Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Eidgenössisches Finanzdepartement EFD

Bundesamt für Informatik und Telekommunikation BIT
Lösungszentrum

Christian Zeiler

e-dec Web Services

Best Practice WS-Stacks

Projektname: **e-dec**
Version: **0.4**
Datum: **2010-06-01**

Status	in Arbeit	in Prüfung	genehmigt zur Nutzung
	☐	☐	☑

Beteiligter Personenkreis	
Autoren:	Christian Zeiler, Patrick Schweizer
Genehmigung:	PL
Benützer/Anwender:	Projektgruppe, Zollkunden
zur Information/Kenntnis:	Projektgruppe

Änderungskontrolle, Prüfung, Genehmigung			
Wann	**Version**	**Wer**	**Beschreibung**
22.10.08	0.1	cze	Grundversion
22.01.09	0.1	mru	C# Teil wurde hinzugefügt
29.01.09	0.3	pas	AXIS2 Teil hinzugefügt
03.05.10	0.4	cze	Kapitel 2.7 erstellt

Inhaltsverzeichnis

1 Einleitung

In diesem Dokument werden die Erfahrungen und Best Practices mit den verschiedenen Web Service Stacks dokumentiert. Die Erfahrungen pro Web Service Stack werden jeweils in einem eigenen Kapitel aufgeführt.

1.1 Referenzen

Die folgenden Quellen werden im Dokument referenziert oder haben als Grundlage gedient:

Ref	Titel	Version
[1]	Eclipse Projekt JAX-WS Client: edec_wsclient_jaxws	-
[2]	Viusal Studio Express 2008 Projekt: edec_wsclient_csharp	-

2 JAX-WS

Bei JAX-WS handelt es sich um eine Spezifikation die im JSR 224 beschrieben wird. Als Implementierung des JSR 224 wird die Referenzimplementierung von Sun verwendet. Die beschriebenen Erfahrungen beziehen sich auf die Version 2.1.3 der Referenzimplementation. Der Sourcecode stammt aus dem Eclipse Projekt für JAX-WS [1].

2.1 Links

Die folgenden Links enthalten Informationen zu JAX-WS:

- JSR 224 - http://jcp.org/en/jsr/detail?id=224
- JAX-WS RI - https://jax-ws.dev.java.net/
- JAX-WS UserGuide - https://jax-ws.dev.java.net/guide/

2.2 Generierung der Stub Klassen

Beim Generieren der Stub Klassen müssen die extension von JAX-WS verwendet werden damit die Klassen generiert werden. Die Extension können via Konsole oder über den Ant-Task definiert werden.

```
<target name="create.stub" depends="clean">

  <wsimport

    sourcedestdir="${prod.source.dir}"

    destdir="${build.classes.dir}"

    debug="true"

    verbose="true"

    extension="true"

    wsdl="${wsdl.dir}/${wsdl.file}.wsdl"/>

</target>
```

Als Alternative zu den JAX-WS Extension können auch die Namen der Response-Messages umbenannt werden. Details dazu befinden sich im Kapitel 3.2.

2.3 Endpoint Adresse definieren

Der Endpoint für den Web Service kann über das BindingProvider Interface gesetzt werden.

```
((BindingProvider)
port).getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
          "http://<HOST>/<SERVICE-URL>");
```

2.4 Http Header

Http Header können über das BindingProvider Interface gesetzt werden. Dafür muss eine Map mit einer Liste von Werten erzeugt werden, die anschliessend in den RequestContext eingefügt werden.

```
Map<String, List<String>> headers = new HashMap<String, List<String>>();

List<String> values = new ArrayList<String>();

values.add("<KEY>");

headers.put("<VALUE>", values);

((BindingProvider)
port).getRequestContext().put(MessageContext.HTTP_REQUEST_HEADERS, head-
ers);
```

2.5 Holder Objekte für Attachments

Die Daten der Holder Objekte werden mit dem Encoding BASE64 zurückgeliefert und müssen vor dem Schreiben auf die Festplatte konvertiert werden.

