Laverca

Technical description

Eemeli Miettinen (eemeli@methics.fi)
Jan 2017

Table of contents

History

Versio	Date	Who	What
1.0	2011-06-28	Asko Saura, Methics Oy	The first version.
1.2	2013-07-23	Eemeli Miettinen	Updated.
1.3	2017-01-01	Eemeli Miettinen	Translation
1.4	2017-06-20	Eemeli Miettinen	Fixed translation errors
1.5	2017-06-29	Atte Walden, Matti Aarnio	Keytool instructions

1 Overview

The Laverca library is an Open Source Java implementation of ETSI TS 102 204-client library. In addition to the standard ETSI interface the Laverca includes also FiCom 2.0 integration interface functionality. It aims to offer the most common FiCom functions easily for the developer – especially asynchronous communication mode and FiCom services are done so that they look like any other Laverca interface.

This document describes Laverca technical Java library implementation. This paper was originally only in Finnish because the FiCom guidelines were only in Finnish. Now the FiCom translation is ready and this document was translated too.
.

2 Architecture

The Laverca has been done by the book as the Java XML/SOAP implementation guidelines define. This means that Releases 1.X is based on Castor XML marshalling and Release 2 is based on JAXB XML marshaling.

About release 1

The main focus has been in supporting API defined in the FiCom AP intergation guide. There is a special layer for this API.

Layered architecture was selected to be able to support future development and library tailoring by other projects. Laverca supports completely the FiCom API and allows developers to use also ETSI interface directly.

Lower layers are straight forward implementations. The implementation tries just to follow standards. The base principle in Laverca is to hide C/S messaging by using Java standard tools (java.util.concurrent).

3 FiCom-layer

The FiCom API is closest to a developer aka FiCom layer. It provides only a few methods, which are actually the same as FiCom integration guide's five signature profiles:

- authenticate(..)
- authenticateAnon(..)
- signText(..)
- signData(..)
- consent(..)

The idea in this layer is to provide a developer high level functions, which hide ETSI details. The main detail here is C/S asynchronous messaging mode. Developer sees the layer as a asynchronous method call.

```
import fi.laverca.*;
import org.etsi.uri.TS102204.v1_1_2.Service;
...

String apId  = "http://my_ap_id";
String apPwd = "my_ap_pass";
String sigUrl = "https://www.myae.fi/soap/services/MSS_SignaturePort";
String statUrl= "https://www.myae.fi/soap/services/MSS_StatusQueryPort";
String statUrl= "https://www.myae.fi/soap/services/MSS_ReceiptPort";

FiComClient fiComClient = new FiComClient(apId, apPwd, sigUrl, statUrl, recUrl);
.. setup SSL

String phoneNumber = .. ask user for it ..
String nospamCode = .. ditto ..
String apTransId = .. generate unique id for this transaction ..
String challenge = .. generate authentication challenge ..
String eventId = .. generate unique id for this transaction ..

Service noSpamService = FiComAdditionalServices.createNoSpamService(.., false);
Service eventIdService = FiComAdditionalServices.createEventIdService(eventId);

try {
    FiComRequest req = fiComClient.authenticate(apTransId,
                                                challenge,
                                                phoneNumber,
                                                noSpamService,
                                                eventIdService,
                                                null, // additionalServices,
                                                new FiComResponseHandler() {
        @Override
        public void onResponse(FiComRequest req, FiComResponse resp) {
           log.debug("got resp");
           log.debug(resp.getPkcs7Signature().getSignerCn());
        }
        @Override
        public void onError(FiComRequest req, Throwable throwable) {
            log.debug("got error", throwable);
        }
        @Override
        public void onOutstandingProgress(FicomRequest req, ProgressUpdate prg) {
            log.debug("got progress update");
        }
    });
}
catch (IOException e) {
    log.debug("error establishing connection", e);
}
```

4 ETSI-layer

The technical core of Laverca lies under FiCom layer. The core implements ETSI TS 102 204 data types and SOAP calls. Here is the mechanism which allows developer to produce ETSI TS 102 204 traffic. The design and implementation follow standards and there should be no "advanced" coding.

