Apache Maven Doxia

v. 1.8

User Guide

The Apache Software Foundation **2018-04-12**

Table of Contents

1. Table of Contents i
2. What is Doxia? ....
3. Overview 2
4. FAQ 4
5. What's new in 1.1 7
6. References 8
7. Apt Format 10
8. Doxia Apt Enhancements 20
9. FML Format 23
10. Xdoc Format 26
11. Doxia Modules Guide 31
12. Doxia Macros Guide 34
13. Writing Books 39
14. Developer Centre 42
15. Create a Doxia Module 43
16. Create a Doxia Macro 45
17. Using the Sink API 47
18. Integration With Maven 51
19. Issues &amp; Gotchas 53
20. External Resources 55

1 What is Doxia?

1.1 Maven Doxia

Doxia is a content generation framework which aims to provide its users with powerful techniques for generating static and dynamic content: Doxia can be used in web-based publishing context to generate static sites, in addition to being incorporated into dynamic content generation systems like blogs, wikis and content management systems.

Doxia supports markup languages with simple syntaxes. Lightweight markup languages are used by people who might be expected to read the document source as well as the rendered output.

Doxia is used extensively by Maven and it powers the entire documentation system of Maven. It gives Maven the ability to take any document that Doxia supports and output it any format.

The current version of Doxia base framework is 1.8.

1.1.1 Maven Doxia Enhancements

- Upgrading from earlier versions

1.1.2 Brief History

Based on the (now defunct) Aptconvert project developed by Xmlmind, Doxia was initially hosted by Codehaus, to become a sub-project of Maven early in 2006.

1.1.3 Main Features

- Developed in Java
- Support of several markup formats: APT (Almost Plain Text), Confluence, Simplified DocBook, Markdown, FML (FAQ Markup Language), FO, iText, LaTeX, RTF, TWiki, XDoc (popular in Apache land), XHTML
- Easy to learn the syntax of the supported markup formats
- Macro support
- No need to have a corporate infrastructure (like wiki) to host your documentation
- Extensible framework
- Site Tools extension for site or document rendering
- Additional Tools like Doxia Converter
- IDE integration

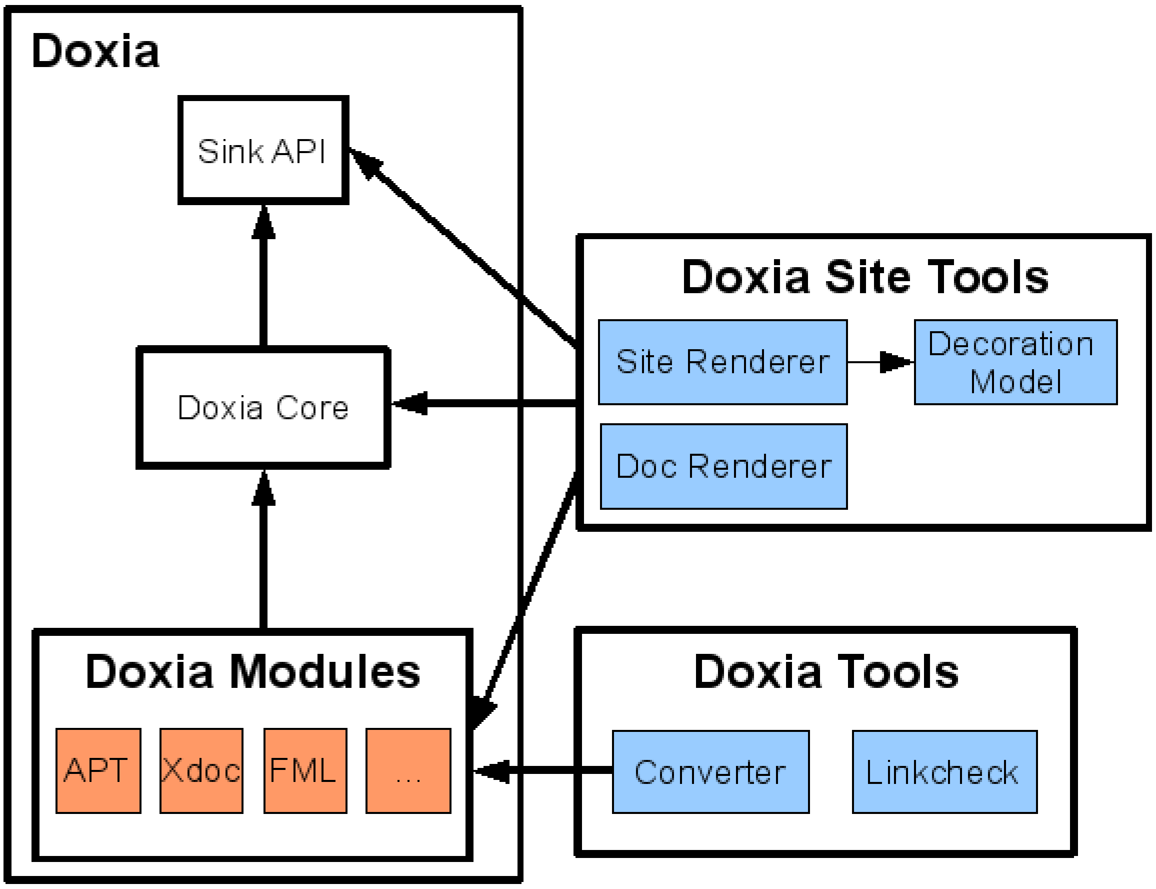
1.1.4 Doxia Reference Pages

See Doxia Markup Languages References page for a listing of all supported markups for each format.

2 Overview

2.1 Overview Of The Doxia Framework

The following figure represents the main components of the Doxia Framework.



Note: Just like Maven, Doxia uses Plexus extensively.

2.1.1 Sink API

The *Sink* interface is a generic markup language interface. It contains several methods that encapsulate common text syntax. A start tag is denoted by *xxxx()* method and a end of tag by *xxxx_()* method.

For instance, you could do things like:

```
sink.paragraph();
sink.text( "my text" );
sink.paragraph_();
```

similar to this HTML markup:

```
<p>my text</p>
```

To find out more about the Sink API, you could read the Javadoc here.

2.1.2 Doxia Core

The *Core* is the API to parse a source and populate it in a *Sink* object. The *Parser* interface contains only one method:

```
void parse( Reader source, Sink sink )
    throws ParseException;
```

The *ParseException* class has the responsibility to catch all parsing exceptions. It provides an helper method, *getLineNumber()*, which helps to find where an error occurred.

The *AbstractParser* class is an abstract implementation of the *Parser*. It provides a macro mechanism to give dynamic functionalities for the parsing. For more information on macros, read the Doxia Macro Guide.

Finally, the *SiteModule* interface is the last part of the puzzle. It provides main definitions of a given Doxia module and it is used by the *doxia-site-renderer* site tools.

2.1.3 Doxia Modules

A Doxia module is an implementation of a given markup language like APT or Xdoc. Each module should implement these interfaces:

- *Parser* interface, more specifically the *AbstractParser* class
- *SiteModule* interface

Several modules provide also a *Sink* implementation to handle a specific output markup language.

For more information on modules, read the Doxia Module Guide.

2.1.4 Doxia Site Tools

The *Site Tools* are a collection of tools to renderer an output. The main tool used by Maven, specifically the Maven Site Plugin, is the *doxia-site-renderer* which renders in HTML any documents written with supported markup syntax. It uses Velocity templates to customize the renderer and the *site-decoration-model* tool to decorate the renderer. This component describes the layout of the site defined in the *site.xml* file.

The *doxia-doc-renderer* tool is used to renderer any document in another document.

3 FAQ

3.1 Frequently Asked Questions

1. How to handle style in the APT markup language?
2. How to export in PDF?
3. Is it possible to create a book?
4. Why XML based sinks don't generate nicely formatted documents?
5. Where are the Maven Doxia XSD schemas files?
6. How to define character entities in Doxia XML files with XSD?
7. How to integrate Doxia 1.1 with Maven?

How to handle style in the APT markup language?

APT does not support style. If you need more control you should use xdoc instead.

