, Genes, Proteins, and Techniques. These terms are deployed by the NCI in its automated systems for uses such as key wording and database coding.

NCI Metathesaurus

The NCI Metathesaurus maps terms from one standard vocabulary to another, facilitating collaboration, data sharing, and data pooling for clinical trials and scientific databases. The Metathesaurus is based on the Unified Medical Language System (UMLS) developed by the National Library of Medicine (NLM). It is composed of over 70 biomedical vocabularies.

Installing and Configuring the LexEVS 5.0 Java API

The LexEVS 5.0 Java API bundled with the caCORE LexEVS 5.x release provides direct access to domain objects and all service methods. Because caCORE LexEVS is natively built in Java, this API provides the fullest set of features and capabilities.

The EVS API home page provides a user interface (UI) to the Java API. The interface is available at http://lexevsapi.nci.nih.gov/lexevsapi50.

> **Note**
>
> The caCORE 3.2 release also provides an EVS 3.2 Java client API. The difference between the 3.2 and the 4.x clients is the back-end terminology server. caCORE 3.2 uses the proprietary Apelon DTS and caCORE LexEVS 4.x uses LexBIG. The API is the same and should only require minor updates to a client application wanting to migrate to the EVS 3.2 Java API provided with caCORE LexEVS 5.x.

Software Requirements

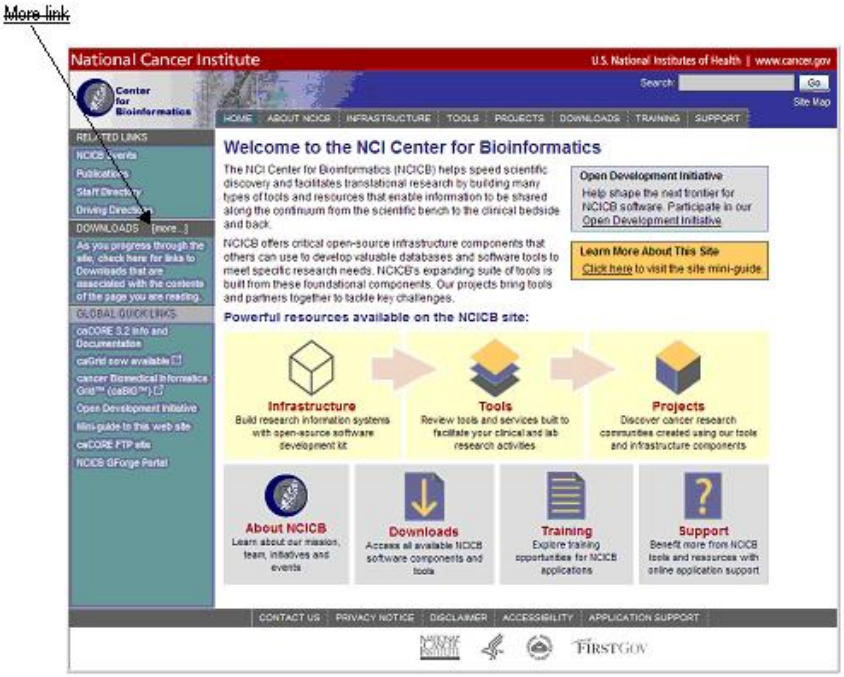
The caCORE LexEVS Java 3.2 API uses the following software on the client machine (caCORE LexEVS Java API client software).

Software	Version	Required?
Java 2 Platform Standard Edition Software 5.0 Development Kit (JDK 5.0)	1.5.0 or higher	Yes
Apache Ant	1.6.5 or higher	Yes

Downloading the Java API Client Package

The client package is available on the NCICB Web site. To download it, follow these steps:

1. Using your browser, go to http://ncicb.nci.nih.gov.
2. On the left navigation bar of the NCICB welcome page, click the more link to the right of the DOWNLOADS category (Downloads section of the NCICB Web site).



3. When prompted, enter your name, e-mail address, and institution name.
4. Click Enter the **Download Area**.
5. Read and accept the license agreement.
6. On the caCORE LexEVS downloads page, download the EVS Zip file from the Primary Distribution section.

