Web services -- table of contents

Development

Administration

4.8: Web services - an overview

Web services are self-contained, modular applications that can be described, published, located, and invoked over a network.Web services could be weather reports or stock quotes. Transaction Web services,supporting business-to-business (B2B) or business-to-client (B2C) operations, could be airline reservationsor purchase orders.

Web services reflect a new "service-oriented" approach to programming, based on the idea of building applications by discovering and implementing network-available services, or by invoking available applicationsto accomplish some task. This "service-oriented" approach is independent of specific programming languages or operating systems.Instead, Web services rely on pre-existing transport technologies (such as HTTP) and standard data encoding techniques (such as XML)for their implementation.

The Web services architecture describes three roles:

1. Service provider
2. Service requester
3. Service broker

Web services components provide three basic operations:

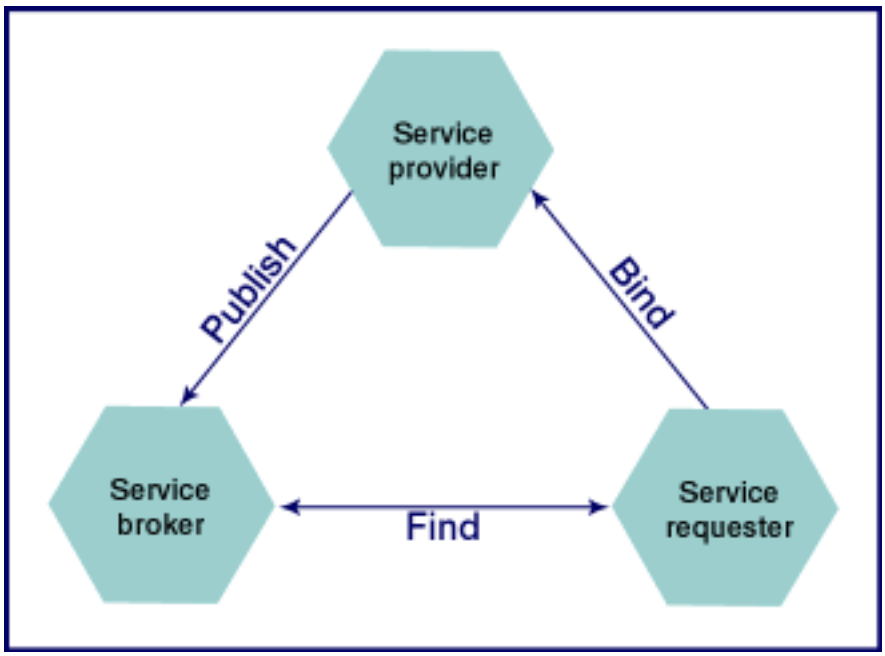
1. Publish
2. Find
3. Bind

In order for some component to become a Web service, it must be:

- Created, and its interfaces and invocation methods must be defined
- Published to some repository
- Easy to locate by potential users
- Invoked and implemented by users
- Unpublished when it is no longer available

As illustrated in the graphic,

- Web service descriptions can be created and published by service providers who create on-line resources for personal and business use.
- Web services can be categorized and searched by specific broker services.
- Web services can be located and invoked by requesters of the services.



With Web services, programming complexity is reduced because application designers do not have to worry about implementing the services they are invoking. Interactions in Web services are bound dynamically at runtime.A service requester describes the features of the required service and uses the service broker to find an appropriate service.

WebSphere Application Server supports making the following artifacts into Web services:

- Java beans
- Enterprise Java Beans
- BSF supported scripts
- DB2 stored procedures

See article Web services components for a description of the key components that comprisea Web service.

Visit URL, www.alphaworks.ibm.com/tech/webservicestoolkit,to access the Web services toolkit on Alphaworks. This site provides tools for creating WSDL files and SOAP clients, and describes working examples.

Learn more about Web services. Register for the Web services tutorialon Alphaworks.

4.8.1: Web services components

These are the key components of a Web service:

- SOAP (simple object access protocol)
- WSDL (Web Services Description Language)
- UDDI (Universal Discovery , Description and Integration Protocol)
- UDDI4J (client version of UDDI)

- ## SOAP or Simple Object Access Protocol

  is a new protocol created by IBM, Microsoft, Userland, and DevelopMentor to support remote procedure calls and other requests over HTTP.Built on HTTP and XML, SOAP attempts to convert application servers into object servers.

  See the W3C SOAP protocol site for more information on SOAP messages, supported datatypes, and attributes. For SOAP implementation guidelines, visit the Apache site.

  SOAP requests and the responses are XML based. The following examples illustrate a SOAP request and response:

  Sample SOAP Request

```
 Sample SOAP Request   POST /Supplier HTTP/1.1  Host: www.somesupplier.com  Content-Type: text/xml;
charset="utf-8"  Content-Length: nnnn  SOAPAction: "Some-URI"  <SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">    <SOAP-ENV:Body>
<m:OrderItem xmlns:m="Some-URI">     <RetailerID>557010<</RetailerID>
<ItemNumber>1050420459</ItemNumber>     <ItemName>AMF Night Hawk Pearl M2</ItemName>
<ItemDesc>Bowling Ball</ItemDesc>     <OrderQuantity>100</OrderQuantity>
<WholesalePrice>130.95</WholeSalePrice>     <OrderDateTime>2000-06-19 10:09:56</OrderDateTime>
</m:OrderItem>   </SOAP-ENV:Body>  </SOAP-ENV:Envelope>
```

  The SOAP request indicates that the OrderItem method, from the "Some-URI" namespace, should be invoked from *http://www.somesupplier.com/Supplier*.Upon receiving this request, the supplier application at *www.somesupplier.com* executes the business logic that corresponds to OrderItem.

  The SOAP protocol does not specify how to process the order. The supplier could run a CGI script, invoke a servlet, or perform any other process that generates the appropriate response.

  > See article SOAP support for the list of artifacts that WebSphere Application Server supports as Web services.

  In this example, the SOAP Envelope element is the top element of the XML document that represents the SOAP message. The reference to the XML namespace (`xmlns:m="Some-URI"`) specifies the namespace to use for the SOAP identifiers. This request is asking the application to place an order for the item identified by the elements:

  - RetailerId
  - ItemNumber
  - ItemName
  - ItemDesc
  - OrderQuantity
  - WholesalePrice
  - OrderDateTime

  The response comes in the form of an XML document that contains the results of the processing, in this case, the order number for the order placed by the retailer. The response is sent by the service provider located at *http://www.somesupplier.com/Supplier*.

  Sample SOAP Response

```
 HTTP/1.1 200 OK  Content-Type: text/xml; charset="utf-8"  Content-Length: nnnn  <SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>    <SOAP-ENV:Body>
<m:OrderItemResponse xmlns:m="Some-URI">      <OrderNumber>561381</OrderNumber>
</m:OrderItemResponse>    </SOAP-ENV:Body>  </SOAP-ENV:Envelope>
```

  The response does not include a SOAP-specified header.The results are placed in an element whose name matches the method name (*OrderItem*) with the suffix, "Response" as in*OrderItemResponse*.

  Although Apache SOAP allows for SOAP over SMTP, WebSphere Application Server only supports SOAP over HTTP.

  The SOAP Javadoc is shipped with WebSphere Application Server.

  Review WebSphere Application Server's Javadoc for SOAP implementation details.

- ## WSDL or Web Services Description Language

is an XML-based interface definition language that provides operational information about a service, such as the service interface, implementation details, access protocol, and contact endpoints. Compliant server applications must support these interfaces, and client users can learn from thedocument how a service should be accessed.

WebSphere Application Server does not provide toolsfor generating WSDL files.

View a WSDL representation in the AddressBook2 sample.

See article UDDI4J samples for more information.

Review the WSDL specifications at W3C WSDL protocol site.

- ## UDDI or Universal Discovery Description and Integration (Project)

is a comprehensive, open industry initiative enabling businesses to:

1. Discover each other
2. Define how they interact over the Internet, and share information in a global registry architecture.