Für die Konvertierung kann die Klasse org.apache.commons.codec.binary.Base64 verwendet werden. Die Klasse ist Teil des Projekts Apache Commons Codec (http://commons.apache.org/codec/).

```
out = new FileOutputStream(file);

Base64 decoder = new Base64();

byte[] decoded_data = decoder.decode(data);

out.write(decoded_data);

out.flush();

out.close();
```

2.6 Monitoring von Request/Response Daten

JAX-WS bietet die Möglichkeit über eine Systemparameter die Ausgabe der Request/Response Daten in der Konsole zu steuern.

com.sun.xml.ws.transport.http.client.HttpTransportPipe.dump=true

Als Alternative können auch die beiden Tools TCP Monitor oder WSMonitor verwendet werden. Die beiden Tools werden als Proxy zwischen dem Client und dem Server geschaltet.

2.7 Bekannte Probleme

2.7.1 Signaturprüfung mit JAX-WS Handler Implementation

In der aktuellen Version wird die Payload des SOAP-Responses inkl. dem Body-Tag signiert. Bei der Verwendung von JAX-WS Handlern bei der Überprüfung der Signatur gibt es Probleme weil der DOM-Tree von der Standard-Implementation verändert wird.

2.7.1.1 Vorgehen Fehleranalyse

Die folgenden Kapitel beschreiben das Vorgehen bei der Fehleranalyse.

HTTP-Dump erstellen

Mit dem Parameter aus Kapitel 2.6 kann ein Dump des HTTP-Responses erstellt werden. Dieser Dump enthält die Daten wie sie über das Protokoll geliefert werden.

Zeilenumbruch und zwei Leerschläge zw. <SOAP-ENV:Body..> und <ns1:receiptRequestResponse ..>

```
<SOAP-ENV:Body xmlns:wsu="http://docs.oasis-open ...>
  <ns1:receiptRequestResponse xmlns:ns1="http://www.e-
dec.ch/xml/schema/edecReceiptResponse/v1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" schemaVersion="0.6" xsi:schemaLocation="http://www.e-
dec.ch/xml/schema/edecReceiptResponse/v1
http://www.ezv.admin.ch/pdf_linker.php?doc=edecReceiptResponse_v_0_6">
   <requestorTraderIdentificationNumber xmlns="http://www.e-
dec.ch/xml/schema/edecReceiptResponse/v1">1000029</requestorTraderIdentificationNumber>
```

Zeilenumbruch und ein Leerschlag zw. </ns1:receiptRequestResponse> und </SOAP-ENV:Body>

```
  </ns1:receiptRequestResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

DOM-Tree Dump erstellen

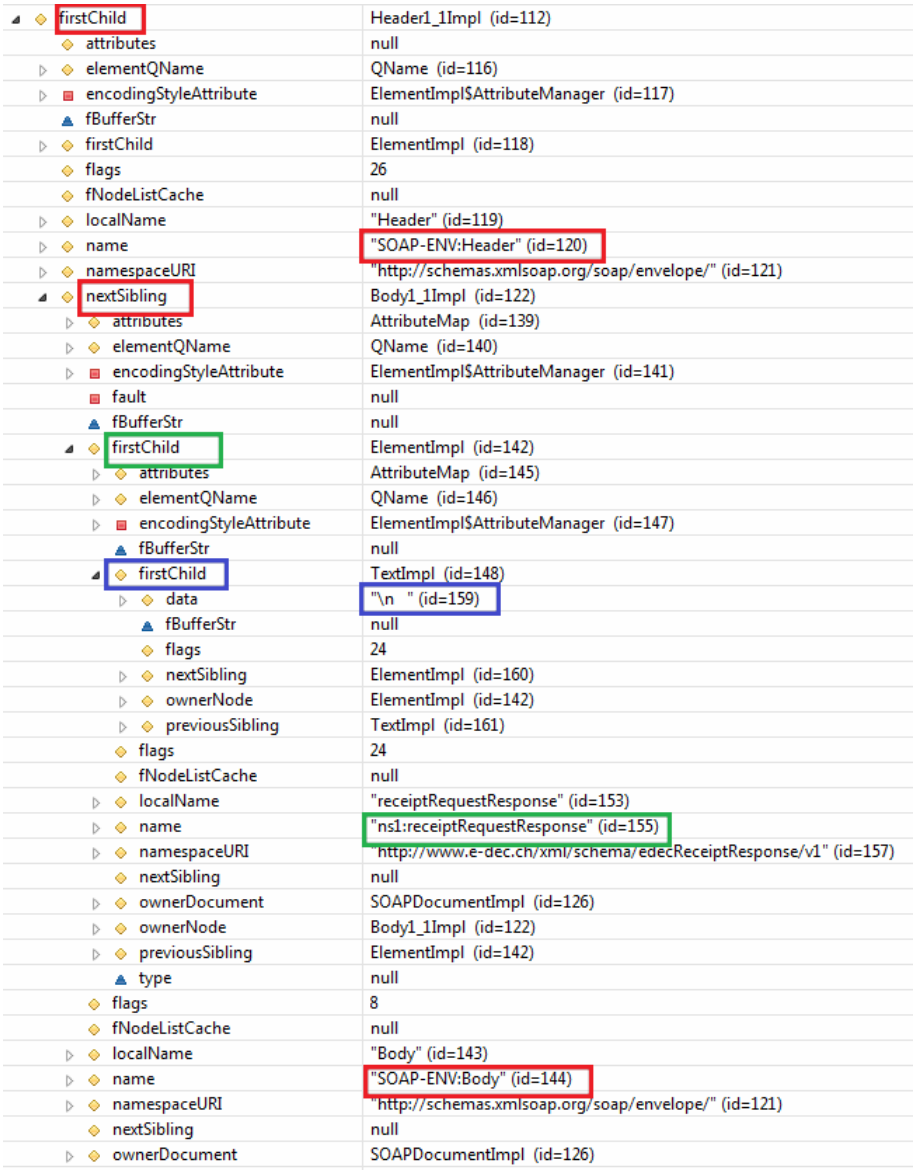
Mit dem folgenden Code kann ein Dump des DOM-Trees erstellt werden.