Installing the Package

After downloading the caCORE LexEVS client package, extract the contents of the downloadable archive to a directory on your hard drive (for example, `c:\evsapi` on Windows or `/usr/local/evsapi` on Linux).

The extraction includes the directories and files listed below. This table shows the extracted directories and files in caCORE LexEVS client package.

Directory	Files	Description	Component
`./build`	`build.xml`	Ant build file	Build file
`./conf`	`application-config-client.xml`	–	–
	`xml-mapping.xml`	–	–
	`log4j.properties.xml`	Logging utilities configuration properties	–
	`*.xsd`	–	–
`./lib`	`spring.jar`	Spring framework	HTTP remoting
	`acegi-security-1.0.4.jar`	Spring Security	–
	`asm.jar`	–	–
	`antlr-2.7.6.jar`	Apache Ant	–
	`log4j-1.2.14.jar`	Logging utilities	Logging
	`commons-*.jar`	Various Apache Commons utilities	Utilities
	`lexevsapi50-beans.jar`	Holder for Generated Beans (other than the Castor-generated LexBIG/LexGrid beans). Currently empty.	–
	`lexevsapi50-framework.jar`	caCORE LexEVS framework	–
	`lexbig.jar`	LexBIG classes. Contains the Castor-generated LexBIG/LexGrid domain Objects.	–
	`lucene*.jar`	Index search	LexBIG
	`castor-1.0.2.jar`	Castor serializer/deserializer	XML conversion
	`xercesImpl.jar`	Apache Xerces XML parser	–
	`sdk*.jar`	Base caCORE SDK functionality	–

The following files are required to use the Java API:

- all of the files in the `conf` directory, and
- all of the `jar` files in the `lib` directory of the caCORE LexEVS client package.

When building applications, include these files in the Java classpath. The included `build.xml` file demonstrates how to do this when using Ant for command-line builds. If you are using an integrated development environment (IDE) such as Eclipse, refer to the tool's documentation for information on how to set the classpath.

Search Paradigm

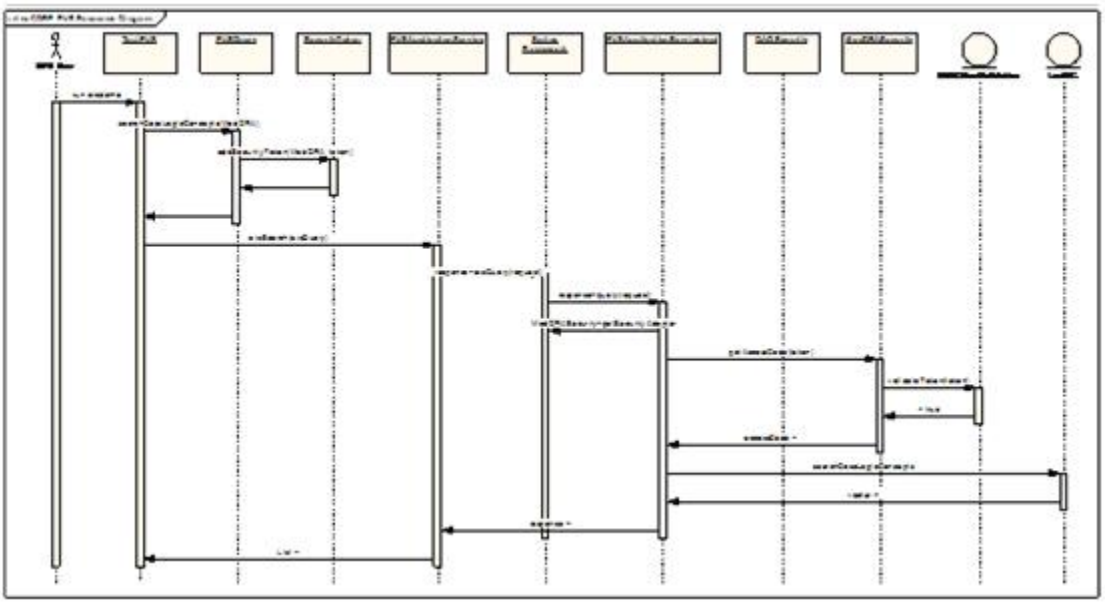
The caCORE LexEVS architecture includes a service layer that provides a single, common access paradigm to clients that use any of the provided interfaces. As an object-oriented middleware layer designed for flexible data access, caCORE LexEVS relies heavily on strongly typed objects and an *objec t-in/object-out* mechanism.

