

Guide	<b>Integration with QMWISe in Java</b>
Version	
Author	Matt Elton, VLE Genius
Date	March 10, 2006
Revised	

<b>Simple GetAccessAssessmentList Example</b> .....	<b>3</b>
<b>Simple XML Example</b> .....	<b>3</b>
<b>Apache SOAP Example</b> .....	<b>7</b>
<b>Summary</b> .....	<b>12</b>
<b>Security Headers</b> .....	<b>12</b>
<b>Element Attributes</b> .....	<b>12</b>
<b>Running the Java examples</b> .....	<b>13</b>
<b>You Will Need</b> .....	<b>13</b>
<b>Executing the examples</b> .....	<b>13</b>
<b>References</b> .....	<b>14</b>

## Simple GetAccessAssessmentList Example

---

All requests made to QMWISe should be made using SOAP, but SOAP is just a HTTP POST query containing XML content as in the simple example below.

```
POST /QMWISE4/QMWISE.asmx HTTP/1.1
Host: perception.my.edu
Content-Type: text/xml; charset=utf-8
Content-Length: 1003
SOAPAction: "http://questionmark.com/QMWISE/GetAccessAssessmentList"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <Security xmlns="http://questionmark.com/QMWISE/">
      <ClientID>Manager</ClientID>
      <Checksum>5803539209095334</Checksum>
    </Security>
  </soap:Header>
  <soap:Body>
    <GetAccessAssessmentList xmlns="http://questionmark.com/QMWISE/">
      <Participant_Name>bob</Participant_Name>
    </GetParticipant>
  </soap:Body>
</soap:Envelope>
```

In Java one can use one of three methods to create such SOAP requests and an implementation of each is included in the java examples zip file. The first would be to generate the query by building the XML component and HTTP headers using an `URLConnection` and a string buffer, but XML content could easily be built the using an XML document builder. Finally, the most high level method and method recommended for developers planning to create data and return complex objects, would be to use a SOAP implementation.

All the data returned from QMWISe is also in XML form and any implementation will need to parse this information unless the SOAP implementation you are using does this for you.

## Simple XML Example

This section is devoted to a simple example in which the body of the request is generated using an XML document builder. This example can be run using any SDK version of Java 1.4 or 1.5.

To start obtain an instance of the XML document builder so that you can start building the XML content.

```
// We will use the default DocumentBuilderFactory to generate an XML document
```

```
DocumentBuilderFactory domFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder domBuilder = domFactory.newDocumentBuilder();
// create a new XML Document for this SOAP request
Document doc = domBuilder.newDocument();
```

Now add the outer soap tags;

```
Element soapEnvelope = doc.createElement( "soap:Envelope" );
// add the SOAP schema attribute
soapEnvelope.setAttribute( "xmlns:soap", "http://schemas.xmlsoap.org/soap/envelope/" );
// add to the root
doc.appendChild( soapEnvelope );
```

The SOAP envelope tags can contain a header element and will always include a body element. You need to pass the header element when Perception requires security parameters, but you can still pass them if not required.

```
// to add security parameters create a header element
Element headerElement = doc.createElement( "soap:Header" );

// create Security elements
Element securityElement = doc.createElement( "Security" );
// add the QMWISe namespace
securityElement.setAttribute( "xmlns", "http://questionmark.com/QMWISe/" );

// add security parameters
addSimpleTextNode( securityElement, "ClientID", "Manager" );
addSimpleTextNode( securityElement, "Checksum", "7d7c48132f368983a0b12a877b1a59ab" );

// add security element to header
headerElement.appendChild( securityElement );

// add headerElement to envelope
soapEnvelope.appendChild( headerElement );
```

Notice that the Security header includes an attribute, **xmlns**, this is required and should always be <http://questionmark.com/QMWISe/> - this is the SOAP namespace and defines the object within SOAP. There are also two elements within the security header, **ClientID** and **Checksum**, it is important that you understand what these are and how you generate the value of checksum. Firstly, **ClientID** is the username of a Perception Manager account, such as Manager. This account will need maximum privileges within the Perception system in order to read, write and delete any content accessible from QMWISe. The **Checksum** is an encrypted value created using MD5 and the combination of the **ClientID** and the encrypted password for the same account. It is not possible to generate the checksum within Java; instead one must obtain this using the QMWISe Test Harness. Please see

<http://www.questionmark.com/perception/help/v4/manuals/qm/harn/use/md5.htm> for more details.

