The Java EE 5 Tutorial

Previous: SAAJ Tutorial

Next: Further Information about SAAJ

Code Examples

The first part of this tutorial uses code fragments to walk you through the fundamentals of using the SAAJ API. In this section, you will use some of those code fragments to create applications. First, you will see the program `Request.java`. Then you will see how to run the programs `HeaderExample.java`, `DOMExample.java`, `DOMSrcExample.java`, `Attachments.java`, and `SOAPFaultTest.java`.

Note –

Before you run any of the examples, follow the preliminary setup instructions in Building the Examples.

Request Example

The class `Request` puts together the code fragments used in the section SAAJ Tutorial and adds what is needed to make it a complete example of a client sending a request-response message. In addition to putting all the code together, it adds `import` statements, a `main` method, and a `try` / `catch` block with exception handling.

```
import javax.xml.soap.*;
import javax.xml.namespace.QName;
import java.util.Iterator;
import java.net.URL;

public class Request {
    public static void main(String[] args)    {
        try {
            SOAPConnectionFactory soapConnectionFactory =
                SOAPConnectionFactory.newInstance();
            SOAPConnection connection =
                soapConnectionFactory.createConnection();

            MessageFactory factory = MessageFactory.newInstance();
            SOAPMessage message = factory.createMessage();

            SOAPHeader header = message.getSOAPHeader();
            SOAPBody body = message.getSOAPBody();
            header.detachNode();

            QName bodyName = new QName("http://wombat.ztrade.com",
                "GetLastTradePrice", "m");
            SOAPBodyElement bodyElement = body.addBodyElement(bodyName);

            QName name = new QName("symbol");
            SOAPElement symbol = bodyElement.addChildElement(name);
            symbol.addTextNode("SUNW");

            URL endpoint = new URL("http://wombat.ztrade.com/quotes");
            SOAPMessage response = connection.call(message, endpoint);

            connection.close();

            SOAPBody soapBody = response.getSOAPBody();

            Iterator iterator = soapBody.getChildElements(bodyName);
            bodyElement = (SOAPBodyElement)iterator.next();
            String lastPrice = bodyElement.getValue();

            System.out.print("The last price for SUNW is ");
            System.out.println(lastPrice);
```

```
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }
```

For the `Request` class to be runnable, the second argument supplied to the `call` method would have to be a valid existing URI, and this is not true in this case.

Header Example

The example `HeaderExample.java` , based on the code fragments in the section Adding Attributes, creates a message that has several headers. It then retrieves the contents of the headers and prints them. The example generates either a SOAP 1.1 message or a SOAP 1.2 message, depending on arguments you specify. You will find the code for `HeaderExample` in the following directory:

tut-install`/javaeetutorial5/examples/saaj/headers/src/`

Building and Running the Header Example

To build the program using NetBeans IDE, follow these steps:

1. In NetBeans IDE, choose Open Project from the File menu.
2. In the Open Project dialog, navigate to *tut-install*`/javaeetutorial5/examples/saaj/` .
3. Select the `headers` folder.
4. Select the Open as Main Project check box.
5. Click Open Project.

   A Reference Problems dialog appears. Click Close.
6. Right-click the `headers` project and choose Resolve Reference Problems.
7. In the Resolve Reference Problems dialog, select the first of the missing JAR files and click Resolve.

   The missing files are `activation.jar` , `javaee.jar` , and `appserv-ws.jar` .
8. Navigate to the *as-install*`/lib/` directory.
9. Select the missing JAR file (`activation.jar` , for example) and click Open.

   In the Resolve Reference Problems dialog, all the files have green check marks to the left of their names.
10. Click Close.
11. Right-click the project and choose Build.

To run the program using NetBeans IDE, follow these steps:

1. Right-click the `headers` project and choose Properties.
2. Select Run from the Categories tree.
3. In the Arguments field, type the following:

   ```
   1.1
   ```

   This argument specifies the version of SOAP to be used in generating the message.
4. Click OK.
5. Right-click the project and choose Run.
6. Right-click the project and choose Properties.
7. Select Run from the Categories tree.
8. In the Arguments field, type the following:

   ```
   1.2
   ```
9. Click OK.
10. Right-click the project and choose Run.

