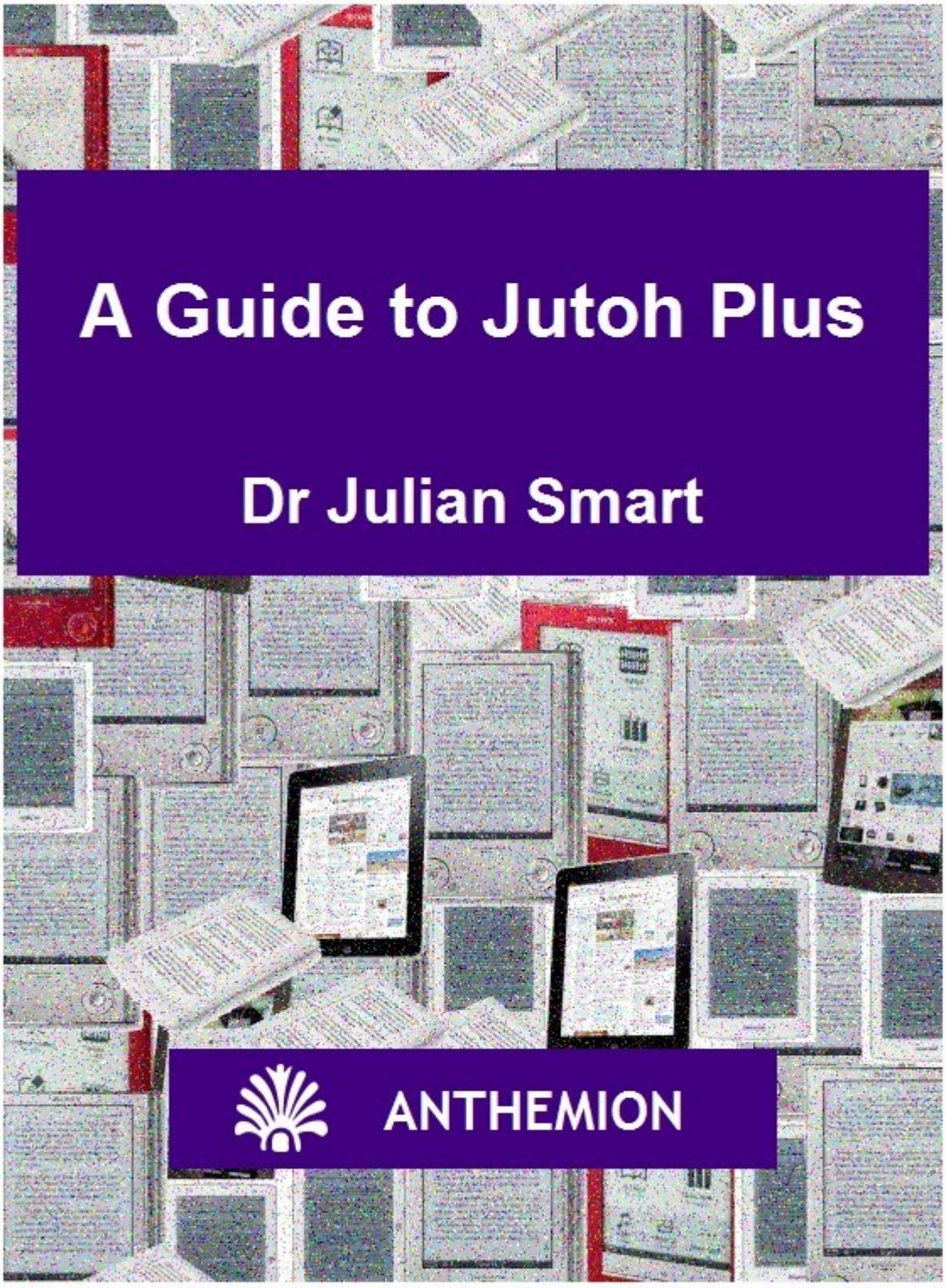

A Guide to Jutoh Plus
Dr Julian Smart
ANTHEMION

# A Guide to Jutoh Plus

by Julian Smart

Published by Anthemion



Edition 1.10


# Table of Contents

# Preface

## *About this book*

This guide is a supplement to *Creating Great Ebooks Using Jutoh* and the online help, and details the extra features in Jutoh Plus. You will learn how to use the scripting facilities which will let you automate ebook production and alteration of existing ebooks; and you will find out how to configure HTML generation via templates for complex projects.

Jutoh Plus is a mode that you can unlock by buying either the full Jutoh Plus key, or a Jutoh Plus Upgrade key which upgrades the licence from Jutoh to Jutoh Plus. If you already have the latest version of Jutoh, whether downloaded as a trial or by purchasing standard Jutoh, you can unlock Jutoh Plus by clicking on **Help | Register Jutoh** on the Jutoh menubar and entering your new Jutoh Plus key.

## *About the author*

Dr Julian Smart is technical director of Edinburgh-based Anthemion Software. He is the founder of the wxWidgets project, a popular construction kit for applications that run on a variety of computer platforms. Julian is the creator of Jutoh, and, with his novelist wife Harriet Smart, the Writer's Café toolkit for writers, as well as the DialogBlocks and HelpBlocks tools for programmers.

## *How this book is structured*

In Chapter 1: Introduction to Jutoh Plus, we describe the available features and how Jutoh helps to automate ebook production.

In Chapter 2: Script Syntax, the details of scripting are covered.

In Chapter 3: Using the Command Line, the options available on the command line are described.

In Chapter 4: Using HTML Templates, we describe how each HTML document can be tailored, allowing you to add further styles, JavaScript and custom HTML.

In Chapter 5: Creating HTML Help, we look at how Jutoh can be used to import and create Windows or wxWidgets application help files.

## *Conventions and terms used in this book*

- The convention **Menu | Command**, such as **View | Preferences**, indicates a menu and the command on that menu.
- The notation **Ctrl+S** refers to holding down the control key while pressing the 'S' key. On Mac, you can interpret this as **Command+S**.
- Where we refer to right-clicking, if you're on a Mac, this action will be peformed by control-clicking since there's only one mouse button.
- The terms *compiling, building* and *generating* an ebook all refer to the same act of creating an ebook from the information in your project.
- A *context menu* is the menu you get when right-clicking (or control-clicking on a Mac), or

pressing the context menu button on a Windows keyboard.

