Using SOAP Message Handlers

Part No: 821–2600
March 2011

120126@25097

Contents

Using SOAP Message Handlers

This document provides information about SOAP message handlers and instructions for implementing default and custom message handlers in Java CAPS products. It covers the following topics:

About SOAP Message Handlers

As defined in the JAX-RPC specification, message handlers provide a mechanism for adding, reading, and manipulating header blocks in SOAP messages that are sent and received by JAX-RPC clients and web service endpoints. By providing access to SOAP headers, you can import custom code to support SOAP standards that are not supported in basic versions of Java CAPS.

Although message handlers are Project-specific, they are added in Java CAPS Environments to allow easy reuse across multiple Projects. Multiple handlers can be installed or removed as needed, and they also may be chained, which allows staged execution of different types of processing modules.

Java CAPS version control is applied to all Environment sub-nodes and associated artifacts in the NetBeans IDE. You also may employ your own version control system for the handler code you import into Java CAPS.

Uses for SOAP Message Handlers

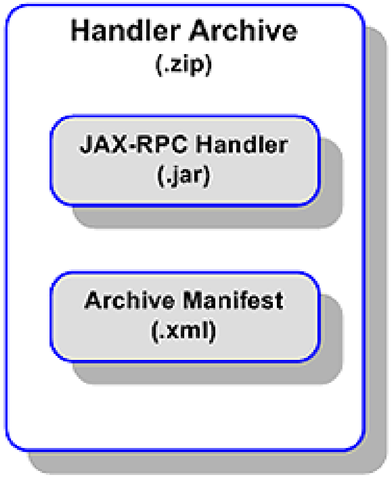
Since SOAP message handlers have access to the entire SOAP message during request and response operations, there are many use cases for them. Examples of message handler uses include:

- Authentication and authorization of the security tokens (typically packaged in the SOAP header)
- Encryption and decryption
- Logging and auditing
- Caching
- Gathering various metrics on transaction

JAX-RPC Message Handler Archive Files

JAX-RPC message handlers for use with Java CAPS are packaged in *handler archives*. A Handler archive consists of a ZIP file containing the following components:

- One or more JAX-RPC handler implementation classes and any support classes
- Any third-party JAR files on which the handler implementation depends (that are not available in the classpath in the application server runtime)
- An XML descriptor file that describes the handlers packaged in the archive



The handler implementations might be dependent on the runtime configuration, so at design-time the handlers are configured in the Java CAPS Environment. This allows the reuse of the same message handler implementations across multiple projects in the same environment.

SOAP Message Handler Chains

As defined in the JAX-RPC specification, the message handlers are ordered in a message handler chain and configured on the consumer or provider web service endpoints. This chained configuration enables the staged execution of custom SOAP message processing. When a web service request arrives at an endpoint, the `handleRequest` method for these handlers is called in the same order as they are ordered in the handler chain. When the response is sent back, the `handleResponse` method on these handlers is called in the reverse order as defined in he handler chain.

SOAP Message Handlers in Java CAPS

JAX-RPC message handlers are incorporated into Java CAPS Projects by means of a three-stage process:

1. Importing a handler archive, which can contain one or more individual handlers.
2. Creating one or more handler chains, each of which can contain one or more individual handlers in a specific order.
3. Associating the handler chains with specific SOAP/HTTP Web Service External Services.

The following topics provide instructions on how to proceed through this process.

- "Importing a JAX-RPC Handler Archive" on page 8
- "Creating a JAX-RPC Handler Chain" on page 9
- "Assigning Handler Chains to External Systems" on page 12

Note – For more information about SOAP message handlers, along with sample projects, see the Java CAPS sample site at `http://javacaps-samples.samplecode.oracle.com`. Click the JAX-WS and JAX-RPC tab for more information.

Java CAPS provides pre-installed handler implementations in the Repository. These include a logging handler and a security handler that enables ws-security support.

Importing a JAX-RPC Handler Archive

FIGURE 1 Message Handler Archive Icon

To begin, you need to have a JAX-RPC handler archive available on your system.

▼ To Import an Existing Message Handler Archive

1 **On the NetBeans IDE, click the Services tab.**

2 **Expand CAPS Environments.**

3 **Right-click the Environment to which you are importing, point to Import, and then select Message Handler Archive.**

The Import Message Handler Archive Wizard appears.