To build and run HeaderExample using Ant, go to the directory *tut-install*`/javaeetutorial5/examples/saaj/headers/` . Use one of

the following commands:

```
ant run-headers -Dsoap=1.1
```

```
ant run-headers -Dsoap=1.2
```

When you run HeaderExample to generate a SOAP 1.1 message, you will see output similar to the following:

```
----- Request Message ----

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header>
<ns:orderDesk xmlns:ns="http://gizmos.com/NSURI"
 SOAP-ENV:actor="http://gizmos.com/orders"/>
<ns:shippingDesk xmlns:ns="http://gizmos.com/NSURI"
 SOAP-ENV:actor="http://gizmos.com/shipping"/>
<ns:confirmationDesk xmlns:ns="http://gizmos.com/NSURI"
 SOAP-ENV:actor="http://gizmos.com/confirmations" SOAP-ENV:mustUnderstand="1"/>
<ns:billingDesk xmlns:ns="http://gizmos.com/NSURI"
 SOAP-ENV:actor="http://gizmos.com/billing"/>
</SOAP-ENV:Header><SOAP-ENV:Body/></SOAP-ENV:Envelope>

Header name is {http://gizmos.com/NSURI}orderDesk
Actor is http://gizmos.com/orders
mustUnderstand is false

Header name is {http://gizmos.com/NSURI}shippingDesk
Actor is http://gizmos.com/shipping
mustUnderstand is false

Header name is {http://gizmos.com/NSURI}confirmationDesk
Actor is http://gizmos.com/confirmations
mustUnderstand is true

Header name is {http://gizmos.com/NSURI}billingDesk
Actor is http://gizmos.com/billing
mustUnderstand is false
```

When you run HeaderExample to generate a SOAP 1.2 message, you will see output similar to the following:

```
----- Request Message ----

<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
<env:Header>
<ns:orderDesk xmlns:ns="http://gizmos.com/NSURI"
 env:role="http://gizmos.com/orders"/>
<ns:shippingDesk xmlns:ns="http://gizmos.com/NSURI"
 env:role="http://gizmos.com/shipping"/>
<ns:confirmationDesk xmlns:ns="http://gizmos.com/NSURI"
 env:mustUnderstand="true" env:role="http://gizmos.com/confirmations"/>
<ns:billingDesk xmlns:ns="http://gizmos.com/NSURI"
 env:relay="true" env:role="http://gizmos.com/billing"/>
</env:Header><env:Body/></env:Envelope>

Header name is {http://gizmos.com/NSURI}orderDesk
Role is http://gizmos.com/orders
mustUnderstand is false
relay is false

Header name is {http://gizmos.com/NSURI}shippingDesk
Role is http://gizmos.com/shipping
mustUnderstand is false
relay is false

Header name is {http://gizmos.com/NSURI}confirmationDesk
Role is http://gizmos.com/confirmations
mustUnderstand is true
relay is false

Header name is {http://gizmos.com/NSURI}billingDesk
Role is http://gizmos.com/billing
mustUnderstand is false
relay is true
```

DOM and DOMSource Examples

The examples `DOMExample.java` and `DOMSrcExample.java` show how to add a DOM document to a message and then traverse its contents. They show two ways to do this:

- `DOMExample.java` creates a DOM document and adds it to the body of a message.
- `DOMSrcExample.java` creates the document, uses it to create a `DOMSource` object, and then sets the `DOMSource` object as the content of the message's SOAP part.

You will find the code for `DOMExample` and `DOMSrcExample` in the following directory:

tut-install/javaeetutorial5/examples/saaj/dom/src/

Examining the `DOMExample` Class

`DOMExample` first creates a DOM document by parsing an XML document. The file it parses is one that you specify on the command line.

```
static Document document;
...
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    factory.setNamespaceAware(true);
    try {
        DocumentBuilder builder = factory.newDocumentBuilder();
        document = builder.parse( new File(args[0]) );
        ...
```

Next, the example creates a SOAP message in the usual way. Then it adds the document to the message body:

```
        SOAPBodyElement docElement = body.addDocument(document);
```

This example does not change the content of the message. Instead, it displays the message content and then uses a recursive method, `getContents`, to traverse the element tree using SAAJ APIs and display the message contents in a readable form.

```
public void getContents(Iterator iterator, String indent) {

    while (iterator.hasNext()) {
        Node node = (Node) iterator.next();
        SOAPElement element = null;
        Text text = null;
        if (node instanceof SOAPElement) {
            element = (SOAPElement)node;
            QName name = element.getElementQName();
            System.out.println(indent + "Name is " + name.toString());
            Iterator attrs = element.getAllAttributesAsQNames();
            while (attrs.hasNext()){
                QName attrName = (QName)attrs.next();
                System.out.println(indent + " Attribute name is " +
                     attrName.toString());
                System.out.println(indent + " Attribute value is " +
                     element.getAttributeValue(attrName));
            }
            Iterator iter2 = element.getChildElements();
            getContents(iter2, indent + " ");
        } else {
            text = (Text) node;
            String content = text.getValue();
            System.out.println(indent + "Content is: " + content);
            }
        }
    }
```

