Marakana Course Style Guide

Contents

Chapter 1

Coursebook Structure

1.1 Directory Structure

Each class' coursebook should be stored in a separate directory in Marakana's `courseware` Subversion project. The coursebook directory should have the following contents:

`book.asc`
: The main book file, used to include all other book files.

***`fileName`*`.asc`**
: A separate file for each course module. In rare cases, an especially long module may by split into multiple files, in which case all such files should be included into a top-level module file.

> **Tip**
> In general, modules that long should be avoided, as it would tend to indicate too much content for presentation as a single module. Consider re-designing that section of the course into a set of smaller, more focused modules if possible.

`attributes.asc`
: A file containing document attributes, included in the book file and each module file.

`docinfo.html`
: A set of HTML `<head>` elements that are added to the generated HTML files. This is used to supplement the set of `<head>` elements generated automatically by `asciidoc`.

`docinfo.xml`
: A set of XML DocBook elements containing the frontmatter for the coursebook. This is used to generate the PDF version of the book.

`images/`
: A directory containing all images included in the coursebook aside from common images stored in the `common` courseware directory in Subversion.

`code/`
: A directory containing all code samples included in the coursebook.

1.2 The `book.asc` Book File Structure

The `book.asc` file serves primarily as a container for including the other course module files. It also defines various attributes used to format the book.

1.2.1 The Book Header

The `book.asc` should start with a *header*, consisting of a set of lines much like this:

```
= The Hitchhiker's Guide to the Galaxy
Ford Prefect <ford.prefect@marakana.com>
v0.1, 11-Dec-2010
:leveloffset:   1
:docinfo1:
:doctype:   book
include::attributes.asc[]
```

The title line should be a "Level 0" header, as indicated by the single =. Next comes the primary author, version, and revision date information.

Note
Each time a course is updated, remember to update the version number and (final) revision date. This helps ensure that outdated materials are not used in class.

The `leveloffset` attribute increases the "indentation level" of sections in included files. For example, a Level 1 heading in an included file is treated as a Level 2 heading. This allows us to process individual modules for slideshow presentation with the proper section levels required by those tools (that is, Level 1 sections becoming individual slides), and still maintain the proper outline structure for print production.

The `docinfo1` attribute should be present, but does not require a value. It instructs AsciiDoc to include the `docinfo.xml` and `docinfo.html` meta information files for DocBook and HTML output generation, respectively.

The `doctype` attribute of "book" instructs the AsciiDoc toolchain to generate PDF output in a two-sided book format.

The included `attributes.asc` file contains additional document attributes that need to apply both to the entire coursebook when generating PDF output and to individual modules when generating slide output. Therefore, individual course modules also include this `attributes.asc` file independently.

1.2.2 The Book Preamble

AsciiDoc allows an optional untitled section body, known as a *preamble*, to be present between the document header and the first section title. Marakana coursebooks should not include a book preamble.

1.2.3 The Book Sections

Following the header, each course module should be included in order with a block `include` directive, as in:

```
include::structure.asc[]
```

Tip
Include a blank line after each `include` directive to ensure the title of the subsequent included document is not seen as part of the last paragraph of the previous document.

Any section that serves as an appendix, such as installation and setup instructions, should be proceeded by an `[appendix]` attribute, as in:

```
[appendix]
include::installation.asc[]
```

1.3 The Document Information Files

There are two document information files that should be provided for each coursebook: `docinfo.html` and `docinfo.xml`.

1.3.1 The `docinfo.html` File

The `docinfo.html` file provides a set of HTML `<head>` elements that are added to the generated HTML files. This is used to supplement the set of `<head>` elements generated automatically by `asciidoc`.

1.3.2 The `docinfo.xml` File

The `docinfo.xml` file provides a set of XML DocBook elements containing the frontmatter for the coursebook. This information includes the copyright notice and other legal information, which is then included in the PDF output.

1.4 The `attributes.asc` File

The included `attributes.asc` file contains additional document attributes that need to apply both to the entire coursebook when generating PDF output and to individual modules when generating slide output.

The following attributes and values are recommended for Marakana courses:

`copyright`
: The copyright year(s), the copyright owner, and the phrase "All rights reserved." This attribute becomes a `<meta>` tag in the HTML generated for Slidy, and Slidy also presents the information in a footer on each slide.

`miscellaneous.tabsize`
: The number of space characters to be used for tab expansion. `4` is the recommended value.