After the headers the next step is to add the body element. This part is required and will include the SOAP method name and any parameters, including objects that are required by the SOAP method. Note that the method requires an **xmlns** attribute, as the method is unknown to SOAP and needs defining. Parameters should not have any attributes when calling QMWISe.

```
Element bodyElement = doc.createElement( "soap:Body" );

// This adds the SOAP method to the request: GetAccessAssessmentList
Element methodElement = doc.createElement( "GetAccessAssessmentList" );
// add the QMWISe namespace
methodElement.setAttribute( "xmlns", "http://questionmark.com/QMWISe/" );

// now add the parameters for this method
addSimpleTextNode( methodElement, "Participant_Name", "bob" );

// add method to body
bodyElement.appendChild( methodElement );
// add body to soap envelope
soapEnvelope.appendChild( bodyElement );
```

Now that the SOAP envelope has been created the next step is to open a connection to QMWISe on the Perception server;

```
// create an URL object
URL QMWISUrl = new URL( "http://perception.my.edu/QMWISe4/QMWISe.asmx" );
// Create a connection with QMWISe
URLConnection connection = QMWISUrl.openConnection();
// This is always a HTTP connection
HttpURLConnection httpConn = (HttpURLConnection) connection;
```

Some headers are required;

```
httpConn.setRequestProperty( "Content-Length", length_of_xml_content );
httpConn.setRequestProperty( "Content-Type", "text/xml; charset=utf-8" );
httpConn.setRequestProperty( "SOAPAction", SOAPAction);
httpConn.setRequestMethod( "POST" );
httpConn.setDoOutput(true);
httpConn.setDoInput(true);
```

The **length\_of\_xml\_content** is the length of the XML document when converted to a byte array. This byte array is written to `httpConn.getOutputStream()` and you'll find an example of how to do this in the examples source code. The **SOAPAction** value is the

namespace appended with the method name, which would be **http://questionmark.com/QMWISe/GetAccessAssessmentList** in this example.

Once the request output stream has been closed it is safe to call the response object. You can parse the response manually or convert the response into an XML document and navigate the elements as required. The following is an example response;

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <Security xmlns="http://questionmark.com/QMWISe/">
      <ClientID>Manager</ClientID>
      <Checksum>7d7c48132f368983a0b12a877b1a59ab</Checksum>
    </Security>
  </soap:Header>
  <soap:Body>
    <GetAccessAssessmentListResponse
xmlns="http://questionmark.com/QMWISe/">
      <URL>http://</URL>
    </GetAccessAssessmentListResponse>
  </soap:Body>
</soap:Envelope>
```

When the response is successful QMWISe will return the security headers (if sent in the request) that can be used to authenticate the response, and a response object in the form method name + "Response". A detailed XML parser is beyond the scope of this document but here is a simple example that writes the access URL to the standard output stream;

```
// Read the response and write it to standard out.
if( httpConn.getResponseCode() == 500 ){ // error
    System.out.println( "ERROR!" );
}else{ // success

    // get the content input stream – the XML
    InputStreamReader isr = new InputStreamReader( httpConn.getInputStream() );

    // create an XML document from the response
    Document document = domBuilder.parse( new InputSource(isr) );
    // get a list of elements with the name URL (should be just one)
    NodeList urlList = document.getElementsByTagName( "URL" );
    for( int i=0; i < urlList.getLength(); i++ ){
```

```
        // node should be an simple text node element
        Element urlElement = (Element) urlList.item(i);
        // print URL value, text is first child in this case
        System.out.println( "URL = " + urlElement.getFirstChild().getNodeValue() );
    }
}
```

## Apache SOAP Example

The following details how to integrate with QMWISe using the Apache SOAP libraries. This method isn't entirely straightforward because some standard SOAP attributes added by the Apache tools are not accepted by Microsoft .Net, the underlying structure used in QMWISe. Therefore, you will need to create serializers for all parameters and objects sent in the request.

To use the Apache SOAP libraries you may need access to the following jars; soap.jar, activation.jar, mail.jar. These are often available on the Apache web server, but you can download the libraries using the links included in the examples source code.

The first steps to create a SOAP request are as follows;

```
// create a new SOAP Call - the underlying code formats the HTTP request and response
Call call = new Call();

// set the namespace
call.setTargetObjectURI( "http://questionmark.com/QMWISe/" );

// set the QMWISe method
call.setMethodName( "GetAccessAssessmentList" );
// and encoding style
call.setEncodingStyleURI( Constants.NS_URI_SOAP_ENC );
```

You may choose to add more parameters to the Call but these are beyond the scope of this document.

To add parameters to the body and header elements you must create Vector objects. The body must contain SOAP Parameter objects, but the header can contain other types of recognised objects such as an XML Element. As QMWISe expects the security parameters within it's own Security tag it is easier to use XML to add the security header.