4 **Do one of the following:**

- **To import the archive provided with Java CAPS, select Import Pre-installed Archive From Repository.**
- **To import a custom archive, select Import Custom Archive From Local Directory.**

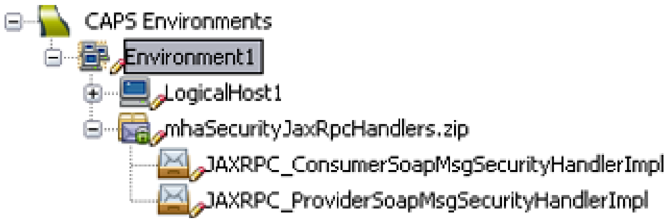
5 **Click Next.**

6 **Select the archive file you want to import, and click Next.**

7 **If the selected archive file imports correctly, the Summary page appears. Review the information provided, and then click Finish.**

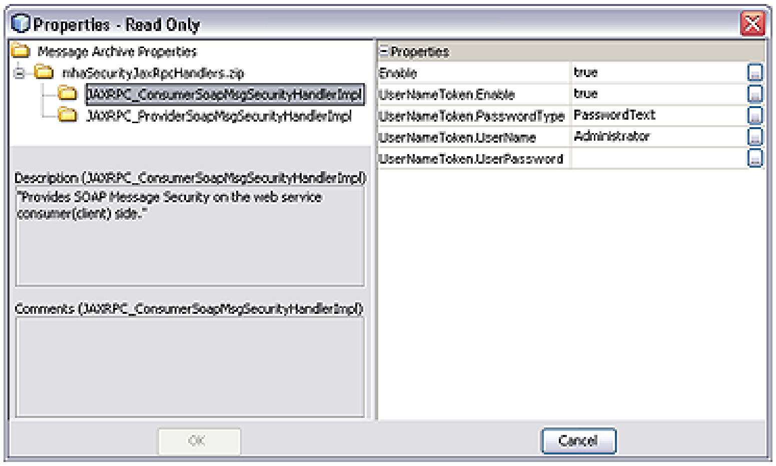
The message handler archive now appears in the Services window as a child node of the Environment, and the message handler itself appears as a child node of the archive, as shown in the following figure.

FIGURE 2 Message Handler Nodes


CAPS Environments
Environment1
LogicalHost1
mhaSecurityJaxRpcHandlers.zip
JAXRPC_ConsumerSoapMsgSecurityHandlerImpl
JAXRPC_ProviderSoapMsgSecurityHandlerImpl

To display the Message Handler Properties window, right-click the message handler in the Environment and select Properties.

FIGURE 3 Example Message Handler Properties Dialog


Properties - Read Only
Message Archive Properties
mhaSecurityJaxRpcHandlers.zip
JAXRPC_ConsumerSoapMsgSecurityHandlerImpl
JAXRPC_ProviderSoapMsgSecurityHandlerImpl
OK
Cancel

Description (JAXRPC_ConsumerSoapMsgSecurityHandlerImpl)
"Provides SOAP Message Security on the web service consumer(client) side."

Comments (JAXRPC_ConsumerSoapMsgSecurityHandlerImpl)

Properties	
Enable	true
UserNameToken.Enable	true
UserNameToken.PasswordType	PasswordText
UserNameToken.UserName	Administrator
UserNameToken.UserPassword	

8 **If the archive file did not import properly, you will see an error message on the `Import Message Handler Archive` wizard `Summary` page. You will then need to determine the reason for the failure to import.**

Creating a JAX-RPC Handler Chain

FIGURE 4 Message Handler Chain Icon

To begin, you need to have imported one or more JAX-RPC message handlers as described in "Importing a JAX-RPC Handler Archive" on page 8.