WebSphere Application Server does not provide a private UDDI directory. IBM, among others,provides public UDDI registries. For more information about UDDI, see www.uddi.org.Also visitAlphaworksfor the Web services toolkit, which includes an IBM implementation of a private UDDI registry.

UDDI is the building block which enables businesses to quickly, easily, and dynamically find and transact with one another by means of their preferred applications.

As described in the Web services overview, UDDI provides the three basic Web services functions: publish, find, and bind.

- ## UDDI4J

is an open-source Java implementation of the Universal Discovery, Description, and Integration protocol (UDDI).UDDI4J contains an implementation of the client side of UDDI (everything your application needs to publish, find, and bind a Web service).It also includes the source code, and the complete Javadoc for the APIs. For more information,visit the UDDI4J open source site at oss.software.ibm.com/developerworks/projects/uddi4j.

Review IBM's Javadoc for UDDI4J implementation details.

4.8.1.1: UDDI4J Overview

UDDI4J is a Java class library that provides an API that is used to interact with a UDDI registry. This class library generates and parses messages sent toand received from a UDDI server.

The central class in this set of APIs is:

`com.ibm.uddi.client.UDDIProxy`

This class is a proxy for the UDDI server thatis accessed from the client code. Its methods map to theUDDI Programmer's API Specification.Review IBM's Javadoc foradditional implementation details.

The classes within `com.ibm.uddi.datatype` represent data objects that send or receiveUDDI information. In the business and servicemodel, the data objects are also known as subpackages.

The subpackage `com.ibm.uddi.request` contains messages sent to the server.Generally, these classes are not used directly;rather, they are invoked by the UDDIProxy class.

Similarly, the subpackage `com.ibm.uddi.response` represents response messages from a UDDIserver.

UDDI4J error handling

When invoking `UDDIProxy` inquiry methods, `UDDIException` is thrown when errors are received fromthe UDDI proxy.`UDDIException` can contain a DispositionReport with information regarding the error.

APIs that do not return a data object, providethe disposition report.

`SOAPException` is thrown if a communication error occursor if the resulting data cannot be parsed as a valid SOAP message.

View the file 4.8.1.1.1: UDDI4J Samples for API usage examples.

For more information,visit the UDDI4J open source site at oss.software.ibm.com/developerworks/projects/uddi4j.

4.8.1.1.1: UDDI4J samples

A set of samples is provided to illustrateusing the inquiry and publishAPIs, and to demonstrate error handling.

Note: WebSphere Application Server does ***not***provide a UDDI registry. The IBM UDDI test registry is located at www.ibm.com/services/uddi/

Any sample that requires you to "publish," "save," or "delete" requires a userid and password. You can only invoke the "find" sample without a userid and password.

To get a userid and password:

1. Access the UDDI test registry
2. Register for your userid and password

   The registration process requires you to activate your id beforeattempting to use the publish or delete examples.

   Note: If the registry is not operational, keep trying. This is a test registry andat times it is not available.
3. Use your registered userid and password when running the *SaveBusinessExample*and *DeleteBusinessExample* samples.

Your samples consist of:

- *FindExample* - is the "Hello world" of UDDI programs. It is the simplest sample of the UDDI API.
- *SaveBusinessExample* - is an example of using the publish API. It logs into the server using the `get_authToken` method; then attempts to save a business.
- *DeleteBusinessExample* - searches for a particular business using the inquiry API, finds the associated businesskey, logs into the server, and then attempts to delete the business it found.

  When running *DeleteBusinessExample*, you might receive the following error messages:

  ```
  Get authtokenReturned authToken:ADA3DC40-2531-11D5-9EB0-832611502FD0Search for 'Sample business' to
  deleteFound business key:D3DD4036-00E4-F124-050B-C6113996AA77Errno:10140   ErrCode:E_userMismatch
  ErrText:E_userMismatch (10140)   Cannot change data that is controlled by another party.
  businessEntity = D3DD4036-00E4-F124-050B-C6113996AA77Found business
  key:61AE2CC0-0F2C-11D5-BC1E-B763254A2930Errno:10140   ErrCode:E_userMismatch
  ErrText:E_userMismatch (10140)    Cannot change data that is controlled by another party.
  businessEntity = 61AE2CC0-0F2C-11D5-BC1E-B763254A2930Found business
  key:3BB274CF-00E3-FA94-9B72-C6113996AA77
  ```

  This is not a problem with the sample. *DeleteBusinessExample* issues a query for the business name specified in the code and receives a list of entries with that name. The sample then tries to delete each entry in the list. These error messages occur when the sample tries to delete entries that you do not own.

Accessing the samples

To access these samples, you can either installthe soapsamples.ear, or you canexpand the `soapsamples.ear` using the EarExpandertool.

These are the steps to access the samples:

1. Create a directory to hold the expanded soapsamples.earcontents.
2. From the *`product_installation_root`*`\bin` directory,enter the following commands:

   ```
   EarExpander -ear ..\installableApps\soapsamples.ear-expandDir ..\temp\soapsamples -operationexpand
   -expansionFlags war
   ```
3. Issue the `cd` command to change to the `installedApps/soapsamples.ear` orto the target directory specified in the-expandDir argument
4. Issue the `cd` command to change to `UDDISamples` directory. The source for the samplesis included in the `src` directory.

The samples require several pieces of information.The sample source files can be edited and these valuessubstituted. The required values are:

- `InquiryURL`: The URL of the UDDI server against which to run inquiries.
- `PublishURL`: The URL of the UDDI server to run publish requests. Typically, this is a SSL connection.
- `Userid`: When publishing, a userid is required for authentication.
- `Password`: This is the password for the referenced userid. Password is referred to as a credential in UDDI terminology.

Running the samples

WebSphere Application Server provides an number of UNIX scripts and DOS .bat files to run the samples. These scripts (or .bat files) add the required jar files to the classpath. Use a text editor (such as Notepad on Windows NT or VI or E3 on UNIX) to view the scripts (or .bat files). They describethe resources that you need to run the samples.

The scripts are located in directory `UDDISamples/unix_scripts`.On Windows NT, the .bat files are located in directory `UDDISamples\nt_bat`.

The scripts are put in this locationas a result of running the `EarExpander`command.

All the scripts (or .bat files) are named after the samples they run. So, for example, to invoke the *FindExample* sample, you would run the*FindExample.sh* script.

A UDDI registry might limit the number of business entities that you publish.The IBM Test registry limits you to one business entity.This means, for example, that after running the *SaveBusinessExample*, youmust run the *DeleteBusinessExample* before attempting to publish another business entity.

See the related information links for an enablement scenario.

4.8.1.2: SOAP support

Version 2.2 of the Apache SOAP implementation is integrated into WebSphere Application Server Version 4.0. Apache SOAP Version 2.2 is a Java-based implementation of the SOAP 1.1 specification with support for SOAP with attachments.

WebSphere Application Server Version 4.0 allows you to expose the following artifacts as SOAP services:

- Standard Java classes
- Enterprise beans
- Bean Scripting Framework (BSF) supported scripts
- DB2 stored procedures

Tools are provided to assist you with deployingthese artifacts as SOAP services. See article Deploying a programming artifact as a SOAP accessible Web service for more information.

As part of deploying your services,you can choose to enable the XML-SOAP Admin tool,which allows you to manage your SOAP-enabled services.

WebSphere Application Server also contains an implementation of the security extensions for SOAP. These security extensions provide secure connections and enable digitally signed messages.See article Securing SOAP services for more information.

See the related information links for an enablement scenario.

4.8.1.2.1: SOAP samples

WebSphere Application Server 4.0 provides sample services and clients that demonstrate how to access SOAP services. The SOAP samples code is based on the the Apache SOAP 2.2 samples.These samples are contained in the `soapsamples.ear` that is located in the `installableApps` directory. The source for the sample services is located in the `soapsamples.ear`.