Therefore a standard `attributes.asc` file would contain the following (with the date adjusted as appropriate):

```
:copyright:  2010 Markana, Inc. All rights reserved.
:miscellaneous.tabsize:  4
```

1.5 The Module File Structure

Each course module should appear in a separate file. The file should have a descriptive basename and an `.asc` extension.

Tip
The Marakana TextMate bundle includes a New File template for creating a new module file. Select `File ⇒ New From Template ⇒ Marakana ⇒ New Marakana AsciiDoc Course Module`.

1.5.1 The Module Header

The module file should start with a *header*, consisting of a set of lines much like this:

```
= Lemur Husbandry
include::attributes.asc[]
```

The title line should be a "Level 0" header, as indicated by the single =.

The included `attributes.asc` file contains additional document attributes that need to apply both to the entire course-book when generating PDF output and to individual modules when generating slide output. Therefore, individual course modules also include this `attributes.asc` file independently. Any attributes explicitly required by this module should be set after including this file.

1.5.2 The Module Preamble

AsciiDoc allows an optional untitled section body, known as a *preamble*, to be present between the document header and the first section title. Each module should have a preamble consisting of an outline of topics covered in the module.

This preamble appears on the first page of the PDF output for the module. For a Slidy presentation, it appears as a separate slide following the title slide.

1.5.3 Slide Structure

Each course slide should have the following basic structure, replacing the `TODO` notes with actual content:

```
== Slide Title

// TODO: Slide content goes here

[role="handout"]
--
'''

// TODO: Student notes content goes here

--
```

The slide title should be a "Level 1" header, as indicated by a double ==.

Tip
The Marakana TextMate bundle includes a tab trigger to insert a new slide. Type `slide` followed by the Tab key.

1.5.3.1 Main Slide Content

The main slide content will be the only slide content visible in a Slidy presentation of the course. The primary concepts of this slide should appear here in bullet form.

1.5.3.2 Slide Notes

The two `--` lines delimit what AsciiDoc refers to as an "open block," which is used to group a set of block elements but otherwise provides no special formatting. When generating HTML output, AsciiDoc wraps the content of an open block in a `<div>` element. The `[role="handout"]` attribute preceding the block becomes an HTML class applied to the `<div>`. The `''` line provides a horizontal rule separating the main slide content from the notes.

The AsciiDoc toolchain renders the slide notes normally for the PDF output. On the other hand, Slidy hides all `<div>` elements that have a "handout" class.

If no student notes are required for a particular slide, the lines from `[role="handout"]` through the end may be omitted.

1.5.4 Module Review

Try to include a review slide at the end of the module summarizing key points covered. Don't simple repeat the outline from the beginning of the module. Instead, try to provide concrete information, such as:

- You must terminate each Java statement with a semicolon (;).
- A Tcl dictionary is an ordered collection of key-value pairs.

1.5.5 Labs

Try to include at least one hands-on lab at the end of each module. For longer modules, consider labs at intermediate points as well.

A lab should be presented as a slide. The slide body should have a brief summary of the lab. The notes section of the slide should go into more detail to guide the students in the lab.

The lab instructions should not be a list of explicit steps for the students to follow. A lab should require students to apply the knowledge that they gained during the module. More detailed instructions may be required to guide students through obscure steps of the lab (e.g., directing them to specific menus or dialogs), and explicit steps might be required on occasion if some aspect of the lab was not covered in class or is tangential to the main class subject matter (e.g., expanding tarballs or setting file permissions), could result in lost data, or could require extensive time and effort to recover from a mis-step.

1.5.6 Exercises and Quizzes

To be written

1.6 Other Coursebook Sections

1.6.1 Installation and Configuration Sections

Unless software installation and configuration are covered as in-class activities, they should be documented in an appendix.

In the Marakana `courseware` Subversion project, the `common/installation` directory contains installation and configuration instructions for common software packages used in the courses. In your installation appendix, simply `include` as many of these files as needed for the class.

Whenever possible, new installation instructions should be provided as a new `common/installation` file so that it can be reused in other courses. The content should be simple book text, not presentation slides. The top-level title should be a "Level 1" header, as indicated by a double ==. The installation and configuration process should be presented as a numbered set of tasks. Make the instructions concrete and specific as much as possible, to avoid student mistakes. If the installation and configuration procedure has variations, such as different steps to follow on Linux vs Windows, clearly differentiate between the variations. If differences are minor, present them as alternatives to specific steps. If they are major, include each installation procedure in a separate section.