The example below illustrates how developer can implement a synchronous signature transaction.

```
import fi.laverca.*;
...

String apId  = "http://my_ap_id";
String apPwd = "my_ap_pass";
String sigUrl = "https://www.myae.fi/soap/services/MSS_SignaturePort";
String statUrl= "https://www.myae.fi/soap/services/MSS_StatusQueryPort";
.. all the other URLs

EtsiClient etsiClient = new EtsiClient(apId,
                                       apPwd,
                                       aeMsspIdUri,
                                       msspSignatureUrl,
                                       msspStatusUrl,
                                       msspReceiptUrl,
                                       msspRegistrationUrl,
                                       msspProfileUrl,
                                       msspHandshakeUrl);

String apTransId = "A"+System.currentTimeMillis();
String msisdn = "+35847001001";
DTBS dtbs = new DTBS("sign this", "UTF-8");
String dataToBeDisplayed = null;
String signatureProfile = FiComSignatureProfiles.SIGNATURE;
String mss_format = FiComMSS_Formats.PKCS7;
MessagingModeType messagingMode = MessagingModeType.SYNCH;

MSS_SignatureReq sigReq =
  etsiClient.createSignatureRequest(apTransId,
                                    msisdn,
                                    dtbs,
                                    dataToBeDisplayed,
                                    signatureProfile,
                                    mss_format,
                                    messagingMode);

MSS_SignatureResp sigResp = null;
try {
    sigResp = etsiClient.send(sigReq);
}
catch(AxisFault af) {
    log.error("got soap fault", af);
    return;
}
catch(IOException ioe) {
    log.error("got IOException ", ioe);
    return;
}
```

Running example can be found here:
~/examples/src/fi/laverca/examples/EtsiSigReqCaller.java

5 Other Layers

SOAP implementation is based on Apache Axis 1.4 ja Castor 1.3 (release 1.x)/JAXB (release 2). Axis takes care of SOAP traffic and Castor/JAXB produces Java-XML-Java marshaling.

Axis uses external Apache HttpClient library for HTTP connections.

SSL/TLS implemenation is based on standard Java SSL support. In practice developer can define SSL identity by using environment variables and define trusted certificates in a Truststore. The class fi.laverca.JvmSsl implements the laverca SSL support.

6 Packaging

Laverca distribution contains the following parts:

- laverca-core.jar
- libs -folder libraries
- examples
- API docs
- this document
- MSS FiCom implementation guideline 2.1

The package contains also contact details for testing server maintained by Methics.

7 License and Dependencies

Laverca license is Apache Version 2.0 . Laverca utilizes various open source libraries. Software distribution is limited by those licenses.

Here we have identified only the most important libraries. Complete list of licenses is in a directory docs/licenses/.

Library	License	Note
Apache Axis	©Apache Software Foundation Apache License, Version 2.0	axis-src-1_4.tar.gz/LICENSE
Apache HttpClient	©Apache Software Foundation Apache License, Version 2.0	http://hc.apache.org/

8 Step-by-step Instructions Keystore Operations

Follow these steps to add your AP to a Kiuru MSSP.
These steps use the Java keytool to manage keystore and certificate.

1. Create a keystore and a self-signed certificate. Replace the value of `<days>` by how many days you want the certificate to be valid. When asked, insert your information to keytool. Use value "3660" for 10 years.
   This tool will be asking required information for the certificate Subject value. You may also supply the Subject value in command line, see '`-dname`' option in keytool documentation.

   ```
   keytool -genkey -keyalg RSA -alias laverca-1 -keysize 2048
           -keystore selfsigned.jks -validity <days>
   ```

2. Verify the certificate by listing the keystore content. You should see the certificate you created in step 1.

   ```
   keytool -list -v -keystore selfsigned.jks
   ```

3. Export certificate in PEM format. This creates a PEM certificate file `laverca_test.pem`.

   ```
   keytool -exportcert -keystore selfsigned.jks -rfc
           -alias laverca-1 -file laverca_test.pem
   ```

4. Send the `laverca_test.pem` to your AE manager, get AP_ID and AP_PWD from your AE manager.

   4.1. Alternatively you can determine your own AP_ID and AP_PWD value if it agrees with your AE manager's policy.

   4.2. AP_ID could be in format
   uri://<your_domain_name>

   4.3. AP_ID could be in format
   uri://aps.ae-operator.dom/<your_domain_name>

   4.4. AP_PWD is typically "x", and it is used as mandatory space filler. It is not used for security verification!

5. Edit the Laverca `examples.conf` configuration file.

   5.1. Set ssl.keystore.file to point to `selfsigned.jks`.

   5.2. Set ssl.keystore.password to the keystore password.

   5.3. Set ssl.keystore.type to JKS.

   5.4. Set ap.id to your AP_ID created in step 4.

   5.5. Set ap.password to your AP_PWD created in step 4.