[top]

How to export in PDF?

There are two modules available that can be used to generate pdf output: an iText module that uses the iText framework, and a FO module, that can be used e.g. in conjunction with Apache FOP to generate a pdf. Unfortunately, the iText team has discontinued the XML to PDF functionalities, so probably only the fo module is going to be supported in the future.

For Maven there is a pdf plugin available.

[top]

Is it possible to create a book?

Doxia also has a fairly simple tool for writing books. It comes complete with a Maven plugin to produce PDFs, LaTeX documents and Xdoc for direct integration in your Maven site. The Doxia Book code is still limited but fully functional.

See Writing Books in Doxia for more information.

[top]

Why XML based sinks don't generate nicely formatted documents?

We decided to keep pretty printing out of the core modules. So, XML based sinks like Xdoc or XHTML are intentionally unformatted. You could always do this after the document generation or directly by creating a specialized end-user sink (see DOXIA-255).

[top]

Where are the Maven Doxia XSD schemas files?

The Doxia XSD files are located here:

Xdoc XSD 2.0

https://maven.apache.org/xsd/xdoc-2.0.xsd

FML XSD 1.0.1

https://maven.apache.org/xsd/fml-1.0.1.xsd

Book XSD 1.0

https://maven.apache.org/xsd/book-1.0.0.xsd

Document XSD 1.0.1

https://maven.apache.org/xsd/document-1.0.1.xsd

Decoration XSD 1.0

https://maven.apache.org/xsd/decoration-1.0.0.xsd

Your favorite IDE probably supports XSD schema's for Xdoc and FML files. You need to specify the following:

```
<document xmlns="http://maven.apache.org/XDOC/2.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/XDOC/2.0 http://maven.apache.org/x
  ...
</document>
```

```
<faqs xmlns="http://maven.apache.org/FML/1.0.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/FML/1.0.1 http://maven.apache.org/
  ...
</faqs>
```

```
<book xmlns="http://maven.apache.org/BOOK/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/BOOK/1.0.0 http://maven.apache.org
  ...
</book>
```

```
<document xmlns="http://maven.apache.org/DOCUMENT/1.0.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/DOCUMENT/1.0.1 http://maven.apache
  outputName="...">
  ...
</document>
```

```
<project xmlns="http://maven.apache.org/DECORATION/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/DECORATION/1.0.0 http://maven.apac
  ...
</project>
```

Note: for preformance reasons, all XSDs/DTDs use a cache in ${java.io.tmpdir}.

[top]

How to define character entities in Doxia XML files with XSD?

Since it is not possible to define character entity references (like &copy;) in XSDs (unlike DTDs), each XML file should have a <!DOCTYPE> to define the character entity set.

For instance, you could add the following in your Xdoc XML files to be similar to XHTML 1.0 Transitional dtd:

```
<!DOCTYPE document [
  <!-- These are the entity sets for ISO Latin 1 characters for the XHTML -->
  <!ENTITY % HTMLlat1 PUBLIC "-//W3C//ENTITIES Latin 1 for XHTML//EN" "http://w
  %HTMLlat1;
  <!-- These are the entity sets for special characters for the XHTML -->
  <!ENTITY % HTMLsymbol PUBLIC "-//W3C//ENTITIES Symbols for XHTML//EN" "http:/
  %HTMLsymbol;
  <!-- These are the entity sets for symbol characters for the XHTML -->
  <!ENTITY % HTMLspecial PUBLIC "-//W3C//ENTITIES Special for XHTML//EN" "http:
  %HTMLspecial;
]>
<document xmlns="http://maven.apache.org/XDOC/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/XDOC/2.0 http://maven.apache.org/xs
...
</document>
```

Note: if CDATA is used to specify entity, Doxia will replace & by &amp; (i.e "<![CDATA[&iexcl;]]>" becomes "&amp;iexcl;").

[top]

How to integrate Doxia 1.1 with Maven?

See this page.

[top]

4 What's new in 1.1

4.1 What's new in 1.1?

This document describes the changes made to Maven Doxia between versions 1.0 and 1.1.

4.1.1 Notable New Features

- **Logging**: added logging support with a new project called doxia-logging-api.
- **Sink API improvements**: added a new `SinkEventAttributes` interface to handle attributes. The `Sink` interface has been updated to use this new interface. Added new methods for comments and unknown events to the `Sink` interface.
- **SinkFactory**: all Sink implementations can be retrieved via a factory.
- **XSDs**: created several XSDs, in particular for FML and Xdoc.
- **Tools**: created some tools like a converter and a linkchecker to use Doxia outside of Maven.
- **New Sinks**: for Confluence, XLS-FO and Twiki.
- Applied a few modifications to the APT format.

The full list of changes for 1.1 can be found in our issue management system: 1.1 :: 1.1.1 :: 1.1.2 :: 1.1.3 :: 1.1.4

4.1.2 Binary Incompatibility

Please note that in version 1.1 a number of new methods were added to the Sink and SinkFactory APIs, which makes them binary incompatible with version 1.0.

However, maven reporting plugins have been kept binary compatible. If you are a Maven Plugin developer and you plan to switch to Doxia 1.1, please read this Maven Integration page to understand how to integrate correctly Doxia 1.1 with Maven.

5 References

5.1 Doxia Markup Languages References

The following table gives an overview of the markup languages currently supported by Doxia:

- if a *Parser* is available for a given format, it means that you can write your documentation in this language and Doxia can generate output from it,
- if a *Sink* is available, it means you can generate output in this format.

The source directory is the directory under which Maven expects source documents in this format (e.g. `src/site/apt/` for Apt), the file extension is the default file extension, and the parser id is gives the unique identifier that is used by plexus to lookup the corresponding component.

Format	Short description	Parser (input)	Sink (output)	Source Directory	File Extension	Doxia Module	Parser Id
Apt	Almost Plain Text	✓	✓	`apt`	`apt`	`doxia-module-apt`	`apt`
AsciiDoc	Asciidoctor Maven Plugin	✓	✗	`asciidoc`	`adoc`, `asciidoc`	`asciidoct maven-plugin`	`asciidoc`
Confluence	Confluence Enterprise Wiki	✓	✓ *	`confluenc`	`confluenc`	`doxia-module-confluenc`	`confluence`
Simplified DocBook	Simplified DocBook XML Standard	✓	✓	`docbook`	`xml`	`doxia-module-docbook-simple`	`docbook`
FML	FAQ Markup Language	✓	✗	`fml`	`fml`	`doxia-module-fml`	`fml`
iText	iText PDF Library	✗	✓			`doxia-module-itext`	
FO*	XSL formatting objects (XSL-FO)	✗	✓			`doxia-module-fo`	
LaTeX	LaTeX typesetting system	✗	✓			`doxia-module-latex`	
Markdown**	Markdown markup language	✓	✗	`markdown`	`md`, `markdown`**	`doxia-module-markdown`	`markdown`
RTF	Microsoft Rich Text Format	✗	✓			`doxia-module-rtf`	
TWiki*	TWiki Structured Wiki	✓	✓	`twiki`	`twiki`	`doxia-module-twiki`	`twiki`

Xdoc	XML Documentatic Format	✓	✓	xdoc	xml	doxia-module-xdoc	xdoc
XHTML	Extensible Hypertext Markup Language	✓	✓	xhtml	xhtml	doxia-module-xhtml	xhtml

Note some modules are not included per default with the site plugin. Have a look at the available modules here: http://repo.maven.apache.org/maven2/org/apache/maven/doxia/.
If you need to add module for the maven site plugin simply add it as a dependency of the plugin

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-site-plugin</artifactId>
  <version>3.2</version>
  <dependencies>
    <dependency>
      <groupId>org.apache.maven.doxia</groupId>
      <artifactId>doxia-module-markdown</artifactId>
      <version>1.3</version>
    </dependency>
  </dependencies>
</plugin>
```

* Since Doxia 1.1

** Since Doxia 1.3

*** Since Doxia 1.7

6 Apt Format

6.1 The APT format

APT stands for "Almost Plain Text". APT is a markup language that takes the hassle out of documentation writing by striving for simplicity. Its syntax resembles plain-text more than it resembles most other markup formats (such as HTML).

This document provides some examples of available APT formatting.

6.1.1 Important Notice

The information contained in this document corresponds to the original APT format as published by Xmlmind. In version 1.1 Maven Doxia has applied several modifications to this original format, see this separate document for a detailed description. Notable differences are highlighted below with a [Change] link.

The following sections contain formatted text that demonstrates the use of APT to create paragraphs, headers, sections, lists, code samples, figures, tables, rules, breaks, and text level elements such as font styles, anchors, and special characters. Boxes containing text in typewriter-like font are examples of APT source.

6.1.2 Document structure

A short APT document is contained in a single text file. A longer document may be contained in a ordered list of text files. For instance, first text file contains section 1, second text file contains section 2, and so on.

Note:

Splitting the APT document in several text files on a section boundary is not mandatory. The split may occur anywhere. However doing so is recommended because a text file containing a section is by itself a valid APT document.

A file contains a sequence of paragraphs and ``displays" (non paragraphs such as tables) separated by open lines.

A paragraph is simply a sequence of consecutive text lines.