Examining the `DOMSrcExample` Class

`DOMSrcExample` differs from `DOMExample` in only a few ways. First, after it parses the document, `DOMSrcExample` uses the document to create a `DOMSource` object. This code is the same as that of `DOMExample` except for the last line:

```
    static DOMSource domSource;
    ...
    try {
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document document = builder.parse(new File(args[0]));
        domSource = new DOMSource(document);
        ...
```

Then, after `DOMSrcExample` creates the message, it does not get the header and body and add the document to the body, as `DOMExample` does. Instead, `DOMSrcExample` gets the SOAP part and sets the `DOMSource` object as its content:

```
// Create a message
SOAPMessage message = messageFactory.createMessage();

// Get the SOAP part and set its content to domSource
SOAPPart soapPart = message.getSOAPPart();
soapPart.setContent(domSource);
```

The example then uses the `getContents` method to obtain the contents of both the header (if it exists) and the body of the message.

The most important difference between these two examples is the kind of document you can use to create the message. Because `DOMExample` adds the document to the body of the SOAP message, you can use any valid XML file to create the document. But because `DOMSrcExample` makes the document the entire content of the message, the document must already be in the form of a valid SOAP message, and not just any XML document.

Building and Running the DOM and DOMSource Examples

When you run `DOMExample` and `DOMSrcExample`, you can specify one of two sample XML files in the directory *tut-install*`/javaeetutorial5/examples/saaj/dom/`:

- `slide.xml`, a file that consists only of a message body
- `domsrc.xml`, an example that has a SOAP header (the contents of the `HeaderExample` SOAP 1.1 output) and the same message body as `slide.xml`

You can use either of these files when you run `DOMExample`. You can use `domsrc.xml` to run `DOMSrcExample`.

To build the programs using NetBeans IDE, follow these steps:

1. In NetBeans IDE, choose Open Project from the File menu.
2. In the Open Project dialog, navigate to *tut-install*`/javaeetutorial5/examples/saaj/`.
3. Select the `dom` folder.
4. Select the Open as Main Project check box.
5. Click Open Project.

   A Reference Problems dialog appears. Click Close.
6. Right-click the `dom` project and choose Resolve Reference Problems.
7. In the Resolve Reference Problems dialog, select the first of the missing JAR files and click Resolve.

   The missing files are `activation.jar`, `javaee.jar`, and `appserv-ws.jar`.
8. Navigate to the *as-install*`/lib/` directory.
9. Select the missing JAR file (`activation.jar`, for example) and click Open.

   In the Resolve Reference Problems dialog, all the files have green check marks to the left of their names.
10. Click Close.
11. Right-click the project and choose Build.

To run `DOMExample` using NetBeans IDE, follow these steps:

1. Right-click the `dom` project and choose Properties.
2. Select Run from the Categories tree.
3. Click Browse next to the Main Class field.
4. In the Browse Main Classes dialog, select `DomExample`.
5. Click Select Main Class.
6. In the Arguments field, type the following:

   ```
   slide.xml
   ```
7. Click OK.
8. Right-click the project and choose Run.

To run `DOMSrcExample` using NetBeans IDE, follow these steps:

1. Right-click the `dom` project and choose Properties.
2. Select Run from the Categories tree.
3. Click Browse next to the Main Class field.

4. In the Browse Main Classes dialog, select `DomSrcExample` .
5. Click Select Main Class.
6. In the Arguments field, type the following:

```
domsrc.xml
```

7. Click OK.
8. Right-click the project and choose Run.

To run the examples using Ant, go to the directory *tut-install*`/javaeetutorial5/examples/saaj/dom/` .

