CXF

[WS-Security](#h.or2rxwxfzxpp)

[UsernameToken](#h.bj03rd27990x)

[How a CXF client is created](#h.6ugt3l17ly29)

[Create the factories](#h.jp6dt8i5swn)

[Call the setter on the factories](#h.ob7e4lcrtlq6)

[Create a Client](#h.9ai0lb57fyzc)

[org.apache.cxf.endpoint.Client](#h.mujlu2gwqbtg)

[IssuedTokenInterceptorProvider](#h.ftwobm88inib)

[PKI](#h.1p04nqpt3aof)

[Client](#h.an3jkhhgavq1)

[HTTPS](#h.r3fnnjsxyzi)

[WS-Security](#h.71f7ng9b76nx)

[TSM transport](#h.j2lgykyn2zuh)

[Tomcat](#h.p53owm36vl3d)

[Communication Module](#h.ll163i5mcaki)

[HTTPS](#h.gyo49tehe3dy)

[WS-Security](#h.8f0y8cltt919)

# WS-Security

## UsernameToken

Tutorial for using UsernameToken: <http://www.jroller.com/gmazza/entry/cxf_usernametoken_profile>

Properties:

**ws-security.callback-handler**

implements javax.security.auth.callback.CallbackHandler

-> org.apache.cxf.ws.security.wss4j.WSS4JInInterceptor

-> org.apache.ws.security.validate.UsernameTokenValidator

-> calls the handler

**ws-security.ut.validator**

implements org.apache.ws.security.validate.Validator

-> org.apache.cxf.ws.security.wss4j.WSS4JInInterceptor

-> calls the validator

# How a CXF client is created

<http://cxf.apache.org/docs/jax-ws-configuration.html>

<bean id="proxyFactory"   
 class="org.apache.cxf.jaxws.JaxWsProxyFactoryBean">  
 <property name="serviceClass" value="demo.spring.HelloWorld"/>  
 <property name="address" value="http://localhost:9002/HelloWorld"/>  
 </bean>  
  
 <bean id="client" class="demo.spring.HelloWorld"  
 factory-bean="proxyFactory" factory-method="create"/>

This is equivalent to

<jaxws:client id="helloClient"  
 serviceClass="demo.spring.HelloWorld"  
 address="http://localhost:9002/HelloWorld" />

## Create the factories

JaxWsProxyFactoryBean

ClientProxyFactoryBean

AbstractBasicInterceptorProvider : InterceptorProvider

JaxWsClientFactoryBean

ClientFactoryBean

AbstractWSDLBasedEndpointFactory

AbstractEndpointFactory

AbstractBasicInterceptorProvider : InterceptorProvider

JaxWsServiceFactoryBean

ReflectionServiceFactoryBean

AbstractServiceFactoryBean

---------

**Create JaxWsProxyFactoryBean**

public JaxWsProxyFactoryBean() {

super(new JaxWsClientFactoryBean());

}

**Create JaxWsClientFactoryBean**

public JaxWsClientFactoryBean() {

super(new JaxWsServiceFactoryBean());

}

**Create JaxWsServiceFactoryBean**

public JaxWsServiceFactoryBean() {

getIgnoredClasses().add(Service.class.getName());

//the JAXWS-RI doesn't qualify the schemas for the wrapper types

//and thus won't work if we do.

setQualifyWrapperSchema(false);

initSchemaLocations();

}

… see the super classes too (ReflectionServiceFactoryBean, AbstractServiceFactoryBean)

**\* super: Create ClientFactoryBean**

public ClientFactoryBean(ReflectionServiceFactoryBean factory) {

super(factory);

}

**\* super: Create AbstractWSDLBasedEndpointFactory**

protected AbstractWSDLBasedEndpointFactory(ReflectionServiceFactoryBean sbean) {

serviceFactory = sbean;

serviceClass = sbean.getServiceClass();

serviceName = sbean.getServiceQName(false);

endpointName = sbean.getEndpointName(false);

sbean.getServiceConfigurations().add(new SoapBindingServiceConfiguration());

}

.. super classes AbstractEndpointFactory and AbstractBasicInterceptorProvider dont do anything

**\* super: Create ClientProxyFactoryBean**

public ClientProxyFactoryBean(ClientFactoryBean fact) {

super();

this.clientFactoryBean = fact;

