

Wikbook

Wikbook Reference Guide

Julien Viet

eXo Platform

Copyright © 2010 eXo Platform SAS

Table of Contents

Preface

1. Introduction

1.1. Document syntax

1.2. Wiki macros

2. How-to

2.1. Document structure

2.2. Document content

3. Project integration

3.1. Javatm code embedding

3.2. Maven integration

List of Examples

2.1. A code block

2.2. increment x

2.3. Document embedding

3.1. Example of Wikbook Maven plugin

3.2. Wikbook archetype usage

3.3. Configuration of the maven-source-plugin

3.4. Declaring the right dependency

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 Javatm 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.

Introduction

1.1. Document syntax

Wiki syntaxes are not unique, Wikbook relies on an external framework used at the hear 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 explicitly 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
<i>//italic//</i>	<i>italic</i>
<u>__underline__</u>	<u>underline</u>
„subscript,,	subscript
^^superscript^^	superscript
--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

<pre>= 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 /}}</pre>
<pre><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></pre>

Table 2.3. External links

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

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

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

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

Table 2.5. Char escape

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

Table 2.6. Backslash escape

<code>\\</code>
<code><para>\</para></code>

2.2.3. Blocks

2.2.3.1. Paragraphs

Unlike docbook, wiki paragraphs are implicitly 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

<pre>= 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.</pre>
<pre><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></pre>

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

<pre>* item 1 ** item 1.1 ** item 1.2 * item 2</pre>	• item 1 • item 1.1 • item 1.2 • item 2
<pre>1. item 1 11. item 1.1 11. item 1.2 1. item 2</pre>	1. item 1 1. item 1.1 2. item 1.2 2. item 2
<pre>(% style="upperroman" %) 1. item 1 11. item 1.1 11. item 1.2 1. item 2</pre>	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
 - lowerroman
 - upperalpha
 - upperroman

- arabicindic

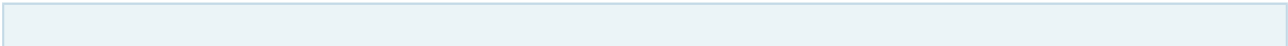
2.2.3.3. Tables

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

Table 2.9. Table examples

<pre> 1 2 3 4 5 6</pre>	Table 2.10. A simple table <table><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr></table>	1	2	3	4	5	6
1	2	3					
4	5	6					
<pre> =1 =2 =3 4 5 6</pre>	Table 2.11. A table with a row header <table><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr></table>	1	2	3	4	5	6
1	2	3					
4	5	6					
<pre> 1 2 3 =4 =5 =6</pre>	Table 2.12. A table with a row footer <table><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr></table>	1	2	3	4	5	6
1	2	3					
4	5	6					
<pre>(% title="The table" %) 1 2 3 4 5 6</pre>	Table 2.13. The table <table><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr></table>	1	2	3	4	5	6
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 group is declared inside a block like `((...))`.

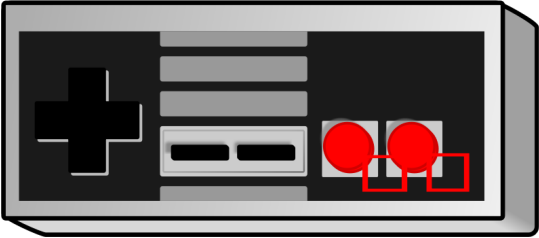
Table 2.14. A list inside a table with a group block

<pre> (((* item 1 * item 2)))</pre>	• item 1 • item 2
---------------------------------------	---

2.2.3.5. Images

Simple images can be displayed by using the image syntax.

Table 2.15. Image example

<code>[[image:images/controller.png]]</code>


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

Table 2.16. Image example

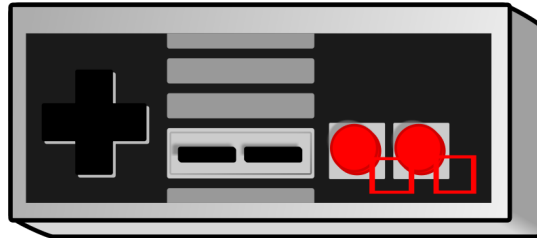
<code>An inline [[image:images/smallcontroller.png]] image</code>
An inline  image

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 embodied 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

<pre>{{code}}x = x + 1;{{/code}}</pre>
<pre>x = x + 1;</pre>

Table 2.20. A sample Java code

<pre>{{code language=java}}x = x + 1;{{/code}}</pre>
<pre>x = x + 1;</pre>

Table 2.21. A sample Groovy code

<pre>{{code language=groovy}}x = x + 2;{{/code}}</pre>
<pre>x = x + 2;</pre>

Table 2.22. A sample XML code

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

Table 2.23. A sample code from a file

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

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

<pre>{{java}}x = x + 1;{{/java}}</pre>
<pre>x = x + 1;</pre>

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

<pre>{{code language=java}}{{@include org.wikbook.doc.AnObject}}{{/code}}</pre>
<pre>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 } }</pre>

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

<pre>{{code language=java}}{{@include org.wikbook.doc.AnObject#foo(java.lang.String</pre>
<pre> public void foo(String s) { // An empty method }</pre>

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

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

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 implicitly declared namespace, mapped to the *wikbook* prefix.

Table 2.30. XML inclusion

<pre>{{xml}}<wikbook:include href="document.xml"/>{{/xml}}</pre>
<pre><foo> <bar/> </foo></pre>

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

Table 2.31. XML inclusion

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

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

<pre>{{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}}</pre>
<pre>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</pre>

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[™] language. A callout is declared inside a code block using comments in special format.

Table 2.33. A simple callout

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

```
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"); ❸
    System.out.println("This is going to the output"); ❹
}
```

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

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

2.2.4. Definition list

Wiki definition lists are managed as docbook variable lists.

Table 2.36. Definition lists

<pre> ; term : definition</pre>
<pre> <variablelist> <varlistentry> <term>term</term> <listitem> <para>definition</para> </listitem> </varlistentry> </variablelist></pre>
term definition

<pre>(% title="the title" %) ; term : definition</pre>
<pre><variablelist> <title>the title</title> <varlistentry> <term>term</term> <listitem> <para>definition</para> </listitem> </varlistentry> </variablelist></pre>
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

<pre>{{example}}an example{{/example}}</pre>
an example

<pre>{{example title="my example"}}an example with a title{{/example}}</pre>
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

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

2.2.6. Quotation

Quotation generates a set of DocBook blockquote structure.

Table 2.39. Quotation

<pre>>John said this >>Marie answered that I said ok</pre>
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

<pre>{{include document="embedded.wiki" /}}</pre>

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.

Project integration

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

3.1. Java[™] code embedding

Wikbook allows to embbed Java[™] 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</destinationDirectory>
    <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 fonctionnality:

- 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=0.9.34 \
  -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>
```

