Wikbook

Wikbook Reference Guide

Julien Viet

eXo Platform

Alain Defrance

eXo Platform

Table of Contents

List of Examples

Preface

Wikbook is a set of tools for efficiently writing docbook documentation. For this purpose several choices have been made that are explained below

- The Docbook system can produce high quality documents with different outputs.
- Documentation written in wiki language is a good trade off between document richness and simplicity

Wikbook aims also to provide exclusive features that makes documentation easier to write. The perfect example we like to give is the inclusion of Java^{tm} source code at the heart of the documentation as it allows to have always up to date example that stay in sync with the documentation. The goal is to make the documentation easier to write and maintain.

Wikbook is also a lab to experiment new things and we are sure that it will get richer over time. We hope that people will find Wikbook interesting to write their documentation, experiment new ideas and contribute them.

Wikbook aims to be open and does not want to lock users into a proprietary system. The proof is that Wikbook is based on the open Docbook format.

This document is a work in progress and will get richer over time.

1

Introduction

1.1. Document syntax

Wiki syntaxes are not unique, Wikbook relies on an external framework used at the heart of XWiki that is able to make wiki syntaxes coexist. Any wiki syntax recognized by this framework can be used for documentation writing. The following syntaxes are recognized:

- XWiki 1.0 and 2.0
- Creole
- JSP Wiki
- Media Wiki
- Confluence
- TWiki

For the purpose of simplicity, the Wikbook documentation uses the XWiki 2.0 syntax.

Wikbook attempts to fully leverage the wiki syntax, however the horizontal rule wiki syntax is not supported. Indeed there is no such concept in the docbook system and horizontal rule are simply ignored.

1.2. Wiki macros

A wiki macro is way to complete the wiki syntax. Wiki syntax is usually not enough to express some ideas and a wiki macro provides a way to plug new behavior into a wiki syntax.

Wikbook makes use of wiki macro to integrate specific docbook features.

2

How-to

Wiki concepts are naturally mapped onto docbook concepts.

2.1. Document structure

A book is structured into chapters and each chapter is structured into sections. A wiki document provides a syntax for creating nested sections with different levels. The book structure can naturally leverage the wiki document structure with the following rules:

- Top level section correspond to book chapters
- Other wiki sections correspond to book sections

```
= Section 1 =
== Section 2 ==
```

```
<chapter>
  <title>Section 1</title>
  <section>
    <title>Section 2</title>
  </section>
</chapter>
```

All wiki sections don't have to be explicitely declared, it is possible to *omit* the declaration of a section. When it occurs, implicit docbook section will be created with no title.

2.2. Document content

The content of a book can be seen as a collection of content, each content can be categorized into blocks or rich text. Rich text can either be simple text or it can be made richer, like a word in bold. A block contains usually rich text and gives a meaning to the text, the most natural block is the paragraph.

2.2.1. Rich text

2.2.1.1. Emphasis

The following syntax can be used to put the emphasis for creating richtext

Table 2.1. Text emphasis

bold	**bold**
//italic//	*italic*
__underline__	underline
,,subscript,,	$_{\text{subscript}}$
^^upperscript^^	$^{\text{upperscript}}$
--strikethrough--	~~strikethrough~~
##monospaced##	`monospaced`
{{code}}inlineCode(){{/code}}	`inlineCode()`

2.2.1.2. Links

We can distinguish two kinds of links. A link can reference a target inside the document such as an anchor or a section or it can reference an URL.

The anchor macro plays an important role in internal links as it specifies the potential target of a link. Any internal link should reference a valid anchor inside the document. An anchor can be placed anywhere in text but it can also be present in a section title.

Table 2.2. Document links

```
= Chapter 1 {{anchor id=chapter_1 /}} =
A [[link>>#chapter_1]] to the chapter 1
The [[#chapter_1]] can be linked
A [[link>>#foo]] to an anchor {{anchor id=foo /}}
```
```
<chapter>
  <title>Chapter 1</title>
  <para>A<link linkend="chapter_1">link</link>to the chapter 1</para>
  <para>The<xref linkend="chapter_1"/>can be linked</para>
  <para>A<link linkend="foo">link</link>to an anchor</para>
</chapter>
```

Table 2.3. External links

The http://www.foobar.com site
The FooBar site
Send a mail to <`foo@bar.com`>

2.2.2. Verbatim text

Verbatim escapes the wiki syntax and is useful for citing wiki in a wiki document. It is useful for creating document such as this documentation.

Table 2.4. Verbatim text

`{{{[#bar]}}}`
`<para>[#bar]</para>`

It is also possible to escape a single character with a backslash. A backslash will generate a vertatim block around the escaped char. A double back slash will produce a single backslash.

Table 2.5. Char escape

`\[#bar\]`
`<para>[#bar]</para>`

Table 2.6. Backslash escape

`\\`
`<para>\</para>`

2.2.3. Blocks

2.2.3.1. Paragraphs

Unlike docbook, wiki paragraphs are implicitely defined, the general rule is that any text content that does not contain wiki instruction is one paragraph. The wikbook system creates docbook paragraphs when it is required. The simplest example is that any content in a section is a paragraph.

Table 2.7. Paragraph generation

```
= Chapter 1 =
The paragraph of the chapter 1.
== Section 1 ==
The paragraph of the section 1.
== Section 2 ==
The first paragraph of the section 2.

The second paragraph of the section 2.
```

```
<chapter>
  <title>Chapter 1</title>
  <para>The paragraph of the chapter 1.</para>
  <section>
    <title>Section 1</title>
    <para>The paragraph of the section 1.</para>
  </section>
  <section>
    <title>Section 2</title>
    <para>The first paragraph of the section 2.</para>
    <para>The second paragraph of the section 2.</para>
  </section>
</chapter>
```

2.2.3.2. Lists

It is easy to create lists in wiki syntax, whereas the docbook XML is very tedious. Several types of lists are possible such as bullet or ordered list.

Table 2.8. List examples

`* item 1`
`** item 1.1`
`** item 1.2`
`* item 2`
	• item 1
	    • item 1.1
	    • item 1.2
	• item 2

`1. item 1`
`11. item 1.1`
`11. item 1.2`
`1. item 2`
	1. item 1
	    1. item 1.1
	    2. item 1.2
	2. item 2

`(% style="upperroman" %)`
`1. item 1`
`11. item 1.1`
`11. item 1.2`
`1. item 2`
	1. item 1
	    1. item 1.1
	    2. item 1.2
	2. item 2

It is possible to configure also the list style according to the docbook capabilities.

