

CALIBRE

User Manual



Table of Contents

[calibre User Manual](#)

[The Graphical User Interface](#)

[Actions](#)

[Add books](#)

[Edit metadata](#)

[Convert books](#)

[View](#)

[Send to device](#)

[Fetch news](#)

[Library](#)

[Device](#)

[Save to disk](#)

[Connect/share](#)

[Remove books](#)

[Preferences](#)

[Catalogs](#)

[Search & sort](#)

[The search interface](#)

[Saving searches](#)

[Virtual libraries](#)

[Guessing metadata from file names](#)

[Book details](#)

[Tag browser](#)

[Cover grid](#)

[Cover browser](#)

[Quickview](#)

[Jobs](#)

[Keyboard shortcuts](#)

[Adding your favorite news website](#)

[Completely automatic fetching](#)

[The calibre blog](#)

[bbc.co.uk](#)

[Customizing the fetch process](#)

[Using the print version of bbc.co.uk](#)

[Replacing article styles](#)

[Slicing and dicing](#)

[Real life example](#)

[Tips for developing new recipes](#)

[Further reading](#)

[API documentation](#)

[API documentation for recipes](#)

[The E-book viewer](#)

[Starting the E-book viewer](#)

[Navigating around an e-book](#)

[Bookmarks](#)

[Table of Contents](#)

[Navigating by location](#)

[Reference mode](#)

[Highlighting text](#)

[Read aloud](#)

[Following links using only the keyboard](#)

[Customizing the look and feel of your reading experience](#)

[Dictionary lookup](#)

[Copying text and images](#)

[Zooming in on images](#)

[Non re-flowable content](#)

[Designing your book to work well with the calibre viewer](#)

[E-book conversion](#)

[Introduction](#)

[Look & feel](#)

[Fonts](#)

[Text](#)

[Layout](#)

[Styling](#)

[Transform styles](#)

[Transform HTML](#)

[Page setup](#)

[Heuristic processing](#)

[Search & replace](#)

[Structure detection](#)

[Chapters and page breaks](#)

[Miscellaneous](#)

[Table of Contents](#)

[Using images as chapter titles when converting HTML input documents](#)

[Using tag attributes to supply the text for entries in the Table of Contents](#)

[How options are set/saved for conversion](#)

[Format specific tips](#)

[Convert Microsoft Word documents](#)

[Older .doc files](#)

[Convert TXT documents](#)

[Convert PDF documents](#)

[Comic book collections](#)

[EPUB advanced formatting demo](#)

[Convert ODT documents](#)

[Converting to PDF](#)

[Headers and Footers](#)

[Printable Table of Contents](#)

[Custom page margins for individual HTML files](#)

[Editing e-books](#)

[Basic workflow](#)

[The File browser](#)

[Renaming files](#)

[Merging files](#)

[Changing text file order](#)

[Marking the cover](#)

[Deleting files](#)

[Exporting files](#)

[Adding new images/fonts/etc. or creating new blank files](#)

[Replacing files](#)

[Linking stylesheets to HTML files efficiently](#)

[Search & replace](#)

[Saved searches](#)

[Function mode](#)

[Search ignoring HTML tags](#)

[Automated tools](#)

[Editing the Table of Contents](#)

[Checking the book](#)

[Adding a cover](#)

[Embedding referenced fonts](#)

[Subsetting embedded fonts](#)

[Smartening punctuation](#)

[Transforming CSS properties](#)

[Removing unused CSS rules](#)

[Fixing HTML](#)

[Beautifying files](#)

[Inserting an inline Table of Contents](#)

[Setting Semantics](#)

[Filtering style information](#)

[Upgrading the book's internals](#)

[Checkpoints](#)

[The Live preview panel](#)

[Splitting HTML files](#)

[The Live CSS panel](#)

[Miscellaneous tools](#)

[The Table of Contents view](#)

[Checking the spelling of words in the book](#)

[Adding new dictionaries](#)

[Inserting special characters](#)

[The code inspector view](#)

[Checking external links](#)

[Downloading external resources](#)

[Arranging files into folders by type](#)

[Importing files in other e-book formats as EPUB](#)

[The Reports tool](#)

[Special features in the code editor](#)

[Syntax highlighting](#)

[Context sensitive help](#)

[Auto-complete](#)

[Snippets](#)

[The calibre Content server](#)

[Accessing the Content server from other devices](#)

[Accessing the server from devices on your home network](#)

[Trouble-shooting the home network connection](#)

[Accessing the server from anywhere on the internet](#)

[The server interface](#)

[The book list](#)

[The book reader](#)

[Browser support](#)

[Enabling offline support](#)

[Managing user accounts from the command-line only](#)

[Integrating the calibre Content server into other servers](#)

[Using a full virtual host](#)

[Using a URL prefix](#)

[Creating a service for the calibre server on a modern Linux system](#)

[Comparing e-books](#)

[Understanding the comparison view](#)

[Launching the comparison tool](#)

[Comparing two e-book files](#)

[Comparing the ORIGINAL FMT to FMT](#)

[Comparing a checkpoint to the current state of the book while editing](#)

[Editing e-book metadata](#)

[Editing the metadata of one book at a time](#)

[Downloading metadata](#)

[Managing book formats](#)

[All about covers](#)

[Editing the metadata of many books at a time](#)

[Search and replace](#)

[Bulk downloading of metadata](#)

[Frequently Asked Questions](#)

[E-book format conversion](#)

[What formats does calibre support conversion to/from?](#)

[What are the best source formats to convert?](#)

[I converted a PDF file, but the result has various problems?](#)

[How do I convert my file containing non-English characters, or smart quotes?](#)

[What's the deal with Table of Contents in MOBI files?](#)

[The covers for my MOBI files have stopped showing up in Kindle for PC/Kindle for Android/iPad etc.](#)

[How do I convert a collection of HTML files in a specific order?](#)

[The EPUB I produced with calibre is not valid?](#)

[How do I use some of the advanced features of the conversion tools?](#)

[Device integration](#)

[What devices does calibre support?](#)

[How can I help get my device supported in calibre?](#)

[My device is not being detected by calibre?](#)

[My device is non-standard or unusual. What can I do to connect to it?](#)

[How do I use calibre with my iPad/iPhone/iPod touch?](#)

[Using the Content server](#)

[How do I use calibre with my Android phone/tablet or Kindle Fire HD?](#)

[Using a USB cable](#)

[Over the air](#)

[Can I access my calibre books using the web browser in my Kindle or other reading device?](#)

[I cannot send emails using calibre?](#)

[My device is getting mounted read-only in Linux, so calibre cannot connect to it?](#)

[Why does calibre not support collections on the Kindle or shelves on the Nook?](#)

[I am getting an error when I try to use calibre with my Kobo Touch/Glo/etc.?](#)

[Covers for books I send to my e-ink Kindle show up momentarily and then are replaced by a generic cover?](#)

[I transferred some books to my Kindle using calibre and they did not show up?](#)

Library management

[Where are the book files stored?](#)

[How does calibre manage author names and sorting?](#)

[Why doesn't calibre let me store books in my own folder structure?](#)

[Why doesn't calibre have a column for foo?](#)

[Can I have a column showing the formats or the ISBN?](#)

[How do I move my calibre data from one computer to another?](#)

[The list of books in calibre is blank!](#)

[I am getting errors with my calibre library on a networked drive/NAS?](#)

Miscellaneous

[I want calibre to download news from my favorite news website.](#)

[Why the name calibre?](#)

[Why does calibre show only some of my fonts on macOS?](#)

[calibre is not starting on Windows?](#)

[calibre freezes/crashes occasionally?](#)

[The calibre E-book viewer and Edit book tools do not work on Windows?](#)

[Using the viewer or doing any conversions results in a permission denied error on Windows](#)

[calibre is not starting/crashing on macOS?](#)

[I downloaded the installer, but it is not working?](#)

[My antivirus program claims calibre is a virus/trojan?](#)

[How do I backup calibre?](#)

[How do I use purchased EPUB books with calibre \(or what do I do with .acsm files\)?](#)

[I am getting a "Permission Denied" error?](#)

[Can I have the comment metadata show up on my reader?](#)

[How do I get calibre to use my HTTP proxy?](#)

[I want some feature added to calibre. What can I do?](#)

[Why doesn't calibre have an automatic update?](#)

[How is calibre licensed?](#)

[How do I run calibre from my USB stick?](#)

[How do I run parts of calibre like news download and the Content server on my own Linux server?](#)

[Tutorials](#)

[Adding your favorite news website](#)

[Completely automatic fetching](#)

[The calibre blog](#)

[bbc.co.uk](#)

[Customizing the fetch process](#)

[Using the print version of bbc.co.uk](#)

[Replacing article styles](#)

[Slicing and dicing](#)

[Real life example](#)

[Tips for developing new recipes](#)

[Further reading](#)

[API documentation](#)

[API documentation for recipes](#)

[Managing subgroups of books, for example "genre"](#)

[Setup](#)

[Searching](#)

[Restrictions](#)

[Useful template functions](#)

[XPath tutorial](#)

[Selecting by tag name](#)

[Selecting by attributes](#)

[Selecting by tag content](#)

[Sample e-book](#)

[XPath built-in functions](#)

[The calibre template language](#)

[Basic templates](#)

[Advanced formatting](#)

[Using templates to define custom columns](#)

[Using functions in templates - Single Function Mode](#)

[General Program Mode](#)

[More complex programs in template expressions - Template Program Mode](#)

[Stored general program mode templates](#)

[Providing additional information to templates](#)

[Notes on the difference between modes](#)

[User-defined Python template functions](#)

[Special notes for save/send templates](#)

[Templates and plugboards](#)

[Tips](#)

[Function reference](#)

[Reference for all built-in template language functions](#)

[All about using regular expressions in calibre](#)

[First, a word of warning and a word of courage](#)

[Where in calibre can you use regular expressions?](#)

[What on earth is a regular expression?](#)

[Care to explain?](#)

[That doesn't sound too bad. What's next?](#)

[Hey, neat! This is starting to make sense!](#)

[Well, these special characters are very neat and all, but what if I wanted to match a dot or a question mark?](#)

[So, what are the most useful sets?](#)

[But if I had a few varying strings I wanted to match, things get complicated?](#)

[You missed...](#)

[In the beginning, you said there was a way to make a regular expression case insensitive?](#)

[I think I'm beginning to understand these regular expressions now... how do I use them in calibre?](#)

[Conversions](#)

[Adding books](#)

[Bulk editing metadata](#)

[Quick reference](#)

[Quick reference for regexp syntax](#)

[Credits](#)

[Writing your own plugins to extend calibre's functionality](#)

[Anatomy of a calibre plugin](#)

[A User Interface plugin](#)

[__init__.py](#)

[ui.py](#)

[main.py](#)

[Getting resources from the plugin ZIP file](#)

[Enabling user configuration of your plugin](#)

[Edit book plugins](#)

[main.py](#)

[Running User Interface plugins in a separate process](#)

[Adding translations to your plugin](#)

[The plugin API](#)

[Debugging plugins](#)

[More plugin examples](#)

[Sharing your plugins with others](#)

[Typesetting mathematics in e-books](#)

[A simple HTML file with mathematics](#)

[More information](#)

[Creating AZW3 • EPUB • MOBI catalogs](#)

[Selecting books to catalog](#)

[Included sections](#)

[Prefixes](#)

[Excluded books](#)

[Excluded genres](#)

[Other options](#)

[Custom catalog covers](#)

[Additional help resources](#)

[Virtual libraries](#)

[Creating Virtual libraries](#)

[Examples of useful Virtual libraries](#)

[Working with Virtual libraries](#)

[Using Virtual libraries in searches](#)

[Using additional restrictions](#)

[The calibre:// URL scheme](#)

[Switch to a specific library](#)

[Show a specific book in calibre](#)

[Open a specific book in the E-book viewer at a specific position](#)

[Searching for books](#)

[Hex encoding of URL parameters](#)

[Customizing calibre](#)

[Environment variables](#)

[Tweaks](#)

[Overriding icons, templates, et cetera](#)

[Creating your own icon theme for calibre](#)

[Customizing calibre with plugins](#)

[Command Line Interface](#)

[Documented commands](#)

[calibre](#)

[\[options\]](#)

[calibre-customize](#)

[\[options\]](#)

[calibre-debug](#)

[\[options\]](#)

[calibre-server](#)

[\[options\]](#)

[calibre-smtp](#)

[\[options\]](#)

[calibredb](#)

[Global Options](#)

[list](#)

[add](#)

[remove](#)

[add format](#)

[remove format](#)

[show metadata](#)

[set metadata](#)

[export](#)

[catalog](#)

[saved searches](#)

[add custom column](#)

[custom columns](#)

[remove custom column](#)

[set custom](#)

[restore database](#)

[check library](#)

[list categories](#)

[backup metadata](#)

[clone](#)

[embed metadata](#)

[search](#)

[ebook-convert](#)

[Look and Feel](#)

[Heuristic Processing](#)

[Search and Replace](#)

[Structure Detection](#)

[Table of Contents](#)

[Metadata](#)

[Debug](#)

[AZW4 Input Options](#)

[CHM Input Options](#)

[Comic Input Options](#)

[DJVU Input Options](#)

[DOCX Input Options](#)

[EPUB Input Options](#)

[FB2 Input Options](#)

[HTLZ Input Options](#)

[HTML Input Options](#)

[LIT Input Options](#)

[LRF Input Options](#)

[MOBI Input Options](#)

[ODT Input Options](#)

[PDB Input Options](#)

[PDF Input Options](#)

[PML Input Options](#)

[RB Input Options](#)

[RTF Input Options](#)

[Recipe Input Options](#)

[SNB Input Options](#)

[TCR Input Options](#)

[TXT Input Options](#)

[AZW3 Output Options](#)

[DOCX Output Options](#)

[EPUB Output Options](#)

[FB2 Output Options](#)

[HTML Output Options](#)

[HTMLZ Output Options](#)

[LIT Output Options](#)

[LRF Output Options](#)

[MOBI Output Options](#)

[OEB Output Options](#)

[PDB Output Options](#)

[PDF Output Options](#)

[PML Output Options](#)

[RB Output Options](#)

[RTF Output Options](#)

[SNB Output Options](#)

[TCR Output Options](#)

[TXT Output Options](#)

[TXTZ Output Options](#)

[ebook-edit](#)

[\[options\]](#)

[ebook-meta](#)

[\[options\]](#)

[ebook-polish](#)

[\[options\]](#)

[ebook-viewer](#)

[\[options\]](#)

[fetch-ebook-metadata](#)

[\[options\]](#)

[lrf2lrs](#)

[\[options\]](#)

[lrfviewer](#)

[\[options\]](#)

[lrs2lrf](#)

[\[options\]](#)

[web2disk](#)

[\[options\]](#)

[Undocumented commands](#)

[Setting up a calibre development environment](#)

[Design philosophy](#)

[Code layout](#)

[Getting the code](#)

[Submitting your changes to be included](#)

[Windows development environment](#)

[macOS development environment](#)

[Linux development environment](#)

[Having separate “normal” and “development” calibre installs on the same computer](#)

[Debugging tips](#)

[Using print statements](#)

[Using an interactive Python interpreter](#)

[Using the Python debugger as a remote debugger](#)

[Using the debugger in your favorite Python IDE](#)

[Executing arbitrary scripts in the calibre Python environment](#)

[Using calibre in your projects](#)

[Binary install of calibre](#)

[Source install on Linux](#)

[API documentation for various parts of calibre](#)

[API documentation for recipes](#)

[API documentation for plugins](#)

[Plugin](#)

[FileTypePlugin](#)

[Metadata plugins](#)

[Catalog plugins](#)

[Metadata download plugins](#)

[Conversion plugins](#)

[Device drivers](#)

[User interface actions](#)

[Preferences plugins](#)

[API documentation for the database interface](#)

[API documentation for the e-book editing tools](#)

[The Container object](#)

[Managing component files in a container](#)

[Pretty printing and auto fixing parse errors](#)

[Managing book jackets](#)

[Splitting and merging of files](#)

[Managing covers](#)

[Working with CSS](#)

[Working with the Table of Contents](#)

[Edit book tool](#)

[Controlling the editor's user interface](#)

[Digital Rights Management \(DRM\)](#)

[What does DRM imply for me personally?](#)

[What does DRM do for authors?](#)

[DRM and freedom](#)

[Why does calibre not support DRM?](#)

[What is calibre's view on content providers?](#)

[How can I help fight DRM?](#)

[Glossary](#)

calibre User Manual

calibre is an e-book library manager. It can view, convert and catalog e-books in most of the major e-book formats. It can also talk to many e-book reader devices. It can go out to the Internet and fetch metadata for your books. It can download newspapers and convert them into e-books for convenient reading. It is cross platform, running on Linux, Windows and macOS.

You've just started calibre. What do you do now? Before calibre can do anything with your e-books, it first has to know about them. Drag and drop a few e-book files into calibre, or click the "Add books" button and browse for the e-books you want to work with. Once you've added the books, they will show up in the main view looking something like this:

110	The Trouble With Physics	Lee Smolin	18 Mar 2011	0.9	★★★★★
111	The Wise Man's Fear	Patrick Rothfuss	08 Mar 2011	1.4	★★★★
112	The Heroes	Joe Abercrombie	08 Mar 2011	1.2	★★★

Once you've admired the list of books you just added to your heart's content, you'll probably want to read one. In order to do that you'll have to convert the book to a format your reader understands. When first running calibre, the *Welcome wizard* starts and will set up calibre for your reader device. Conversion is a breeze. Just select the book you want to convert then click the "Convert books" button. Ignore all the options for now and click "OK". The little icon in the bottom right corner will start spinning. Once it's finished spinning, your converted book is ready. Click the "View" button to read the book.

If you want to read the book on your reader, connect the reader to the computer, wait till calibre detects it (10-20 seconds) and then click the "Send to device" button. Once the icon stops spinning again, disconnect your reader and read away! If you didn't convert the book in the previous step, calibre will auto convert it to the format your reader device understands.

To get started with more advanced usage, you should read about [The Graphical User Interface](#). For even more power and versatility, learn the [Command Line Interface](#). You will find the list of [Frequently Asked Questions](#) useful as well.

If you have more questions, or want to discuss calibre with other users or ask for help with specific things, there are [forums and other help resources available](#) [https://calibre-ebook.com/help].

Sections

- [The Graphical User Interface](#)
- [Adding your favorite news website](#)
- [The E-book viewer](#)
- [E-book conversion](#)
- [Editing e-books](#)
- [The calibre Content server](#)
- [Comparing e-books](#)
- [Editing e-book metadata](#)
- [Frequently Asked Questions](#)
- [Tutorials](#)
- [The calibre:// URL scheme](#)
- [Customizing calibre](#)
- [Command Line Interface](#)
- [Setting up a calibre development environment](#)
- [Digital Rights Management \(DRM\)](#)
- [Glossary](#)

The Graphical User Interface

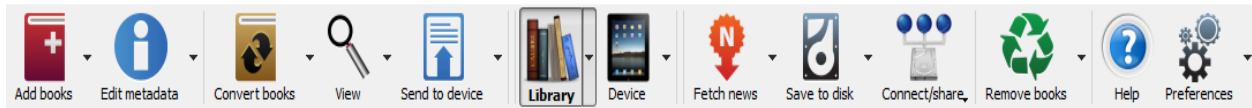
The Graphical User Interface (*GUI*) provides access to all library management and e-book format conversion features. The basic workflow for using calibre is to first add books to the library from your hard disk. calibre will automatically try to read metadata from the books and add them to its internal database. Once they are in the database, you can perform various [Actions](#) on them that include conversion from one format to another, transfer to the reading device, viewing on your computer, and editing metadata. The latter includes modifying the cover, description, and tags among other details. Note that calibre creates copies of the files you add to it. Your original files are left untouched.

The interface is divided into various sections:

- [Actions](#)
- [Preferences](#)
- [Catalogs](#)
- [Search & sort](#)
- [The search interface](#)
- [Saving searches](#)
- [Virtual libraries](#)
- [Guessing metadata from file names](#)
- [Book details](#)
- [Tag browser](#)
- [Cover grid](#)
- [Cover browser](#)
- [Quickview](#)
- [Jobs](#)

- [Keyboard shortcuts](#)

Actions

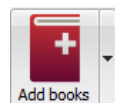


The actions toolbar provides convenient shortcuts to commonly used actions. If you right-click the buttons, you can perform variations on the default action. Please note that the actions toolbar will look slightly different depending on whether you have an e-book reader attached to your computer.

- [Add books](#)
- [Edit metadata](#)
- [Convert books](#)
- [View](#)
- [Send to device](#)
- [Fetch news](#)
- [Library](#)
- [Device](#)
- [Save to disk](#)
- [Connect/share](#)
- [Remove books](#)

Add books

The *Add books* action has seven variations accessed by doing a right-click on the button.



1. **Add books from a single folder:** Opens a file chooser dialog and allows you to specify which books in a folder should be added. This action is *context sensitive*, i.e. it depends on which [catalog](#) you have selected. If you have

selected the *Library*, books will be added to the library. If you have selected the e-book reader device, the books will be uploaded to the device, and so on.

2. **Add books from folders and sub-folders:** Allows you to choose a folder. The folder and all its sub-folders are scanned recursively, and any e-books found are added to the library. You can choose whether to have calibre add all files present in a single folder to a single book record or multiple book records. calibre assumes that each folder contains a single book. All e-book files in a folder are assumed to be the same book in different formats. This action is the inverse of the [Save to disk](#) action, i.e. you can *Save to disk*, delete the books and re-add them in single book per folder mode, with no lost information except for the date (this assumes you have not changed any of the setting for the Save to disk action).
3. **Add multiple books from archive (ZIP/RAR):** Allows you to add multiple e-books that are stored inside the selected ZIP or RAR files. It is a convenient shortcut that avoids having to first unzip the archive and then add the books via one of the above two options.
4. **Add empty book (Book Entry with no formats):** Allows you to create a blank book record. This can be used to then manually fill out the information about a book that you may not have yet in your collection.
5. **Add from ISBN:** Allows you to add one or more books by entering their ISBNs.
6. **Add files to selected book records:** Allows you to add or update the files associated with an existing book in your library.
7. **Add an empty file to selected book records:** Allows you to add an empty file of the specified format to the selected book records.

The *Add books* action can read metadata from a wide variety of e-book formats. In addition, it tries to guess metadata from the filename. See the [Guessing metadata from file names](#) section, to learn how to configure this.

To add an additional format for an existing book you can do any of three things:

1. Drag and drop the file onto the Book details panel on the right side of the main window
2. Right click the *Add books* button and choose *Add files to selected books*.
3. Click the *Add books* button in the top right area of the *Edit metadata* dialog, accessed by the [Edit metadata](#) action.

[Edit metadata](#)

The *Edit metadata* action has four variations which can be accessed by doing a right-click on the button.

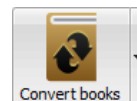


1. **Edit metadata individually:** Allows you to edit the metadata of books one-by-one with the option of fetching metadata, including covers, from the Internet. It also allows you to add or remove particular e-book formats from a book.
2. **Edit metadata in bulk:** Allows you to edit common metadata fields for large numbers of books simultaneously. It operates on all the books you have selected in the [Library view](#).
3. **Download metadata and covers:** Downloads metadata and covers (if available) for the books that are selected in the book list.
4. **Merge book records:** Gives you the capability of merging the metadata and formats of two or more book records. You can choose to either delete or keep the records that were not clicked first.

For more details see [Editing e-book metadata](#).

[Convert books](#)

E-books can be converted from a number of formats into whatever format your e-book reader prefers. Many e-books available for purchase will be protected by [Digital Rights Management](#) (DRM) technology. calibre will not convert these e-books. It is easy to remove the DRM from many formats, but as this may be illegal, you will have to find tools to liberate your books yourself and then use calibre to convert them.



For most people, conversion should be a simple one-click affair. If you want to learn more about the conversion process, see [E-book conversion](#).

The *Convert books* action has three variations, accessed by doing a right-click on the button.

1. **Convert individually:** Allows you to specify conversion options to customize the conversion of each selected e-book.
2. **Bulk convert:** Allows you to specify options only once to convert a number of e-books in bulk.
3. **Create a catalog of the books in your calibre library:** Allows you to generate a complete listing of the books in your library, including all metadata, in several formats such as XML, CSV, BiBTeX, EPUB and MOBI. The catalog will contain all the books currently showing in the library view. This allows you to use the search

features to limit the books to be catalogued. In addition, if you select multiple books using the mouse, only those books will be added to the catalog. If you generate the catalog in an e-book format such as EPUB, MOBI or AZW3, the next time you connect your e-book reader the catalog will be automatically sent to the device. For more information on how catalogs work, read the [Creating AZW3 • EPUB • MOBI catalogs](#).

[View](#)

The *View* action displays the book in an e-book viewer program. calibre has a built-in viewer for many e-book formats. For other formats it uses the default operating system application. You can configure which formats should open with the internal viewer via *Preferences* → *Interface* → *Behavior*. If a book has more than one format, you can view a particular format by doing a right-click on the button.



[Send to device](#)

The *Send to device* action has eight variations, accessed by doing a right-click on the button.



1. **Send to main memory:** The selected books are transferred to the main memory of the e-book reader.
2. **Send to card (A):** The selected books are transferred to the storage card (A) on the e-book reader.
3. **Send to card (B):** The selected books are transferred to the storage card (B) on the e-book reader.
4. **Send specific format to:** The selected books are transferred to the selected storage location on the device, in the format that you specify.
5. **Eject device:** Detaches the device from calibre.
6. **Set default send to device action:** Allows you to specify which of the options, 1 through 5 above or 7 below, will be the default action when you click the main button.
7. **Send and delete from library:** The selected books are transferred to the selected storage location on the device and then **deleted** from the Library.

8. **Fetch Annotations (experimental):** Transfers annotations you may have made on an e-book on your device to the comments metadata of the book in the calibre library.

You can control the file name and folder structure of files sent to the device by setting up a template in *Preferences* → *Import/export* → *Sending books to devices*. Also see [The calibre template language](#).

[Fetch news](#)

The *Fetch news* action downloads news from various websites and converts it into an e-book that can be read on your e-book reader. Normally, the newly created e-book is added to your e-book library, but if an e-book reader is connected at the time the download finishes, the news is also uploaded to the reader automatically.



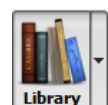
The *Fetch news* action uses simple recipes (10-15 lines of code) for each news site. To learn how to create recipes for your own news sources, see [Adding your favorite news website](#).

The *Fetch news* action has three variations, accessed by doing a right-click on the button.

1. **Schedule news download:** Allows you to schedule the download of your selected news sources from a list of hundreds available. Scheduling can be set individually for each news source you select and the scheduling is flexible allowing you to select specific days of the week or a frequency of days between downloads.
2. **Add a custom news source:** Allows you to create a simple recipe for downloading news from a custom news site that you wish to access. Creating the recipe can be as simple as specifying an RSS news feed URL, or you can be more prescriptive by creating Python-based code for the task. For more information see [Adding your favorite news website](#).
3. **Download all scheduled news sources:** Causes calibre to immediately begin downloading all news sources that you have scheduled.

[Library](#)

The *Library* action allows you to create, switch between, rename or remove a Library. calibre allows you to create as many libraries as you wish. You could, for instance, create a fiction library, a non-fiction library, a foreign language library, a project library, or any structure that suits your needs. Libraries are the highest



organizational structure within calibre. Each library has its own set of books, tags, categories and base storage location.

1. **Switch/create library...:** Allows you to; a) connect to a pre-existing calibre library at another location, b) create an empty library at a new location or, c) move the current library to a newly specified location.
2. **Quick switch:** Allows you to switch between libraries that have been registered or created within calibre.
3. **Rename library:** Allows you to rename a Library.
4. **Pick a random book:** Chooses a random book in the library for you
5. **Remove library:** Allows you to unregister a library from calibre.
6. **Export/import all calibre data:** Allows you to either export calibre data for migration to a new computer or import previously exported data.
7. **<library name>:** Actions 7, 8 etc... give you immediate switch access between multiple libraries that you have created or attached to. This list contains only the 5 most frequently used libraries. For the complete list, use the Quick Switch menu.
8. **Library maintenance:** Allows you to check the current library for data consistency issues and restore the current library's database from backups.

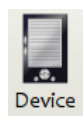
Note

Metadata about your e-books, e.g. title, author, and tags, is stored in a single file in your calibre library folder called metadata.db. If this file gets corrupted (a very rare event), you can lose the metadata. Fortunately, calibre automatically backs up the metadata for every individual book in the book's folder as an OPF file. By using the Restore database action under Library Maintenance described above, you can have calibre rebuild the metadata.db file from the individual OPF files for you.

You can copy or move books between different libraries (once you have more than one library setup) by right clicking on the book and selecting the action *Copy to library*.

Device

The *Device* action allows you to view the books in the main memory or storage cards of your device, or to eject the device (detach it from calibre). This icon shows up automatically on the main calibre toolbar when you connect a supported device. You can click on it to see the books on your device. You can also drag and drop books



from your calibre library onto the icon to transfer them to your device. Conversely, you can drag and drop books from your device onto the library icon on the toolbar to transfer books from your device to the calibre library.

Save to disk

The *Save to disk* action has five variations, accessed by doing a right-click on the button.



1. **Save to disk:** Saves the selected books to disk organized in folders.
The folder structure looks like:

```
Author_  
(  
sort  
)
```

```
Title
```

```
Book
```

```
Files
```

You can control the file name and folder structure of files saved to disk by setting up a template in *Preferences* → *Import/export* → *Saving books to disk*. Also see [The calibre template language](#).

2. **Save to disk in a single folder:** Saves the selected books to disk in a single folder.

For 1. and 2., all available formats, as well as metadata, are stored to disk for each selected book. Metadata is stored in an OPF file. Saved books can be re-imported to the library without any loss of information by using the [Add books](#) action.

3. **Save only *<your preferred>* format to disk:** Saves the selected books to disk in the folder structure as shown in (1.) but only in your preferred e-book format. You can set your preferred format in *Preferences* → *Interface* → *Behaviour* → *Preferred output format*

4. **Save only *<your preferred>* format to disk in a single folder:** Saves the selected books to disk in a single folder but only in your preferred e-book format. You can set your preferred format in *Preferences → Interface → Behaviour → Preferred output format*
5. **Save single format to disk...:** Saves the selected books to disk in the folder structure as shown in (1.) but only in the format you select from the popup list.

Connect/share

The *Connect/share* action allows you to manually connect to a device or folder on your computer. It also allows you to set up your calibre library for access via a web browser or email.



The *Connect/share* action has four variations, accessed by doing a right-click on the button.

1. **Connect to folder:** Allows you to connect to any folder on your computer as though it were a device and use all the facilities calibre has for devices with that folder. Useful if your device cannot be supported by calibre but is available as a USB disk.
2. **Start Content server:** Starts calibre's built-in web server. When started, your calibre library will be accessible via a web browser from the Internet (if you choose). You can configure how the web server is accessed by setting preferences at *Preferences → Sharing → Sharing over the net*
3. **Setup email based sharing of books:** Allows sharing of books and news feeds by email. After setting up email addresses for this option, calibre will send news updates and book updates to the entered email addresses. You can configure how calibre sends email by setting preferences at *Preferences → Sharing → Sharing books by email*. Once you have set up one or more email addresses, this menu entry will be replaced by menu entries to send books to the configured email addresses.

Remove books

The *Remove books* action **deletes books permanently**, so use it with care. It is *context sensitive*, i.e. it depends on which [catalog](#) you have selected. If you have selected the *Library*, books will be removed from the library. If you have selected the e-book reader device, books will be removed from the device. To remove only a particular format for a given book use the [Edit metadata](#) action. Remove books also has five variations which can be accessed by doing a right-click on the button.



