Java Reference Code for

NemLog-in STS Integration

(System User Scenario)

(Bootstrap Scenario)

(Signature Scenario)

OIO IDWS 1.1 Profile

Status: Version 1.2

Version: 17.07.2018

[Changelog 3](#_Toc519685068)

[1 Introduction 4](#_Toc519685069)

[1.1 Intended audience 5](#_Toc519685070)

[1.2 Prerequisites 5](#_Toc519685071)

[1.3 Apache CXF Version 5](#_Toc519685072)

[1.4 Disclaimer 5](#_Toc519685073)

[2 Differences between OIO IDWS 1.1 and Liberty Basic SOAP Binding 6](#_Toc519685074)

[2.1 Update to SOAP 1.2 6](#_Toc519685075)

[2.2 Configuring a secure transport mechanism (TLS 1.2) 7](#_Toc519685076)

[2.3 Remove Framework header 8](#_Toc519685077)

[2.4 Changes for interoperability with .NET reference code 9](#_Toc519685078)

[2.5 Removed SOAP Fault example 11](#_Toc519685079)

[3 Example payloads 12](#_Toc519685080)

[3.1 WSC-TO-STS 12](#_Toc519685081)

[3.2 STS-TO-WSC 12](#_Toc519685082)

[3.3 WSC-TO-WSP 13](#_Toc519685083)

[3.4 WSP-TO-WSC 13](#_Toc519685084)

[4 References 14](#_Toc519685085)

# Changelog

01-12-2015 Initial release

09-03-2018 Updated with OIO IDWS profile, and updated certificates

17-07-2018 Extracted general CXF and WS-SecurityPolicy information into separate document

17-07-2018 Extracted OIO IDWS profile documentation into separate document

# Introduction

This document is a companion to the Java reference source code that showcase how to use the Apache CXF framework to implement a OIO IDWS SOAP Profile 1.1 solution consisting of

* A Web Service Provider (WSP) that requires clients to present a token issued by the NemLog-in STS, showing how to perform the following validations
  + Require the client to conform to the OIO IDWS SOAP Profile 1.1 [OIOIDWS]
  + Parse the issued Basic Privilege Profile [OIO-BPP] information in the presented token
  + Ensure that the request is no older than 5 minutes
* A Web Service Consumer (WSC) that can
  + Interact with the NemLog-in STS to get a token issued
  + Call the above WSP with the issued token
  + Validate that the response from the WSP conforms to the OIO IDWS SOAP Profile 1.1 [OIOIDWS]

Note that the NemLog-in STS supports a number of different usage scenarios that differ in how authentication to the STS is performed, including bootstrap token case, local token case and signature case illustrated below:



This reference code only covers the following scenarios

1. The system user scenario where the WSC authenticates to the STS as a system user using a signature (e.g. a FOCES or VOCES certificate). Here, the WSC gets a token to act as the WSC (a system user).
2. The bootstrap token scenario, where the WSC is also a SAML 2.0 Service Provider, integrated with the NemLog-in SSO. Here the WSC authenticates to the STS using a signature (e.g. a FOCES or VOCES certificate) and exchanges a bootstrap token (representing the user) received during the Web SSO roundtrip. The WSC gets a token to act on behalf of the end-user.
3. The signature scenario where the WSC authenticates to the STS as a user using the users signature (e.g. a MOCES certificate). The WSC gets a token to act on behalf of the end-user.

Please refer to the NemLog-in documentation for descriptions of the other usage scenarios.

The Apache CXF specific configuration is covered in [CXF], and it is recommended to read that document first.

This document will refer to the reference code a lot, so it is also recommended to have the reference code available while reading this document.

## Intended audience

This document is written for developers, and while all configuration and customization of Apache CXF concerning security is dealt with, some experience with Apache CXF or a similar web service framework is recommended. The reader is also expected to have experience with Java development in general.

## Prerequisites

The source code uses Apache Maven 3 [MAVEN] as a build tool, and the source code requires at least Java 7 with Strong Crypto [CRYPTO] to compile and run. The reader is expected to have these tools available before using the reference source code. Maven handles all other dependencies.