See article DB2 Stored procedure sample setupfor information on configuring a datasource to set the *db2-userid* and *db2-password* entries.

Perform the following steps to install the samples in your server:

1. In a single-server configuration, do the following:
2. Change the directory to:
   `product_installation_root``/bin`
3. Install the EAR file by entering the following data at a command prompt:

   > The line breaks in this example are added to make the information legible. This information really exists as one line of unformatted data.

   ```
   Seappinstall -install ..\installableApps\soapsamples.ear                    -ejbdeploy false
   -interactive false
   ```
4. To access the sample services from an external Web server, run the file `GenPluginCfg.sh` on UNIX or `GenPluginCfg.bat` on Windows NT. This file makes the Web server aware of the SOAP samples.
5. Start the product.
6. Check on the availability of the sample services using the XML-SOAP Admin tool:
   a. From a browser, go to URL
      `http://localhost/soapsamples/admin/index.html`
   b. At this site, you can:
      - List available services
      - View the Apache SOAP descriptors
      - Stop and start sample services

Running the sample clients

Sample clients are provided to demonstrate how to access the installed SOAP services.These scripts require you to specify the server that will handle the request.

> If you run the script with **no** arguments, as for example *StockQuoteSample*, you will be provided with help on how to use the sample, and you will receive a description of the command line arguments that the script requires.

To access the samples, change the directory to the following on Windows NT:
`product_installation_root``\installedApps\soapsamples.ear\ClientCode\nt_bat`

On UNIX platforms, the samples directory is:
`product_installation_root``/installedApps/soapsamples.ear/ClientCode/unix_scripts`

> Issue the `chmod 755 *.sh` command to restore the executionpermissions of the UNIX scripts.

Sample	Command (entered on a single line)
Stock quote (requires Internet access)	`stockquotesample localhost IBM` > If the request appears to hang, and then you receive an "Operation timed out" error, the service was unable to reach a server on the Internet to obtain the stock quote information. You may need a direct connection to the Internet.

Address book	`AddressBookSample`
	`GET localhost "John B. Good"`
	`AddressBookSample`
	`ALL localhost`
	`AddressBookSample`
	`PUT localhost "Herman Munster" 1313 "Mockingbird Lane" Salem MA 10013 111 222 3434`

Address book example 2	`Addressbook2sample localhost`

EJB	`EJBAdderSample localhost`
	On UNIX platforms, enter:
	`EJBAdderSample.sh localhost`

Send Message	`sendMessageSample localhost ..\data\msg1.xml`
Calculator Sample	`CalculatorSample localhost` > Unlike the other SOAP samples, which are either java or enterprise beans, the Calculator Sample is a JavaScript sample. The actual calculator processing is performed by the Web service.
Mime Client sample	`MimeClientSample localhost ..\data\foo.txt`

DB2SPSample sample	`DB2SPSample localhost`
	On UNIX platforms, enter:
	`DB2SPSample.sh localhost`

Troubleshooting SOAP sample problems

If you cannot run the SOAP samples, check for the following problems:

- Can you run any of the samples, such as `http://localhost/servlet/snoop`?If not, make sure the Web server is running.

  If you can run the `snoop` sample, try accessing one of the SOAP samples again, butthis time specify the port number 9080 in addition to the host name, as for example:

  ```
  MimeClientSample localhost:9080 ...\data\foo.txt
  ```

  If adding the port number resolves the problem, you need to update the plugin configurationby running the `GenPluginCfg.bat` file on the Windows platform, or the`GenPluginCfg.sh` file on UNIX platforms.
- If the stockquote sample fails but the other samples work, you are having problems accessing the external Internet.

See the Related topics section for links to an enablement tutorial.

4.8.1.2.2: Building a SOAP client

Creating clients to access the SOAP services published in WebSphere Application Server is a straightfoward process.The Apache SOAP implementation, integrated with WebSphere Application Server, contains a clientAPI to assist in SOAP client application development.

The SOAP API documentation is available in WebSphere Application Server's javadoc.

These are the steps for creating a client thatinteracts with a SOAP RPC service:

1. **Obtain the interface descriptionof the SOAP service**

   This provides you with thesignatures of the methods that you wish toinvoke.You can either look at a WSDL file for theservice, or view the service itself to see its implementation.
2. **Create the "Call" object**

   The SOAP "Call" object is the main interfaceto the underlying SOAP RPC code.
3. **Set the target URI (Uniform Resource Identifier) in the "Call"object using the `setTargetObjectURI()` method.**

   Pass the URN (Uniform Resource Name, a type of URI), that the service uses for itsidentifier, in the deployment descriptor.
4. **Set the method name that you want to invoke in the "Call" object using the `setMethodName()` method**

   This method must be one of the methods exposed bythe service located at the URN from the previousstep.
5. **Create the necessary "Parameter" objectsfor the RPC call and then set them in the "Call"object using the `setParams()` method.**

   Ensure you have the same number and same type ofparameters as those required by the service.
6. **Execute the "Call" object's `invoke()`method and retrieve the "Response" object**

   Remember the RPC call is synchronous, so it may take some time to complete.
7. **Check the response for a fault using the`getFault()` method, and then extract any results or returnedparameters**

   While most of the providers only returna result, the DB2 stored procedure providercan also return output parameters.

Interacting with a "document-oriented" SOAP service requires you to use lower-levelApache SOAP API calls. You must first construct an "Envelope" object which containsthe contents of the message (including thebody and any headers) that you wish to send.Then create a "Message" object where you invoke the `send()` method to perform theactual transmission.

To create a secure SOAP service, do the following:

1. Create a simple object
2. Define an envelope editor
3. Specify a pluggable envelope editor
4. Define the transports

Your code may look like the following example:

```
EnvelopeEditor editor=new PluggableEnvelopeEditor(new InputSource(conf), home);SOAPTransport
transport =new FilterTransport(editor, new SOAPHTTPConnection());call.setSOAPTransport(transport);
```

The characteristics of the secure session are specified by the configuration file, "conf."

See article Securing SOAP servicesfor more information on creating secure Web services.

See article 4.8.1.2.2.1: Accessing enterprise beans through SOAPfor information on calling an EJB service.

Since the SOAP API is a standard for Web services, any clients that you create to access the WebSphere Application Server SOAP services can also runin different implementations.

See the related information links for an enablement scenario.

4.8.1.2.2.1: Accessing enterprise beans through SOAP

Calling enterprise beans through SOAP is handled in the same manner ascalling Java bean methods through SOAP. The SOAP runtime handles the bean cases for you, such as calling anenterprise bean's create method if the create was not calledpreviously.

A Web service can be a simple stateless sessionbean that performs number processing and returns a data value.When the client code makes a call to the data processing method of this service and aninstance of the stateless session is not available, the SOAP runtimedoes the following:

- Calls the EJB create method to obtain a stateless session
- Calls the requested method

At times the client code must do additional work to use enterprise beans throughSOAP. For example, if a Web application intends to use stateful or entity beans that persist data between calls, the clientrequires a reference to identify the bean instance that must be accessedin subsequent calls to methods. This reference/key can be obtained from the response objectthat the client receives on the initial call to the bean.

Response objects are created:

- When the client explicitly calls a create method
- From a `findByPrimaryKey()` Entity Bean method call
- From a regular bean method call

The following code example demonstrates calling a bean'screate method with parameters:

```
/*This  code snippet is from a simple MessageBoard bean thatstores strings sent to it for retrieval
at a later date.*/
...
/*Call create with \"This is a test\"to initialize the EJB*/
   call = new Call();
   call.setTargetObjectURI("urn:messageboard");

/*Note, you can explicitly call a create. Parameters for the bean's create can be passed  like
parameters to any SOAP RPC call.*/
  call.setMethodName("create");
  call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
  params = new Vector();
  params.addElement(new Parameter("msg", String.class, "This is a test", null));
  call.setParams(params);

   System.out.println("Calling create with \"This is a test\"");
  resp = call.invoke(url, "");

/*Now use the same instance of the bean that you just 'created' and initialized.Obtain the reference
from the response object  through the method getFullTargetObjectURI()*/
  ejbKeyURI = resp.getFullTargetObjectURI();

/*Subsequent calls to this bean can now be made by using theobtained ejb key.*/
/*Call getMessage using the handle from the create*/
  call = new Call();
  call.setFullTargetObjectURI(ejbKeyURI);
  call.setMethodName("getMessage");
  call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
  System.out.println("Calling getMessage:");
  resp = call.invoke(url, "");
...
```

4.8.1.2.3: Deploying a programming artifact as a SOAP accessible Web service

Complete these steps to deploy a SOAP accessible Web service in WebSphere Application Server:

1. **Create or locate the software resourceto be exposed as a service**

   To deploy a service,create a programming artifact, one of the supported types,or locate an existing piece of code of the supported type.