```
  First line of first paragraph.
  Second line of first paragraph.
  Third line of first paragraph.

  Line 1 of paragraph 2 (separated from first paragraph by an open line).
  Line 2 of paragraph 2.
```

The indentation of the first line of a paragraph is the main method used by an APT processor to recognize the type of the paragraph. For example, a section title must not be indented at all.

A ``plain" paragraph must be indented by a certain amount of space. For example, a plain paragraph which is not contained in a list may be indented by two spaces.

```
My section title (not indented).

  My paragraph first line (indented by 2 spaces).
```

Indentation is not rigid. Any amount of space will do. You don't even need to use a consistent indentation all over your document. What really matters for an APT processor is whether the paragraph is not indented at all or, when inside a list, whether a paragraph is more or less indented than the first item of the list (more about this later).

```
    First paragraph has its first line indented by four
spaces. Then the author did not even bother to indent the
other lines of the paragraph.

  Second paragraph contains several lines which are all
  indented by two spaces. This style is much nicer than
  the one used for the previous paragraph.
```

Note that tabs are expanded with a tab width set to 8.

6.1.3 Document elements

6.1.3.1 Block level elements

6.Title

A title is optional. If used, it must appear as the first block of the document.

```
            ------
            Title
            ------
            Author
            ------
             Date
```

A title block is indented (centering it is nicer). It begins with a line containing at least 3 dashes (`---`).

After the first `---` line, one or several consecutive lines of text (implicit line break after each line) specify the title of the document.

This text may immediately be followed by another `---` line and one or several consecutive lines of text which specifies the author of the document.

The author sub-block may optionally be followed by a date sub-block using the same syntax.

The following example is used for a document with an title and a date but with no declared author.

```
            ------
            Title
            ------
            ------
             Date
            ------
```

The last line is ignored. It is just there to make the block nicer.

Note: the date has no specific format and will be parsed as string. But we recommend to use the ISO-8601 standard, since formatting a date varies around the world:

YYYY-MM-DD

where **YYYY** is the year in the Gregorian calendar, **MM** is the month of the year between 01 (January) and 12 (December), and **DD** is the day of the month between 01 and 31.

6.Paragraph

Paragraphs other than the title block may appear before the first section.

```
  Paragraph 1, line 1.
  Paragraph 1, line 2.

  Paragraph 2, line 1.
  Paragraph 2, line 2.
```

Paragraphs are indented. They have already been described in the Document structure section.

6.Section

Sections are created by inserting section titles into the document. Simple documents need not contain sections.

```
Section title

* Sub-section title

** Sub-sub-section title

*** Sub-sub-sub-section title

**** Sub-sub-sub-sub-section title
```

Section titles are not indented. A sub-section title begins with one asterisk (`*`), a sub-sub-section title begins with two asterisks (`**`), and so forth up to four sub-section levels.

6.List

```
      * List item 1.

      * List item 2.

        Paragraph contained in list item 2.

            * Sub-list item 1.

            * Sub-list item 2.

      * List item 3.
```

List items are indented and begin with a asterisk (`*`).

Plain paragraphs more indented than the first list item are nested in that list. Displays such as tables (not indented) are always nested in the current list.

To nest a list inside a list, indent its first item more than its parent list. To end a list, add a paragraph or list item less indented than the current list.

Section titles always end a list. Displays cannot end a list but the `[]` pseudo-element may be used to force the end of a list.

```
      * List item 3.
        Force end of list:

      []

--------------------------------------------
Verbatim text not contained in list item 3
--------------------------------------------
```

In the previous example, without the `[]`, the verbatim text (not indented as all displays) would have been contained in list item 3.

A single `[]` may be used to end several nested lists at the same time. The indentation of `[]` may be used to specify exactly which lists should be ended. Example:

```
      * List item 1.

      * List item 2.

        * Sub-list item 1.

        * Sub-list item 2.

        []

-------------------------------------------------------------------
Verbatim text contained in list item 2, but not in sub-list item 2
-------------------------------------------------------------------
```

There are three kind of lists, the bulleted lists we have already described, the numbered lists and the definition lists.

```
      [[1]] Numbered item 1.

                [[A]] Numbered item A.

                [[B]] Numbered item B.

      [[2]] Numbered item 2.
```

A numbered list item begins with a label between two square brackets. The label of the first item establishes the numbering scheme for the whole list:

`[[1]]`

Decimal numbering: 1, 2, 3, 4, etc.

`[[a]]`

Lower-alpha numbering: a, b, c, d, etc.

`[[A]]`

Upper-alpha numbering: A, B, C, D, etc.

`[[i]]`

Lower-roman numbering: i, ii, iii, iv, etc.

[[I]]

Upper-roman numbering: I, II, III, IV, etc.

The labels of the items other than the first one are ignored. It is recommended to take the time to type the correct label for each item in order to keep the APT source document readable.

```
      [Defined term 1] of definition list 2.

      [Defined term 2] of definition list 2.
```

A definition list item begins with a defined term: text between square brackets.

6.Verbatim text

```
----------------------------------------
Verbatim
         text,
                preformatted,
                                escaped.
----------------------------------------
```

A verbatim block is not indented. It begins with a non indented line containing at least 3 dashes (`---`). It ends with a similar line.

`+--` instead of `---` draws a box around verbatim text.

Like in HTML, verbatim text is preformatted. Unlike HTML, verbatim text is escaped: inside a verbatim display, markup is not interpreted by the APT processor.

6.Figure

```
[Figure name] Figure caption
```

A figure block is not indented. It begins with the figure name between square brackets. The figure name is optionally followed by some text: the figure caption.

The figure name is the pathname of the file containing the figure but without an extension. Example: if your figure is contained in `/home/joe/docs/mylogo.jpeg`, the figure name is `/home/joe/docs/mylogo`. [Change]

If the figure name comes from a relative pathname (recommended practice) rather than from an absolute pathname, this relative pathname is taken to be relative to the directory of the current APT document (a la HTML) rather than relative to the current working directory.

Why not leave the file extension in the figure name? This is better explained by an example. You need to convert an APT document to PostScript and your figure name is `/home/joe/docs/mylogo`. An APT processor will first try to load `/home/joe/docs/mylogo.eps`. When the desired format is not found, a APT processor tries to convert one of the existing formats. In our example, the APT processor tries to convert `/home/joe/docs/mylogo.jpeg` to encapsulated PostScript.

6.Table

A table block is not indented. It begins with a non indented line containing an asterisk and at least 2 dashes (`*--`). It ends with a similar line.

The first line is not only used to recognize a table but also to specify column justification. In the following example,

- the second asterisk (`*`) is used to specify that column 1 is centered,
- the plus sign (`+`) specifies that column 2 is left aligned,
- the colon (`:`) specifies that column 3 is right aligned.