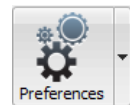
1. **Remove selected books:** Allows you to **permanently** remove all books that are selected in the book list.
2. **Remove files of a specific format from selected books...:** Allows you to **permanently** remove e-book files of a specified format from books that are selected in the book list.
3. **Remove all formats from selected books, except...:** Allows you to **permanently** remove e-book files of any format except a specified format from books that are selected in the book list.
4. **Remove all formats from selected books:** Allows you to **permanently** remove all e-book files from books that are selected in the book list. Only the metadata will remain.
5. **Remove covers from selected books:** Allows you to **permanently** remove cover image files from books that are selected in the book list.
6. **Remove matching books from device:** Allows you to remove e-book files from a connected device that match the books that are selected in the book list.

Note

Note that when you use *Remove books* to delete books from your calibre library, the book record is permanently deleted, but the files are placed into the *Recycle Bin/Trash*. This allows you to recover the files if you change your mind.

Preferences

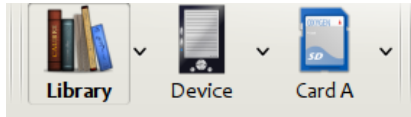
The *Preferences* action allows you to change the way various aspects of calibre work. It has four variations, accessed by doing a right-click on the button.



1. **Preferences:** Allows you to change the way various aspects of calibre work. Clicking the button also performs this action.
2. **Run Welcome wizard:** Allows you to start the *Welcome wizard* which appeared the first time you started calibre.
3. **Get plugins to enhance calibre:** Opens a new window that shows plugins for calibre. These plugins are developed by third parties to extend calibre's functionality.
4. **Restart in debug mode:** Allows you to enable a debugging mode that can assist the calibre developers in solving problems you encounter with the program. For

most users this should remain disabled unless instructed by a developer to enable it.

Catalogs



A *catalog* is a collection of books. calibre can manage two types of different catalogs:

1. **Library:** This is a collection of books stored in your calibre library on your computer.
2. **Device:** This is a collection of books stored in your e-book reader. It will be available when you connect the reader to your computer.

Many operations, such as adding books, deleting, viewing, etc., are context sensitive. So, for example, if you click the *View* button when you have the **Device** catalog selected, calibre will open the files on the device to view. If you have the **Library** catalog selected, files in your calibre library will be opened instead.

Search & sort

Search:

	Title	Author(s)	Size (MB)	Date	Rating	Publisher	Tags	Series
1	The Complete Works of William Shakespeare	William Shakespeare	2.4	02 Jan 2007	★★★★★	manybooks.net		
2	Stalky and Co.	Rudyard Kipling	0.2	19 Jan 2007	★★★★★	manybooks.net		
3	The Comedies of William Shakespeare	William Shakespeare	2.1	15 Mar 2007	★★★★★			
4	The Histories of William Shakespeare	William Shakespeare	1.5	15 Mar 2007	★★★★★		england, historical fiction	
5	The Tragedies of William Shakespeare	William Shakespeare	1.6	15 Mar 2007	★★★★★			
6	War and Peace	Leo Tolstoy	3.1	22 Aug 2007	★★★★★	gutenberg.org	classic	
7	Anna Karenina	Leo Tolstoy	1.9	22 Aug 2007	★★★★★	gutenberg.org	classic	
8	Guns, germs, and steel: the fates of human societies	Jared Diamond	0.4	29 Nov 2007	★★★★★	New York : W.W. Norton, c1997.		
9	A Game of Thrones	George R. R. Martin	1.3	23 Jan 2007	★★★★★		fantasy	
10	A Clash of Kings	George R. R. Martin	1.4	25 Jan 2007	★★★★★		fantasy	
11	A Storm of Swords	George R. R. Martin	1.9	27 Jan 2007	★★★★★		fantasy	
12	A Feast for Crows	George R. R. Martin	1.7	29 Jan 2007	★★★★★		fantasy	Song of Ice and Fire [4]
13	Renegades	Isabel Cole	0.0	00 May 2007	★★★★★		fantasy	The Sundering [1]

The Search & Sort section allows you to perform several powerful actions on your book collections.

- You can sort them by title, author, date, rating, etc. by clicking on the column titles. You can also sub-sort, i.e. sort on multiple columns. For example, if you click on the title column and then the author column, the book will be sorted by author and then all the entries for the same author will be sorted by title.
- You can search for a particular book or set of books using the Search bar. More on that below.
- You can quickly and conveniently edit metadata by selecting the entry you want changed in the list and pressing the E key.
- You can perform [Actions](#) on sets of books. To select multiple books you can either:
 - Keep the Ctrl key pressed and click on the books you want selected.
 - Keep the Shift key pressed and click on the starting and ending book of a range of books you want selected.
- You can configure which fields you want displayed by using the [Preferences](#) dialog.

The search interface

You can search all the metadata by entering search terms in the Search bar. Searches are case insensitive. For example:

Asimov

Foundation

format

:

lrf

This will match all books in your library that have Asimov Foundation and in their metadata and are available in the LRF format. Some more examples:

author

:

Asimov

and

not

series

:

Foundation

title

:

"The Ring"

or

"This book is about a ring"

format

:

epub

```
publisher
:
feedbooks
.
com
```

Searches are by default 'contains'. An item matches if the search string appears anywhere in the indicated metadata. Two other kinds of searches are available: equality search and search using [regular expressions](https://en.wikipedia.org/wiki/Regular_expression) [https://en.wikipedia.org/wiki/Regular_expression].

Equality searches are indicated by prefixing the search string with an equals sign (=). For example, the query tag:"=science" will match "science", but not "science fiction" or "hard science". Regular expression searches are indicated by prefixing the search string with a tilde (~). Any [Python-compatible regular expression](https://docs.python.org/library/re.html) [https://docs.python.org/library/re.html] can be used. Note that backslashes used to escape special characters in regular expressions must be doubled because single backslashes will be removed during query parsing. For example, to match a literal parenthesis you must enter \(. Regular expression searches are 'contains' searches unless the expression contains anchors.

Should you need to search for a string with a leading equals or tilde, prefix the string with a backslash.

Enclose search strings with quotes (") if the string contains parenthesis or spaces. For example, to search for the tag Science

Fiction you would need to search for tag:"=science

fiction". If you search for tag:=science

fiction you will find all books with the tag 'science' and containing the word 'fiction' in any metadata.

You can build advanced search queries easily using the *Advanced search dialog* accessed by clicking the button .

Available fields for searching are: tag, title,

author,
publisher,
series,
series_index,
rating,
cover,
comments,
format,
identifiers,
date,
pubdate,
search,
size,

vl

and custom columns. If a device is plugged in, the ondevice field becomes available, when searching the calibre library view. To find the search name (actually called the *lookup name*) for a custom column, hover your mouse over the column header in the library view.

The syntax for searching for dates is:

```
pubdate  
:  
>  
2000  
-  
1  
  
Will  
  
find
```

all

books

published

after

Jan

,

2000

date

:

<=

2000

-

1

-

3

Will

find

all

books

added

to

calibre

before

3

Jan

,

2000

pubdate

:=

2009

Will

find

all

books

published

in

2009

If the date is ambiguous, the current locale is used for date comparison. For example, in an mm/dd/yyyy locale 2/1/2009 is interpreted as 1 Feb 2009. In a dd/mm/yyyy locale it is interpreted as 2 Jan 2009. Some special date strings are available. The string `today` translates to today's date, whatever it is. The strings `yesterday` and `thismonth` (or the translated equivalent in the current language) also work. In addition, the string `daysago` (also translated) can be used to compare to a date some number of days ago. For example:

date

:

>

10

daysago

date

:

<=

45

daysago

To avoid potential problems with translated strings when using a non-English version of calibre, the strings `_today`, `_yesterday`, `_thismonth`, `_daysago` and `_daysago` are always available. They are not translated.

You can search for books that have a format of a certain size like this:

```
size
:
>
1.1
M

Will

find

books

with

a

format

larger

than

1.1
MB

size
:
<=
1
K

Will
```


find
books
with
a
format
smaller
than
1
KB

Dates and numeric fields support the relational operators = (equals), > (greater than), >= (greater than or equal to), < (less than), <= (less than or equal to), and != (not equal to). Rating fields are considered to be numeric. For example, the search rating:>=3 will find all books rated 3 or higher.

You can search for the number of items in multiple-valued fields such as tags. These searches begin with the character #, then use the same syntax as numeric fields. For example, to find all books with more than 4 tags use tags:#>4. To find all books with exactly 10 tags use tags:#=10.

Series indices are searchable. For the standard series, the search name is 'series_index'. For custom series columns, use the column search name followed by _index. For example, to search the indices for a custom series column named #my_series, you would use the search name #my_series_index. Series indices are numbers, so you can use the relational operators described above.

The special field search is used for saved searches. So if you save a search with the name "My spouse's books" you

can enter search:"My

spouse's

books"

in the Search bar to reuse the saved search. More about saving searches below.

The special field `vl` is used to search for books in a Virtual library. For example, `vl:Read` will find all the books in the *Read* Virtual library. The search `vl:Read`

and

`vl:"Science`

`Fiction"`

will find all the books that are in both the *Read* and *Science Fiction* Virtual libraries. The value following `vl:`

must be the name of a Virtual library. If the Virtual library name contains spaces then surround it with quotes.

You can search for the absence or presence of a field using the special "true" and "false" values. For example:

```
cover
:
false
```

```
will
```

```
give
```

```
you
```

```
all
```

```
books
```

```
without
```

```
a
```

```
cover
```

series

:

true

will

give

you

all

books

that

belong

to

a

series

comments

:

false

will

give

you

all

books

with

an

empty

comment

format

:

false

will

give

you

all

books

with

no

actual

files

(

empty

records

)

Yes/no custom columns are searchable. Searching for false
, empty
, or blank
will find all books with undefined values in the column. Searching for true
will find all books that do not have undefined values in the column. Searching for yes
or checked
will find all books with Yes
in the column. Searching for no
or unchecked
will find all books with No

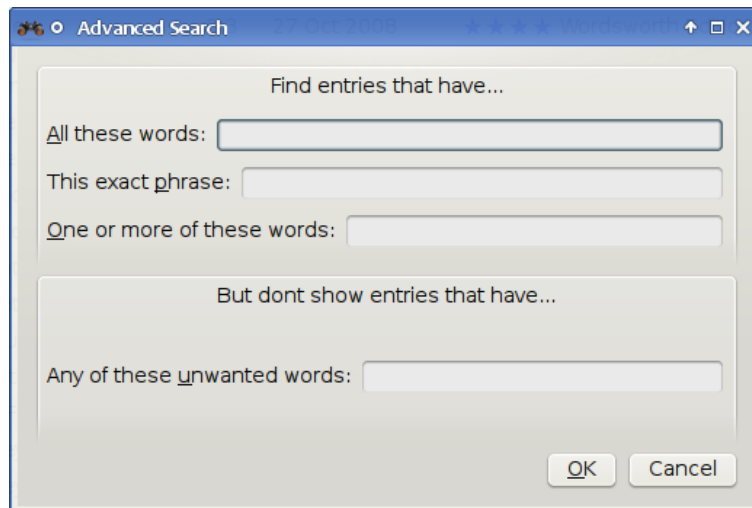
in the column. Note that the words yes
, no
, blank
, empty
, checked
and unchecked
are translated; you can use either the current language's equivalent word or the English
word. The words true
and false
and the special values _yes
, _no
, and _empty
are not translated.

Hierarchical items (e.g. A.B.C) use an extended syntax to match initial parts of the hierarchy. This is done by adding a period between the exact match indicator (=) and the text. For example, the query tags:=.A will find the tags *A* and *A.B*, but will not find the tags *AA* or *AA.B*. The query tags:=.A.B will find the tags *A.B* and *A.B.C*, but not the tag *A*.

Identifiers (e.g., ISBN, DOI, LCCN, etc.) also use an extended syntax. First, note that an identifier has the form type:value, as in isbn:123456789. The extended syntax permits you to specify independently which type and value to search for. Both the type and the value parts of the query can use *equality*, *contains*, or *regular expression* matches. Examples:

- identifiers:true
will find books with any identifier.
- identifiers:false
will find books with no identifier.
- identifiers:123
will search for books with any type having a value containing *123*.
- identifiers:=123456789
will search for books with any type having a value equal to *123456789*.
- identifiers:=isbn:
and identifiers:isbn:true
will find books with a type equal to ISBN having any value
- identifiers:=isbn:false
will find books with no type equal to ISBN.

- `identifiers:=isbn:123`
will find books with a type equal to ISBN having a value containing *123*.
- `identifiers:=isbn:=123456789`
will find books with a type equal to ISBN having a value equal to *123456789*.
- `identifiers:i:1`
will find books with a type containing an *i* having a value containing a *1*.



Advanced search dialog

You can search using a template in the [The calibre template language](#) instead of a metadata field. To do so you enter a template, a search type, and the value to search for. The syntax is:

```
template
:
(
the
template
)

#@#: (search type) : (the value)
```

The is any valid calibre template language template. The search type

must be one of the following search types:
 (text search),
 (date search),
 (numeric search), or
 (set/not set (boolean)).
 The value is whatever you want. It can use the special operators described above for the various search types. You must quote the entire search string if there are spaces anywhere in it.

Examples:

- template:"program:
 connected_device_name('main')#@#:t:kindle"
 – is true when the kindle
 device is connected
- template:"program:
 select(formats_sizes(),
 'Epub')#@#:n:>1000000"
 – finds books with EPUB files larger than 1 MB
- template:"program:
 select(formats_modtimes('iso'),
 'Epub')#@#:d:>10daysago"
 – finds books with EPUB files newer than 10 days ago

You can build template search queries easily using the *Advanced search dialog* accessed by clicking the button . You can test templates on specific books using the calibre *Template tester*. This can be added to the toolbars or menus via *Preferences → Toolbars & menus*. It can also be assigned a keyboard shortcut via *Preferences → Shortcuts*.

[Saving searches](#)

calibre allows you to save a frequently used search under a special name and then reuse that search with a single click. To do this, create your search either by typing it in the Search bar or using the Tag browser. Then type the name you would like to give to the search in the Saved Searches box next to the Search bar. Click the plus icon next to the saved searches box to save the search.

Now you can access your saved search in the Tag browser under *Saved searches*. A single click will allow you to reuse any arbitrarily complex search easily, without needing to re-create it.

Virtual libraries

A *Virtual library* is a way to pretend that your calibre library has only a few books instead of its full collection. This is an excellent way to partition your large collection of books into smaller, manageable chunks. To learn how to create and use Virtual libraries, see the tutorial: [Virtual libraries](#).

Guessing metadata from file names

Normally, calibre reads metadata from inside the book file. However, it can be configured to read metadata from the file name instead, via *Preferences* → *Import/export* → *Adding books* → *Read metadata from file contents*.

You can also control how metadata is read from the filename using regular expressions (see [All about using regular expressions in calibre](#)). In the *Adding books* section of the configuration dialog, you can specify a regular expression that calibre will use to try and guess metadata from the names of e-book files that you add to the library. The default regular expression is:

```
title
-
author
```

that is, it assumes that all characters up to the first - are the title of the book and subsequent characters are the author of the book. For example, the filename:

```
Foundation
and
Earth
-
```


Isaac

Asimov

.

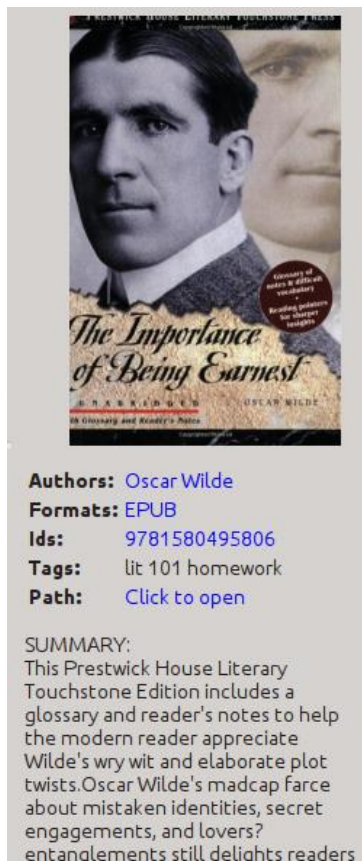
txt

will be interpreted to have the title: Foundation and Earth and author: Isaac Asimov

Tip

If the filename does not contain the hyphen, the above regular expression will fail.

Book details



shows the cover and all the metadata for the currently selected book. If any of the *Layout* button in the lower right corner of the main calibre interface is shown in the Book details panel are click-able, they will by default link to the Wikipedia page for the author. This can be customized by right-clicking on the link and selecting *Manage this author*.

When clicking on the metadata for the book, the Book details panel will automatically open the web pages for the book on Amazon, WorldCat, etc. from where you can purchase or borrow the book.

You can click on individual e-book formats in the Book details panel to delete them, download them, or save original versions, save them to disk, open them with an external program, etc.

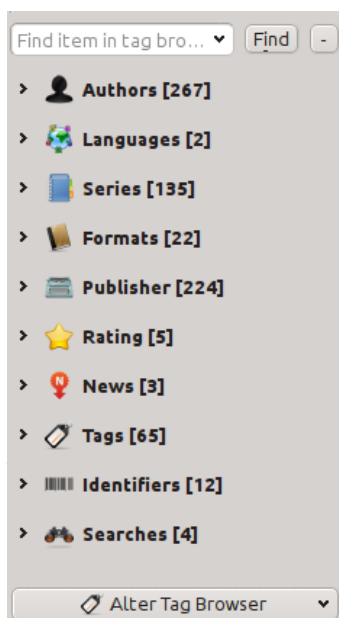
You can change the cover of the book by simply drag and dropping an image onto the Book details panel. To edit the cover image in an external program, simply right click on the cover image and select *Edit cover*.

You can add new files to the current book by drag and dropping the files onto the Book details panel.

If you right-click on the Book details panel will open it up in a separate popup window.

Finally, you can customize exactly what information is displayed in the Book details panel via *Preferences* → *Interface* → *Look & feel* → *Book details*.

Tag browser



The Tag browser allows you to easily browse your collection by Author/Tags/Series/etc. When you click on a tag in the Tag browser, for example the author name Isaac Asimov, then the list of books shown is restricted to showing books by that author. You can click on any other tag to change the restriction. For example, clicking on "Series" will show you all books in any series.

You can also restrict the list of books to those that contain or match the item. For example, clicking on Isaac Asimov will show books by that author. The list of books will change what is shown, depending on whether the item has a series or not (and other series and hierarchical items below). Continuing the Isaac Asimov example, clicking on Isaac Asimov will restrict the list of books to those not by Isaac Asimov. To remove the restriction, showing all books. If you hold down the Ctrl key while clicking on a tag, it will remove the restriction, showing all books.

or Shift keys and click on multiple items, then restrictions based on multiple items are created. For example you could hold Ctrl and click on the tags History and Europe for finding books on European history. The Tag browser works by constructing search expressions that are automatically entered into the Search bar. Looking at what the Tag browser generates is a good way to learn how to construct basic search expressions.

Items in the Tag browser have their icons partially colored. The amount of color depends on the average rating of the books in that category. So for example if the books by Isaac Asimov have an average of four stars, the icon for Isaac Asimov in the Tag browser will be 4/5th colored. You can hover your mouse over the icon to see the average rating.

The outer-level items in the *Tag browser*, such as Authors and Series, are called categories. You can create your own categories, called *User categories*, which are useful for organizing items. For example, you can use the *User categories editor* (click the *Configure* button at the lower-left of the *Tag browser* and choose *Manage authors, tags, etc* → *User categories*) to create a User category called Favorite

Authors

, then put the items for your favorites into the category. User categories can have sub-categories. For example, the User category Favorites.Authors is a sub-category of Favorites. You might also have Favorites.Series, in which case there will be two sub-categories under Favorites. Sub-categories can be created by right-clicking on a User category, choosing *Add sub-category to...*, and entering the sub-category name; or by using the *User categories editor* by entering names like the Favorites example above.

You can search User categories in the same way as built-in categories, by clicking on them. There are four different searches cycled through by clicking:

1. “everything matching an item in the category” indicated by a single green plus sign.
2. “everything matching an item in the category or its sub-categories” indicated by two green plus signs.
3. “everything not matching an item in the category” shown by a single red minus sign.
4. “everything not matching an item in the category or its sub-categories” shown by two red minus signs.

It is also possible to create hierarchies inside some of the text categories such as tags, series, and custom columns. These hierarchies show with the small triangle, permitting the sub-items to be hidden. To use hierarchies of items in a category, you must first go to *Preferences* → *Interface* → *Look & feel* and enter the category name(s) into the “Categories with

hierarchical items” field. Once this is done, items in that category that contain periods will be shown using the small triangle. For example, assume you create a custom column called “Genre” and indicate that it contains hierarchical items. Once done, items such as Mystery.Thriller and Mystery.English will display as Mystery with the small triangle next to it. Clicking on the triangle will show Thriller and English as sub-items. See [Managing subgroups of books, for example “genre”](#) for more information.

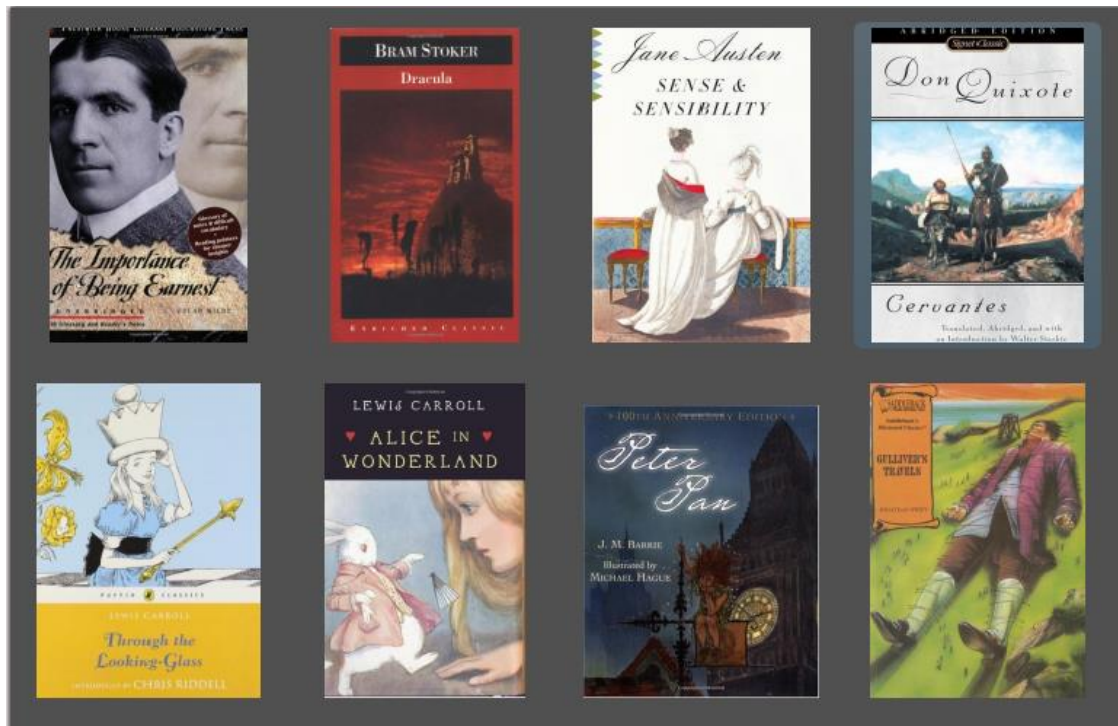
Hierarchical items (items with children) use the same four ‘click-on’ searches as User categories. Items that do not have children use two of the searches: “everything matching” and “everything not matching”.

You can drag and drop items in the Tag browser onto User categories to add them to that category. If the source is a User category, holding the Shift key while dragging will move the item to the new category. You can also drag and drop books from the book list onto items in the Tag browser; dropping a book on an item causes that item to be automatically applied to the dropped books. For example, dragging a book onto Isaac Asimov will set the author of that book to Isaac Asimov. Dropping it onto the tag History will add the tag History to the book’s tags.

You can easily find any item in the Tag browser by clicking the search button at the lower-right corner. In addition, you can right click on any item and choose one of several operations. Some examples are to hide it, rename it, or open a “Manage x” dialog that allows you to manage items of that kind. For example, the “Manage Authors” dialog allows you to rename authors and control how their names are sorted.

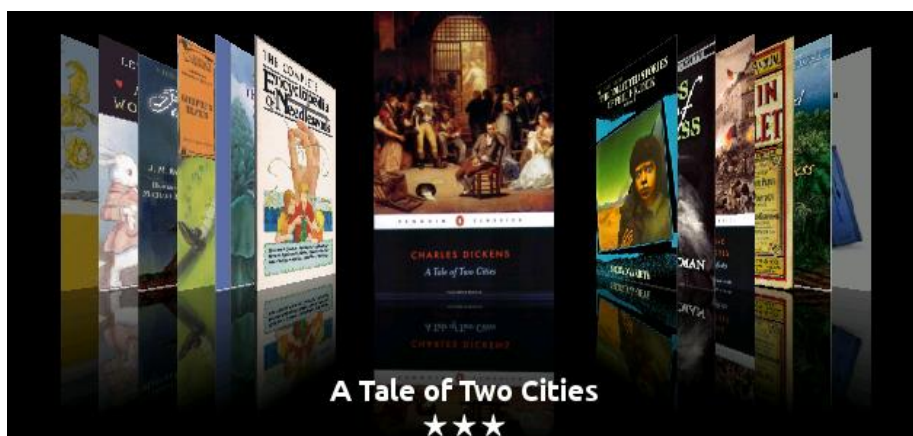
You can control how items are sorted in the Tag browser via the *Configure* button at the lower-left of the Tag browser. You can choose to sort by name, average rating or popularity (popularity is the number of books with an item in your library; for example, the popularity of Isaac Asimov is the number of books in your library by Isaac Asimov).

[Cover grid](#)



You can have calibre display a grid of book covers instead of a list of books, if you prefer to browse your collection by covers instead. The *Cover grid* is activated by clicking the *Layout* button in the bottom right corner of the main calibre window. You can customize the cover sizes and the background of the *Cover grid* via *Preferences* → *Interface* → *Look & feel* → *Cover grid*. You can even have calibre display any specified field under the covers, such as title or authors or rating or a custom column of your own devising.

[Cover browser](#)



In addition to the *Cover grid* described above, you can also have calibre display covers in the single row. This is activated via the *Layout* button in the lower right corner of the main window. In *Preferences* → *Interface* → *Look & feel* → *Cover browser* you can change the

number of covers displayed, and even have the *Cover browser* display itself in a separate popup window.

Quickview

Sometimes you want to select a book and quickly get a list of books with the same value in some category (authors, tags, publisher, series, etc.) as the currently selected book, but without changing the current view of the library. You can do this with Quickview. Quickview opens either a second window or a panel in the book list showing the list of books matching the value of interest. For example, assume you want to see a list of all the books with the one or more of the authors of the currently-selected book. Click in the author cell you are interested in and press the 'Q' key or click the *Quickview* icon in the *Layout* section of the calibre window. A window or panel will open with all the authors for that book on the left, and all the books by the selected author on the right.

Some example Quickview usages: quickly seeing what other books:

- have some tag(s) applied to the currently selected book,
- are in the same series as the current book
- have the same values in a custom column as the current book
- are written by one of the same authors of the current book
- share values in a custom column

There are two choices for where the Quickview information appears:

1. It can open “undocked”: on top of the calibre window and will stay open until you explicitly close it.
2. It can open “docked”: as a panel in the book list section of the calibre main window.

You can move the window from docked to undocked as desired using the “Dock/Undock” button.

The Quickview panel can be left open permanently, in which case it follows movements on the book list. For example, if you click in the calibre library view on a category column (tags, series, publisher, authors, etc.) for a book, the Quickview window contents will change to show you in the left-hand side panel the values in that category for the selected book (e.g., the tags for that book). The first item in that list will be selected, and Quickview will show you on the right-hand side panel all the books in your library that use that value. Click on a different value in the left-hand panel to see the books with that different value.

Double-click on a book in the Quickview window to select that book in the library view. This will also change the items display in the QuickView window (the left-hand panel) to show the items in the newly-selected book.

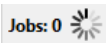
Shift- or Ctrl- double-click on a book in the Quickview window to open the edit metadata dialog on that book in the calibre window. The edited book will be Quickview'ed when you close the edit metadata dialog.

You can see if a column can be Quickview'ed by hovering your mouse over the column heading and looking at the tooltip for that heading. You can also know by right-clicking on the column heading to see if the "Quickview" option is shown in the menu, in which case choosing that Quickview option is equivalent to pressing 'Q' in the current cell.

Options (in *Preferences* → *Look & feel* → *Quickview*):

- Respect (or not) the current Virtual library. If checked then Quickview shows only books in the current Virtual library. Default: respect Virtual libraries
- Change the Quickview window contents when the column is changed on the book list using the cursor keys. Default: don't follow changes made with cursor keys
- Change the column being "quickview'ed" when a cell in the Quickview window is double-clicked. Otherwise the book is changed but the column being examined is not. Default: change the column
- Change the column being "quickview'ed" to the current column when the return key is pressed in the Quickview panel. Otherwise the book is changed but the column being examined is not. Default: change the column
- Choose which columns are shown in the Quickview window/panel.

[Jobs](#)

 Jobs panel shows the number of currently running jobs. Jobs are tasks that run in a separate process. They include converting e-books and talking to your reader device. You can click on the jobs panel to access the list of jobs. Once a job has completed you can see a detailed log from that job by double-clicking it in the list. This is useful to debug jobs that may not have completed successfully.

[Keyboard shortcuts](#)

calibre has several keyboard shortcuts to save you time and mouse movement. These shortcuts are active in the book list view (when you're not editing the details of a particular book), and most of them affect the title you have selected. The calibre E-book viewer has its own shortcuts which can be customised by clicking the *Preferences* button in the viewer.

Note

Note: The calibre keyboard shortcuts do not require a modifier key (Command, Option, Control, etc.), unless specifically noted. You only need to press the letter key, e.g. E to edit.

Keyboard shortcuts for the main calibre program

Keyboard shortcut	Action
F2 (Enter in macOS)	Edit the metadata of the currently selected field in the book list.
A	Add books
Shift+A	Add formats to the selected books
C	Convert selected books
D	Send to device
Del	Remove selected books
E	Edit metadata of selected books
G	Get books
I	Show Book details
K	Edit Table of Contents

M	Merge selected records
Alt+M	Merge selected records, keeping originals
O	Open containing folder
P	Polish books
S	Save to disk
T	Edit book
V	View
Shift+V	View last read book
Alt+V/Cmd+V in macOS	View specific format
Alt+Shift+J	Toggle jobs list
Alt+Shift+B	Toggle Cover browser
Alt+Shift+D	Toggle Book details panel
Alt+Shift+T	Toggle Tag browser
Alt+Shift+G	Toggle Cover grid
Alt+A	Show books by the same author as the current book
Alt+T	Show books with the same tags as current book

Alt+P	Show books by the same publisher as current book
Alt+Shift+S	Show books in the same series as current book
/, Ctrl+F	Focus the Search bar
Shift+Ctrl+F	Open the Advanced search dialog
Shift+Alt+F	Toggle the Search bar
Esc	Clear the current search
Shift+Esc	Focus the book list
Ctrl+Esc	Clear the Virtual library
Alt+Esc	Clear the additional restriction
Ctrl+*	Create a temporary Virtual library based on the current search
Ctrl+Right	Select the next Virtual library tab
Ctrl+Left	Select the previous Virtual library tab
N or F3	Find the next book that matches the current search (only works if search highlighting is turned on in search preferences)
Shift+N or Shift+F3	Find the previous book that matches the current search (only works if search highlighting is turned on in search preferences)
Ctrl+D	Download metadata and covers

Ctrl+R	Restart calibre
Ctrl+Shift+R	Restart calibre in debug mode
Shift+Ctrl+E	Add empty books to calibre
Ctrl+M	Toggle mark/unmarked status on selected books
Q	Open the Quick View popup for viewing books in related series/tags/etc.
Shift+Q	Focus the opened Quick View panel
Shift+S	Perform a search in the Quick View panel
F5	Re-apply the current sort
Ctrl+Q	Quit calibre
X	Toggle auto scroll of the book list

Adding your favorite news website

calibre has a powerful, flexible and easy-to-use framework for downloading news from the Internet and converting it into an e-book. The following will show you, by means of examples, how to get news from various websites.