- Bullet style
  - disc
  - circle
  - square
- Numbering style
  - arabic
  - loweralpha
  - loweroman
  - upperalpha
  - uperroman

- arabicindic

2.2.3.3. Tables

Tables are mapped to the docbook tables, here are a few examples

Table 2.9. Table examples

```
|1|2|3
|4|5|6
```

Table 2.10. A simple table

1	2	3
4	5	6

```
|=1|=2|=3
|4|5|6
```

Table 2.11. A table with a row header

1	2	3
4	5	6

```
|1|2|3
|=4|=5|=6
```

Table 2.12. A table with a row footer

1	2	3
4	**5**	**6**

```
(% title="The table" %)
|1|2|3
|4|5|6
```

Table 2.13. The table

1	2	3
4	5	6

By default a table expects inline content, that means that any content inside the table will not be interpreted as wiki text but as normal text. That behavior can be changed by using the group syntax block explained in the Section 2.2.3.4, “Groups ”.

This document makes an extensive usage of this feature.

Inside a complex content block, the usage of structural elements such as section is not allowed.

2.2.3.4. Groups

Groups are mostly useful for embedding complex structure inside a table. A grou is declared inside a block like (((...))).

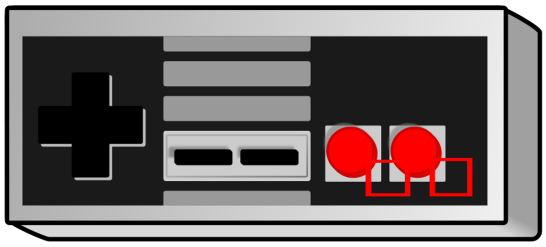
Table 2.14. A list inside a table with a group block

```
|(((
* item 1
* item 2
)))
```
	• item 1
	• item 2

2.2.3.5. Images

Simple images can be displayed by using the image syntax.

Table 2.15. Image example

`[[image:images/controller.png]]`


Images can naturally be inlined when some text is around the image.

Table 2.16. Image example

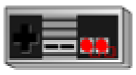
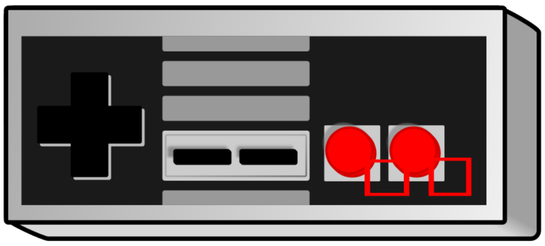
`An inline [[image:images/smallcontroller.png]] image`
An inline  image

Image display can be parameterized for all output but it is possible to target a specific output with a prefix. The *fo* prefix affects the PDF output and the *html* targets the HTML content. More details about docbook images parameterization can be found in the docbook imagedata reference.

Table 2.17. Image example

```
[[image:images/controller.png||title="The controller" align="center" html:sca
```

Figure 2.1. The controller



2.2.3.6. Admonitions

Table 2.18. Admonitions

Note	{{note}}Some noticeable text{{/note}}	some noticeable text
Warning	{{warning}}you should not do that{{/warning}}	you should not do that
Tip	{{tip}}a usefull tip{{/tip}}	a usefull tip
Caution	{{caution}}beware!!!{{/caution}}	beware!!!
Important	{{important}}something important{{/important}}	something important

2.2.3.7. Special blocks

A set of special blocks are available, they allow to give a special representation to the emboddied text.

The *code* macro creates a docbook programlisting XML element to display anything related to code. Special features are available that makes it very powerful

- XML code can be validated and pretty printed with a configuration indentation
- External Java code can be embedded inside the document

Table 2.19. A sample generic code

```
{{code}}x = x + 1;{{/code}}
```

```
x = x + 1;
```

Table 2.20. A sample Java code

```
{{code language=java}}x = x + 1;{{/code}}
```

```
x = x + 1;
```

Table 2.21. A sample Groovy code

```
{{code language=groovy}}x = x + 2;{{/code}}
```

```
x = x + 2;
```

Table 2.22. A sample XML code

`{{code language=xml}}<valid/>{{/code}}`
`<valid/>`

Table 2.23. A sample code from a file

`{{code language=java href="org/wikbook/A.java"/}}{{/code}}`
`// Content of the A.java file`

2.2.3.7.1. Java code

2.2.3.7.1.1. Declaring a code section

Code section can be declared with the generic `code` macro with the language attribute set to *java*. To make it shorter, the `java` macro can be used directly.

Table 2.24. Java code

`{{java}}x = x + 1;{{/java}}`
`x = x + 1;`

2.2.3.7.1.2. Java code citation

Wikbook has a special integration with Java project allowing to cite Java code inside the documentation from the existing source code. This feature requires the code source and binary to be available when Wikbook compiles the wikbook sources, the integration inside a Maven build is trivial and is covered in the Section 3.2.4, “Integration of code block citation ”.

The syntax is taken from the Javadoc syntax that allows to create reference to code elements via the {@link ref} where ref is a reference to a code element such as a class, a method or a field.

Table 2.25. A class citation

```
{{code language=java}}{@include org.wikbook.doc.AnObject}{{/code}}
```

```java
public class AnObject
{
   public void foo()
   {
      String a1 = "the first";
      String b1 = "block";
      String c1 = "of code";

      String a2 = "the second";
      String b2 = "block";
      String c2 = "of code";

      String a3 = "the third";
      String b3 = "block";
      String c3 = "of code";
   }

   public void foo(String s)
   {
      // An empty method
   }
}
```

Citing an whole class is often cumbersome and Wikbook gives the capability to cite class members, i.e methods or fields.

Table 2.26. A method citation

```
{{code language=java}}{@include org.wikbook.doc.AnObject#foo(java.lang.String
```

```java
   public void foo(String s)
   {
      // An empty method
   }
```

Going farther is even possible thanks to the code block citation feature. First let's define the notion of code block. A code block is a block of code inside a method that begins with a special comment like `// -1-` and terminates with a blank line or the end of the method, the number indicating a reference that can be used within an include instruction:

Example 2.1. A code block

```
// -1-
int a = 0;
```

Code blocks can be cited by adding a curly brace list of the blocks to cite inside an include instruction.

Table 2.27. A block code citation

```
{{code language=java}}{@include org.wikbook.doc.AnObject#foo() {1}}{{/code}}
```

```
      String a1 = "the first";
      String b1 = "block";
      String c1 = "of code";
      String a3 = "the third";
      String b3 = "block";
      String c3 = "of code";
```

Code block citations can be combined

Table 2.28. A block code citation

```
{{code language=java}}{@include org.wikbook.doc.AnObject#foo() {1,2}}{{/code}
```

```
      String a1 = "the first";
      String b1 = "block";
      String c1 = "of code";
      String a2 = "the second";
      String b2 = "block";
      String c2 = "of code";
      String a3 = "the third";
      String b3 = "block";
      String c3 = "of code";
```

Code block citation never cites the method declaration itself.

2.2.3.7.2. XML code

Code section can be declared with the generic `code` macro with the language attribute set to *xml*. To make it shorter the `xml` macro can be used directly.

2.2.3.7.2.1. XML code reformat

XML code is formatted when the *indent* macro attribute with a valid value.

Table 2.29. XML pretty printed

```
{{code language=xml indent=2}}
<bar><bar>bar</bar></bar><bar><bar>bar</bar></bar>
{{/code}}
```

```
<bar>
  <bar>bar</bar>
</bar>
<bar>
  <bar>bar</bar>
</bar>
```

The XML can have any number of sibling elements and does not require to wrap the content with a root element.