2. **Assemble an Enterprise Archive (EAR)file**

   Package the code artifactinto an Enterprise Archive (EAR). This stepis a deployment packaging requirement of WebSphere Application Server. Use the Application Assembly Tool (AAT)to package the artifact. See article Application Assembly Toolfor information on using the tool.

3. **Create the Apache SOAP deploymentdescriptor for the desired service**

   In order to deploy an artifact as a SOAPservice, create a Apache SOAP deploymentdescriptor that describes the service you are creating.This step exposes the programming artifact as a "service." The descriptor describes and defines the parts of the code that will be invoked with the SOAP calls.

   The information containedin the deployment descriptor varies, depending on the type of artifact you are exposing.For example, the following deployment descriptor might be used with the *StockQuoteSample*:

   ```
   <isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"             id="urn:service-urn"
   [type="message"]>             <isd:provider type="java"             scope="Request | Session |
   Application"             methods="exposed-methods">             <isd:java class="implementing-class"
   [static="true|false"]/>             </isd:provider>
     <isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>
   </isd:service>
   ```

   View the Apache SOAP deployment descriptor documentationfor more information.

4. **Execute the `SoapEarEnabler` tool toenable your Web service**

   As mentioned above, your code artifact must first be packagedinto an Enterprise Archive (EAR). Next, using thedeployment descriptor as input, add thenecessary pieces to the EAR file to enable the artifact as a Web service. To facilitate this process,use the Java based tool called `SoapEarEnabler`.Depending on whether you secure the Web service, this tool will add two Web modules: `soap.war` and `soap-sec.war` to the EAR file.These Web modules include the SOAP deployment descriptors plus the necessary parts to deploy the service into the WebSphere Application Server runtime.

   > The service does not become available until the soap-enabled EAR file is installed, and the server is restarted.

   View the SoapEarEnabler tool documentationfor more information on `SoapEarEnabler`.

5. **Install the service-enabled EAR file**

   Take the modified EARfile, created in the previous step,and install it in WebSphere Application Server.

   View article Installing applications with the application installer command linefor information on installing EAR files.

6. **Update the Web server plugin configuration**

   Run the `GenPluginCfg.bat` file on Windows NT or the `GenPluginCfg.sh` script on UNIX to regenerate the plugin configuration.

7. **Restart the application server**

See the related information links for an enablement scenario.

4.8.2: Apache SOAP deployment descriptors

Apache SOAP utilizes XML documents called"deployment descriptors" to provide the SOAP runtime with information on client services.

Deployment descriptors provide an array ofinformation such as the:

- Service's URN (Uniform Resource Name)(which is used to route the request whenit arrives)
- Method and class details, ifthe service is being provided by a Java class
- User ID and password information, if the serviceprovider must connect to a database

The contents of the deployment descriptor vary,depending on the type of artifact that isbeing exposed using SOAP.

4.8.2.1: SOAP deployment descriptors in WebSphere Application Server

This article describes the different types of deploymentdescriptors that can be used in WebSphere Application Server.Deployment descriptors for each of the soap samples are included in the `soapsamples.ear` file in the `ServerSamplesCode` directory(for example, `<product_installation>/installedApps/soapsamples.ear/ServerSampleCode/src/addressbook/DeploymentDescriptor`)

Standard Java class deployment descriptor

A deployment descriptor which exposes a servicethat is implemented with a standard Javaclass (including a normal java bean) lookslike this example:

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"            id="urn:service-urn"
[type="message"]>            <isd:provider type="java"             scope="Request | Session |
Application"           methods="exposed-methods">           <isd:java class="implementing-class"
[static="true|false"]/>            </isd:provider>
  <isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>
</isd:service>
```

where:

- ***service-urn*** is the URN that you give to a service.(All services deployed within a single EARfile must have URNs that are unique withinthat EAR file.)
- ***exposed-methods*** is a list of methods, separated by spaces,which are being exposed
- ***implementing-class*** is a fully qualified class name (that is, a *packagename.classname*)that provides the methods that are beingexposed.

On the <service> element,there is an optional attribute called ***type*** which is set to the value "message"if the service is document-oriented insteadof RPC-invoked.

On the <java>element, there is an optional attribute called***static***, which may be set to either "true"or "false", depending on whetherthe methods are exposed or not exposed. If exposed, this attribute indicates whetherthe method is static or not static.

On the <provider> element, there isa ***scope*** attribute which indicates the lifetime ofthe instantiation of the implementing class.

- "Request" indicates the objectis removed after the request completes.
- "Session" indicates the objectlasts for the current lifetime of theHTTP session.
- "Application" indicates the object lasts until the servlet thatis servicing the requests, is terminated.

EJB deployment descriptor

A deployment descriptor that exposes a servicewhich is implemented with an Enterprise JavaBean looks like this next example:

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"            id="urn:service-urn">
<isd:provider type="provider-class"            scope="Application"
methods="exposed-methods">           <isd:option key="JNDIName" value="jndi-name"/>
<isd:option key="FullHomeInterfaceName" value="home-name" />           </isd:provider>
<isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>
</isd:service>
```

> The default values for the iiop URL and context provider keys are:
>
> ```
> <isd:option key="ContextProviderURL" value="iiop://localhost:900" />    <isd:option
> key="FullContextFactoryName" value="com.ibm.websphere.naming.WsnInitialContextFactory" />
> ```
>
> To use your own values, you must specify:
>
> ```
> <isd:option key="ContextProviderURL" value="<URL to the JNDI provider>" />    <isd:option
> key="FullContextFactoryName" value="<Context factory full class name>" />
> ```

A description of the keys and variables follows:

- ***service-urn*** and ***exposed-methods*** have the same meaning as in the standardJava class deployment descriptor
- ***provider-class*** is one of the following depending on the implementation of the bean:

Provider class	Bean implementation
com.ibm.soap.providers.WASStatelessEJBProvider	stateless session bean
com.ibm.soap.providers.WASStatefulEJBProvider	stateful session bean
com.ibm.soap.providers.WASEntityEJBProvider	entity bean

- ***jndi-name*** is the registered JNDI name of the EJB
- ***home-name*** is the fully qualified class name of theEJB's home.