}

**\* super: Create AbstractBasicInterceptorProvider**

nothing meaningful

## Call the setter on the factories

**ClientProxyFactoryBean.setServiceClass(org.ib.sso.service.service1.Service1Endpoint) - Class<?>**

*public void setServiceClass(Class<?> serviceClass) {*

*clientFactoryBean.setServiceClass(serviceClass);*

*}*

**JaxWsClientFactoryBean.setServiceClass**

*public void setServiceClass(Class<?> serviceClass) {*

*super.setServiceClass(serviceClass);*

*if (((JaxWsServiceFactoryBean)getServiceFactory()).getJaxWsImplementorInfo() == null) {*

*JaxWsImplementorInfo implInfo = new JaxWsImplementorInfo(serviceClass);*

*((JaxWsServiceFactoryBean)getServiceFactory()).setJaxWsImplementorInfo(implInfo);*

*}*

*}*

**\* super: AbstractWSDLBasedEndpointFactory.setServiceClass**

*public void setServiceClass(Class<?> serviceClass) {*

*this.serviceClass = serviceClass;*

*}*

**ClientProxyFactoryBean.setServiceName(svc:Service1Service) - QName**

*public void setServiceName(QName serviceName) {*

*getServiceFactory().setServiceName(serviceName);*

*}*

**ReflectionServiceFactoryBean.setServiceName**

*public void setServiceName(QName serviceName) {*

*this.serviceName = serviceName;*

*}*

**ClientProxyFactoryBean.setEndpointName(svc:Service1ExtTransportPort) - QName**

*public void setEndpointName(QName endpointName) {*

*clientFactoryBean.setEndpointName(endpointName);*

*}*

**AbstractEndpointFactory.setEndpointName()**

*public void setEndpointName(QName endpointName) {*

*this.endpointName = endpointName;*

*}*

**ClientProxyFactoryBean.setAddress(“http://...”) - String**

*public void setAddress(String add) {*

*clientFactoryBean.setAddress(add);*

*}*

**AbstractEndpointFactory.setAddress()**

*public void setAddress(String address) {*

*this.address = address;*

*}*

**ClientProxyFactoryBean.setWsdlLocation(“path to WSDL”) - String**

*public void setWsdlLocation(String wsdlURL) {*

*setWsdlURL(wsdlURL);*

*}*

*public void setWsdlURL(String wsdlURL) {*

*clientFactoryBean.getServiceFactory().setWsdlURL(wsdlURL);*

*}*

**ReflectionServiceFactoryBean.setWsdlURL**

*public void setWsdlURL(String wsdlURL) {*

*// create a unique string so if its an interned string (like*

*// from an annotation), caches will clear*

*this.wsdlURL = new String(wsdlURL);*

*}*

## Create a Client

**JaxWsProxyFactoryBean.create()**

**ClientProxyFactoryBean.create()**

create the org.apache.cxf.endpoint.Client object:

Client c = clientFactoryBean.create();

create the ClientProxy

ClientProxy handler = new ClientProxy(c);

This is a java.lang.reflect.InvocationHandler

create the client proxy:

Object proxyObj = Proxy.newProxyInstance(clientFactoryBean.getServiceClass().getClassLoader(),

classes,

handler);

get the Client of a proxyObj:

Client c = ClientProxy.getClient(proxyObj);

## org.apache.cxf.endpoint.Client

Service1Ext: it receives the message sent by the client, calls service1 (and STS before)

**client.getEndpoint**:

endpoint.entrySet()

*key=ws-security.encryption.properties; value=comm.properties*

*key=ws-security.sts.client; value=org.apache.cxf.ws.security.trust.STSClient@7e42f8*

*key=ws-security.signature.username; value=commkey*

*key=ws-security.callback-handler; value=org.ib.sso.comm.lib.UsernamePasswordCallbackHandler@15ec4c1*

*key=ws-security.cache.issued.token.in.endpoint; value=false*

*key=ws-security.encryption.username; value=service1*

*key=ws-security.signature.properties; value=comm.properties*

endpoint.getOutInterceptors()

out interceptor: org.apache.cxf.interceptor.MessageSenderInterceptor@1d03cf

out interceptor: org.apache.cxf.jaxws.interceptors.SwAOutInterceptor@e524bd

out interceptor: org.apache.cxf.jaxws.interceptors.WrapperClassOutInterceptor@1cefbee

out interceptor: org.apache.cxf.jaxws.interceptors.HolderOutInterceptor@1fdfa79

endpoint.getEndpointInfo().getProperties().entrySet()

ep info key=ws-security.crypto.cache; value={comm.properties=org.apache.ws.security.components.crypto.Merlin@1475b7a}

ep info key=ws-security.encryption.crypto; value=org.apache.ws.security.components.crypto.Merlin@1da498

ep info key=policy-engine-info-client-endpoint; value=org.apache.cxf.ws.policy.EndpointPolicyImpl@1c6c918

ep info key=URI; value=<http://localhost:10080/sso-demo-business-service1/Service1?wsdl>

ep info key=org.apache.cxf.ws.security.tokenstore.TokenStore; value=org.apache.cxf.ws.security.tokenstore.EHCacheTokenStore@1bc4791