To gain an understanding of how to use the framework, follow the examples in the order listed below:

- [Completely automatic fetching](#)
 - [The calibre blog](#)
 - [bbc.co.uk](#)
- [Customizing the fetch process](#)
 - [Using the print version of bbc.co.uk](#)
 - [Replacing article styles](#)
 - [Slicing and dicing](#)
 - [Real life example](#)
- [Tips for developing new recipes](#)
- [Further reading](#)
- [API documentation](#)

[Completely automatic fetching](#)

If your news source is simple enough, calibre may well be able to fetch it completely automatically, all you need to do is provide the URL. calibre gathers all the information needed to download a news source into a [recipe](#). In order to tell calibre about a news source, you have to create a [recipe](#) for it. Let's see some examples:

[The calibre blog](#)

The calibre blog is a blog of posts that describe many useful calibre features in a simple and accessible way for new calibre users. In order to download this blog into an e-book, we rely on the [RSS](#) feed of the blog:

```
http
:
//
blog
.
calibre
-
ebook
.
com
/
feeds
/
posts
/
default
```

I got the RSS URL by looking under “Subscribe to” at the bottom of the blog page and choosing *Posts* → *Atom*. To make calibre download the feeds and convert them into an e-book, you should right click the *Fetch news* button and then the *Add a custom news source* menu item and then the *New Recipe* button. A dialog similar to that shown below should open up.

Create a basic news recipe, by adding RSS feeds to it.
For some news sources, you will have to use the "Switch to advanced mode" button below to further customize the fetch process.

Recipe title:

Oldest article:

Max. number of articles per feed:

Feeds in recipe

▲

✖

▼

Add feed to recipe

Feed title:

Feed URL:

First

enter

Calibre

Blog

into the *Recipe title* field. This will be the title of the e-book that will be created from the articles in the above feeds.

The next two fields (*Oldest article* and *Max. number of articles*) allow you some control over how many articles should be downloaded from each feed, and they are pretty self explanatory.

To add the feeds to the recipe, enter the feed title and the feed URL and click the *Add feed* button. Once you have added the feed, simply click the *Save* button and you're done! Close the dialog.

To test your new [recipe](#), click the *Fetch news* button and in the *Custom news sources* submenu click *calibre Blog*. After a couple of minutes, the newly downloaded e-book of blog posts will appear in the main library view (if you have your reader connected, it will be put onto the reader instead of into the library). Select it and hit the *View* button to read!

The reason this worked so well, with so little effort is that the blog provides *full-content* [RSS](#) feeds, i.e., the article content is embedded in the feed itself. For most news sources that provide news in this fashion, with *full-content* feeds, you don't need any more effort to convert them to e-books. Now we will look at a news source that does not provide full content feeds. In such feeds, the full article is a webpage and the feed only contains a link to the webpage with a short summary of the article.

bbc.co.uk

Let's try the following two feeds from *The BBC*:

1. News Front Page:
https://newsrss.bbc.co.uk/rss/newsonline_world_edition/front_page/rss.xml
2. Science/Nature:
https://newsrss.bbc.co.uk/rss/newsonline_world_edition/science/nature/rss.xml

Follow the procedure outlined in [The calibre blog](#) above to create a recipe for *The BBC* (using the feeds above). Looking at the downloaded e-book, we see that calibre has done a creditable job of extracting only the content you care about from each article's webpage. However, the extraction process is not perfect. Sometimes it leaves in undesirable content like menus and navigation aids or it removes content that should have been left alone, like article headings. In order, to have perfect content extraction, we will need to customize the fetch process, as described in the next section.

[Customizing the fetch process](#)

When you want to perfect the download process, or download content from a particularly complex website, you can avail yourself of all the power and flexibility of the [recipe](#) framework. In order to do that, in the *Add custom news sources* dialog, simply click the *Switch to Advanced mode* button.

The easiest and often most productive customization is to use the print version of the online articles. The print version typically has much less cruft and translates much more smoothly to an e-book. Let's try to use the print version of the articles from *The BBC*.

[Using the print version of bbc.co.uk](#)

The first step is to look at the e-book we downloaded previously from bbc.co.uk. At the end of each article, in the e-book is a little blurb telling you where the article was downloaded from. Copy and paste that URL into a browser. Now on the article webpage look for a link that points to the "Printable version". Click it to see the print version of the article. It looks much neater! Now compare the two URLs. For me they were:

Article URL

<https://news.bbc.co.uk/2/hi/science/nature/7312016.stm>

Print version URL

<https://newsvote.bbc.co.uk/mpapps/pagetools/print/news.bbc.co.uk/2/hi/science/nature/7312016.stm>

So it looks like to get the print version, we need to prefix every article URL with:

newsvote.bbc.co.uk/mpapps/pagetools/print/

Now in the *Advanced mode* of the Custom news sources dialog, you should see something like (remember to select *The BBC* recipe before switching to advanced mode):

A screenshot of a code editor window titled "Recipe source code (python)". The code defines a class "AdvancedUserRecipe1206418393(BasicNewsRecipe)". Inside the class, there are attributes: "title" set to "The BBC", "oldest_article" set to 7, "max_articles_per_feed" set to 100, and "feeds" set to a list containing a tuple with "News Front Page" and a URL "http://newsrss.bbc.co.uk/rss/newsonlin...".

```
Recipe source code (python)
class AdvancedUserRecipe1206418393(BasicNewsRecipe):
    title = u'The BBC'
    oldest_article = 7
    max_articles_per_feed = 100

    feeds = [(u'News Front Page', u'http://newsrss.bbc.co.uk/rss/newsonlin...
```

You can see that the fields from the *Basic mode* have been translated to Python code in a straightforward manner. We need to add instructions to this recipe to use the print version of the articles. All that's needed is to add the following two lines:

```
def
print_version
(
self
,
url
):

return
url
.
replace
(
'https://'
,
'https://newsvote.bbc.co.uk/mpapps/pagetools/print/'
)
```


This is Python, so indentation is important. After you've added the lines, it should look like:

```
Recipe source code (python)
class AdvancedUserRecipe1206418393(BasicNewsRecipe):
    title = u'The BBC'
    oldest_article = 7
    max_articles_per_feed = 100

    feeds = [(u'News Front Page', u'http://newsrss.bbc.co.uk/rss/newsonlin

    def print_version(self, url):
        return url.replace('http://', 'http://newsvote.bbc.co.uk/mpapps/pagetools/p
```

In the above, the `def print_version(self,`

`url)` defines a *method* that is called by calibre for every article. `url` is the URL of the original article. What `print_version` does is take that `url` and replace it with the new URL that points to the print version of the article. To learn about [Python](https://www.python.org) [https://www.python.org] see the [tutorial](https://docs.python.org/tutorial/) [https://docs.python.org/tutorial/].

Now, click the *Add/update recipe* button and your changes will be saved. Re-download the e-book. You should have a much improved e-book. One of the problems with the new version is that the fonts on the print version webpage are too small. This is automatically fixed when converting to an e-book, but even after the fixing process, the font size of the menus and navigation bar become too large relative to the article text. To fix this, we will do some more customization, in the next section.

Replacing article styles

In the previous section, we saw that the font size for articles from the print version of *The BBC* was too small. In most websites, *The BBC* included, this font size is set by means of [CSS](#) stylesheets. We can disable the fetching of such stylesheets by adding the line:

```
no_stylesheets
```

```
=
```

```
True
```

The recipe now looks like:

```
Recipe source code (python)
class AdvancedUserRecipe1206419520(BasicNewsRecipe):
    title = u'The BBC'
    oldest_article = 7
    max_articles_per_feed = 100
    no_stylesheets = True

    feeds = [(u'News Front Page', u'http://newsrss.bbc.co.uk/rss/newsonli

    def print_version(self, url):
        return url.replace('http://', 'http://newsvote.bbc.co.uk/mpapps/pagetools/
```

The new version looks pretty good. If you're a perfectionist, you'll want to read the next section, which deals with actually modifying the downloaded content.

Slicing and dicing

calibre contains very powerful and flexible abilities when it comes to manipulating downloaded content. To show off a couple of these, let's look at our old friend the [The BBC](#) recipe again. Looking at the source code ([HTML](#)) of a couple of articles (print version), we see that they have a footer that contains no useful information, contained in

```
<
div

class
=
"footer"
>

...
</
div
>
```

This can be removed by adding:

```
remove_tags
```

```
=  
  
[  
dict  
(  
name  
=  
'div'  
,  
  
attrs  
=  
{  
'class'  
:  
'footer'  
}}]  
]]
```

to the recipe. Finally, lets replace some of the [CSS](#) that we disabled earlier, with our own [CSS](#) that is suitable for conversion to an e-book:

```
extra_css  
  
=  
  
'<div class="headline">{font-size: x-large;}  
<div class="fact">{padding-top: 10pt }'  
</div>
```

With these additions, our recipe has become “production quality”.

This [recipe](#) explores only the tip of the iceberg when it comes to the power of calibre. To explore more of the abilities of calibre we’ll examine a more complex real life example in the next section.

[Real life example](#)

A reasonably complex real life example that exposes more of the [API](#) of BasicNewsRecipe is the [recipe](#) for *The New York Times*

```
import
string
,
re
from
calibre
import
strftime
from
calibre.web.feeds.recipes
import
BasicNewsRecipe
from
calibre.ebooks.BeautifulSoup
import
BeautifulSoup
class
NYTimes
(
BasicNewsRecipe
):
title
```

=

'The New York Times'

__author__

=

'Kovid Goyal'

description

=

'Daily news from the New York Times'

timefmt

=

' [
%a
,
%d
%b, %Y]'

needs_subscription

=

True

remove_tags_before

=

dict

(

```
id
=
'article'
)

remove_tags_after

=

dict
(
id
=
'article'
)

remove_tags

=

[
dict
(
attrs
=
{
'class'
: [
'articleTools'
,
'post-tools'
,
'side_tool'
,
'nextArticleLink clearfix'
]
}),
```

```
dict
(
id
=
[
'footer'
,

'toolsRight'
,

'articleInline'
,

'navigation'
,

'archive'
,

'side_search'
,

'blog_sidebar'
,

'side_tool'
,

'side_index'
]),
```

```
dict
(
name
=
[
'script'
,

'noscript'
,
```

```
'style'
```

```
]])
```

```
encoding
```

```
=
```

```
'cp1252'
```

```
no_stylesheets
```

```
=
```

```
True
```

```
extra_css
```

```
=
```

```
'h1 {font: sans-serif large;}
```

```
\n
```

```
.byline {font:monospace;}'
```

```
def
```

```
get_browser
```

```
(
```

```
self
```

```
):
```

```
br
```

```
=
```

```
BasicNewsRecipe
```

```
.
```



```
get_browser
()

if
self
.
username

is

not

None

and

self
.
password

is

not

None
:

br
.
open
(
https://www.nytimes.com/auth/login
)

br
.
select_form
(
name
```

```
=  
'login'  
)  
  
br  
[  
'USERID'  
]  
  
=  
  
self  
.  
username  
  
br  
[  
'PASSWORD'  
]  
  
=  
  
self  
.  
password  
  
br  
.  
submit  
()  
  
return  
  
br  
  
def
```

```
parse_index
(
self
):

soup

=

self
.
index_to_soup
(
'https://www.nytimes.com/pages/todayspaper/index.html'
)

def

feed_title
(
div
):

return

"
.
join
(
div
.
findAll
(
text
=
True
,
recursive
```

```
=  
False
```

```
))
```

```
.  
strip  
(
```

```
articles
```

```
=
```

```
{}
```

```
key
```

```
=
```

```
None
```

```
ans
```

```
=
```

```
[]
```

```
for
```

```
div
```

```
in
```

```
soup
```

```
.  
findAll  
(  
True  
,
```

```
attrs
=
{
'class'
:[
'section-headline'
,

'story'
,

'story headline'
] }):
```

```
if
```

```
"
.
join
(
div
[
'class'
])
```

```
==
```

```
'section-headline'
:
```

```
key
```

```
=
```

```
string
```

```
.
capwords
(
feed_title
```

```
(
div
))

articles
[
key
]

=

[]

ans
.
append
(
key
)

elif

"
.
join
(
div
[
'class'
])

in

[
'story'
,
'story headline'
]:
```

```
a
```

```
=
```

```
div
```

```
.
```

```
find
```

```
(
```

```
'a'
```

```
,
```

```
href
```

```
=
```

```
True
```

```
)
```

```
if
```

```
not
```

```
a
```

```
:
```

```
continue
```

```
url
```

```
=
```

```
re
```

```
.
```

```
sub
```

```
(
```

```
r
```

```
'\?.*'
```

```
,
```

```
"
```

```
,

a
[
'href'
])

url

+=

'?pagewanted=all'

title

=

self
.
tag_to_string
(
a
,
use_alt
=
True
)
.
strip
()

description

=

"

pubdate
```


=

strftime

```
(  
,  
%a  
,  
%d  
%b'  
)
```

summary

=

div

```
.  
find  
(  
True  
,
```

attrs

```
=  
{  
'class'  
:  
'summary'  
})
```

if

summary

:

description

=

```
self
.  
tag_to_string  
(  
summary  
,  
  
use_alt  
=  
False  
)  
  
feed  
  
=  
  
key  
  
if  
  
key  
  
is  
  
not  
  
None  
  
else  
  
'Uncategorized'  
  
if  
  
feed  
  
not  
  
in
```

```
articles
:
```

```
articles
[
feed
]
```

```
=
```

```
[]
```

```
if
```

```
not
```

```
'podcasts'
```

```
in
```

```
url
:
```

```
articles
[
feed
]
```

```
.
append
(
```

```
dict
```

```
(
title
=
title
,
```

```
url
```

```
=
url
,

date
=
pubdate
,

description
=
description
,

content
=
"
))

ans

=

self
.
sort_index_by
(
ans
,

{
'The Front Page'
:
-
1
,

'Dining In, Dining Out'
:
1
```

```
,  
  
'Obituaries'  
:  
2  
)
```

```
ans
```

```
=
```

```
[(  
key  
,
```

```
articles  
[  
key  
)
```

```
for
```

```
key
```

```
in
```

```
ans
```

```
if
```

```
key
```

```
in
```

```
articles  
]
```

```
return
```

```
ans
```

```
def

preprocess_html
(
self
,

soup
):

refresh

=

soup
.
find
(
'meta'
,

{
'http-equiv'
:
'refresh'
})

if

refresh

is

None
:

return
```

```
soup

content

=

refresh

.
get
(
'content'
)
.
partition
(
'='
)[
2
]

raw

=

self

.
browser

.
open
(
'https://www.nytimes.com'
+
content
)
.
read
()

return
```

```
BeautifulSoup
```

```
(  
raw  
.  
decode  
(  
'cp1252'  
,  
'replace'  
)  
)
```

We see several new features in this [recipe](#). First, we have:

```
timefmt
```

```
=  
  
' [  
%a  
,  
%d  
%b, %Y]'
```

This sets the displayed time on the front page of the created e-book to be in the format, Day,

Day_Number

Month,

Year

.
.

See

[timefmt](#)

Then we see a group of directives to cleanup the downloaded [HTML](#):

```
remove_tags_before
```

```
=
```



```
dict
(
name
=
'h1'
)

remove_tags_after

=

dict
(
id
=
'footer'
)

remove_tags

=

...
```

These remove everything before the first <h1>
tag and everything after the first tag whose id is footer
. See **remove_tags**
, **remove_tags_before**
, **remove_tags_after**
.

The next interesting feature is:

```
needs_subscription

=

True

...
```

```
def
get_browser
(
self
):
...

```

needs_subscription

=

True

tells calibre that this recipe needs a username and password in order to access the content. This causes, calibre to ask for a username and password whenever you try to use this recipe. The code in [calibre.web.feeds.news.BasicNewsRecipe.get_browser\(\)](#) actually does the login into the NYT website. Once logged in, calibre will use the same, logged in, browser instance to fetch all content. See [mechanize](https://mechanize.readthedocs.io/en/latest/) [https://mechanize.readthedocs.io/en/latest/] to understand the code in [get_browser](#).

The next new feature is the [calibre.web.feeds.news.BasicNewsRecipe.parse_index\(\)](#) method. Its job is to go to <https://www.nytimes.com/pages/todayspaper/index.html> and fetch the list of articles that appear in *today's* paper. While more complex than simply using [RSS](#), the recipe creates an e-book that corresponds very closely to the days paper.

`parse_index` makes heavy use of [BeautifulSoup](https://www.crummy.com/software/BeautifulSoup/bs4/doc/) [https://www.crummy.com/software/BeautifulSoup/bs4/doc/] to parse the daily paper webpage. You can also use other, more modern parsers if you dislike BeautifulSoup. calibre comes with [lxml](https://lxml.de/) [https://lxml.de/] and [html5lib](https://github.com/html5lib/html5lib-python) [https://github.com/html5lib/html5lib-python], which are the recommended parsers. To use them, replace the call to `index_to_soup()` with the following:

```
raw
=
self
.
index_to_soup

```

```
(
url
,

raw
=
True
)

# For html5lib

import

html5lib

root

=

html5lib
.
parse
(
raw
,

namespaceHTMLElements
=
False
,

treebuilder
=
'xml'
)

# For the lxml html 4 parser

from

lxml

import
```

```
html
root
=
html
.
fromstring
(
raw
)
```

The final new feature is the [calibre.web.feeds.news.BasicNewsRecipe.preprocess_html\(\)](#) method. It can be used to perform arbitrary transformations on every downloaded HTML page. Here it is used to bypass the ads that the nytimes shows you before each article.

[Tips for developing new recipes](#)

The best way to develop new recipes is to use the command line interface. Create the recipe using your favorite Python editor and save it to a file say myrecipe.recipe. The *.recipe* extension is required. You can download content using this recipe with the command:

```
ebook
-
convert

myrecipe
.
recipe

.
epub

--
test

-
vv
```

```
--
debug
-
pipeline

debug
```

The command **ebook-convert** will download all the webpages and save them to the EPUB file `myrecipe.epub`. The `-vv` option makes ebook-convert spit out a lot of information about what it is doing. The **ebook-convert-recipe-input**

--test

option makes it download only a couple of articles from at most two feeds. In addition, ebook-convert will put the downloaded HTML into the `debug/input` folder, where the `debug` is the folder you specified in the **ebook-convert**

--debug-pipeline

option.

Once the download is complete, you can look at the downloaded [HTML](#) by opening the file `debug/input/index.html` in a browser. Once you're satisfied that the download and preprocessing is happening correctly, you can generate e-books in different formats as shown below:

```
ebook
-
convert

myrecipe
-
recipe

myrecipe
-
epub

ebook
-
```

```
convert
myrecipe
.
recipe
myrecipe
.
mobi
...
```

If you're satisfied with your recipe, and you feel there is enough demand to justify its inclusion into the set of built-in recipes, post your recipe in the [calibre recipes forum](https://www.mobileread.com/forums/forumdisplay.php?f=228) [https://www.mobileread.com/forums/forumdisplay.php?f=228] to share it with other calibre users.

Note

On macOS, the command line tools are inside the calibre bundle, for example, if you installed calibre in /Applications the command line tools are in /Applications/calibre.app/Contents/MacOS/.

See also

[ebook-convert](#)

The command line interface for all e-book conversion.

[Further reading](#)

To learn more about writing advanced recipes using some of the facilities, available in BasicNewsRecipe you should consult the following sources:

[API documentation](#)

Documentation of the BasicNewsRecipe class and all its important methods and fields.

[BasicNewsRecipe](https://github.com/kovidgoyal/calibre/blob/master/src/calibre/web/feeds/news.py) [https://github.com/kovidgoyal/calibre/blob/master/src/calibre/web/feeds/news.py]

The source code of BasicNewsRecipe

[Built-in recipes](https://github.com/kovidgoyal/calibre/tree/master/recipes) [https://github.com/kovidgoyal/calibre/tree/master/recipes]

The source code for the built-in recipes that come with calibre

[The calibre recipes forum](https://www.mobileread.com/forums/forumdisplay.php?f=228) [https://www.mobileread.com/forums/forumdisplay.php?f=228]

Lots of knowledgeable calibre recipe writers hang out here.

API documentation

- [API documentation for recipes](#)

API documentation for recipes

The API for writing recipes is defined by the [BasicNewsRecipe](#)

class calibre.web.feeds.news.BasicNewsRecipe(*options, log, progress_reporter*)[\[source\]](#)

Base class that contains logic needed in all recipes. By overriding progressively more of the functionality in this class, you can make progressively more customized/powerful recipes. For a tutorial introduction to creating recipes, see [Adding your favorite news website](#).

abort_article(msg=None)[\[source\]](#)

Call this method inside any of the preprocess methods to abort the download for the current article. Useful to skip articles that contain inappropriate content, such as pure video articles.

abort_recipe_processing(msg)[\[source\]](#)

Causes the recipe download system to abort the download of this recipe, displaying a simple feedback message to the user.

add_toc_thumbnail(article, src)[\[source\]](#)

Call this from `populate_article_metadata` with the `src` attribute of an `<img>` tag from the article that is appropriate for use as the thumbnail representing the article in the Table of Contents. Whether the thumbnail is actually used is device dependent (currently only used by the Kindles). Note that the referenced image must be one that was successfully downloaded, otherwise it will be ignored.

classmethod adeify_images(*soup*)[\[source\]](#)

If your recipe when converted to EPUB has problems with images when viewed in Adobe Digital Editions, call this method from within [postprocess_html\(\)](#).

canonicalize_internal_url(url, is_link=True)[\[source\]](#)

Return a set of canonical representations of url. The default implementation uses just the server hostname and path of the URL, ignoring

any query parameters, fragments, etc. The canonical representations must be unique across all URLs for this news source. If they are not, then internal links may be resolved incorrectly.

Parameters **is_link** – Is True if the URL is coming from an internal link in an HTML file. False if the URL is the URL used to download an article.

`cleanup()`[\[source\]](#)

Called after all articles have been download. Use it to do any cleanup like logging out of subscription sites, etc.

`clone_browser(br)`[\[source\]](#)

Clone the browser *br*. Cloned browsers are used for multi-threaded downloads, since mechanize is not thread safe. The default cloning routines should capture most browser customization, but if you do something exotic in your recipe, you should override this method in your recipe and clone manually.

Cloned browser instances use the same, thread-safe CookieJar by default, unless you have customized cookie handling.

`default_cover(cover_file)`[\[source\]](#)

Create a generic cover for recipes that don't have a cover

`download()`[\[source\]](#)

Download and pre-process all articles from the feeds in this recipe. This method should be called only once on a particular Recipe instance. Calling it more than once will lead to undefined behavior. :return: Path to index.html

`extract_readable_article(html, url)`[\[source\]](#)

Extracts main article content from 'html', cleans up and returns as a (article_html, extracted_title) tuple. Based on the original readability algorithm by Arc90.

`get_article_url(article)`[\[source\]](#)

Override in a subclass to customize extraction of the [URL](#) that points to the content for each article. Return the article URL. It is called with *article*, an object representing a parsed article from a feed. See [feedparser](https://pythonhosted.org/feedparser/) [https://pythonhosted.org/feedparser/]. By default it looks for the original link (for feeds syndicated via a service like FeedBurner or Pheedo) and if found, returns that or else returns [article.link](https://pythonhosted.org/feedparser/reference-entry-link.html) [https://pythonhosted.org/feedparser/reference-entry-link.html].

`get_browser(*args, **kwargs)`[\[source\]](#)

Return a browser instance used to fetch documents from the web. By default it returns a [mechanize](https://mechanize.readthedocs.io/en/latest/) [https://mechanize.readthedocs.io/en/latest/] browser instance that supports cookies, ignores robots.txt, handles refreshes and has a mozilla firefox user agent.

If your recipe requires that you login first, override this method in your subclass. For example, the following code is used in the New York Times recipe to login for full access:

```
def
get_browser
(
self
):

br

=

BasicNewsRecipe
.
get_browser
(
self
)

if
self
.
username

is
not
None
and
self
.
```

password

is

not

None

:

br

.

open

(

'https://www.nytimes.com/auth/login'

)

br

.

select_form

(

name

=

'login'

)

br

[

'USERID'

]

=

self

.

username

br

[

'PASSWORD'

```
]

=

self
.
password

br
.
submit
()

return

br
```

`get_cover_url()`[\[source\]](#)

Return a [URL](#) to the cover image for this issue or *None*. By default it returns the value of the member *self.cover_url* which is normally *None*. If you want your recipe to download a cover for the e-book override this method in your subclass, or set the member variable *self.cover_url* before this method is called.

`get_extra_css()`[\[source\]](#)

By default returns *self.extra_css*. Override if you want to programmatically generate the extra_css.

`get_feeds()`[\[source\]](#)

Return a list of [RSS](#) feeds to fetch for this profile. Each element of the list must be a 2-element tuple of the form (title, url). If title is *None* or an empty string, the title from the feed is used. This method is useful if your recipe needs to do some processing to figure out the list of feeds to download. If so, override in your subclass.

`get_masthead_title()`[\[source\]](#)

Override in subclass to use something other than the recipe title

`get_masthead_url()`[\[source\]](#)

Return a [URL](#) to the masthead image for this issue or *None*. By default it returns the value of the member *self.masthead_url* which is normally *None*. If you want your recipe to download a masthead for the e-book override this method in your subclass, or set the member variable *self.masthead_url* before this method is called. Masthead images are used in Kindle MOBI files.

`get_obfuscated_article(url)`[\[source\]](#)

If you set *articles_are_obfuscated* this method is called with every article URL. It should return the path to a file on the filesystem that contains the article HTML. That file is processed by the recursive HTML fetching engine, so it can contain links to pages/images on the web.

This method is typically useful for sites that try to make it difficult to access article content automatically.

classmethod `image_url_processor(baseurl, url)`[\[source\]](#)

Perform some processing on image urls (perhaps removing size restrictions for dynamically generated images, etc.) and return the precessed URL.

`index_to_soup(url_or_raw, raw=False, as_tree=False, save_raw=None)`[\[source\]](#)

Convenience method that takes an URL to the index page and returns a [BeautifulSoup](#) [\[https://www.crummy.com/software/BeautifulSoup/bs4/doc\]](https://www.crummy.com/software/BeautifulSoup/bs4/doc) of it.

url_or_raw: Either a URL or the downloaded index page as a string

`is_link_wanted(url, tag)`[\[source\]](#)

Return True if the link should be followed or False otherwise. By default, raises `NotImplementedError` which causes the downloader to ignore it.

Parameters

- **url** – The URL to be followed
- **tag** – The tag from which the URL was derived

`parse_feeds()`[\[source\]](#)

Create a list of articles from the list of feeds returned by [BasicNewsRecipe.get_feeds\(\)](#).
Return a list of **Feed** objects.

`parse_index()`[\[source\]](#)

This method should be implemented in recipes that parse a website instead of feeds to generate a list of articles. Typical uses are for news sources that have a “Print Edition” webpage that lists all the articles in the current print edition. If this function is implemented, it will be used in preference to [BasicNewsRecipe.parse_feeds\(\)](#).

It must return a list. Each element of the list must be a 2-element tuple of the form ('feed title',

list

of

articles)

.

Each list of articles must contain dictionaries of the form:

```
{
  'title'
  :
  article
  title
  ,
  'url'
  :
  URL
  of
  print
  version
  ,
```

'date'

:

The

publication

date

of

the

article

as

a

string

,

'description'

:

A

summary

of

the

article

'content'

:

The

full

article

(
can

be

an

empty

string
)

.

Obsolete

do

not

use

,

instead

save

the

content

to

a

temporary


```
file
and
pass
a
file
:
///
path
/
to
/
temp
/
file
.
html
as
the
URL
.
}
```

For an example, see the recipe for downloading *The Atlantic*. In addition, you can add ‘author’ for the author of the article.

If you want to abort processing for some reason and have calibre show the user a simple message instead of an error, call [abort_recipe_processing\(\)](#).

`populate_article_metadata(article, soup, first)`[\[source\]](#)

Called when each HTML page belonging to article is downloaded. Intended to be used to get article metadata like author/summary/etc. from the parsed HTML (soup).

Parameters

- **article** – A object of class **calibre.web.feeds.Article**. If you change the summary, remember to also change the text_summary
- **soup** – Parsed HTML belonging to this article
- **first** – True iff the parsed HTML is the first page of the article.

`postprocess_book(oeb, opts, log)`[\[source\]](#)

Run any needed post processing on the parsed downloaded e-book.

Parameters

- **oeb** – An OEBBook object
- **opts** – Conversion options

`postprocess_html(soup, first_fetch)`[\[source\]](#)

This method is called with the source of each downloaded [HTML](#) file, after it is parsed for links and images. It can be used to do arbitrarily powerful post-processing on the [HTML](#). It should return *soup* after processing it.

Parameters

- **soup** – A [BeautifulSoup](#) [https://www.crummy.com/software/BeautifulSoup/bs4/doc/] instance containing the downloaded [HTML](#).
- **first_fetch** – True if this is the first page of an article.

`preprocess_html(soup)`[\[source\]](#)

This method is called with the source of each downloaded [HTML](#) file, before it is parsed for links and images. It is called after the cleanup as specified by `remove_tags` etc. It can be used to do arbitrarily powerful pre-processing on the [HTML](#). It should return *soup* after processing it.

soup: A [BeautifulSoup](#) [https://www.crummy.com/software/BeautifulSoup/bs4/doc/] instance containing the downloaded [HTML](#).

`preprocess_image(img_data, image_url)`[\[source\]](#)

Perform some processing on downloaded image data. This is called on the raw data before any resizing is done. Must return the processed raw data. Return None to skip the image.

`preprocess_raw_html(raw_html, url)`[\[source\]](#)

This method is called with the source of each downloaded [HTML](#) file, before it is parsed into an object tree. `raw_html` is a unicode string representing the raw HTML downloaded from the web. `url` is the URL from which the HTML was downloaded.

Note that this method acts *before* `preprocess_regexps`.

This method must return the processed `raw_html` as a unicode object.

classmethod `print_version(url)`[\[source\]](#)

Take a `url` pointing to the webpage with article content and return the [URL](#) pointing to the print version of the article. By default does nothing. For example:

```
def
print_version
(
self
,
url
):

return
url
+
'?'&pagewanted=print'
```

`skip_ad_pages(soup)`[\[source\]](#)

This method is called with the source of each downloaded [HTML](#) file, before any of the cleanup attributes like `remove_tags`, `keep_only_tags` are applied. Note that

`preprocess_regexps` will have already been applied. It is meant to allow the recipe to skip ad pages. If the soup represents an ad page, return the HTML of the real page. Otherwise return `None`.

soup: A [BeautifulSoup](https://www.crummy.com/software/BeautifulSoup/bs4/doc/) [https://www.crummy.com/software/BeautifulSoup/bs4/doc/] instance containing the downloaded [HTML](#).