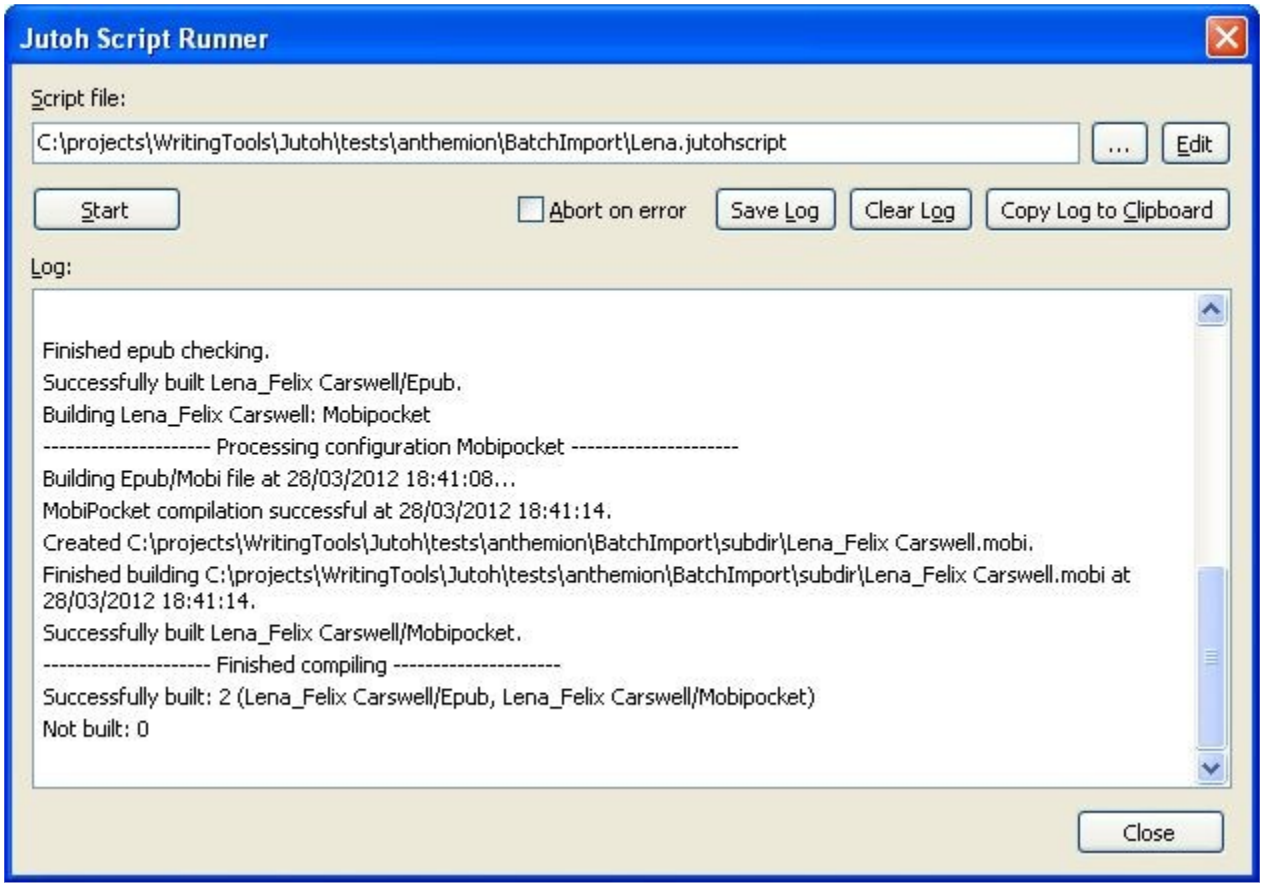
- *Document* is a general term for each separate item that can appear in a Jutoh project outline, whether it's a chapter of your book, an embedded font, an audio file, or any other supported document type.
- *Book section* refers to a specific kind of document in which you can edit text and graphics; it can contain a chapter, a title page, a table of contents or any individual part of a book. This may sometimes be abbreviated to *section*.
- A *dialog* is a window that opens in response to some command or condition; usually (but not always) it needs to be dismissed before you can continue working in the main window. Dialogs usually have a **Help** button that will give more detailed information than this guide can cover.
- Screenshots are taken on Windows, but the functionality is identical on Linux and Mac even if it looks slightly different.

# Chapter 1: Introduction to Jutoh Plus

Scripting and HTML templates are the two main features of Jutoh Plus. Jutoh Plus also supports the creation of CHM and HTB help files, and custom compile messages. In this chapter we explore the concepts behind these features, and how you might make use of them.

## *Scripting in Jutoh*

Scripting is the main feature of Jutoh Plus. A script is a file with the *jutohscript* extension, and you can run a script from the command line or by opening a script using **File | Open**. If you open a script from Jutoh, you will see the Script Runner window:




*The script runner window*

Click on **Start** to run the script. A progress gauge will be shown below the log window, and information, warnings and errors will be written to the log window. Errors will be shown in red. Click on **Edit** to edit the script. You can choose a new script file with the "**...**" button, and you can save, clear or copy the current log text. If you check **Abort on error**, the script will terminate immediately an error is encountered; otherwise it will carry on with processing as far as it can.

When running Jutoh from the command line using the `--batch` and `--quiet` switches and supplying a Jutoh script file, Jutoh will complete the script processing without any prompting or visible user interface, so you can use it as a step in other shell or DOS command scripts, or indeed from any other application. You can view or parse the log file to determine if the script was successful.

Here is an example of command line operation:

```
jutoh --append=log.txt --batch --quiet MyScript.jutohscript
```

Please note that if running a script by clicking it within Finder on a Mac, Jutoh may hang during script execution, so it's best to run it by opening the script from **File | Open** in Jutoh.

## *Editing scripts*

Jutoh doesn't provide an editor for scripts, since they are simple text files (preferably encoded in UTF-8, without a BOM marker) that you can use with almost any text editor. For example, on Windows, you can use Windows Notepad, or a third-party application such as Programmer's Notepad 2. If you are using the script runner window, you can click on **Edit** to edit the current script using whatever application is associated with the *txt* extension. On Windows, that's usually Windows Notepad unless you install a different editor.

## *Running scripts using custom tools*

The **Tools** menu, on the menubar at the top of the application window, lists built-in tools (such as a 'Dictionary' web link) and also any custom ones that you add via the Preferences/Tools pane. The tools can also be shown on the Jutoh Desktop. Normally custom tools are documents, web links or executable files, but you can also specify Jutoh script files. When a script is executed from a tool, there is no script window, and messages, warnings and errors appear on the Jutoh log window.

You may find custom script tools useful for accomplishing certain tasks that would be laborious to do via the regular user interface, even though you use Jutoh interactively most of the time. For example, you can use it to perform reimport and subsequent property setting and compilation for the current project.

## *What can scripting be used for?*

A Jutoh script can create a Jutoh project file from input such as a DOCX or ODT file; it can specify most of the options that you can change by hand, such as how to split the file into sections, and it can check and create ebook files. You can import styles common to your projects, replace strings, and insert content fragments (such as a copyright notice that is the same in all your projects). You can change the way different configurations behave, for example specifying different cover designs for different formats.

These capabilities are especially useful when you are dealing with many ebooks. One scenario is coordinating incoming content that is formatted in DOCX or ODT to your specifications. You can outsource the editing to people who just use a regular word processor and who don't have to know how Jutoh works. You can then use scripts to convert the files you receive into Jutoh projects and various ebook formats, adding standard copyright notices, extra content, and covers; all without a lot of manual operations.

Another use would be to stamp a book with the customer's name to discourage redistribution or to make it more personal. This could be done on a web server, if necessary using a virtual X server such

as Xvfb since Jutoh still needs a notional display to be present: see Chapter 3 for more on this.

In summary, these are the main operations you can perform in a Jutoh script:

- import from DOCX, ODT, HTML, Epub or text;
- run a reimport and compile, for the current project, without going through all the New Project Wizard steps;
- open an existing Jutoh project for modification;
- specify how the input will be split into sections;
- generate ebooks;
- check the Epub after generation;
- create multiple configurations, including more than one for the same basic format;
- change section titles;
- specify a cover or a different cover per configuration;
- insert DOCX and ODT fragments into sections, putting the fragment at the start or end of a section or using a keyword to position it;
- insert images in specific sections or using keywords;
- replace text or paragraph styles based on a search for text or paragraph style;
- merge an external style sheet with the default style sheet;
- set string table values, for one configuration per command or all configurations at once.

Note that you can't influence the ordering of different types of commands; they are done in this order:

1. file import;
2. file insertions;
3. document, configuration, string table and image property settings;
4. text and style replacements;
5. book compilation and checking.

## *HTML templates*

Normally, Jutoh generates the HTML code for each book section. You can already customize HTML a little by marking content with the "HTML" paragraph style, and adding custom CSS though Jutoh style sheets and per document. HTML templates allow you to go a step further and replace parts or all of each HTML file with custom code edited within Jutoh.

Jutoh introduces the concept of 'assets', which are simply fragments of text to be inserted into each document file according to block markers in a special asset, the template. An asset could be JavaScript code, or custom CSS, or HTML content to insert into the body of the file. Each document can have an arbitrary number of named assets, including the template itself, and by inserting block markers in the template, you can instruct Jutoh to insert other assets into the template. The project itself also has a set

of assets that can be used to create each HTML file, which means you can make document-specific assets override the project assets. Or you can use only project assets.

Jutoh's 'asset editor' is used to add, delete and edit templates and other fragments for documents and for projects. To edit a document's assets, you can use **Format | Asset Editor** to switch from the normal editor view to the asset view, and back. The configurable syntax highlighting editor is a convenient way to edit code without recourse to an external editor.

In addition to assets, further code can be supplied using *source code documents* that you can add to your project and edit using the built-in code editor. They will be placed under the *Resources* folder in the project outline, and will be included in the ebook for you to refer to from document or project assets. For example, you could include JavaScript files that will be referenced from script tags in HTML documents. For a given source code document, you can specify which configurations the code will apply to, since you may wish to have the ability to generate different variants of your ebook with varying levels of complexity.

Currently, assets cannot be specified in scripts, but you can specify a Jutoh project template which may have customised project assets.

See Chapter 4: Using HTML Templates for more details.