▼ To Create a New Chain of Message Handlers

1. **In the NetBeans Services window, expand CAPS Environments.**

2. **Right–click the Environment in which you want to create a handler chain, point to New, and select SOAP Message Handler Chain.**

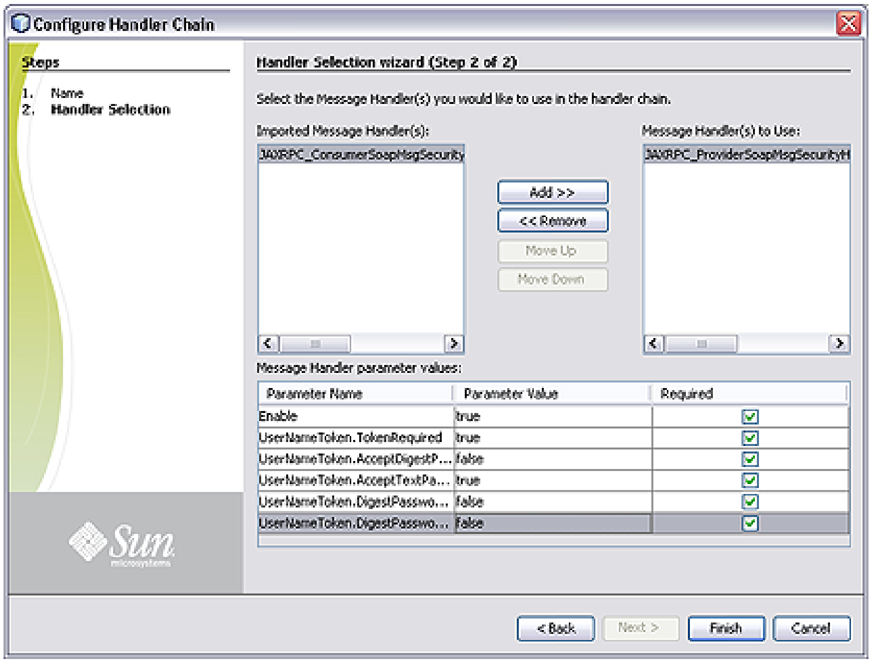
   The Configure Handler Chain wizard appears.

3. **Enter a name and a description for the handler chain.**

4. **Click Next.**

   The Handler Selection wizard appears with a list of all imported message handlers in the left box.

FIGURE 5 Configure Handler Chain Wizard: Handler Selection



Handler Selection wizard (Step 2 of 2)

Select the Message Handler(s) you would like to use in the handler chain.

Imported Message Handler(s): JAXRPC_ConsumerSoapMsgSecurity

Message Handler(s) to Use: JAXRPC_ProviderSoapMsgSecurityH

Message Handler parameter values:

Parameter Name	Parameter Value	Required
Enable	true	✓
UserNameToken.TokenRequired	true	✓
UserNameToken.AcceptDigestP...	false	✓
UserNameToken.AcceptTextPa...	true	✓
UserNameToken.DigestPasswo...	false	✓
UserNameToken.DigestPasswo...	false	✓

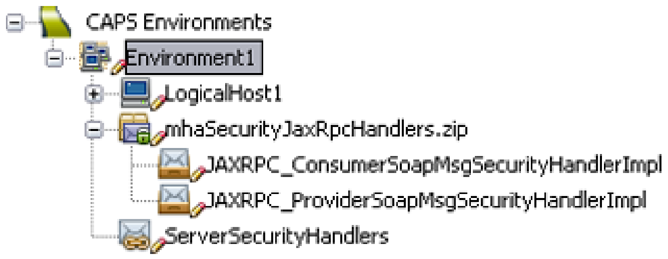
5 Select the handlers you want to include in the new chain, using the Add and Remove buttons to populate the right pane.

6 Use the Move Up and Move Down buttons to modify the order of the handlers in the chain. Handlers are applied in the sequence you specify.

Note – You can return to this wizard to modify the settings at a later time by selecting Edit in the handler chain's context menu.

The message handler chain now appears in the Services window as a child node of the Environment, as shown in the following figure.

FIGURE 6 Message Handler Nodes


CAPS Environments
Environment1
LogicalHost1
mhaSecurityJaxRpcHandlers.zip
JAXRPC_ConsumerSoapMsgSecurityHandlerImpl
JAXRPC_ProviderSoapMsgSecurityHandlerImpl
ServerSecurityHandlers

Note – You can define multiple handler chains containing different combinations of handlers, or different sequences of the same handlers.