`sort_index_by(index, weights)` [\[source\]](#)

Convenience method to sort the titles in *index* according to *weights*. *index* is sorted in place. Returns *index*.

index: A list of titles.

weights: A dictionary that maps weights to titles. If any titles in *index* are not in *weights*, they are assumed to have a weight of 0.

classmethod `tag_to_string(tag, use_alt=True, normalize_whitespace=True)` [\[source\]](#)

Convenience method to take a [BeautifulSoup](https://www.crummy.com/software/BeautifulSoup/bs4/doc/) Tag [https://www.crummy.com/software/BeautifulSoup/bs4/doc/] and extract the text from it recursively, including any CDATA sections and alt tag attributes. Return a possibly empty Unicode string.

use_alt: If *True* try to use the alt attribute for tags that don't have any textual content

tag: [BeautifulSoup](https://www.crummy.com/software/BeautifulSoup/bs4/doc/) Tag [https://www.crummy.com/software/BeautifulSoup/bs4/doc/]

`articles_are_obfuscated = False`

Set to `True` and implement [get_obfuscated_article\(\)](#) to handle websites that try to make it difficult to scrape content.

`auto_cleanup = False`

Automatically extract all the text from downloaded article pages. Uses the algorithms from the readability project. Setting this to `True`, means that you do not have to worry about cleaning up the downloaded HTML manually (though manual cleanup will always be superior).

`auto_cleanup_keep = None`

Specify elements that the auto cleanup algorithm should never remove. The syntax is a XPath expression. For example:

auto_cleanup_keep

=

'//div[@id="article-image"]'

will

keep

all

divs

with

id

=

"article-image"

auto_cleanup_keep

=

'//*[@class="important"]'

will

keep

all

elements

with

class

=

"important"

auto_cleanup_keep

```
=  
  
'//div[@id="article-image"]|//span[@class="important"]'  
  
will  
keep  
all  
divs  
  
with  
  
id  
=  
"article-image"  
  
and  
  
spans  
  
with  
  
class  
=  
"important"
```

`center_navbar = True`

If True the navigation bar is center aligned, otherwise it is left aligned

`compress_news_images = False`

Set this to False to ignore all scaling and compression parameters and pass images through unmodified. If True and the other compression parameters are left at their default values, JPEG images will be scaled to fit in the screen dimensions set by the output profile and compressed to size at most $(w * h)/16$ where $w \times h$ are the scaled image dimensions.

`compress_news_images_auto_size = 16`

The factor used when auto compressing JPEG images. If set to `None`, auto compression is disabled. Otherwise, the images will be reduced in size to $(w * h) / \text{compress\_news\_images\_auto\_size}$ bytes if possible by reducing the quality level, where $w \times h$ are the image dimensions in pixels. The minimum JPEG quality will be 5/100 so it is possible this constraint will not be met. This parameter can be overridden by the parameter `compress_news_images_max_size` which provides a fixed maximum size for images. Note that if you enable `scale_news_images_to_device` then the image will first be scaled and then its quality lowered until its size is less than $(w * h) / \text{factor}$ where w and h are now the *scaled* image dimensions. In other words, this compression happens after scaling.

`compress_news_images_max_size = None`

Set JPEG quality so images do not exceed the size given (in KBytes). If set, this parameter overrides auto compression via `compress_news_images_auto_size`. The minimum JPEG quality will be 5/100 so it is possible this constraint will not be met.

`conversion_options = {}`

Recipe specific options to control the conversion of the downloaded content into an e-book. These will override any user or plugin specified values, so only use if absolutely necessary. For example:

```
conversion_options
```

```
=
```

```
{
```

```
    'base_font_size'
```

```
:
```

```
    16
```

```
,
```

```
    'linearize_tables'
```

```
:
```

```
    True
```

```
,
```

```
}
```

`cover_margins = (0, 0, '#ffffff')`

By default, the cover image returned by `get_cover_url()` will be used as the cover for the periodical. Overriding this in your recipe instructs calibre to render the downloaded cover into a frame whose width and height are expressed as a percentage of the downloaded cover. `cover_margins = (10, 15, '#ffffff')` pads the cover with a white margin 10px on the left and right, 15px on the top and bottom. Color names are defined [here](https://www.imagemagick.org/script/color.php) [https://www.imagemagick.org/script/color.php]. Note that for some reason, white does not always work in Windows. Use `#ffffff` instead

`delay = 0`

Delay between consecutive downloads in seconds. The argument may be a floating point number to indicate a more precise time.

`description = "`

A couple of lines that describe the content this recipe downloads. This will be used primarily in a GUI that presents a list of recipes.

`encoding = None`

Specify an override encoding for sites that have an incorrect charset specification. The most common being specifying latin1 and using cp1252. If `None`, try to detect the encoding. If it is a callable, the callable is called with two arguments: The recipe object and the source to be decoded. It must return the decoded source.

`extra_css = None`

Specify any extra [CSS](#) that should be added to downloaded [HTML](#) files. It will be inserted into `<style>` tags, just before the closing `</head>` tag thereby overriding all [CSS](#) except that which is declared using the style attribute on individual [HTML](#) tags. Note that if you want to programmatically generate the `extra_css` override the [get_extra_css\(\)](#) method instead. For example:

```
extra_css
```

```
=
```

```
'.heading { font: serif x-large }'
```



```
feeds = None
```

List of feeds to download. Can be either [url1, url2, ...] or [('title1', url1), ('title2', url2),...]

```
filter_regexps = []
```

List of regular expressions that determines which links to ignore. If empty it is ignored. Used only if `is_link_wanted` is not implemented. For example:

```
filter_regexps
=
[
r
'ads\.doubleclick\.net'
]
```

will remove all URLs that have *ads.doubleclick.net* in them.

Only one of `BasicNewsRecipe.match_regexps` or `BasicNewsRecipe.filter_regexps` should be defined.

```
handle_gzip = True
```

Set to False if you don't want to use gzipped transfers. Note that some old servers flake out with gzip

`ignore_duplicate_articles = None`

Ignore duplicates of articles that are present in more than one section. A duplicate article is an article that has the same title and/or URL. To ignore articles with the same title, set this to:

```
ignore_duplicate_articles
=
{
'title'
}
```

To use URLs instead, set it to:

```
ignore_duplicate_articles
=
{
'url'
}
```

To match on title or URL, set it to:

```
ignore_duplicate_articles
=
{
'title'
,
'url'
}
```

`keep_only_tags = []`

Keep only the specified tags and their children. For the format for specifying a tag see [BasicNewsRecipe.remove_tags](#)

. If this list is not empty, then the `<body>` tag will be emptied and re-filled with the tags that match the entries in this list. For example:

```
keep_only_tags
```

```
=
```

```
[
```

```
dict
```

```
(
```

```
id
```

```
=
```

```
[
```

```
'content'
```

```
,
```

```
'heading'
```

```
]]
```

will keep only tags that have an *id* attribute of “*content*” or “*heading*”.

```
language = 'und'
```

The language that the news is in. Must be an ISO-639 code either two or three characters long

```
masthead_url = None
```

By default, calibre will use a default image for the masthead (Kindle only). Override this in your recipe to provide a url to use as a masthead.

```
match_regexps = []
```

List of regular expressions that determines which links to follow. If empty, it is ignored. Used only if `is_link_wanted` is not implemented. For example:

```
match_regexps
```

```
=
```

```
[
```

```
r
'page=[0-9]+'
```

will match all URLs that have *page=some number* in them.

Only one of [BasicNewsRecipe.match_regexps](#) or [BasicNewsRecipe.filter_regexps](#) should be defined.

`max_articles_per_feed = 100`

Maximum number of articles to download from each feed. This is primarily useful for feeds that don't have article dates. For most feeds, you should use [BasicNewsRecipe.oldest_article](#)

`needs_subscription = False`

If True the GUI will ask the user for a username and password to use while downloading. If set to "optional" the use of a username and password becomes optional

`no_stylesheets = False`

Convenient flag to disable loading of stylesheets for websites that have overly complex stylesheets unsuitable for conversion to e-book formats. If True stylesheets are not downloaded and processed

`oldest_article = 7.0`

Oldest article to download from this news source. In days.

`preprocess_regexps = []`

List of [regex](#) substitution rules to run on the downloaded [HTML](#). Each element of the list should be a two element tuple. The first element of the tuple should be a compiled regular expression and the second a callable that takes a single match object and returns a string to replace the match. For example:

```
preprocess_regexps
```

```
=
```

```
[
```

```

(
re
.
compile
(
r
'<!--Article ends here-->.*</body>'
,

re
.
DOTALL
|
re
.
IGNORECASE
),

lambda

match
:

'</body>'
),

]

```

will remove everything from *<!--Article ends here-->* to *</body>*.

`publication_type = 'unknown'`

Publication type Set to newspaper, magazine or blog. If set to None, no publication type metadata will be written to the opf file.

`recipe_disabled = None`

Set to a non empty string to disable this recipe. The string will be used as the disabled message

`recursions = 0`

Number of levels of links to follow on article webpages

`remove_attributes = []`

List of attributes to remove from all tags. For example:

```
remove_attributes
```

```
=
```

```
[
```

```
'style'
```

```
,
```

```
'font'
```

```
]
```

`remove_empty_feeds = False`

If True empty feeds are removed from the output. This option has no effect if `parse_index` is overridden in the sub class. It is meant only for recipes that return a list of feeds using `feeds` or `get_feeds()`

. It is also used if you use the `ignore_duplicate_articles` option.

`remove_javascript = True`

Convenient flag to strip all JavaScript tags from the downloaded HTML

`remove_tags = []`

List of tags to be removed. Specified tags are removed from downloaded HTML. A tag is specified as a dictionary of the form:

```
{
```

```
name
```

```
:
```

```
'tag name'
```

```
,  
  
#e.g. 'div'  
  
attrs  
  
:  
  
a  
  
dictionary  
,  
  
#e.g. {'class': 'advertisement'}  
  
}
```

All keys are optional. For a full explanation of the search criteria, see [Beautiful Soup](https://www.crummy.com/software/BeautifulSoup/bs4/doc/#searching-the-tree) [https://www.crummy.com/software/BeautifulSoup/bs4/doc/#searching-the-tree] A common example:

```
remove_tags  
  
=  
  
[  
dict  
(  
name  
=  
'div'  
,  
  
class_  
=  
'advert'  
)]
```

This will remove all `<div class="advert">` tags and all their children from the downloaded [HTML](#).

```
remove_tags_after = None
```

Remove all tags that occur after the specified tag. For the format for specifying a tag see [BasicNewsRecipe.remove_tags](#). For example:

```
remove_tags_after
```

```
=
```

```
[
```

```
dict
```

```
(
```

```
id
```

```
=
```

```
'content'
```

```
)]
```

will remove all tags after the first element with `id="content"`.

```
remove_tags_before = None
```

Remove all tags that occur before the specified tag. For the format for specifying a tag see [BasicNewsRecipe.remove_tags](#). For example:

```
remove_tags_before
```

```
=
```

```
dict
```

```
(
```

```
id
```

```
=
```

```
'content'
```

```
)
```

will remove all tags before the first element with `id="content"`.

`requires_version = (0, 6, 0)`

Minimum calibre version needed to use this recipe

`resolve_internal_links = False`

If set to True then links in downloaded articles that point to other downloaded articles are changed to point to the downloaded copy of the article rather than its original web URL. If you set this to True, you might also need to implement [canonicalize_internal_url\(\)](#) to work with the URL scheme of your particular website.

`reverse_article_order = False`

Reverse the order of articles in each feed

`scale_news_images = None`

Maximum dimensions (w,h) to scale images to. If `scale_news_images_to_device` is True this is set to the device screen dimensions set by the output profile unless there is no profile set, in which case it is left at whatever value it has been assigned (default None).

`scale_news_images_to_device = True`

Rescale images to fit in the device screen dimensions set by the output profile. Ignored if no output profile is set.

`simultaneous_downloads = 5`

Number of simultaneous downloads. Set to 1 if the server is picky. Automatically reduced to 1 if [BasicNewsRecipe.delay](#) > 0

`summary_length = 500`

Max number of characters in the short description

```
template_css = '\n .article_date {\n color: gray; font-family: monospace;\n }\n\n .article_description {\n text-indent: 0pt;\n }\n\n a.article {\n font-weight: bold; text-align:left;\n }\n\n a.feed {\n font-weight: bold;\n }\n\n .calibre_navbar {\n font-family:monospace;\n }\n'
```

The CSS that is used to style the templates, i.e., the navigation bars and the Tables of Contents. Rather than overriding this variable, you should use *extra_css* in your recipe to customize look and feel.

`timefmt = ' [%a, %d %b %Y]'`

The format string for the date shown on the first page. By default: Day_Name, Day_Number Month_Name Year

`timeout = 120.0`

Timeout for fetching files from server in seconds

`title = 'Unknown News Source'`

The title to use for the e-book

`use_embedded_content = None`

Normally we try to guess if a feed has full articles embedded in it based on the length of the embedded content. If *None*, then the default guessing is used. If *True* then we always assume the feeds has embedded content and if *False* we always assume the feed does not have embedded content.



The E-book viewer

calibre includes a built-in E-book viewer that can view all the major e-book formats. The E-book viewer is highly customizable and has many advanced features.

- [Starting the E-book viewer](#)
- [Navigating around an e-book](#)
- [Highlighting text](#)
- [Read aloud](#)
- [Following links using only the keyboard](#)
- [Customizing the look and feel of your reading experience](#)
- [Dictionary lookup](#)
- [Copying text and images](#)
- [Zooming in on images](#)
- [Non re-flowable content](#)
- [Designing your book to work well with the calibre viewer](#)

[Starting the E-book viewer](#)

You can view any of the books in your calibre library by selecting the book and pressing the *View* button. This will open up the book in the E-book viewer. You can also launch the E-book viewer by itself from the Start menu in Windows. On macOS, you can pin it to the dock and launch it from there. On Linux you can use its launcher in the desktop menus or run the command **ebook-viewer**.

[Navigating around an e-book](#)

You can “turn pages” in a book by either:

- Clicking in the left or right margin or the page with the mouse
- Pressing the spacebar, page up, page down or arrow keys
- On a touchscreen tapping on the text or swiping left and right

You can access the viewer controls by either:

- Right clicking on the text
- Pressing the Esc or Menu keys
- On a touchscreen by tapping the top 1/3rd of the screen

The viewer has two modes, “paged” and “flow”. In paged mode the book content is presented as pages, similar to a paper book. In flow mode the text is presented continuously, like in a web browser. You can switch between them using the viewer *Preferences* under *Page layout* or by pressing the Ctrl+M key.

Bookmarks

When you are in the middle of a book and close the E-book viewer, it will remember where you stopped reading and return there the next time you open the book. You can also set bookmarks in the book by using the *Bookmarks* button in the E-book viewer controls or pressing Ctrl+B. When viewing EPUB format books, these bookmarks are actually saved in the EPUB file itself. You can add bookmarks, then send the file to a friend. When they open the file, they will be able to see your bookmarks. You can turn off this behavior in the *Miscellaneous* section of the viewer preferences.

Table of Contents

If the book you are reading defines a Table of Contents, you can access it by pressing the *Table of Contents* button. This will bring up a list of sections in the book. You can click on any of them to jump to that portion of the book.

Navigating by location

E-books, unlike paper books, have no concept of pages. You can refer to precise locations in e-books using the *Go to → Location* functionality in the viewer controls.

You can use this location information to unambiguously refer to parts of the books when discussing it with friends or referring to it in other works. You can enter these locations under *Go to → Location* in the viewer controls.

There is a URL you can copy to the clipboard and paste into other programs or documents. Clicking on this URL will open the book in the calibre E-book viewer at the current location.

If you click on links inside the e-book to take you to different parts of the book, such as an endnote, you can use the *Back* and *Forward* buttons in the top left corner of the viewer controls. These buttons behave just like those in a web browser.

Reference mode

calibre also has a very handy *Reference mode*. You can turn it on by clicking the *Reference mode* button in the viewer controls. Once you do this, every paragraph will have a unique number displayed at the start, made up of the section and paragraph numbers.

You can use this number to unambiguously refer to parts of the books when discussing it with friends or referring to it in other works. You can enter these numbers in the *Go to function* to navigate to a particular reference location.

Highlighting text

When you select text in the viewer, a little popup bar appears next to the selection. You can click the highlight button in that bar to create a highlight. You can add notes and change the color of the highlight. On a touch screen, long tap a word to select it and show the popup bar. Once in highlight mode you can change what text is selected, using touch screen friendly selection handles. Drag the handles to the top or bottom margins to scroll while selecting. You can also Shift+click or right click to extend the selection, particularly useful for multi-page selections.

You can use the *Highlights* button in the viewer controls to show a separate panel with a list of all highlights in the book, sorted by chapter.

You can browse *all highlights* in your entire calibre library by right clicking the *View* button and choosing *Browse annotations*.

Finally, if you use the calibre Content server's in browser viewer, you can have the viewer sync its annotations with the browser viewer by going to *Preferences* → *Miscellaneous* in the viewer preferences and entering the username of the Content server viewer to sync with.

Use the special value * to sync with anonymous users.

Read aloud

The viewer can read book text aloud. To use it you can simply click the *Read aloud* button in the viewer controls to start reading book text aloud. The word being currently read is highlighted. Speech is synthesized from the text using your operating system services for text-to-speech. You can change the voice being used by clicking the gear icon in the bar that is displayed while *Read aloud* is active.

You can also read aloud highlighted passages by adding the *Read aloud* button to the selection bar in the viewer preferences under *Selection behavior*.

Note

Support for text-to-speech in browsers is very incomplete and bug-ridden so how well *Read aloud* will work in the in-browser viewer is dependent on how well the underlying browser

supports text-to-speech. In particular, highlighting of current word does not work, and changing speed or voice will cause reading to start again from the beginning.

Note

On Linux, *Read aloud* requires [Speech Dispatcher](https://freebsoft.org/speechd) [https://freebsoft.org/speechd] to be installed and working.

Note

On Windows, not all installed voices may be visible to the SAPI sub-system that is used for text-to-speech. There are [instructions to make all voices visible](https://www.mobileread.com/forums/showpost.php?p=4084051&postcount=108) [https://www.mobileread.com/forums/showpost.php?p=4084051&postcount=108].

Following links using only the keyboard

The E-book viewer has a *Hints mode* that allows you to click links in the text without using the mouse. Press the Alt+F key and all links in the current screen will be highlighted with a number or letter over them. Press the letter on your keyboard to click the link. Pressing the Esc key will abort the *Hints mode* without selecting any link.

If more than thirty five links are on-screen then some of them will have multiple letters, in which case type the first and second, or the first and press Enter to activate. You can also use the Backspace key to undo a mistake in typing.

Customizing the look and feel of your reading experience

You can change font sizes on the fly by using *Font size* in the viewer controls or Ctrl++ or Ctrl+- or holding the Ctrl key and using the mouse wheel.

Colors can be changed in the *Colors* section of the viewer preferences.

You can change the number of pages displayed on the screen as well as page margins in *Page layout* in the viewer preferences.

You can display custom headers and footers such as time left to read, current chapter title, book position, etc. via the *Headers and footers* section of the viewer preferences.

More advanced customization can be achieved by the *Styles* settings. Here you can specify a background image to display under the text and also a stylesheet you can set that will be applied to every book. Using it you can do things like change paragraph styles, text justification, etc. For examples of custom stylesheets used by calibre's users, see [the forums](https://www.mobileread.com/forums/showthread.php?t=51500) [https://www.mobileread.com/forums/showthread.php?t=51500].

Dictionary lookup

You can look up the meaning of words in the current book by double clicking or long tapping the word you want to lookup and then clicking the lookup button that looks like a library.

Copying text and images

You can select text and images by dragging the content with your mouse and then right clicking and selecting *Copy* to copy to the clipboard. The copied material can be pasted into another application as plain text and images.

Zooming in on images

You can zoom in to show an image at full size in a separate window by either double clicking or long tapping on it. You can also right click on it and choose *View image*.

Non re-flowable content

Some books have very wide content that cannot be broken up at page boundaries. For example tables or `<pre>` tags. In such cases, you should switch the viewer to *flow mode* by pressing Ctrl+M to read this content. Alternately, you can also add the following CSS to the *Styles* section of the viewer preferences to force the viewer to break up lines of text in `<pre>` tags:

```
code
,
pre
{
white
-
space
:
pre
-
wrap
}
```


Designing your book to work well with the calibre viewer

The calibre viewer will set the is-calibre-viewer class on the root element. So you can write CSS rules that apply only for it. Additionally, the viewer will set the following classes on the body element:

body.calibre-viewer-dark-colors

Set when using a dark color scheme

body.calibre-viewer-light-colors

Set when using a light color scheme

body.calibre-viewer-paginated

Set when in paged mode

body.calibre-viewer-scrolling

Set when in flow (non-paginated) mode

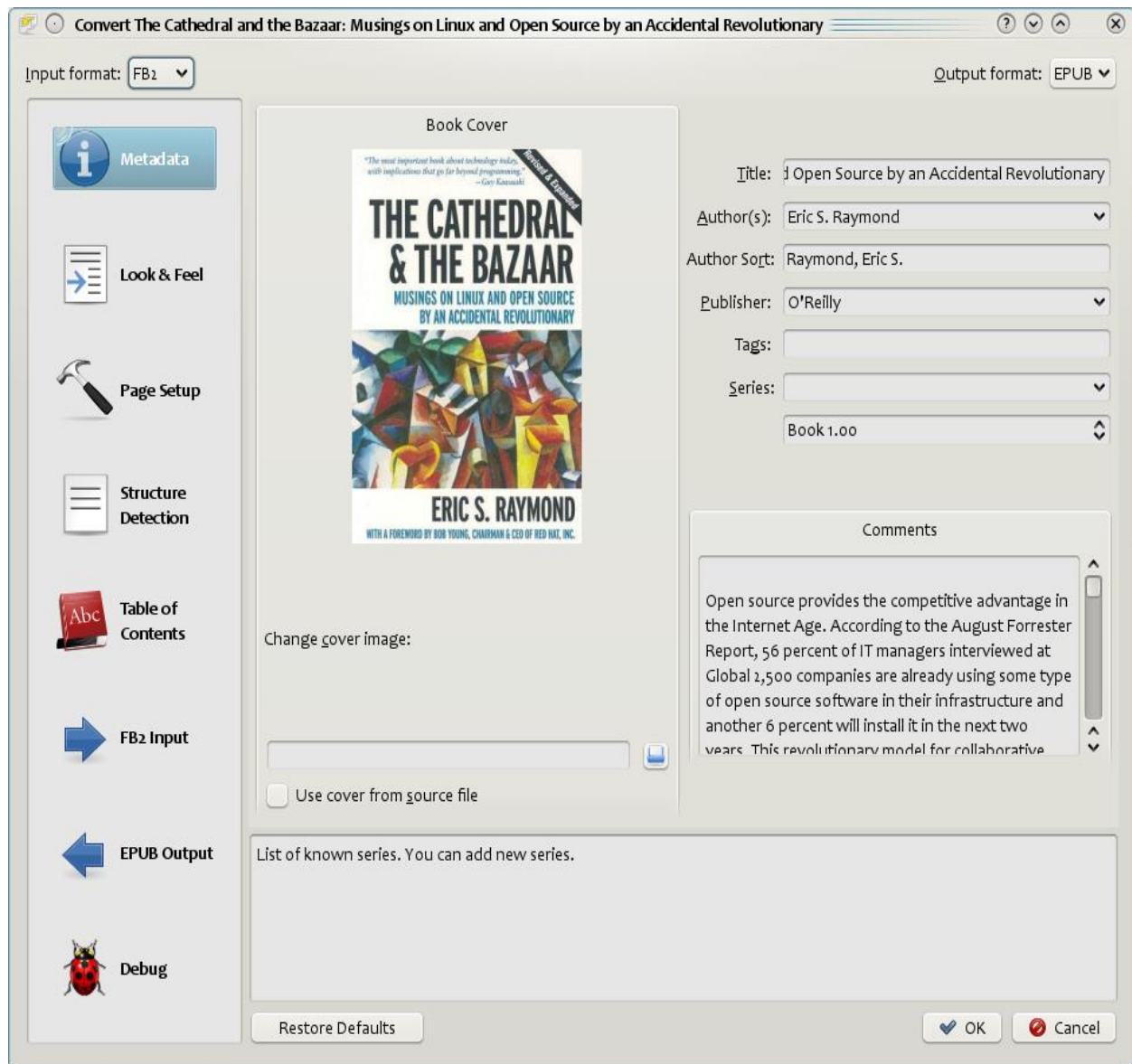
Finally, you can use the calibre color scheme colors via [CSS variables](https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_custom_properties) [https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_custom_properties]. The calibre viewer defines the following variables: --calibre-viewer-background-color, --calibre-viewer-foreground-color, and optionally --calibre-viewer-link-color in color themes that define a link color.



E-book conversion

calibre has a conversion system that is designed to be very easy to use. Normally, you just add a book to calibre, click convert and calibre will try hard to generate output that is as close as possible to the input. However, calibre accepts a very large number of input formats, not all of which are as suitable as others for conversion to e-books. In the case of such input formats, or if you just want greater control over the conversion system, calibre has a lot of options to fine tune the conversion process. Note however that calibre's conversion system is not a substitute for a full blown e-book editor. To edit e-books, I recommend first converting them to EPUB or AZW3 using calibre and then using the *Edit book* feature to get them into perfect shape. You can then use the edited e-book as input for conversion into other formats in calibre.

This document will refer mainly to the conversion settings as found in the conversion dialog, pictured below. All these settings are also available via command line interface to conversion, documented at [ebook-convert](#). In calibre, you can obtain help on any individual setting by holding your mouse over it, a tooltip will appear describing the setting.



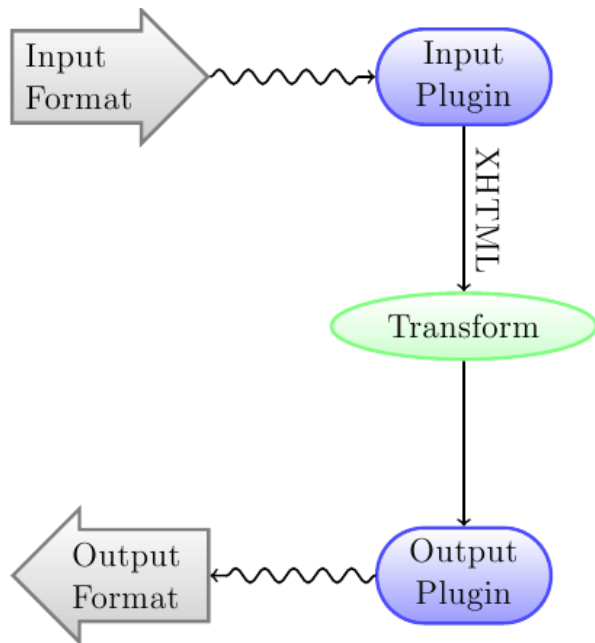
Contents

- [Introduction](#)
- [Look & feel](#)
- [Page setup](#)
- [Heuristic processing](#)
- [Search & replace](#)
- [Structure detection](#)

- [Table of Contents](#)
- [Using images as chapter titles when converting HTML input documents](#)
- [Using tag attributes to supply the text for entries in the Table of Contents](#)
- [How options are set/saved for conversion](#)
- [Format specific tips](#)

Introduction

The first thing to understand about the conversion system is that it is designed as a pipeline. Schematically, it looks like this:



The input format is first converted to XHTML by the appropriate *Input plugin*. This HTML is then *transformed*. In the last step, the processed XHTML is converted to the specified output format by the appropriate *Output plugin*. The results of the conversion can vary greatly, based on the input format. Some formats convert much better than others. A list of the best source formats for conversion is available [here](#).

The transforms that act on the XHTML output are where all the work happens. There are various transforms, for example, to insert book metadata as a page at the start of the book, to detect chapter headings and automatically create a Table of Contents, to proportionally adjust font sizes, et cetera. It is important to remember that all the transforms act on the XHTML output by the *Input plugin*, not on the input file itself. So, for example, if you ask

calibre to convert an RTF file to EPUB, it will first be converted to XHTML internally, the various transforms will be applied to the XHTML and then the *Output plugin* will create the EPUB file, automatically generating all metadata, Table of Contents, et cetera.



You can see this process in action by using the debug option . Just specify the path to a folder for the debug output. During conversion, calibre will place the XHTML generated by the various stages of the conversion pipeline in different sub-folders. The four sub-folders are:

Stages of the conversion pipeline

Folder	Description
input	This contains the HTML output by the Input plugin. Use this to debug the Input plugin.
parsed	The result of pre-processing and converting to XHTML the output from the Input plugin. Use to debug structure detection.
structure	Post structure detection, but before CSS flattening and font size conversion. Use to debug font size conversion and CSS transforms.
processed	Just before the e-book is passed to the Output plugin. Use to debug the Output plugin.

If you want to edit the input document a little before having calibre convert it, the best thing to do is edit the files in the input sub-folder, then zip it up, and use the ZIP file as the input format for subsequent conversions. To do this use the *Edit meta information* dialog to add the ZIP file as a format for the book and then, in the top left corner of the conversion dialog, select ZIP as the input format.

This document will deal mainly with the various transforms that operate on the intermediate XHTML and how to control them. At the end are some tips specific to each input/output format.

[Look & feel](#)

Contents

- [Fonts](#)
- [Text](#)
- [Layout](#)
- [Styling](#)
- [Transform styles](#)
- [Transform HTML](#)

This group of options controls various aspects of the look and feel of the converted e-book.

[Fonts](#)

One of the nicest features of the e-reading experience is the ability to easily adjust font sizes to suit individual needs and lighting conditions. calibre has sophisticated algorithms to ensure that all the books it outputs have a consistent font sizes, no matter what font sizes are specified in the input document.

The base font size of a document is the most common font size in that document, i.e., the size of the bulk of text in that document. When you specify a *Base font size*, calibre automatically rescales all font sizes in the document proportionately, so that the most common font size becomes the specified base font size and other font sizes are rescaled appropriately. By choosing a larger base font size, you can make the fonts in the document larger and vice versa. When you set the base font size, for best results, you should also set the font size key.

Normally, calibre will automatically choose a base font size appropriate to the output profile you have chosen (see [Page setup](#)). However, you can override this here in case the default is not suitable for you.

The *Font size key* option lets you control how non-base font sizes are rescaled. The font rescaling algorithm works using a font size key, which is simply a comma-separated list of font sizes. The font size key tells calibre how many “steps” bigger or smaller a given font size should be compared to the base font size. The idea is that there should be a limited number of font sizes in a document. For example, one size for the body text, a couple of sizes for different levels of headings and a couple of sizes for super/sub scripts and footnotes. The font size key allows calibre to compartmentalize the font sizes in the input documents into separate “bins” corresponding to the different logical font sizes.

Let’s illustrate with an example. Suppose the source document we are converting was produced by someone with excellent eyesight and has a base font size of 8pt. That means the

bulk of the text in the document is sized at 8pts, while headings are somewhat larger (say 10 and 12pt) and footnotes somewhat smaller at 6pt. Now if we use the following settings:

Base

font

size

:

12

pt

Font

size

key

:

7

,

8

,

10

,

12

,

14

,

16

,

18

,

20

The output document will have a base font size of 12pt, headings of 14 and 16pt and footnotes of 8pt. Now suppose we want to make the largest heading size stand out more and make the footnotes a little larger as well. To achieve this, the font key should be changed to:

New

font

size

key

:

7

,

9

,

12

,

14

,

18

,

20

,

22

The largest headings will now become 18pt, while the footnotes will become 9pt. You can play with these settings to try and figure out what would be optimum for you by using the font rescaling wizard, which can be accessed by clicking the little button next to the *Font size key* setting.

All the font size rescaling in the conversion can also be disabled here, if you would like to preserve the font sizes in the input document.