### *Creating CHM and HTB HTML help files*

Jutoh Plus can create HTML Help files used by Windows and wxWidgets applications. Windows HTML Help files have the extension CHM, and wxWidgets HTML Help files have the extension HTB. HTB files contain simplified HTML with no CSS, but the two formats otherwise have similar project, contents and keywords files. For more on this, please see Chapter 5: Creating HTML Help.

### *Custom compile messages*

Jutoh Plus can be configured to emit additional messages during compilation, tailored to the user's needs. Normally *find and replace presets* are used within the Find and Replace dialog, for quick access to frequently-used find commands. But they can also be used to find issues in a project whenever you compile it; examples include:

- highlighting any paragraphs formatted with the 'Normal' paragraph style rather than a more specific style;
- highlighting any blank paragraphs;
- checking for words that are frequently accidentally repeated, such as "his his", "to to", "at at", and so on;
- checking for known typos of character names;
- checking for clichés or over-used words that would not show up in a spell-check;
- checking for references to book distributors that are not supposed to be mentioned in the final ebook.

This facility can use preset libraries stored in the project itself, or in the global store, or a combination of both. For more information, please see the topic *Working with custom compile messages* in the Jutoh application help.

## *Summary*

We've seen how to edit and run a script, and what scripts can be used for. We've also introduced the concepts of HTML templates, assets and source code documents, and touched briefly on the other features of Jutoh Plus. Next we'll delve into the details of script syntax.

# Chapter 2: Script Syntax

In this chapter we'll describe the anatomy of a Jutoh script in detail.

## *General script syntax*

A Jutoh script has extension jutohscript and is composed of groups of variable/value pairs, in this form:

```
[Group Name]
Variable1=Value2
Variable2=Value2
```

The group names are arbitrary and not used by Jutoh except to report errors, but they must be unique. You can use multiple groups to process multiple files.

Jutoh scripting is not a procedural programming language and everything is specified using variable/value pairs. The value part may have a special syntax depending on the type of variable, for example comprising semicolon-separated named parameters – see the *insertfile* and *search* examples below. Variables can refer to project-wide or configuration-wide settings. Here's a configuration-specific setting:

```
configs.Epub.HTML left margin=10
```

And a project-wide setting:

```
title=Lena
```

You can add a comment on its own line using a semicolon.

By default, Jutoh assumes that the script is encoded in UTF-8. If you want to change this, you can set the encoding in a comment at the top of the file, for example:

```
;;; encoding=windows-1252
```

Note that all backslashes, for example in file paths, must be escaped with an extra backslash. Alternatively, you can use forward slashes in paths on all operating systems.

If you wish to include a double quotation mark within a string, you can escape it with a single preceding backslash within a simple right-hand value (a string with or without quotation marks). However if the value is within a parameter, you must use two backslashes to escape the quotation mark. For example:

```
documents.Chapter 1.name="Chapter \"One\""
```

and:

```
insertfile=file:"none";new-section:"Chapter \\"One\\""
```

If you need to include an "=" character in the variable part of a command, you can escape it with a

backslash to stop it being interpreted as the separator between variable and value.

## *Actions*

You must specify the 'actions', a comma-separated list of keywords. For example:

```
actions=import,setproperties,compile,check
```

**open**
Opens the specified project.

**import**
Creates the specified project from the named source file, deleting the project first if it already exists.

**reimport**
Creates the specified project from the named source file, or if it exists, reimports into the existing project. Use `project=this` to specify the current project, and leave *source* blank to use the import file already existing in the project.

**openorimport**
Opens the project if it already exists, and creates it if it doesn't exist.

**compile**
Compiles the project after opening or creation.

**check**
Checks the ebook (Epub only).

**setproperties**
Sets the configuration and other properties if the main action is *open*, *import* or *openorimport*.

**close**
Closes the project after other actions have been completed.

**compact**
Compacts the project after other actions have been completed and before closing.

## *Configurations*

You must include the names of the configurations you're interested in, using the *configs* variable, which should be of the form:

```
configs=<Configuration Name 1>:<format 1>,<Configuration Name 2>:<format
2>, ...
```

Formats are represented by the normal extension: mobi, epub, odt, html, txt, htb, and mp3.

For example:

```
configs=Epub:epub,Mobipocket:mobi
```

## *Important variables*

**source**

For example: `source=Lena.odt`. This indicates the file to import, and can be ODT, DOCX, text, HTML or Epub. You can leave this blank if you are reimporting and the project contains an import file name (that is, you imported a file at least once before).

**project**
For example: `project=%TITLE%_%AUTHOR%`. This indicates the name of the project and can contain metadata keywords. You can also specify the value 'this', to indicate the currently loaded and selected project.

**template**
The template file to base the new project on. For example: `template=MyTemplate.jutoh`. For more information on templates, see below.

**deleteexistingproject**
Specify *yes* to delete an existing project if it exists, to avoid a prompt.

**abortonerror**
Specify *no* to continue processing other configurations if an error occurs.

**titlepage**
Specify *simple* to generate a title page from the metadata, or *none*. Valid on import only, and sets the value of the *Generate title page* property for all configurations. You can also set the property with the syntax `configs.*.Generate title page=yes`.

## *Metadata*

You can specify these properties for metadata, for either import or open actions:

*author, title, copyright, date, description, ISBN, EAN, language, contributors, subject, publisher, website, metadata_source, metadata_identifier*

(the case is significant). These can either be in the form:

```
ISBN=12345678
```

or:

```
ISBN:Epub=12345678
```

to specify either a configuration-independent or configuration-dependent value. If configuration-independent, the value will be inserted directly into the metadata. Otherwise, a keyword will be set in the metadata, and a configuration-specific string table value used which is substituted for the keyword when the ebook is generated.

*metadata_identifier* is an alternative to using ISBN or EAN and cannot be used in a configuration-specific form. The syntax of this property is:

```
metadata_identifier=value:"1234567";scheme:"URL"
```

## *Cover image*

You can specify the cover image or .dtempl cover design template file, on import only.

**cover**

Specify the cover image, with or without the configuration, for example `cover=Lena.jpg` or `cover:Epub=Lena.jpg` and `cover:Mobipocket=Trees.dtempl`

## *Splitting the file*

You can specify the method, pattern and style used for splitting the file into sections.

**splitmethod**
Can be one of *pattern, style, pagebreak*.

**splitpattern**
For example, *Chapter**. The value can include wildcards and pipe ('|') separators for multiple patterns.

**splitstyle**
For example, *Heading 1*. The value can contain wildcards.

## *Import variables*

These variables control aspects of import for ***HTML/Epub import only***.

**import.parsecss**
*yes* to parse CSS when importing HTML or Epub, *no* to use simple styles.

**import.firstparagraphstyle**
The style for first paragraphs after a title if *import.parsecss* is *no*.

**import.subsequentparagraphstyle**
The style for subsequent paragraphs if *import.parsecss* is *no*.

## *Table of contents*

These variables determine how the table of contents will be created. *removefromtoc* is one of the few variables that does not have a direct equivalent in the GUI. The equivalent is a sequence of manual actions to remove items from the compiled table of contents.

**toc**
Can be *simple*, *fromheadings* or *none*.

**tocmaxlevel**
A maximum level to search, for example 3.

**removefromtoc**
A pipe ('|') separated list of section names to ignore.

**toctitle**
The table of contents title, default *Table of Contents*.

**toctitlestyle**
The style for the ToC title, default *TOC Heading*.

**tocmatchstyleN**
The style match at level N (between 1 and 6).

**tocstyleN**

The table of contents entry style to apply at level N (between 1 and 6).

## *Footnotes*

These variables determine how footnotes will be formatted.

**footnotes.autobuild**
If using endnotes, this automatically rebuilds the endnotes when compiling if the value is *yes*.

**footnotes.endnotes**
If *yes*, an endnotes section will be created.

**footnotes.alwaysapplyfootnotestyle**
If *yes*, the designated footnote style will always be applied.

**footnotes.epubtypemarkup**
If *yes*, Epub 3 types will be applied to the endnotes page.

**footnotes.asidemarkup**
If *yes*, Epub 3 asides will be used for endnotes.

**footnotes.title**
The title for the endnotes section.

**footnotes.titlestyle**
The paragraph style for the endnotes title.

**footnotes.notestyle**
The paragraph style for each endnote.

**footnotes.citationstyle**
The character style for each footnote/endnote citation.

**footnotes.numberingstyle**
Either *global* or *chapter*.

**footnotes.citationappearance**
One of *superscript*, *squarebrackets*, *none*, or a custom string containing the keyword *%REF%*.