1.6.2 Glossary of Terms

Consider whether students would benefit from a glossary. If so, provide it as a separate file included in the `book.asc` after all appendices. The structure of the file should be:

```
[glossary]
= Glossary of Terms
include::attributes.asc[]

[glossary]
A term::
Its definition

Another term::
Its definition

...
```

1.6.3 Resource List

The last module should be a list of resources. Include books, web sites, and any other applicable resources. In general try to follow the *Chicago Manual of Style* format for bibliographic entries. However, there is no need to get overly formal following the style; especially for web resources, the Chicago and other formats can be overly complex, so feel free to simplify.

Tip
The site BibMe offers a free service for compiling bibliographic entries. It allows searching for books by title, author, ISBN, etc. and automatically filling out most of the bibliographic information. Be careful, though, as it sometimes misparses some of the information. It's safest to check the publication information against the Library of Congress Online Catalog.

1.7 Images

All images required for the course should be included in an `images/` subdirectory. Images used in multiple courses, such as the Marakana logo, should go in the `common/images` directory in Marakana's `courseware` Subversion project.

Where possible, images should be in PNG format. JPG or GIF format may also be used. When creating images using a tool with a different native file format (such as OmniGraffle), provide the native source for the image as well.

To be written

1.8 Code Files

To be written

1.9 Lab Files

To be written

Chapter 2

Formatting and Style

2.1 Typographical Conventions

In AsciiDoc, you format text by *quoting* it within what AsciiDoc refers to as quote characters.

There are actually two kinds of quotes:

Constrained quotes
: Quotes must be bounded by white space or commonly adjoining punctuation characters. These are the most commonly used type of quote.

Unconstrained quotes
: Unconstrained quotes have no boundary constraints and can be placed anywhere within inline text. Unconstrained quotes are double-ups of the _, *, + and # constrained quotes. For example `**N**ew` renders as **N**ew.

Keep in mind:

- Quoting cannot be overlapped.
- Different quoting types can be nested.
- To suppress quoted text formatting place a backslash character immediately in front of the leading quote character(s). In the case of ambiguity between escaped and non-escaped text you will need to escape both leading and trailing quotes, in the case of multi-character quotes you may even need to escape individual characters.

2.1.1 Italicized Text

Italicize text by quoting it with 'single quotes' or _underscores_. Generally prefer using underscores over quotes, but use whichever is most convenient. If the italicized text does not begin or end at a word boundary, use the unconstrained version, with two `__under__score` characters quoting the text.

Explicitly italicize the following content:

Glossary term first use
: In AsciiDoc, you format text by *quoting* it.

2.1.2 Bold Text

Format text as bold text by quoting it with *asterisks*. If the bold text does not begin or end at a word boundary, use the unconstrained version, with two `**aster**isk` characters quoting the text.

Explicitly format in bold the following content:

Emphasized text
: Make **emphasized** text bold.

2.1.3 Monospaced Text

Format text as monospaced text by quoting it with `back quotes` or +plus+ characters. If the monospaced text does not begin or end at a word boundary, use the unconstrained version, with two `++pl++us` characters quoting the text.

Note
The back-quote technique for producing monospace text also disables all other quote expansion within the text. This is typically used for literal text such as code that might include other AsciiDoc significant characters.

Explicitly format in monospace text the following content:

Class names
: Create a subclass of the `Activity` class.

Method and function names
: The `toString()` method returns a printable representation. Invoke `myObj.getLength()` to obtain the length.

Field and variable names
: Set `name` to "Ken Jones".

Language keywords
: `while` executes the loop statement if the condition evaluates to `true`.

Menu names, menu items, window and dialog field names
: Display the `File` menu and select `Save`. Enter your name in the `Name` field of the dialog.

File and directory paths
: Install the application into `/opt/local`. The `images` directory contains PNG and JPG images.

HTML and XML elements and attributes
: Provide a reasonable value for the `alt` attribute of your `<img>` tags.

2.1.4 Parameterized Literal Text

Sometimes, a portion of literal text needs a parameter or placeholder value. Ideally, the parameter or placeholder should be rendered in italicized monospace text. Unfortunately, AsciiDoc does not make this easy, especially if the parameter or placeholder does not begin and end at word boundaries. In that situation, you must use the "unbounded" version of the italics quotes.

