
IST-2001-37264



Creating a Smart Space for Learning

Implementing SQI via SOAP Web-Services

Date:	10-02-2004
Version:	0.7
Editor(s):	Stefan Brantner, Thomas Zillinger (BearingPoint)

1 *Java Archive for SQI service implementation*

In order to facilitate the development of SQI services as well as the invocation of remote services a Java archive file (JAR) has been developed that contains SOAP wrapper classes for all the services needed for SQI¹. This chapter describes this Java archive and provides source code examples on how to implement and invoke a service. The JAR file named `queryservice.jar` which is part of the ZIP file `queryservice.zip` can be obtained from the following web-address: <http://nm.wu-wien.ac.at/e-learning/interoperability>.

The ZIP file also contains the WSDL files of all the involved services in the sub-directory "wsdl". These files can be used for example to automatically create the wrapper classes for a non-Java system or for a web-service package other than Apache Axis.

Furthermore the ZIP file contains all the Java-Doc files for the classes described in section 1.1.

1.1 *Java archive structure*

Each service is implemented in its own Java package below the base package `org.elena.services`. Each of these packages contains all the wrapper classes in order to implement this service or to invoke this service on a remote host. Figure 1 depicts all the packages as well as the Java files that are part of the JAR file².

There are two different types of services specified in the SQI specification:

- Services that implement classical client-server type of interfaces (client calls server), i.e. Session Management.
- Services that can not be classified as client-server because the communication is two-way – both sides of the communication need to be server and client for different types of methods. This is the case for the query and the resource management services. Here, for example both sides carrying out a query need to act as client and server.

For the second type of services where both communication partners act as client as well as server the respective interface definition have been split into two separate classes (with the same name!) that reside in two different sub-packages of the service base package. For example the part of the query interface that must be implemented by the source resides in package `org.elena.services.query.source`. The part of the interface that must be implemented by the target is in `org.elena.services.query.target`.

¹ The SOAP wrapper classes can be only used with the Java Open-Source SOAP package Apache AXIS.

² The JAR file obviously also contains the compiled Java class files.

```
+-- org
  +- elena
    +- services
      +- sessionmgmt
        +- SessionManagement.java
        +- SessionManagementService.java
        +- SessionManagementServiceLocator.java
        +- SessionManagementSoapBindingImpl.java
        +- SessionManagementSoapBindingStub.java
      +- query
        +- source
          +- Query.java
          +- QueryService.java
          +- QueryServiceLocator.java
          +- QuerySoapBindingImpl.java
          +- QuerySoapBindingStub.java
        +- target
          +- Query.java
          +- QueryService.java
          +- QueryServiceLocator.java
          +- QuerySoapBindingImpl.java
          +- QuerySoapBindingStub.java
    +- tools
      +- sessionmgmt
        +- ComSession.java
        +- ComSessionManager.java
```

Figure 1: Java archive packages and classes

1.2 Service implementation

In order to implement one of the above services as a SOAP service the empty implementation classes provided in each of the above packages can be used. These implementation classes (file suffix is "Impl.java") implement the respective interface and can be used as a starting point for ones own service implementation.

As an example we describe here the implementation of the target query interface, which is implemented in the Java interface class `org.elena.services.query.target.Query.java`. This interface consists of the following methods:

- `void setQueryLanguage(String, String)`
- `void setResultsFormat(String, String)`
- `void setResultsSetSize(String, Integer)`
- `void setMaxQueryResults(String, Integer)`
- `void setMaxDuration(String, Integer)`
- `String synchronousQuery(String, String)`
- `String getAdditionalQueryResults(String, Integer)`
- `Integer getTotalResultsCount(String)`

- void **asynchronousQuery**(String, String, String)
- void **setSourceLocation**(String, String)
- String **getResourceDescription**(String, String)

In order to write a new implementation of the target query service the Java class `org.elena.services.query.target.QuerySoapBindingImpl.java` can be taken as starting point. In this file all the above methods are already there but with an empty implementation.

Implementers are strongly recommended to change the package name of the default implementation and write their own classes based on those default implementations. As a consequence the WSDD deployment files have to be adapted accordingly (see section 1.4).

As soon as the new service implementation is finished, it can be deployed to a TomCat 4 installation via one of the WSDD files part of the ComAPI.zip file available under <http://nm.wu-wien.ac.at/e-learning/interoperability>. This deployment process is described in greater detail in section 1.4.

1.3 Service Invocation

In order to invoke a SOAP service implemented on a LM network node, one can directly use the classes provided in the `soapservice.jar` archive. If, for example, a source network node wants to query a target network node it needs to use the wrapper classes provided in package `org.elena.services.query.target`.