**footnotes.referenceappearance**
One of *superscript*, *squarebrackets*, *none*, or a custom string containing the keyword *%REF%*.

## *Index, bibliography, and fields*

These variables control settings for the alphabetical index, bibliography, and fields (including automatic heading numbering).

**index.autobuild**
If *yes,* builds an alphabetical index section when compiling.

**bibliography.autobuild**
If *yes,* builds a bibliography section when compiling.

**fields.autobuild**
If *yes,* updates fields and numbered headings when compiling.

**fields.autonumbering**

If *yes,* enables automatic heading numbering based on outline settings.

## *Setting properties*

You can set arbitrary image, configuration and string table values using this syntax:

```
<context>.<object name>.<property name>=<property value>
```

or

```
<configuration or table name>.<property name>=<property value>
```

where `<context>` is one of *configs*, *strings, globalstrings* and *objects*. Currently *objects* is used to specify image properties, but may be extended to other content objects in the future.

For example:

```
configs.*.Maximum image width=999
configs.Epub.HTML left margin=10
objects.butterfly.maxwidth=100%
strings.*.TESTSTRING=Hello, this is a test string!
```

or just:

```
Epub.HTML left margin=10
```

You can use an asterisk for the configuration or table name, to denote all configurations or tables.

Use *strings* to specify project string tables, and *globalstrings* to specify global string tables. If a table is referenced but not found, it will be created.

You can set document properties with the syntax:

```
documents.<document title>.<property name>=<property value>
```

For example:

```
documents.Chapter 1.guide=start
```

Valid document properties are *name* (the title of the document)*, tags, guide, filename, showintoc, showinnavmap, showinspine, scripted, uses-svg, uses-mathml, uses-remote-resources, javascript, javascript-filename, javascript-at-bottom, css, css-filename, notes.*

Valid image (object) properties are *leftpadding, rightpadding, toppadding, bottompadding, alt, width, height, maxwidth, maxheight, preserveoriginalformat, svg, alignment, name, id, url, pagelink*. Values of these properties are as specified in the image properties dialog, apart from the *pagelink* property who syntax is:

```
objects.butterfly.pagelink=Section Title#bookmarkname
```

In image assignments such as the above, the image id can be used instead of the image name to identify individual instances of an image that might share the same name.

Unlike other types of property, object properties can also use parameter syntax. Using this, we can speed up object property setting by specifying a section document as follows:

```
objects.butterfly.width=document:Section Title;value:600
```

Note that all property setting specified in a script will be done in one lump, after other operations have taken place; you cannot interleave file import and property setting.

## *Searching and replacing*

Specify a search operation with *search*. Search operations take parameters in the value part, of the form:

```
name1:"Value1";name2:"Value2";...
```

Valid search operations are: *find-text*, *match-text* (as *find-text* but with wildcards), and *find-paragraph-style*. Specify the title of a document (with optional wildcards) as the value of *in-section* to limit the replacement to one or more sections.

Valid replacement operations are *replace-text* and *replace-paragraph-style*.

*replace-text* cannot be used with *match-text* since pattern matching doesn't specify a unique match string. If you specify a value of $EMPTY$ for replace-text, the empty string will be used.

Examples:

```
search=match-text:"*\\* \\* \\**";replace-paragraph-style:"Centre"
search=find-text:"In my job";replace-text:"In my incredibly interesting
job";in-section:"Chapter One"
search=find-paragraph-style:"Normal";replace-paragraph-style:"Body Text"
```

The first example searches for two space-separated asterisks; they are escaped because normally an asterisk is a wildcard character matching zero or more characters. The escape character is a double-backslash because the backslash character is also a special character in script syntax, and has to be escaped itself.

## *Inserting files*

You can use the *insertfile* operation to insert files into existing sections or as new sections. DOCX, ODT, HTML, text, bitmap image and media files may currently be specified. If inserting text, the file should be in the encoding specified for the project, which defaults to UTF-8, or you can use the *encoding* parameter.

**Creating a new document section with new-section**

The value of the *new-section* parameter is the new section title. Also specify the document position with *after-section* or *before-section*, and/or *parent-section*, or omit these to append to the end of the book. Specify the file name with *file*. You cannot use **new-section** for image insertion.

*after-section* and *before-section* can take the values "first-section" and "last-section" as well as section titles.

You can specify the new document's guide type (such as "toc") with the *guide-type* parameter.

**Inserting content into an existing section with in-section**

For the *in-section* parameter, specify the name of the section to insert into (or '*' for all documents). Specify the new content file name with *file*. Also specify one of *at-start* ("yes"), *at-end* ("yes"), or a keyword to replace. The paragraph containing the keyword will be deleted. To insert before or after the paragraph containing certain text without deleting the existing text, specify *before-text* or *after-text*. This text is case-sensitive, matches anywhere in the paragraph and doesn't take wildcards.

You can specify a paragraph style to apply to all paragraphs in the newly-inserted content using the *style* parameter. For text files, you can specify the encoding to use with the *encoding* parameter; otherwise the project encoding will be used, defaulting to UTF-8.

**Inserting images into an existing section with in-section**

The parameters are as above, but also specify *name* to identify a new image, *id* to set the image identifier, and *class* to set the CSS class. You can also use *width, height, max-width, max-height, padding, left-padding, top-padding, right-padding, bottom-padding* (include units with dimensions), and *alt* (for alternate text). Specify an optional *alignment* with the value "none", "left" or "right".

**Inserting media objects into an existing section with in-section**

The parameters are as above, but you must specify *media-kind* as "video", "audio", or "image." The *file* parameter is the media file, which will be added to the project's Resources section if it has not already been added, and you can specify an image poster file with the *poster* parameter. Pass the value "yes" to *show-controls* if you wish media controls to be shown.

Examples:

```
insertfile=file:"TextChunk.odt";new-section:"My New Section";after-
section:"Lena"
insertfile=file:"TextChunk.odt";in-section:"Lena";at-end:"yes"
insertfile=file:"TextChunk.docx";new-section:"My New Section";parent-
section:"Lena"
insertfile=file:"TextChunk.odt";in-section:"Lena";at-start:"yes"
insertfile=file:"TextChunk.odt";in-section:"*";keyword:"%MYTEXT%"
insertfile=file:"butterfly.png";in-section:"*";keyword:"%MYIMAGE
%";name:"butterfly";id:"imageid1";class:"animals"
insertfile=media-kind:video;poster:poster.png;alignment:none;width:"90%";max-
width:"100%";padding:"1cm";right-padding:"1cm";alt:"Fox video";title:"A
fox";show-controls:yes;file:"fox.m4v";in-section:"Chapter 3";at-end:"yes"
insertfile=file:"TextChunk.odt";in-section:"*";before-text:"In my"
```

## *Inserting resource documents*

The *insertresource* command lets you insert arbitrary resources into the project. Use the parameters *file*, *mimetype* and *resource-location*. For example:

```
insertresource=file:"MyAudio.mp4";mimetype:"audio/mp4";location:"resources/MyA
udio.mp4"
```

## *Style sheets*

You can specify a Jutoh style sheet file to import or merge on project creation. If you specify *importstylesheet*, the style sheets will be added to the existing style sheets and any name clash will

result in the new sheets overriding the old sheets.

If you specify *mergestylesheet*, only the first sheet in the file will be used and it will be merged with the default style sheet, with any new styles overriding existing ones. You can use this to incorporate custom styles, which you might then use by doing a style search and replace operation.

Examples:

```
importstylesheet=MyStyles.stylesheet
mergestylesheet=MyStyles.stylesheet
```

Style sheets can be used when importing, or applied when opening an existing project file.

## *Templates*

You can specify a Jutoh project to use as a template using the *template* variable, which will be applied before import, or after opening an existing project. You can control whether parts of the template will be merged with imported data, discarded, or kept. These specifiers follow the pattern of `template:<element>=<value>` where `<value>` is one of *keep* (retain this template information and ignore imported/specified information), *discard* (ignore this template information) and *merge* (merge this template information with specified or imported information, with priority given to the latter). Currently, *merge* only works for metadata, and otherwise is simply a synonym for *keep*.