Complicating matters further, the DocBook DTD does not allow any formatted text to occur within literal text spans. In essence, this HTML equivalent is not valid in DocBook: `<tt>MyApp_<i>version</i>.exe</tt>`. The `asciidoc` tool does not complain, but the `a2x` tool aborts when the `xmllint` application detects the invalid structure.

Note
You can disable the `xmllint` step by providing the `a2x -L` option. However, you should not use this hack, as it could hide other structural errors in your document.

You can work around this limitation by using an obvious placeholder, like `XXX` or `VERSION` in the literal, and call out the replacement in the explanatory text. Otherwise, at the point you need the parameter or placeholder:

1. Turn off literal text
2. Turn on italics
3. Turn on literal text
4. Write the placeholder text
5. Turn off literal text
6. Turn off italics
7. Turn on literal text

For example, `++MyApp_++__++version++__++.exe++` renders as `MyApp_`*`version`*`.exe`. Yuck.

Examples of parameterized literal text include:

Version numbers
: Download `MyApp_`*`version`*`.zip` to your local system.

File or directory names
: The file should go in the `workspace/`*`projectName`*`/src` directory

2.1.5 URLs

AsciiDoc automatically creates a hyperlink when it encounters a URL. Type display link text other than the URL, append it to the URL enclosed in `[]` characters, as in `http://www.marakana.com[Marakana]`, which produces Marakana.

2.1.6 Quoted Text

Aside from the typical use of quotation characters in English, use quotation marks around literal strings in the following situations in normal text:

Text a user enters into a field on window or dialog
: Type "Ken Jones" in the `Name` field.

Text a user enters at a command-line prompt
: Enter "San Francisco" at the `City:` prompt.

Literal strings to be used in code
: Assign "Ken Jones" to the `name` variable.

2.2 User Interface Documentation

2.2.1 Windows and Dialogs

Use literal text format (monospace formatting) for:

- Window and dialog titles
- Window and dialog field names

Use quoted text (quotation marks) for literal text a user should enter into a field.

2.2.2 Menus

Use literal text format (monospace formatting) for menu names and menu items. To indicate a sequence of selecting an item from a menu, separate the menu from the menu item with a =>. For example:

- The text `+File=>Save+` produces `File⇒Save`
- The text `+Text=>Align=>Left+` produces `Text⇒Align⇒Left`

2.2.3 Command Line Interaction

When illustrating a sequence of command line interactions, use an AsciiDoc *listing block* (delimited by lines containing at least four − characters). The commands typed by the user should appear in bold monospace, which requires enabling quote substitutions within the listing block.

For example, the following code:

```
[subs="quotes"]
----
$ *ls*
one.txt two.txt two.txt.bak
$ *rm *.bak*
$ *ls*
one.txt two.txt
----
```

produces this output:

```
$ ls
one.txt two.txt two.txt.bak
$ rm *.bak
$ ls
one.txt two.txt
```

2.3 Code Examples

For code examples, use and AsciiDoc *listing block* (delimited by lines containing at least 4 - characters). The listing block should be preceded by a `[source]` attribute identifying the programming language for syntax highlighting. For example:

```
[source,tcl]
----
set a 3
set b 4
set c [expr { sqrt( $a*$a + $b*$b)}]
----
```

which produces this output:

```tcl
set a 3
set b 4
set c [expr { sqrt( $a*$a + $b*$b) }]
```

When code is available in an external source file and you want to include the entire content of the file as an example, use the `include` directive within the listing block, as in:

```
[source,java]
----
include::code/MyClass.java[]
----
```

> **Tip**
> The Marakana TextMate bundle includes a drag-and-drop shortcut. You can select any file in the TextMate Project Drawer or in the Mac OS Finder, and drag it to the desired include location in your file. The drag-and-drop shortcut automatically inserts an appropriate `include` directive for the file. Additionally, if the file is a recognized language type, as identified by its extension, the shortcut automatically adds the listing block syntax and a `[source]` attribute for that language.

Because a listing block is rendered with a border and background, a set of short code examples interspersed with text can appear visually cluttered. In that case, you can use a *literal block* (delimited by lines containing at least 4 `.` characters), which renders its contents in a monospace font but without a border or background. For example:

```
....
set a 3
set b 4
....
```

produces the output:

```
set a 3
set b 4
```

An alternative to using a literal block, especially for a single line, is to use a *literal paragraph*, which is a paragraph where the first line is indented by one or more space or tab characters.