2.2.3.7.2.2. XML inclusion

Like Java code citation, Wikbook can also include XML documents. The same kind of integration is available with Maven explained in the Section 3.2.4, “Integration of code block citation ”. The syntax uses a special implicitely declared namespace, mapped to the `wikbook` prefix.

Table 2.30. XML inclusion

```
{{xml}}<wikbook:include href="document.xml"/>{{/xml}}
```

```
<foo>
  <bar/>
</foo>
```

The inclusion can be combined with an xpath attribute to select a fragment of the included document.

Table 2.31. XML inclusion

`{{xml}}<wikbook:include href="document.xml" xpath="//bar"/>{{/xml}}`
`<bar/>`

Originally the xinclude mechanism was chosen but the weak support of xpointer precluded its usage and instead we decided to introduce a custom wikbook inclusion mechanism

2.2.3.7.3. Screen output

The *screen* macro creates a docbook screen XML element to display anything related to the computer screen:

Table 2.32. A screen example

```
{{screen}}
julien:core julien$ ls -l
total 24
-rw-r--r--  1 julien  staff  1878 Apr 26 15:08 pom.xml
drwxr-xr-x  5 julien  staff   170 Apr 26 15:08 src
drwxr-xr-x  5 julien  staff   170 Apr 26 18:46 target
-rw-r--r--  1 julien  staff  4090 Apr 26 15:09 wikbook.core.iml
{{/screen}}
```

```
julien:core julien$ ls -l
total 24
-rw-r--r--  1 julien  staff  1878 Apr 26 15:08 pom.xml
drwxr-xr-x  5 julien  staff   170 Apr 26 15:08 src
drwxr-xr-x  5 julien  staff   170 Apr 26 18:46 target
-rw-r--r--  1 julien  staff  4090 Apr 26 15:09 wikbook.core.iml
```

2.2.3.7.4. Callouts

Callouts are useful for creating a set of references that refers to the code source. At the moment they are only available for Java^{tm} language. A callout is declared inside a code block using comments in special format.

Table 2.33. A simple callout

```
{{code language=java}}public void foo()
{
   System.out.println("This is going to the output"); // <1> A callout
}
{{/code}}
```

```
public void foo()
{
   System.out.println("This is going to the output"); ❶
}
```

❶	A callout

The syntax use angle brackets like **<1>** to define a callout because visually it is very similar to a callout bug. The value inside the brackets must be a number or it can be empty. The callout list will use this numbering to sort the callouts.

Most of the time it is not necessary to declare a number and the callout index can be empty. In that case, it uses a number that is greater than the previous index and lower than the next one. Eventually it is possible to use only empty callout, it is displayed in the definition order.

Table 2.34. Anonymous callouts

```
{{code language=java}}public void foo()
{
   System.out.println("This is going to the output"); // <> Callout 2
   System.out.println("This is going to the output"); // <> Callout 3
   System.out.println("This is going to the output"); // <2> Callout 4
   System.out.println("This is going to the output"); // <1> Callout 1
}
{{/code}}
```

```java
public void foo()
{
   System.out.println("This is going to the output"); ❶
   System.out.println("This is going to the output"); ❷
   System.out.println("This is going to the output"); ❹
   System.out.println("This is going to the output"); ❸
}
```

❶	Callout 1
❷	Callout 2
❸	Callout 3
❹	Callout 4

Callout anchor don't have to be mixed inside the text, a simple declaration without any text will just create an anchor. Somewhere else in the code block, the callout text can be declared by adding an equal sign between the right angle bracket and the text. As a consequence, several lines can be referenced with the same callout.

Table 2.35. A callout anchor and its text

```
{{code language=java}}public void foo()
{
   int a = 0; // <1>
   int b = 0; // <1>
}
// =1= An assignment
{{/code}}
```

```
public void foo()
{
   int a = 0; ❶
   int b = 0; ❷
}
```

❶ ❷	An assignment

2.2.4. Definition list

Wiki definition lists are managed as docbook variable lists.

Table 2.36. Definition lists

```
; term
: definition
```

```
<variablelist>
  <varlistentry>
    <term>term</term>
    <listitem>
      <para>definition</para>
    </listitem>
  </varlistentry>
</variablelist>
```

term

definition

```
(% title="the title" %)
; term
: definition
```

```
<variablelist>
  <title>the title</title>
  <varlistentry>
    <term>term</term>
    <listitem>
      <para>definition</para>
    </listitem>
  </varlistentry>
</variablelist>
```

the title

term
    definition

2.2.5. Example

Docbook defines a tag for creating examples. It gives the capability to give the example a title. Most importantly it generates a list of all examples in the book.

Table 2.37. An example

```
{{example}}an example{{/example}}
```

an example

```
{{example title="my example"}}an example with a title{{/example}}
```

an example with a title

The example macro can easily be combined, for instance a code example can be created by combining the example and code macros.

Table 2.38. A code example combining the example and code macros

`{{example title="increment x"}}{{code}}x = x + 1;{{/code}}{{/example}}`
Example 2.2. increment x `x = x + 1;`

2.2.6. Quotation

Quotation generates a set of DocBook blockquote structure.

Table 2.39. Quotation

`>John said this`
`>>Marie answered that`
`I said ok`

John said this Marie said that I said ok

2.2.7. Modularization

The *include* macro is a good way to provide modularization of a complex document.

Example 2.3. Document embedding

```
{{include document="embedded.wiki" /}}
```

Embedding does a special treatment to sections: when a document is embedded, its section are reconsidered with respect to the most inner section embedding the document.

3

Project integration

This chapter focuses on the integration of the documentation with the project it documents.

3.1. Java^{tm} code embedding

Wikbook allows to embbed Java^{tm} source code inside the documentation. It is very powerful when it comes to explain a tutorial or to explain an API.

todo

3.2. Maven integration

The Maven Wikbook plugin converts wiki document to a Docbook document. In order to produce final consumable documents (PDF, HTML), the Docbook document must be converted to new documents, this process is usually done via an XSL stylesheet that takes the Docbook document and transforms it.

3.2.1. The Maven Wikbook plugin

The Maven Wikbook plugin focus on transforming the Wiki content into a compliant DocBook XML document.

Example 3.1. Example of Wikbook Maven plugin

```
<plugin>
  <groupId>org.wikbook</groupId>
  <artifactId>wikbook.maven</artifactId>
  <executions>
    <execution>
      <phase>prepare-package</phase>
      <goals>
        <goal>transform</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <sourceDirectory>${pom.basedir}/src/main/wikbook/en/en-US</sourceDirectory
    <sourceFileName>book.wiki</sourceFileName>
    <destinationDirectory>${project.build.directory}/wikbook/src</destinationDi
    <destinationFileName>index.xml</destinationFileName>
    <emitDoctype>false</emitDoctype>
  </configuration>
</plugin>
```

3.2.2. The Maven transformation plugins

A transformation plugin plays an important role in the documentation generation because it takes the docbook generated by wikbook and transforms it into the real documentation. There are at least two plugins that provides this functionnality:

- The Docbkx Tools project: it contains a plugin that is actually used by the Wikbook archetype.
- The Maven jDocBook Plugin project.

