Arousa Documentation

Contents

Arousa dependency system

Introduction

Arousa is an example of dependency management on data/content transformation projects. It is aimed as a tool for managing dependencies between Xslts, XQuery, Xproc, scripts and related resources. It is also intended as an example for getting started with Apache Ivy on such kind of projects, that rely frequently on reusable steps and components.

Apache Ivy is a general purpose dependency management tool. It is used frequently with Apache Ant. Ant is a mature build tool appropriate for projects that need detail and flexibility. Ivy is an abstract dependency management framework, open to a wide range of use cases.

Arousa has two main parts:

1. A management Script that helps your create and update the project templates and configuration files.

2. A base (or set of base) template project with a pre-built Apache Ivy configuration.

Why did I do it ?

I think that Ivy adapts really well to transformation projects, where detail is key and every case may have some specific needs. Many of this tools are frequently handled with Ant.

I already had some small experience with Ivy, and was struggling with non DRY practices on my xslt scripts. So, I hope that this can be useful as a jump in aid or as a hint for other people. Please use it and enjoy it.

Quick Start.

Installing the base script.

1. Dowload and unpack the Arousa zip distribution.

2. Set the AROUSA_HOME var on your system in a similar way to the JAVA_HOME,ANT_HOME, etc. Just set the AROUSA_HOME pointing to your installation.

```
export AROUSA_HOME=<Your_Installaton_path>

export PATH=$PATH:$AROUSA_HOME
```

3. Navigate to a work folder and type:

```
arousa
```

If you can see the arousa command line help output you should be ready to start your first project.

Type:

```
arousa template-project <your-new-project-name>
```

(you could also copy the template by hand and adjust name and configuration files).

At this point you should have created your first Arousa project with a build and configuration samples.

Running your first project:

In your new project you can run the usual clean, build ant commands. You can adjust the build commands to suit your specific needs.

We are now going to start managing dependencies. Type one of the following commands

```
arousa update-dependencies
ant arousa-update-dependencies
```

At this point you should notice that apache Ivy is downloading and a report on dependencies should also be showing. If everything looks ok, it's time to publish your first dependency.

Open the conf/arousa-ivy.xml and adjust name, path and version. Prepare the artifact for the build (adjust the name of the package and type

```
ant package
```

At this point a zip file with your name and version should be in the dist folder. Type one of the following:

```
ant dist
ant arousa-publish-dependency
arousa publish-dependency
```

If everything is ok, you should have published your first dependency and your dist zip file should be in a subfolder of your ivy repository.

Declaring dependencies

You will need to create a second dependency project and add your first project as a dependency.

Example:

Run one of the following:

```
ant arousa-update-dependencies
```

```
arousa update-dependencies
```

Now take a look at the lib and deps folders. On the lib folder you should have your zip files. On the dep folder you should have your xslts, xprocs, ..etc.

On the build folder your static resources (css, img, svg, etc.) should also be placed.

The clean Command

If you are updating dependencies to your local repository you will need to clean the local cache of the dependant projects.

When Ivy resolve the dependencies from one of your projects it stores a local cache with them and the dependencies data. The next calls will reuse that information unless you tell Ivy to update it.

This is **especially important when you are working with two projects on your local environment** .

Lets say that you are working on two interrelated projects, A and B on your computer. B project uses project A, you have previously built it, integrate the dependency configuration and test it. Now you have performed additional changes on A and published. You update dependencies on B, but dependencies does not seem to be updated.

You need to clean the cache first. You can either do:

```
arousa clean-cache
arousa update dependencies
```

or

```
ant arousa-clean-cache arousa-update dependencies
```

on project B.

The dependencies/lib folder convention.

Until now, what we have shown is 100% Ant/Ivy with nothing else added or strings attached. In order to differentiate dependency scripts from your current ones, and also in order to work on the next dependency chain step, we need to add some sort of folder naming convention.

We take source filesfrom

src/css

src/xslt

src/xproc

and so on.

We place (expanded) dependency files on

dep/xslt

dep/xproc

etc.