For example, the line:

```
  set a 3
```

produces the output:

```
set a 3
```

2.4 Tables

To be written

Warning
Don't use the `literal` or `monospace` column types (i.e., in the `cols` attribute). I noticed weird rendering artifacts for both, especially when generating PDF. The row heights were unnecessarily tall, and cell contents sometimes were misaligned.

2.5 Images

The `image` directive includes a reference to an image to include in the document. The syntax is:

```
image:pathName[attributes]   // Inline image
image::pathName[attributes]  // Block image
```

Tip
The Marakana TextMate bundle includes a drag-and-drop shortcut. You can select any image file in the TextMate Project Drawer or in the Mac OS Finder, and drag it to the desired include location in your file. The drag-and-drop shortcut automatically inserts an appropriate `image::` directive for the file. If you want the image as an inline rather than block image, simply delete one of the `:` characters.

2.5.1 Inline vs Block Images

AsciiDoc supports both "inline" and "block" images.

Inline images
: An inline image does not cause a line break. It can appear anywhere within a paragraph or table. Use `image:` to include an inline image.

Block images
: A block image causes a line break at the point of insertion. It is roughly equivalent to a paragraph in terms of the document structure. It must be contained in a single line separated on each side by a blank line or a block delimiter. Use `image::` to include a block image.

2.5.2 Image Attributes

All image attributes are optional in AsciiDoc. Within the `[]`, you can provide a comma-separated list of *`attribute=value`* pairs. Optionally, the first attribute can be a simple string value, in which case it is used as the value for the `alt` attribute.

See the AsciiDoc documentation for a complete list of supported attributes. The following are attributes typically used in Marakana courses:

alt
: Alternative text which is displayed if the output application is unable to display the image file.

> **Note**
> In general, you should always provide a value for the `alt` attribute.

title
: A title for the image. The block image renders the title alongside the image. The inline image displays the title as a popup "tooltip" in visual browsers.

height , width
: HTML: A size expressed as a number of pixels. DocBook: A size expressed as a measurement, such as "2in". If you specify only one of these attributes, the image is scaled proportionally. If you specify both, the image may be stretched to fit the space.

align
: One of `center`, `left`, or `right`. Mutually exclusive with the `float` attribute.

float
: HTML only: One of `left` or `right`, floating the image on the page. Use the `unfloat::[]` block macro to stop floating.

2.5.3 Specifying Separate Attributes for HTML and DocBook

Getting an image to appear properly in both HTML and DocBook/PDF output formats can be challenging. For example, you might want to specifying a different size for HTML vs PDF production. Or you might want to display a title for PDF output, but not for HTML.

AsciiDoc supports conditional macros. Within the conditional section, you can include output text and other AsciiDoc directives. A simpler, condensed conditional syntax is also available for one-line conditional inclusion. See the AsciiDoc documentation for complete information on using conditional directives.

One condition that can be tested is the *basebackend*, which is the toolchain used to produce output. For the purpose of producing HTML, such as Slidy slides, then `basebackend-html` is defined. For the purpose of producing PDF, then `basebackend-docbook` is defined. This allows us to specify different sets of attributes for each target output.

The following is an example of exploiting this feature:

```
ifdef::basebackend-docbook[[width="2in",title="Compiling and running a Java program"]]
ifdef::basebackend-html[[float="left",width="250"]]
image::images/HelloWorld-CompileRun.png["Compiling and running a Java program"]
// Some content
unfloat::[]
```

In this case, the image `images/HelloWorld-CompileRun.png` is included as a block image, with an `alt` value of "Compiling and running a Java program". When generating PDF, the image will be scaled to 2 inches wide and be displayed with a title of "Compiling and running a Java program". When generating HTML, the image will be scaled to 250 pixels wide and float on the left side of the page. Text will wrap around the image until reaching the `unfloat::[]` directive.

Chapter 3

Resource List

[1] Ousterhout, John K., and Ken Jones. Tcl and the Tk toolkit. 2nd ed. Boston: Addison-Wesley, 2009.

[2] AsciiDoc Home Page. http://www.methods.co.nz/asciidoc

[3] BibMe: Fast & Easy Bibliography Maker. http://www.bibme.org