A related setting is *Line height*. Line height controls the vertical height of lines. By default, (a line height of 0), no manipulation of line heights is performed. If you specify a non-default value, line heights will be set in all locations that don't specify their own line heights. However, this is something of a blunt weapon and should be used sparingly. If you want to adjust the line heights for some section of the input, it's better to use the [Extra CSS](#).

In this section you can also tell calibre to embed any referenced fonts into the book. This will allow the fonts to work on reader devices even if they are not available on the device.

Text

Text can be either justified or not. Justified text has extra spaces between words to give a smooth right margin. Some people prefer justified text, others do not. Normally, calibre will preserve the justification in the original document. If you want to override it you can use the *Text justification* option in this section.

You can also tell calibre to *Smarten punctuation* which will replace plain quotes, dashes and ellipses with their typographically correct alternatives. Note that this algorithm is not perfect so it is worth reviewing the results. The reverse, namely, *Unsmarten punctuation* is also available.

Finally, there is *Input character encoding*. Older documents sometimes don't specify their character encoding. When converted, this can result in non-English characters or special characters like smart quotes being corrupted. calibre tries to auto-detect the character encoding of the source document, but it does not always succeed. You can force it to assume a particular character encoding by using this setting. *cp1252* is a common encoding for documents produced using Windows software. You should also read [How do I convert my file containing non-English characters, or smart quotes?](#) for more on encoding issues.

Layout

Normally, paragraphs in XHTML are rendered with a blank line between them and no leading text indent. calibre has a couple of options to control this. *Remove spacing between paragraphs* forcefully ensure that all paragraphs have no inter paragraph spacing. It also sets the text indent to 1.5em (can be changed) to mark the start of every paragraph. *Insert blank line* does the opposite, guaranteeing that there is exactly one blank line between each pair of paragraphs. Both these options are very comprehensive, removing spacing, or inserting it for *all* paragraphs (technically <p> and <div> tags). This is so that you can just set the option and be sure that it performs as advertised, irrespective of how messy the input file is. The one exception is when the input file uses hard line breaks to implement inter-paragraph spacing.

If you want to remove the spacing between all paragraphs, except a select few, don't use these options. Instead add the following CSS code to [Extra CSS](#):

```
p
,
div
{
margin
:
0
pt
;

border
:
0
pt
;

text
-
indent
:
1.5
em
}

.
spacious
{
margin
-
bottom
:
```

```

1
em
;

text
-
indent
:

0
pt
;

}

```

Then, in your source document, mark the paragraphs that need spacing with *class="spacious"*. If your input document is not in HTML, use the Debug option, described in the Introduction to get HTML (use the input sub-folder).

Another useful options is *Linearize tables*. Some badly designed documents use tables to control the layout of text on the page. When converted these documents often have text that runs off the page and other artifacts. This option will extract the content from the tables and present it in a linear fashion. Note that this option linearizes *all* tables, so only use it if you are sure the input document does not use tables for legitimate purposes, like presenting tabular information.

[Styling](#)

The *Extra CSS* option allows you to specify arbitrary CSS that will be applied to all HTML files in the input. This CSS is applied with very high priority and so should override most CSS present in the **input document** itself. You can use this setting to fine tune the presentation/layout of your document. For example, if you want all paragraphs of class *endnote* to be right aligned, just add:

```

.
endnote

{

text

```

```
-  
align  
:  
  
right  
  
}
```

or if you want to change the indentation of all paragraphs:

```
p  
  
{  
  
text  
-  
indent  
:  
  
5  
mm  
;  
  
}
```

Extra CSS is a very powerful option, but you do need an understanding of how CSS works to use it to its full potential. You can use the debug pipeline option described above to see what CSS is present in your input document.

A simpler option is to use *Filter style information*. This allows you to remove all CSS properties of the specified types from the document. For example, you can use it to remove all colors or fonts.

[Transform styles](#)

This is the most powerful styling related facility. You can use it to define rules that change styles based on various conditions. For example you can use it to change all green colors to blue, or remove all bold styling from the text or color all headings a certain color, etc.

[Transform HTML](#)

Similar to transform styles, but allows you to make changes to the HTML content of the book. You can replace one tag with another, add classes or other attributes to tags based on their content, etc.

Page setup

The *Page setup* options are for controlling screen layout, like margins and screen sizes. There are options to setup page margins, which will be used by the output plugin, if the selected output format supports page margins. In addition, you should choose an Input profile and an output profile. Both sets of profiles basically deal with how to interpret measurements in the input/output documents, screen sizes and default font rescaling keys.

If you know that the file you are converting was intended to be used on a particular device/software platform, choose the corresponding input profile, otherwise just choose the default input profile. If you know the files you are producing are meant for a particular device type, choose the corresponding output profile. Otherwise, choose one of the Generic output profiles. If you are converting to MOBI or AZW3 then you will almost always want to choose one of the Kindle output profiles. Otherwise, your best bet for modern E-book reading devices is to choose the *Generic e-ink HD* output profile.

The output profile also controls the screen size. This will cause, for example, images to be auto-resized to be fit to the screen in some output formats. So choose a profile of a device that has a screen size similar to your device.

Heuristic processing

Heuristic processing provides a variety of functions which can be used to try and detect and correct common problems in poorly formatted input documents. Use these functions if your input document suffers from poor formatting. Because these functions rely on common patterns, be aware that in some cases an option may lead to worse results, so use with care. As an example, several of these options will remove all non-breaking-space entities, or may include false positive matches relating to the function.

Enable heuristic processing

This option activates calibre's *Heuristic processing* stage of the conversion pipeline. This must be enabled in order for various sub-functions to be applied

Unwrap lines

Enabling this option will cause calibre to attempt to detect and correct hard line breaks that exist within a document using punctuation clues and line length. calibre will first attempt to

detect whether hard line breaks exist, if they do not appear to exist calibre will not attempt to unwrap lines. The line-unwrap factor can be reduced if you want to 'force' calibre to unwrap lines.

Line-unwrap factor

This option controls the algorithm calibre uses to remove hard line breaks. For example, if the value of this option is 0.4, that means calibre will remove hard line breaks from the end of lines whose lengths are less than the length of 40% of all lines in the document. If your document only has a few line breaks which need correction, then this value should be reduced to somewhere between 0.1 and 0.2.

Detect and markup unformatted chapter headings and sub headings

If your document does not have chapter headings and titles formatted differently from the rest of the text, calibre can use this option to attempt to detect them and surround them with heading tags. <h2> tags are used for chapter headings; <h3> tags are used for any titles that are detected.

This function will not create a TOC, but in many cases it will cause calibre's default chapter detection settings to correctly detect chapters and build a TOC. Adjust the XPath under Structure detection if a TOC is not automatically created. If there are no other headings used in the document then setting "//h:h2" under Structure detection would be the easiest way to create a TOC for the document.

The inserted headings are not formatted, to apply formatting use the *Extra CSS* option under the Look and Feel conversion settings. For example, to center heading tags, use the following:

```
h2
,
h3
{
text-align: center
}
```

Renumber sequences of <h1> or <h2> tags

Some publishers format chapter headings using multiple <h1> or <h2> tags sequentially. calibre's default conversion settings will cause such titles to be split into two pieces. This option will re-number the heading tags to prevent splitting.

Delete blank lines between paragraphs

This option will cause calibre to analyze blank lines included within the document. If every paragraph is interleaved with a blank line, then calibre will remove all those blank paragraphs. Sequences of multiple blank lines will be considered scene breaks and retained as a single paragraph. This option differs from the *Remove paragraph spacing* option under *Look and Feel* in that it actually modifies the HTML content, while the other option modifies the document styles. This option can also remove paragraphs which were inserted using calibre's *Insert blank line* option.

Ensure scene breaks are consistently formatted

With this option calibre will attempt to detect common scene-break markers and ensure that they are center aligned. 'Soft' scene break markers, i.e. scene breaks only defined by extra white space, are styled to ensure that they will not be displayed in conjunction with page breaks.

Replace scene breaks

If this option is configured then calibre will replace scene break markers it finds with the replacement text specified by the user. Please note that some ornamental characters may not be supported across all reading devices.

In general you should avoid using HTML tags, calibre will discard any tags and use pre-defined markup. <hr /> tags, i.e. horizontal rules, and tags are exceptions. Horizontal rules can optionally be specified with styles, if you choose to add your own style be sure to include the 'width' setting, otherwise the style information will be discarded. Image tags can be used, but calibre does not provide the ability to add the image during conversion, this must be done after the fact using the 'Edit book' feature.

Example image tag (place the image within an 'Images' folder inside the EPUB after conversion):

```

```

Example horizontal rule with styles:

```
<hr style="width:20%;padding-top: 1px;border-top: 2px ridge black;border-bottom: 2px groove black;" />
```

Remove unnecessary hyphens

calibre will analyze all hyphenated content in the document when this option is enabled. The document itself is used as a dictionary for analysis. This allows calibre to accurately remove hyphens for any words in the document in any language, along with made-up and obscure scientific words. The primary drawback is words appearing only a single time in the document will not be changed. Analysis happens in two passes, the first pass analyzes line endings. Lines are only unwrapped if the word exists with or without a hyphen in the document. The second pass analyzes all hyphenated words throughout the document, hyphens are removed if the word exists elsewhere in the document without a match.

Italicize common words and patterns

When enabled, calibre will look for common words and patterns that denote italics and italicize them. Examples are common text conventions such as ~word~ or phrases that should generally be italicized, e.g. latin phrases like 'etc.' or 'et cetera'.

Replace entity indents with CSS indents

Some documents use a convention of defining text indents using non-breaking space entities. When this option is enabled calibre will attempt to detect this sort of formatting and convert them to a 3% text indent using CSS.

Search & replace

These options are useful primarily for conversion of PDF documents or OCR conversions, though they can also be used to fix many document specific problems. As an example, some conversions can leave behind page headers and footers in the text. These options use regular expressions to try and detect headers, footers, or other arbitrary text and remove or replace them. Remember that they operate on the intermediate XHTML produced by the conversion pipeline. There is a wizard to help you customize the regular expressions for your document. Click the magic wand beside the expression box, and click the 'Test' button after composing your search expression. Successful matches will be highlighted in Yellow.

The search works by using a Python regular expression. All matched text is simply removed from the document or replaced using the replacement pattern. The replacement pattern is optional, if left blank then text matching the search pattern will be deleted from the document. You can learn more about regular expressions and their syntax at [All about using regular expressions in calibre](#).

Structure detection

Structure detection involves calibre trying its best to detect structural elements in the input document, when they are not properly specified. For example, chapters, page breaks,

headers, footers, etc. As you can imagine, this process varies widely from book to book. Fortunately, calibre has very powerful options to control this. With power comes complexity, but if once you take the time to learn the complexity, you will find it well worth the effort.

Chapters and page breaks

calibre has two sets of options for *chapter detection* and *inserting page breaks*. This can sometimes be slightly confusing, as by default, calibre will insert page breaks before detected chapters as well as the locations detected by the page breaks option. The reason for this is that there are often locations where page breaks should be inserted that are not chapter boundaries. Also, detected chapters can be optionally inserted into the auto generated Table of Contents.

calibre uses *XPath*, a powerful language to allow the user to specify chapter boundaries/page breaks. XPath can seem a little daunting to use at first, fortunately, there is a [XPath tutorial](#) in the User Manual. Remember that Structure detection operates on the intermediate XHTML produced by the conversion pipeline. Use the debug option described in the [Introduction](#) to figure out the appropriate settings for your book. There is also a button for a XPath wizard to help with the generation of simple XPath expressions.

By default, calibre uses the following expression for detecting chapters:

```
//*  
[(  
  name  
  )  
  =  
  'h1'  
  
or  
  
  name  
  )  
  =  
  'h2'  
)  
  
and  
  
re  
:  
test  
(  
.
```

```
,
'chapter|book|section|part\s+'
,
'i'
))

or

@class

=

'chapter'
]
```

This expression is rather complex, because it tries to handle a number of common cases simultaneously. What it means is that calibre will assume chapters start at either `<h1>` or `<h2>` tags that have any of the words (*chapter, book, section or part*) in them or that have the `class="chapter"` attribute.

A related option is *Chapter mark*, which allows you to control what calibre does when it detects a chapter. By default, it will insert a page break before the chapter. You can have it insert a ruled line instead of, or in addition to the page break. You can also have it do nothing.

The default setting for detecting page breaks is:

```
/**
[
name
()
=
'h1'

or

name
()
=
'h2'
]
```

which means that calibre will insert page breaks before every `<h1>` and `<h2>` tag by default.

Note

The default expressions may change depending on the input format you are converting.

Miscellaneous

There are a few more options in this section.

Insert metadata as page at start of book

One of the great things about calibre is that it allows you to maintain very complete metadata about all of your books, for example, a rating, tags, comments, etc. This option will create a single page with all this metadata and insert it into the converted e-book, typically just after the cover. Think of it as a way to create your own customised book jacket.

Remove first image

Sometimes, the source document you are converting includes the cover as part of the book, instead of as a separate cover. If you also specify a cover in calibre, then the converted book will have two covers. This option will simply remove the first image from the source document, thereby ensuring that the converted book has only one cover, the one specified in calibre.

Table of Contents

When the input document has a Table of Contents in its metadata, calibre will just use that. However, a number of older formats either do not support a metadata based Table of Contents, or individual documents do not have one. In these cases, the options in this section can help you automatically generate a Table of Contents in the converted e-book, based on the actual content in the input document.

Note

Using these options can be a little challenging to get exactly right. If you prefer creating/editing the Table of Contents by hand, convert to the EPUB or AZW3 formats and select the checkbox at the bottom of the Table of Contents section of the conversion dialog that says *Manually fine-tune the Table of Contents after conversion*. This will launch the ToC Editor tool after the conversion. It allows you to create entries in the Table of Contents by simply clicking the place in the book where you want the entry to point. You can also use the ToC Editor by itself, without doing a conversion. Go to *Preferences → Interface → Toolbars*

and add the *ToC Editor* to the main toolbar. Then just select the book you want to edit and click the *ToC Editor* button.

The first option is *Force use of auto-generated Table of Contents*. By checking this option you can have calibre override any Table of Contents found in the metadata of the input document with the auto generated one.

The default way that the creation of the auto generated Table of Contents works is that, calibre will first try to add any detected chapters to the generated table of contents. You can learn how to customize the detection of chapters in the [Structure detection](#) section above. If you do not want to include detected chapters in the generated table of contents, check the *Do not add detected chapters* option.

If less than the *Chapter threshold* number of chapters were detected, calibre will then add any hyperlinks it finds in the input document to the Table of Contents. This often works well: many input documents include a hyperlinked Table of Contents right at the start. The *Number of links* option can be used to control this behavior. If set to zero, no links are added. If set to a number greater than zero, at most that number of links is added.

calibre will automatically filter duplicates from the generated Table of Contents. However, if there are some additional undesirable entries, you can filter them using the *TOC Filter* option. This is a regular expression that will match the title of entries in the generated table of contents. Whenever a match is found, it will be removed. For example, to remove all entries titles “Next” or “Previous” use:

```
Next
|
Previous
```

The *Level 1,2,3 TOC* options allow you to create a sophisticated multi-level Table of Contents. They are XPath expressions that match tags in the intermediate XHTML produced by the conversion pipeline. See the [Introduction](#) for how to get access to this XHTML. Also read the [XPath tutorial](#), to learn how to construct XPath expressions. Next to each option is a button that launches a wizard to help with the creation of basic XPath expressions. The following simple example illustrates how to use these options.

Suppose you have an input document that results in XHTML that look like this:

```
<
html

xmlns
=
```

```
"http://www.w3.org/1999/xhtml"
```

```
>
```

```
<
```

```
head
```

```
>
```

```
<
```

```
title
```

```
>
```

```
Sample document
```

```
</
```

```
title
```

```
>
```

```
</
```

```
head
```

```
>
```

```
<
```

```
body
```

```
>
```

```
<
```

```
h1
```

```
>
```

```
Chapter 1
```

```
</
```

```
h1
```

```
>
```

```
...
```

```
<
```

```
h2
```

```
>
```

```
Section 1.1
```

```
</
```

```
h2
>
...

<
h2
>
Section 1.2
</
h2
>
...

<
h1
>
Chapter 2
</
h1
>
...

<
h2
>
Section 2.1
</
h2
>
...

</
body
>

</
html
>
```

Then, we set the options as:

Level

1

TOC

:

//

h

:

h1

Level

2

TOC

:

//

h

:

h2

This will result in an automatically generated two level Table of Contents that looks like:

Chapter

1

Section

1.1

Section

1.2

Chapter

2

Section

2.1

Warning

Not all output formats support a multi level Table of Contents. You should first try with EPUB output. If that works, then try your format of choice.

Using images as chapter titles when converting HTML input documents

Suppose you want to use an image as your chapter title, but still want calibre to be able to automatically generate a Table of Contents for you from the chapter titles. Use the following HTML markup to achieve this:

```
<
html
>

<
body
>

<
h2
>
Chapter 1
```

```
</  
h2  
>  
  
<  
p  
>  
chapter 1 text...  
</  
p  
>  
  
<  
h2  
  
title  
=  
"Chapter 2"  
><  
img  
  
src  
=  
"chapter2.jpg"  
  
></  
h2  
>  
  
<  
p  
>  
chapter 2 text...  
</  
p  
>  
  
</  
body
```

```
>

</
html
>
```

Set the *Level 1 TOC* setting to //h:h2 . Then, for chapter two, calibre will take the title from the value of the title attribute on the <h2> tag, since the tag has no text.

Using tag attributes to supply the text for entries in the Table of Contents

If you have particularly long chapter titles and want shortened versions in the Table of Contents, you can use the title attribute to achieve this, for example:

```
<
html
>

<
body
>

<
h2
title
=
"Chapter 1"
>
Chapter 1: Some very long title
</
h2
>

<
```

```


>
chapter 1 text...
</p>
<h2>
title
=
"Chapter 2"
>
Chapter 2: Some other very long title
</h2>
<
<p>
>
chapter 2 text...
</p>
</body>
</html>


```

Set the *Level 1 TOC* setting to //h:h2/@title . Then calibre will take the title from the value of the title attribute on the <h2>

tags, instead of using the text inside the tag. Note the trailing `/@title` on the XPath expression, you can use this form to tell calibre to get the text from any attribute you like.

How options are set/saved for conversion

There are two places where conversion options can be set in calibre. The first is in Preferences->Conversion. These settings are the defaults for the conversion options. Whenever you try to convert a new book, the settings set here will be used by default.

You can also change settings in the conversion dialog for each book conversion. When you convert a book, calibre remembers the settings you used for that book, so that if you convert it again, the saved settings for the individual book will take precedence over the defaults set in *Preferences*. You can restore the individual settings to defaults by using the *Restore defaults* button in the individual book conversion dialog. You can remove the saved settings for a group of books by selecting all the books and then clicking the *Edit metadata* button to bring up the bulk metadata edit dialog, near the bottom of the dialog is an option to remove stored conversion settings.

When you bulk convert a set of books, settings are taken in the following order (last one wins):

- From the defaults set in Preferences->Conversion
- From the saved conversion settings for each book being converted (if any). This can be turned off by the option in the top left corner of the Bulk conversion dialog.
- From the settings set in the Bulk conversion dialog

Note that the final settings for each book in a Bulk conversion will be saved and re-used if the book is converted again. Since the highest priority in Bulk Conversion is given to the settings in the Bulk conversion dialog, these will override any book specific settings. So you should only bulk convert books together that need similar settings. The exceptions are metadata and input format specific settings. Since the Bulk conversion dialog does not have settings for these two categories, they will be taken from book specific settings (if any) or the defaults.

Note

You can see the actual settings used during any conversion by clicking the rotating icon in the lower right corner and then double clicking the individual conversion job. This will bring up a conversion log that will contain the actual settings used, near the top.

Format specific tips

Here you will find tips specific to the conversion of particular formats. Options specific to particular format, whether input or output are available in the conversion dialog under their own section, for example *TXT input* or *EPUB output*.

Convert Microsoft Word documents

calibre can automatically convert .docx files created by Microsoft Word 2007 and newer. Just add the file to calibre and click convert.

Note

There is a [demo .docx file](https://calibre-ebook.com/downloads/demos/demo.docx) [https://calibre-ebook.com/downloads/demos/demo.docx] that demonstrates the capabilities of the calibre conversion engine. Just download it and convert it to EPUB or AZW3 to see what calibre can do.

calibre will automatically generate a Table of Contents based on headings if you mark your headings with the Heading

1
, Heading

2
, etc. styles in Microsoft Word. Open the output e-book in the calibre E-book viewer and click the *Table of Contents* button to view the generated Table of Contents.

Older .doc files

For older .doc files, you can save the document as HTML with Microsoft Word and then convert the resulting HTML file with calibre. When saving as HTML, be sure to use the “Save as Web Page, Filtered” option as this will produce clean HTML that will convert well. Note that Word produces really messy HTML, converting it can take a long time, so be patient. If you have a newer version of Word available, you can directly save it as .docx as well.

Another alternative is to use the free OpenOffice. Open your .doc file in OpenOffice and save it in OpenOffice’s format .odt. calibre can directly convert .odt files.

Convert TXT documents

TXT documents have no well defined way to specify formatting like bold, italics, etc, or document structure like paragraphs, headings, sections and so on, but there are a variety of conventions commonly used. By default calibre attempts automatic detection of the correct formatting and markup based on those conventions.

TXT input supports a number of options to differentiate how paragraphs are detected.

Paragraph style: Auto

Analyzes the text file and attempts to automatically determine how paragraphs are defined. This option will generally work fine, if you achieve undesirable results try one of the manual options.

Paragraph style: Block

Assumes one or more blank lines are a paragraph boundary:

This

is

the

first
.

This

is

the

second

paragraph
.

Paragraph style: Single

Assumes that every line is a paragraph:

This

is

the

first
.

This

is

the

second

.

This

is

the

third

.

Paragraph style: Print

Assumes that every paragraph starts with an indent (either a tab or 2+ spaces). Paragraphs end when the next line that starts with an indent is reached:

This

is

the

first

.

This

is

the

second

.

This

is

the

third
.

Paragraph style: Unformatted

Assumes that the document has no formatting, but does use hard line breaks. Punctuation and median line length are used to attempt to re-create paragraphs.

Formatting style: Auto

Attempts to detect the type of formatting markup being used. If no markup is used then heuristic formatting will be applied.

Formatting style: Heuristic

Analyzes the document for common chapter headings, scene breaks, and italicized words and applies the appropriate HTML markup during conversion.

Formatting style: Markdown

calibre also supports running TXT input through a transformation preprocessor known as Markdown. Markdown allows for basic formatting to be added to TXT documents, such as bold, italics, section headings, tables, lists, a Table of Contents, etc. Marking chapter headings with a leading # and setting the chapter XPath detection expression to “//h:h1” is the easiest way to have a proper table of contents generated from a TXT document. You can learn more about the Markdown syntax at [daringfireball](https://daringfireball.net/projects/markdown/syntax) [https://daringfireball.net/projects/markdown/syntax].

Formatting style: None

Applies no special formatting to the text, the document is converted to HTML with no other changes.

Convert PDF documents

PDF documents are one of the worst formats to convert from. They are a fixed page size and text placement format. Meaning, it is very difficult to determine where one paragraph ends and another begins. calibre will try to unwrap paragraphs using a configurable, *Line unwrapping factor*. This is a scale used to determine the length at which a line should be unwrapped. Valid values are a decimal between 0 and 1. The default is 0.45, just under the median line length. Lower this value to include more text in the unwrapping. Increase to include less. You can adjust this value in the conversion settings under *PDF Input*.

Also, they often have headers and footers as part of the document that will become included with the text. Use the *Search and replace* panel to remove headers and footers to mitigate this issue. If the headers and footers are not removed from the text it can throw off the paragraph unwrapping. To learn how to use the header and footer removal options, read [All about using regular expressions in calibre](#).

Some limitations of PDF input are:

- Complex, multi-column, and image based documents are not supported.
- Extraction of vector images and tables from within the document is also not supported.
- Some PDFs use special glyphs to represent ll or ff or fi, etc. Conversion of these may or may not work depending on just how they are represented internally in the PDF.
- Links and Tables of Contents are not supported
- PDFs that use embedded non-Unicode fonts to represent non-English characters will result in garbled output for those characters
- Some PDFs are made up of photographs of the page with OCR'd text behind them. In such cases calibre uses the OCR'd text, which can be very different from what you see when you view the PDF file
- PDFs that are used to display complex text, like right to left languages and math typesetting will not convert correctly

To re-iterate **PDF is a really, really bad** format to use as input. If you absolutely must use PDF, then be prepared for an output ranging anywhere from decent to unusable, depending on the input PDF.

Comic book collections

A comic book collection is a .cbc file. A .cbc file is a ZIP file that contains other CBZ/CBR files. In addition the .cbc file must contain a simple text file called comics.txt, encoded in UTF-8. The comics.txt file must contain a list of the comics files inside the .cbc file, in the form filename:title, as shown below:

```
one
.
cbz
:
Chapter

One

two
.
cbz
:
Chapter

Two

three
.
cbz
:
Chapter

Three
```

The .cbc file will then contain:

```
comics
.
txt

one
.
cbz

two
.
cbz
```

```
three
```

```
.
```

```
cbz
```

calibre will automatically convert this .cbc file into a e-book with a Table of Contents pointing to each entry in comics.txt.

EPUB advanced formatting demo

Various advanced formatting for EPUB files is demonstrated in this [demo file](https://calibre-ebook.com/downloads/demos/demo.epub) [https://calibre-ebook.com/downloads/demos/demo.epub]. The file was created from hand coded HTML using calibre and is meant to be used as a template for your own EPUB creation efforts.

The source HTML it was created from is available [demo.zip](https://calibre-ebook.com/downloads/demos/demo.zip) [https://calibre-ebook.com/downloads/demos/demo.zip]. The settings used to create the EPUB from the ZIP file are:

```
ebook
```

```
-
```

```
convert
```

```
demo
```

```
.
```

```
zip
```

```
.
```

```
epub
```

```
-
```

```
vv
```

```
--
```

```
authors
```

```
"Kovid Goyal"
```

```
--
```

```
language
```

```
en
```

```
--
```

```
level1
-
toc

'//*[ @class="title"]'

--
disable
-
font
-
rescaling

--
page
-
breaks
-
before

/

--
no
-
default
-
epub
-
cover
```

Note that because this file explores the potential of EPUB, most of the advanced formatting is not going to work on readers less capable than calibre's built-in EPUB viewer.

Convert ODT documents

calibre can directly convert ODT (OpenDocument Text) files. You should use styles to format your document and minimize the use of direct formatting. When inserting images into your document you need to anchor them to the paragraph, images anchored to a page will all end up in the front of the conversion.

To enable automatic detection of chapters, you need to mark them with the built-in styles called *Heading 1*, *Heading 2*, ..., *Heading 6* (*Heading 1* equates to the HTML tag <h1>, *Heading 2* to <h2>, etc). When you convert in calibre you can enter which style you used into the *Detect chapters at* box. Example:

- If you mark Chapters with style *Heading 2*, you have to set the 'Detect chapters at' box to //h:h2
- For a nested TOC with Sections marked with *Heading 2* and the Chapters marked with *Heading 3* you need to enter //h:h2|//h:h3. On the Convert - TOC page set the *Level 1 TOC* box to //h:h2 and the *Level 2 TOC* box to //h:h3.

Well-known document properties (Title, Keywords, Description, Creator) are recognized and calibre will use the first image (not too small, and with good aspect-ratio) as the cover image.

There is also an advanced property conversion mode, which is activated by setting the custom property opf.metadata ('Yes or No' type) to Yes in your ODT document (File->Properties->Custom Properties). If this property is detected by calibre, the following custom properties are recognized (opf.authors overrides document creator):

```
opf
.
titlesort

opf
.
authors

opf
.
authorsort

opf
.
publisher

opf
```

```
.  
pubdate  
  
opf  
  
.  
isbn  
  
opf  
  
.  
language  
  
opf  
  
.  
series  
  
opf  
  
.  
seriesindex
```

In addition to this, you can specify the picture to use as the cover by naming it `opf.cover` (right click, Picture->Options->Name) in the ODT. If no picture with this name is found, the 'smart' method is used. As the cover detection might result in double covers in certain output formats, the process will remove the paragraph (only if the only content is the cover!) from the document. But this works only with the named picture!

To disable cover detection you can set the custom property `opf.nocover` ('Yes or No' type) to Yes in advanced mode.

Converting to PDF

The first, most important, setting to decide on when converting to PDF is the page size. By default, calibre uses a page size of "U.S. Letter". You can change this to another standard page size or a completely custom size in the *PDF Output* section of the conversion dialog. If you are generating a PDF to be used on a specific device, you can turn on the option to use the page size from the *output profile* instead. So if your output profile is set to Kindle, calibre will create a PDF with page size suitable for viewing on the small Kindle screen.

Headers and Footers

You can insert arbitrary headers and footers on each page of the PDF by specifying header and footer templates. Templates are just snippets of HTML code that get rendered in the

header and footer locations. For example, to display page numbers centered at the bottom of every page, in green, use the following footer template:

```
<
footer
><
div

style
=
"margin: auto; color: green"
>
_PAGENUM_
</
div
></
footer
>
```

calibre will automatically replace `_PAGENUM_` with the current page number. You can even put different content on even and odd pages, for example the following header template will show the title on odd pages and the author on even pages:

```
<
header

style
=
"justify-content: flex-end"
>

<
div

class
=
"even-page"
>
_AUTHOR_
</
```



```

div
>

<
div

class
=
"odd-page"
><
i
>
_ TITLE _
</
i
></
div
>

</
header
>

```

calibre will automatically replace `_TITLE_` and `_AUTHOR_` with the title and author of the document being converted. Setting `justify-content` to `flex-end` will cause the text to be right aligned.

You can also display text at the left and right edges and change the font size, as demonstrated with this header template:

```

<
header

style
=
"justify-content: space-between; font-size: smaller"
>

```

```

<
div
>
  _TITLE_
</
div
>

<
div
>
  _AUTHOR_
</
div
>

</
header
>

```

This will display the title at the left and the author at the right, in a font size smaller than the main text.

You can also use the current section in templates, as shown below:

```

<
header
><
div
>
  _SECTION_
</
div
></
header
>

```

`_SECTION_` is replaced by whatever the name of the current section is. These names are taken from the

metadata Table of Contents in the document (the PDF Outline). If the document has no table of contents then it will be replaced by empty text. If a single PDF page has multiple sections, the first section on the page will be used. Similarly, there is a variable named `_TOP_LEVEL_SECTION_` that can be used to get the name of the current top-level section.

You can even use JavaScript inside the header and footer templates, for example, the following template will cause page numbers to start at 4 instead of 1:

```
<
footer
>

<
div
></
div
>

<
script
>
document
.
currentScript
.
parentNode
.
querySelector
(
"div"
)
.
innerHTML

=

""

+
```

```
(
_PAGENUM_

+

3
)
</
script
>

</
footer
>
```

In addition there are some more variables you can use in the headers and footers, documented below:

- `_TOTAL_PAGES_`
- total number of pages in the PDF file, useful for implementing a progress counter, for example.
- `_TOP_LEVEL_SECTION_PAGES_`
- total number of pages in the current top level section
- `_TOP_LEVEL_SECTION_PAGENUM_`
- the page number of the current page within the current top level section

Note

When adding headers and footers make sure you set the page top and bottom margins to large enough values, under the *PDF Output* section of the conversion dialog.

Printable Table of Contents

You can also insert a printable Table of Contents at the end of the PDF that lists the page numbers for every section. This is very useful if you intend to print out the PDF to paper. If you wish to use the PDF on an electronic device, then the PDF Outline provides this functionality and is generated by default.