```
*----------*-------------+----------------:
| Centered | Left-aligned | Right-aligned  |
| cell 1,1 | cell 1,2     | cell 1,3       |
*----------*-------------+----------------:
| cell 2,1 | cell 2,2     | cell 2,3       |
*----------*-------------+----------------:
Table caption
```

Rows are separated by a non indented line beginning with `*--`.

An optional table caption (non indented text) may immediately follow the table.

Rows may contain single line or multiple line cells. Each line of cell text is separated from the adjacent cell by the pipe character (`|`). (`|` may be used in the cell text if quoted: `\|`.)

The last `|` is only used to make the table nicer. The first `|` is not only used to make the table nicer, but also to specify that a grid is to be drawn around table cells.

The following example shows a simple table with no grid and no caption.

```
*-----*------*
 cell | cell
*-----*------*
 cell | cell
*-----*------*
```

6.Horizontal rule

```
=====================
```

A non indented line containing at least 3 equal signs (===).

6.Page break

```
^L
```

A non indented line containing a single form feed character (Control-L).

6.1.3.2 Text level elements

6.Font

```
  <Italic> font. <<Bold>> font. <<<Monospaced>>> font.
```

Text between < and > must be rendered in italic. Text between << and >> must be rendered in bold. Text between <<< and >>> must be rendered using a monospaced, typewriter-like font.

Font elements may appear anywhere except inside other font elements.

It is not recommended to use font elements inside titles, section titles, links and defined terms because an APT processor automatically applies appropriate font styles to these elements.

6.Anchor and link

```
{Anchor}. Link to {{anchor}}. Link to {{http://www.pixware.fr}}.
Link to {{{anchor}showing alternate text}}.
Link to {{{http://www.pixware.fr}Pixware home page}}.
```

Text between curly braces (`{}`) specifies an anchor. Text between double curly braces (`{{}}`) specifies a link.

It is an error to create a link element that does not refer to an anchor of the same name. The name of an anchor/link is its text with all non alphanumeric characters stripped. [Change]

This rule does not apply to links to *external* anchors. Text beginning with `http:/`, `https:/`, `ftp:/`, `file:/`, `mailto:`, `../`, `./` (`..\` and `.\` on Windows) is recognized as an external anchor name.

When the construct **{{{** *name* **}** *text* **}}** is used, the link text *text* may differ from the link name *name*.

Anchor/link elements may appear anywhere except inside other anchor/link elements.

Section titles are implicitly defined anchors. [Change]

6.Line break

```
Force line\
break.
```

A backslash character (`\`) followed by a newline character.

Line breaks must not be used inside titles and tables (which are line oriented blocks with implicit line breaks).

6.Non breaking space

```
Non\ breaking\ space.
```

A backslash character (`\`) followed by a space character.

6.Special character

```
Escaped special characters: \~, \=, \-, \+, \*, \[, \], \<, \>, \{, \}, \\.
```

In certain contexts, these characters have a special meaning and therefore must be escaped if needed as is. They are escaped by adding a backslash in front of them. The backslash may itself be escaped by adding another backslash in front of it.

Note that an asterisk, for example, needs to be escaped only if its begins a paragraph. (`*` has no special meaning in the middle of a paragraph.)

```
Copyright symbol: \251, \xA9, \u00a9.
```

Latin-1 characters (whatever is the encoding of the APT document) may be specified by their codes using a backslash followed by one to three octal digits or by using the `\x` *NN* notation, where *NN* are two hexadecimal digits.

Unicode characters may be specified by their codes using the `\u` *NNNN* notation, where *NNNN* are four hexadecimal digits.

6.Comment

```
~~Commented out.
```

Text found after two tildes (~~) is ignored up to the end of line.

A line of ~ is often used to ``underline'' section titles in order to make them stand out of other paragraphs.

```
            ------
            Title
            ------
            Author
            ------
             Date

  Paragraph 1, line 1.
  Paragraph 1, line 2.

  Paragraph 2, line 1.
  Paragraph 2, line 2.

Section title

* Sub-section title

** Sub-sub-section title

*** Sub-sub-sub-section title

**** Sub-sub-sub-sub-section title

      * List item 1.

      * List item 2.

        Paragraph contained in list item 2.

            * Sub-list item 1.

            * Sub-list item 2.

      * List item 3.
        Force end of list:

      []

+------------------------------------------+
Verbatim text not contained in list item 3
+------------------------------------------+

      [[1]] Numbered item 1.

                [[A]] Numbered item A.

                [[B]] Numbered item B.

      [[2]] Numbered item 2.

  List numbering schemes: [[1]], [[a]], [[A]], [[i]], [[I]].

      [Defined term 1] of definition list.

      [Defined term 2] of definition list.

+-------------------------------+
Verbatim text
                        in a box
+-------------------------------+

  --- instead of +-- suppresses the box around verbatim text.
```

7 Doxia Apt Enhancements

7.1 Enhancements to the APT format

In the following we provide a list of differences/enhancements to the original APT format that were incorporated in Doxia. Note that the original specification still applies to **Doxia-1.0** (used e.g. by Maven-2.0.x), the changes outlined here only apply from **Doxia-1.1** (used by Maven >= 2.1.x). Apart from some exceptions, these differences are usually 'backwards-compatible', i.e. any document that gets correctly processed by Aptconvert should also be a valid Doxia input file and lead to identical results when processed by a Doxia parser.

- Paragraphs in list items
- Table header cells
- Multi-lines cells in table
- Anchors
- Links
- Figure extensions

7.1.1 Paragraphs in list items

Contrary to the original APT parser, the Doxia APT parser does not put list items within paragraphs. Eg, the example given in the APT guide:

```
    * List item 1.
```

will, for instance, produce the following html:

```
    <li>List item 1.</li>
```

To get the same result as aptconvert, use

```
    *

      List item 1.
```

which will produce:

```
    <li><p>List item 1.</p></li>
```

7.1.2 Table header cells

Header cells are defined by an additional pipe character '|' at the beginning of the cell:

```
*-----------+-----------+
|| Header 1 || Header 2 |
*-----------+-----------+
| Cell 1    | Cell 2    |
*-----------+-----------+
```

produces:

Header 1	Header 2
Cell 1	Cell 2

7.1.3 Multi-lines cells in table

Since 1.1, multi-lines cells are recognized with the character '\' at the end of the cells:

```
*-----------+-----------+
|| Header 1 || Header 2 |
*-----------+-----------+
| Cell\     | Cell 2    |
| 1         |           |
*-----------+-----------+
```

produces:

Header 1	Header 2
Cell 1	Cell 2

7.1.4 Anchors

7.1.4.1 Anchors for section titles

Contrary to the original APT format, section titles are **not** implicitly defined anchors. If you want an anchor for a section title you need to define it explicitly as such:

```
* {Anchors for section titles}
```

7.1.4.2 Anchor construction

Contrary to the original APT format, an anchor/link is **not** its text with all non alphanumeric characters stripped. Ideally, an anchor should be a valid Doxia id, ie it must begin with a letter ([A-Za-z]) and may be followed by any number of letters, digits ([0-9]), hyphens ("-"), underscores ("_"), colons (":"), and periods ("."). Any anchor that does not satisfy this pattern is transformed according to the following rules:

- Any whitespace at the start and end is removed
- If the first character is not a letter, prepend the letter 'a'
- Any spaces are replaced with an underscore '_'
- Any characters not matching the above pattern are stripped.

Note in particular that case is preserved in this conversation and that APT anchors and links are case-sensitive. So the anchor for the section title in the previous example would be `Anchors_for_section_titles`.

7.1.5 Links

In **Doxia-1.1** the notion of a ' *local*' link was introduced in addition to *internal* links and *external* links.

- An **internal** link is a link to an anchor within the same source document.

  In the APT format used by **Doxia-1.1**, internal links have to be valid Doxia ids, as specified in the anchors section above.

  Note in particular that internal links in APT do **not** start with '#'.
- A **local** link is a link to another document within the same site.

  In the APT format used by **Doxia-1.1**, local links **have to** start with either `./` or `../` to distinguish them from internal links. E.g.,

  ```
  {{{doc/standalone.html}Standalone}}
  ```

  is not valid, it should be

  ```
  {{{./doc/standalone.html}Standalone}}
  ```

  Note in particular that the following link

  ```
  {{{standalone.html}Standalone}}
  ```

  will be interpreted as an internal link (dots are valid characters in anchor names). Since you most likely meant to link to another source document, you should again prepend a "./".
- An **external** link is a link that is neither local nor internal.

  An external link should be a valid URI.

Anchors are always translated to a valid id, including escaping. In some situation this can cause issues, especially when referring to a javadoc-link.
Since Doxia 1.4 there is support for literal anchors, by using 2 hashed (##) instead of 1. This implies that the writer is responsible for using the right URL encoding!