## IssuedTokenInterceptorProvider

IssuedTokenInterceptorProvider.IssuedTokenOutInterceptor.handleMessage(Message message)

if the “ws-security.cache.issued.token.in.endpoint” property is set “true” in jaxws:properties, then the received token is cached in the Endpoint:

if (cacheIssuedToken) {

message.getExchange().get(Endpoint.class).put(SecurityConstants.TOKEN, tok);

message.getExchange().put(SecurityConstants.TOKEN, tok);

message.getExchange().put(SecurityConstants.TOKEN\_ID, tok.getId());

message.getExchange().get(Endpoint.class).put(SecurityConstants.TOKEN\_ID, tok.getId());

}

This way the token is available from the Client.getEndpoint

# PKI

## Client

1) create private key / cert

keytool -genkey -keyalg RSA -dname "CN=**www.client.com**, OU=Client, O=Client, ST=Budapest, C=HU" -validity 1000 -alias clientkey -keypass clientkeypass -keystore **client.jks** -storepass clientstorepass

2) create a CSR

keytool -certreq -alias clientkey -file client\_csr.pem -keypass clientkeypass -keystore client.jks -storepass clientstorepass

3) issue cert with CA1

ca\_create\_server\_cert.bat client\_csr.pem client

4) concatenate the issued cert and the CA cert

copy client.cer + ca.cer client\_chain.cer

5) update keystore

keytool -import -file client\_chain.cer -keypass clientkeypass -keystore client.jks -storepass clientstorepass -alias clientkey

keytool -import -file ca.cer -keypass ca1keypass -keystore client.jks -storepass clientstorepass -alias ca1

keytool -import -file ca.cer -keypass ca2keypass -keystore client.jks -storepass clientstorepass -alias ca2

keytool -import -file comm.cer -keypass commkeypass -keystore client.jks -storepass clientstorepass -alias comm

cert CN: **www.client.com**

issuer: CA2

keystore:

**client.jks**; pass: clientstorepass

- PK clientkey; pass: clientkeypass

- X509 ca1, ca2, comm

### HTTPS

<http:conduit name="{http://sso.ib.org/service/service1/}Service1ExtTransportPort.http-conduit">

<http:tlsClientParameters disableCNCheck="true">

<sec:keyManagers keyPassword="clientkeypass"> <sec:keyStore type="jks" password="clientstorepass" resource="**client.jks**"/>

</sec:keyManagers>

<sec:trustManagers>

<sec:keyStore type="jks" password="clientstorepass" resource="**client.jks**"/>

</sec:trustManagers>

</http:tlsClientParameters>

</http:conduit>

### WS-Security

client.properties:

org.apache.ws.security.crypto.merlin.keystore.password=clientstorepass

org.apache.ws.security.crypto.merlin.keystore.alias=clientkey

org.apache.ws.security.crypto.merlin.keystore.file=**client.jks**

UsernamePasswordCallback:

<util:map id="passwords">

<entry key="clientkey" value="clientkeypass" />

</util:map>

jaxws:client properties:

<jaxws:properties>

<entry key="ws-security.callback-handler" value-ref="usernamePasswordCallbackHandler"/>

<entry key="ws-security.encryption.properties" value="client.properties"/>

<entry key="ws-security.signature.properties" value="client.properties"/>

<entry key="ws-security.encryption.username" value="comm"/> </jaxws:properties>

## TSM transport

1) create private key / cert

keytool -genkey -keyalg RSA -dname "CN=**www.tsm-transport.com**, OU=TSM-TRANSPORT, O=TSM-TRANSPORT, ST=Budapest, C=HU" -validity 1000 -alias tsmtransportkey -keypass tsmtransportpass -keystore **tsm-transport.jks** -storepass tsmtransportpass

2) create a CSR

keytool -certreq -alias tsmtransportkey -file tsm-transport\_csr.pem -keypass tsmtransportpass -keystore tsm-transport.jks -storepass tsmtransportpass