Assigning Handler Chains to External Systems

To begin, you need to have created one or more handler chains as described in "Creating a JAX-RPC Handler Chain" on page 9

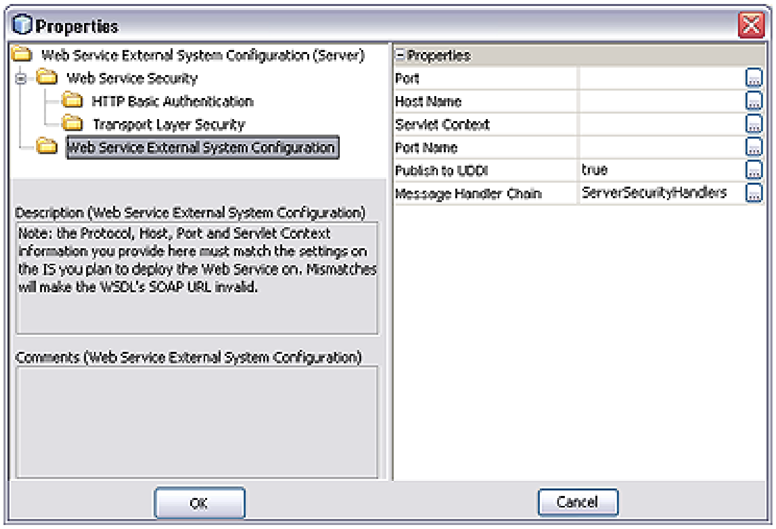
▼ To Assign the Handler Chain to a SOAP/HTTP Web Services External System

1 **In the Services window, select the SOAP/HTTP Web Service External System to which you want to assign a handler.**

2 **Right-click the External System and select Properties.**

The Properties window appears, as shown in the following figure.

FIGURE 7 Web Services External System Properties Dialog


Properties
Web Service External System Configuration (Server)
Web Service Security
HTTP Basic Authentication
Transport Layer Security
Web Service External System Configuration
OK
Cancel

Description (Web Service External System Configuration)

Note: the Protocol, Host, Port and Servlet Context information you provide here must match the settings on the IS you plan to deploy the Web Service on. Mismatches will make the WSDL's SOAP URL invalid.

Comments (Web Service External System Configuration)

Properties	
Port	
Host Name	
Servlet Context	
Port Name	
Publish to UDDI	true
Message Handler Chain	ServerSecurityHandlers

3 **Click the ellipses in the Message Handler Chain value field, and select the desired handler chain from the dialog box that appears.**

4 **Click OK on the Settings dialog box, and then click OK on the Properties window.**

Implementing Custom SOAP Message Handlers

The SOAP message handler implementation class is required to implement the `javax.xml.rpc.handler.Handler` interface as defined in the JAX-RPC specification. It should contain only the default constructor with no arguments in order to be instantiated at runtime. For example:

```
package javax.xml.rpc.handler;
   public interface Handler {
      public boolean handleRequest(javax.xml.rpc.handler.MessageContext context);
      public boolean handleResponse(javax.xml.rpc.handler.MessageContext context);
      public boolean handleFault(javax.xml.rpc.handler.MessageContext context);
      public void destroy();
      public void init(javax.xml.rpc.handler.HandlerInfo config);
      public javax.xml.namespace.QName[] getHeaders();
   }
```

To make it easier to implement these handlers, the JAX-RPC specification also defines an abstract helper class, javax.xml.rpc.handler.GenericHandler, which has default implementations for all the methods of this interface. You can use this as a starting point and incorporate custom logic for SOAP message processing by overriding the necessary methods.

Creating a SOAP Message Handler Archive File

In order to use the implemented message handlers, they need to be made available in NetBeans and the Java CAPS Repository. This is done by importing a SOAP message handler archive file into the Java CAPS Environment, as described in "Importing a JAX-RPC Handler Archive" on page 8. If you create a custom message handler, you need to package an archive file to import.

▼ To Create the SOAP Message Handler Archive File

1 **Create and compile the message handler implementation classes and support classes, and package them into a JAR file.**

2 **Create a directory in which to store the archive files.**

3 **Copy the JAR file to this directory, and if the handler implementations depend on any other JAR files, copy them to this directory as well.**

4 **Create a descriptor XML file in the same directory and name it `messagehandlers.xml`.**

This file describes the packaged handlers and should contain the following elements. You can view a sample XML file under "Sample Descriptor File for the Console Logger Handler" on page 16.