## Apache CXF Version

The reference code is based on Apache CXF 3.0.16, but the same concepts outlined in this document can be applied to the 3.1.x and 3.2.x branches of CXF.

## Disclaimer

The Danish Agency for Digitisation provides the reference code as is and assumes no responsibility for the code by service providers. Service Providers should understand the limitations of the code and deal with these according to their own needs.

# Differences between OIO IDWS 1.1 and Liberty Basic SOAP Binding

The Liberty Basic SOAP Binding (LBSB) profile has been replaced by the Danish profile OIO IDWS Profile version 1.1, which is at its core a revision of LBSB.

The major changes are

* Requires a secure transport mechanism (i.e. TLS 1.2 or better)
* Is based on SOAP 1.2 instead of SOAP 1.1
* No longer requires the use of the Framework SOAP header
* Does not govern the use of SOAP Faults

The change to SOAP version, and the omission of the Framework header will affect service contract (WSDL) of a service that migrates from LBSB to OIO IDWS, so please keep this in mind when upgrading.

The code that accompanies this document resides in the “oioidws-scenarios” folder, which contains the same scenario implementations as in the “lbsb-scenarios”, just with different security profile implementations.

As the LBSB and OIO IDWS reference code is almost identical, instead of creating an identical document with just a few changes, this document only contains the documentation of the differences between the LBSB and OIO IDWS implementations.

Start by reading the LBSB documentation, but use the OIO IDWS reference code, combined with the rest of this document, which describes the changes applied to the LBSB reference code to make it the OIO IDWS reference code.

## Update to SOAP 1.2

The reference code is written as contract-first, so the WSDL governs how CXF performs SOAP calls, including the version of SOAP used.

The following change was made to all the WSDL files used in the example code

Replace this

xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"

with this

xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"

We also need to tell the WSP implementation about this, so add this @BindingType annotation on top of the Java class (HelloWorldPortType.java) that implements the service endpoint (in the service-hok and service-bearer projects)

@BindingType(value = javax.xml.ws.soap.SOAPBinding.SOAP12HTTP\_BINDING)

public class HelloWorldPortTypeImpl implements HelloWorldPortType {

No changes are required on the WSC, it should just be pointed at the updated WSDL file, and it will start using SOAP 1.2.

## Configuring a secure transport mechanism (TLS 1.2)

While existing LBSB implementations in production will already use a secure transport mechanism (SSL/TLS), the LBSB reference code run on HTTP without any transport layer security.

As the OIO IDWS Profile 1.1 mandates the use of a secure transport mechanism, the reference code has been updated to illustrate how this could be done using Tomcat.

In a production setting, this is likely handled by whatever application-server, proxy or loadbalancer that hosts the application.

**The following changes were made to the WSP projects**

1. An SSL certificate was added to the project

A self-signed certificate was used, stored in the keystore filed named ‘ssl-keystore’.

1. The Maven plugin in the pom.xml was configured to use the keystore

<configuration>

<port>0</port>

<httpsPort>8443</httpsPort>

<keystoreFile>${project.basedir}/ssl-keystore</keystoreFile>

<keystorePass>Test1234</keystorePass>

</configuration>

Note that the Tomcat plugin used by Maven does not allow configuring the TLS version, but this is doable by setting the following properties directly in Tomcat in a production setting

<sslEnabledProtocols>TLSv1.2</sslEnabledProtocols>

<sslProtocol>TLSv1.2</sslProtocol>

1. The endpoint in the WSDL file was updated to the following value to reflect the changed Tomcat settings

<soap:address location="https://localhost:8443/HelloWorld/services/helloworld" />

**The following changes were made to the WSC projects**

The WSC projects need to use the updated WSDL, so they know about the changes to the address, and then they need to trust the SSL certificate used by the WSP.

In a production setting, the SSL certificate used will likely be issued by some CA that is trusted by the WSC, but as the example code uses a self-signed certificate, a trust-store called ‘ssl-trust.jks’ is added to the WSC projects, which contains the SSL certificate used by the WSP.