Configuration of these plugins is not covered in this guide, however each of them contain a fairly good documentation, both are comparable in term of feature and can generate the main formats like HTML or PDF.

3.2.3. The Maven Wikbook archetype

The Wikbook archetype creates a Wikbook module that contains a basic documentation and its POM contains the plugin configuration to generate HTML and PDF formats. Its usage is very simple:

Example 3.2. Wikbook archetype usage

```
>mvn archetype:generate \
  -DarchetypeGroupId=org.wikbook \
  -DarchetypeArtifactId=wikbook.archetype \
  -DarchetypeVersion=0.9.37 \
  -DgroupId=<my.groupid> \
  -DartifactId=<my-artifactId>
```

After that, it is ready to be used and tweaked.

3.2.4. Integration of code block citation

We have covered how code can be cited in the Section 2.2.3.7.1.2, “Java code citation ” and the Section 2.2.3.7.2.2, “XML inclusion ”. Inside a Maven project, the Wikbook plugin requires access to the dependencies containing the code, source and binaries.

3.2.4.1. Producing the source dependencies

By default Maven does not create source code artifacts, however the *maven-source-plugin* does. Here is an example of the plugin configuration that can be added to a project to activate the generation of the source code artifact along with the compiled code artifact:

Example 3.3. Configuration of the maven-source-plugin

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-source-plugin</artifactId>
  <executions>
    <execution>
      <id>attach-sources</id>
      <goals>
        <goal>jar-no-fork</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

3.2.4.2. Declaring the right dependency

The source code artifact needs to be declared as a dependency of the Maven module processing the Wikbook document:

Example 3.4. Declaring the right dependency

```
<!-- Declares a dependency on the code -->
<dependency>
  <groupId>groupId</groupId>
  <artifactId>artifactId</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>
<!-- Declares a dependency on the source code -->
<dependency>
  <groupId>groupId</groupId>
  <artifactId>artifactId</artifactId>
  <version>1.0-SNAPSHOT</version>
  <classifier>sources</classifier>
</dependency>
```

4

Generation

4.1. Setup

4.1.1. Create your processing module

The documentation generation use annotation processing. Wikbook module only provide an abstract processing : AbstractTemplateProcessor In this module you will extend the abstract processor and write your templates.

You need to use wikbook/template/core module :

Example 4.1. Maven wikbook dependency

```
<dependency>
  <groupId>org.wikbook</groupId>
  <artifactId>wikbook.template.core</artifactId>
  <version>0.9.33</version>
</dependency>
```

4.1.2. Create your processor

Don't freak out, your processor will not be a full processor. All the processing will be done by AbstractTemplateProcessor, you only have to configure it.

The requited values data are :

- Processed annotations as Class[]
- Template name as String
- Output directory as String
- Generated extension as String (only used for file name)

Example 4.2. Typical jaxrs processor

```
@SupportedSourceVersion(SourceVersion.RELEASE_5)
@SupportedAnnotationTypes({"*"})
public class JaxrsTemplateProcessor extends AbstractTemplateProcessor {

  @Override
  protected Class[] annotations() {
    return new Class[] {
        Path.class,
        GET.class,
        POST.class,
        PUT.class,
        DELETE.class,
        PathParam.class,
        QueryParam.class,
        Consumes.class
    };
  }

  @Override
  protected String templateName() {
    return "jaxrs.tmpl";
  }

  @Override
  protected String generatedDirectory() {
    return "generated";
  }

  @Override
  protected String ext() {
    return "wiki";
  }

}
```

Example 4.3. Typical chromattic processor

```
@SupportedSourceVersion(SourceVersion.RELEASE_5)
@SupportedAnnotationTypes({"*"})
public class ChromatticTemplateProcessor extends AbstractTemplateProcessor {

  @Override
  protected Class[] annotations() {
    return new Class[] {
        PrimaryType.class,
        MixinType.class,
        Property.class,
        Properties.class,
        MappedBy.class,
        OneToOne.class,
        OneToMany.class,
        ManyToOne.class,
        Owner.class
    };
  }

  @Override
  protected String templateName() {
    return "chromattic.tmpl";
  }

  @Override
  protected String generatedDirectory() {
    return "generated";
  }

  @Override
  protected String ext() {
    return "wiki";
  }

}
```

4.1.3. Enable processing with service

To enable your processors as service just create the following file :

Example 4.4. META-INF/services/javax.annotation.processing.Processor

```
org.wikbook.doc.JaxrsTemplateProcessor
org.wikbook.doc.ChromatticTemplateProcessor
```

You can have only one processor.

4.1.4. Create your template

Each processor will use a template. A section is dedicated to these templates.

4.1.5. Package template with classifier

To be able to use the templates during the processing, you have to package your templates with a classifier (we will use "templates" classifier).

Example 4.5. Package templates with maven

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <executions>
        <execution>
          <phase>package</phase>
          <goals>
            <goal>jar</goal>
          </goals>
          <configuration>
            <classifier>templates</classifier>
            <includes>
              <include>**/templates/*</include>
            </includes>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

4.1.6. Generate !

Now you are ready to generate the documentation of existing module. Since you have declared your processors as service, just add it to classpath. You need to add some maven configuration to perform some stuff like unpack templates, use it, and package the generated files.

Example 4.6. Add your processors to the classpath

```
<dependency>
  <groupId>my.group.id</groupId>
  <artifactId>my.artifact.id</artifactId>
  <scope>compile</scope>
  <optional>true</optional>
</dependency>

<dependency>
  <groupId>my.group.id</groupId>
  <artifactId>my.artifact.id</artifactId>
  <classifier>templates</classifier>
</dependency>
```

Example 4.7. Unpack your templates and use it

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactId>
  <version>2.1</version>
  <executions>
    <execution>
      <id>unpack-templates</id>
      <phase>generate-sources</phase>
      <goals>
        <goal>unpack-dependencies</goal>
      </goals>
      <configuration>
        <includeGroupIds>org.exoplatform.social</includeGroupIds>
        <includeArtifactIds>exo.social.docs.gen</includeArtifactIds>
        <includeClassifiers>templates</includeClassifiers>
        <excludes>META-INF/**</excludes>
        <outputDirectory>${project.build.directory}/templates</outputDirectory
      </configuration>
    </execution>
  </executions>
</plugin>

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>2.3.2</version>
  <executions>
    <execution>
      <id>default-compile</id>
      <goals>
        <goal>compile</goal>
      </goals>
      <phase>compile</phase>
      <configuration>
        <compilerArguments>
          <sourcepath>${project.build.directory}/templates</sourcepath>
        </compilerArguments>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Example 4.8. Package generated files