Accessing and using a caCORE LexEVS system requires the following steps:

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

caCORE LexEVS systems use four native application programming interfaces (APIs). Each interface uses the same paradigm to provide access to the caCORE LexEVS domain model, with minor changes specific to the syntax and structure of the clients. The following sections describe each API, identify installation and configuration requirements, and provide code examples.

The sequence diagram that follows shows the caCORE 4.0 LexEVS API search mechanism implemented to access the NCI EVS vocabularies.



Querying the System

LexEVS conforms to the caCORE SDK API - for more information see : **(Link to caCORE SDK doc)**

QueryOptions

QueryOptions (link to javadoc) are designed to give the user extra control over the query before it is sent to the system. QueryOptions may be used to modify a query in these ways:

- 'CodingScheme' - Restricts the query to the specified Coding Scheme, instead of querying every available Coding Scheme.
- CodingSchemeVersionOrTag' - Restricts the query to the specified Version of the Coding Scheme. Note that:
  - This may NOT be specified without also specifying the 'CodingScheme' attribute.
  - If left unset, it will default to the version of the Coding Scheme tagged as "PRODUCTION" in the system.
- 'SecurityTokens' - Security Tokens to use with the specified query. These Security Tokens are scoped to the current query ONLY. An subsequent queries will also need to specify the necessary Query Options.

- 'LazyLoad' - Some high use-case model Objects have bee 'lazy-load' enabled. This means that some attributes and associations of a model Object may not be fully populated when returned to the user. This allows for faster query times. This defaults to **false**, meaning that all attributes and associations will be eagerly fetched by the server and model Objects will always be fully populated. To enable this on applicable Objects, set to **true**.

  > **Note**
  >
  > Lazy Loading may only be used in conjunction with specifying a Coding Scheme and Version with the 'CodingScheme' and 'CodingSchemeVersionOrTag' attributes above.

- 'ResultPageSize' - the page size of results to return. The higher the number, the more results the system will return to the user at once. The client will request the next group of query results transparenly. This parameter is useful for performance tuning. For example, if a query returns a result of10,000 Objects, a 'ResultPageSize' of '1000' would make 10 calls to the server returning a page of 1000 results each time. If left unset, this value will default to the default set Page Size **(LINK CACORE SDK GUIDE page 65)**

Examples of Use

Example 4.1: Query By Example with No Query Options

```
1  public static void main(String[] args)
2  {
3      try {
4          LexEVSApplicationService appService =
5            (LexEVSApplicationService)ApplicationServiceProvider.
6             getApplicationService("EvsServiceInfo");
7          Entity entity = new Entity()
8          entity.setEntityCode("C1234");
9          List&lt;Entity&gt; list = appService.search(Entity.class, entity);
10     } catch(ApplicationException ex){
11     }
12  }
```

The following table explains specific statements in the code by line number.

Line Number	Explanation
4	Creates an instance of a class that implements the LexEVSApplicationService interface. This interface defines the service methods used to access data objects.
7	Construct the Query By Example Object and populate it with the desired search critieria. For this example, seach for any 'Entity' with an 'entityCode' attribute equaling 'C1234'.
9	Calls the search method of the LexEVSApplicationService object. This method returns a List Collection. This list will contain all of the 'Entity' Objects that match the search critieria. It this case, it will return all 'Entity' Objects with an 'entityCode' of "C1234".