Specify a template file like this:

```
template=MyTemplate.jutoh
```

These specifiers are supported (the defaults are asterisked):

- **template:accessibility** – accessibility options, as set via Project Properties/Accessibility. One of keep*, discard
- **template:bibinfo** – the bibliography options, as set via Project Properties/Indexes/Bibliography. One of keep*, discard
- **template:bullets** – the ODT bullet information, as set via Project Properties/Options/ODT Options. One of keep*, discard
- **template:configurations** – the configurations. One of keep*, discard
- **template:content** – contents of the "Content" folder. One of keep*, discard
- **template:covers** – the cover design(s). One of keep*, discard
- **template:fixedlayout** – the fixed layout project options, as set via Project Properties/Fixed Layout. One of keep*, discard
- **template:footnotes** – the footnote options. One of keep*, discard
- **template:htmlhelp** – the HTML Help options, as set via Project Properties/HTML Help. One of keep*, discard
- **template:index** – the index options. One of keep*, discard
- **template:metadata** – the metadata such as title, author. One of keep, discard, merge*

- **template:linkstyles** – link styles as edited via Project Properties/Styles/Link Styles One of keep*, discard
- **template:options** – miscellaneous options such as *titlepage*; for internal Jutoh use only. One of keep*, discard
- **template:outlinestyles** – outline styles as edited via Project Properties/Fields & Numbering. One of keep*, discard
- **template:pagelayout** – page layout options as edited via Project Properties/Page Layout. One of keep*, discard
- **template:pagestyles** – page styles as edited via Project Properties/Page Layout/Manage Page Styles. One of keep*, discard
- **template:projectoptions** – various project options as edited via Project Properties/Options. One of keep*, discard
- **template:regions** – fixed layout magnification region settings, as edited via Project Properties/Fixed Layout. One of keep*, discard
- **template:resources** – contents of the "Resources" folder. One of keep*, discard
- **template:sequencedefinitions** – sequence definitions as edited via Project Properties/Fields & Numbering/Edit Sequence Definitions. One of keep*, discard
- **template:shortcuts** – the custom project shortcuts (associated with styles). One of keep*, discard
- **template:stringtables** – the string tables. One of keep*, discard
- **template:styles** – the style sheets. One of keep, discard
- **template:toc** – the table of contents options. One of keep*, discard

## *Summary*

You have learned how to create a script for creating and altering ebooks. For an example, please see Appendix A: A sample script. Next we will look at how to control scripting from the command line.

# Chapter 3: Using the Command Line

This chapter describes how to use the command line to drive Jutoh's scripting facility.

## *Command line syntax*

When running Jutoh from the command line, you can specify a Jutoh project file name (.jutoh) or a Jutoh script file name (.jutohscript).

You can specify switches in a long or a short form. The long form uses two dashes, and the short form uses a single dash or a forward slash. Where a parameter is specified, it is separated from the switch by equals ('=').

These are the switches you can use:

**--output=filename (-o=filename)**
Writes log information and errors to the specified file.

**--append=filename (-a=filename)**
Appends log information and errors to the specified file.

**--info (-i)**
Writes verbose informational output, only when writing to a log file.

**--batch (-b)**
Reduces GUI prompting to a minimum, and exits after processing. Some GUI will still be shown.

**--quiet (-q)**
Shows no GUI in batch mode.

**--help (-h)**
Shows help on command line switches.

**--version (-v)**
Shows the Jutoh version number.

**--local-settings (-l)**
Specifies that the application settings should be stored in a data file next to the Jutoh executable.

**--display=hostname:displaynumber.screennumber**
This is available on Linux and other X11-based operating systems only. The value is normally : 0.0. This overrides the value of the DISPLAY environment variable. You can use this argument to redirect display output, for example to a virtual display such as Xvfb.

When executing a Jutoh script, you are likely to specify **--batch**, and one of **--output** or **--append** to save the log to a file. The **--batch** switch will cause Jutoh to execute the script and return after processing. Specify **--quiet** to ensure that the processing is not interrupted by dialogs.

For example:

```
jutoh --append=log.txt --batch --quiet MyScript.jutohscript
```

### *Using a virtual display*

On X11-based systems, you can use Xvfb to run Jutoh without the need for a physical display. Use the xvfb-run command to simplify the use of Xvfb and Jutoh. For example:

```
xvfb-run jutoh --batch myscript.jutohscript
```

This runs the virtual display, then runs Jutoh using this display (by setting the DISPLAY environment variable). When Jutoh terminates, so does Xvfb.

### *Using the command line on Mac OS X*

On Mac OS X, you can use the 'open' command to run Jutoh with command line arguments, passing -W to wait until processing has terminated. For example:

```
open -W Jutoh.app --args myscript.jutohscript --batch --quiet
```

Please note that if running a script by clicking it within Finder, Jutoh may hang during script execution, so it's best to run it by opening the script from **File | Open** in Jutoh.

### *Using the command line on Windows*

In a Windows command window, you can use start /WAIT to return control to the script only when processing has terminated. For example:

```
start /WAIT jutoh.exe myscript.jutohscript --batch --quiet
```

### *Summary*

This chapter has covered the details of running Jutoh on the command line. Next, we look in detail at the use HTML templates for adding code and custom HTML.

# Chapter 4: Using HTML Templates

This chapter describes how to use Jutoh Plus templates to customise your HTML-based ebooks. You can use this to add further styling, JavaScript code, and entire custom HTML sections.

Note that if your needs are relatively simply, you may be able to get away with adding custom CSS and JavaScript code via the document properties dialog. In Jutoh 2, HTML identifiers and other HTML attributes are made available in a document, and a range of HTML tag objects can be inserted, reducing the need to replace a whole section with HTML.

## *Introduction to templates and assets*

By default, no templates are used to generate HTML – Jutoh will use a standard HTML file layout and code generated from style and content in each document. If you switch on project-wide templates via the **Assets** page in Project Properties, Jutoh will now use the default template called 'HTML Template' and insert generated code into special blocks marked with keywords within the template. These are the keyword names of the standard blocks that Jutoh will use:

**BODY**
Substitutes the content generated from the text and graphics in the current document.

**PROLOG**
Substitutes the first couple of lines from an XHTML file, containing the *?xml* tag and *! DOCTYPE* directive.

**HTML_OPEN**
Substitutes the opening *head* tag.

**HEAD**
Substitutes the contents of the HTML head portion.

By default, there is a single asset called 'HTML Template' that has the **TEMPLATE** keyword to indicate that it is a template rather than any other kind of asset. Other assets you can add are CSS, JavaScript, and HTML fragments. As far as the template system is concerned, they are just text portions that are placed in the relevant block. If you define new assets with new keywords, you will also want to add the relevant block to your template. So if you add an asset called 'Fancy JavaScript' with the keyword **FANCYJAVASCRIPT**, then the relevant portion of your template might look like this:

```
<head>
<script>
!{{{ FANCYJAVASCRIPT
!}}} FANCYJAVASCRIPT
</script>
!{{{ HEAD
!}}} HEAD
</head>
```

A quick way to add a block is to right-click over the editor and select **Insert Block**; if there any non-

template fragments defined for the current document, you will get a choice of blocks to add and Jutoh will insert the appropriate markers at the current position.

Note that although we've edited the template, we'll still get the usual head code generated by Jutoh since we left the **HEAD** block in. If you want to replace the standard code, delete the block and add your own code.