```
{{{../apidocs/groovyx/net/http/ParserRegistry.html##parseText(org.apache.http.HttpRes
```

7.1.6 Figure extensions

Contrary to the original APT format, a figure name has to be given fully **with** extension. For instance:

```
[/home/joe/docs/mylogo.jpeg] Figure caption
```

8 FML Format

8.1 The FML format

8.1.1 Overview

An 'fml' (FAQ Markup Language) is an XML document conforming to a small and simple set of tags. The format was first used in the Maven 1, version of the FAQ plugin.

8.1.2 The FML format

The full documentation is available at here.

8.1.3 FML Sample

The following is a sample FML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<faqs xmlns="http://maven.apache.org/FML/1.0.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/FML/1.0.1 http://maven.apache.org/xsd
  title="Frequently Asked Questions"
  toplink="false">

  <part id="general">
    <title>General</title>

    <faq id="whats-foo">
      <question>
        What is Foo?
      </question>
      <answer>
        <p>some markup goes here</p>

        <source>some source code</source>

        <p>some markup goes here</p>
      </answer>
    </faq>

    <faq id="whats-bar">
      <question>
        What is Bar?
      </question>
      <answer>
        <p>some markup goes here</p>
      </answer>
    </faq>
  </part>

  <part id="install">

    <title>Installation</title>

    <faq id="how-install">
      <question>
        How do I install Foo?
      </question>
      <answer>
        <p>some markup goes here</p>
      </answer>
    </faq>

  </part>

</faqs>
```

8.2 Validation

Doxia is able to validate your fml files as described here.

9 Xdoc Format

9.1 Content

- Content
- The XDoc format
    - Overview
    - The XDoc xsd
    - XDoc Sample
    - The <source> tag
    - Additional sectioning
    - Referencing sections and subsections
- Validation
    - Don't nest block level elements
    - Put inline elements inside block level elements
    - Right order of elements in <properties>
    - Dont put <source> inside paragraphs

9.2 The XDoc format

9.2.1 Overview

An 'xdoc' is an XML document conforming to a small and simple set of tags. Xdoc was the primary documentation format in Maven 1, Maven 2 largely replaced this by Apt, but xdoc is still supported.

Historically, the xdoc format can be traced back to the Anakia format, as once used by the Apache Jakarta project then moved to Velocity.

The Maven 1 Xdoc plugin introduced a few additions to the Anakia format, they are highlighted in the plugin documentation.

9.2.2 The XDoc xsd

The full documentation is available here.

9.2.3 XDoc Sample

The following is a sample XDoc document:

```
<?xml version="1.0" encoding="UTF-8"?>
<document xmlns="http://maven.apache.org/XDOC/2.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/XDOC/2.0 http://maven.apache.org/xsd/

  <properties>
    <title>Page Title</title>
    <author email="user@company.com">John Doe</author>
  </properties>

  <!-- Optional HEAD element, which is copied as is into the XHTML <head> element -
  <head>
    <meta ... />
  </head>

  <body>

    <!-- The body of the document contains a number of sections -->
    <section name="section 1">

      <!-- Within sections, any XHTML can be used -->
      <p>Hi!</p>

      <!-- in addition to XHTML, any number of subsections can be within a section
      <subsection name="subsection 1">
        <p>Subsection!</p>
      </subsection>

    </section>

    <section name="other section">

      <!-- You can also include preformatted source blocks in the document -->
      <source>
code line 1
code line 2
      </source>

    </section>

  </body>

</document>
```

9.2.4 The <source> tag

`<source>` tags are special. Anything within this tag is rendered within a "verbatim box" as pre-formatted text. If you are embedding other XML/XHTML markup within the source tags, then you need to place a CDATA section within the source section. Example:

```
<source><![CDATA[ content here <a href="">foo</a>]]></source>
```

9.2.5 Additional sectioning

Doxia will produce `<h2>` and `<h3>` headings for `<section>` and `<subsection>` elements, respectively. It is therefore perfectly valid to put some sub-headings (`<h4>`, `<h5>`, `<h6>`) inside a subsection. For instance,

```
<h4>A subsubsection</h4>
```

will produce:

9.2.5.1 A subsubsection

9.2.6 Referencing sections and subsections

The core doxia modules do **not** construct anchors from section/subsection names. If you want to reference a section, you should either provide an explicit anchor:

```
<a name="Section1"/>
<section name="Section">

  <a name="SubSection1"/>
  <subsection name="SubSection">
  </subsection>

</section>
```

or use an `id` attribute for section and subsections (note that `id`'s have to be unique within one xdoc source document):

```
<section name="Section" id="Section1">

  <subsection name="SubSection" id="SubSection1">
  </subsection>

</section>
```

Note that this differs from previous behavior, where anchors were constructed from section/ subsection names, replacing special characters by underscores. This behavior presents two shortcomings:

- If two sections or subsections have identical names (within one source document), you will get an ambiguity when referencing them. Also the resulting html document will not be valid XHTML. For other output formats (eg pdf), it might even be impossible to generate the target document.
- For long section titles, this leads to rather cumbersome anchor names.

If automatic anchor generation is desired for a particular output format, it should be implemented / overridden by the corresponding low-level Sink.

9.3 Validation

Doxia is able to validate your xdoc files as described here.

Here is a list of common mistakes to be aware of:

9.3.1 Don't nest block level elements

Wrong:

```
<p>
  Here's a list:
  <ul>
    <li>item 1</li>
    <li>item 2</li>
  </ul>
  of things to do.
</p>
```

Correct:

```
<p>
  Here's a list:
</p>
<ul>
  <li>item 1</li>
  <li>item 2</li>
</ul>
<p>
  of things to do.
</p>
```

Typical block level elements are list elements, `<table>`, `<source>`, `<div>`, `<p>` and `<pre>`.

9.3.2 Put inline elements inside block level elements

Wrong:

```
<section name="Downloads">
  <a href="downloads.html">Downloads</a>
</section>
```

Correct:

```
<section name="Downloads">
  <p>
    <a href="downloads.html">Downloads</a>
  </p>
</section>
```

Typical inline elements are `<a>`, `<strong>`, `<code>`, `<font>`, `<br>` and `<img>`.

9.3.3 Right order of elements in <properties>

The `<title>` element has to come before `<author>`.

9.3.4 Dont put <source> inside paragraphs

Wrong:

```
<p>
  The following command executes the program:
  <source>java -jar CoolApp.jar</source>
</p>
```

Correct:

```
<p>
  The following command executes the program:
</p>
<source>java -jar CoolApp.jar</source>
```

However, you may put `<source>` elements inside list items or table rows.

10 Doxia Modules Guide

10.1 Doxia Modules Guide

Doxia has several built-in modules that support some standard markup languages, see the References page for an overview. The following is just a collection of reference links for the individual formats.

- APT
- Confluence
- Simplified DocBook
- FML
- iText
- FO (since Doxia 1.1)
- LaTeX
- Markdown (since Doxia 1.3)
- RTF
- TWiki (since Doxia 1.1)
- XDoc
- XHTML

10.1.1 APT

APT (Almost Plain Text) is a simple text format.

References:

- Apt Reference

10.1.2 Confluence

Confluence is an Enterprise wiki from Atlassian. It uses Textile inside as an APT language.

References:

- Confluence Notation Guide Overview
- Confluence Element Reference

10.1.3 Simplified DocBook

DocBook is a markup language for technical documentation. Simplified DocBook is a simpler subset.

References:

- Simplified DocBook Introduction
- Simplified DocBook Element Reference

10.1.4 FML

FML (FAQ Markup Language) is a FAQ markup language.

