

GSoC Project Summary

[blocked URL](#)

Summary

The aim of this page is to describe my work during the Google Summer of Code project on Fedora Repository. I have divided this page into x subcategories depending on into which part of "project's life cycle" it belongs. I have created the topic branch for this purpose in Git and all my changes are available from <https://github.com/Jiri-Kremser/fcrepo/tree/fcrepo-452>.

Here, I would like to thank to all the members of Fedora commiter team, especially to my GSoC mentor [A. Soroka](#) and [Chris Wilper](#). I hope that my work will be beneficial for the future release.

- 1 [Changes to the Build Process](#)
 - 1.1 [Dependencies](#)
 - 1.2 [WSDL to Java Code Generation](#)
- 2 [Changes to the Installation Process](#)
 - 2.1 [Fedora Home Directory](#)
- 3 [Changes to the Deployment Process of WS \(Including the Spring Configuration\)](#)
- 4 [Changes in the Code](#)
 - 4.1 [WS implementations](#)
 - 4.2 [Exceptions](#)
 - 4.3 [fcrepo-client-admin](#)
 - 4.4 [fcrepo-security-pep](#)
- 5 [Conclusion](#)

Changes to the Build Process

This section describes the changes within the build process scope such a dependencies and code generation from the WSDL files.

Dependencies

CXF can be attached by Maven to the fcrepo project in several ways:

1. two (three with embedded Jetty) Maven artifacts

```
<dependency>
<groupId>org.apache.cxf</groupId>
<artifactId>cxf-rt-frontend-jaxws</artifactId>
<version>${cxf.version}</version>
</dependency>

<dependency>
<groupId>org.apache.cxf</groupId>
<artifactId>cxf-rt-transports-http</artifactId>
<version>${cxf.version}</version>
</dependency>

<!-- Jetty is needed if you're using the CXFServlet -->
<dependency>
<groupId>org.apache.cxf</groupId>
<artifactId>cxf-rt-transports-http-jetty</artifactId>
<version>${cxf.version}</version>
</dependency>
```

2. one Maven artifact with bundled CXF

```

    <dependency>
    <groupId>org.apache.cxf</groupId>
    <artifactId>cxf-bundle</artifactId>
    <version>${cxf.version}</version>
    <exclusions>
    <exclusion>
    <groupId>commons-logging</groupId>
    <artifactId>commons-logging</artifactId>
    </exclusion>
    </exclusions>
    </dependency>

```

3. one Maven artifact with "minimal-bundled" CXF

```

    <dependency>
    <groupId>org.apache.cxf</groupId>
    <artifactId>cxf-bundle-minimal</artifactId>
    <exclusions>
    <exclusion>
    <groupId>commons-logging</groupId>
    <artifactId>commons-logging</artifactId>
    </exclusion>
    </exclusions>
    <version>${cxf.version}</version>
    </dependency>

```

All the artifact are available in the Maven Central. Dependencies should be transitively distributed throughout the fcrepo project, but sometimes I had to add them again.

I have used the second variant, because it seems to be the simplest one, but if you find the size of the jar file of the CXF to be high, you can go with 1 or 3.

Because the CXF uses internally the Spring framework, I had to attach the Spring jars to the installed fedora client (*\$FEDORA_HOME/client*) see [Fedora home directory](#)

WSDL to Java Code Generation

Since the Fedora follow the contract-first method, the SEIs (Service Endpoint Interfaces) have to be generated from the attached WSDLs as well as classes for types defined in the XML Schema attached to the WSDLs. WSDL file define a contract by means of XML language meanwhile SEI and type classes are annotated POJOs in Java doing the same thing. This process is done four times: API-A, API-A-MTOM, API-M, API-M-MTOM.

The old WS stack (Axis 1.3) used the *org.codehaus.mojo:axistools-maven-plugin* Maven plugin and my solution uses the *org.apache.cxf:cxf-codegen-plugin* as it was proposed on the homepage of the CXF, but the more general wsimport plugin for maven can be used as well (*com.sun.xml.ws:jaxws-rt* see <http://mojo.codehaus.org/jaxws-maven-plugin/usage.html>)

[Here](#) is the maven pom file with the "wsdl2java" configuration. I have removed all the references and tasks related to axis, but I am not sure whether the "cleaning" is still needed, so I have only commented the old plugin definition in the [pom](#).