The following hardcoded trust is then added to the startup of the code (not recommended for production settings – use the application-servers truststore or used SSL certificates issued by a trusted CA)

System.setProperty("javax.net.ssl.trustStore",

"src/main/resources/ssl-trust.jks");

System.setProperty("javax.net.ssl.trustStorePassword",

"Test1234");

This will ensure that the WSC can establish an SSL connection to the WSP without complaining about not trusting the certificate.

## Remove Framework header

As the Framework header from the LBSB profile is not a part of the OIO IDWS profile, we simply remove the configuration and code that deals with this header from both the WSC and WSP

**The following changes are made to the WSP**

The following Java classes are removed from the codebase

* FrameworkHeaderInterceptor
* UnderstandFrameworkHeaderInterceptor
* SbFrameworkHeader

The first two classes are responsible for adding the Framework header on responses, and validating the Framework header on requests, and the last is just the model class for marshelling to/from XML.

As we have removed the classes, we need to remove them from the cxf-servlet.xml file, where they are configured to be used – so remove these interceptor configuration sections

<cxf:inInterceptors>

<ref bean="UnderstandFrameworkHeaderInterceptor" />

</cxf:inInterceptors>

<cxf:outInterceptors>

<ref bean="FrameworkHeaderInterceptor" />

</cxf:outInterceptors>

<cxf:outFaultInterceptors>

<ref bean="FrameworkHeaderInterceptor" />

</cxf:outFaultInterceptors>

And these two bean definitions

<bean id="FrameworkHeaderInterceptor"

class="service.interceptor.FrameworkHeaderInterceptor" />

<bean id="UnderstandFrameworkHeaderInterceptor"

class="service.interceptor.UnderstandFrameworkHeaderInterceptor" />

**The following changes are made to the WSC**

Similar changes are made to the WSC projects, where the following Java classes are deleted

* FrameworkHeaderInterceptor
* SbFrameworkHeader

As they are used for setting the Framework header in the SOAP requests.

And the cxf.xml configuration file is updated, by removing these interceptors and beans in a similar fashion as the updates performed on the WSP

<jaxws:outInterceptors>

<ref bean="FrameworkHeaderInterceptor" />

</jaxws:outInterceptors>

<bean id="FrameworkHeaderInterceptor"

class="service.interceptor.FrameworkHeaderInterceptor" />

## Changes for interoperability with .NET reference code

During interoperability testing with the .NET reference code for OIO IDWS, a few changes were made to the WS-SecurityPolicy section of the WSDL for the HelloWorld service.

These changes make it possible for the .NET frameworks SVCUTIL tool to parse the policy section of the WSDL without errors, without changing the behaviour of the Java reference implementations.

These changes are as follows

1. Declare the following namespace in the top of the WSDL

xmlns:wsap="http://www.w3.org/2006/05/addressing/wsdl"

1. Change the <InitiatorToken> specification

The SVCUTIL from the .NET framework does not understand the <SamlToken> element, so it is replaced with a <Issuedtoken> element, with a <RequestSecurityTokenTemplate> that sets the type of the issued token to SAML

The original configuration is changed from

<sp:InitiatorToken>

<wsp:Policy>

<sp:SamlToken sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/Never">

<wsp:Policy>

<sp:WssSamlV20Token11 />

</wsp:Policy>

</sp:SamlToken>

</wsp:Policy>

</sp:InitiatorToken>

To the following value

<sp:InitiatorToken>

<wsp:Policy>

<sp:IssuedToken sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/Never">

<sp:RequestSecurityTokenTemplate>

<wsap:KeyType>

http://docs.oasis-open.org/ws-sx/ws-trust/200512/Symmetric

</wsap:KeyType>

<wsap:TokenType>

http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0

</wsap:TokenType>

</sp:RequestSecurityTokenTemplate>

<wsp:Policy/>

</sp:IssuedToken>

</wsp:Policy>

</sp:InitiatorToken>

1. Change the <SignedSupportingTokens> specification

In a similar way, the SignedSupportingTokens element is updated, so the SVCUTIL tool can parse the policy, this is done by replacing this value

<sp:SignedSupportingTokens xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">