- **message-handlers**: The parent element, which contains a list of message handlers.
- **handler**: Contains the following elements, which define the configuration of a handler.
  - **name**: The name of the message handler.
  - **description**: A brief description of the message handler.
  - **type**: The type of handler. Currently, only the `javax.xml.rpc.handler.Handler` is supported.
  - **class**: The handler implementation class name.
  - **init-properties**: Contains a list of properties for initializing the handler at runtime. For each property, you can specify the following attributes: name, default-value, required, and description. If the required attribute is set to true, a value must be specified for that property in order for the handler to function properly at runtime.

5 **Use a file archiving tool, such as WinZip, to archive the contents of the directory into a ZIP file.**

Sample SOAP Message Handler: Console Logger Handler

The following sample handler implementation logs the entire SOAP message and the MIME headers when the logging level is set to INFO on the handler class.

```
package com.stc.ws.samples.rpc.handler;

import java.io.ByteArrayOutputStream;
import java.util.Iterator;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.xml.namespace.QName;
import javax.xml.rpc.handler.GenericHandler;
import javax.xml.rpc.handler.HandlerInfo;
import javax.xml.rpc.handler.MessageContext;
import javax.xml.rpc.handler.soap.SOAPMessageContext;
import javax.xml.soap.Mimeheader;
import javax.xml.soap.MimeHeaders;
import javax.xml.soap.SOAPMessage;


public class ConsoleLoggerHandler extends GenericHandler {
   private Logger logger=Logger.getLogger(getClass().getCanonicalName());
   private QName [] qnames;
   private boolean initialized;

   public void init(HandlerInfo info) {
      qnames=info.getHeaders();
      initialized=true;
   }

   public javax.xml.namespace.QName[] getHeaders() {
      return qnames;
   }

/** Logs the SOAP request to the log file
*/
public boolean handleResponse(MessageContext mc) {
   return logMessage(mc, "SOAP Request Message is: ");
}

/** Logs the SOAP response to the log file
*/
public boolean handleResponse(MessageContext mc) {
   return log Message(mc, "SOAP Response Message is: ");
}

/** Logs the SOAP fault to the log file
*/
public boolean handleFault(MessageContext mc) {
   return logMessage(mc, "SOAP Fault Message is: ");
}

private boolean logMessage(MessageContext mc, String type) {
   if (!initialized) {
      return false;
   }

   try {
      if (logger.isLoggable(Level.INFO)) {
         logger.log(Level.INFO, type);
         SOAPMessage msg = ((SOAPMessageContext)mc).getMessage();
         //Print out the Mime Headers
         MimeHeaders mimeHeaders = msg.getMimeHeaders();
         Iterator mhIterator = mimeHeaders.getAllHeaders();
         MimeHeader mh;
         String header;
         logger.log(Level.INFO, "  Mime Headers:");
         while (mhIterator.hasNext()) {
            mh = (MimeHeader) mhIterator.next();
            header = new StringBuffer(" Name: ")
               .append(mh.getName())
               .append(" Value: ")
               .append(mh.getValue()).toString();
            logger.log(Level.INFO, header);
         }

         logger.log(Level.INFO, " SOAP Message: ");
         ByteArrayOutputStream baos = new ByteArrayOutputStream();
         msg.writeTo(baos);
         logger.log(Level.INFO,"   " + baos.toString());
         baos.close();
      }

      return true;
   } catch (Exception e) {
      if(logger.isLoggable(Level.SEVERE)) {
         logger.log(Level.SEVERE,  "Error logging SOAP message", e);
      }

      return false;
   }
}
```

Sample Descriptor File for the Console Logger Handler

The following example could be used with the sample Console Logger Handler, defined above.

```
<?xml version="1.0" encoding="UTF-8"?>
<message-handlers>
   <handler>
      <name>ConsoleLoggerHandler</name>
      <description>"Prints out the SOAP request/response/fault messages to
           the logger."
      </description>
      <type>javax.xml.rpc.handler.Handler</type>
      <class name="com.stc.ws.samples.rpc.handler.ConsoleLoggerHandler"/>
      <init-properties>
         <property name="http-return-status" default-value="503"
                   required="false" description="The HTTP status to
                   return in the response message."/>
      </init-properties>
   </handler>
</message-handlers>
```