References:

- FML Reference

10.1.5 iText

iText is a free Java/PDF library.

References:

- iText tutorial

10.1.6 FO (since Doxia 1.1)

XSL formatting objects (XSL-FO)

References:

- XSL-FO Recommendation (05 December 2006)
- XSL FO reference
- Apache FOP

10.1.7 LaTeX

LaTeX is a popular document markup language.

References:

- LaTeX2e for authors
- Latex reference sheet

10.1.8 Markdown (since Doxia 1.3)

Markdown is a widespread Markup language.

References:

- Official Markdown project at Daring Fireball

10.1.9 RTF

RTF is a proprietary document file format.

References:

- Microsoft Office Word 2007 Rich Text Format (RTF) Specification
- RTF Cookbook

10.1.10 TWiki (since Doxia 1.1)

TWiki is a structured wiki.

References:

- TWiki Text Formatting

10.1.11 XDoc

XDoc is a generic format for document into a styled HTML document.

References:

- XDoc Reference

10.1.12 XHTML

XHTML is a markup language with the same expressions as HTML, but also conforms to XML syntax.

References:

- HTML and XHTML Quick Reference Charts: Head and Body Markup
- XHTML 1.0 Specification

11 Doxia Macros Guide

11.1 Doxia Macros Guide

The Doxia *Core* includes macro mechanisms to facilitate the documentation writing.

Macros are currently only supported for APT, Xdoc and FML formats. Starting with Doxia 1.7 (maven-site-plugin 3.5), macros are also supported for XHTML and Markdown. Macros are not (and probably will never be) supported by Confluence, Docbook and Twiki modules.

A macro in an APT source file is a **non-indented** line that looks like this:

```
%{macro_name|param1=value1|param2=value2|...}
```

An Xdoc or FML macro has the following syntax:

```
<macro name="macro_name">
  <param name="param1" value="value1"/>
  <param name="param2" value="value2"/>
  ...
</macro>
```

Since Doxia 1.7, an XHTML or Markdown macro has the following syntax:

```
<!-- MACRO{macro_name|param1=value1|param2=value2|...} -->
```

As of Doxia 1.7, the following macros are available:

- Echo Macro
- Snippet Macro
- TOC Macro
- SWF Macro
- SSI Macro

11.1.1 Echo Macro

The *Echo* macro is a very simple macro: it prints out the key and value of any supplied parameters. For instance, in an APT file, you could write:

```
%{echo|param1=value1|param2=value2}
```

Similarly, it will be for xdoc file:

```
<macro name="echo">
  <param name="param1" value="value1"/>
  <param name="param2" value="value2"/>
</macro>
```

and it will output

```
  param1 ---> value1
  param2 ---> value2
```

11.1.2 Snippet Macro

The *Snippet* macro is a very useful macro: it prints out the content of a file or a URL. For instance, in an APT file, you could write:

```
%{snippet|id=myid|url=http://myserver/path/to/file.txt}
```

In a xdoc file, it will be:

```
<macro name="snippet">
  <param name="id" value="myid"/>
  <param name="url" value="http://myserver/path/to/file.txt"/>
</macro>
```

The `id` parameter is not required if you want to include the entire file. If you want to include only a part of a file, you should add start and end demarcators: any line (typically a comment) that contains the strings " `START`", " `SNIPPET`" and " `myid`" (where `myid` is the `id` of the snippet) is a start demarcator, and similarly " `END SNIPPET myid`" denotes the end of the snippet to include. For example:

- Start and end snippets in a Java file

```
public class MyClass
{
    // START SNIPPET: myid
    public static void main( String[] args ) throws Exception
    {
        ...
    }
    // END SNIPPET: myid
}
```

- Start and end snippets in a XML file

```
<project>
...
  <build>
    <plugins>
<!-- START SNIPPET: myid -->
      <plugin>
        ...
      </plugin>
<!-- END SNIPPET: myid -->
    </plugins>
  </build>
</project>
```

Parameter	Description
id	The id of the snippet to include. If omitted the whole file/url will be included (since Doxia 1.1).
url	The path of the URL to include.
file	The path of the file to include (since doxia-1.0-alpha-9).

verbatim	If the content should be output as verbatim escaped text. If this is set to `false` then the content of the snippet will not be escaped. This means that you can use it like Server-Side Includes on a webserver. Default value is `true`.
encoding	The encoding of the file to read (since Doxia 1.6). If omitted the default JVM encoding will be used.

11.1.3 TOC Macro

The *TOC* macro prints a Table Of Content of a document. It is useful if you have several sections and subsections in your document. For instance, in an APT file, you could write:

```
%{toc|section=2|fromDepth=2|toDepth=3}
```

This displays a TOC for the second section in the document, including all subsections (depth 2) and sub-subsections (depth 3).

Note that in Doxia, apt section titles are not implicit anchors (see Enhancements to the APT format), so you need to insert explicit anchors for links to work!

In a xdoc file, it will be:

```
<macro name="toc">
  <param name="section" value="2"/>
  <param name="fromDepth" value="0"/>
  <param name="toDepth" value="4"/>
</macro>
```

Parameter	Description
section	Display a TOC for the specified section only, or all sections if 0. Positive int, not mandatory, 0 by default.
fromDepth	Minimum section depth to include in the TOC (sections are depth 1, sub-sections depth 2, etc.). Positive int, not mandatory, 0 by default.
toDepth	Maximum section depth to include in the TOC. Positive int, not mandatory, 5 by default.

From **Doxia 1.1.1** on you may also specify any of the html base attributes (*i.e.* any of `id`, `class`, `style`, `lang`, `title`) as parameters, e.g.:

```
%{toc|class=myTOC}
```

This can be used for styling the TOC via css.

11.1.4 SWF Macro

The *Swf* macro prints Shockwave Flash assets in the documentation. For instance, in an APT file, you could write:

```
%{swf|src=swf/myfile.swf|id=MyMovie|width=600|height=200}
```

In a xdoc file, it will be:

```
<macro name="swf">
  <param name="src" value="swf/myfile.swf"/>
  <param name="id" value="MyMovie"/>
  <param name="width" value="600"/>
  <param name="height" value="200"/>
</macro>
```

Parameter	Description
src	Specifies the location (URL) of the movie to be loaded.
id	Identifies the Flash movie to the host environment (a web browser, for example) so that it can be referenced using a scripting language.
width	Specifies the width of the movie in either pixels or percentage of browser window.
height	Specifies the height of the movie in either pixels or percentage of browser window.
quality	Possible values: low, high, autolow, autohigh, best.
menu	True displays the full menu, allowing the user a variety of options to enhance or control playback. False displays a menu that contains only the Settings option and the About Flash option.
loop	Possible values: true, false. Specifies whether the movie repeats indefinitely or stops when it reaches the last frame. The default value is true if this attribute is omitted.
play	Possible values: true, false. Specifies whether the movie begins playing immediately on loading in the browser. The default value is true if this attribute is omitted.
version	Specifies the width of the movie in either pixels or percentage of browser window.
allowScript	Specifies the width of the movie in either pixels or percentage of browser window.

For more information, see the SWF Macro page.

11.1.5 SSI Macro

Since Doxia 1.7, the *SSI* macro prints a server side include. For instance, in an APT file, you could write:

```
%{ssi|function=include|file=included-file.html}
```

In a xdoc file, it will be:

```
<macro name="ssi">
  <param name="function" value="include"/>
  <param name="file" value="included-file.html"/>
</macro>
```

Parameter	**Description**
function	The SSI function to insert.
any	Parameter that will be added to the SSI directive.

12 Writing Books

12.1 Introduction

Doxia allows you to write books like user manuals and guides in any format supported by Doxia. Combined with the Doxia Book Maven you are able to include the manuals directly in your generated site with links to the off-line friendly formats like XDoc, PDF, RTF and LaTeX.

The Xdoc output which has been rendered into this site can be viewed here.

The generated PDF is here and the generated RTF here.

12.1.1 How It Works

The only thing you need in addition to the content files itself is a simple book descriptor which is used to specify the ordering of the sections and the names for the chapters.