To run `DOMExample` using Ant, use the following command:

```
ant run-dom -Dxml-file=slide.xml
```

To run `DOMSrcExample` using Ant, use the following command:

```
ant run-domsrc -Dxml-file=domsrc.xml
```

When you run `DOMExample` using the file `slide.xml` , you will see output that begins like the following:

```
    Running DOMExample.
    Name is slideshow
     Attribute name is author
     Attribute value is Yours Truly
     Attribute name is date
     Attribute value is Date of publication
     Attribute name is title
     Attribute value is Sample Slide Show
     Content is:
...
```

When you run `DOMSrcExample` using the file `domsrc.xml` , you will see output that begins like the following:

```
    Running DOMSrcExample.
    Header contents:
    Content is:

    Name is {http://gizmos.com/NSURI}orderDesk
     Attribute name is SOAP-ENV:actor
     Attribute value is http://gizmos.com/orders
    Content is:
   ...
```

If you run `DOMSrcExample` with the file `slide.xml` , you will see runtime errors.

Attachments Example

The example `Attachments.java` , based on the code fragments in the sections Creating an `AttachmentPart` Object and Adding Content and Accessing an `AttachmentPart` Object, creates a message that has a text attachment and an image attachment. It then retrieves the contents of the attachments and prints the contents of the text attachment. You will find the code for the `Attachments` class in the following directory:

tut-install`/javaeetutorial5/examples/saaj/attachments/src/`

`Attachments` first creates a message in the usual way. It then creates an `AttachmentPart` for the text attachment:

```
AttachmentPart attachment1 = message.createAttachmentPart();
```

After it reads input from a file into a string named `stringContent` , it sets the content of the attachment to the value of the string and the type to `text/plain` and also sets a content ID.

```
attachment1.setContent(stringContent, "text/plain");
attachment1.setContentId("attached_text");
```

It then adds the attachment to the message:

```
message.addAttachmentPart(attachment1);
```

The example uses a `javax.activation.DataHandler` object to hold a reference to the graphic that constitutes the second attachment. It creates this attachment using the form of the `createAttachmentPart` method that takes a `DataHandler` argument.

```
// Create attachment part for image
URL url = new URL("file:///../xml-pic.jpg");
DataHandler dataHandler = new DataHandler(url);
AttachmentPart attachment2 = message.createAttachmentPart(dataHandler);
attachment2.setContentId("attached_image");

message.addAttachmentPart(attachment2);
```

The example then retrieves the attachments from the message. It displays the `contentId` and `contentType` attributes of each attachment and the contents of the text attachment.

Building and Running the Attachments Example

The `Attachments` class takes a text file as an argument. You can specify any text file. The `attachments` directory contains a file named `addr.txt` that you can use.

To build the program using NetBeans IDE, follow these steps:

1. In NetBeans IDE, choose Open Project from the File menu.
2. In the Open Project dialog, navigate to *tut-install*`/javaeetutorial5/examples/saaj/` .
3. Select the `attachments` folder.
4. Select the Open as Main Project check box.
5. Click Open Project.

   A Reference Problems dialog appears. Click Close.
6. Right-click the `attachments` project and choose Resolve Reference Problems.
7. In the Resolve Reference Problems dialog, select the first of the missing JAR files and click Resolve.

   The missing files are `activation.jar` , `javaee.jar` , and `appserv-ws.jar` .
8. Navigate to the *as-install*`/lib/` directory.
9. Select the missing JAR file (`activation.jar` , for example) and click Open.

   In the Resolve Reference Problems dialog, all the files have green check marks to the left of their names.
10. Click Close.
11. Right-click the project and choose Build.

To run the program using NetBeans IDE, follow these steps:

1. Right-click the `attachments` project and choose Properties.
2. Select Run from the Categories tree.
3. In the Arguments field, type the name of a text file:

   ```
   addr.txt
   ```
4. Click OK.
5. Right-click the project and choose Run.

To run `Attachments` using Ant, go to the directory *tut-install*`/javaeetutorial5/examples/saaj/attachments/` . Use the following command:

```
ant run-att -Dfile=path-name
```

Specify a text file as the *path-name* argument:

```
ant run-att -Dfile=addr.txt
```

When you run `Attachments` using this file, you will see output like the following:

```
Running Attachments.
```

```
Attachment attached_text has content type text/plain
Attachment contains:
Update address for Sunny Skies, Inc., to
10 Upbeat Street
Pleasant Grove, CA 95439
USA

Attachment attached_image has content type image/jpeg
```

SOAP Fault Example

The example `SOAPFaultTest.java`, based on the code fragments in the sections Creating and Populating a `SOAPFault` Object and Retrieving Fault Information, creates a message that has a `SOAPFault` object. It then retrieves the contents of the `SOAPFault` object and prints them. You will find the code for `SOAPFaultTest` in the following directory:

tut-install`/javaeetutorial5/examples/saaj/fault/src/`

Like `HeaderExample`, the `SOAPFaultTest` class contains code that allows you to generate either a SOAP 1.1 or a SOAP 1.2 message.