```
Vector headerParams = new Vector();
// create a new XML Document for the header elements
```

```
Document headerDoc = domBuilder.newDocument();

// create Security element
Element securityElement = headerDoc.createElement( "Security" );

// add the QMWISe namespace
securityElement.setAttribute( "xmlns", "http://questionmark.com/QMWISe/" );

// add security parameters
addSimpleTextNode( securityElement, "ClientID", "Manager" );
addSimpleTextNode( securityElement, "Checksum", checksum );

// add to header params
headerParams.add( securityElement );

// create SOAP Header object
Header header = new Header();
header.setHeaderEntries( headerParams );

// add header to call
call.setHeader( header );
```

Note that this is very similar to the previous example in standard XML, the only difference is that the security XML element is added as the only entry in a Vector object, which is used in the call to `header.setHeaderEntries()`.

When using the SOAP Call, parameters must be created as Parameter objects and added to the Call as elements of a Vector. In the following example the only parameter is Participant\_Name, so there will be only one entry in the Vector, but there are some QMWISe methods that accept more than one parameter, all of which would need to be created as a Parameter and added to the Vector. When passing objects to QMWISe, as with `SetParticipant` (the method used to create a participant in Perception), the participant object is the only parameter and you'll need to create a serializer for the Participant object and not each of its fields.

```
// create params vector
Vector params = new Vector();

// create participant parameter
Parameter participantParameter = new Parameter("Participant_Name", String.class, "bob", null);

params.addElement( participantParameter );

// add the params to the SOAP call
call.setParams( params );
```

When creating a new Parameter instance, the variables required are the parameter name, the type and the value of the object (the final option is the encoding – for simplicity this is null). In the above case Participant\_Name is just a String, but the parameter could be an object such as a Participant in which case the object type could be Participant.class or the internal class the Participant details are created from. Now we need to setup a registry to map each parameter with a serializer (Note: the SOAP request can be made without creating a mapping registry or custom serializers, but QMWISe will not recognise the requests).

```
// step one, create a new SOAPMappingRegistry to map types to serializers
SOAPMappingRegistry smr = new SOAPMappingRegistry();

// step two, create a QName to register a new type
QName qname = new QName("http://questionmark.com/QMWISe/", "Participant_Name" );

// Now declare a Serializer for this parameter
SimpleSerializer serializer = new SimpleSerializer();

// add the Participant_Name mapping to the mapping registry
smr.mapTypes( Constants.NS_URI_SOAP_ENC, qname, String.class, serializer, deserializer);
```

The SOAP Call object when generating the string form of the XML content calls the SimpleSerializer. It's function is to generate the tags and content of the parameter, EG;

```
<Participant_Name>bob</Participant_Name>
```

This would also include fields if the serializer produced the tags for an object.

A simple example of a serializer follows, for more information please see the Apache SOAP documentation. Remember that only the namespace should be included, as an attribute on objects and parameters, fields should have no attributes.

```
private static class SimpleSerializer implements Serializer {

    public void marshal(      String inScopeEncStyle, Class javaType, Object src,
                            Object context, Writer sink, NSStack nsStack,
                            XMLJavaMappingRegistry xjmr, SOAPContext ctx) throws
    IllegalArgumentException, IOException {

        // open tags
        nsStack.pushScope();

        if( src != null ){

            sink.write("<" + context);
```

```

sink.write(" xmlns=\"" + "http://questionmark.com/QMWISe/" + "\"");
sink.write(">");
// note: you will need to escape reserved characters here
sink.write( (String) src );
sink.write("</" + context + '>');

    } else {
        // add null structure
        sink.write("<" + context);
        sink.write(">");
        sink.write("</" + context + '>');
    }

    // close tags (no children)
    nsStack.popScope();

}
}

```

In addition to serializers SOAP requires the mapping of deserializers. These will read the parameters, including objects, returned in the response from QMWISe. In this example the response contains just one XML tag, URL. The following is a very simple example that will read the URL tags and return the text within as a String.