The following Java code first gets a session-ID from the target's session management service and then invokes a sample synchronous query at the given target:

```
import java.net.URL;

// classes from the soapservice.jar archive
import org.elena.service.sqi.target.*;
import org.elena.service.sessionmgmt.*;
import org.elena.tools.sessionmgmt.MD5;

public class QueryTester
{
    public static void main(String[] args)
    {
        String targetServiceBaseURL = "http://my.target.host:8080/services/";
        // get a session ID from the target node
        SessionManagementService service1 =
            new SessionManagementServiceLocator();
        SessionManagement sessionMgmt = service1.getSessionManagement(
            new URL(targetServiceBaseURL + "Query"));
        String sessionID = sessionMgmt.createSession("username",
            MD5.getMD5FromString("passwordMD5"));

        // send a synchronous query to the target node
    }
}
```

```
    QueryService service = new QueryServiceLocator();
    Query port =
        service.getQuery(new URL(targetServiceBaseURL + "Query"));
    String queryResult =
        port.synchronousQuery(sessionId, "Hello Query");
    System.out.println("Result = " + queryResult);
}
}
```

Listing 1: A simple Java client that requests a session ID at the target and then sends a synchronous query to the target.

In order for the above code to work the following libraries need to be in the CLASSPATH:

- axis.jar
- axis-ant.jar
- commons-discovery.jar
- commons-logging.jar
- jaxrpc.jar
- saaj.jar
- wsdl4j.jar

All of these Java archive files are part of the Axis 1.1 distribution which can be downloaded at <http://ws.apache.org/axis/index.html>.

1.4 Service Deployment with Axis and TomCat

This chapter only deals with using Axis in existing web-applications. Documentation about other possibilities of using axis can be found at the Axis home-page.

In case Axis web-services shall be used in an existing web-application the following two steps need to be taken first:

- Add axis.jar, wsdl4j.jar, saaj.jar, jaxrpc.jar and the other dependent libraries (see above) to your web-application's /WEB-INF/lib directory.
- Copy the following Axis Servlet declarations and mappings to your own web.xml file:

```
<servlet>
    <servlet-name>AxisServlet</servlet-name>
    <display-name>Apache-Axis Servlet</display-name>
    <servlet-class>
        org.apache.axis.transport.http.AxisServlet
    </servlet-class>
</servlet>

<servlet>
    <servlet-name>AdminServlet</servlet-name>
    <display-name>Axis Admin Servlet</display-name>
    <servlet-class>
        org.apache.axis.transport.http.AdminServlet
```

```

        </servlet-class>
        <load-on-startup>100</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>AxisServlet</servlet-name>
        <url-pattern>/servlet/AxisServlet</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>AxisServlet</servlet-name>
        <url-pattern>*.jws</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>AxisServlet</servlet-name>
        <url-pattern>/services/*</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>AdminServlet</servlet-name>
        <url-pattern>/servlet/AdminServlet</url-pattern>
    </servlet-mapping>

```

As soon as you have created your own web-services classes according to 1.2 Axis needs to be told how to expose these web-services. Axis takes a Web Service Deployment Descriptor (WSDD) file that describes in XML what the service is, what methods it exports and other aspects of the SOAP endpoint.

In order to deploy your own web-service follow the following three steps:

- **CLASSPATH setup:**

Java must be able to find `axis.jar`, `commons-discovery.jar`, `commons-logging.jar`, `jaxrpc.jar`, `saaj.jar`, `log4j-1.2.8.jar` (or whatever is appropriate for your chosen logging implementation), and the XML parser jar file or files (e.g., `xerces.jar`) in your CLASSPATH.

```

set AXIS_HOME=c:\axis
set AXIS_LIB=%AXIS_HOME%\lib
set CLASSPATH=%AXIS_LIB%\axis.jar;
                %AXIS_LIB%\commons-discovery.jar;
                %AXIS_LIB%\commons-logging.jar;
                %AXIS_LIB%\jaxrpc.jar;
                %AXIS_LIB%\saaj.jar;
                %AXIS_LIB%\log4j-1.2.8.jar;
                %AXIS_LIB%\xml-apis.jar;
                %AXIS_LIB%\xercesImpl.jar

```

- **Find the appropriate deployment descriptor:**

In the appropriate service sub-directory of the `soapservice.zip` ZIP file (see directory `org.elena.service`) you can find the appropriate web-service deployment descriptor file `deploy.wsdd` for your web-service. Copy this file to a directory on your hard disc.

In case you have changed the package name of your service implementation class you will have to adapt the following entry to show the correct package and class name:

```
<parameter name="className"
```

```
value="your.package.YourServiceImplClass" />
```

- **Run the Axis admin client:**

Execute the following command from the directory you copied the `deploy.wsdd` file to:

```
java org.apache.axis.client.AdminClient
-lhttp://localhost:8080/[WebAppContext]/services/AdminService
deploy.wsdd
```

The above URL of the `AdminService` must be constructed according to the context name of your web-application.

More detailed information about service deployment and Axis installation can be found in the Axis installation instructions at <http://ws.apache.org/axis/java/install.html>.