<wsp:Policy>

<sp:IssuedToken sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient">

<sp:RequestSecurityTokenTemplate />

<wsp:Policy>

<sp:WssSamlV20Token11 />

</wsp:Policy>

</sp:IssuedToken>

</wsp:Policy>

</sp:SignedSupportingTokens>

With this value

<sp:SignedSupportingTokens xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">

<wsp:Policy>

<sp:IssuedToken sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient">

<sp:RequestSecurityTokenTemplate>

<wsap:KeyType>

http://docs.oasis-open.org/ws-sx/ws-trust/200512/Symmetric

</wsap:KeyType>

<wsap:TokenType>

http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0

</wsap:TokenType>

</sp:RequestSecurityTokenTemplate>

<wsp:Policy/>

</sp:IssuedToken>

</wsp:Policy>

</sp:SignedSupportingTokens>

## Removed SOAP Fault example

As LBSB also covers SOAP Faults, the LBSB reference code includes an example on how to deal with SOAP Faults. OIO IDWS does not govern how SOAP Faults are handled, so this is an implementation detail that is left up to the individual solution.

To not give the impression that SOAP Faults must be handled in a specific way, the OIO IDWS reference code is stripped of SOAP Fault handling.

When migrating an existing LBSB based solution to OIO IDWS, it is safe to keep whatever methods are used for dealing with SOAP Faults, as OIO IDWS does not set any requirements.

# Example payloads

The reference code has trace logging enabled on both the client and the service, so by running the reference code, it is possibly to recreate example payloads mentioned in this chapter.

Full traces can be found in the “traces/OIOIDWS” folder inside the “doc” folder. There are separate folders for each scenario, with a full trace for each request and corresponding response.

The following files are available for inspection

├── bootstrap-scenario

│   ├── 1-WSC-TO-STS.XML

│   ├── 2-STS-TO-WSC.XML

│   ├── 3-WSC-TO-WSP.XML

│   └── 4-WSP-TO-WSC.XML

├── signature-scenario

│   ├── 1-WSC-TO-STS.XML

│   ├── 2-STS-TO-WSC.XML

│   ├── 3-WSC-TO-WSP.XML

│   └── 4-WSP-TO-WSC.XML

└── system-user-scenario

├── 1-WSC-TO-STS.XML

├── 2-STS-TO-WSC.XML

├── 3-WSC-TO-WSP.XML

└── 4-WSP-TO-WSC.XML

In each scenario, 4 xml files are available.

## WSC-TO-STS

These xml files contain the requests to the STS, and the main difference between the files is the contents of the soap-body. The soap-headers are almost identical, differing only in the wsa:To field, that decides which scenario on the STS is being called.

The soap-body for the bootstrap scenario contains an ActAs element not found in the other scenarios, and in the signature scenario, the request is signed with a MOCES certificate rather than a FOCES as in the other scenarios.

## STS-TO-WSC

These xml files contains the responses from the STS, and are for all practical purposes identical, as the Assertion element is encrypted, and it is the content of the Assertion element that differs.

## WSC-TO-WSP

These xml files contains the requests to the web service provider, and have identical soap-bodies – the only difference lies in the soap-headers, where the signature scenario differs from the bootstrap and system user scenarios.

The signature scenario uses a bearer-token, hence the request contains a BinarySecurityToken element, containing the WSC’s x509 certificate, which is not present in the other scenarios.

## WSP-TO-WSC

These xml files contains the responses from the web service provider, and are identical across the scenarios.

# References

[OIOIDWS] OIO IDWS SOAP Profile 1.1 <https://digitaliser.dk/resource/3457606>

[LIBERTY] Liberty Basic SOAP Binding v 1.0

<https://digitaliser.dk/resource/414852>

[CXF] CXF and WS-SecurityPolicy.docx

[MAVEN] Apache Maven Build Tool v 3.x

<https://maven.apache.org/download.cgi>

[CRYPTO] Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files

**Java 7**

<http://www.oracle.com/technetwork/java/javase/downloads/jce-7-download-432124.html>

**Java 8** <http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html>

[OIO-BPP] OIOSAML Basic Privilege Profile 1.0

<https://digitaliser.dk/resource/2377872>

[WS-SEC-POL] WS-Security Policy 1.2

<http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html>