```

private static class SimpleDeserializer implements Deserializer {

    public Bean unmarshall(String inScopeEncStyle, QName elementType, Node src,
XMLJavaMappingRegistry xjmr, SOAPContext ctx) throws IllegalArgumentException {

        // tag is in the form <URL>http://perception.my.edu/.....</URL>
        // so just read the first child, which is the URL
        if( src.getFirstChild() instanceof Element ){

            String nodeValue = src.getFirstChild().getFirstChild().getNodeValue();
            return new Bean( String.class, nodeValue );

        }
        return new Bean( String.class, src.getFirstChild().getNodeValue() );

    }
}

```

This should be used to map the URL type in the mapping registry as follows;

```

// create a QName to match the URL sent in the response from Perception
QName urlName = new QName("http://questionmark.com/QMWISe/", "URL" );
// declare the deserializer
SimpleDeserializer deserializer = new SimpleDeserializer();
// map the deserializer

```

```
smr.mapTypes( Constants.NS_URI_SOAP_ENC, urlName, String.class, serializer, deserializer );

// add mapping registry to the SOAP Call object
call.setSOAPMappingRegistry( smr );
```

Note that each mapping can have both a serializer and deserializer should the same parameter be found in the both the request and in the response.

To make the Call and return a response execute as follows.

```
String SOAPMethod = "http://questionmark.com/QMWISE/GetAccessAssessmentList";
Response resp = call.invoke( QMWISEUrl, SOAPMethod );
```

The SOAP Call will attempt to connect to QMWISE and generate a list of parameters from the data returned using the specified deserializers. Should QMWISE return an unknown parameter type an error will be thrown.

This sample code will check for SOAP errors sent back from QMWISE, such as "invalid security credentials" or "participant not found", then print the URL to standard out if successful.

```
if ( resp.generatedFault () ) { // error found
    Fault fault = resp.getFault();
    StringBuffer errorBuf = new StringBuffer();
    errorBuf.append( "Perception SOAP call failed: " );
    errorBuf.append(" Code = " + fault.getFaultCode());
    errorBuf.append(" String = " + fault.getFaultString());
    System.out.println( errorBuf.toString() );
} else {
    Parameter result = resp.getReturnValue ( );
    String accessUrl = result.getValue();
    System.out.println( accessUrl );
}
```

## Summary

---

### Security Headers

Security headers, when required, should be in the form;

```
<Security xmlns="http://questionmark.com/QMWISe/">
  <ClientID>Manager</ClientID>
  <Checksum>7d7c48132f368983a0b12a877b1a59ab</Checksum>
</Security>
```

Where ClientID is a manager account username and the checksum, which must be obtained using the QMWISe Test Harness.

### Element Attributes

All Perception objects, parameters and SOAP methods should have the namespace attribute and no other. Apache SOAP will add type and encoding attributes by default, which are not recognised by QMWISe.

```
<Participant xmlns="http://questionmark.com/QMWISe/">
  <Participant_ID>12345678</Participant_ID>
  <Participant_Name>Bob</Participant_Name>
  ...
</Participant>
```

To format data in this way you will need to create serializers if using Apache SOAP.

## Running the Java examples

---

### You Will Need

The java examples jar file from the Question site.

Java SDK 1.4 or later

To run the SOAP example you may need additional jars, soap.jar from [Apache SOAP](#), activation.jar from <http://java.sun.com/products/javabeans/glasgow/jaf.html> and mail.jar from <http://java.sun.com/products/javamail/>.

### Executing the examples

Copy the jar file to a location on an accessible drive, change to this directory in a shell or command prompt and execute as follows;

```
java -cp perception-examples4.2.jar com.qm.examples.SSOExample
```

Replace SSOExample with ParticipantSSOExample, ParticipantSSOSOAPExample or ParticipantSSOXMLExample as required. All of these examples accept parameters (run without parameters for a list).

Note: if java is not in the system classpath it will be necessary to execute with the full java path eg;

```
/usr/local/jdk1.4.2_06/bin/java -cp perception...
```

or

```
c:\Program Files\Java\jdk1.4.2_06\bin\java -cp perception...
```

## References

---

QMWISE Error Codes:

[www.questionmark.com/perception/help/v4/manuals/qm/mgt/err/code.htm](http://www.questionmark.com/perception/help/v4/manuals/qm/mgt/err/code.htm)

Obtaining the checksum from QMWISE:

[www.questionmark.com/perception/help/v4/manuals/qm/harn/use/md5.htm](http://www.questionmark.com/perception/help/v4/manuals/qm/harn/use/md5.htm)

Apache SOAP:

<http://ws.apache.org/soap/>

Java SDK:

<http://java.sun.com/>