Example 4.2: Query By Example with Query Options

```
1  public static void main(String[] args)
2  {
3     try {
4        LexEVSApplicationService appService =
5           (LexEVSApplicationService)ApplicationServiceProvider.
6           getApplicationService("EvsServiceInfo");
7           QueryOptions queryOptions = new QueryOptions();
8           queryOptions.setCodingScheme("NCI Thesaurus");
9           CodingSchemeVersionOrTag csvt = new CodingSchemeVersionOrTag();
10          csvt.setVersion("09.10d");
11          queryOptions.setCodingSchemeVersionOrTag(csvt);
12       Entity entity = new Entity()
13       entity.setEntityCode("C1234");
14       List&lt;Entity&gt; list = appService.search(Entity.class, entity, queryOptions);
15     } catch(ApplicationException ex){
16     }
17  }
```

The following table explains specific statements in the code by line number.

Line Number	Explanation
4	Creates an instance of a class that implements the LexEVSApplicationService interface. This interface defines the service methods used to access data objects.
7	Construct the QueryOptions Object.
8	Populate the QueryOptions with the desired Coding Scheme.
9	Construct a CodingSchemeVersionOrTag Object.
10	Populate the CodingSchemeVersionOrTag Object with the desired Version.
11	Populate the QueryOptions with the above CodingSchemeVersionOrTag Object.
12	Construct the Query By Example Object and populate it with the desired search critieria. For this example, seach for any 'Entity' with an 'entityCode' attribute equaling 'C1234'.
14	Calls the search method of the LexEVSApplicationService object, along with the QueryOptions. This method returns a List Collection. This list will contain all of the 'Entity' Objects that match the search critieria, while being further modified by the QueryOptions. It this case, it will return all 'Entity' Objects with an 'entityCode' of "C1234" belonging to the CodingScheme "NCI Thesaurus" Version "09.10d".

Web Services API

The caCORE LexEVS Web Services API enables access to caCORE LexEVS data and vocabulary data from development environments where the Java API cannot be used, or where use of XML Web services is more desirable. This includes non-Java platforms and languages such as Perl, C/C++, .NET framework (C#, VB.Net), and Python.

The Web services interface can be used in any language-specific application that provides a mechanism for consuming XML Web services based on the Simple Object Access Protocol (SOAP). In those environments, connecting to caCORE LexEVS can be as simple as providing the end-point URL. Some platforms and languages require additional client-side code to handle the implementation of the SOAP envelope and the resolution of SOAP types.To view a list of packages that cater to different programming languages, visit http://www.w3.org/TR/SOAP/ and http://www.soapware.org/.

To maximize standards-based interoperability, the caCORE Web service conforms to the Web Services Interoperability Organization (WS-I) basic profile. The WS-I basic profile provides a set of non-proprietary specifications and implementation guidelines that enable interoperability between diverse systems. For more information about WS-I compliance, visit http://www.ws-i.org.

On the server side, Apache Axis is used to provide SOAP-based, inter-application communication. Axis provides the appropriate serialization and deserialization methods for the JavaBeans to achieve an application-independent interface. For more information about Axis, visit http://ws.apache.org/axis/

Configuration

The caCORE/LexEVS WSDL file is located at http://lexevsapi.nci.nih.gov/lexevsapi50/services/lexevsapi50Service?wsdl. In addition to describing the protocols, ports, and operations exposed by the caCORE LexEVS Web service, this file can be used by a number of IDEs and tools to generate stubs for caCORE LexEVS objects. This enables code on different platforms to instantiate native objects for use as parameters and return values for the Web service methods. For more information on how to use the WSDL file to generate class stubs, consult the specific documentation for your platform.

The caCORE LexEVS Web services interface has a single end point called `lexevsapi50Service`, which is located at http://lexevsapi.nci.nih.gov/lexevsapi50/services/lexevsapi50Service. Client applications should use this URL to invoke Web service methods.

Building a Java SOAP Client

LexEVSAPI provides a tool to create a Java SOAP client capable of connecting to a LexEVSAPI SOAP service.