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jar-plugin</artifactId>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>jar</goal>
      </goals>
      <configuration>
        <classifier>doc-generated</classifier>
        <includes>
          <include>**/generated/*</include>
        </includes>
      </configuration>
    </execution>
  </executions>
</plugin>
```

4.2. Templates

Each type which are annoted by one processed annotation will generate a file. When a file is generated the template is applied. In the template you can print and browse the annotation. Freemarker is used as template engine, that means you can use all freemarker feature like loop or conditional structure. Please have a look to the freemarker documentaton : http://freemarker.sourceforge.net/docs

In the template you can do simple actions like print type name, element name, but you can also do more complex operations like get children, get sibling, get the javadoc and more things.

4.2.1. Root elements

The root data are the annotation name like "@Path" for jaxrs using. Only the type annotations are considered as root data.

4.2.2. Annotation name

To print the annotation name you can use "name" property.

Example 4.9. Simple annotation name

Code	`@Annotation`
	`class A {}`

Template	`Name : ${@Annotation.name}`
Output	`Name : Annotation`

Actually this time we already know the name, but sometimes it can be useful (when you don't know which annotation you are working on).

4.2.3. Element name

It's the element's name. It can be the class name, method name, or parameter name which are annoted by the current annotation.

Example 4.10. Class element name

Code	`@Annotation`
	`class A {}`

Template	`Name : ${@Annotation.elementName}`
Output	`Name : A`

Example 4.11. Method element name

Code	`// ...`
	`  @Annotation`
	`  String m() { /* ... */ }`
	`// ...`

Template	`// ...`
	`Name : ${a.elementName}`
	`// ...`

Output	`// ...`
	`Name : m`
	`// ...`

Example 4.12. Parameter element name

Code	`// ...`
	`  String m(@Annotation String p) { /* ... */ }`
	`// ...`

Template	`// ...`
	`Name : ${a.elementName}`
	`// ...`

Output	`// ...`
	`Name : p`
	`// ...`

4.2.4. Type

4.2.4.1. Data

The type property have many values and provide a way to get more things about the type. It returns type, parameter type.

Example 4.13. Class type values

Code	`package p;`
	
	`@Annotation`
	`class A {}`

Template	`Name : ${@Annotation.type.name}`
	`Full name : ${@Annotation.type.fqn}`
	`Is array : ${@Annotation.type.isArray}`

Output	`Name : A`
	`Full name : p.A`
	`Is array : false`

Example 4.14. Method type values

Code	`// ...`
	`  @Annotation`
	`  String[] m() { /* ... */ }`
	`// ...`

Template	`// ...`
	`Name : ${a.type.name}`
	`Full name : ${a.type.fqn}`
	`Is array : ${a.type.isArray}`
	`// ...`

Output	`// ...`
	`Name : String[]`
	`Full name : java.lang.String[]`
	`Is array : true`
	`// ...`

Example 4.15. Parameter type values

Code	
	```
	// ...
	  Integer[] m(@Annotation String p) { /* ... */ }
	// ...
	```

Template	
	```
	// ...
	Name : ${a.type.name}
	Full name : ${a.type.fqn}
	Is array : ${a.type.isArray}
	// ...
	```

Output	
	```
	// ...
	Name : String
	Full name : java.lang.String
	Is array : false
	// ...
	```

4.2.4.2. Type parameter

Generics type parameter are also supported. You can reach the parameters with "parameter" value.

Example 4.16. Reach type parameter

Code	`// ...`
	`  @Annotation`
	`  Map<String, Integer[]> m(String p) { /* ... */ }`
	`// ...`

Template	`// ...`
	`Parameter 1 (key) :`
	`Name : ${a.type.parameter[0].name}`
	`Full name : ${a.type.parameter[0].fqn}`
	`Is Array : ${a.type.parameter[0].isArray}`
	
	`Parameter 2 (value) :`
	`Name : ${a.type.parameter[1].name}`
	`Full name : ${a.type.parameter[1].fqn}`
	`Is Array : ${a.type.parameter[1].isArray}`
	`// ...`

Output	`// ...`
	`Parameter 1 (key) :`
	`Name : String`
	`Full name : java.lang.String`
	`Is Array : false`
	
	`Parameter 2 (value) :`
	`Name : Integer`
	`Full name : java.lang.Integer`
	`Is Array : true`
	`// ...`

4.2.4.3. Get annotation

Type data have special member "annotation" which allow to get annotations data of return type.

Example 4.17. Reach annotation of type

Code

```
@Annotation("foo")
class A {
}
// ...
  @AnotherAnnotation
  A m() { return null; }
// ...
```

Template

```
// ...
<#assign type = anotherAnnotation.type>
<#assign a = type.annotation("@Annotation")>
<#assign v = a.attribute("value")>
Value : ${v}
// ...
```

Output

```
// ...
Value : foo
// ...
```

4.2.5. Javadoc

The javadoc is accessible with the "doc" property.

4.2.5.1. General Javadoc

Example 4.18. Class or method general documentation

Code

```
/**
 * This is
 * the documentation.
 */
@Annotation
class A {}
```

Template

```
Documentation : ${@Annotation.doc()}
```

Output

```
Documentation : This is the documentation.
```

You can use doc() for class or method documentation. You can also get the javadoc from a parameter. In this case you have only default javadoc which are corresponding do the parameter's javadoc

Example 4.19. Parameter documentation

Code	
	```
	// ...
	  /**
	   * General method documentation.
	   *
	   * @param p The parameter documentation
	   */
	  Integer m(@Annotation String p) { /* ... */ }
	// ...
	```

Template	
	```
	// ...
	Documentation : ${a.doc()}
	// ...
	```

Output	
	```
	// ...
	Documentation : The parameter documentation
	// ...
	```

4.2.5.2. Javadoc with one key

You can get a specific documentation part like "author" in this way :

Example 4.20. Get the author

Code	
	```
	/**
	 * This is the documentation.
	 *
	 * @author foo
	 */
	@Annotation
	class A {}
	```

Template	
	```
	Documentation : ${@Annotation.doc()}
	Author : ${@Annotation.doc("author")}
	```

Output	
	```
	Documentation : This is the documentation.
	Author : foo
	```

4.2.5.3. Javadoc with many keys

Sometimes the documentation can have many times the same key. You can either iterate on each key, or ask wikbook to display all items.

Example 4.21. Use 'flat' prefix to display all values separated by coma

Code	
	```
	/**
	 * This is the documentation.
	 *
	 * @author foo
	 * @author bar
	 */
	@Annotation
	class A {}
	```

Template	
	```
	Documentation : ${@Annotation.doc()}
	Authors : ${@Annotation.doc("flat:author")}
	```

Output	
	```
	Documentation : This is the documentation.
	Authors : foo, bar
	```

Example 4.22. Use 'list' prefix to make iteration possible

Code	
	```
	/**
	 * This is the documentation.
	 *
	 * @author foo
	 * @author bar
	 */
	@Annotation
	class A {}
	```

Template	
	```
	<#list @Annotation.doc("list:author") as author>
	Author : ${author}
	</#list>
	```

Output	
	```
	Author : foo
	Author : bar
	```

4.2.5.4. Get documentation as block

You could need to keep the documentation as it. For exemple you could wish to store some exemple without remove formatting.

Example 4.23. Use 'block' prefix to keep formatting

Code