Bean Scripting Framework (BSF) script deployment descriptor

A deployment descriptor that exposes a servicewhich is implemented with a BSF script lookslike the following example:

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"            id="urn:service-urn">
<isd:provider type="script"            scope="Request | Session | Application"
methods="exposed-methods">           <isd:script language="language-name"
[source="source-filename"]>[script-body]           </isd:script>           </isd:provider>
```

```
<isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>
</isd:service>
```

where:

- ***service-urn***, ***exposed-methods***, and *scope* have the same meaning as in the standardJava class deployment descriptor
- ***language-name*** is the name of the BSF-supported language that isused to write the script.

The deploymentdescriptor must also have a ***source*** attribute on the <script> element,or a ***script-body*** attribute. The ***script-body*** attribute contains the actual script that is used to providethe service. If the deployment descriptorhas the ***source*** attribute, then ***source-filename*** refers to the file which contains the serviceimplementation.

DB2 stored procedure deployment descriptor

A deployment descriptor which exposes oneor more DB2 stored procedures as a serviceslooks like the following example:

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"           id="urn:service-urn">
<isd:provider type="com.ibm.soap.providers.WASDB2SPProvider"           scope="Application"
methods="* | exposed-methods">           <isd:option key="userID" value="db-userid"/>
<isd:option key="password" value="db-password"/>           [<isd:option
key="fullContextFactoryName" value="context-factory"/>           <isd:option key="datasourceJNDI"
value="jndi-name"/>]           [<isd:option key="dbDriver" value="db-driver"/>
<isd:option key="dbURL" value="db-url"/>]           </isd:provider>
<isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>
</isd:service>
```

where:

- ***service-urn*** and ***exposed-methods*** have the same meaning as in the standardJava class deployment descriptor.
- ***db-userid*** is a valid user ID used to access the databasewhere the stored procedures reside.
- ***db-password*** is a valid password for the specified user ID

  The ***db-userid*** and ***db-password***entries are optional. These entries can be set in the datasource.In WebSphere Application Server, the preferred way for administering the ***db-userid*** and ***db-password*** entriesis with a datasource. Changing the user ID and password is easier when the information is located in a datasource rather than in a separate deployment descriptor file. See article DB2 Stored procedure sample setupfor more information.
- ***context-factory*** is the name of the context factoryused to access the database
- ***jndi-name*** is the datasource used to accessthe database
- ***db-driver*** is the database driver usedto access the database.

  A *db-driver* is not required if a datasource JNDI name is specified.
- ***db-url*** is the URL that specifies the database to access

The ***methods*** attribute on the <provider> elementcan contain a list of space separatedprocedure names to expose, or an "*" (asterisk).An asterisk indicates all available stored procedures shouldbe exposed.

See the related topics section for links to an enablement scenario.

4.8.3: Quick reference of Web services resources

Use the following table to link directly to Web services descriptions, and additional resources.

Click on any heading in the *Topic* category for a description of that topic.

Click on any heading in the *Resources* category for links to external sites that provide sample scenarios, toolkits, tutorials, and additional information.

Reference the *Related topics* section for links to the SOAP EAR enabler tool and to a Web servicesenablement tutorial.

Topic	Resources
Web services overview	• Web services topics and development environment • Web services wizard • Web services toolkit • Web services tutorial
• SOAP overview • SOAP support in WebSphere Application Server • SOAP samples • Building a SOAP client • Deploying a programming artifact as a SOAP accessible Web service	Apache SOAP implementation
• UDDI overview • IBM's UDDI test registry	IBM's UDDI registry implementation
• UDDI4J overview • UDDI4J support in WebSphere Application Server • UDDI4J samples • IBM's Javadoc	UDDI4J topics
WSDL overview	WSDL topics

See the Related topics section for links to an enablement tutorial.

4.8.4: Securing SOAP services

Since the SOAP specification left security issuesopen, several proposals evolved to bridge the security gaps. Recentlythe SOAP Security Extension [SOAP-SEC] was published as a W3C Note, specifically addressing the XML Digital Signature.

The SOAP security extension, included withWebSphere Application Server Version 4.0, is a security architecturebased on the SOAP security specification,and widely-accepted security technologiessuch as Secure Sockets Layer or SSL.

There are three options for security when using HTTP as the transport protocol.

- HTTP basic authentication
- SSL (HTTPS)
- SOAP signature

Application developers are free to combine these security options according to their security requirements.The following scenarios describe the implementation of the security options.

HTTP basic authentication

Many applications require users toprovide identifying information.You cannot provide access control for individual services. You can onlyprovide access control for the router servlets (as for example the rpcrouter servlet URI). If auser can get to a servlet, he can access any of the Web services served through the servlet.Therefore, if you have a set of "secure" services and "unprotected" services, you have to partition them differently so that "secure" servicesare accessed through an URI that is secured (for example, `/secureRPCRouter`) andthe unprotected services are open for everyone to access (for example, `/uprotectedRCPRouter`).

Using the ApplicationAssembly tool, you can set authorization levels by assigning roles to HTTP methods and byassigning users to roles. You can then *authenticate* users, verifying they are authorized to view specific information. There are many ways to prompt users for authentication data.See articles Overview: Using programmatic and custom login andThe WebSphere authorization modelfor more information on different authentication methods, and on role-based authorization scenarios.

SOAP on SSL with HTTP basic authentication

To make a request over HTTPS, using the SSL support of Apache SOAP, you needa separate Java Secure Socket Extension (JSSE) provider.

WebSphere Application Server includesthe `ibmjsse.jar` in the JDK extensions.

The "SOAP on SSL" scenario is useful for many *business-to-business* (B2B) applications because:

- The data in transit is protected from eavesdropping or forgery by SSL.
- The client identity is authenticated through userid and password, which are encrypted by the SSL transport.

For example, if an inventory application is configured as a Web service, the service provider has the following two SOAP service entries:

- `https://foo.com/inventory/inquiry`
- `https://foo.com/inventory/update`

Each SOAP service entry should be deployedas a separate enterprise application (EAR)because each service has a different access controlpolicy, which is: anyone can inquire about the inventorybut only the inventory clerks can updatethe contents.

The SOAP enablement model limits you to one context root for theunsecured services and another for the secured services. In this example, you want to make the inquiry service unsecured and the update service secured. Ifyou want different levels of security for a "secured" service, then you mustdeploy the entries in the "secured" service as separate EAR files.

Do the following to enable the "SOAP on SSL" scenario:

- Configure the web server (httpd.conf) so that it only allows SSL access to these servlets.
- Configure the security role for the `RPCRouterServlet` in the inquiry services EAR so that the `RPCRouterServlet` for the 'inquiry' service is accessible by everyone, while the `RPCRouterServlet` for the 'update' service requires authentication based on the HTTP basic authentication (userid/pasword).

In this case, the 'update' application does not know the identity of the requester; it only knows that access is granted. In other words, the "update" application is not concerned with the identity ofthe user because it knows WebSphere Application Server is ensuring that only authenticated usershave access.

SOAP on SSL with SOAP Signature

Applications might need non-repudiable proof of exchanged messages.One example is a web service that accepts part orders.The business partners establish a form of trust relationship based on public keys. This can be done using the public key infrastructure (PKI) through a third party certificate authority (CA), or by exchanging public keys with a secure channel. The following service is deployed with a *signature verification* function:

```
https://foo.com/partorder
```

Configure *signature verification* with the following information:

- Scope of signature (indicates the portion of the SOAP envelope that must be authenticated. The default is the content of *SOAP-ENV:Body*).
- Trusted keys or trusted root keys.
- Default key to verify signature if no KeyInfo is specified.
- Other policies regarding signature validation.
- Behavior when signature verification fails.
- Additional requirements on signature (as for example, specific requirements on hash/C14N algorithms to be used, timestamp validity, and so forth).

If the signature is missing or if *signatureverification* fails, the signature verification function canbe configured so that the servlet returnsa SOAP fault.

To send part orders to the `https://foo.com/partorder` service,the service requester should sign his SOAP messages with a signature component.The signature component is initialized using two templates:

1. <ds:SignedInfo> template
2. <ds:KeyInfo> template

The `<ds:SignedInfo>` template controls the following:

- What parts of the SOAP envelope must be signed
- What algorithms (canonicalization, transformation, digest, sign) should be used

The `<ds:KeyInfo>` template controls the following:

- Whether or not to include the entire certificate chain in `<ds:KeyInfo>`
- Decision to include only certificate and serial number
- Public key value
- Decision to provide no key information (so that the default key must be used for verification).

You can combine the service request with HTTP basic authentication, if necessary.

4.8.4.1: Running the security samples

The process for running the SOAP signed samples is identical to the process forrunning the non-signed samples. The `soapsamples.ear` must be installed, and the server must be started before these samples are invoked.

See article SOAP samples for informationon installing the SOAP samples.