In the ./webServiceSoapClient contains a build.xml file that will construct a LexEVSAPI SOAP client. Before building, you may edit this build.xml file to customize the build process. Editable properties include 'wsdlURL' and 'webServiceNamespace'. An example configuration is below:

```
<property name="wsdlURL" value="http://bmidev4:8180/lexevsapi50/services/lexevsapi50Service?wsdl"/>
<property name="webServiceNamespace" value="http://bmidev4:8180/lexevsapi50/services/lexevsapi50Service"/>
```

To build the client, use the command `ant all` from the ./webServiceSoapClient directory.

XML-HTTP API

The caCORE LexEVS XML-HTTP API, based on the REST (Representational State Transfer) architectural style, provides a simple interface using the HTTP protocol. In addition to its ability to be invoked from most Internet browsers, developers can use this interface to build applications that do not require any programming overhead other than an HTTP client. This is particularly useful for developing Web applications using AJAX (Asynchronous JavaScript and XML).

Service Location and Syntax

The CORE EVS XML-HTTP interface uses the following URL syntax:

```
http://{server}/{servlet}?query={returnClass}&{criteria}
        &startIndex={index}
        &codingSchemeName={codingSchemeName}
        &codingSchemeVersion={codingSchemeVersion}
```

The following table explains the URL syntax used by the caCORE LexEVS XML-HTTP interface, indicates whether specific elements are required, and gives examples.

Element	Meaning	Required	Example
`server`	Name of the Web server on which the caCORE LexEVS 5.0 Web application is deployed.	Yes	`lexevsapi.nci.nih.gov /lexevsapi50`
`servlet`	URI and name of the servlet that will accept the HTTP GET requests.	Yes	`lexevsapi50/GetXML` `lexevsapi50/GetHTML`
`returnClass`	Class name indicating the type of objects that this query should return.	Yes	–
`criteria`	Search request criteria describing the requested objects.	Yes	–
`index`	Starting index of the result set.	No	–
`codingSchemeName`	Restrict the query to a specific Coding Scheme Name.	No	–
`codingSchemeVersion`	Restrict the query to a specific Coding Scheme Version.	No NOTE: Must be used in conjunction with a 'codingSchemeName'	–

The caCORE LexEVS architecture currently provides two servlets that accept incoming requests:

- *GetXML* returns results in an XML format that can be parsed and consumed by most programming languages and many document authoring and management tools.
- *GetHTML* presents result using a simple HTML interface that can be viewed by most modern Internet browsers.

Within the request string of the URL, the criteria element specifies the search criteria using XQuery-like syntax. Within this syntax, square brackets (/[/]) represent attributes and associated roles of a class, the *at* symbol (@) signals an attribute name/value pair, and a forward slash character ✅ specifies nested criteria.

Criteria statements in XML-HTTP queries generally use the following syntax (although you can also build more complex statements):

```
{ClassName}[@{attributeName}={value}] [@{attributeName}={value}]...

ClassName}[@{attributeName}={value}]/

{ClassName}[@{attributeName}={value}]/...
```

The following table explains the syntax for criteria statements within XML-HTTP queries and gives examples.

Parameter	Meaning	Example
`ClassName`	The name of a class.	`Entity`
`attributeName`	The name of an attribute of the return class or an associated class	`_entityCode`
`value`	The value of an attribute.	`C123*`

Examples of Use

The examples in the following table demonstrate the usage of the XML-HTTP interface. In actual usage, these queries would either be submitted by a block of code or entered in the address bar of a Web browser. The servlet name *GetXML* in each of the examples can be replaced with *GetHTML* to view with layout and markup in a browser.

Query	`http://evsapi.nci.nih.gov/evsapi41/GetXML?query=DescLogicConcept[_entityCode=C123*`
Semantic Meaning	Find all objects of type Entity that contain an 'entityCode' matching the pattern *'C123*'*.

Working with Result Sets

Because HTTP is a stateless protocol, the caCORE LexEVS server cannot detect the context of any incoming request. Consequently, each invocation of *GetXML* or *GetHTML* must contain all of the information necessary to retrieve the request, regardless of previous requests. Developers should consider this when working with the XML-HTTP interface.

