# Distro Tools Maven Plugin

This project is still at the experimental stage though is being actively used within the KenyaEMR distribution

This Maven plugin provides support for OpenMRS based distributions. Such distributions typically bundle a lot of content (metadata, concepts, forms, reports) with their code and this plugin seeks to help developers more easily manage that content.

| | |
|---|---|
| Source code | https://github.com/I-TECH/openmrs-contrib-maven-plugin-distrotools |
| Developers | Rowan Seymour |

## Background

A common challenge for distributions (or any module that bundles metadata) is referencing metadata objects, of which a large distribution will have many. The OpenMRS API provides several options which have their own strengths and weaknesses:

- **Database id** (all metadata classes)
  - Pros: succinct...
  - Cons: not portable between different databases
- **Name** (some metadata classes can be fetched by name, e.g. `EncounterType`, `Program` etc)
  - Pros: human readable
  - Cons: the primary purpose of name is for display in the UI layer so it might be changed
- **UUID** (all metadata classes)
  - Pros: guaranteed to be globally unique and unchanging
  - Cons: not human readable
- **Reference term** (so far just `Concept` but soon also `Drug`)
  - Pros: can be human readable if descriptive text is used (debatable whether this is a good practice)
  - Cons: not guaranteed unique and small performance hit due to extra database table joins

There also may be several places in the distribution where references to database objects exist. For example one might reference metadata objects in:

- Configuration resources (e.g. application context, JSON files)
- Java code (e.g. a constants class)
- Liquibase changeset files
- HTML form content (whether in resource files or database)
- Localization files (e.g. messages.properties used in conjunction with UIFR metadata localization)
- XML dataset files for testing

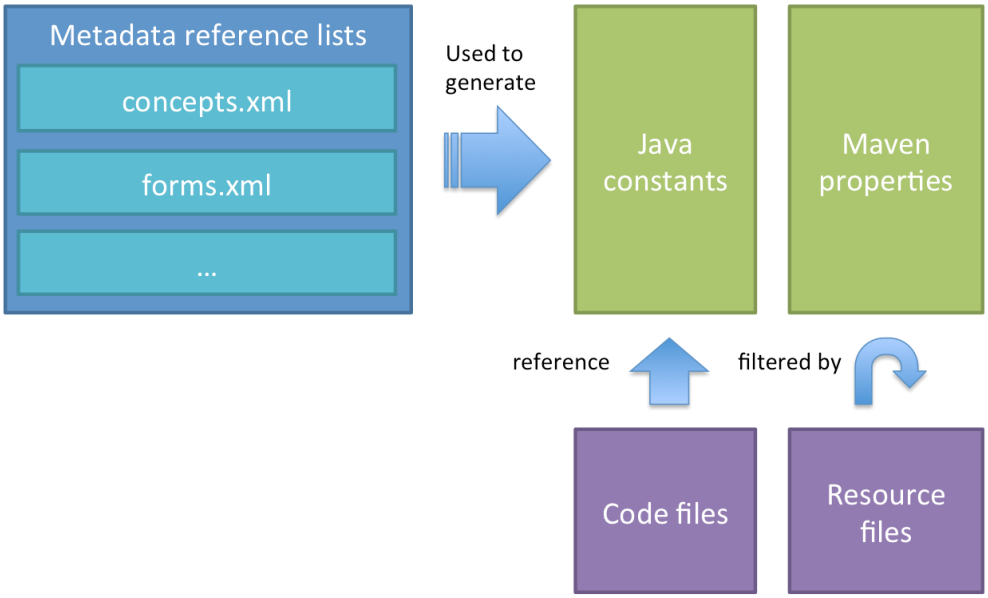
For a large distribution it can be difficult to keep all those references in consistent and there is no way of knowing at build time whether references are correct, e.g. you don't know if a reference term used in an HTML form is correct until you open that form on a real system.

## Solution

This project seeks to provide a new unified way of referencing metadata. The general principles are:

- There should be *one* central place for metadata references which is accessible throughout the distribution
- There should be more validation of metadata references at build time

The implementation borrows ideas from the Android SDK which generates some Java sources based on resource descriptions in XML files. It leverages Maven's resource filtering system to resolve metadata references in non-Java files.