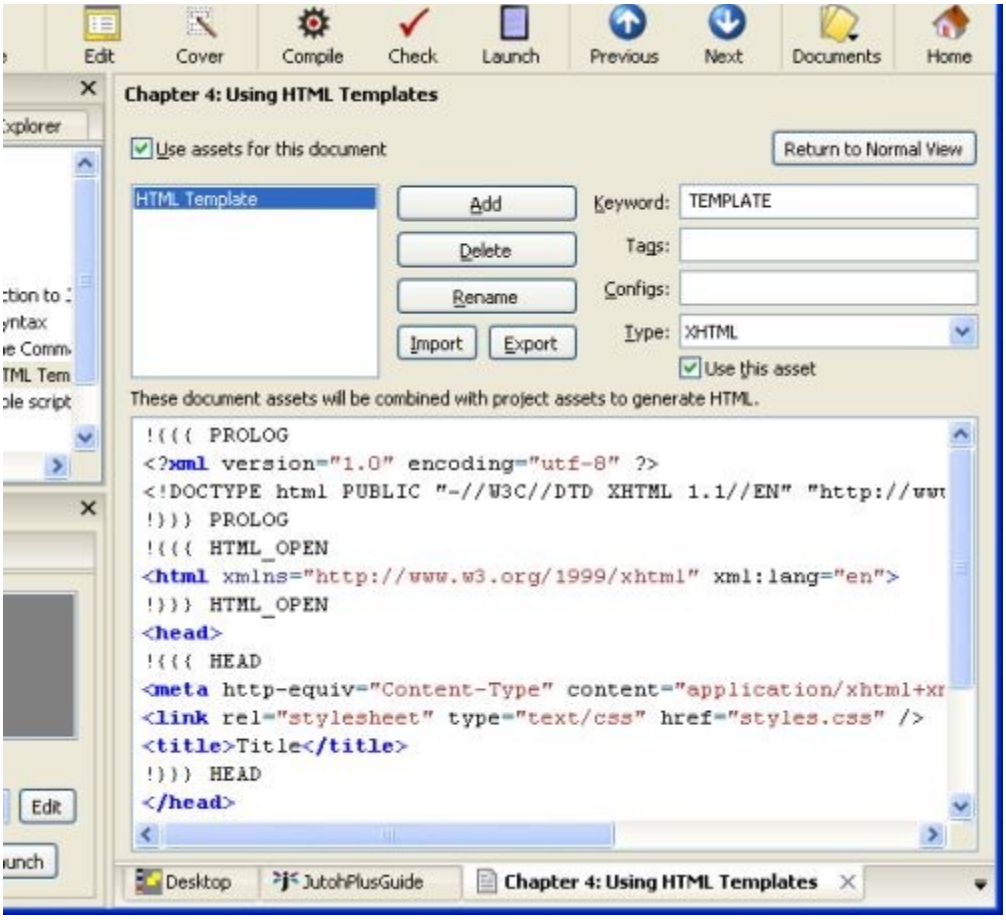
Block markers, *!{{{* to start the block and *!}}}* to end the block, are stripped out before the final HTML is written to the ebook file. Block markers must not have any space before them, and the block keyword must be preceded by just one space.

You can disable assets by clearing the *Use this asset* checkbox in the asset editor. A disabled asset will not take place in any substitutions.

Asset names are not used during code generation by Jutoh, but they must be unique within the project or document. You can have several assets with different names but the same keyword; we will see shortly how we can choose between assets.

## *Combining document and project assets*

It's convenient to specify a template for the whole project, but what if you want to customise each document, or maybe just one or two of them? Each document in your book has its own template and assets, initially disabled. You can edit them by toggling the asset editor for each document, with **Format | Asset Editor** (Ctrl+Alt+W). Use the command again to get back to the regular editor view. This is not shown as a tab to cut down on user interface clutter; instead, you can get back to the regular editor view by pressing the **Return to Normal View** button or via the **Asset Editor** command.




*The asset editor for a book section*

Now we potentially have clashing templates and assets, since we can have project and document assets active simultaneously. Jutoh resolves this by searching through enabled document assets first, then project assets. If it has used an asset from the document, it won't use the same-named asset from the project.

You can use this in various ways to differentiate one document from another. For example, say you want to add an animated logo using JavaScript to the top of several, but not all, documents. However, you don't want to copy and paste the whole template between documents – you want to use the same project template for all documents.

To do this, add a **LOGO** block to your project template just before the **BODY** block. Then, for a document that needs a logo, enable assets for the document, disable the document template, and add a **LOGO** asset containing the JavaScript. If the **LOGO** asset is found in a document, it will be inserted into the template, otherwise it will be omitted.

A variation of this is to keep the **LOGO** JavaScript asset in your project, only, and change the template of the individual document, adding the **LOGO** block markers.

## *Viewing and editing the substituted HTML*

When a document template is being used, Jutoh will do the substitutions and then save the result (before block marker removal) for later editing. This lets you see the whole HTML file, and gives you

the opportunity to ‘freeze’ parts of the HTML by removing the block markers, so Jutoh no longer replaces these parts.

If the document-specific template is not being used, perhaps because you disabled it or it didn’t match the given criteria so the project template was used instead, then Jutoh will not store the template after substitution of fragments. This is because the template is no longer document-specific, and therefore cannot be stored per-document. It’s worth bearing this in mind if you’re surprised to see no changes in a document template.

Disabled fragments or fragments that do not match the criteria for the current document and configuration will have the contents of their blocks removed. However, if you delete or change the keyword for a particular block so that the keyword is no longer known to Jutoh, you will need to manually delete that block in your templates.

Note that if you edit within block markers, these edits will be lost the next time Jutoh generates an HTML-based ebook (using Epub, Mobipocket, or HTML configurations).

### *Using tags and configuration names*

If you specify comma-delimited tags for an asset, they will be used to choose between alternative assets, depending on the tags that have been defined in the current document using the Document Properties dialog. If no tags are specified for the asset, this is considered to be a match (the asset will be used). If one or more of the tags match tags in the curent document, this is a match. If one or more tags are specified for the asset, but none match tags in the current document, then this is not a match and assets is not used.

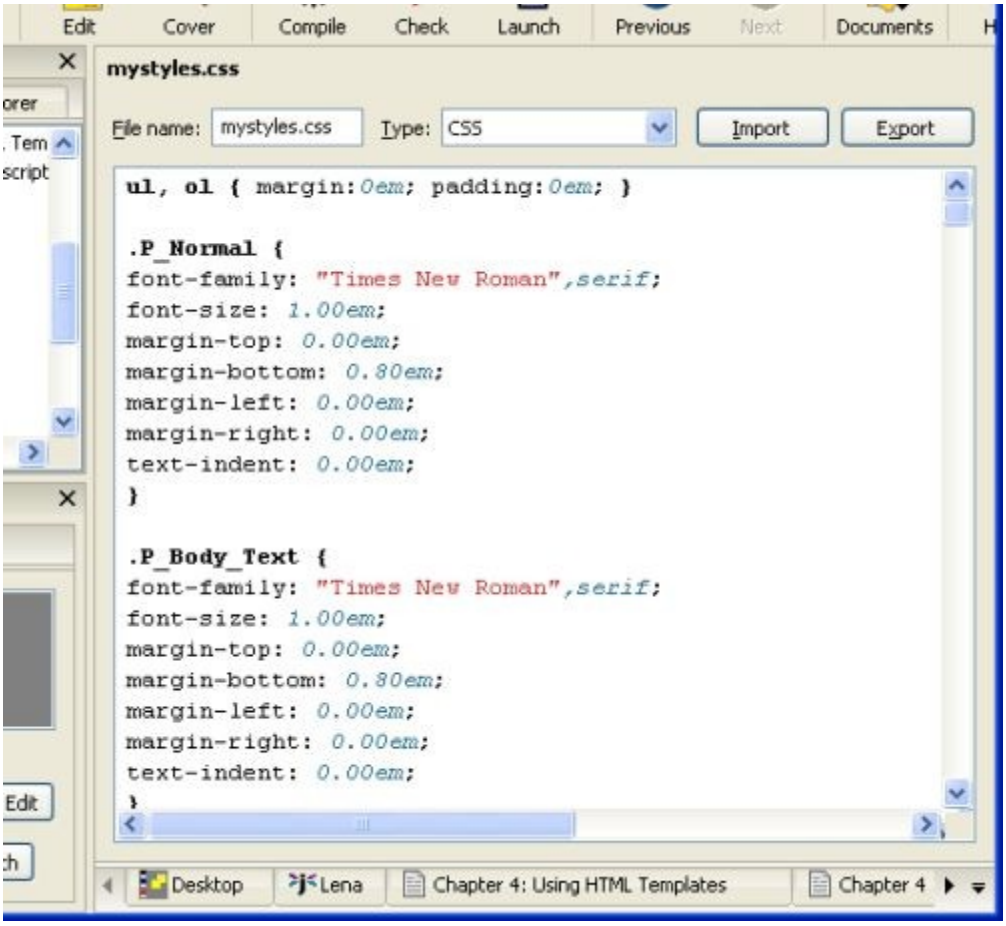
This provides an even simpler way to implement our logo example – have the logo fragment in project assets, and simply specify tags for the chapters to which it should apply. We don’t even need to edit any document assets; just edit document properties and provide tags.

Similarly, you can specify one or more configuration per asset (again comma-delimited), and this way you can have different templates or fragments depending on the current configuration. For example, you might have an ‘iBooks Epub’ configuration that uses JavaScript, and a ‘Generic Epub’ configuration that doesn’t.