- **Controlling the Start Index** - To specify a specific start position in the result set, specify the *&startIndex* parameter. This will scroll to the desired position within the set of results.
- **Internal-Use Parameters** - A number of parameters, such as &resultCounter, &pageSize, and &page, are used internally by the system and are not designed to be set by the user.

> **Note**
>
> When specifying attribute values in the query string, note that use of the following characters generates an error:
> [ ] / \ # & %

Distributed LexBIG API

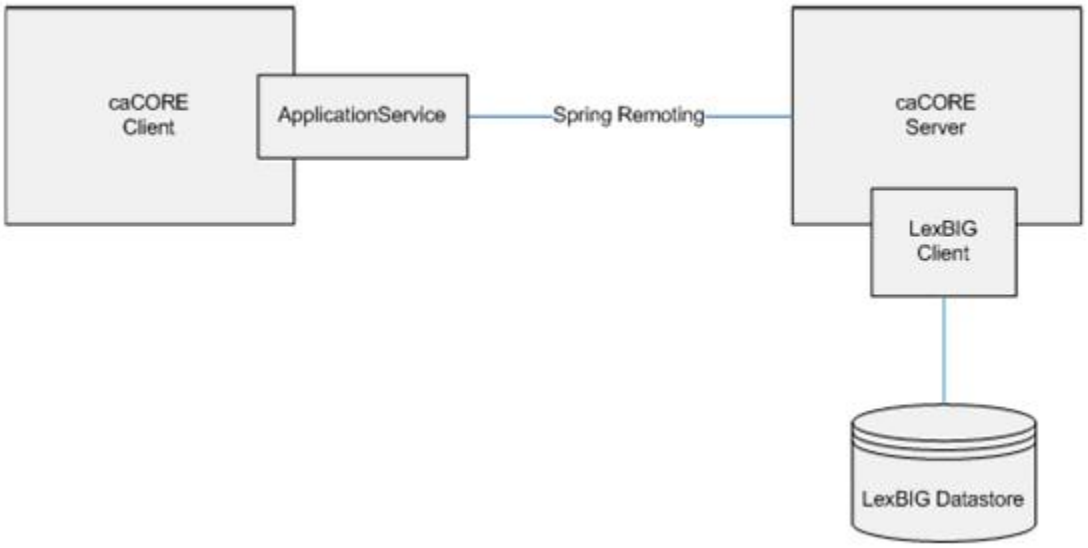
Overview

In place of the existing EVS 3.2 object model, caCORE LexEVS is making a gradual transition toward a pure LexBIG back-end terminology server and exposure of the LexBIG Service object model. caCORE 3.2 and earlier required a custom API layer between external users of the system and the proprietary Apelon Terminology Server APIs. With the transition to LexBIG, caCORE LexEVS can publicly expose the open source terminology service API without requiring a custom API layer.

Architecture

The LexBIG API is exposed by the LexEVS caCORE System for remote, distributed access (Figure 4.5). The caCORE System's `LexEVSApplicationService` class implements the `LexBIGService` interface, effectively exposing LexBIG via caCORE.

Since in many cases the objects returned from the `LexBIGService` are not merely beans, but full-fledged data access objects (DAOs), the caCORE LexEVS client is configured to proxy method calls into the LexBIG objects and forward them to the caCORE server so that they execute within the LexBIG environment.



The DLB environment will be configured on the caCORE LexEVS Server (http://lexevsapi.nci.nic.gov/lexevsapi50). This will give the server access to the LexBIG database and other resources. The client must therefore go through the caCORE LexEVS server to access any LexBIG data.

