Using Maven with XML projects

Introduction

Introduction

- **We all organize our work in projects**
  - Third-party library use
  - Unit tests
  - Deliveries definition
- **In Java world, Maven is widely used since 2007**
- **Maven provides a common way to work**
  - A Project Model
  - Strong conventions, mainly on directory tree structure
  - Dependency management
  - A lifecycle

Introduction

- **We should share constraints between Java and XML projects**
  - We should not duplicate code
  - All our code should be unit tested
  - Deliveries should be build all time in the same manner
- **XML projects have their own constraints**
  - XML programs can not be run from command line
  - XML code has to be nested in Java wrappers (engines) to be run
  - So we deploy Java programs, even if XML technologies are mainly used
    - Exception, we may deliver XQuery or other XML code to database engines

Introduction

- **I'm a Java developer, surprised that XML has no build standard**
  - Each team does its own stuff
    - build.bat, build.sh
    - how_to_build.txt
  - There is no standardized build environment
    - Saxon version
    - Java version
    - Platform encoding
  - There is no simple way to re-use existing code without duplicating it
  - There is no standard definition of a delivery

- **We've plan to make Maven work correctly for XML technologies**
  - A common project to make all developers walk the same way

Using Maven

Using Maven

- **Requirements**
  - Avoiding code duplication
  - Running successfuly unit tests before building delivery
  - Being able to generate code
  - Producing full set of deliveries
    - Deployable artifact
    - Source code documentation
- **Use Oxygen as an IDE**
  - All stuff must run perfectly when developping XSL or XSpec under Oxygen
  - vi was not an option...

Avoid code duplication

- **Maven has a dependency management system**
  - If you need code from other project, just declare a dependency to that project
- **`Artifact` is the smallest referenceable part**
  - Identified by **`groupId:artifactId:version`**
  - Deployed in repositories
  - Actually a jar file
- **Just declare dependency**

```
<dependency>
  <groupId>net.sf.saxon</groupId>
  <artifactId>Saxon-HE</artifactId>
  <version>9.8.0-8</version>
</dependency>
```

```
<dependency>
  <groupId>eu.els.lib</groupId>
  <artifactId>myXslLib</artifactId>
  <version>1.0</version>
</dependency>
```

- **Maven knows how to get dependencies and make them available**
  - It adds jar file to project classpath

Avoid code duplication

- **Most XML code references resources via URI**
  - XSL, XQuery
  - DTD, Relax NG, XML Schema
  - XSpec, XProc, …
- **How to reference a dependency resource via URI ?**
  - Use **`artifactId:/`** as URI protocol

```
<xsl:import href="myXslLib:/dateFormat.xsl"/>
```

- **Use a `catalogBuilder-maven-plugin`**
  - To map **`artifactId:/`** to jar file location
  - Based on dependency declarations
  - This generates a catalog, platform dependant

Avoid code duplication

- **Catalog is a `rewriteURI` list**
  - Maven has downloaded dependency artifact jar file to local repository
  - Each dependency artifact is map to dependency jar file

```
<rewriteURI
 uriStartString="xf-lib:/"
 rewritePrefix="jar:file:~/.m2/repo/eu/els/lib/myXslLib/1.3.2/myXslLib-1.3.2.jar!/"
/>
```

- **Catalog content is platform dependant**
  - Each developer has its own
  - It is generated at each build
- **Catalog is always generated at the same place**
  - Convention
  - Oxygen uses this location : `${pdu}/catalog.xml`
  - Resources can be resolved in project context

Avoid code duplication

- **We have a way to re-use code from external libraries**
  - Declare a dependency
    - Maven resolves dependency
  - Use URI based on the **`artifactId:/`** protocol
    - XMLResolver resolves these URIs, based on generated catalog

```
<dependencies>
  <dependency>
    <groupId>eu.els.lib</groupId>
    <artifactId>myXslLib</artifactId>
    <version>1.0</version>
  </dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>top.marchand.xml.maven</groupId>
      <artifactId>catalogBuilder-maven-plugin</artifactId>
    </plugin>
  <plugins>
</build>
```

Unit testing

- **XSpec is a unit testing framework for XSLT, XQuery & Schematron**
- **Let's use the `xspec-maven-plugin` to run XML unit tests**

```
<build>
  <plugins>
    <plugin>
      <groupId>io.xspec.maven</groupId>
      <artifactId>xspec-maven-plugin</artifactId>
      <configuration>
        <catalogFile>catalog.xml</catalogFile>
      </configuration>
      <executions>
        <execution>
          <phase>test</phase>
          <goals>
            <goal>run-xspec</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

- **If one XSpec fails, plugin execution fails, build fails**

Unit testing

- `xspec-maven-plugin` **actually only supports XSLT**
- **Testing XQuery and Schematron will be quickly available**

- **A report is generated for each XSpec file**
- **A index is generated and shows a resume of each test file**

- **A Junit report will be quickly available**
  - This simplifies integration in Jenkins

Deliveries

- **Maven produces an artifact**
  - It contains everything produced by the build
  - No dependency included
- **Artifacts are deployed on enterprise repository**
  - Available for other projects
- **Code documentation**
  - **`xslDoc-maven-plugin`** for XSLT code
  - **`xquerydoc-maven-plugin`** for Xqurey
- **To deploy a program on a server, we produce a fat jar**
  - It includes generated artifact, and all dependencies packaged with
  - We are able to start program from command line
  - **`java -jar our-program-with-dependencies-3.1.2.jar`** …
  - We do generate a special catalog, which maps **`artifactId:/`** to classpath

Demo !

Demo

- **Demo projects are on GitHub :**
  - Library : https://github.com/mricaud/xml-prague-2018-demo_myLib
  - Main project : https://github.com/mricaud/xml-prague-2018-demo_myXMLproject

Questions ?