```
SOAPEnvelope envelope = smc.getMessage().getSOAPPart().getEnvelope();
  try{
    // Set up the output transformer
    TransformerFactory transfac = TransformerFactory.newInstance();
    Transformer trans = transfac.newTransformer();
    trans.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "no");
    trans.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
    trans.setOutputProperty(OutputKeys.INDENT, "no");
    // Print the DOM node
    StringWriter sw = new StringWriter();
    StreamResult result = new StreamResult(sw);
```

```
    DOMSource source = new DOMSource(envelope);
    trans.transform(source, result);
    String xmlString = sw.toString();
    System.out.println("DOM Begin");
    System.out.println("----------------------------------------");
    System.out.print(xmlString);
    System.out.println("----------------------------------------");
    System.out.println("DOM End");
    }
  catch (TransformerException e){
    e.printStackTrace();
  }
```

Debugdetails der Variablen envelope:

Das firstChild ist der Header (rot). Sein nextSibling ist der Body (rot). Das firstChild des Body ist der receiptRequestResponse (grün). Es fehlt eine TextNode mit dem Zeilenumbruch und den beiden Leerschlägen. Das firstChild von receiptRequestResponse ist eine TextNode mit einem Zeilenumbruch und drei Leerschlägen (blau) - was auch korrekt ist. Im Envelope werden beim Header und Body immer alle Leerschläge und Zeilenumbrüche entfernt bis zum ersten SubElement. Unterhalb dieser SubElemente sind alle Leerschläge und Zeilenumbrüche vorhanden.



DOM-Ausgaben auf der Konsole:

Bei der Ausgabe des DOM's werden keine Leerschläge oder Zeilenumbrüche zwischen Body und receiptRequestResponse eingefügt.

```
<SOAP-ENV:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" wsu:Id="id-16340840"><ns1:receiptRequestResponse xmlns:ns1="http://www.e-
dec.ch/xml/schema/edecReceiptResponse/v1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" schemaVersion="0.6" xsi:schemaLocation="http://www.e-
dec.ch/xml/schema/edecReceiptResponse/v1
http://www.ezv.admin.ch/pdf_linker.php?doc=edecReceiptResponse_v_0_6">
   <requestorTraderIdentificationNumber xmlns="http://www.e-
dec.ch/xml/schema/edecReceiptResponse/v1">1000029</requestorTraderIdentificationNumber>
```

2.7.1.2 Fazit

Das XML aus dem HHTP-Dump kann fehlerfrei validiert werden. Der Zugriff über den Envelope gibt einen bereits modifizierten DOM zurück. Mit diesem Ansatz ist es nicht möglich den Original Response (siehe http-Dump) auszulesen. Die Validierung des DOM-Outputs schlägt fehl.

2.7.1.3 Alternative Implementation:

Als Alternative zu JAX-WS und Handler könnte die Abfrage via createDispatch() des Services durchgeführt werden – Details siehe:

- http://blogs.sun.com/artf/entry/operating_at_the_xml_message

Diese Variante wurde noch nicht getestet

3 CXF

Bei CXF handelt es sich um einen Web Service Stack von Apache.

3.1 Links

Die folgenden Links enthalten Informationen zu CXF:

- Projekt Webseite - http://cxf.apache.org/

3.2 Generierung der Stub Klassen

Damit die Klassen mit CXF generiert werden können müssen bei den „multipart“-Operationen die Namen der Antwort-Parts in den Messages und Bindings umbenannt werden. (z.B. parameters → result).