### *Using source code documents*

In Jutoh Plus, you can add Source Code documents under the Resources folder in your project. These documents can contain HTML, XML, JavaScript, or CSS, and the code editor will reflect the language choice with appropriate colouring. The data you paste, type or import into these documents are stored within the project file as usual, but will be written to the ebook file under the OPS folder. You can specify relative paths if you want them to be stored in a separate folder under OPS.

Here’s an example of a source code document, showing CSS data:

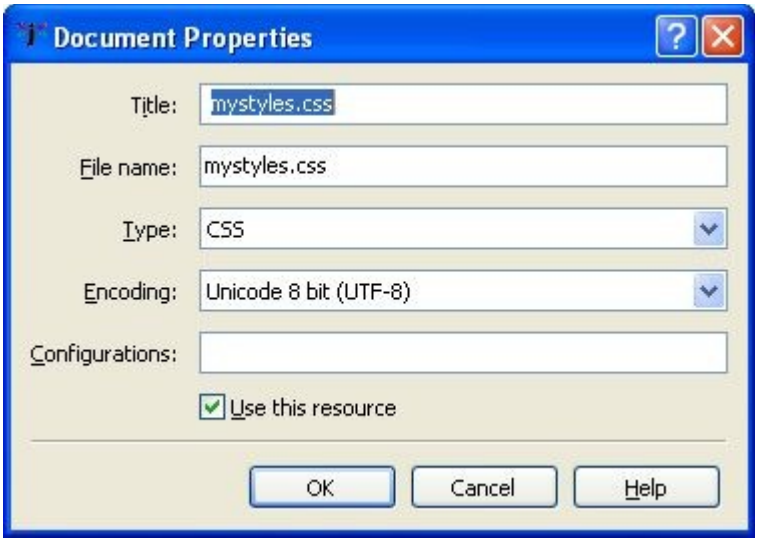



*A source code document*

Now you can refer to the file in your document or project assets, so you don't have to paste the contents of the file verbatim into your template or other fragments.

Keyboard and mouse editing operations are as you would expect for a source editor, and the **Find** and **Replace** commands on the **Edit** menu work in source code documents (but only one document at a time).

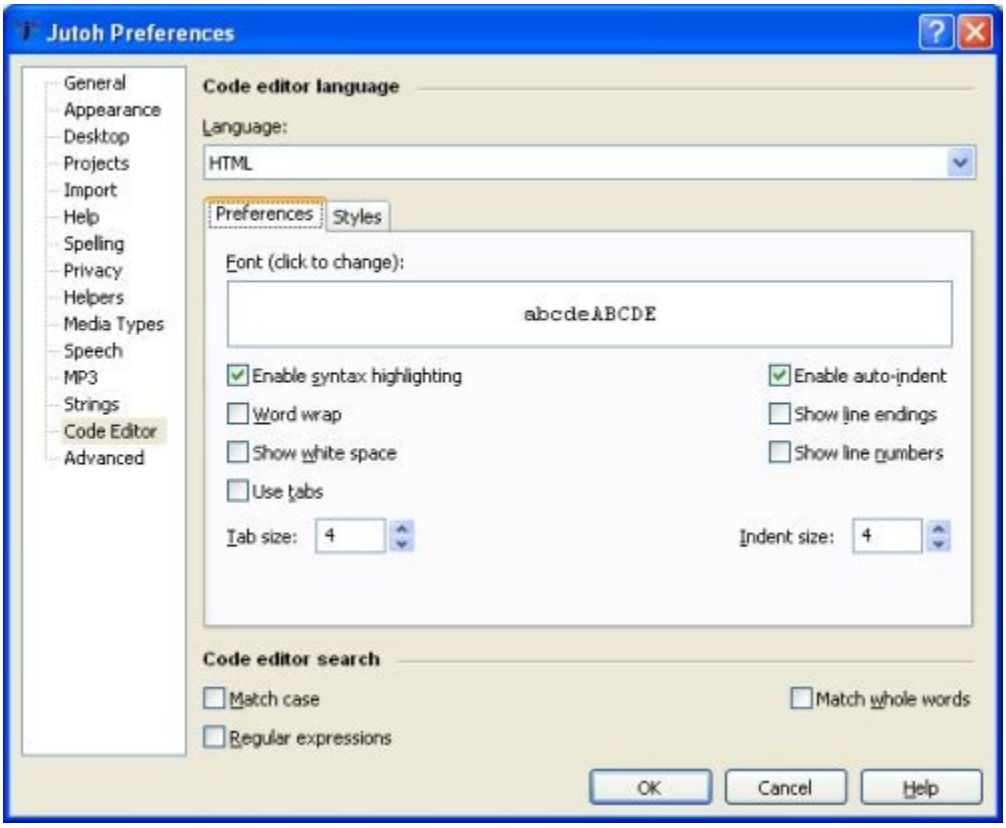
The properties dialog for source code documents lets you specify the encoding the file should be written with, whether the file should be included at all, and the configurations that the source code is relevant to (a comma-separated list of configuration names). If the configuration list is left blank, the file will be written for all HTML-based configurations, except if *Use this resource* is cleared.




*The source code document properties dialog*

## *Editing code editor preferences*

Various aspects of the code editor appearance and behaviour, such as syntax colouring and auto-indentation, can be changed per code language in the Code Editor page of the Preferences dialog.




*The code editor preferences*

Styles are defined for particular aspects of code syntax. When you click on the Styles tab in the code editor preferences, you will see all the styles relating to that language. However, note that some styles, such as ‘Comment’ and ‘Default’, are shared between languages.

In the Preferences tab, you can specify a global font for all languages (font family, style and size) but this can be overridden per style. To reset this overridden font so that the global font setting dominates again, click **Restore Style Defaults** on the Styles tab.

## *Summary*

This chapter has covered the details of using HTML assets and source code documents to control the way HTML document files are created.

Next, we look at how Jutoh Plus can be used to create application help.

# Chapter 5: Creating HTML Help

This chapter describes how to use Jutoh Plus to create HTML help. This is useful for application developers and technical authors who want to create integrated application help from Jutoh projects, as well as documentation in the other formats supported by Jutoh.

## *Introduction to HTML help*

Jutoh Plus can create HTML Help files used by Windows and wxWidgets applications. Windows HTML Help files have the extension CHM, and wxWidgets HTML Help files have the extension HTB. HTB files contain simplified HTML with no CSS, but the two formats otherwise have similar project, contents and keywords files. CHM files are mainly used on Windows, but there are CHM viewers for other platforms. HTB files are, like wxWidgets, multiplatform.

In a wxWidgets application, you can use a help controller class to show help topics at appropriate points, either using HTB/wxHTML Help for all platforms, or using CHM for Windows and wxHTML Help for other platforms. The built-in wxHTML Help viewer can be used within an integrated window in the application, or in a separate window. Jutoh uses both of these modes depending on whether a modal dialog is active or not.

Since the wxWidgets HTML renderer is quite simplistic, you can only put very basic HTML into an HTB file. You can't use CSS, and paragraph spacing is not adjustable, but you can use image and tables. The lack of CSS is mitigated by the fact that Jutoh will generate appropriate inline formatting within the restrictions of HTB. A full list of supported tags can be found in the wxWidgets documentation under the topic *wxHTML Overview*.

A more advanced version of wxHTML Help is currently being written that uses the wxWebView control and can therefore handle full HTML and CSS. So in future you will be able to set the *HTML version* option in your HTB configuration to 4 or 5 and Jutoh will generate modern HTML instead of the simplified HTML that the current system requires.

A CHM help file uses Microsoft Internet Explorer (up to version 7) to render content on Windows, so richer formatting is possible with CHM.

## *Anatomy of an HTML help file*

An HTB file is simply a zip file containing the content, whereas a CHM file must be created by a help compiler from an HHP project file and other files. The official compiler is included in Microsoft HTML Help Workshop, which is a free download, and is called `hhc.exe`.

These are the constituents of an HTML help file:

- A set of HTML files;
- a project file with extension HHP with options and a list of HTML and other files;
- a contents file with extension HHC describing a hierarchy of topic titles, topic identifiers and HTML files;

- an optional index file with extension HHK listing keywords with one or more links to HTML files, adding an **Index** tab to the help viewer;
- an alias file consisting of lines of the form IDENTIFIER=FILE, relating topic identifiers to the relevant HTML file;
- a map file consisting of lines of the form #define IDENTIFIER INTEGER which you can include in your application source files so that you can refer to topics by symbol.

