

OCSSW Web Services Developer Manual

Aynur Abdurazik

March 29, 2016

Daily Progress

To Do

- Progress Monitor
 - Save process input stream, error stream, and output stream on the server side database
 - Access process information from the client using pull mechanism
 - test running a few processors
- Test l2gen
- Test multi-level processor

Learning Objectives

- Progress Monitor
- regular expression for single data with percentage information

1 Packaging and Deployment

1.1 Packaging

The *ocssw client-server module* is independent of the rest of SeaDAS. The jar file that will be deployed on the ocsw server is packaged using the following command in the \$SEADAS_HOME/seadas/seadas-ocswrest directory:

```
mvn install assembly:assembly
```

2 OCSSW Server Side Representation

2.1 Execution

The jar file is packaged on a developer machine and deployed on the server.

Run the following command from the command line to execute:

```
java -Xmx2048m -jar seadas-ocsswserver-jar-with-dependencies.jar
```

3 Virtual Box Configuration

3.1 Basic Configuration

1. Need to install “guest editions” to be able to resize the vm window.

3.2 File Sharing

- Manually sharing a folder between host and guest machines:

In VirtualBox **Devices** → **Shared Folder Settings...** → **Shared Folders** → **Machine Folders**, select the folder from the host to be shared with the guest.

1. `sudo rm /sbin/mount.vboxsf`
2. `sudo ln -s /opt/VBoxGuestAdditions-4.3.20/lib/VBoxGuestAdditions/mount.vboxsf /sbin/mount.vboxsf`
3. `mkdir ocsswws`
4. `sudo mount -t vboxsf seadas-ocsswws /home/aabduraz/ocsswws`
5. To be able to write in the shared directory, it needs to be mounted in this way:
`sudo mount -t vboxsf -o uid=1000,gid=1000 seadas-ocsswrest /home/aabduraz/ocsswrest`

- Commands to manually mount a directory:

```
sudo mount -t vboxsf seadas-ocsswrest /home/aabduraz/ocsswrest
```

where *seadas – ocsswrest* is the name of folder, which has the development source code for web services, shared from the host machine, and */home/aabduraz/ocsswrest* is an empty folder in the virtual machine. The *seadas – ocsswrest* is shared to deploy the jar file from its *target* directory after each build.

- need to install git (error message:Error - Could not execute system command "git -version > /dev/null")

3.3 Network Configuration

1. The server must use 0.0.0.0 as its IP address.
2. The client should still use *localhost*
3. The virtual machine uses “NAT” port-forwarding, which is set through [Devices → Network → Network Settings ...](#) .
4. Between SeaDAS and OCSSWWS, we chose to use port number 6400 and 6401. The server side presents services using address “0.0.0.0 : 6401”, and a SeaDAS client will access the services using “*http : //localhost : 6400*” .

4 Security Concepts and Implementation

4.1 Security Concepts

4.1.1 Java Keystore

Java keystore is a repository of security certificates. JDK provides a tool named `{keytool}` to manipulate keystores. Java `keytool` stores the keys and certificates in a keystore, protected by a keystore password.

TrustManager: Determines whether the remote authentication credentials *and the connection* should be trusted.

KeyManager: Determines which authentication credentials to send to the remote host.

4.1.2 Security Key Generation

1. Create a keystore for server

```
keytool -genkey -alias server -keyalg RSA -keystore server.jks
```

My password for server keystore is "ocsswsws". The generated file is "server.jks".

2. Create a keystore for client

```
keytool -genkey -alias client -keyalg RSA -keystore client.jks
```

My password for server keystore is "ocsswswsclient". The generated file is "client.jks".

3. View the content of keystore files:

```
keytool -list -v -keystore server.jks -storepass ocsswsws
keytool -list -v -keystore client.jks -storepass ocsswswsclient
```

4. Get server's self signed public key certificate and store it in client's keystore.

```
keytool -export -file server.cert -keystore server.jks -storepass ocsswsws -alias server
```

5. Get client's self signed public key certificate and store it in server's keystore.

```
keytool -export -file client.cert -keystore client.jks -storepass ocsswswsclient -alias client
```

Note: First we needed to export both server and client public key certificates into files.

6. Use following commands to view certificate contents.

```
keytool -printcert -v -file server.cert
keytool -printcert -v -file client.cert
```

7. As the last step, import server.cert into client keystore and client.cert into server keystore.

- store client's self signed public key certificate(client.cert) in server.jks against the alias "client".

```
keytool -import -file client.cert -keystore server.jks -storepass ocsswws -alias client
```

- store server.cert within client.jks against the alias "server".

```
keytool -import -file server.cert -keystore client.jks -storepass ocsswwsclient -alias server
```

8. View the content of both keystore again using following commands.

```
keytool -list -v -keystore server.jks -storepass ocsswws
keytool -list -v -keystore client.jks -storepass ocsswwsclient
```

4.1.3 Setting up SSL Configuration on OCSSW (Jersey) Client

The SSL configuration is setup in the ClientBuilder class. The client builder contains methods for definition of KeyStore, TrustStore or entire SslContext.

- *KeyStore* - Represents a storage facility for cryptographic keys and certificated; Manages different types of entries. The keystore in javax.net.ssl.keyStore contains private keys and certificates.
- *TrustStore* - The *javax.net.ssl.trustStore* contain CA certificates that a server trusts when a remote party presents its certificate.
- *SslContext* -

```
SslConfigurator sslConfig = SslConfigurator.newInstance()
    .trustStoreFile("truststore.jks")
    .trustStorePassword("asdfgh")
    .trustStoreType("JKS")
    .trustManagerFactoryAlgorithm("PKIX")

    .keyStoreFile("keystore.jks")
    .keyPassword("asdfgh")
    .keyStoreType("JKS")
    .keyManagerFactoryAlgorithm("SunX509")
    .keyStoreProvider("SunJSSE")

    .securityProtocol("SSL");

SSLContext sslContext = sslConfig.createSSLContext();
```

Daily Progress

To Do

- Configure SSLContext and SslConfigurator
 - have an architectural design document
- Repackage ocswrest
- deploy on virtual box

Learning Objectives

- SSLContext
- SslConfigurator

To Do List for OCSSW Web Services

1. Figure out the secure communication configuration between server and client;
 - document all concepts
 - document directory structure and location of keystores

Resources

1. This site provides information about problems with host identification:

`http://java.globinch.com/enterprise-java/
security/fix-java-security-certificate-exception-no-matching-localhost-found/`