You can customize the look of the generated Table of contents by using the Extra CSS conversion setting under the Look & feel part of the conversion dialog. The default CSS used is listed below, simply copy it and make whatever changes you like.

```
.  
calibre-pdf-toc
```

```
table
```

```
{
```

```
width
```

```
:
```

```
100
```

```
%%
```

```
}
```

```
.  
calibre-pdf-toc
```

```
table
```

```
tr
```

```
td
```

```
:
```

```
last-of-type
```

```
{
```

```
text-align
```

```
:
```

```
right
```

```
}
```

```
.  
calibre-pdf-toc
```

```
.  
level-0
```

```
{

font-size
:

larger
;

}

.
calibre-pdf-toc

.
level-1

td
:
first-of-type

{

padding-left
:

1.4
em

}

.
calibre-pdf-toc

.
level-2

td
:
first-of-type

{
```

```
padding-left
```

```
:
```

```
2.8
```

```
em
```

```
}
```

Custom page margins for individual HTML files

If you are converting an EPUB or AZW3 file with multiple individual HTML files inside it and you want to change the page margins for a particular HTML file you can add the following style block to the HTML file using the calibre E-book editor:

```
<
```

```
style
```

```
>
```

```
@
```

```
page
```

```
{
```

```
margin-left
```

```
:
```

```
10pt
```

```
;
```

```
margin-right
```

```
:
```

```
10pt
```

```
;
```

```
margin-top
```

```
:
```

```
10pt
```

```
;
```

```
margin-bottom
```

```
:
```

```
10pt
```

```
;
```

```
}
```

```
</
```

```
style
```

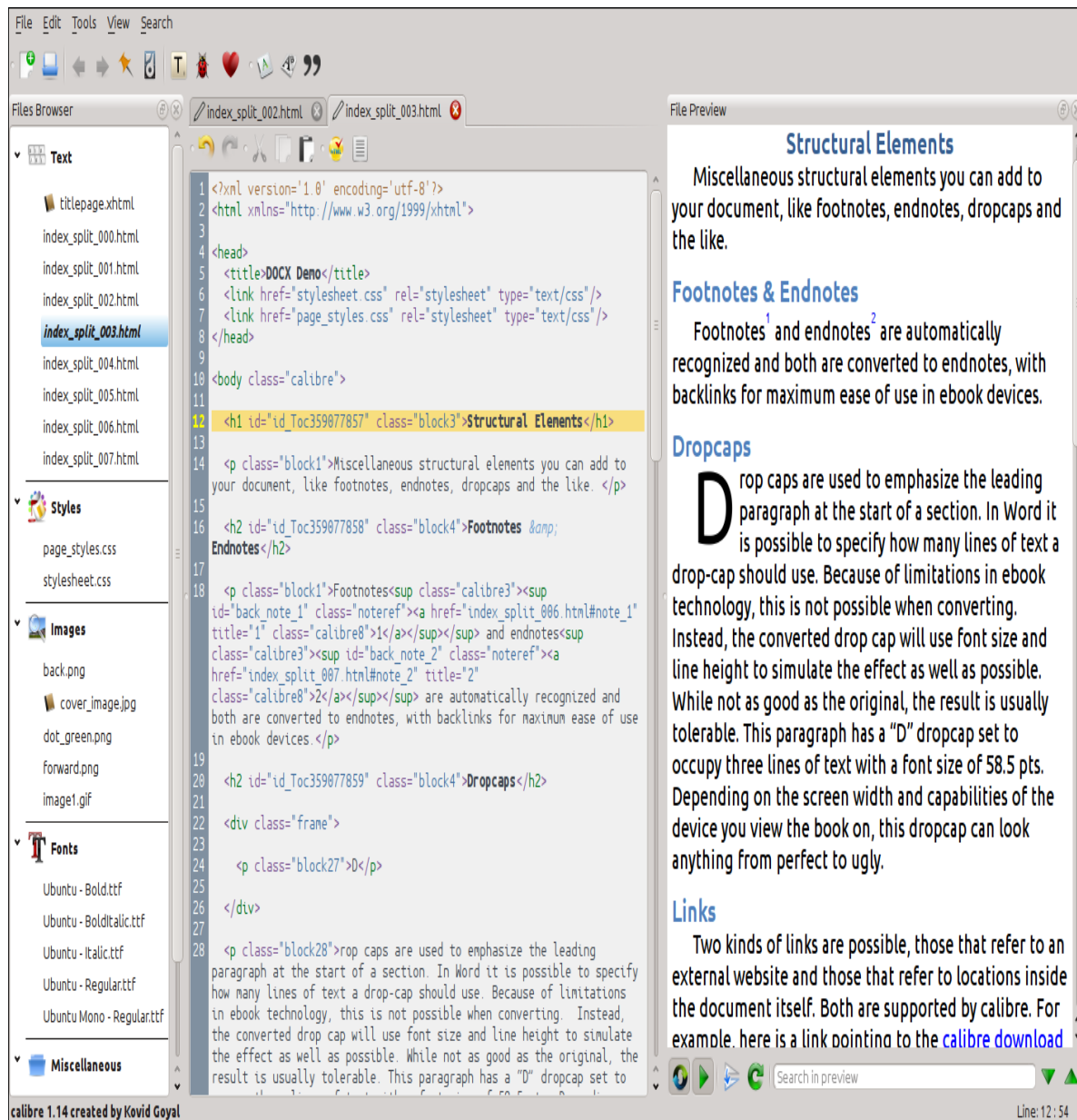
```
>
```

Then, in the PDF output section of the conversion dialog, turn on the option to *Use page margins from the document being converted*. Now all pages generated from this HTML file will have 10pt margins.

Editing e-books

calibre has an integrated e-book editor that can be used to edit books in the EPUB and AZW3 (Kindle) formats. The editor shows you the HTML and CSS that is used internally inside the book files, with a live preview that updates as you make changes. It also contains various automated tools to perform common cleanup and fixing tasks.

You can use this editor by right clicking on any book in calibre and selecting *Edit book*.



Contents

- [Basic workflow](#)
- [The File browser](#)
 - [Renaming files](#)
 - [Merging files](#)
 - [Changing text file order](#)
 - [Marking the cover](#)

- [Deleting files](#)
- [Exporting files](#)
- [Adding new images/fonts/etc. or creating new blank files](#)
- [Replacing files](#)
- [Linking stylesheets to HTML files efficiently](#)
- [Search & replace](#)
 - [Saved searches](#)
 - [Function mode](#)
 - [Search ignoring HTML tags](#)
- [Automated tools](#)
 - [Editing the Table of Contents](#)
 - [Checking the book](#)
 - [Adding a cover](#)
 - [Embedding referenced fonts](#)
 - [Subsetting embedded fonts](#)
 - [Smartening punctuation](#)
 - [Transforming CSS properties](#)
 - [Removing unused CSS rules](#)
 - [Fixing HTML](#)
 - [Beautifying files](#)
 - [Inserting an inline Table of Contents](#)
 - [Setting Semantics](#)
 - [Filtering style information](#)
 - [Upgrading the book's internals](#)

- [Checkpoints](#)
- [The Live preview panel](#)
 - [Splitting HTML files](#)
- [The Live CSS panel](#)
- [Miscellaneous tools](#)
 - [The Table of Contents view](#)
 - [Checking the spelling of words in the book](#)
 - [Inserting special characters](#)
 - [The code inspector view](#)
 - [Checking external links](#)
 - [Downloading external resources](#)
 - [Arranging files into folders by type](#)
 - [Importing files in other e-book formats as EPUB](#)
 - [The Reports tool](#)
- [Special features in the code editor](#)
 - [Syntax highlighting](#)
 - [Context sensitive help](#)
 - [Auto-complete](#)
 - [Snippets](#)

Basic workflow

Note

A video tour of the calibre E-book editor is available [here](https://calibre-ebook.com/demo#tutorials) [https://calibre-ebook.com/demo#tutorials].

When you first open a book with the Edit book tool, you will be presented with a list of files on the left. These are the individual HTML files, stylesheets, images, etc. that make up the content of the book. Simply double click on a file to start editing it. Note that if you want to

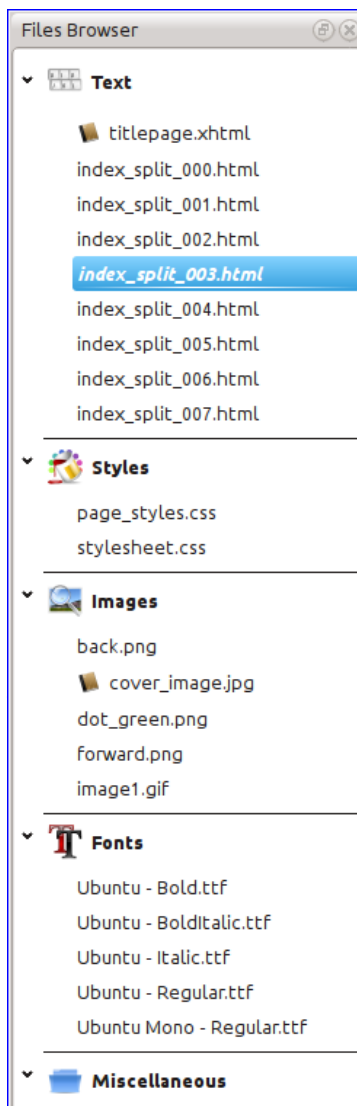
do anything more sophisticated than making a few small tweaks, you will need to know [HTML Tutorial](http://html.net/tutorials/html/) and [CSS Tutorial](http://html.net/tutorials/css/).

As you make changes to the HTML or CSS in the editor, the changes will be previewed, live, in the preview panel to the right. When you are happy with how the changes you have made look, click the *Save* button or use *File* → *Save* to save your changes into the e-book.

One useful feature is *Checkpoints*. Before you embark on some ambitious set of edits, you can create a checkpoint. The checkpoint will preserve the current state of your book, then if in the future you decide you don't like the changes you have made to you can go back to the state when you created the checkpoint. To create a checkpoint, use *Edit* → *Create checkpoint*. Checkpoints will also be automatically created for you whenever you run any automated tool like global search and replace. The checkpointing functionality is in addition to the normal undo/redo mechanism when editing individual files. Checkpoints are useful for when changes are spread over multiple files in the book.

That is the basic work flow for editing books – Open a file, make changes, preview and save. The rest of this manual will discuss the various tools and features present to allow you to perform specific tasks efficiently.

The File browser



you an overview of the various files inside the book you are editing. category, with text (HTML) files at the top, followed by stylesheet on. Simply double click on a file to start editing it. Editing is and image files. The order of text files is the same order that they if you were reading the book. All other files are arranged

over an entry, you can see its size, and also, at the bottom of the file inside the book. Note that files inside e-books are compressed, book is not the sum of the individual file sizes.

eaning, in the book. These will typically have an icon next to their ial meaning. For example, in the picture to the left, you can see that and *titlepage.xhtml* have the icon of a cover next to them, this book cover image and titlepage. Similarly, the *content.opf* file has a indicating the book metadata is present in it and the *toc.ncx* file has ng it is the Table of Contents.

tions on individual files, by right clicking them.

You can rename an individual file by right clicking it and selecting *Rename*. Renaming a file automatically updates all links and references to it throughout the book. So all you have to do is provide the new name, calibre will take care of the rest.

You can also bulk rename many files at once. This is useful if you want the files to have some simple name pattern. For example you might want to rename all the HTML files to have names Chapter-1.html, Chapter-2.html and so on. Select the files you want bulk renamed by holding down the Shift or Ctrl key and clicking the files. Then right click and select *Bulk rename*. Enter a prefix and what number you would like the automatic numbering to start at, click OK and you are done. The bulk rename dialog also lets you rename files by the order they appear in the book instead of the order you selected them in, useful, for instance to rename all images by the order they appear.

Finally, you can bulk change the file extension for all selected files. Select multiple files, as above, and right click and choose *Change the file extension for the selected files*.

Merging files

Sometimes, you may want to merge two HTML files or two CSS files together. It can sometimes be useful to have everything in a single file. Be wary, though, putting a lot of content into a single file will cause performance problems when viewing the book in a typical e-book reader.

To merge multiple files together, select them by holding the Ctrl key and clicking on them (make sure you only select files of one type, either all HTML files or all CSS files and so on). Then right click and select merge. That's all, calibre will merge the files, automatically taking care of migrating all links and references to the merged files. Note that merging files can sometimes cause text styling to change, since the individual files could have used different stylesheets.

You can also select text files and then drag and drop the text files onto another text file to merge the dropped text files into the target text file.

Changing text file order

You can re-arrange the order in which text (HTML) files are opened when reading the book by simply dragging and dropping them in the Files browser. For the technically inclined, this is called re-ordering the book spine. Note that you have to drop the items *between* other items, not on top of them, this can be a little fiddly until you get used to it. Dropping on top of another file will cause the files to be merged.

Marking the cover

E-books typically have a cover image. This image is indicated in the *File browser* by the icon of a brown book next to the image name. If you want to designate some other image as the cover, you can do so by right clicking on the file and choosing *Mark as cover*.

In addition, EPUB files have the concept of a *titlepage*. A title page is a HTML file that acts as the title page/cover for the book. You can mark an HTML file as the titlepage when editing EPUBs by right-clicking. Be careful that the file you mark contains only the cover information. If it contains other content, such as the first chapter, then that content will be lost if the user ever converts the EPUB file in calibre to another format. This is because when converting, calibre assumes that the marked title page contains only the cover and no other content.

[Deleting files](#)

You can delete files by either right clicking on them or by selecting them and pressing the Delete key. Deleting a file removes all references to the file from the OPF file, saving you that chore. However, references in other places are not removed, you can use the Check Book tool to easily find and remove/replace them.

[Exporting files](#)

You can export a file from inside the book to somewhere else on your computer. This is useful if you want to work on the file in isolation, with specialised tools. To do this, simply right click on the file and choose *Export*.

Once you are done working on the exported file, you can re-import it into the book, by right clicking on the file again and choosing *Replace with file...* which will allow you to replace the file in the book with the previously exported file.

You can also copy files between multiple editor instances. Select the files you want to copy in the *File browser*, then right click and choose, *Copy selected files to another editor instance*. Then, in the other editor instance, right click in the *File browser* and choose *Paste file from other editor instance*.

[Adding new images/fonts/etc. or creating new blank files](#)

You can add a new image, font, stylesheet, etc. from your computer into the book by clicking *File → New file*. This lets you either import a file by clicking the *Import resource file* button or create a new blank HTML file or stylesheet by simply entering the file name into the box for the new file.

You can also import multiple files into the book at once using *File->Import files into book*.

[Replacing files](#)

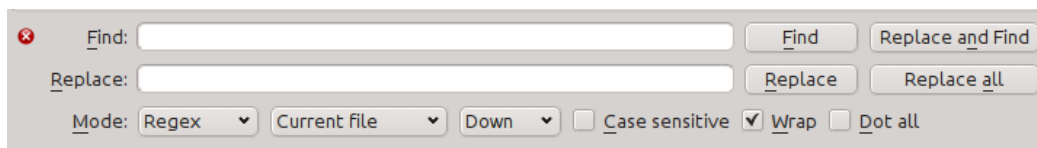
You can easily replace existing files in the book, by right clicking on the file and choosing *replace*. This will automatically update all links and references, in case the replacement file has a different name than the file being replaced.

[Linking stylesheets to HTML files efficiently](#)

As a convenience, you can select multiple HTML files in the File browser, right click and choose Link stylesheets to have calibre automatically insert the <link> tags for those stylesheets into all the selected HTML files.

Search & replace

Edit book has a very powerful search and replace interface that allows you to search and replace text in the current file, across all files and even in a marked region of the current file. You can search using a normal search or using regular expressions. To learn how to use regular expressions for advanced searching, see [All about using regular expressions in calibre](#).



Start the search and replace via the *Search → Find/replace* menu entry (you must be editing an HTML or CSS file).

Type the text you want to find into the Find box and its replacement into the Replace box. You can click the appropriate buttons to Find the next match, replace the current match and replace all matches.

Using the drop downs at the bottom of the box, you can have the search operate over the current file, all text files, all style files or all files. You can also choose the search mode to be a normal (string) search or a regular expression search.

You can count all the matches for a search expression via *Search → Count all*. The count will run over whatever files/regions you have selected in the dropdown box.

You can also go to a specific line in the currently open editor via *Search → Go to line*.

Note

Remember, to harness the full power of search and replace, you will need to use regular expressions. See [All about using regular expressions in calibre](#).

Saved searches

You can save frequently used search/replace expressions (including function mode expressions) and reuse them multiple times. To save a search simply right click in the Find box and select *Save current search*.

You can bring up the saved searches via *Search* → *Saved searches*. This will present you with a list of search and replace expressions that you can apply. You can even select multiple entries in the list by holding down the Ctrl key while clicking so as to run multiple search and replace expressions in a single operation.

[Function mode](#)

Function mode allows you to write arbitrarily powerful Python functions that are run on every Find/replace. You can do pretty much any text manipulation you like in function mode. For more information, see [Function mode for Search & replace in the Editor](#).

[Search ignoring HTML tags](#)

There is also a dedicated tool for searching for text, ignoring any HTML tags in between. For example,

on

a

<i>word</i>.

you can search for on

a

word

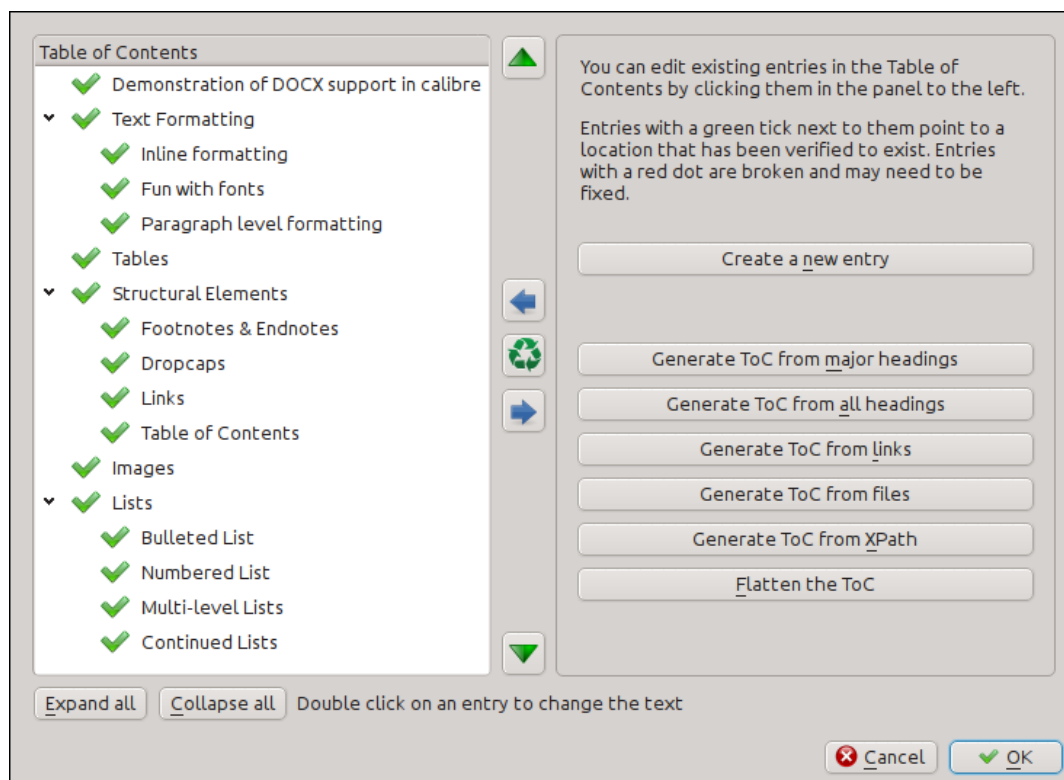
and it will be found even though there is an <i> tag in the middle. Use this tool via the *Search* → *Search ignoring HTML markup* menu item.

[Automated tools](#)

Edit book has various tools to help with common tasks. These are accessed via the *Tools* menu.

[Editing the Table of Contents](#)

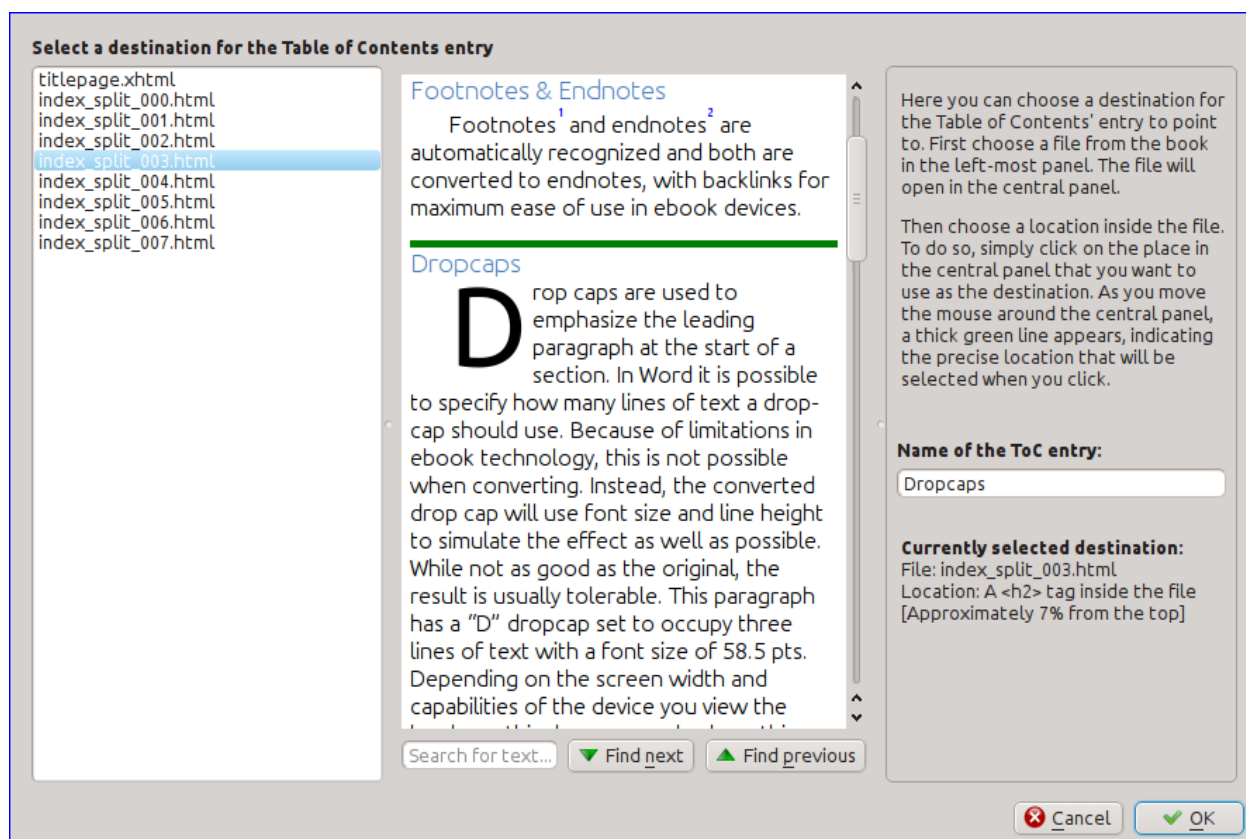
There is a dedicated tool to ease editing of the Table of Contents. Launch it with *Tools* → *Table of Contents* → *Edit Table of Contents*.



The Edit Table of Contents tool shows you the current Table of Contents (if any) on the left. Simply double click on any entry to change its text. You can also re-arrange entries by drag and drop or by using the buttons to the right.

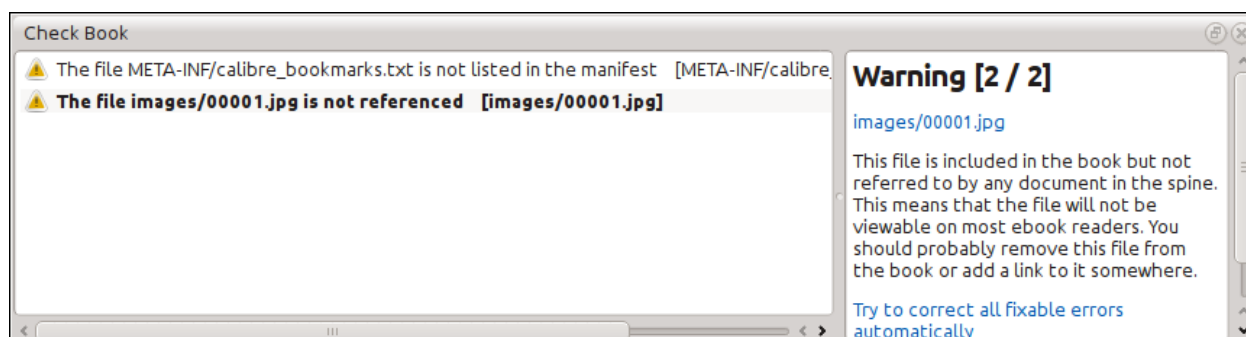
For books that do not have a pre-existing Table of Contents, the tool gives you various options to auto-generate a Table of Contents from the text. You can generate from the headings in the document, from links, from individual files and so on.

You can edit individual entries by clicking on them and then clicking the *Change the location this entry points to* button. This will open up a mini-preview of the book, simply move the mouse cursor over the book view panel, and click where you want the entry to point to. A thick green line will show you the location. Click OK once you are happy with the location.



Checking the book

The *Check book* tool searches your book for problems that could prevent it working as intended on actual reader devices. Activate it via *Tools* → *Check book*.



Any problems found are reported in a nice, easy to use list. Clicking any entry in the list shows you some help about that error as well as giving you the option to auto-fix that error, if the error can be fixed automatically. You can also double click the error to open the location of the error in an editor, so you can fix it yourself.

Some of the checks performed are:

- Malformed HTML markup. Any HTML markup that does not parse as well-formed XML is reported. Correcting it will ensure that your markup works as intended in all contexts. calibre can also auto-fix these errors, but auto-fixing can sometimes have unexpected effects, so use with care. As always, a checkpoint is created before auto-fixing so you can easily revert all changes. Auto-fixing works by parsing the markup using the HTML5 algorithm, which is highly fault tolerant and then converting to well formed XML.
- Malformed or unknown CSS styles. Any CSS that is not valid or that has properties not defined in the CSS 2.1 standard (plus a few from CSS 3) are reported. CSS is checked in all stylesheets, inline style attributes and <style> tags in HTML files.
- Broken links. Links that point to files inside the book that are missing are reported.
- Unreferenced files. Files in the book that are not referenced by any other file or are not in the spine are reported.
- Various common problems in OPF files such as duplicate spine or manifest items, broken idrefs or meta cover tags, missing required sections and so on.
- Various compatibility checks for known problems that can cause the book to malfunction on reader devices.

[Adding a cover](#)

You can easily add a cover to the book via *Tools* → *Add cover*. This allows you to either choose an existing image in the book as the cover or import a new image into the book and make it the cover. When editing EPUB files, the HTML wrapper for the cover is automatically generated. If an existing cover in the book is found, it is replaced. The tool also automatically takes care of correctly marking the cover files as covers in the OPF.

[Embedding referenced fonts](#)

Accessed via *Tools* → *Embed reference fonts*, this tool finds all fonts referenced in the book and if they are not already embedded, searches your computer for them and embeds them into the book, if found. Please make sure that you have the necessary copyrights for embedding commercially licensed fonts, before doing this.

[Subsetting embedded fonts](#)

Accessed via *Tools* → *Subset embedded fonts*, this tool reduces all the fonts in the book to only contain glyphs for the text actually present in the book. This commonly reduces the size of the font files by ~ 50%. However, be aware that once the fonts are subset, if you add new text whose characters are not previously present in the subset font, the font will not work for the new text. So do this only as the last step in your workflow.

Smartening punctuation

Convert plain text dashes, ellipsis, quotes, multiple hyphens, etc. into their typographically correct equivalents. Note that the algorithm can sometimes generate incorrect results, especially when single quotes at the start of contractions are involved. Accessed via *Tools* → *Smarten punctuation*.

Transforming CSS properties

Create rules to transform the styling of the book. For example, create a rule to convert all red text to green or to double the font size of all text in the book or make text of a certain font family italic, etc.

Creating the rules is simple, the rules follow a natural language format, that looks like:

- If the property *color* is *red* change it to *green*
- If the property *font-size* is *any value* multiply the value by 2

Accessed via *Tools* → *Transform styles*.

Removing unused CSS rules

Remove all unused CSS rules from stylesheets and <style> tags. Some books created from production templates can have a large number of extra CSS rules that don't match any actual content. These extra rules can slow down readers that need to process them all. Accessed via *Tools* → *Remove unused CSS*.

Fixing HTML

This tool simply converts HTML that cannot be parsed as XML into well-formed XML. It is very common in e-books to have non-well-formed XML, so this tool simply automates the process of fixing such HTML. The tool works by parsing the HTML using the HTML5 algorithm (the algorithm used in all modern browsers) and then converting the result into XML. Be aware that auto-fixing can sometimes have counter-intuitive results. If you prefer, you can use the Check Book tool discussed above to find and manually correct problems in the HTML. Accessed via *Tools* → *Fix HTML*.

Beautifying files

This tool is used to auto-format all HTML and CSS files so that they “look pretty”. The code is auto-indented so that it lines up nicely, blank lines are inserted where appropriate and so on. Note that beautifying also auto-fixes broken HTML/CSS. Therefore, if you don't want any auto-fixing to be performed, first use the Check Book tool to correct all problems and only then run beautify. Accessed via *Tools* → *Beautify all files*.

Note

In HTML any text can have significant whitespace, via the CSS white-space directive. Therefore, beautification could potentially change the rendering of the HTML. To avoid this as far as possible, the beautify algorithm only beautifies block level tags that contain other block level tags. So, for example, text inside a <p> tag will not have its whitespace changed. But a <body> tag that contains only other <p> and <div> tags will be beautified. This can sometimes mean that a particular file will not be affected by beautify as it has no suitable block level tags. In such cases you can try different beautification tools, that are less careful, for example: [HTML Tidy](https://infohound.net/tidy/) [https://infohound.net/tidy/].

Inserting an inline Table of Contents

Normally in e-books, the Table of Contents is separate from the main text and is typically accessed via a special Table of Contents button/menu in the e-book reading device. You can also have calibre automatically generate an *inline* Table of Contents that becomes part of the text of the book. It is generated based on the currently defined Table of Contents.

If you use this tool multiple times, each invocation will cause the previously created inline Table of Contents to be replaced. The tool can be accessed via *Tools* → *Table of Contents* → *Insert inline Table of Contents*.

Setting Semantics

This tool is used to set *semantics* in EPUB files. Semantics are simply, links in the OPF file that identify certain locations in the book as having special meaning. You can use them to identify the foreword, dedication, cover, table of contents, etc. Simply choose the type of semantic information you want to specify and then select the location in the book the link should point to. This tool can be accessed via *Tools* → *Set semantics*.

Filtering style information

This tool can be used to easily remove specified CSS style properties from the entire book. You can tell it what properties you want removed, for example, color,

background-color,

line-height

and it will remove them from everywhere they occur — stylesheets, <style> tags and inline style attributes. After removing the style information, a summary of all the changes made is displayed so you can see exactly what was changed. The tool can be accessed via *Tools* → *Filter style information*.

Upgrading the book's internals

This tool can be used to upgrade the book's internals, if possible. For instance it will upgrade EPUB 2 books to EPUB 3 books. The tool can be accessed via *Upgrade book internals*.