We avoid using the lib folder directly for the expanded files, since this is the default ivy folder and there is risk of collisions in file names. The zip files will be downloaded to the lib folder. It could be difficult to manage expanded and downloaded packages in the same location.

In order to refer to the dependendant script files, you should declare dependencies like the following example:

```
....
<p:xslt name="tasks-average">
  <p:input port="stylesheet">
    <p:document href="../../deps/xsl/grouped-tasks-average.xsl"/>
  </p:input>
  <p:input port="parameters"><p:empty/></p:input>
</p:xslt>
...
```

The above example shows a call to a dependency xslt script from and xproc file

```
...
<target name="build" >
```

```
   <getodcontent odsfile="data/plan/Planificacion.ods" destfolder="temp" />
   <calabdoc src="./deps/xpl/extract-ods-tasks.xpl" doc="./temp/content.xml"
  />
</target>
...
```

The above example shows a call to a dependency xproc script from and ant build file

You can refer your dependencies from the same folder in the following way.

```
<xsl:stylesheet
    version="3.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    >

    <xsl:output method="xml" indent="yes" ></xsl:output>
    <xsl:import href="planning-elements.xsl" />
    ...
    <xsl:template match="@*">
      ....
    </xsl:template>
    ...
</xsl:stylesheet>
```

```
     <p:declare-step name="extract-tasks"
       xmlns:p="http://www.w3.org/ns/xproc"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       version="1.0"
       >
....
       <p:xslt name="empty-cells">
  <p:with-input
    port="stylesheet"
      href="../xsl/process-empty-cells.xsl"/>
       </p:xslt>
....
     </p:declare-step>
```

This is the only convention added by the tool/scrip/example. What you get is 100% compatible Ant/Ivy projects apart from that. You can run the projects with Ant on a computer without the Arousa Script without problems, ... hopefully... .

Tailoring Arousa.

Arousa is a generic example for data/content transformation and dependency management.

This kind of projects need a great level of detail and adjustment/flexibility.

You can adjust the solution by modifying the Ant task in the build and conf/arousa-build.xml files. You can do it in one specific project or add a modified template to the tool.

The configuration Scripts

The Arousa template project is based on the following set of scripts.

/build.xml

This is the generic/usual Ant build file with the usual clean, build, package tasks.

conf/arousa-build.xml

The base Arousa script with the targets that handle dependency publishing and retrieving.

conf/arousa-ivy.xml

The project publications and dependencies. Despite its name, this is a regular ivy.xml file.

conf/arousa-config.xml

This file configures the repositories location, chain, etc. It is also an Ivy regular configuration file.

Where can I get more information

1. Apache Ant Manual
2. Apache Ivy Documentation

Requirements

1. Ant installation and some basic knowledge. Since all the solution is based on Apache Ant, some knowledge is needed. A working Apache Ant installation is needed.
2. **Bash or Cygwin.**

   The Arousa script has been only tested on a bash environment based on Cygwin. There are lots of aspects that need testing on different systems. Also, the script may be ported to windows .bat file.

   Even if your system does not support the script you can copy and paste the template by hand and use them with pure Ant.

   The ant parts should work in different environments with small or no adjustment.

Configuration

The directory structure convention

In order to maintain a clean distribution we needed to adopt a folder convention. We also needed to adopt a naming schema that (kind of) transparently integrate the dependency references both from the current project and the next ones.

We use a two level folder configuration.

<your-project>/src/xsl

Tools and Aknoledgements

Apache Ant. The scripts are based on the mature/popular build tool.

https://ant.apache.org

Apache Ivy. Basically, this tool/script is an example of an Apache Ivy configuration.

https://ant.apache.org/ivy/

Saxon/Saxonica. More or less needed for everything, Xslts and all the transformation tools that deppend on it.

https://www.saxonica.com

Morgana XProc/Calabash. Althought we don't provide xproc implementation XProc projects will require one of them and some ant script adjustment.

https://www.xml-project.com/morganaxproc/

http://xmlcalabash.com