See The Book Descriptor Reference for a reference to the descriptor.

12.1.2 Creating a Book Descriptor

An XML file is used to describe the layout of the book.

A sample is given below:

```
<book xmlns="http://maven.apache.org/BOOK/1.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/BOOK/1.0.0 http://maven.apache.org/xsd
  <id>doxia-example-book</id>
  <title>XFire User Manual</title>
  <chapters>
    <chapter>
      <id>bind</id>
      <title>Bindings</title>
      <sections>
        <section>
          <id>bindings</id>
        </section>
        <section>
          <id>aegis-binding</id>
        </section>
        <section>
          <id>castor</id>
        </section>
      </sections>
    </chapter>
    <chapter>
      <id>transports</id>
      <title>Transports</title>
      <sections>
        <section>
          <id>transport-and-channel-api</id>
        </section>
        <section>
          <id>http-transport</id>
        </section>
        <section>
          <id>jms-transport</id>
        </section>
        <section>
          <id>local-transport</id>
        </section>
      </sections>
    </chapter>
  </chapters>
</book>
```

12.1.3 Configuring Doxia Book Maven Plugin

This example illustrates how to configure the Doxia Book Maven Plugin. It will render the book in three different formats. By default the output will be under `target/generated-site/<format>/<book id>` .

The currently supported format ids are: `xdoc`, `pdf`, `latex`, `rtf`, `xhtml`, `doc-book`

A sample `pom.xml` is given below:

```
<plugin>
  <groupId>org.apache.maven.doxia</groupId>
  <artifactId>doxia-book-maven-plugin</artifactId>
  <version>1.3-SNAPSHOT</version>
  <executions>
    <execution>
      <phase>pre-site</phase>
      <goals>
        <goal>render-books</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <books>
      <book>
        <directory>content/books/example-book</directory>
        <descriptor>content/books/example-book.xml</descriptor>
        <formats>
          <format>
            <id>latex</id>
          </format>
          <format>
            <id>xdoc</id>
          </format>
          <format>
            <id>pdf</id>
          </format>
          <format>
            <id>rtf</id>
          </format>
        </formats>
      </book>
    </books>
  </configuration>
</plugin>
```

13 Developer Centre

13.1 Doxia Developers Centre

This documentation centre is for those that are developing Doxia modules or macro.

Currently you can find information on the following topics:

- Creating Doxia Modules
- Creating Doxia Macros
- Using the Doxia Sink API

14 Create a Doxia Module

14.1 Create a New Doxia Module

First, you need to create a POM with *doxia-modules* as parent:

```
<project>
  <parent>
    <groupId>org.apache.maven.doxia</groupId>
    <artifactId>doxia-modules</artifactId>
    <version>1.0</version> <!-- Latest release -->
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <artifactId>doxia-module-my</artifactId>
  <name>Doxia :: MY Module</name>

  ...
</project>
```

Secondly, you should implement some Doxia classes:

- *MyParser* class

```
import org.apache.maven.doxia.parser.AbstractParser;

/**
 * @plexus.component role="org.apache.maven.doxia.parser.Parser" role-hint="my"
 */
public class MyParser
    extends AbstractParser
{
...
}
```

- *MyParseException* class (optional)

```
import org.apache.maven.doxia.parser.ParseException;

public class MyParseException
    extends ParseException
{
...
}
```

- *MySiteModule* class (optional, will be used by doxia-sitetools)

```
import org.apache.maven.doxia.module.site.AbstractSiteModule;

/**
 * @plexus.component role="org.apache.maven.doxia.module.site.SiteModule" role-h
 */
public class MySiteModule
    extends AbstractSiteModule
{
...
}
```

- *MySink* class

```
import org.apache.maven.doxia.sink.SinkAdapter;

public class MySink
    extends SinkAdapter
{
...
}
```

- *MySinkFactory* class

```
import org.apache.maven.doxia.sink.SinkFactory;

/**
 * @plexus.component role="org.apache.maven.doxia.sink.SinkFactory" role-hint="m
 */
public class MySinkFactory
    extends SinkFactory
{
...
}
```

14.1.1 References

- Doxia Modules Guide
- Doxia Macros Guide
- Doxia API Reference
- Doxia Sitetools API Reference

15 Create a Doxia Macro

15.1 Create a New Doxia Macro

You need to add the following plugin configuration to generate the correct Plexus *component.xml* file for the project containing your macro:

```
<project>
  ...
  <build>
    ...
    <plugins>
      <plugin>
        <groupId>org.codehaus.plexus</groupId>
        <artifactId>plexus-maven-plugin</artifactId>
        <executions>
          <execution>
            <goals>
              <goal>descriptor</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
      ...
    </plugins>
  ...
  </build>
  ...
</project>
```

You should implement the *AbstractMacro* class:

```
import org.apache.maven.doxia.macro.AbstractMacro;

/**
 * @plexus.component role="org.apache.maven.doxia.macro.Macro" role-hint="my"
 */
public class MyMacro
    extends AbstractMacro
{
...
    public void execute( Sink sink, MacroRequest request )
        throws MacroExecutionException
    {
        String myValue = (String) request.getParameter( "myParam" );
...
    }
...
}
```

To use it, you need to write the following markups:

- APT

```
%{my|myParam=myValue} <!-- my is the macro name defined by role-hint -->
```

- XDoc

```
<macro name="my"> <!-- my is the required macro name defined by role-hint -->
  <param name="myParam" value="myValue"/>
</macro>
```

15.1.1 References

- Doxia Modules Guide
- Doxia Macros Guide
- Doxia API Reference
- Doxia Sitetools API Reference

16 Using the Sink API

16.1 Using the Doxia Sink API

- Transforming documents
- Generating documents
- Passing attributes to Sink events
- Avoid sink.rawText!
- How to inject javascript code into HTML
- References

16.1.1 Transforming documents

Doxia can be used to transform an arbitrary input document to any supported output format. The following snippet shows how to use a Doxia *Parser* to transform an apt file to html:

```
File userDir = new File( System.getProperty ( "user.dir" ) );
File inputFile = new File( userDir, "test.apt" );
File outputFile = new File( userDir, "test.html" );

SinkFactory sinkFactory = (SinkFactory) lookup( SinkFactory.ROLE, "html" ); // Pl

Sink sink = sinkFactory.createSink( outputFile.getParentFile(), outputFile.getNam

Parser parser = (AptParser) lookup( Parser.ROLE, "apt" ); // Plexus lookup

Reader reader = ReaderFactory.newReader( inputFile, "UTF-8" );

parser.parse( reader, sink );
```

It is recommended that you use Plexus to look up the parser. In principle you could instantiate the parser directly (`Parser parser = new AptParser();`) but then some special features like macros will not be available.

You could also use the Doxia Converter Tool to parse a given file/dir to another file/dir.

16.1.2 Generating documents

The snippet below gives a simple example of how to generate a document using the Doxia Sink API.

```
    /**
     * Generate a simple document and emit it
     * into the specified sink. The sink is flushed but not closed.
     *
     * @param sink The sink to receive the events.
     */
    public static void generate( Sink sink )
    {
        sink.head();

        sink.title();
        sink.text( "Title" );
        sink.title_();

        sink.author();
        sink.text( "Author" );
        sink.author_();

        sink.date();
        sink.text( "Date" );
        sink.date_();

        sink.head_();


        sink.body();

        sink.paragraph();
        sink.text( "A paragraph of text." );
        sink.paragraph_();

        sink.section1();
        sink.sectionTitle1();
        sink.text( "Section title" );
        sink.sectionTitle1_();

        sink.paragraph();
        sink.text( "Paragraph in section." );
        sink.paragraph_();

        sink.section1_();

        sink.body_();

        sink.flush();
    }
```

A more complete example that also shows the 'canonical' order of events to use when generating a document, can be found in the Doxia SinkTestDocument class.

16.1.3 Passing attributes to Sink events

With Doxia 1.1 a number of methods have been added to the Sink API that allow to pass a set of attributes to many sink events. A typical use case would be:

```
SinkEventAttributeSet atts = new SinkEventAttributeSet();
atts.addAttribute( SinkEventAttributes.ALIGN, "center" );

sink.paragraph( atts );
```

What kind of attributes are supported depends on the event and the sink implementation. The sink API specifies a list of suggested attribute names that sinks are expected to recognize, and parsers are expected to use preferably when emitting events.