Checkpoints

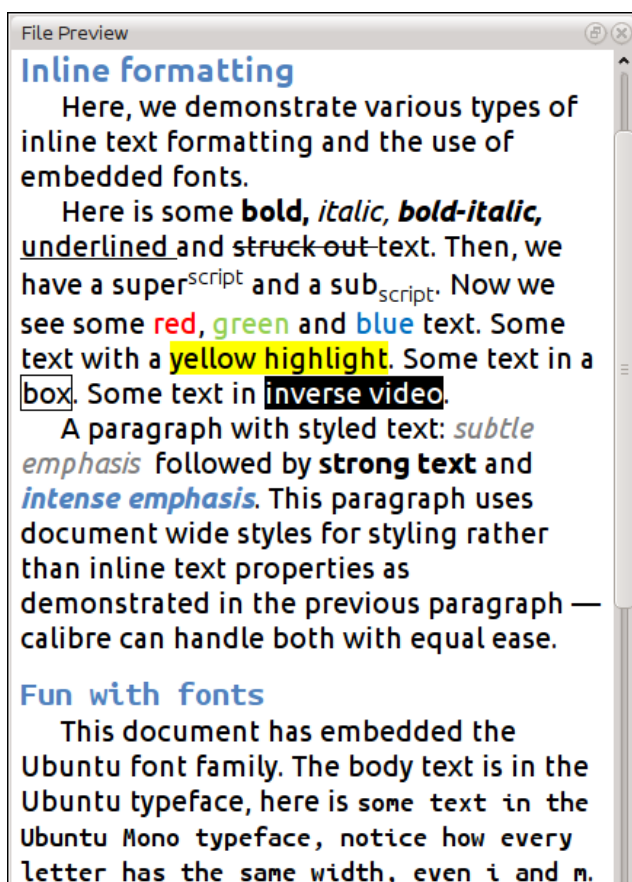
Checkpoints are a way to mark the current state of the book as “special”. You can then go on to do whatever changes you want to the book and if you don't like the results, return to the checkpointed state. Checkpoints are automatically created every time you run any of the automated tools described in the previous section.

You can create a checkpoint via *Edit → Create checkpoint*. And go back to a previous checkpoint with *Edit → Revert to ...*

The check pointing functionality is in addition to the normal Undo/redo mechanism when editing individual files. Checkpoints are particularly useful for when changes are spread over multiple files in the book or when you wish to be able to revert a large group of related changes as a whole.

You can see a list of available checkpoints via *View → Checkpoints*. You can compare the current state of the book to a specified checkpoint using the [Comparing e-books](#) tool – by selecting the checkpoint of interest and clicking the *Compare* button. The *Revert to* button restores the book to the selected checkpoint, undoing all changes since that checkpoint was created.

The Live preview panel



the various files inside The live preview panel (with a second or two of delay). As you edit HTML automatically to reflect your changes. As you move the preview panel will track its location, showing you the current position in the preview panel, will cause the cursor in the document to move to the position you clicked. If you click a link pointing to another file, the editor and the preview panel, automatically.

Navigation and live preview of changes – by buttons in the preview panel only happens when you are editing. This is not distracting or slow you down, waiting for the preview to update.

The preview will look when viewed. However, the preview is not a real book or an actual reader device. It is both more, and less, tolerant of errors and sloppy markup much better than a real reader. It shows you page margins, page breaks and embedded

fonts that use font name aliasing. Use the preview panel while you are working on the book, but once you are done, review it in an actual reader device or software emulator.

Note

The preview panel does not support embedded fonts if the name of the font inside the font file does not match the name in the CSS @font-face rule. You can use the Check Book tool to quickly find and fix any such problem fonts.

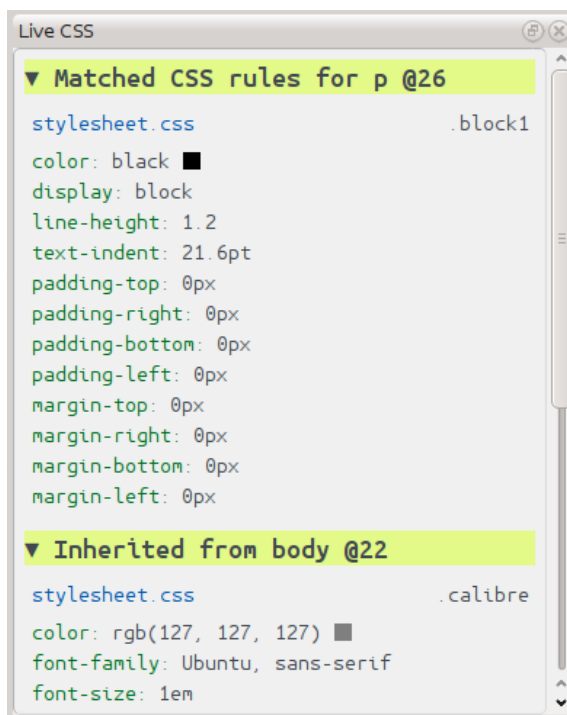
Splitting HTML files

One, perhaps non-obvious, use of the preview panel is to split long HTML files. While viewing the file you want to split, click the *Split mode* button under the preview panel . Then simply move your mouse to the place where you want to split the file and click. A thick green line will show you exactly where the split will happen as you move your mouse. Once you have found the location you want, simply click and the split will be performed.

Splitting the file will automatically update all links and references that pointed into the bottom half of the file and will open the newly split file in an editor.

You can also split a single HTML file at multiple locations automatically, by right clicking inside the file in the editor and choosing *Split at multiple locations*. This will allow you to easily split a large file at all heading tags or all tags having a certain class and so on.

The Live CSS panel



style rules that apply to the tag you are currently viewing. The number in the editor are displayed, followed by

rules apply to any tag. The view also has clickable links to the location where the style was defined, in case you want to edit the rules. Style rules that apply directly to the tag, as opposed to parent tags are shown.

calculated styles for the tag are. Properties in the 'Inherited' section are shown with a line through them.

Click the 'Live CSS' button in the top right corner of the panel.

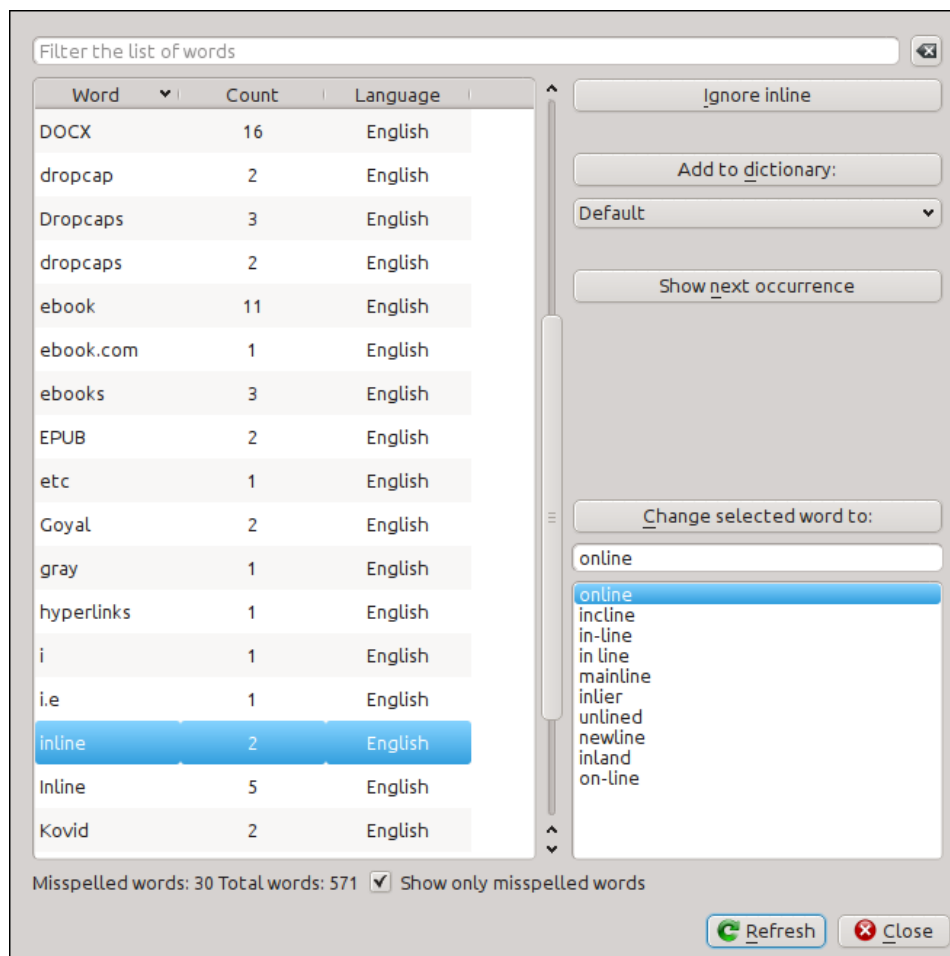
There are a few more tools that can be useful while you edit the book.

[The Table of Contents view](#)

The Table of Contents view shows you the current table of contents in the book. Double clicking on any entry opens the place that entry points to in an editor. You can right click to edit the Table of Contents, refresh the view or expand/collapse all items. Access this view via *View → Table of Contents*.

[Checking the spelling of words in the book](#)

You can run a spelling checker via *Tools → Check spelling*.



Words are shown with the number of times they occur in the book and the language the word belongs to. Language information is taken from the books metadata and from lang attributes in the HTML files. This allows the spell checker to work well even with books that contain text in multiple languages. For example, in the following HTML extract, the word color will be checked using American English and the word colour using British English:

```
<
div

lang
=
"en_US"
>
color

<
span

lang
=
"en_GB"
>
colour
</
span
></
div
>
```

Note

You can double click a word to highlight the next occurrence of that word in the editor. This is useful if you wish to manually edit the word, or see what context it is in.

To change a word, simply double click one of the suggested alternative spellings on the right, or type in your own corrected spelling and click the *Change selected word to* button. This will replace all occurrences of the word in the book. You can also right click on a word in the main word list to change the word conveniently from the right click menu.

You can have the spelling checker ignore a word for the current session by clicking the *Ignore* button. You can also add a word to the user dictionary by clicking the *Add to dictionary* button. The spelling checker supports multiple user dictionaries, so you can select the dictionary you want the word added to.

You can also have the spelling checker display all the words in your book, not just the incorrectly spelled ones. This is useful to see what words are most common in your book and to run a simple search and replace on individual words.

Note

If you make any changes to the book by editing files while the spell check tool is open, you should click the *Refresh* button in the Spell check tool. If you do not do this and continue to use the Spell check tool, you could lose the changes you have made in the editor.

Note

To exclude an individual file from being spell checked when running the spell check tool, you can use the *Exclude files* button or add the following comment just under the opening tag in the file:

```
<!-- calibre-no-spell-check -->
```

Adding new dictionaries

The spelling checker comes with builtin dictionaries for the English and Spanish languages. You can install your own dictionaries via *Preferences → Editor → Manage spelling dictionaries*. The spell checker can use dictionaries from the LibreOffice program (in the .oxf format). You can download these dictionaries from [The LibreOffice Extensions repository](https://extensions.libreoffice.org/?Tags%5B%5D=50) [https://extensions.libreoffice.org/?Tags%5B%5D=50].

Inserting special characters

You can insert characters that are difficult to type by using the *Edit → Insert special character* tool. This shows you all Unicode characters, simply click on the character you want to type. If you hold Ctrl while clicking, the window will close itself after inserting the selected character. This tool can be used to insert special characters into the main text or into any other area of the user interface, such as the Search and replace tool.

Because there are a lot of characters, you can define your own *Favorite* characters, that will be shown first. Simply right click on a character to mark it as favorite. You can also right click on a character in favorites to remove it from favorites. Finally, you can re-arrange the order of characters in favorites by clicking the *Re-arrange favorites* button and then drag and dropping the characters in favorites around.

You can also directly type in special characters using the keyboard. To do this, you type the Unicode code for the character (in hexadecimal) and then press the Alt+X key which will convert the previously typed code into the corresponding character. For example, to type ÿ you would type ff and then Alt+X. To type a non-breaking space you would use a0 and then Alt+X, to type the horizontal ellipsis you would use 2026 and Alt+X and so on.

Finally, you can type in special characters by using HTML named entities. For example, typing will be replaced by a non breaking space when you type the semi-colon. The replacement happens only when typing the semi-colon.

[The code inspector view](#)

This view shows you the HTML coding and CSS that applies to the current element of interest. You open it by right clicking a location in the preview panel and choosing *Inspect*. It allows you to see the HTML coding for that element and more importantly, the CSS styles that apply to it. You can even dynamically edit the styles and see what effect your changes have instantly. Note that editing the styles does not actually make changes to the book contents, it only allows for quick experimentation. The ability to live edit inside the Inspector is under development.

[Checking external links](#)

You can use this tool to check all links in your book that point to external websites. The tool will try to visit every externally linked website, and if the visit fails, it will report all broken links in a convenient format for you to fix.

[Downloading external resources](#)

You can use this tool to automatically download any images/stylesheets/etc. in the book that are not bundled with the book (i.e. they have URLs pointing to a location on the internet). The tool will find all such resources and automatically download them, add them to the book and replace all references to them to use the downloaded files.

[Arranging files into folders by type](#)

Often when editing EPUB files that you get from somewhere, you will find that the files inside the EPUB are arranged haphazardly, in different sub-folders. This tool allows you to automatically move all files into sub-folders based on their types. Access it via *Tools* → *Arrange into folders*. Note that this tool only changes how the files are arranged inside the EPUB, it does not change how they are displayed in the File browser.

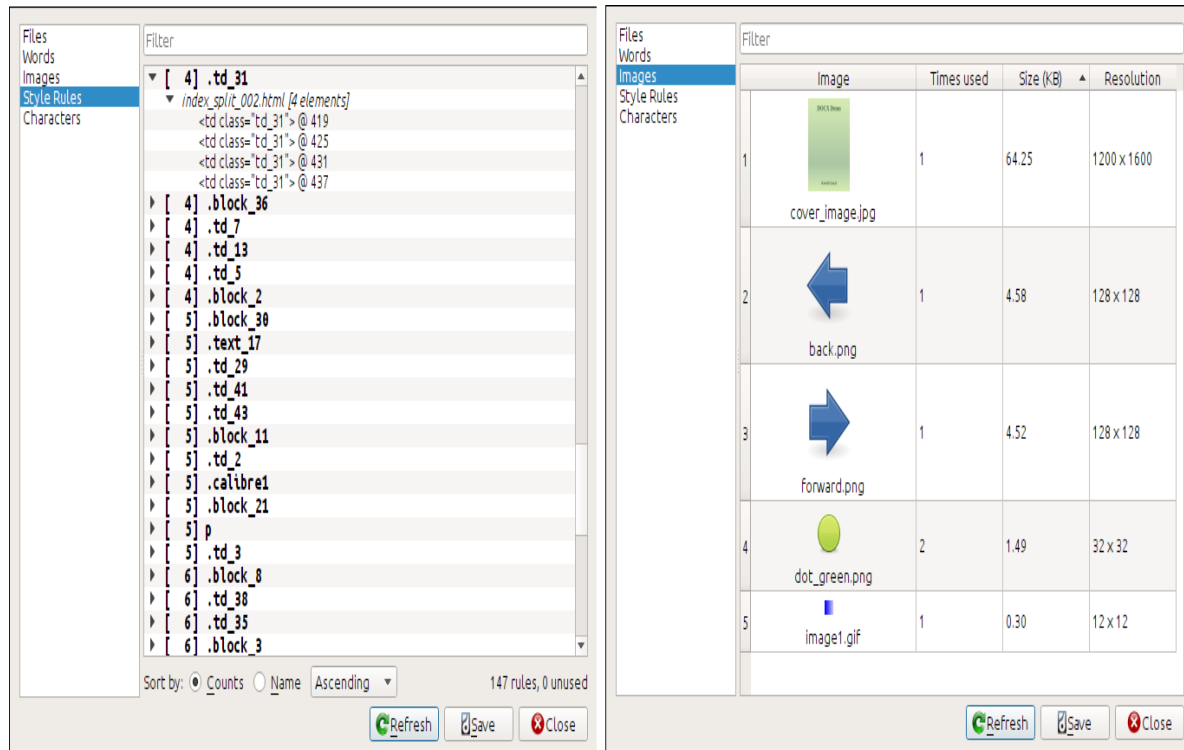
[Importing files in other e-book formats as EPUB](#)

The editor includes the ability to import files in some other e-book formats directly as a new EPUB, without going through a full conversion. This is particularly useful to directly create EPUB files from your own hand-edited HTML files. You can do this via *File* → *Import an HTML or DOCX file as a new book*.

[The Reports tool](#)

The editor includes a nice *Reports* tool (via *Tools* → *Reports*) that shows summaries of the files, images, links, words, characters and styles used in the book. Every line in the report is hot-linked. Double clicking a line jumps to the place in the book where that item is used or defined (as appropriate). For example, in the *Links* view, you can double click entries the

Source column to jump to where the link is defined and entries in the *Target* column to jump to where the link points.



Special features in the code editor

The calibre HTML editor is very powerful. It has many features that make editing of HTML (and CSS) easier.

Syntax highlighting

The HTML editor has very sophisticated syntax highlighting. Features include:

- The text inside bold, italic and heading tags is made bold/italic
- As you move your cursor through the HTML, the matching HTML tags are highlighted, and you can jump to the opening or closing tag with the keyboard shortcuts Ctrl+{ and Ctrl+}. Similarly, you can select the contents of a tag with Ctrl+Alt+T.
- Invalid HTML is highlighted with a red underline
- Spelling errors in the text inside HTML tags and attributes such as title are highlighted. The spell checking is language aware, based on the value of the lang attribute of the current tag and the overall book language.

- CSS embedded inside `<style>` tags is highlighted
- Special characters that can be hard to distinguish such as non-breaking spaces, different types of hyphens, etc. are highlighted.
- Links to other files in `<a>` tags, `<img>` and `<link>` tags all have the filenames highlighted. If the filename they point to does not exist, the filename is marked with a red underline.

Context sensitive help

You can right click on an HTML tag name or a CSS property name to get help for that tag or property.

You can also hold down the Ctrl key and click on any filename inside a link tag to open that file in the editor automatically. Similarly, Ctrl clicking a class name will take you to the first style rule that matches the tag and class.

Right clicking a class name in an HTML file will allow you to rename the class, changing all occurrences of the class throughout the book and all its stylesheets.

Auto-complete

When editing an e-book, one of the most tedious tasks is creating links to other files inside the book, or to CSS stylesheets, or images. You have to figure out the correct filename and relative path to the file. The editor has auto-complete to make that easier.

As you type a filename, the editor automatically pops up suggestions. Simply use the Tab key to select the correct file name. The editor even offers suggestions for links pointing to an anchor inside another HTML file. After you type the # character, the editor will show you a list of all anchors in the target file, with a small snippet of text to help you choose the right anchor.

Note that unlike most other completion systems, the editor's completion system uses subsequence matching. This means that you can type just two or three letters from anywhere in the filename to complete the filename. For example, say you want the filename `../images/arrow1.png`

, you can simply type `ia1` and press Tab to complete the filename. When searching for matches, the completion system prioritizes letters that are at the start of a word, or immediately after a path separator. Once you get used to this system, you will find it saves you a lot of time and effort.

[Snippets](#)

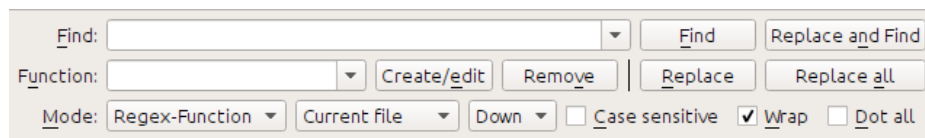
The calibre E-book editor supports *snippets*. A snippet is a piece of text that is either re-used often or contains a lot of redundant text. The editor allows you to insert a snippet with only a few key strokes. The snippets are very powerful, with many features, such as placeholders you can jump between, automatic mirroring of repeated text and so on. For more information, see [Snippets](#).

Function mode for Search & replace in the Editor

The *Search & replace* tool in the editor support a *function mode*. In this mode, you can combine regular expressions (see [All about using regular expressions in calibre](#)) with arbitrarily powerful Python functions to do all sorts of advanced text processing.

In the standard *regex* mode for search and replace, you specify both a regular expression to search for as well as a template that is used to replace all found matches. In function mode, instead of using a fixed template, you specify an arbitrary function, in the [Python programming language](#) [https://docs.python.org]. This allows you to do lots of things that are not possible with simple templates.

Techniques for using function mode and the syntax will be described by means of examples, showing you how to create functions to perform progressively more complex tasks.



Automatically fixing the case of headings in the document

Here, we will leverage one of the builtin functions in the editor to automatically change the case of all text inside heading tags to title case:

Find expression: <([Hh][1-6])(^>)*>.+?</1>

For the function, simply choose the *Title-case text (ignore tags)* builtin function. The will change titles that look like: <h1>some

TITLE</h1>

to

<h1>Some

Title</h1>

. It will work even if there are other HTML tags inside the heading tags.

Your first custom function - smartening hyphens

The real power of function mode comes from being able to create your own functions to process text in arbitrary ways. The Smarten Punctuation tool in the editor leaves individual hyphens alone, so you can use the this function to replace them with em-dashes.

To create a new function, simply click the *Create/edit* button to create a new function and copy the Python code from below.

```
def
replace
(
match
,
number
,
file_name
,
metadata
,
dictionaries
,
data
,
functions
,
*
args
,
**
kwargs
):
return
```

```

match
.
group
()
.
replace
(
'_'
,
'_'
)
.
replace
(
'_'
,
'_'
)

```

Every *Search & replace* custom function must have a unique name and consist of a Python function named `replace`, that accepts all the arguments shown above. For the moment, we won't worry about all the different arguments to `replace()` function. Just focus on the `match` argument. It represents a match when running a search and replace. Its full documentation is available [here](https://docs.python.org/library/re.html#match-objects) [<https://docs.python.org/library/re.html#match-objects>]. `match.group()` simply returns all the matched text and all we do is replace hyphens in that text with em-dashes, first replacing double hyphens and then single hyphens.

Use this function with the find regular expression:

```

>
[
^<>
]
+<

```

And it will replace all hyphens with em-dashes, but only in actual text and not inside HTML tag definitions.

The power of function mode - using a spelling dictionary to fix mis-hyphenated words

Often, e-books created from scans of printed books contain mis-hyphenated words – words that were split at the end of the line on the printed page. We will write a simple function to automatically find and fix such words.

```
import
regex

from
calibre

import

replace_entities

from
calibre

import

prepare_string_for_xml

def
replace
(
match
,
number
,
file_name
,
metadata
,
```

```
dictionaries
,

data
,

functions
,

*
args
,

**
kwargs
):

def

replace_word
(
wmatch
):

# Try to remove the hyphen and replace the words if the resulting

# hyphen free word is recognized by the dictionary

without_hyphen

=

wmatch
.
group
(
1
```

```
)  
  
+  
  
wmatch  
.  
group  
(  
2  
)  
  
if  
  
dictionaries  
.  
recognized  
(  
without_hyphen  
):  
  
return  
  
without_hyphen  
  
return  
  
wmatch  
.  
group  
(  
  
# Search for words split by a hyphen  
  
text  
  
=
```

```
replace_entities
(
match
.
group
()[
1
:
-
1
])

# Handle HTML entities like &amp;
```

```
corrected

=

regex
.
sub
(
r
'(\w+)\s*-\s*(\w+)'
,
```

```
replace_word
,
```

```
text
,
```

```
flags
=
regex
```

```
.
VERSION1
```

```
|
```

```
regex
```

```
.
```

```

UNICODE
)

return

'>
%s
<'

%

prepare_string_for_xml
(
corrected
)

# Put back required entities

```

Use this function with the same find expression as before, namely:

```

>
[
^<>
]
+<

```

And it will magically fix all mis-hyphenated words in the text of the book. The main trick is to use one of the useful extra arguments to the replace function, `dictionaries`. This refers to the dictionaries the editor itself uses to spell check text in the book. What this function does is look for words separated by a hyphen, remove the hyphen and check if the dictionary recognizes the composite word, if it does, the original words are replaced by the hyphen free composite word.

Note that one limitation of this technique is it will only work for mono-lingual books, because, `by` default, `dictionaries.recognized()` uses the main language of the book.

Auto numbering sections

Now we will see something a little different. Suppose your HTML file has many sections, each with a heading in an `<h2>` tag that looks like `<h2>Some`

`text</h2>`

. You can create a custom function that will automatically number these headings with consecutive section numbers, so that they look like `<h2>1.`

Some

`text</h2>`

.

```
def
replace
(
match
,
number
,
file_name
,
metadata
,
dictionaries
,
data
,
functions
,
*
args
,
**
```



```
kwargs
):

section_number

=

'
%d
.'
'

%

number

return

match
.
group
(
1
)

+

section_number

+

match
.
group
(
2
)

# Ensure that when running over multiple files, the files are processed
# in the order in which they appear in the book
```

```
replace
.
file_order

=

'spine'
```

Use it with the find expression:

```
(?s)(<h2[^\<>]*>)(.+?</h2>)
```

Place the cursor at the top of the file and click *Replace all*.

This function uses another of the useful extra arguments to `replace()`: the `number` argument. When doing a *Replace All* number is automatically incremented for every successive match.

Another new feature is the use of `replace.file_order` – setting that to `'spine'` means that if this search is run on multiple HTML files, the files are processed in the order in which they appear in the book. See [Choose file order when running on multiple HTML files](#) for details.

Auto create a Table of Contents

Finally, lets try something a little more ambitious. Suppose your book has headings in `h1` and `h2` tags that look like `<h1`

`id="someid">Some`

`Text</h1>`

. We will auto-generate an HTML Table of Contents based on these headings. Create the custom function below:

```
from
calibre
```

```
import
replace_entities

from
calibre.ebooks.oeb.polish.toc

import
TOC
,
toc_to_html

from
calibre.gui2.tweak_book

import
current_container

from
calibre.ebooks.oeb.base

import
xml2str

def
replace
(
match
,
number
,
file_name
```

```
,
metadata
,
dictionaries
,
data
,
functions
,
*
args
,
**
kwargs
):

if
match

is

None
:

# All matches found, output the resulting Table of Contents.

# The argument metadata is the metadata of the book being edited

if

'toc'
```

```
in

data
:

toc

=

data
[
'toc'
]

root

=

TOC
()

for

(
file_name
,
tag_name
,
anchor
,
text
)

in

toc
:
```

```
parent
=
root
.
children
[
-
1
]
if
tag_name
```

```
==
'h2'
```

```
and
root
.
children
```

```
else
```

```
root
```

```
parent
.
add
(
text
,
file_name
,
```

```
anchor
)

toc

=

toc_to_html
(
root
,

current_container
()),

'toc.html'
,

'Table of Contents for '

+

metadata
.
title
,

metadata
.
language
)

print

(
xml2str
(
toc
))
```

```
else
:

print

(
'No headings to build ToC from found'
)

else
:

# Add an entry corresponding to this match to the Table of Contents

if

'toc'

not

in

data
:

# The entries are stored in the data object, which will persist

# for all invocations of this function during a 'Replace All' operation

data
[
'toc'
]

=
```



```
[]

tag_name
,

anchor
,

text

=

match
.
group
(
1
),

replace_entities
(
match
.
group
(
2
)),

replace_entities
(
match
.
group
(
3
))

data
[
'toc'
]
```

```
.
append
((
file_name
,
tag_name
,
anchor
,
text
))

return

match
.
group
()

# We don't want to make any actual changes, so return the original matched text

# Ensure that we are called once after the last match is found so we can

# output the ToC

replace
.
call_after_last_match

=

True

# Ensure that when running over multiple files, this function is called,

# the files are processed in the order in which they appear in the book

replace
```

```
.  
file_order  
  
=  
  
'spine'
```

And use it with the find expression:

```
<  
(  
h  
[  
12  
)  
  
[  
^<>  
]  
*  
  
id  
=  
[  
"]([  
")+["  
]  
^<>  
]  
*>  
([  
^<>  
]  
+  
)
```

Run the search on *All text files* and at the end of the search, a window will popup with “Debug output from your function” which will have the HTML Table of Contents, ready to be pasted

into

toc.html

.

The function above is heavily commented, so it should be easy to follow. The key new feature is the use of another useful extra argument to the `replace()` function, the `data` object. The `data` object is a Python *dict* that persists between all successive invocations of `replace()` during a single *Replace All* operation.

Another new feature is the use of `call_after_last_match` – setting that to `True` on the `replace()` function means that the editor will call `replace()` one extra time after all matches have been found. For this extra call, the match object will be `None`.

This was just a demonstration to show you the power of function mode, if you really needed to generate a Table of Contents from headings in your book, you would be better off using the dedicated Table of Contents tool in *Tools* → *Table of Contents*.

The API for the function mode

All function mode functions must be Python functions named `replace`, with the following signature:

```
def
replace
(
match
,
number
,
file_name
,
metadata
,
dictionaries
```

```
,
data
,
functions
,
*
args
,
**
kwargs
):

return

a_string
```

When a find/replace is run, for every match that is found, the `replace()` function will be called, it must return the replacement string for that match. If no replacements are to be done, it should return `match.group()` which is the original string. The various arguments to the `replace()` function are documented below.

The match argument

The `match` argument represents the currently found match. It is a [Python Match object](https://docs.python.org/library/re.html#match-objects) [https://docs.python.org/library/re.html#match-objects]. Its most useful method is `group()` which can be used to get the matched text corresponding to individual capture groups in the search regular expression.

The number argument

The `number` argument is the number of the current match. When you run *Replace All*, every successive

match will cause replace()
to be called with an increasing number. The first match has number 1.

The file_name argument

This is the filename of the file in which the current match was found. When searching inside marked text, the file_name is empty. The file_name is in canonical form, a path relative to the root of the book, using / as the path separator.

The metadata argument

This represents the metadata of the current book, such as title, authors, language, etc. It is an object of class [calibre.ebooks.metadata.book.base.Metadata](#). Useful attributes include, title, authors, (a list of authors) and language (the language code).

The dictionaries argument

This represents the collection of dictionaries used for spell checking the current book. Its most useful method is dictionaries.recognized(word) which will return True if the passed in word is recognized by the dictionary for the current book's language.

The data argument

This is a simple Python dict. When you run *Replace all*, every successive match will cause replace() to be called with the same dict as data. You can thus use it to store arbitrary data between invocations of replace() during a *Replace all* operation.

The functions argument

The functions argument gives you access to all other user defined functions. This is useful for code re-use. You can define utility functions in one place and re-use them in all your other functions. For

example, suppose you create a function name My

Function
like this:

```
def
utility
():

# do something

def
replace
(
match
,
number
,
file_name
,
metadata
,
dictionaries
,
data
,
functions
,
*
args
,
```

```
**
kwargs
):

...

```

Then, in another function, you can access the utility() function like this:

```
def
replace
(
match
,
number
,
file_name
,
metadata
,
dictionaries
,
data
,
functions
,
*
args
,
**
kwargs

```



```
):  
  
utility  
  
=  
  
functions  
[  
  'My Function'  
][  
  'utility'  
]  
  
...
```

You can also use the functions object to store persistent data, that can be re-used by other functions. For example, you could have one function that when run with *Replace All* collects some data and another function that uses it when it is run afterwards. Consider the following two functions:

```
# Function One  
  
persistent_data  
  
=  
  
{  
  
  def  
  
  replace  
  (  
    match  
    ,  
  
    number  
    ,
```

```
file_name
,

metadata
,

dictionaries
,

data
,

functions
,

*
args
,

**
kwargs
):

...

persistent_data
[
'something'
]

=

'some data'

# Function Two

def

replace
(
```

```
match
,

number
,

file_name
,

metadata
,

dictionaries
,

data
,

functions
,

*
args
,

**
kwargs
):

persistent_data

=

functions
[
'Function One'
][
'persistent_data'
]

...
```

Debugging your functions

You can debug the functions you create by using the standard `print()` function from Python. The output of `print` will be displayed in a popup window after the Find/replace has completed. You saw an example of using `print()` to output an entire table of contents above.