SOAP Signature

The client samples are included in the `soapsamples.ear` file.Do the following to locate and execute the samples:

1. Change your directory (cd) to
   `product_installation_root``/installedApps/soapsamples.ear/ClientCode`

   A set of batch files or script files (on UNIX platforms) have beenincluded to facilitate running the client samples. These batch or script files are located inthe `nt_bat` subdirectory on Windows NT, or in the `unix_scripts`subdirectory on UNIX platforms. These scripts setthe classpath and supply parameters.
2. Invoke the samples using the following scripts:

   ```
   DSigAddressSample localhost "c:\WebSphere\AppServer\installedApps\soapsamples.ear""John B. Good"
   DSigMessageSample localhost "c:\WebSphere\AppServer\installedApps\soapsamples.ear"..\data\msg1.xml
   ```

   If you run the script with **no** arguments, as for example *DSigAddressSample*, you will be provided with help on how to use the sample, and you will receive a description of the command line arguments that the script requires.
3. View the output.

   For each sample, at the server, you should see that the signature ofthe request is validated. At the client,you should see that the signature of the responseis validated.

   The validation results for both theclient and server are logged to thefollowing files that are created in the*`product_installation_root`*`/InstalledApps/soapsamples.ear/soapsec.war/logs` directory
   - `SOAPVHH-all-cl.log`
   - `SOAPVHH-fail-cl.log`
   - `SOAPVHH-all-sv.log`
   - `SOAPVHH-fail-sv.log`

Soap signature with SSL connection

Ensuring that a connection is over SSL isnot specific to Web services. You must configurethe Web server to ensure that the clientto Web server connection is over SSL. You must also configure WebSphere Application Server to ensure thatthe Web server to WebSphere Application Server connection isover SSL.

Article Configuring SSL in WebSphere Application Serverdiscusses how to configure SSL in WebSphere.See your Web server documentation for information on configuring the SSL server.

For testing purposes, sample client and server keystoredatabases are shipped with the SOAP samples.You must use the IBM Key ManagementTool to extract the certificates located in files:

- test
- keystore
- databases

Import the certificatesinto your key databases. See article, Tools for managing certificates and keysfor more information on the IBM Key Management tool.

The test keystores are described in articleKeystore files.

Export the client certificates from the testkeystore file

Perform the following steps to export the client cerificates:

1. Invoke the Key Management Tool (IKeyman)
2. From the file menu, select open
3. Change directory (CD) to
   `product_installation_root``/InstalledApps/soapsamples.ear/soapsec.war/key/`
4. Select the SOAPClient keystore file.
   (The keystore password is "client".)
5. Change the key database content type to "Signer Certificates".
6. Highlight the **soapca** certificate.

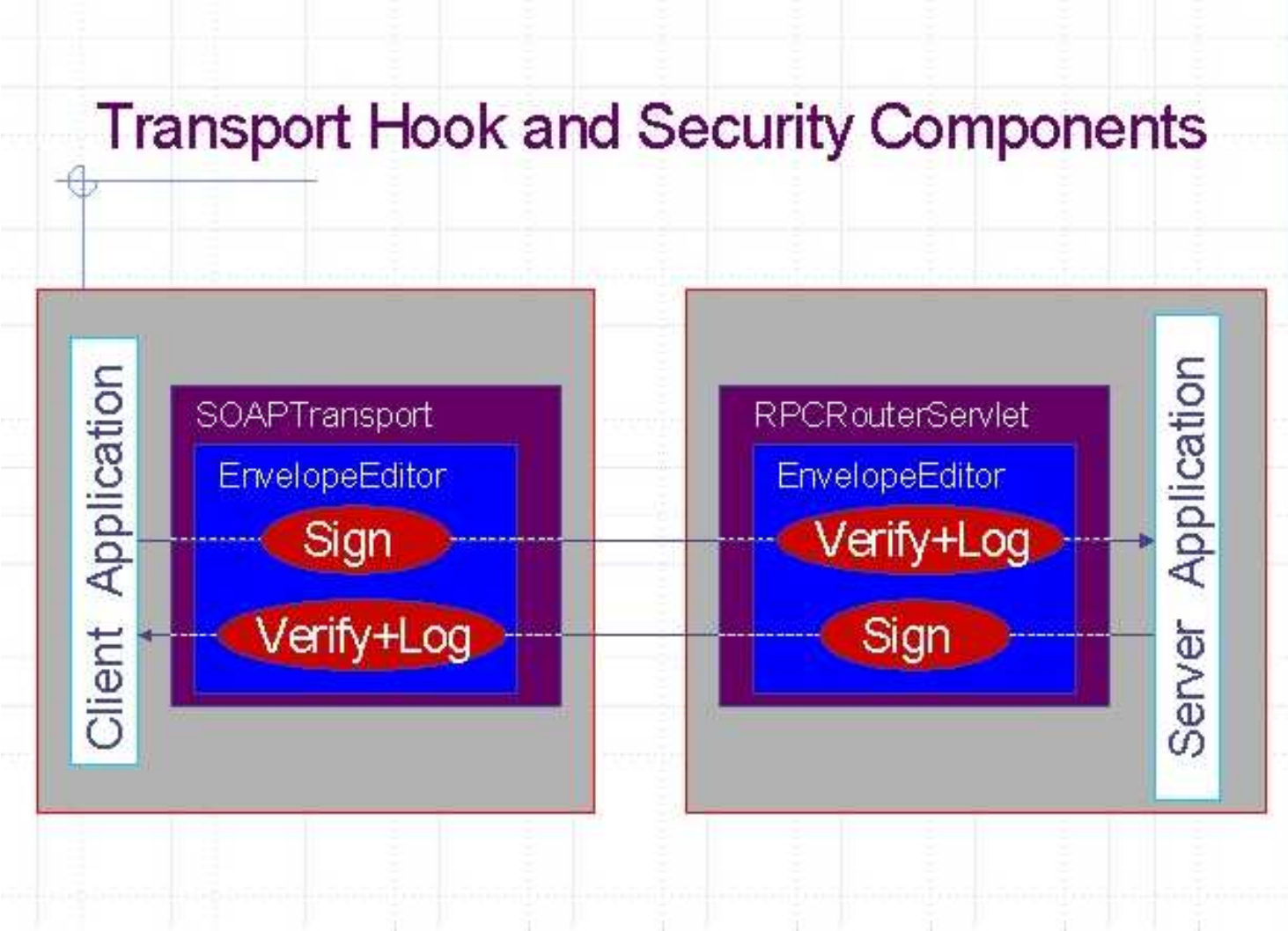
7. Click the **Export** button.
8. Change the exported file name to "soapca.arm".
9. Highlight the "intca1" certificate
10. Click the **Export** button.
11. Change the exported file name to "intca1.arm".

Import the certificates into the web serverkey database

1. Invoke the Key Management Tool (IKeyman)
2. From the file menu, select open (or new if you are creating a new keystore)
3. Change directory (CD) to the directory where the keystore file is located.
4. Select the file.
5. For Signer Certificates, add the "intca1.arm" and the "soapca.arm" you exported in the previous section.
6. For Personal Certificates, click **Import**.
7. Specify a key type of **PKCS12**
8. Browse the *sslserver.p12* file located in:
   product_installation_root`/InstalledApps/soapsamples.ear/soapsec.war/key/`
9. Click OK.
10. Enter "server" when prompted for a password.
11. Select "sslserver" from the key list and press OK.
12. Save the updated keystore file

4.8.4.2: SOAP signature components

An overview of the SOAP signature architecture is illustrated in the figure below.



Using the SOAP transport hook, you can plug-in the security components:

- signer
- verifier with logging capability

The transport hook is called the *EnvelopeEditor*. A *PluggableEnvelopeEditor* is also provided, which allows you to plugin your security components. As illustrated, the *EnvelopeEditor* is encapsulated in the *SOAPTransport* on the client side.On the server side, *EnvelopeEditor* is encapsulated in *RPC/MessageRouterServlet*.This means the same components can be used on either side.