```
/**
 * This is the documentation.
 *
 * @exemple
 * class A {
 *   // Here there is the code
 * }
 }
 */
@Annotation
class A {}
```

Template

```
Code :
${@Annotation.doc("bloc:exemple")}
```

Output

```
Code :
class A {
  // Here there is the code
}
```

4.2.6. Browse children

Each annotation allow to get children annotations. You just need to provide what annotation you are looking for and iterate. You can also set many param to look for many annotations in the same browsing.

Example 4.24. Browse children

Code

```
@Annotation
class A {

  @A
  void m() {}

  @B
  void m2() {}

  @A
  void m3() {}

}
```

Template

```
Annoted by @A :
<#list @Annotation.children("@A") as a>
Method : ${a.elementName}
</#list>

Annoted by @A or @B :
<#list @Annotation.children("@A", "@B") as ab>
Method : ${ab.elementName} (${ab.name})
</#list>
```

Output

```
Annoted by @A:
Method : m
Method : m3

Annoted by @A or @B :
Method : m (A)
Method : m2 (B)
Method : m3 (A)
```

4.2.7. Get sibling

Each annotation can reach a sibling annotations.

Example 4.25. Get sibiling

Code

```
// ...
  @A
  @B
  void m() {}
// ...
```

Template

```
// ...
Name : ${a.name}
Sibling @B : ${a.sibling("@B").name}
// ...
```

Output

```
// ...
Name : A
Sibling @B : B
// ...
```

After got sibling annotation you can reach the attribute then (see next section)

4.2.8. Get attributes

The attribute method allow to enter inside the annotation to get any contained value by the annotation. To use it you only have to call "attribute" method with member name. Don't forget that the default value name is "value".

Example 4.26. Get simple value

Code

```
@Annotation("foo", name = "bar")
class A {}
```

Template

```
Value : ${@Annotation.attribute("value")}
Name : ${@Annotation.attribute("name")}
```

Output

```
Name : foo
Value : bar
```

Some annotation can have more complex value like array, you can print or iterate on it.

Example 4.27. Get array value

Code

```
@Annotation({"foo", "bar"})
class A {}
```

Template

```
Values : ${@Annotation.attribute("flat:value")}

or

<#list @Annotation.attribute("list:value") as v>
Value : ${v}
</#list>
```

Output

```
Values : foo, bar

or

Value : foo
Value : bar
```

An annotation can also contains another nested annotation. Let use attribute in the same way to get it.

Example 4.28. Get annotation value

Code

```
@Annotation(@B("foo", name = "bar"))
class A {}
```

Template

```
Value : ${@Annotation.attribute("value").attribute("value")}
Name : ${@Annotation.attribute("value").attribute("name")}
```

Output

```
Value : foo
Name : bar
```

Example 4.29. Get array annotation value

Code

```
@Annotation({@B("foo"), @B("bar")})
class A {}
```

Template

```
<#list ${Annotation.attribute("value") as sub}
Value : ${sub.attribute("value")}
</#list>
```

Output

```
Value : foo
Value : bar
```

4.2.9. Browse package

Some framework use package annotation and wikbook templating allow to use these annotations. The only one difference is that package annotation doesn't have type member. You can use attribute, doc, name and children member to browse package content.

Example 4.30. Browse package annotation

Code	
	```
	// p/package-info.java

	@P("foo")
	package p;
	```
	```
	// p/A.java

	package p;

	@Annotation("a")
	class A {}
	```
	```
	// p/B.java

	package p;

	@Annotation("b")
	class B {}
	```

Template	
	```
	@P value : ${@P.attribute("value")}
	<#list ${@P.children("@Annotation") as c}
	Found : ${c.type.fqn} (${c.attribute("value")})
	</#list>
	```

Output	
	```
	@P value : foo
	Found : p.A (a)
	Found : p.B (b)
	```

4.2.10. Full sample

Now we will see a concrete use case with typical source class and full template. Two use cases will be considered : JAX-RS API and Chromattic API. Meanwhile you can create yours.

4.2.10.1. JAX-RS

Example 4.31. Source code

```
/**
 * User end point.
 * @author foobar
 * @since 1.0
 */
@Path("api/project/v1/{portalContainerName}/")
public class MyRestService {

  /**
   * This is the way to get an user by its id.
```

```
     *
     * @param uriInfo The URI Request info.
     * @param portalContainerName The associated portal container name
     * @param userId The specified user Id
     * @param format The expected returned format
     * @authentication
     * @request
     * {code}
     * GET: http://my.domain.lan/rest/private/api/project/v1/portal/user/1a2b3c4
     * {code}
     * @response
     * {code:json}
     * {
     *   "id": "1a2b3c4d5e6f7g8h9j",
     *   "firstName": "foo",
     *   "lastNameName": "bar",
     *   "createdAt": "Fri Jun 17 06:42:26 +0000 2011", //The Date follows ISO 8
     * }
     * {code}
     * @return The user
     *
     */
    @GET
    @Path("user/{userId}.{format}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response getById(@Context UriInfo uriInfo,
                                @PathParam("portalContainerName") String por
                                @PathParam("userId") String userId,
                                @PathParam("format") String format) {

      return null; // Do implementation

    }

    /**
     * This is the way to get all the users a new user. The service return the c
     *
     * @param uriInfo The URI Request info.
     * @param portalContainerName The associated portal container name
     * @param format The expected returned format
     * @param number The returned number
     * @authentication
     * @request
     * {code}
     * GET: http://my.domain.lan/rest/private/api/project/v1/portal/users.json
     * {code}
     * @response
     * {code:json}
     * [
     *   {
     *     "id": "1a2b3c4d5e6f7g8h9i",
     *     "firstName": "foo1",
     *     "lastNameName": "bar1",
     *     "createdAt": "Fri Jun 17 06:42:26 +0000 2011", //The Date follows ISO
     *   },
     *   {
     *     "id": "1a2b3c4d5e6f7g8h9k",
     *     "firstName": "foo2",
     *     "lastNameName": "bar2",
     *     "createdAt": "Fri Jun 18 06:42:26 +0000 2011", //The Date follows ISO
     *   }
     * ]
     * {code}
     * @return The users
     *
     */
    @GET
    @Path("users.{format}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response getById(@Context UriInfo uriInfo,
                                      @PathParam("portalContainerName") String por
                                      @PathParam("format") String format) {
                                      @QueryParam("number") String number) {
```

```
      return null; // Do implementation

    }
```

Example 4.32. Template

```
<#list @Path.children("@GET", "@POST", "@PUT", "@DELETE") as verb>
h3. ${verb.name} ${verb.sibling("@Path").attribute("flat:value")?replace("{", '