16.1.4 Avoid sink.rawText!

In **Doxia 1.0** it was a common practice to use sink.rawText() to generate elements that were not supported by the Sink API. For example, the following snippet could be used to generate a styled HTML <div> block:

```
sink.RawText( "<div style=\"cool\">" );
sink.text( "A text with a cool style." );
sink.rawText( "</div>" );
```

This has a major drawback however: it only works if the receiving Sink is a HTML Sink. In other words, the above method will not work for target documents in any other format than HTML (think of the FO Sink to generate a pdf, or a LaTeX sink,...).

In **Doxia 1.1** a new method unknown() was added to the Sink API that can be used to emit an arbitrary event without making special assumptions about the receiving Sink. Depending on the parameters, a Sink may decide whether or not to process the event, emit it as raw text, as a comment, log it, etc.

The correct way to generate the above <div> block is now:

```
SinkEventAttributeSet atts = new SinkEventAttributeSet();
atts.addAttribute( SinkEventAttributes.STYLE, "cool" );

sink.unknown( "div", new Object[]{new Integer( HtmlMarkup.TAG_TYPE_START )}, atts )
sink.text( "A text with a cool style." );
sink.unknown( "div", new Object[]{new Integer( HtmlMarkup.TAG_TYPE_END )}, null );
```

Read the javadocs of the unknown() method in the Sink interface and the XhtmlbaseSink for information on the method parameters. Note that an arbitrary sink may be expected to ignore the unknown event completely!

In general, the rawText method should be avoided alltogether when emitting events into an arbitrary Sink.

16.1.5 How to inject javascript code into HTML

Related to the above, here is the correct way of injecting a javascript snippet into a Sink:

```
// the javascript code is emitted within a commented CDATA section
// so we have to start with a newline and comment the CDATA closing in the end
// note that the sink will replace the newline by the system EOL
String javascriptCode = "\n function javascriptFunction() {...} \n //";

SinkEventAttributeSet atts = new SinkEventAttributeSet();
atts.addAttribute( SinkEventAttributes.TYPE, "text/javascript" );

sink.unknown( "script", new Object[]{new Integer( HtmlMarkup.TAG_TYPE_START )}, att
sink.unknown( "cdata", new Object[]{new Integer( HtmlMarkup.CDATA_TYPE ), javascrip
sink.unknown( "script", new Object[]{new Integer( HtmlMarkup.TAG_TYPE_END )}, null
```

16.1.6 References

- Doxia Modules Guide
- Doxia Macros Guide
- Doxia API Reference
- Doxia Sitetools API Reference

17 Integration With Maven

17.1 Integration With Maven

This page presents how to use Doxia 1.1 under Maven 2.0.x and 2.1.x with a Maven reporting plugin. Its goal is to help the Maven reporting plugin developer to integrate it.

17.1.1 Maven 2.0.x

Doxia 1.0 API is embedded in Maven 2.0.x (see MNG-3402), so your Maven reporting plugin needs to shade Doxia 1.1 API and Logging to be backward compatible with Maven 2.0.x.

```
<project>
  ...
  <build>
    ...
    <plugins>
      <plugin>
        <artifactId>maven-shade-plugin</artifactId>
        <version>1.2</version>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>shade</goal>
            </goals>
            <configuration>
              <finalName>${project.build.finalName}</finalName>
              <createDependencyReducedPom>false</createDependencyReducedPom>
              <keepDependenciesWithProvidedScope>true</keepDependenciesWithProvided
              <transformers>
                <transformer implementation="org.apache.maven.plugins.shade.resourc
              </transformers>
              <artifactSet>
                <includes>
                  <include>org.apache.maven.doxia:doxia-sink-api</include>
                  <include>org.apache.maven.doxia:doxia-logging-api</include>
                </includes>
              </artifactSet>
            </configuration>
          </execution>
        </executions>
      </plugin>
      ...
    </plugins>
  </build>
  ...
</project>
```

17.1.2 Maven 2.1.x

Doxia 1.1 API and Logging are embedded in Maven 2.1.x, your Maven reporting plugin is directly compatible with 2.1.x.

17.2 Common Bugs and Pitfalls

Please read the Doxia Issues page.

18 Issues &amp; Gotchas

18.1 Doxia Issues & Gotchas

This document collects some infos about specific issues and 'gotchas' when working with Doxia. Please check also the Frequently Asked Questions.

- Apt anchors and links
- TOC macro issues
- Verbatim blocks not boxed
- Figure sink events
- Empty Generated Page

18.1.1 Apt anchors and links

Using **Doxia 1.1** you may see a lot of warnings when processing old Apt source files:

```
[WARNING] [Apt Parser] Ambiguous link: 'doxia-apt.html'. If this is a local link, p
```

and

```
[WARNING] [Apt Parser] Modified invalid link: references/doxia-apt.html
```

The reason is that in Apt, links to other source documents have to start with either `./` or `../` to distinguish them from internal links. Please read the sections on anchors and links in our Apt guide. Note in particular that internal links in Apt do **not** start with '#'.

You should pay attention to these warnings since your links will most likely be broken. Unfortunately, the warning message cannot indicate the source file with the broken link (see eg MPDF-11), however, if you run in `DEBUG` mode, eg invoke maven with the `-X` switch, you can see which source document is being parsed when the warning is emitted.

18.1.2 TOC macro issues

There was a particular bug in the TOC macro in version 1.0 that has been fixed but leads to a backward incompatibility in some cases. If you have specified the `section` parameter in your TOC, as for instance:

```
%{toc|section=1|fromDepth=1|toDepth=1}
```

then the generated TOC is probably different from the result with Doxia 1.0. In **Doxia 1.1**, depth=1 is section, depth=2 is sub-section, etc, as documented in the macro guide.

18.1.3 Verbatim blocks not boxed

There was a particular bug in Doxia 1.0 that verbatim blocks were always boxed. If after an upgrade to **Doxia 1.1** you find that your blocks are not boxed anymore, check that you actually start your verbatim block with `+--` (and not `---`).

18.1.4 Figure sink events

Doxia distinguishes between figures, which are block-level elements, and images (or icons), which are in-line elements. For instance, the following sequence of sink events

```
sink.figure( null );

sink.figureGraphics( "figure.png", null );

sink.figureCaption( null );
sink.text( "Figure caption", null );
sink.figureCaption_();

sink.figure_();
```

should output the equivalent of this html snippet:

```
<div class="figure">
  <p><img src="figure.png"/></p>
  <p>Figure caption</p>
</div>
```

while the `figureGraphics( ... );` event alone can be used to generate an in-line image, i.e. just the `<img>` tag in case of html.

Note that we are using the forms that take a `SinkEventAttributeSet` above, even though we are just passing in null values. The reason is that the alternative forms (without `SinkEventAttributeSet`) have a different behavior, which is kept for backward compatibility (but the methods have been deprecated). Using the same sequence of sink events as above, but omitting the `null` method parameters, will generate

```
<img src="figure.png" alt="Figure caption"/>
```

18.1.5 Empty Generated Page

After running `mvn site` using your Maven reporting plugin, you see that the generated page is empty. Be sure that the the code calls `sink.close()`.

19 External Resources

19.1 External Resources

19.1.1 Extensions

Name	Author
Doxia :: Include Macro	Juergen Kellerer

19.1.2 Articles

Title	Publisher	Author
Quick and dirty typesetting with APT	linux.com	Scott Nesbitt
Lightweight markup language	wikipedia.org	?
Simple (Ascii-Based) Text Formats	project.knowledgeforge.net	?

19.1.3 Tools

Name	Author
APT Editor (Eclipse plugin)	Mathieu Avoine
nb-doxia-support (NetBeans plugin)	Allan Lykke Christensen

19.2 Related External Projects

- XWiki which uses Doxia in its rendering.
- Mylyn WikiText (originally known as Textile-J).