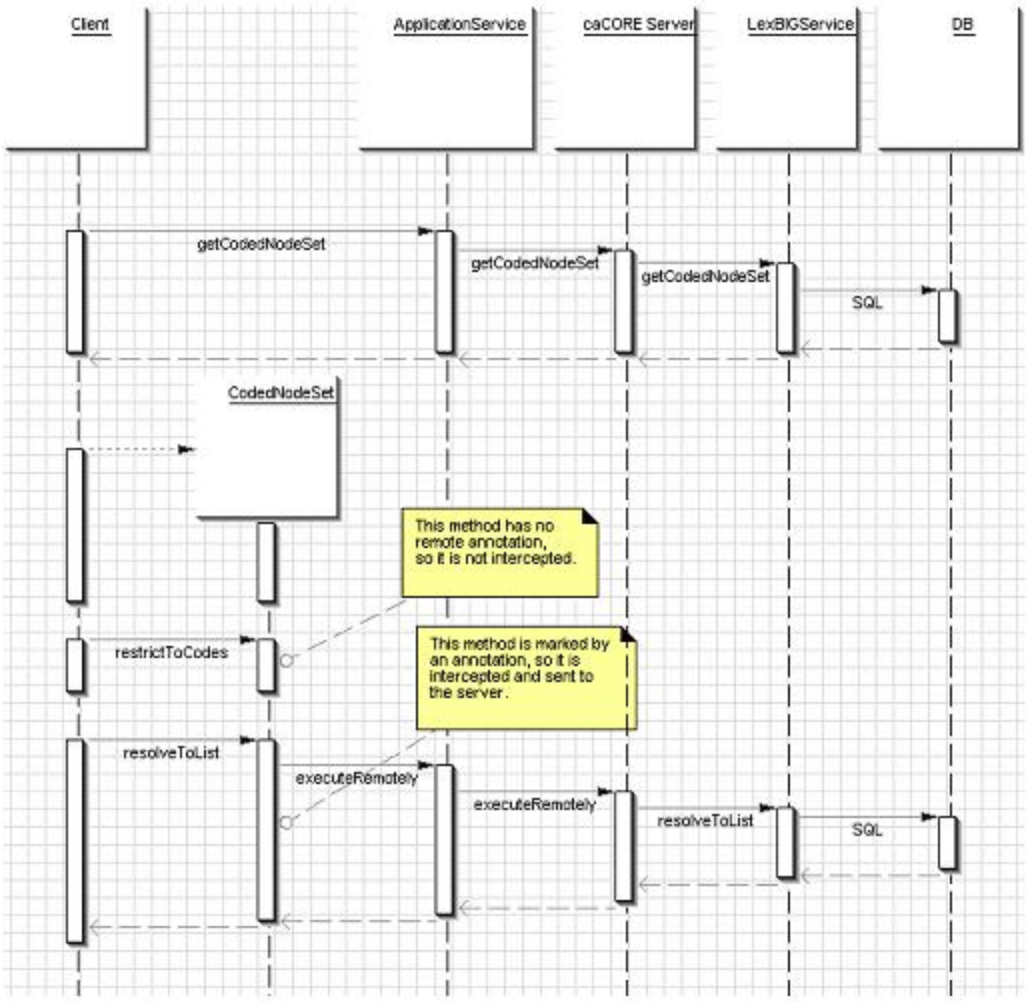
LexBIG Annotations

To address LexBIG DAOs, the LexBIG API integration incorporated the addition of (1) Java annotation marking methods that can be safely executed on the client side; and (2) classes that can be passed to the client without being wrapped by a proxy. The annotation is named `@lgClientSideSafe`. Every method in the LexBIG API that is accessible to the caCORE LexEVS user had to be considered and annotated if necessary.

Aspect Oriented Programming Proxies

LexBIG integration with caCORE LexEVS was accomplished using Spring Aspect Oriented Programming (AOP) to proxy the LexBIG classes and intercept calls to their methods. The caCORE LexEVS client wraps every object returned by the `LexBIGService` inside an AOP Proxy with advice from a `LexBIGMethodInterceptor` ("the interceptor").

The interceptor is responsible for intercepting all client calls on the methods in each object. If a method is marked with the `@lgClientSideSafe` annotation, it proceeds normally. Otherwise, the object, method name, and parameters are sent to the caCORE LexEVS server for remote execution.



LexBIG API Documentation

The Mayo Clinic wrote the LexBIG 5.0 API. Documentation describing the LexBIG Service Model is available on the LexGRID Vocabulary Services for caBIG GForge site at https://gforge.nci.nih.gov/frs/?group_id=14.