3) issue cert with CA1

ca\_create\_server\_cert.bat tsm-transport\_csr.pem tsm-transport

4) concatenate the issued cert and the CA cert

copy tsm-transport.cer + ca.cer tsm-transport\_chain.cer

5) update keystore

keytool -import -file tsm-transport\_chain.cer -keypass tsmtransportpass -keystore tsm-transport.jks -storepass tsmtransportpass -alias tsmtransportkey

keytool -import -file ca.cer -keystore tsm-transport.jks -storepass tsmtransportpass -alias ca1

keytool -import -file ca.cer -keystore tsm-transport.jks -storepass tsmtransportpass -alias ca2

keytool -import -file client.cer -keystore tsm-transport.jks -storepass tsmtransportpass -alias client

cert CN: **www.tsm-transport.com**

issuer: CA1

keystore:

**tsm-transport.jks**; pass: tsmtransportpass

- PK tsmtransportkey; pass: tsmtransportpass

- X509 ca1, ca2, client

### Tomcat

<Connector port="10443" protocol="HTTP/1.1" SSLEnabled="true"

maxThreads="150" scheme="https" secure="true"

keystoreFile="**tsm-transport.jks**" clientAuth="true"

keystorePass="tsmtransportpass" sslProtocol="TLS"

truststoreFile="**tsm-transport.jks**" truststorePass="tsmtransportpass" />

## Communication Module

1) create private key / cert

keytool -genkey -keyalg RSA -dname "**CN=www.comm.com**, OU=Comm, O=Comm, ST=Budapest, C=HU" -validity 1000 -alias commkey -keypass commpass -keystore **comm.jks** -storepass commpass

2) create a CSR

keytool -certreq -alias commkey -file comm\_csr.pem -keypass commpass -keystore comm.jks -storepass commpass

3) issue cert with CA1

ca\_create\_server\_cert.bat comm\_csr.pem comm

4) concatenate the issued cert and the CA cert

copy comm.cer + ca.cer comm\_chain.cer

5) update keystore

keytool -import -file comm\_chain.cer -keypass commpass -keystore comm.jks -storepass commpass -alias commkey

keytool -import -file ca.cer -keypass ca1pass -keystore comm.jks -storepass commpass -alias ca1

keytool -import -file ca.cer -keypass ca2pass -keystore comm.jks -storepass commpass -alias ca2

keytool -import -file service1.cer -keypass service1pass -keystore comm.jks -storepass commpass -alias service1

cert CN: **www.comm.com**

issuer: CA1

keystore:

**comm.jks**; pass: commpass

- PK commkey; pass: commpass

- X509 ca1, ca2, service1

### HTTPS

Configuration is only needed in Tomcat

### WS-Security

comm-properties:

org.apache.ws.security.crypto.merlin.keystore.password=commpass

org.apache.ws.security.crypto.merlin.keystore.alias=commkey

org.apache.ws.security.crypto.merlin.keystore.file=**comm.jks**

UsernamePasswordCallback:

<util:map id="passwords">

<entry key="commkey" value="commpass" />

<entry key="Libri" value="dummypass" />

<entry key="Libri2" value="dummypass" />

<entry key="BKV" value="dummypass" />

<entry key="www.client.com" value="dummypass" />

</util:map>

jaxws:properties:

<jaxws:properties>

<entry key="ws-security.signature.properties" value="comm.properties" />

<entry key="ws-security.signature.username" value="commkey"/>

<entry key="ws-security.callback-handler" value-ref="usernamePasswordCallbackHandler"/>

<entry key="ws-security.encryption.properties" value="comm.properties"/>

<entry key="ws-security.encryption.username" value="useReqSigCert"/>

</jaxws:properties>

## Secure Token Service

1) create private key / cert

keytool -genkey -dname "**CN=www.sts.com**, OU=STS, O=STS, ST=Budapest, C=HU" -validity 1000 -alias stskey -keypass stspass -keystore sts.jks -storepass stspass

2) create a CSR

keytool -certreq -alias stskey -file sts\_csr.pem -keypass stspass -keystore sts.jks -storepass stspass

3) issue cert with CA1

ca\_create\_server\_cert.bat sts\_csr.pem sts

4) concatenate the issued cert and the CA cert

copy sts.cer + ca.cer sts\_chain.cer

5) update keystore

keytool -import -file sts\_chain.cer -keypass stspass -keystore sts.jks -storepass stspass -alias stskey