```
<message name="goodsDeclarationsResponse">

   <part name="result" element="edecResponse:goodsDeclarationsResponse"/>

   …

</message>

…

<binding name="EdecBinding" type="tns:EdecPortType">

…

<output>

  <mime:multipartRelated>

    <mime:part>

      <soap:body use="literal" parts="result"/>

    </mime:part>

    …

   </mime:multipartRelated>

…

</output>
```

4 Apache AXIS2

Bei AXIS2 handelt es sich um einen Web Service Stack von Apache.

4.1 Links

Die folgenden Links enthalten Informationen zu AXIS2:

- Projekt Webseite
  http://ws.apache.org/axis2/

- SOAP with Attachments mit apache AXIS2:
  http://thilinag.blogspot.com/2007/05/using-soap-with-attachments-in-axis2.html
  http://wso2.org/library/1148

4.2 Http Header

Http Header können über die Options Klasse gesetzt werden. Dafür muss eine ArrayList angelegt werden, welcher anschliessend Objekte vom Typ Header hinzugefügt werden können.

```
EdecServiceStub stub = new EdecServiceStub(EDEC_SERVICE_URL);
Options options = stub._getServiceClient().getOptions();

List http_headers = new ArrayList();
http_headers.add(new Header("SSL_CLIENT_CERT_S_DN_CN", "Firma xy ABCDEF"));
options.setProperty(HTTPConstants.HTTP_HEADERS, http_headers);
```

5 C#

Mit C# gab es einige Problem, da weder mit WCF (Svcutil.exe) oder mit dem Tool Wsdl.exe einen brauchbaren Web Service Client Stub generiert werden konnte. Auch durch das modifizieren des WSDL (entfernen der SwA spezifischen Einträge) konnte auch kein brauchbaren Stub erstellt werden.

Eines der Probleme ist das, dass .Net Framework kein SwA (SOAP Messages with Attachments) unterstützt.

Deshalb wurde ein C# Client erstellt der selber ein SOAP Request zusammenstellt und mit dem MIME Parser SharpMimeTools die Attachments extrahiert.

Nachfolgend sind die wichtigsten Schritte dokumentiert. Eine Beispiel Applikation wurde mit C# Visual Studio Express 2008 erstellt.

5.1 Links

Die folgenden Links enthalten zusätzliche Informationen:

- .Net Framework - http://www.microsoft.de/net
- Visual Studio Express – http://www.microsoft.com/Express/
- SharpMimeTools - http://anmar.eu.org/projects/sharpmimetools/

5.2 Marshalling / Unmarshalling in C#

Mit dem Programm Xsd.exe kann man aus den Schemas C# Klassen generieren. Die erstellten C# Klassen können dann mit dem XmlSerializer serialisiert oder deserialisiert werden.

```
Xsd.exe edec_v_2_0.xsd edecSelectionAndTransit_v_1_0.xsd
      edecResponse_v_2_0.xsd /c /o:c:\out
```

Durch den Parameter /o wird angegeben wohin die generierten Klassen kopiert werden sollen.

5.3 SOAP Request versenden

Mit dem XmlSerializer kann man nun die Objekte in XML deserialisieren. Danach muss man nur noch die SOAP Spezifischen Elemente (SOAP Envelope, SOAP Body und SOAP Header) hinzufügen und mittels der Klasse HttpWebRequest an die entsprechenden Endpoint URL versenden.