LexBIG Installation and Configuration

The DLB API is strictly a Java interface and requires Internet access for remote connectivity to the caCORE LexEVS server. Access to the DLB API requires access to the `lexevsapi-client.jar` file, available for download on the NCICB Web site. The `lexevsapi-client.jar` file needs to be available in the classpath. For more information, see the section, Installing and Configuring the LexEVS 5.0 Java API.

Example of Use

Example 4.6: Using the DLB API

The following code sample shows use of the DLB API to retrieve the list of available coding schemes in the LexBIG repository.

```
1  public class Test {
2      /**
3    * Initialize program variables
4    */
5
6    private String codingScheme = null;
7    private String version = null;
8
9    DLBAdapter adapter = null;
10    LexBIGService lbSvc;
11
```

```
12    public Test(String codingScheme, String version)
13    {
14        //Set the LexEVS URL (for remote access)
15        String evsUrl = "http://lexevsapi.nci.nih.gov/lexevsapi50/http/remoteService";
16
17        boolean isRemote = true;
18        this.codingScheme = codingScheme;
19        this.version = version;
20
21        // Get the LexBIG service reference from LexEVS Application Service
22        lbSvc = (LexEVSApplicationService)ApplicationServiceProvider.
getApplicationServiceFromUrl(evsUrl, "EvsServiceInfo");
23
24        // Set the vocabulary to work with
25        Boolean retval = adapter.setVocabulary(codingScheme);
26
27        codingSchemeMap = new HashMap();
28        try {

32            // Using the LexBIG service, get the supported coding schemes
33            CodingSchemeRenderingList csrl = lbSvc.getSupportedCodingSchemes();
34
35            // Get the coding scheme rendering
36            CodingSchemeRendering[] csrs = csrl.getCodingSchemeRendering();
37
38            // For each coding scheme rendering...
39            for (int i=0; i&lt;csrs.length; i++) {
40                CodingSchemeRendering csr = csrs[i];
41
42                // Determine whether the coding scheme rendering is active or not
43                Boolean isActive = csr.getRenderingDetail().getVersionStatus()
              .equals(CodingSchemeVersionStatus.ACTIVE);
44                if (isActive != null && isActive.equals(Boolean.TRUE)) {
45
46                    // Get the coding scheme summary
47                    CodingSchemeSummary css = csr.getCodingSchemeSummary();
48
49                    // Get the coding scheme formal name
50                    String formalname = css.getFormalName();
51
52                    //Get the coding scheme version
53                    String representsVersion = css.getRepresentsVersion();
54                    CodingSchemeVersionOrTag vt = new;
55                    CodingSchemeVersionOrTag();
56                    vt.setVersion(representsVersion);
57
58                    // Resolve coding scheme based on the formal name
59                    CodingScheme scheme = null;
60
61                    try {
62                    scheme =
                      lbSvc.resolveCodingScheme(formalname, vt);
63                      if (scheme != null)
64                      {
65                          codingSchemeMap.put((Object) formalname, (Object) scheme);
66                      }
67                      } catch (Exception e) {
68                      // Resolve coding scheme based on the URI
69                      String uri = css.getCodingSchemeURI();
70                      try {
71                          scheme = lbSvc.resolveCodingScheme(uri, vt);
72                          if (scheme != null)
73                          {
74                              codingSchemeMap.put((Object) formalname, (Object) scheme);
75                          }
76                          } catch (Exception ex) {
77                          String localname = css.getLocalName();
78
79                          // Resolve coding scheme based on the local name
80                          try {
81                              scheme = lbSvc.resolveCodingScheme(localname, vt);
```

```
                                if(scheme != null)
                                {
                                    codingSchemeMap.put((Object) formalname, (Object) scheme);
                                }
                                } catch (Exception e2) {
                            }
                        }
                    }
                }
            }
            } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
    *Main
    */
     public static void main (String[] args)
     {
         String name = "NCI Thesaurus";
         String version = "06.12d";

         // Instantiate the Test Class
         Test test = new Test(name, version);
     }
}
```