When a client application sends a request, the request is signed and transmitted to the server. At the server side, the request is verified and delivered to a server application or, in the case of a RPC, to a Java object.The response is processed in the same manner.The verifier component also has a logging function to log the verified messages in a file.Signatures and verifier components are configurable. You can specify encryption, digest message algorithm, certificate path policy, and other security technologies.

Signature Components

There are two signature components:

- Signature Header Handler
- Verification Header Handler

Signature Header Handler (SHH)

The Signature Header Handler is a XML-based configuration file, which enables:

- Template for <SignedInfo> (for customizing references, sign/hash algorithms, C14N algorithms, optional timestamp)
- Template for <KeyInfo> (for customizing the public key such as X.509 certificate)

Verification Header Handler (VHH)

The Verification Header Handler is a XML-based configuration file, which enables:

- Configurable policy (required scope of signature, trusted root, certstore, certpathchecker) (more sophisticated policy such as timestamp validation may not be included in 2/15 deliverable)
- Exit for Logging (additional application-specific verification) A reference implementation of logging component is also provided.

The digital signature configuration can be changed by editing the configuration for the following components:

- Envelope Editor
- Signature Component
- Verification Component

SOAP Security-related Files

The following table provides an inventory of the SOAP security elements contained in the SOAP security samples module (`soapsec.war`).a quick reference for SOAP security topics.

Path	Contents	Description
/installedApps/soapsamples.ear/soapsec.war	Web-INF, conf, key, log, etc.	Home of the soap security servlets
/installedApps/soapsamples.ear/soapsec.war/WEB-INF	`web.xml`	Servlet configuration file for SOAP security samples
/installedApps/soapsamples.ear/soapsec.war/conf	config files	Configuration files for envelope editors and signature components
/installedApps/soapsamples.ear/soapsec.war/key	`SOAPclient` `SOAPserver`	See article Keystore files for more information.
/installedApps/soapsamples.ear/soapsec.war/logs	Log files	Logs generated during security exchange
/installedApps/soapsamples.ear/ServerSamplesCode/src/<service_name>	`server side samples`	Source for both the non-secure and secure samples
/installedApps/soapsamples.ear/ClientCode/nt_bat	`scripts to run client samples`	Batch files for invoking the client side samples to interact with the server-side services
/installedApps/soapsamples.ear/ClientCode/unix_scripts	scripts to run client samples	Batch files for invoking the client side samples to interact with the server-side services
/installedApps/soapsamples.ear/ClientCode/data	data files used by samples	
/installedApps/soapsamples.ear/ClientCode/src	client side samples source	
/lib	soap.jar, soap-sec.jar, ws-soap-ext.jar	Location of all jar files

Related Documents

- Simple Object Access Protocol (SOAP) 1.1 - W3C NOTE.
- SOAP Security Extensions: Digital Signature - W3C NOTE.
- XML-Signature Syntax and Processing - W3C CR.
- XML Security Suite - XML digital signature, encryption, access control.

4.8.4.2.1: Keystore files for testing purposes

Two keystore files, (SOAPserver and SOAPclient), are available for testing purposes.These files are located in directory:

product_installation_root`/installedApps/soapsamples.ear/soapsec.war/key`

This article describes the certificates that are stored in these two keystore files.

File name	Store password	Description
SOAPserver	server	This keystore is used by a service provider.
SOAPclient	client	This keystore is used by a service requester.

Common Certificate Authority certificates

The following three certificates are commonly stored in both keystore files.

Alias	Issuer	Description
soapca	soapca itself	The certificate of the root Certificate Authority (CA) used for testing purposes.
intca1	soapca	The certificate of the CA to issue SSL-related certificates.
intca2	soapca	The certificate of the CA to issue SOAP-DSIG-related certificates.

Certificates for service providers

The following two certificates are stored in the `SOAPserver` keystore.

Alias	Issuer	Description
sslserver	intca1	This is the certificate of the SSL server. This is also stored in the SOAPclient keystore as a *trusted* certificate. The PKCS12 file including the corresponding private key for this certificate is sslserver.p12.
soapprovider	intca2	This certificate might be used by a service provider to digitally sign its response message. The key password is "server".

Certificates for service requesters

The following three certificates are stored in the `SOAPclient` keystore.

Alias	Issuer	Description
sslclient	intca1	This certificate might be used for the SSL client authentication. The key password is "client".
sslserver	intca1	This is the certificate of the *trusted* SSL server and the same as the one stored in the SOAPserver keystore. The PKCS12 file, including the corresponding private key for this certificate, is *sslserver.p12*.
soaprequester	intca2	This certificate might be used by a service requester to digitally sign its request message. The key password is "client".

- IBM HTTP Server documentation on configuring SSL
- Tools for managing certificates and keys

4.8.4.2.2: Envelope Editor

The Envelope Editor is a component thatcan be plugged into the Apache SOAP transports.At the server side, it is embedded into theRPC and MessageRouterServlets. At the clientside, it is embedded in the FilterTransport,which implements the SOAPTransport interface.WebSphere Application Server provides a *PluggableEnvelopeEditor*,which can be usedto plug-in some editing componentssuch as signature and verification.

Enabling Envelope Editor

At the client side, the configuration ofthe *eEnvelope eEditor* is explicitly programmed.On the server side, the transport hook isenabled automatically in the `soapsec.war` filewhen you add the "init" param to the RPC and Messagerouter servlets for the `EnvelopeEditorFactory`.This entry in the `web.xml` for the `soapsec.war` fileis added automatically when you "soap enable"an application and indicate the service is secure.

Description of the factory class to instantiate Envelope Editors

A factory class creates *Envelope Editors* at runtime. The factory class is called `DSigFactory`. The `DSigFactory` class consumes an editor configuration file, and creates an instance of *Envelope Editor*. The factory class and the configuration file are specified in:

product_installation_root`\installedApps\`*ear_file_name*`\soapsec.war\WEB-INF\web.xml`

The factory class is describedunder the <servlet id="Servlet_1">and <servlet id="Servlet_2">elements:

```
       <display-name>Apache-SOAP-SEC</display-name>       <description>SOAP Security Enablement
WAR</description>       <servlet id="Servlet_1">         <servlet-name>rpcrouter</servlet-name>
<display-name>Apache-SOAP Secure RPC Router</display-name>         <description>no
description</description>
<servlet-class>com.ibm.soap.server.http.WASRPCRouterServlet</servlet-class>         <init-param
id="InitParam_1">            <param-name>faultListener</param-name>
<param-value>org.apache.soap.server.DOMFaultListener</param-value>         </init-param>
<init-param id="InitParam_2">            <param-name>EnvelopeEditorFactory</param-name>
<param-value>com.ibm.soap.dsig.dsigfactory.DSigFactory</param-value>         </init-param>
<init-param id="InitParam_3">            <param-name>SOAPEvnelopeEditorConfigFilePath</param-name>
<param-value>conf/sv-editor-config.xml</param-value>         </init-param>       </servlet>
<servlet id="Servlet_2">         <servlet-name>messagerouter</servlet-name>
<display-name>Apache-SOAP Secure Message Router</display-name>
<servlet-class>com.ibm.soap.server.http.WASMessageRouterServlet</servlet-class>         <init-param
id="InitParam_5">            <param-name>faultListener</param-name>
<param-value>org.apache.soap.server.DOMFaultListener</param-value>         </init-param>
<init-param id="InitParam_6">            <param-name>EnvelopeEditorFactory</param-name>
<param-value>com.ibm.soap.dsig.dsigfactory.DSigFactory</param-value>         </init-param>
<init-param id="InitParam_7">            <param-name>SOAPEnvelopeEditorConfigFilePath</param-name>
<param-value>conf/sv-editor-config.xml</param-value>         </init-param>      </servlet>
```

`EnvelopeEditorFactory` is a factory class.`SOAPEnvelopeEditorConfigFilePath` is a configurationfile for Envelope Editor.