Building and Running the SOAP Fault Example

To build the program using NetBeans IDE, follow these steps:

1. In NetBeans IDE, choose Open Project from the File menu.
2. In the Open Project dialog, navigate to *tut-install*`/javaeetutorial5/examples/saaj/`.
3. Select the `fault` folder.
4. Select the Open as Main Project check box.
5. Click Open Project.

   A Reference Problems dialog appears. Click Close.
6. Right-click the `fault` project and choose Resolve Reference Problems.
7. In the Resolve Reference Problems dialog, select the first of the missing JAR files and click Resolve.

   The missing files are `activation.jar`, `javaee.jar`, and `appserv-ws.jar`.
8. Navigate to the *as-install*`/lib/` directory.
9. Select the missing JAR file (`activation.jar`, for example) and click Open.

   In the Resolve Reference Problems dialog, all the files have green check marks to the left of their names.
10. Click Close.
11. Right-click the project and choose Build.

To run the program using NetBeans IDE, follow these steps:

1. Right-click the `fault` project and choose Properties.
2. Select Run from the Categories tree.
3. In the Arguments field, type the following:

   ```
   1.1
   ```

   This argument specifies the version of SOAP to be used in generating the message.
4. Click OK.
5. Right-click the project and choose Run.
6. Right-click the project and choose Properties.
7. Select Run from the Categories tree.
8. In the Arguments field, type the following:

   ```
   1.2
   ```
9. Click OK.
10. Right-click the project and choose Run.

To build and run `SOAPFaultTest` using Ant, go to the directory *tut-install*`/javaeetutorial5/examples/saaj/fault/` . Use one of the following commands:

```
ant run-fault -Dsoap=1.1
```

```
ant run-fault -Dsoap=1.2
```

When you run `SOAPFaultTest` to generate a SOAP 1.1 message, you will see output like the following (line breaks have been inserted in the message for readability):

```
Here is what the XML message looks like:

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/><SOAP-ENV:Body>
<SOAP-ENV:Fault><faultcode>SOAP-ENV:Client</faultcode>
<faultstring>Message does not have necessary info</faultstring>
<faultactor>http://gizmos.com/order</faultactor>
<detail>
<PO:order xmlns:PO="http://gizmos.com/orders/">
Quantity element does not have a value</PO:order>
<PO:confirmation xmlns:PO="http://gizmos.com/confirm">
Incomplete address: no zip code</PO:confirmation>
</detail></SOAP-ENV:Fault>
</SOAP-ENV:Body></SOAP-ENV:Envelope>

SOAP fault contains:
 Fault code = {http://schemas.xmlsoap.org/soap/envelope/}Client
 Local name = Client
 Namespace prefix = SOAP-ENV, bound to http://schemas.xmlsoap.org/soap/envelope/
 Fault string = Message does not have necessary info
 Fault actor = http://gizmos.com/order
 Detail entry = Quantity element does not have a value
 Detail entry = Incomplete address: no zip code
```

When you run SOAPFaultTest to generate a SOAP 1.2 message, the output looks like this:

```
Here is what the XML message looks like:

<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
<env:Header/><env:Body>
<env:Fault>
<env:Code><env:Value>env:Sender</env:Value></env:Code>
<env:Reason><env:Text xml:lang="en-US">
Message does not have necessary info
</env:Text></env:Reason>
<env:Role>http://gizmos.com/order</env:Role>
<env:Detail>
<PO:order xmlns:PO="http://gizmos.com/orders/">
Quantity element does not have a value</PO:order>
<PO:confirmation xmlns:PO="http://gizmos.com/confirm">
Incomplete address: no zip code</PO:confirmation>
</env:Detail></env:Fault>
</env:Body></env:Envelope>

SOAP fault contains:
 Fault code = {http://www.w3.org/2003/05/soap-envelope}Sender
 Local name = Sender
 Namespace prefix = env, bound to http://www.w3.org/2003/05/soap-envelope
 Fault reason text = Message does not have necessary info
 Fault role = http://gizmos.com/order
 Detail entry = Quantity element does not have a value
 Detail entry = Incomplete address: no zip code
```

Previous: SAAJ Tutorial

Next: Further Information about SAAJ