*Description*: ${verb.doc()}

*URL*:

{code}
http://platform35.demo.exoplatform.org/rest/private/${@Path.attribute("value")}
{code}
<#if verb.sibling("@Produces")??>

*Supported Format*: ${verb.sibling("@Produces").attribute("flat:value")}
</#if>
<#if verb.doc("authentication") != "">

*Requires Authentication*: true
</#if>

*Parameters*:
* *Required (path parameters)*: <#if verb.children("@PathParam")?size = 0>No</#
<#list verb.children("@PathParam") as param>
** *{noformat}${param.attribute("flat:value")}{noformat}*: ${param.doc()}
</#list>
* *Optional (query paramters)*: <#if verb.children("@QueryParam")?size = 0>No</
<#list verb.children("@QueryParam") as param>
** *{noformat}${param.attribute("flat:value")}{noformat}*: ${param.doc()}
</#list>

*Request*:

${verb.doc("bloc:request")}

*Response*:

${verb.doc("bloc:response")}

</#list>
```

Example 4.33. Output

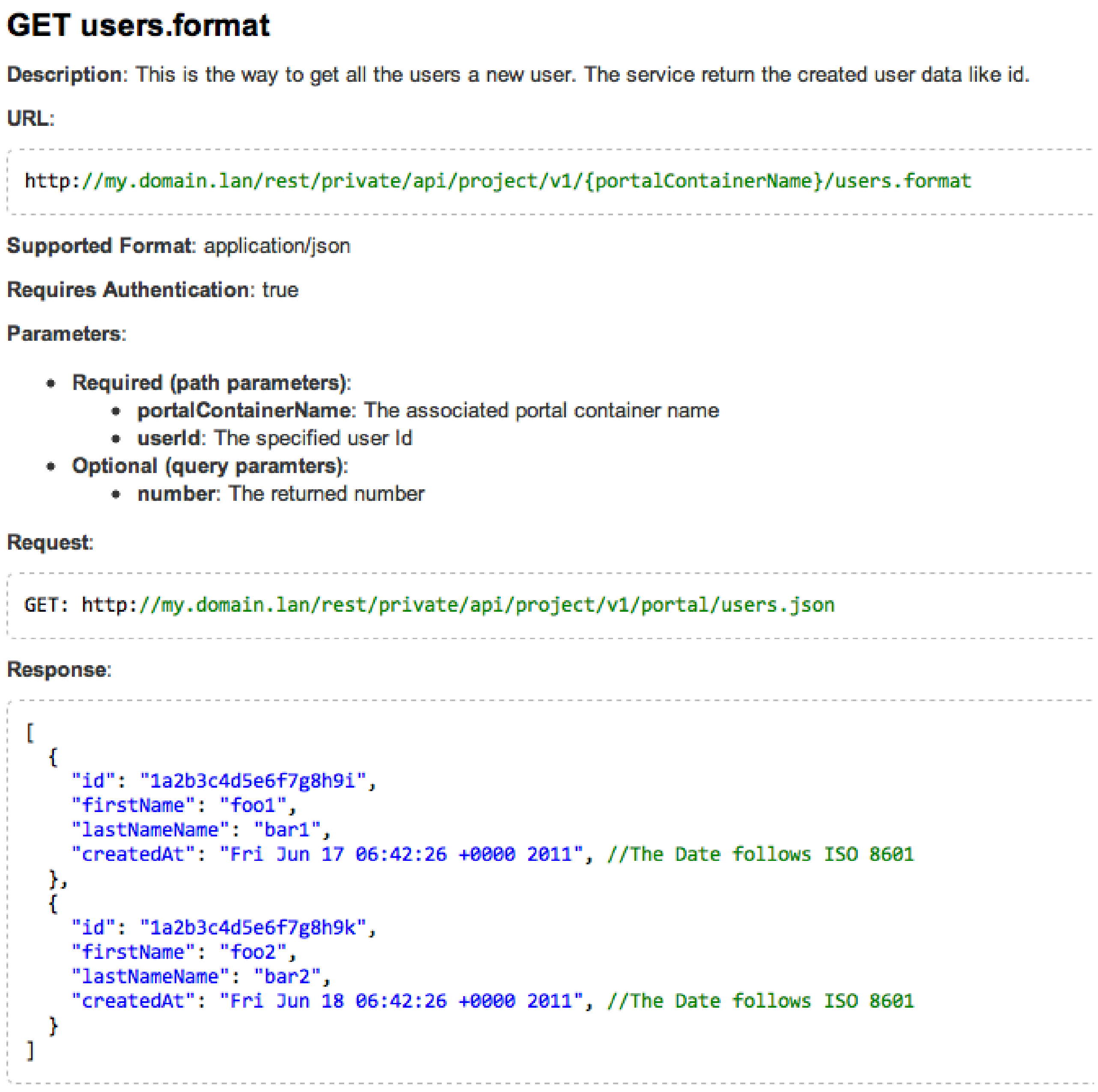
```
h3. GET user/userId.format

*Description*: This is the way to get an user by its id.

*URL*:

{code}
http://my.domain.lan/rest/private/api/project/v1/{portalContainerName}/user/use
```

```
{code}

*Supported Format*: application/json

*Requires Authentication*: true

*Parameters*:
* *Required (path parameters)*:
** *{noformat}portalContainerName{noformat}*: The associated portal container n
** *{noformat}userId{noformat}*: The specified user Id
** *{noformat}format{noformat}*: The expected returned format
* *Optional (query paramters)*: No

*Request*:

{code}
GET: http://my.domain.lan/rest/private/api/project/v1/portal/user/1a2b3c4d5e6f
{code}

*Response*:

{code:json}
{
 "id": "1a2b3c4d5e6f7g8h9j",
 "firstName": "foo",
 "lastNameName": "bar",
 "createdAt": "Fri Jun 17 06:42:26 +0000 2011", //The Date follows ISO 8601
}
{code}

h3. GET users.format

*Description*: This is the way to get all the users a new user. The service re

*URL*:

{code}
http://my.domain.lan/rest/private/api/project/v1/{portalContainerName}/users.f
{code}

*Supported Format*: application/json

*Requires Authentication*: true

*Parameters*:
* *Required (path parameters)*:
** *{noformat}portalContainerName{noformat}*: The associated portal container n
** *{noformat}userId{noformat}*: The specified user Id
* *Optional (query paramters)*:
** *{noformat}number{noformat}*: The returned number

*Request*:

{code}
GET: http://my.domain.lan/rest/private/api/project/v1/portal/users.json
{code}

*Response*:

{code:json}
```

```
[
 {
   "id": "1a2b3c4d5e6f7g8h9i",
   "firstName": "foo1",
   "lastNameName": "bar1",
   "createdAt": "Fri Jun 17 06:42:26 +0000 2011", //The Date follows ISO 8601
 },
 {
   "id": "1a2b3c4d5e6f7g8h9k",
   "firstName": "foo2",
   "lastNameName": "bar2",
   "createdAt": "Fri Jun 18 06:42:26 +0000 2011", //The Date follows ISO 8601
```