Configuration file of Envelope Editor

The configuration file, `sv-editor-config.xml` is located in:

product_install_root`\installedApps\<`*ear_file_name*`>\soapsec.war\conf\sv-editor-config.xml`

Under the *SOAPEnvelopeEditorConfig* element, thereare two optional elements:

- incoming
- outgoing

The *incoming* and *incoming* element definitions look like the following example:

```
  <incoming class="com.ibm.xml.soap.security.dsig.SOAPVerifier">    <init-param>
<param-name>filename</param-name>      <param-value>conf/sv-ver-config.xml</param-value>
</init-param>  </incoming>  <outgoing class="com.ibm.xml.soap.security.dsig.SOAPSigner">
<init-param>      <param-name>filename</param-name>
<param-value>conf/sv-sig-config.xml</param-value>    </init-param>  </outgoing>
```

The incoming element specifies a class which "edits" incoming messages, and a configuration file for the editing class. The outgoing element specifies a class for outgoing message and a configuration file.

Changing the configuration

You do not have a digital signature for response messages if you remove the outgoing element from

product_installation_root`\installedApps\<`*ear_file_name*`>\soapsec.war\conf\sv-editor-config.xml`

and remove the incoming element from

product_installation _root`\installedApps\<`*ear_file_name*`>\soapsec.war\conf\cl-editor-config.xml`

4.8.4.2.3: Signature Header Handler

The Signature Header Handler (SHH) insertsa digital signature header into a SOAP envelope.You can customize the SHH configuration with a configuration file. For example, you canspecify a signing policy and the key storefile.

There are two signature configurationfiles:

product_installation_root\installedApps\<*ear_file_name*>\soapsec\conf\sv-sign-config.xml
product_installation_root\installedApps\<*ear_file_name*>\soapsec\conf\cl-sign-config.xml

The `soapsamples.ear` file contains samples of these configuration files.

An explanation of each configuration element in the Signature Header follows:

- **KeyStore**

  The KeyStore element specifies a keystore file that holds the signingkey. In the following example, the attribute "type" indicates a keystore type, and "jks" indicates Java Key Store. "path" is a keystorefile, and "storepass" is its store password.

  ```
  <KeyStore    type="jks"    path="key\SOAPserver"    storepass="server"  />
  ```

  The Key Management tool (iKeyman) can beused to create a keystore file.

- **Policy**

  The PublicKey element specifies the information that should be included inthe <ds:KeyInfo> element. With the current implementation, you must either include the complete certificate chain, oromit the <ds:KeyInfo> When <ds:KeyInfo> is ommitted, the recipientmust know the default key to verify the signature.

- **Template**

  The contents of the Template element specify all the details relatedto XML Signature, including signature algorithms, digest algorithms,canonicalization algorithms, transform algorithms, the portion of theSOAP envelope to be signed, and so on.

- **Object**

  The template can also have Object element(s) for additionalauthentication information, such as a timestamp.

- **ValueOfTimestamp**

  This SHH understands one special element type, ValueOfTimestamp, whichis replaced with a current time and date before being inserted intothe signature.

4.8.4.2.4: Verification Header Handler

The Verification Header Handler (VHH) validatesa digital signaure header in a SOAP envelope.Its configuration can be customized usinga configuration file where you specify the following:

- a verification policy
- the certificate path
- logging files to record verified messages

There are two signature configuration files:

```
product_installation_root\installedApps\<ear_file_name>\soapsec.war\conf\sv-ver-config.xml
product_installation_root\installedApps\<ear_file_name>\soapsec.war\conf\cl-ver-config.xml
```

Samples of these configuration files areprovided in the `soapsamples.ear` file.

An explanation of each configuration element in the Verification Header follows:

- ## AllowedAlgorithms

  All the algorithms supported by this VHH must be listed in this element. Algorithms other than these cannot be used in `SOAP-SEC:Signature` header. The current implementation supports all required algorithms in the XML Signature specification, except for SHA1-MAC.

- ## RequiredAuthenticatedParts

  This section specifies what parts of SOAP message need to be authenticated through the `SOAP-SEC:Signature` header. Currently two values are supported for the part attribute:

  1. When part="root," the whole evelope must be signed through the enveloped-signature transform.
  2. When part="body," the *SOAP-ENV:Body* element in the SOAP envelope must be referenced by one of the reference elements in the signature.

  Part="" allows an attachment to be specified.

  If the specified parts are not authenticated through the signature header entry, verification fails.

- ## DefaultVerificationKeys

  When `KeyInfo` is missing in the signature, the content of this elementis used as a part of the signature. When communicatingparties know the identity of each other, the default `KeyInfo` can beused to reduce the communication data volume.

- ## Log

  Specifies the logging behavior. The following versions of logging exist:

  - When target="all," all verification attempts are logged.
  - When target="success," only successful verification are logged.
  - When target="fail," only unsuccessful verification are logged.

  Multiple LogFile elements can be specified.

  The following example illustrates how to specify logging:

  ```
  <Log>   <SOAPDSigLogger      class="com.ibm.xml.soap.security.dsig.SOAPDSigLoggerImpl">
  <LogFile target="all" path="SOAPVHH-all.log" append="yes"/>   </SOAPDSigLogger>   <SOAPDSigLogger
  class="com.ibm.xml.soap.security.dsig.SOAPDSigLoggerImpl">     <LogFile target="fail"
  path="SOAPVHH-fail.log" append="yes"/>   </SOAPDSigLogger> </Log>
  ```

- ## PKIXParameters

  Currently VHH supports X.509/PKIX certificates only (no HMAC, no PGP, and so forth). The policies for PKIX certificate verification are specified in this element. This is a straightforward mapping of Java CertPath API. Not all of the entries are meaningful in this initial release.

  Current implementation only allows the useof keystore as the means of specifying trustedroot.

6.6.0.14: XML-SOAP Admin tool

Use the SOAPEarEnabler toolto add administrative interfaces to your EAR files. You can then use the XML-SOAP Admin tool with these EAR files.

WebSphere Application Server provides a modified version of theApache SOAP XML-Admin interface (or XML-SOAP Admin tool) for each SOAP-enabled EAR file.This interface allows you to do the followingfor each context root:

- List configured services, showing active and stopped services
- Stop a service
- Start a service
- View the Apache Soap deployment descriptor for a service

Accessing the XML-SOAP Admin tool

Access the XML-SOAP Admin tool through a Web browser by specifying:

```
http://localhost/<contextroot>/admin/index.html
```

> **Note:** The *context root*, in this example, is the context specified when installing the SOAP-enabled `.ear` file.The context root for SOAP samples is `soapsamples`.

Therefore to use this interface with the SOAP samples,enter:

```
http://localhost/soapsamples/admin/index.html
```

You cannot use the XML-SOAP Admin tool to add or remove a service. Usethe SOAP Ear Enabler tool to add or removeservices. A "stopped" service is persistedacross starts and stops of the applicationserver. Therefore, if you stopa service, it will remain stopped until thenext time you use the XML-SOAP Admin toolto start it again.

You can add the XML-SOAP Admin tool interface to an enterpriseapplication when you SOAP-enable the EAR file.In interactive mode, you are asked whether you want to add the XML-SOAP Admin tool interface.Replying "yes" will add the necessaryJSP files and bindings that allow you to accessthe XML-SOAP Admin tool interface for the application.The interface is an optional addition becauseyou may not want to expose itin a production environment. Optionally,you may choose to use the application assemblytool to assign roles to the XML-SOAP Admin toolso that it is secure.

Updating an existing SOAP-enabled Enterprise Application

The Application Assembly tool is used toupdate the contents and configuration ofan enterprise application. For example, tosecure the XML-SOAP Admin tool interfacefor a particular EAR, use the application assemblytool to secure the resource. (See article Securing applicationsfor security information.) However, you cannot use the application assembly toolto add or remove a Web service.

To add or remove a service to the XML-SOAP Admin tool, startwith the original EAR file and execute the enabling process again.

> **Note:** The SOAPEar Enabler tool saves a backup copy of the EAR file.