```
FileStream fs = new FileStream("C:\data\export.xml", FileMode.Open);

XmlSerializer serializer = new XmlSerializer(typeof(goodsDeclarations));

goodsDeclarations declarations = (goodsDeclarations)
      serializerReq.Deserialize(fs);
```

```
// here you could modify goodsDeclarations

XmlWriterSettings writerSettings = new XmlWriterSettings();

writerSettings.OmitXmlDeclaration = true;

StringWriter stringWriter = new StringWriter();

using (XmlWriter xmlWriter = XmlWriter.Create(stringWriter,
      writerSettings))

      {

                serializerReq.Serialize(xmlWriter, declarations);
      }



string xmlText = stringWriter.ToString();

HttpWebRequest req = (HttpWebRequest)WebRequest.Create(url);

req.ContentType = @"text/xml;charset=""utf-8""";

req.Accept = "text.xml";

req.Method = "POST";

// create soap message (with SOAP Header, Body and Envelope)

String msg = START_MSG + xmlText + END_MSG;

byte[] reqBytes = System.Text.Encoding.UTF8.GetBytes(msg);

req.ContentLength = reqBytes.Length;

Stream reqStream = req.GetRequestStream();

// send soap request

reqStream.Write(reqBytes, 0, reqBytes.Length);

reqStream.Close();
```

5.4 SOAP Request empfangen

Das Problem beim Request ist, dass wir eine MIME Multipart Response erhalten.

Mit der Library SharpMimeTools kann man den Response Stream einer HttpWebResponse parsen. Der Response Stream sieht wie folgt aus.

```
------=_Part_118_24290992.1232636587322
```

```
Content-Type: text/xml
Content-Description: e-dec_Export_edecResponse_524536839_09CHEE000000507663_1_1000054_1
Content-ID: <parameters=-15248ca1:11efdd317b9:-7cc2_0@edec.ezv.admin.ch>

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
       xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">   <SOAP-ENV:Header/>
   <SOAP-ENV:Body>
</…>
------=_Part_118_24290992.1232636587322
Content-Type: application/pdf
Content-Transfer-Encoding: base64
Content-Description: e-dec_Export_AL_524536839_09CHEE000000507663_1_1000054_1
Content-ID: <e-dec_Export_AL=-15248ca1:11efdd317b9:-7cc3_1@edec.ezv.admin.ch>

JVBERi0xLjQKJeLjz9MKOCAwIG9iaiA8PC9GaWx0ZXIvRmxhdGVEZWNvZGUvTGVuZ3RoIDgzOT4+
…
------=_Part_118_24290992.1232636587322--
```

Damit der Response Stream der HttpWebResponse Klasse MIME konform ist, muss man noch einige MIME Headers hinzufügen (Date, Content-Type und Content-Length). Diese erhält man aus den HTTP Header der HttpWebResponse.

```
Date: Thu, 22 Jan 2009 15:03:04 GMT
Content-Type: multipart/related; boundary="----=_Part_118_24290992.1232636587322"
Content-Length: 7732
```

Anschliessend kann man den Request Stream mit der Klasse SharpMessage, aus der Library, MimeSharpTools parsen.

```
HttpWebResponse res = …;
MemoryStream httpStream = …;
MemoryStream mimeStream = …;

Encoding utf8 = Encoding.UTF8;

TextReader readerReq = new StreamReader(httpStream, utf8);
TextWriter writer = new StreamWriter(mimeStream, utf8);


// create a correct mime stream form the HttpResponseStream
// add http headers which were required for a correct mime
// format
writer.WriteLine("Date: " + res.GetResponseHeader("Date"));
writer.WriteLine("Content-Type: " + res.ContentType);
writer.WriteLine("Content-Length: " + res.ContentLength);
writer.WriteLine(Environment.NewLine);

// copy rest of the stream
while (true)
{
      string line = readerReq.ReadLine();
      if (line == null)
```

```
      {
                break;

      }
      writer.WriteLine(line);
}
writer.Flush();
mimeStream.Position = 0;
// parse  the stream for mime attachments
SharpMessage message = new SharpMessage(mimeStream,
   SharpDecodeOptions.Default | SharpDecodeOptions.DecodeTnef |
   SharpDecodeOptions.UuDecode);
```

5.5 SOAP Reuqest und Attachment extrahieren

Auf Sämtliche Daten (SOAP Response und Attachments) kann man mittels der Klasse SharpMessage zugreifen.

```
SharpMessage message = …;

if (message.Attachments != null)
{
  foreach (attachment in message.Attachments)
  {
    Stream stream = attachment.Stream;
    …
  }
```

5.6 Client Zertifikat

Unter der folgenden Microsoft Support Seite wird erklärt, wie man einem WebRequest ein Client Zertifikat mitgeben kann.

http://support.microsoft.com/kb/895971

6 Tools

6.1 Monitoring

Name	Link
TCP Monitor	https://tcpmon.dev.java.net/
WSMonitor	https://wsmonitor.dev.java.net/