Litera

Keeping the documentation up-to-date

Vladimir Ritz Bossicard (vbossica@gmail.com)

Instead of waiting until the end of a project to take care of the documentation I thrive to make this effort a prime artifact of any project. The approach taken to make this happen is to generate as much as possible from the code base and bind everything together into a Docbook document.

Let's face it: documentation is always the bastard child of the development process. It is always difficult to keep the documentation in sync with the code and not only because the code is faster written and modified than documented. More often than not, the documentation is not kept with the code and doesn't follow the code life cycle.

I have decided to write all the documentation for my projects using *Docbook* [http://www.docbook.org/] for several very simple reasons (1) the files are checked in into Subversion and can be merged if a branch is ever created and (2) files can easily be generated and included into a document.

The *Litera* project features tools to generate those Docbook documents and fragments from various sources to facilitate the generation of up-to-date documents.

Dependencies Documentation

If *Apache Ivy* [http://ant.apache.org/ivy/] is used to manage the third party libraries included into an application it is already possible to generate a nicely formatted HTML report listing all these dependencies. however, it is sometimes necessary to include these reports into an official documentation (e. g. a release note or a developer guide) and since I have settle for Docbook as my documentation format, here's a simple Ant target to parse, merge and output into a Docbook fragment the list of all dependencies, extracted from a set of Ivy reports:

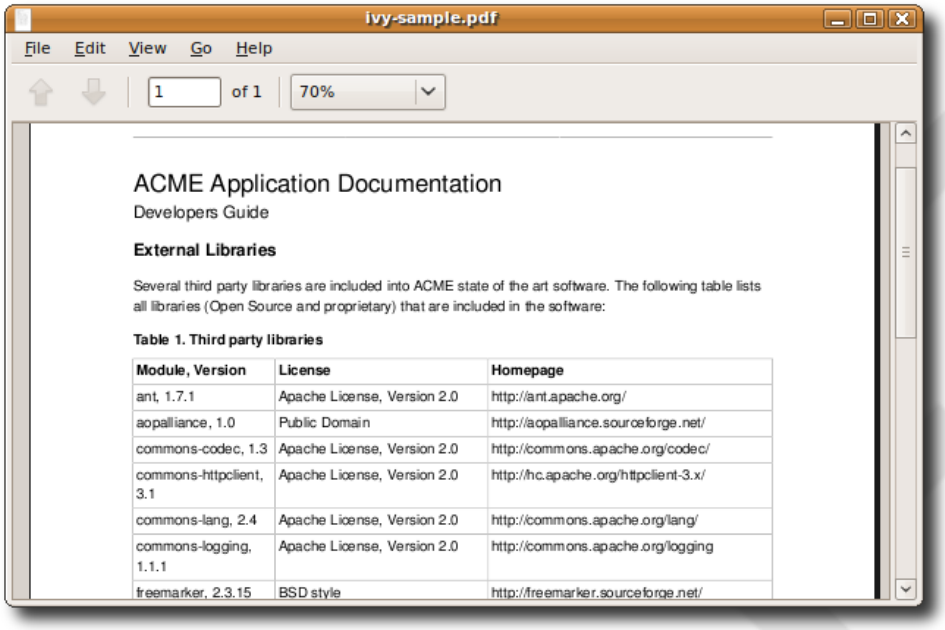
Figure 1. ivyreport2docbook Ant target

```
<taskdef resource="wkg-litera.properties"/>

<ivyreport2docbook outFile="docs/fragments/dependencies-report.docbook"
        title="Third party libraries">
    <fileset dir="target/ivy" includes="*-report.xml"/>
</ivyreport2docbook>
```

After that, it is trivial to automatically include this XML fragment into the final document:

Figure 2. Resulting documentation sample



Exporting Wiki Pages

It is often difficult to find the right place for a piece of documentation: a static document is often more tedious to maintain when a Wiki page is easy to edit but difficult to organize inside a master document. In order to get the best of both worlds, the easiest solution is to aggregate several Wiki documents into a static one, that can easily be included into a release.

As starting point, we write a very basic article on a local *MediaWiki* [http://www.mediawiki.org/] server:

Figure 3. Sample MediaWiki page

The tricky part is of course to write the parser that will transform the MediaWiki page into a valid *Docbook* [http://www.docbook.org/] document. Fortunately, there exist a couple of Open Source

libraries that handle this generation gracefully. Once everything has been glued together, we can export the page using a simple *Ant* [http://ant.apache.org/] target:

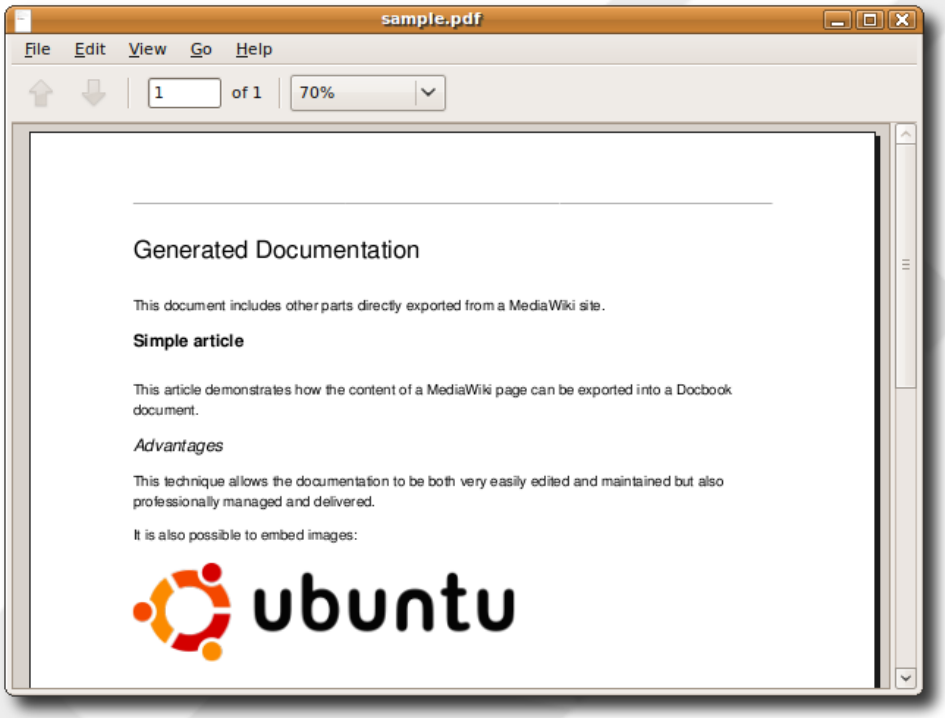
Figure 4. mediawiki2docbook Ant target

```
<taskdef resource="wkg-litera.properties"/>

<mediawiki2docbook url="http://sausalito/mediawiki/index.php/Simple_article"
    outFile="docs/sections/simple-article.docbook"/>
```

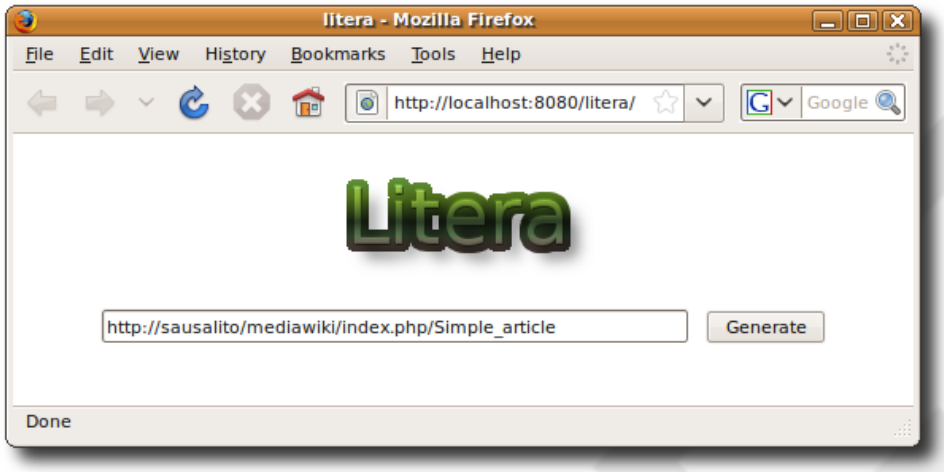
It was decided to export the document as `section` so that several of these exported pages can freely be included into on single document.

Figure 5. MediaWiki exported document



MediaWiki may not offer a PDF exporter out-of-the box but it is trivial to write a specific server to fill in this gap. It is furthermore possible to use one's own XSL templates to easily brand the generated PDF documents:

Figure 6. Litera MediaWiki server



Installation FAQ

This section describes the various steps necessary to install the *Litera* web application onto different environments.

Q: How to install the `litera.war` on Tomcat?

A: copy the `xerces-2.9.1.jar` file into the Tomcat library.

set the following system properties:

```
-Djavax.xml.parsers.SAXParserFactory=org.apache.xerces.jaxp.SAXParserFactoryImpl
-Dorg.xml.parsers.sax.parser=org.apache.xerces.parsers.SAXParser
```

Q: How to run the Eclipse WTP project?

A: Modify the Tomcat launcher's VM arguments to include the following:

```
-Djavax.xml.transform.TransformerFactory=com.icl.saxon.TransformerFactoryImpl
-Djavax.xml.parsers.DocumentBuilderFactory=org.apache.xerces.jaxp.DocumentBuilderFactoryImpl
-Djavax.xml.parsers.SAXParserFactory=org.apache.xerces.jaxp.SAXParserFactoryImpl
-Dorg.xml.parsers.sax.parser=org.apache.xerces.parsers.SAXParser
```