

```

package org.perfsonar.client.testHarness;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.InputStream;
import java.net.URL;
import java.security.Key;
import java.security.KeyFactory;
import java.security.KeyStore;
import java.security.PrivateKey;
import java.security.Provider;
import java.security.cert.Certificate;
import java.security.cert.X509Certificate;
import java.security.spec.PKCS8EncodedKeySpec;
import java.util.Vector;

import javax.xml.namespace.QName;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.apache.axis.Message;
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.message.SOAPBodyElement;
import org.apache.axis.message.SOAPEnvelope;
import org.apache.axis.utils.XMLUtils;
import org.apache.ws.security.SOAPConstants;
import org.apache.ws.security.WSConstants;
import org.apache.ws.security.WSEncryptionPart;
import org.apache.ws.security.components.crypto.Crypto;
import org.apache.ws.security.message.WSSecHeader;
import org.apache.ws.security.message.WSSecSignature;
import org.apache.ws.security.util.Base64;
import org.apache.ws.security.util.WSSecurityUtil;
import org.apache.xml.serialize.OutputFormat;
import org.apache.xml.serialize.XMLSerializer;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.w3c.dom.Document;

import org.perfsonar.commons.auxiliary.components.authn.DynamicCrypto;
import org.perfsonar.commons.auxiliary.components.authn.SOAPUtil;

/**
 * Class which provides the basic web service (Doc/Lit) client capability
 * without stubs but including the possibility of sending the X.509
 * certificate security token.
 *
 * @author kan
 */

```

```
public class SOAPX509Client {
```

```
    @SuppressWarnings(value={"deprecation"})  
    public void makeRequest(String[] args) {  
        try {
```

### STEP: INIT REQUEST

```
        String endPoint = null;  
        String inputFile = null;  
        String outputFile = null;  
        String keyFile = null;  
        String certFile = null;  
        if (args.length == 5) {  
            endPoint = args[0];  
            inputFile = args[1];  
            outputFile = args[2];  
            keyFile = args[3];  
            certFile = args[4];  
        } else {  
            System.out.println("Error: Wrong number (" + args.length + ") of  
parameters!!!");  
            return;  
        }  
        System.out.println("End point: " + endPoint);  
        System.out.println("Request file: " + inputFile);  
        System.out.println("Response file: " + outputFile);  
        System.out.println("Private key file: " + keyFile);  
        System.out.println("Certificate file: " + certFile);  
  
        // read the certificate... more or less...  
        InputStream isCert = new FileInputStream(certFile);
```

### STEP: SET SERVICE ELEMENTS

```
        // prepare to call - set service elements  
        Service service = new Service();  
        Call call = (Call)service.createCall();  
        call.setTargetEndpointAddress(new URL(endPoint));  
        call.setOperationName(new QName("http://  
soapinterop.org/", "submit"));
```

### STEP: CREATE A SOAP ENVELOPE

```
        SOAPEnvelope envelope = new SOAPEnvelope();
```

### STEP: ADD THE MESSAGE INTO A SOAPBODY

```
        // read the request into a org.w3c.DOM.Document  
        Document request = null;
```

```

);
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance
factory.setNamespaceAware(true);

DocumentBuilder builder = factory.newDocumentBuilder();
request = builder.parse(new File(inputFile));

// build a SOAPBodyElement from the document
SOAPBodyElement requestMessage =
    new SOAPBodyElement(request.getDocumentElement());
envelope.addBodyElement(requestMessage);

```

### STEP: ADD THE SECURITY PROVIDER

```

// add the security provider
BouncyCastleProvider bcp = new BouncyCastleProvider();
java.security.Security.addProvider((Provider)bcp);

```

### STEP: READ PRIVATE AND PUBLIC KEYS

```

// add the private and public keys
Crypto crypto = new DynamicCrypto();
KeyStore ks=crypto.getKeyStore();

BufferedReader in = new BufferedReader(new FileReader(keyFile));
String str;
String previousStr="";
String data="";
in.readLine();
while ((str = in.readLine()) != null) {
    data+=previousStr;
    previousStr=str+"\n";
}
in.close();

byte[] bytes=Base64.decode(data);
PKCS8EncodedKeySpec keySpec = new PKCS8EncodedKeySpec(bytes);
KeyFactory keyFactory = KeyFactory.getInstance("RSA");
PrivateKey privateKey = keyFactory.generatePrivate(keySpec);
X509Certificate cert=crypto.loadCertificate(isCert);
ks.setKeyEntry("xmlsec", (Key)privateKey, "security".toCharArray(),
new Certificate[] {(Certificate)cert});