For HTB only, HHC files are generated with an extra parameter for each topic object with the integer identifier, for example:

```
<param name="ID" value=101>
```

In CHM mode, this doesn't compile so is not included.

### *Importing from an existing HHP project*

You can create a new project based on an existing wxHTML Help or CHM project. A CHM project will first need to be decompiled to a folder using a suitable application, which you can find on the web. In the New Project Wizard, specify an HHP project file. Import will be performed after you finish the wizard.

If there is an HHK file in the HTML help project, it will be imported and you can edit it in Project Properties/HTML Help/Keywords.

Be aware that converting HTML to a Jutoh project is a somewhat imprecise process. It's likely that you'll need to do some editing to correct import issues. But it's still a lot faster than reworking your project from scratch.

### *Creating an index (HHK) file*

You have several choices when generating an HTB or CHM file and creating an HHK file:

1. use the keywords as edited via Project Properties/HTML Help/Keywords;
2. use document titles (one link per document, or if using an advanced table of contents, all the found headings);
3. use Jutoh index entries.

In fact you can use all three methods if you wish, or a combination, or none. You can choose which to use via Project Properties/HTML Help/Options.

### *Specifying topic identifiers*

If you specify a symbol in the "Id" property under Document Properties/Advanced Properties, this will be added to the map and alias files, along with an automatically generated integer that the topic identifier is associated with.

If you check the relevant option in Project Properties/HTML Help/Options, the map and alias files will be copied to the Jutoh project folder with the suffixes `_alias.h` and `_map.h`. In fact the alias file is not a valid C++ header file but you may wish to parse it to retrieve the association between topic identifiers and HTML files; normally this is not necessary since you can use the preprocessor symbols in the

_map.h file instead and the help system will know what HTML file to load for a given topic identifier.

Here's an example of loading and showing help. We assume that a section in the Jutoh project has its "Id" defined as ID_HELP_OVERVIEW.

```
#include "myproject_ids.h"
...
m_helpController->Initialize(helpFile);
...
m_helpController->DisplaySection(ID_HELP_OVERVIEW);
```

In the example, ID_HELP_OVERVIEW is defined as an integer in myproject_ids.h, and the help system looks up the topic integer in the HHC file to find the HTML file to show. Currently, you can't use identifiers within a document other than the top-level "Id" value since Jutoh doesn't gather identifiers from the content itself, only document properties.

Please refer to the wxWidgets documentation for further information on how to use the wxHtmlHelpController and wxCHMHelpController classes.

## *File encodings*

Generated HTB and CHM files use slightly different encoding schemes for their constituent parts.

The HHC and HHK file encoding is UTF-8 in HTB, and for CHM files is determined by the *CHM contents file encoding* selection in Project Properties/HTML Help. In HTB, the HHC and HHK files do not specify their encoding since it is specified in the HHP file using the *Charset* option and adding an encoding specification to the individual files can confuse wxWidgets HTML Help.

HTML files are always in UTF-8 in both formats.

Encodings for imported HHP projects can be more varied and Jutoh performs various checks to determine individual file encoding.

## *Summary*

In this final chapter we learned how Jutoh can be used for importing and generating application help.

We hope you enjoy a big productivity increase from using Jutoh Plus! Let us know what you think via the Jutoh support page, and feel free to suggest new features.

# Appendix A: A sample script

Here's a sample of a Jutoh script that creates a new Jutoh project from an ODT file, sets properties, adds different covers for Epub and Mobipocket, inserts an ODT fragment, and compiles Epub and Mobipocket ebook files. You can find a more complete example with the relevant script and input files on the Jutoh web site.

```
;;; encoding=utf-8
[Lena]
actions=import,setproperties,compile,check
configs=Epub:epub,Mobipocket:mobi
source=Lena.odt
title=Lena
project=subdir/%TITLE%_%AUTHOR%
author=Felix Carswell
author_ordered=Carswell, Felix
copyright=(c) Felix Carswell
date=2011
description=A girl meets a boy in a café.
cover:Epub=Lena.jpg
cover:Mobipocket=Trees.dtempl
ISBN:Mobipocket=1111111111111
ISBN:Epub=2222222222222

deleteexistingproject=yes
abortonerror=no

; pattern, style, pagebreak
splitmethod=pagebreak
splitpattern=Chapter*
splitstyle=Heading 1

; simple, fromheadings, none
toc=fromheadings
tocmaxlevel=3
removefromtoc=License Notes|Some other heading|And another

; yes to parse CSS, no to use simple styles
import.parsecss=no
import.firstparagraphstyle=Body Text
import.subsequentparagraphstyle=Body Text First Indent

; simple, none
titlepage=none

; Stylesheet to import on new project creation. Can be importstylesheet or
mergestylesheet
mergestylesheet=MyStyles.stylesheet

insertfile=file:"TextChunk.odt";new-section:"My Text Chunk";after-section:"Lena"
insertfile=file:"TextChunk.odt";in-section:"Lena";at-end:"yes"
```

```
insertfile=file:"TextChunk.odt";in-section:"Lena";at-start:"yes"
insertfile=file:"TextChunk.odt";in-section:"*";keyword:"%MYTEXT%"
insertfile=file:"TextChunk.odt";in-section:"*";before-text:"In my"
insertfile=file:"butterfly.png";in-section:"Chapter 2";keyword:"%BUTTERFLY
%";name:"butterfly";id:"image1"

objects.butterfly.url=value:"http://www.jutoh.com"
objects.image1.maxwidth=100%

configs.*.Maximum image width=999
configs.Epub.HTML left margin=10
strings.*.TESTSTRING=Hello, this is a test string!

; document property can be tags, guide, filename, showintoc, showinnavmap,
showinspine, notes
; This makes Chapter 1 the start section
documents.Chapter 1.guide=start

; The next three lines exclude Chapter 4 and Chapter 5 from the Mobipocket ebook
documents.Chapter 4.tags=notsample
documents.Chapter 5.tags=notsample
configs.Mobipocket.Exclude sections matching tags=notsample

; Experimental
; match-text can take wildcards, with * and ? escaped by double-backslash if
necessary.
; match-text can't be used to replace text, only styles.
search=match-text:"*\\* \\* \\**";replace-paragraph-style:"Centre"
search=find-text:"In my job";replace-text:"In my incredibly interesting job"
;search=find-paragraph-style:"Normal";replace-paragraph-style:"Body Text"
search=find-paragraph-style:"Heading 2";replace-paragraph-style:"Heading 2 With
Page Break"
```

# Change Log

**Version 1.10**, January 22nd 2018

- Mentioned custom compile messages in Chapter 1.

**Version 1.9**, December 8th 2017

- Added *bibliography.autobuild, index.autobuild* and *fields.autobuild* settings.

**Version 1.8**, March 30th 2014

- Added Chapter 5: Creating HTML Help.

**Version 1.7**, March 4th 2014

- Doumented escape characters.
- Corrected **setproperties** documentation.
- Added insertresource documentation and added the document properties *name*, *css*, *css-filename*, *javascript-filename*, *javascript-at-bottom*, *uses-svg*, *uses-mathml*, *uses-remote-resources*.

**Version 1.6**, July 24th 2013

- Added HTML and text to the list of files that can be inserted.
- Documented the further image and media object parameters.
- Expanded 'reimport' documentation and added a section on using custom script tools.
- Added documentation for 'compact'.
- Documented the new insertfile parameters *style, guide-type* and *encoding*.

**Version 1.5**, May 15th 2013

- Added EAN variable to metadata in addition to ISBN, to set the identifier attribute correctly.
- Replacement strings can be $EMPTY$ to indicate the empty string should be used.
- Search can specify *in-section* and a value with wildcards matching section titles, to limit search to one or more sections.

**Version 1.4**, November 27th 2012

- Added 'Using HTML Templates' and explained about HTML templates in the Introduction.
- Documented new *class* parameter when inserting an image file.

**Version 1.3**, October 3rd 2012

- Added 'Using HTML Templates' and explained about HTML templates in the Introduction.

- Removed references to adding unique numbers in script names since this is no longer necessary.

**Version 1.2**, August 18th 2012

- Added image and DOCX support for insertfile.
- Added *parent-section* and *name* parameters for insertfile.
- Added image property assignment using the syntax: `objects.objectname.paramname=value:paramvalue`.

**Version 1.1**, July 18th 2012

- Added template and template specifiers.

**Version 1.0**, April 6th 2012

- First edition.