The input metadata reference lists have a simple format and should be placed in *api/src/main/distro/metadata*. For example the contents of a simple concepts.xml might look like:

**Example concepts.xml**

```
<refs type="Concept">
  <ref key="YES" uuid="1065AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" />
  <ref key="NO" uuid="1066AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" />
</refs>
```

From that we generate a Java source file of constants, e.g.

**Example generated constants file**

```
public class Metadata {
  public static class Concept {
    public static final String YES = "1065AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";
    public static final String NO = "1066AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";
  }
  ...
}
```

Which can be used now in the distribution code to reference the concepts as `Metadata.Concept.YES` and `Metadata.Concept.NO`. We also generate a properties file to be used for resource filtering, e.g.

**Example generated properties file**

```
metadata.concept.YES=1065AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
metadata.concept.NO=1066AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
...
```

Which means distribution resource files can reference the concepts as `${metadata.concept.YES}` and `${metadata.concept.NO}`. At build time those will be replaced with the corresponding concept UUIDs. For example if you are using the UI Framework to localize metadata names then your *messa ges.properties* file can now include something like:

**Example messages file containing metadata references**

```
ui.i18n.Concept.name.${metadata.concept.YES}=Yes
ui.i18n.Concept.name.${metadata.concept.NO}=No
...
```

## Current Limitations

- Keeping the metadata reference lists in the module API makes it harder to re-use them in the module OMOD project.
- Won't work with webapp resources that are being dynamically reloaded in UI Framework's development mode (e.g. HTML form resources)
- No automated way to generate testing datasets (this will hopefully be implemented as part of this GSoC project).

## Plugin Usage

Firstly, add the plugin to your module's main pom.xml:

**Plugin dependency in main pom.xml**

```
<pluginManagement>
  <plugins>
    <plugin>
      <groupId>org.openmrs.maven.plugins</groupId>
      <artifactId>distrotools-maven-plugin</artifactId>
      <version>0.5</version>
    </plugin>
  </plugins>
</pluginManagement>
```

## Goal Configuration

### generate-metadata-sources

Generates metadata reference source files from input XML files (for now this is just concepts and forms). This includes two files:

- A Java source file of constants (`Metadata.java`)
- A properties file for filtering of resources containing metadata references (`metadata.properties`)

Configuration:

- *metadataDirectory* the directory containing the XML files
  - Required: no
  - Type: File
  - Default: *src/main/distro/metadata*
- *outputDirectory* the output directory for generated source files
  - Required: no
  - Type: File
  - Default: *target/generated-sources/distro*
- *outputPackage* the output package for generated source files
  - Required: yes
  - Type: String

**Example configuration of generate-metadata-sources in api/pom.xml**

```
<build>
  <plugins>
    <plugin>
      <groupId>org.openmrs.maven.plugins</groupId>
      <artifactId>distrotools-maven-plugin</artifactId>
      <executions>
        <execution>
          <phase>generate-sources</phase>
          <goals>
            <goal>generate-metadata-sources</goal>
          </goals>
          <configuration>
            <outputPackage>${project.parent.groupId}.${project.parent.artifactId}</outputPackage>
          </configuration>
        </execution>
      </executions>
    </plugin>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>build-helper-maven-plugin</artifactId>
      <version>1.8</version>
      <executions>
        <execution>
          <id>add-source</id>
          <phase>generate-sources</phase>
          <goals>
            <goal>add-source</goal>
          </goals>
          <configuration>
            <sources>
              <source>${project.build.directory}/generated-sources/distro</source>
            </sources>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
  ...
  <filters>
    <filter>${project.build.directory}/metadata.properties</filter>
  </filters>
  ...
</build>
```

## validate-forms

Performs basic validation (DOM validation, macro application) all HFE form files in a specified directory.

Configuration:

- *formsDirectory* the directory containing the form files
  - Required: yes
  - Type: File
- *formsExtension* the file extension used for form files
  - Required: no
  - Type: String
  - Default: html

**Example configuration of validate-forms in omod/pom.xml**

```
<plugin>
  <groupId>org.openmrs.maven.plugins</groupId>
  <artifactId>distrotools-maven-plugin</artifactId>
  <executions>
    <execution>
      <phase>validate</phase>
      <goals>
        <goal>validate-forms</goal>
      </goals>
      <configuration>
        <formsDirectory>src/main/webapp/resources/htmlforms</formsDirectory>
      </configuration>
    </execution>
  </executions>
</plugin>
```