```

### STEP: CREATE A WS-SEC HEADER

```

WSSEHeader secHeader = new WSSEHeader();
secHeader.setActor("test");

```

### STEP: ELEMENTS TO BE SIGNED

```

WSSESignature sec509 = new WSSESignature();

```

```

sec509.setUserInfo("xmlsec", "security");
SOAPConstants soapConstants = WSSecurityUtil.getSOAPConstants
(envelope.getAsDOM());
Vector<WSEncryptionPart> parts = new Vector<WSEncryptionPart>(1,1);

// Set up to use STRTransform to sign the signature token
WSEncryptionPart encP =
    new WSEncryptionPart(
        "STRTransform",
        soapConstants.getEnvelopeURI(),
        "Content");
parts.add(encP);

sec509.setParts(parts);
sec509.setKeyIdentifierType(WSSConstants.BST_DIRECT_REFERENCE);

```

### STEP: ADD THE WS-SEC HEADER

```

Document doc = envelope.getAsDocument();
secHeader.insertSecurityHeader(doc);

// Signing the message
Document signedDoc = sec509.build(doc, crypto, secHeader);

Message signedMsg = (Message) SOAPUtil.toSOAPMessage(signedDoc);
envelope = signedMsg.getSOAPEnvelope();

```

### STEP: SEND THE REQUEST

```

// Saving SOAP message
saveSOAPMessage(envelope.getAsDocument(), inputFile);

// get a timestamp.
double startTime =
    new Long(System.currentTimeMillis()).doubleValue();

// call on the end point
Object resultObject = call.invoke(envelope);

// get another timestamp
double endTime =
    new Long(System.currentTimeMillis()).doubleValue();

SOAPEnvelope envelopeResult;
SOAPBodyElement resultSBE;
Document result = null;
try {

    envelopeResult= (SOAPEnvelope)resultObject;
    saveSOAPMessage(envelopeResult.getAsDocument(), outputFile);
    resultSBE= envelopeResult.getFirstBody();
}

```

```

// change it to document - here is where validity
// can be checked..
result = resultSBE.getAsDocument();

// output it to a file
File response = new File(outputFile);
FileWriter outWriter = new FileWriter(response);

OutputFormat format = new OutputFormat( result );
format.setIndent(4);
format.setIndenting(true);
format.setLineSeparator("\n");

XMLSerializer serial = new XMLSerializer(outWriter, format );
serial.asDOMSerializer();
serial.serialize( result.getDocumentElement() );

outWriter.close();
} catch (ClassCastException e) {
    e.printStackTrace();
    System.out.println("SOAPX509Client.makeRequest: We didn't get a
Vector of SOAPBodyElements!");
} catch (Exception e) {
    System.out.println("SOAPX509Client.makeRequest: General
exception while retrieving report");
    e.printStackTrace();
}
System.out.println("time taken :"+((endTime-startTime)/1000d)+"
secs");
} catch(Exception e) {
    System.err.println("SOAPX509Client.makeRequest: General exception
encountered by client");
    e.printStackTrace();
}

System.out.println("Client exiting");
}

private void saveSOAPMessage(Document doc, String file) {
    try {
        File response = new File(file+".soap.xml");
        FileWriter outWriter = new FileWriter(response);
        XMLUtils.PrettyElementToWriter(doc.getDocumentElement(),outWriter);
        outWriter.close();
    } catch (Exception e) {
        System.out.println("SOAPX509Client.saveSOAPMessage: General
exception while writing SOAP message");
        e.printStackTrace();
    }
}

public static void main(String[] args) {

```

```
SOAPX509Client doclitClient = new SOAPX509Client();  
doclitClient.makeRequest(args);
```

```
    }  
}
```