When generating the SEIs, the default package is taken from the WSDLs targetNamespace. Hence, there has to be said, that another package should be taken. This is done in the [bindings](#) directory. For the generated types (xmls with the "JAX-B" prefix) and for the SEIs (xmls with the "JAX-WS" prefix). These xml binding files are configured in the [pom](#).

Changes to the Installation Process


The purpose of the Maven module fcrepo-installer is to install the repository and deploy it to the bundled tomcat. During this phase the *\$FEDORA_HOME* directory is created and some config and other files are copied there. In this phase the Spring application configs are also modified depending on the values in the installation config, which is passed to the installation jar file as a parameter.

Fedora Home Directory

Because the CXF needs Spring in the runtime, I had to copy the spring jars, and of course the other jars which CXF needs, into the *client/lib* directory. This is done in [fedora-home.xml](#) config. In the same file the new (*MTOMized*) WSDLs are marked for copying.

The [scripts](#) for running the Swing client are copied to Fedora home directory. Unfortunately, the CFX bundled jar cannot be simply copied into the *\$FEDORA_HOME/client/lib* whose content is all linked to the installation jar by the *java.endorsed.dirs* JVM param. It needs to be directly on classpath by using the *-cp* JVM option, therefore I have introduced the new variable in this [config](#) and its value is then written in the bash/bat script.

Warning to the Future Generations

 If the name of the CXF jar bundle is changed (it is imo not likely) or Fedora start to use dependency system as I proposed in [1.](#) or [3.](#), the value in this [config](#) has to be changed in order to allow the Swing client to run.

I have also added this [line](#) in order to reuse it by Spring in the applicationContext and determine ([here](#)) whether FESL should be enforced by the [interceptor](#) or not.

Changes to the Deployment Process of WS (Including the Spring Configuration)

Deployment/publishing of the web service in the CXF can be done:

1. Only in the code
Nicely described on the [CXF homepage](#)
2. In the Spring app context
3. Hybrid - in the Spring context there is an option to declare the WS and then set this WS to be deployed sometimes later in code, perhaps in the runtime on some condition.
4. Some Java EE 6 specific methods
not relevant for Fedora

I have used the second option. Using the "2." there are again two options. The first is to use proprietary cxf elements and the second one is to use jax-ws standard elements in the Spring xml config (app context file). Because we do not want any vendor lock-in in the future, the second alternative is used.

The deployment itself is done in [cxf.xml](#) file where the MTOM is enabled for the MTOMized endpoints and jax-ws handlers (they are called *handlers* in the JAX-WS terminology and *interceptors* in the CXF terminology) are plugged in.

Changes in the Code

The code is of course self descriptive 😊 but I should drop a few lines here as well.

WS implementations

Impl classes:

- [API-A](#)
- [API-M](#)
- [API-A-MTOM](#)
- [API-M-MTOM](#)

All the WS operations implemented in these classes are performed by `_Default{Access|Management}_` classes, therefore there is no complicated business logic here. It only delegates the work to another class and logs the exception if there is any.

In the previous WS impl classes there were different type of context object which has to be passed to `_Default{Access|Management}_` classes. Therefore, I had to make changes also to the static method `ReadOnlyContext.getSoapContext(ctx)`. Also, there are different types than in Axis 1.3 (`String[]` -> `ArrayOfString`, `ObjectMethodsDef[]` -> `List<ObjectMethodsDef>`, `T[]` -> `List<T>`, etc.). Hence, the `TypeUtility` class which provide static methods for converting the types had to be altered. The backward compatibility still holds, because on the SOAP msg level it is still the same. The different types are here because of the different binding strategy in the past. Now it is widely used JAX-B.

Exceptions

AsisFault -> SoapFault

fcrepo-client-admin

fcrepo-security-pep

Conclusion

For the old clients there are backward-compatible SOAP endpoints exposed by default on `localhost:8080/services/{access|management}` and for the clients build, after this changes comes into a public Fedora release, there are the SOAP endpoints (`localhost:8080/services/{access|management}MTOM`) with the MTOM support.