```
  }
]
{code}
```

Figure 4.1. JAX-RS generation result



GET users.format

Description: This is the way to get all the users a new user. The service return the created user data like id.

URL:

```
http://my.domain.lan/rest/private/api/project/v1/{portalContainerName}/users.format
```

Supported Format: application/json

Requires Authentication: true

Parameters:

- **Required (path parameters)**:
  - **portalContainerName**: The associated portal container name
  - **userId**: The specified user Id
- **Optional (query paramters)**:
  - **number**: The returned number

Request:

```
GET: http://my.domain.lan/rest/private/api/project/v1/portal/users.json
```

Response:

```
[
  {
    "id": "1a2b3c4d5e6f7g8h9i",
    "firstName": "foo1",
    "lastNameName": "bar1",
    "createdAt": "Fri Jun 17 06:42:26 +0000 2011", //The Date follows ISO 8601
  },
  {
    "id": "1a2b3c4d5e6f7g8h9k",
    "firstName": "foo2",
    "lastNameName": "bar2",
    "createdAt": "Fri Jun 18 06:42:26 +0000 2011", //The Date follows ISO 8601
  }
]
```

4.2.10.2. Chromattic

Example 4.34. Source code

```
@PrimaryType(name = "ns:myentity")
public abstract class MyEntity {

  /**
   * The property 1. It's a String property.
   */
  @Property(name = "ns:property1")
  public abstract String getProperty1();
  public abstract void setProperty1(String property1);

  /**
   * The property 2. It's an Integer property.
   */
  @Property(name = "ns:property2")
  public abstract Integer getProperty2();
  public abstract void setProperty2(String property2);

  /**
   * The property 3. It's a Boolean property.
   */
  @Property(name = "ns:property3")
  public abstract Boolean isProperty3();
  public abstract void setProperty3(Boolean property3);

  /**
   * The property 4. It's a String array property.
   */
  @Property(name = "ns:property4")
  public abstract String[] getProperty4();
  public abstract void setProperty4(String[] property4);

  /**
   * This is a node child.
   */
  @MappedBy("child")
  @OneToOne
  @Owner
  public abstract AnotherEntity getChild();
  public abstract void setChild(AnotherEntity anotherEntity);
}
```

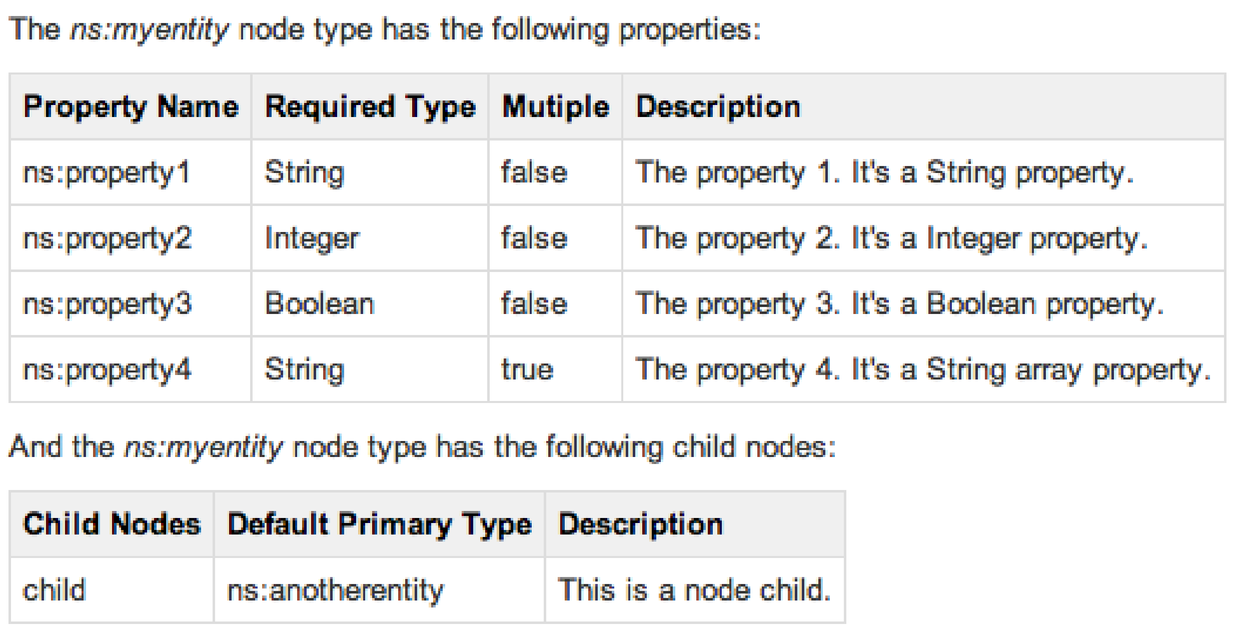
Example 4.35. Template

```
The _{noformat}${@PrimaryType.attribute("name")}{noformat}_ node type has the 
|| Property Name || Required Type ||Mutiple|| Description ||
<#list @PrimaryType.children("@Property") as property>
|${property.attribute("name")}|${property.type.name?replace("[]", "")}|${prope
</#list>
<#list @PrimaryType.children("@ManyToOne") as ref>
<#if ref.sibling("@MappedBy")??>
|${ref.sibling("@MappedBy").attribute("value")}|Reference|${ref.type.isArray}|
</#if>
</#list>

And the _{noformat}${@PrimaryType.attribute("name")}{noformat}_ node type has 
|| Child Nodes || Default Primary Type || Description ||
<#list @PrimaryType.children("@OneToOne", "@OneToMany") as child>
<#if child.sibling("@MappedBy")??>
|${child.sibling("@MappedBy").attribute("value")}|<#if child.type.annotation("
</#if>
</#list>
```

Example 4.36. Output

```
The _{noformat}ns:myentity{noformat}_ node type has the following properties:
|| Property Name || Required Type ||Mutiple|| Description ||
|ns:property1|String|false|The property 1. It's a String property.|
|ns:property2|Integer|false|The property 2. It's a Integer property.|
|ns:property3|Boolean|false|The property 3. It's a Boolean property.|
|ns:property4|String|true|The property 4. It's a String array property.|

And the _{noformat}ns:myentity{noformat}_ node type has the following child no
|| Child Nodes || Default Primary Type || Description ||
|child|ns:anotherentity|This is a node child.|
```

Figure 4.2. Chromattic generation result



The *ns:myentity* node type has the following properties:

Property Name	Required Type	Mutiple	Description
ns:property1	String	false	The property 1. It's a String property.
ns:property2	Integer	false	The property 2. It's a Integer property.
ns:property3	Boolean	false	The property 3. It's a Boolean property.
ns:property4	String	true	The property 4. It's a String array property.

And the *ns:myentity* node type has the following child nodes:

Child Nodes	Default Primary Type	Description
child	ns:anotherentity	This is a node child.