Choose file order when running on multiple HTML files

When you run a *Replace all* on multiple HTML files, the order in which the files are processed depends on what files you have open for editing. You can force the search to process files in the order in which they appear by setting the `file_order` attribute on your function, like this:

```
def
replace
(
match
,
number
,
file_name
,
metadata
,
dictionaries
,
data
,
functions
,
```

```

*
args
,
**
kwargs
):

...

replace
.
file_order

=

'spine'

```

file_order
 accepts two values, spine
 and spine-reverse
 which cause the search to process multiple files in the order they appear in the book, either
 forwards or backwards, respectively.

Having your function called an extra time after the last match is found

Sometimes, as in the auto generate table of contents example above, it is useful to have your function called an extra time after the last match is found. You can do this by setting the call_after_last_match attribute on your function, like this:

```

def

replace
(
match
,
number
,

```

```
file_name
,
metadata
,
dictionaries
,
data
,
functions
,
*
args
,
**
kwargs
):
...

replace
.
call_after_last_match

=

True
```

Appending the output from the function to marked text

When running search and replace on marked text, it is sometimes useful to append so text to the end of the marked text. You can do that by setting the `append_final_output_to_marked`

attribute on your function (note that you also need to set `call_after_last_match`), like this:

```
def
replace
(
match
,
number
,
file_name
,
metadata
,
dictionaries
,
data
,
functions
,
*
args
,
**
kwargs
):
...

return
'some text to append'
```

```
replace
.
call_after_last_match

=

True

replace
.
append_final_output_to_marked

=

True
```

Suppressing the result dialog when performing searches on marked text

You can also suppress the result dialog (which can slow down the repeated application of a search/replace on many blocks of text) by setting the `suppress_result_dialog` attribute on your function, like this:

```
def
replace
(
match
,
number
,
file_name
,
metadata
,
dictionaries
```



```
,  
  
data  
,  
  
functions  
,  
  
*  
args  
,  
  
**  
kwargs  
):  
  
...  
  
replace  
.  
suppress_result_dialog  
  
=  
  
True
```

More examples

More useful examples, contributed by calibre users, can be found in the [calibre E-book editor forum](https://www.mobileread.com/forums/showthread.php?t=237181) [https://www.mobileread.com/forums/showthread.php?t=237181].

Snippets

The calibre E-book editor supports *snippets*. A snippet is a piece of text that is either re-used often or contains a lot of redundant text. The editor allows you to insert a snippet with only a few key strokes. For example, suppose you often find yourself inserting link tags when editing HTML files, then you can simply type `<a` in the editor and press `Control+J`. The editor will expand it to:

```
<
a

href
=
"filename"
></
a
>
```

Not only that, the word `filename` will be selected, with the cursor placed over it, so that you can easily type in the real filename, using the editor's nifty [Auto-complete](#) feature. And once you are done typing the filename, press `Control+J` again and the cursor will jump to the position in between the `<a>` tags so you can easily type in the text for the link.

The snippets system in the editor is very sophisticated, there are a few built-in snippets and you can create your own to suit your editing style.

The following discussion of the built-in snippets should help illustrate the power of the snippets system.

Note

You can also use snippets in the text entry fields in the *Search & replace* panel, however, placeholders (using `Control+J` to jump around) will not work.

The built-in snippets

The built-in snippets are described below. Note that you can override them by creating your own snippets with the same trigger text.

Inserting filler text [Lorem]

The first built-in snippet, and the simplest is used to insert filler text into a document. The filler text is taken from [De finibus bonorum et malorum](https://en.wikipedia.org/wiki/De_finibus_bonorum_et_malorum) [https://en.wikipedia.org/wiki/De_finibus_bonorum_et_malorum] a philosophical work by Cicero (translated to English). To use it simply type Lorem in an HTML file and press Control+J. It will be replaced by a couple of paragraphs of filler.

The definition of this snippet is very simple, the trigger text is defined as Lorem and the template is defined simply as the literal text to be inserted. You can easily customize it to use your favorite form of filler text.

Inserting a self-closing HTML tag [<>]

Now let's look at a simple example of the powerful concept of *placeholders*. Say you want to insert the self-closing tag `<hr/>`. Just type `<>` and press Control+J, the editor will expand the snippet to:

```
<|/>
```

Here, the symbol represents the current cursor position. You can then type `hr` and press Control+J to move the cursor to after the end of the tag. This snippet is defined as:

```
Trigger: <>
Template: <$1/>$2
```

Placeholders are simply the dollar (\$) sign followed by a number. When the snippet is expanded by pressing Control+J the cursor is positioned at the first placeholder (the placeholder with the lowest number). When you press Control+J again the cursor jumps to the next placeholder (the placeholder with the next higher number).

Inserting an HTML link tag [<a]

HTML link tags all share a common structure. They have an href attribute and some text between the opening and closing tags. A snippet to make typing them more efficient will introduce us to some more features of placeholders. To use this snippet, simply type `<a` and press `Control+J`. The editor will expand this to:

```
<
a

href
=
"filename|"
></
a
>
```

Not only that, the word `filename` will be selected, with the cursor placed over it, so that you can easily type in the real filename, using the editor's nifty [Auto-complete](#) feature. And once you are done typing the filename, press `Control+J` again and the cursor will jump to the position in between the `<a>` tags so you can easily type in the text for the link. After you are done typing the text, press `Control+J` again to jump to the point after the closing tag. This snippet is defined as:

```
Trigger: <a
Template: <a href="$${1:filename}">${2*}</a>${3}
```

There are a couple of new features here. First the `$1` placeholder has become more complex. It now includes some *default text* (the word `filename`). If a placeholder contains default text, the default text is substituted for the placeholder when the snippet is expanded. Also when you jump to a placeholder with default text using `Control+J`, the default text is selected. In this way, you can use default text to act as a reminder to you to fill in important parts of the template. You can specify default text for a placeholder by using the syntax: `${<number>:default`

text}

.

The other new feature is that the second placeholder has an asterisk after it (`${2*}`). This means that any text that was selected before expanding the template is substituted for the placeholder. To see this in action, select some text in the editor, press `Control+J`

, and press `Control+J` again, the template will be expanded to:

```
<
a
href
=
"filename"
>
whatever
text
you
selected
</
a
>
```

Inserting a HTML image tag [`<img`]

This is very similar to inserting an HTML link, as we saw above. It allows you to quickly input an `<img`

`src="filename"`

`alt="description"`

`/>`

tag and jump between the src alt attributes:

Trigger: <i
Template: \$3

Insert an arbitrary HTML tag [<<]

This allows you to insert an arbitrary full HTML tag (or wrap previously selected text in the tag). To use it, simply type << and press Control+J. The editor will expand it to:

```
<|></>
```

Type the tag name, for example: span and press Control+J, that will result in:

```
<
span
>|</
span
>
```

You will note that the closing tag has been automatically filled with span. This is achieved with yet another feature of placeholders, *mirroring*. Mirroring simply means that if you specify the sample placeholder more than once in a template, the second and all later positions will be automatically filled in with whatever you type in the first position, when you press Control+J. The definition for this snippet is:

Trigger: <<
Template: <\$1>\${2*}</\$1>\$3

As you can see, the first placeholder (\$1) has been specified twice, the second time in the closing tag, which will simply copy whatever you type in the opening tag.

Inserting an arbitrary HTML tag with a class attribute [<c]

This is very similar to the insert arbitrary tag example above, except that it assumes that you want to specify a class for the tag:

Trigger: <c
Template: <\$1 class="\$2:classname">\${3*}</\$1>\$4

This will allow you to first type the tag name, press Control+J, type the class name, press Control+J, type the contents of the tag and press Control+J one last time to jump out of the tag. The closing tag will be auto-filled.

Creating your own snippets

Snippets really shine because you can create your own to suit your editing style. To create your own snippets go to *Edit* → *Preferences* → *Editor settings* → *Manage snippets* in the editor. This will pop-up an easy to use dialog to help you create your own snippets. Simply click the *Add snippet* button and you will see a dialog that looks like:

Create a snippet
For help with snippets, see the [User Manual](#)

Name:

Trigger:

Template:

File types: ☒ All ☐ css ☐ html ☐ javascript ☐ text ☐ xml

Test:

First give your snippet a name, something descriptive, to help identify the snippet in the future. Then specify the *trigger*. A trigger is simply the text that you have to type in the editor before pressing Control+J in order to expand the snippet.

Then specify the snippet template. You should start with one of the examples above and modify it to suit your needs. Finally, specify which file types you want the snippet to be active for. This way you can have multiple snippets with the same trigger text that work differently in different file types.

The next step is to test your newly created snippet. Use the *Test* box at the bottom. Type in the trigger text and press Control+J to expand the snippet and jump between placeholders.

The calibre Content server

The calibre *Content server* allows you to access your calibre libraries and read books directly in a browser on your favorite mobile phone or tablet device. As a result, you do not need to install any dedicated book reading/management apps on your phone. Just use the browser. The server downloads and stores the book you are reading in an off-line cache so that you can read it even when there is no internet connection.

Contents

- [Accessing the Content server from other devices](#)
 - [Accessing the server from devices on your home network](#)
 - [Accessing the server from anywhere on the internet](#)
- [The server interface](#)
 - [The book list](#)
 - [The book reader](#)
- [Browser support](#)
- [Enabling offline support](#)
- [Managing user accounts from the command-line only](#)
- [Integrating the calibre Content server into other servers](#)
 - [Using a full virtual host](#)
 - [Using a URL prefix](#)
- [Creating a service for the calibre server on a modern Linux system](#)

To start the server, click the *Connect/share* button and choose *Start Content server*. You might get a message from your computer's firewall or anti-virus program asking if it is OK to allow access

. Click to the calibre.exe
or Allow
OK

button. Then open a browser (preferably Chrome or Firefox) in your computer and type in the following address:

<http://127.0.0.1:8080>

This will open a page in the browser showing you your calibre libraries, click on any one and browse the books in it. Click on a book, and it will show you all the metadata about the book, along with buttons to *Read book* and *Download book*. Click the *Read book* button to start reading the book.

Note

The address used above <http://127.0.0.1:8080> will only work on the computer that is running calibre. To access the server from other computers/phones/tablets/etc. you will need to do a little more work, as described in the next section.

Accessing the Content server from other devices

There are two types of remote device access that you will typically need. The first, simpler kind is from within your home network. If you are running calibre on a computer on your home network and you have also connected your other devices to the same home network, then you should be easily able to access the server on those devices.

Accessing the server from devices on your home network

After starting the server in calibre as described above, click the *Connect/share* button again. Instead of the *Start Content server* action, you should see a *Stop Content server* action instead. To the right of this action will be listed an IP address and port number. These look like a bunch of numbers separated by periods. For example:

```
Stop  
  
Content  
  
server  
  
[  
192.168.1.5  
,  
port  
8080  
]
```

These numbers tell you what address to use to connect to the server in your devices. Following the example above, the address becomes:

```
http
:
//
192.168.1.5
:
8080
```

The first part of the address is always `http://` the next part is the IP address, which is the numbers before the comma and finally we have the port number which must be added to the IP address with a colon (:). If you are lucky, that should be all you need and you will be looking at the calibre libraries on your device. If not, read on.

Trouble-shooting the home network connection

If you are unable to access the server from your device, try the following steps:

1. Check that the server is running by opening the address `http://127.0.0.1:8080` in a browser running on the same computer as the server.
2. Check that your firewall/anti-virus is allowing connections to your computer on the port 8080 and to the calibre program. The easiest way to eliminate the firewall/anti-virus as the source of problems is to temporarily turn them both off and then try connecting. You should first disconnect from the internet, before turning off the firewall, to keep your computer safe.
3. Check that your device and computer are on the same network. This means they should both be connected to the same wireless router. In particular neither should be using a cellular or ISP provided direct-WiFi connection.
4. If you have non-standard networking setup, it might be that the IP address shown on the *Connect/share* menu is incorrect. In such a case you will have to figure out what the correct IP address to use is, yourself. Unfortunately, given the infinite diversity of network configurations possible, it is not possible to give you a roadmap for doing so.

5. If you have setup a username and password, first try it without that to see if it is causing issues. Some e-ink devices have browsers that do not handle authentication. You can sometimes work around this by including the username and password in the URL, for example: `http://username:password@192.168.1.2:8080`.
6. If you are stuck, you can always ask for help in the [calibre user forums](https://www.mobileread.com/forums/forumdisplay.php?f=166) [<https://www.mobileread.com/forums/forumdisplay.php?f=166>].

Accessing the server from anywhere on the internet

Warning

Before doing this you should turn on username/password protection in the server, otherwise anyone in the world will be able to access your books. Go to *Preferences* → *Sharing* → *Sharing over the net* and enable the option to *Require username and password to access the content server*.

While the particular details on setting up internet access vary depending on the network configuration and type of computer you are using, the basic schema is as follows.

1. Find out the external IP address of the computer you are going to run the server on. You can do that by visiting the site [What is my IP address](https://www.whatismyip.com/) [<https://www.whatismyip.com/>] in a browser running on the computer.
2. If the computer is behind a router, enable port forwarding on the router to forward the port 8080 (or whatever port you choose to run the calibre Content server on) to the computer.
3. Make sure the calibre server is allowed through any firewalls/anti-virus programs on your computer.
4. Now you should be able to access the server on any internet-connected device using the IP address you found in the first step. For example, if the IP address you found was 123.123.123.123 and the port you are using for the calibre server is 8080, the address to use on your device becomes: `http://123.123.123.123:8080`.
5. Optionally, use a service like [no-ip](https://www.noip.com/free) [<https://www.noip.com/free>] to setup an easy to remember address to use instead of the IP address you found in the first step.

Note

For maximum security, you should also enable HTTPS on the Content server. You can either do so directly in the server by providing the path to the HTTPS certificate to use in the advanced configuration options for the server, or you can setup a reverse proxy as described below, to use an existing HTTPS setup.

The server interface

The server interface is a simplified version of the main calibre interface, optimised for use with touch screens. The home screen shows you books you are currently reading as well as allowing to choose a calibre library you want to browse. The server in calibre gives you access to all your libraries, not just a single one, as before.

The book list

The server book list is a simple grid of covers. Tap on a cover to see the detailed metadata for a book, or to read the book. If you prefer a more detailed list, you can change the default view by clicking the three vertical dots in the top right corner.

Sorting and searching of the book list should be familiar to calibre users. They can be accessed by clicking their icons in the top right area. They both work exactly the same as in the main calibre program. The search page even allows you to construct search queries by clicking on authors/tags/etc., just as you can using the Tag browser in the main program.

A much loved feature of the main program, *Virtual libraries* is present in the server interface as well. Click the three vertical dots in the top right corner to choose a Virtual library.

The book reader

You can read any book in your calibre library by simply tapping on it and then tapping the *Read book* button. The books reader is very simple to operate. You can both tap and swipe to turn pages. Swiping up/down skips between chapters. Tapping the top quarter of the screen gets you the detailed controls and viewer preferences.

If you leave the Content server running, you can even open the same book on multiple devices and it will remember your last read position. If it does not you can force a sync by tapping in the top quarter and choosing *Sync*.

Browser support

The new calibre server makes lots of use of advanced HTML 5 and CSS 3 features. As such it requires an up-to-date browser to use. It has been tested on Android Chrome and iOS Safari as well as Chrome and Firefox on the desktop.

The server is careful to use functionality that has either been already standardised or is on the standards track. As such if it does not currently work with your favorite browser, it probably will once that browser has caught up.

If you are using a particularly old or limited browser or you don't like to run JavaScript, you can use the *mobile* view, by simply adding */mobile* to the server address.

Note

On iOS, Apple allows only a single browser engine, so Firefox, Chrome and Safari are all actually the same browser under the hood. The new server interface requires iOS 10.3.2 or newer. On Android, the server has been tested with Chrome version 58 and newer.

Enabling offline support

Browser makers have been trying to force people to use SSL by disabling advanced features in their browsers for plain HTTP connections. One such casualty is offline support. So you may need to enable HTTPS on the server to get offline support working. In addition, in Firefox on Android, you will need to type `about:config` and search for `browser.tabs.useCache` and toggle it to `true`.

Managing user accounts from the command-line only

The calibre program has a nice section in *Preferences* to allow you to manage user accounts for the server. However, if you want to run the standalone server and cannot run the main calibre program on the same computer/user account, you can also manage users using just the command-line.

You can manage user accounts using the `--manage-users` option to the standalone `calibre-server` program. Suppose you want to store the user database in the folder `/srv/calibre`, then you create it by running:

```
calibre
-
server

--
userdb

/
srv
```

```
/
calibre
/
users
.
sqlite

--
manage
-
users
```

Just follow the prompts to create user accounts, set their permission, etc. Once you are done, you can run the server as:

```
calibre
-
server

--
userdb

/
srv
/
calibre
/
users
.
sqlite

--
enable
-
auth
```

It will use the user accounts you created in the previous step.

[Integrating the calibre Content server into other servers](#)

Here, we will show you how to integrate the calibre Content server into another server. The most common reason for this is to make use of SSL or to serve the calibre library as part of a larger site. The basic technique is to run the calibre server and setup a reverse proxy to it from the main server.

A reverse proxy is when your normal server accepts incoming requests and passes them onto the calibre server. It then reads the response from the calibre server and forwards it to the client. This means that you can simply run the calibre server as normal without trying to integrate it closely with your main server.

Using a full virtual host

The simplest configuration is to dedicate a full virtual host to the calibre server. In this case, run the calibre server as:

```
calibre
-
server
```

Now setup the virtual host in your main server, for example, for nginx:

```
http
{

client_max_body_size

64
M
;

# needed to upload large books

}

server
{
```



```
listen

[::]:
80
;

server_name

myserver
.
example
.
com
;

location

/

{

proxy_pass

http
:
//
127.0.0.1
:
8080
;

}

}
```

Or, for Apache:

LoadModule

proxy_module

modules

/

mod_proxy

.

so

LoadModule

proxy_http_module

modules

/

mod_proxy_http

.

so

<

VirtualHost

*

:

80

>

ServerName

myserver

.

example

.

com

AllowEncodedSlashes

On

```
ProxyPreserveHost
```

```
On
```

```
ProxyPass
```

```
"/"
```

```
"http://localhost:8080/"
```

```
</
```

```
VirtualHost
```

```
>
```

[Using a URL prefix](#)

If you do not want to dedicate a full virtual host to calibre, you can have it use a URL prefix. Start the calibre server as:

```
calibre
```

```
-
```

```
server
```

```
--
```

```
url
```

```
-
```

```
prefix
```

```
/
```

```
calibre
```

```
--
```

```
port
```

```
8080
```

The key parameter here is `--url-prefix`

/calibre

. This causes the Content server to serve all URLs prefixed by /calibre
. To see this in action, visit <http://localhost:8080/calibre>
in your browser. You should see the normal Content server website, but now it will run
under /calibre

With nginx, the required configuration is:

```
http {
    client_max_body_size 64M; # needed to upload large books
}

proxy_set_header X-Forwarded-For $remote_addr;
location /calibre/ {
    proxy_buffering off;
    proxy_pass http://127.0.0.1:8080$request_uri;
}
location /calibre {
    # we need a trailing slash for the Application Cache to work
    rewrite /calibre /calibre/ permanent;
}
```

For Apache, first enable the proxy modules in Apache, by adding the following to httpd.conf
:

```
LoadModule
proxy_module

modules
/
mod_proxy
.
so

LoadModule
proxy_http_module

modules
/
mod_proxy_http
```

.
so

The exact technique for enabling the proxy modules will vary depending on your Apache installation. Once you have the proxy modules enabled, add the following rules to httpd.conf (or if you are using virtual hosts to the conf file for the virtual host in question):

```
AllowEncodedSlashes On
RewriteEngine on
RewriteRule ^/calibre/(.*) http://127.0.0.1:8080/calibre/$1 [proxy]
RedirectMatch permanent ^/calibre$ /calibre/
```

That's all, you will now be able to access the calibre Content server under the /calibre URL in your main server. The above rules pass all requests under /calibre to the calibre server running on port 8080 and thanks to the --url-prefix option above, the calibre server handles them transparently.

Note

When using a reverse proxy, you should tell the calibre Content server to only listen on localhost, by using --listen-on

127.0.0.1

. That way, the server will only listen for connections coming from the same computer, i.e. from the reverse proxy.

Note

If you have setup SSL for your main server, you should tell the calibre server to use basic authentication instead of digest authentication, as it is faster. To do so, pass the --auth-mode=basic

option to calibre-server

[Creating a service for the calibre server on a modern Linux system](#)

You can easily create a service to run calibre at boot on a modern ([systemd](https://www.freedesktop.org/wiki/Software/systemd/) [https://www.freedesktop.org/wiki/Software/systemd/]) based Linux system. Just create the file /etc/systemd/system/calibre-server.service with the contents shown below:

```
[  
Unit  
]
```

```
Description  
=  
calibre
```

```
Content  
  
server
```

```
After  
=  
network  
.  
target
```

```
[  
Service  
]
```

```
Type  
=  
simple
```

```
User  
=  
mylinuxuser
```

```
Group  
=  
mylinuxgroup
```

```
ExecStart  
=/  
opt  
/  
calibre  
/  
calibre  
-
```

```
server
```

```
"/path/to/calibre library folder"
```

```
[  
Install  
]
```

```
WantedBy
```

```
=
```

```
multi
```

```
-
```

```
user
```

```
.
```

```
target
```

Change

and

mylinuxuser

mylinuxgroup

to whatever user and group you want the server to run as. This should be the same user and group that own the files in the calibre library folder. Note that it is generally not a good idea to run the server as root. Also change the path to the calibre library folder to suit your system. You can add multiple libraries if needed. See the help for the calibre-server command.

Now run:

```
sudo
```

```
systemctl
```

```
start
```

```
calibre
```

```
-
```

```
server
```

to start the server. Check its status with:

```
sudo  
  
systemctl  
  
status  
  
calibre  
-  
server
```

To make it start at boot, run:

```
sudo  
  
systemctl  
  
enable  
  
calibre  
-  
server
```

Note

The calibre server *does not* need a running X server, but it does need the X libraries installed as some components it uses link against them.

Note

The calibre server also supports systemd socket activation, so you can use that, if needed, as well.

Comparing e-books

calibre includes an integrated e-book comparison tool that can be used to see what has changed inside an e-book after editing or converting it. It can compare books in the EPUB and AZW3 formats.

To use it, either open the e-book in the tool for [Editing e-books](#) and then click *File* → *Compare to other book* or use the [Book details](#) panel. If you do a conversion from EPUB to EPUB, the original EPUB file will be saved as ORIGINAL_EPUB. Simply right click on the ORIGINAL_EPUB entry in the Book details panel and choose *Compare to EPUB format*.

The comparison tool that opens will look like the screenshot below. It shows you the differences in text, styles and images in the chosen books.



Understanding the comparison view

As can be seen in the screenshot above, the comparison view shows the differences between the two books side by side. Only the differences, with a few lines of context around them are shown. This makes it easy to see at a glance only what was changed inside a large document like a book.

Added text is shown with a green background, removed text with a red background and changed text with a blue background.

The line numbers of all changed text are shown at the sides, making it easy to go to a particular change in the editor. When you open the comparison tool from within the editor, you can also double click on a line in the right panel to go to that line in the editor automatically.

One useful technique when comparing books is to tell the comparison tool to beautify the text and style files before calculating differences. This can often result in cleaner and easier to follow differences. To do this, click the *Options* button in the bottom right and choose *Beautify files before comparing*. Note that beautifying can sometimes have undesired effects, as it can cause invalid markup to be altered to make it valid. You can also change the number of lines of context shown around differences via the *Options* button.

You can search for any text in the differences via the Search bar at the bottom. You will need to specify which panel to search, the *Left* or the *Right*.

Launching the comparison tool

The comparison tool is most useful when you have two versions of the same book and you want to see what is different between them. To that end, there are several ways to launch the tool.

Comparing two e-book files

Open the first file in the [Editing e-books](#) tool. Now click *File* → *Compare to another book* and choose the second file (it must be in the same format as the first). The comparison view will open with the file being edited on the right and the second file on the left.

Comparing the ORIGINAL_FMT to FMT

When you do a conversion in calibre from a FMT to itself, the original file is saved as ORIGINAL_FMT. You can see what was changed by the conversion, by right clicking on the ORIGINAL_FMT entry in the [Book details](#) panel in the main calibre window and selecting *Compare to FMT*. The comparison view will open with ORIGINAL_FMT on the left and FMT on the right.

Comparing a checkpoint to the current state of the book while editing

The [Editing e-books](#) tool has a very useful feature, called [Checkpoints](#). This allows you to save the current state of the book as a named *checkpoint*, to which you can revert if you do not like the changes you have made since creating the checkpoint. Checkpoints are also created automatically when you perform various automated actions in the editor. You can see the list of checkpoints by going to *View* → *Checkpoints* and then use the *Compare* button to compare the book at the selected checkpoint with the current state. The comparison tool will show the checkpoint on the left and the current state on the right.



Editing e-book metadata

Contents

- [Editing the metadata of one book at a time](#)
 - [Downloading metadata](#)
 - [Managing book formats](#)
 - [All about covers](#)
- [Editing the metadata of many books at a time](#)
 - [Search and replace](#)
 - [Bulk downloading of metadata](#)

E-books come in all shapes and sizes and more often than not, their metadata (things like title/author/series/publisher) is incomplete or incorrect. The simplest way to change metadata in calibre is to simply double click on an entry and type in the correct replacement. For more sophisticated, “power editing” use the edit metadata tools discussed below.

[Editing the metadata of one book at a time](#)

Click the book you want to edit and then click the *Edit metadata* button or press the E key. A dialog opens that allows you to edit all aspects of the metadata. It has various features to make editing faster and more efficient. A list of the commonly used tips:

- You can click the button in between title and authors to swap them automatically.
- You can click the button next to author sort to have calibre automatically fill it in using the sort values stored with each author. Use the *Manage authors* dialog to see and change the authors’ sort values. This dialog can be opened by clicking and holding the button next to author sort.
- You can click the button next to tags to use the *Tag editor* to manage the tags associated with the book.
- The “Ids” box can be used to enter an ISBN (and many other types of id), it will have a red background if you enter an invalid ISBN. It will be green for valid ISBNs.

- The author sort box will be red if the author sort value differs from what calibre thinks it should be.

Downloading metadata

The nicest feature of the edit metadata dialog is its ability to automatically fill in many metadata fields by getting metadata from various websites. Currently, calibre uses Google Books and Amazon. The metadata download can fill in Title, author, series, tags, rating, description and ISBN for you.

To use the download, fill in the title and author fields and click the *Fetch metadata* button. calibre will present you with a list of books that most closely match the title and author. If you fill in the ISBN field first, it will be used in preference to the title and author. If no matches are found, try making your search a little less specific by including only some key words in the title and only the author last name.

Managing book formats

In calibre, a single book entry can have many different *formats* associated with it. For example you may have obtained the Complete Works of Shakespeare in EPUB format and later converted it to MOBI to read on your Kindle. calibre automatically manages multiple formats for you. In the *Available formats* section of the Edit metadata dialog, you can manage these formats. You can add a new format, delete an existing format and also ask calibre to set the metadata and cover for the book entry from the metadata in one of the formats.

All about covers

You can ask calibre to download book covers for you, provided the book has a known ISBN. Alternatively you can specify a file on your computer to use as the cover. calibre can even generate a default cover with basic metadata on it for you. You can drag and drop images onto the cover to change it and also right click to copy/paste cover images.

In addition, there is a button to automatically trim borders from the cover, in case your cover image has an ugly border.

Editing the metadata of many books at a time

First select the books you want to edit by holding Ctrl or Shift and clicking on them. If you select more than one book, clicking the *Edit metadata* button will cause the *Bulk* metadata edit dialog to open. Using this dialog, you can quickly set the author/publisher/rating/tags/series etc of a bunch of books to the same value. This is particularly useful if you have just imported a number of books that have some metadata in common. This dialog is very powerful, for example, it has a *Search and replace* tab that you can use to perform bulk operations on metadata and even copy metadata from one column to another.

The normal edit metadata dialog also has *Next* and *Previous* buttons that you can use to edit the metadata of several books one after the other.

Search and replace

The *Edit metadata for many books* dialog allows you to perform arbitrarily powerful search and replace operations on the selected books. By default it uses a simple text search and replace, but it also support *regular expressions*. For more on regular expressions, see [All about using regular expressions in calibre](#).

As noted above, there are two search and replace modes: character match and regular expression. Character match will look in the *Search field* you choose for the characters you type in the *search for* box and replace those characters with what you type in the *replace with* box. Each occurrence of the search characters in the field will be replaced. For example, assume the field being searched contains *a bad cat*. If you search for *a* to be replaced with *HELLO*, then the result will be *HELLO bHELLOd cHELLOt*.

If the field you are searching on is a *multiple* field like tags, then each tag is treated separately. For example, if your tags contain *Horror*, *Scary*, the search expression *r*, will not match anything because the expression will first be applied to *Horror* and then to *Scary*.

If you want the search to ignore upper/lowercase differences, uncheck the *Case sensitive* box.

You can have calibre change the case of the result (information after the replace has happened) by choosing one of the functions from the *Apply function after replace* box. The operations available are:

- *Lower case* – change all the characters in the field to lower case
- *Upper case* – change all the characters in the field to upper case
- *Title case* – capitalize each word in the result.

The *Your test* box is provided for you to enter text to check that search/replace is doing what you want. In the majority of cases the book test boxes will be sufficient, but it is possible that there is a case you want to check that isn't shown in these boxes. Enter that case into *Your test*.

Regular expression mode has some differences from character mode, beyond (of course) using regular expressions. The first is that functions are applied to the parts of the string matched by the search string, not the entire field. The second is that functions apply to the replacement string, not to the entire field.

The third and most important is that the replace string can make reference to parts of the search string by using backreferences. A backreference is `\\n` where n is an integer that refers to the n'th parenthesized group in the search expression.

For example, given the same example as above, *a bad cat*, a search expression *a (...) (...)*, and a replace expression *a \2 \1*, the result will be *a cat bad*. Please see the [All about using regular expressions in calibre](#) for more information on backreferences.

One useful pattern: assume you want to change the case of an entire field. The easiest way to do this is to use character mode, but lets further assume you want to use regular expression mode. The search expression should be *(.*)* the replace expression should be *\1*, and the desired case change function should be selected.

Finally, in regular expression mode you can copy values from one field to another. Simply make the source and destination field different. The copy can replace the destination field, prepend to the field (add to the front), or append to the field (add at the end). The 'use comma' checkbox tells calibre to (or not to) add a comma between the text and the destination field in prepend and append modes. If the destination is multiple (e.g., tags), then you cannot uncheck this box.

Search and replace is done after all the other metadata changes in the other tabs are applied. This can lead to some confusion, because the test boxes will show the information before the other changes, but the operation will be applied after the other changes. If you have any doubts about what is going to happen, do not mix search/replace with other changes.

Bulk downloading of metadata

If you want to download the metadata for multiple books at once, right-click the *Edit metadata* button and select *Download metadata*. You can choose to download only metadata, only covers, or both.

Frequently Asked Questions

Contents

- [E-book format conversion](#)
- [Device integration](#)
- [Library management](#)
- [Miscellaneous](#)

[E-book format conversion](#)

Contents

- [What formats does calibre support conversion to/from?](#)
- [What are the best source formats to convert?](#)
- [I converted a PDF file, but the result has various problems?](#)
- [How do I convert my file containing non-English characters, or smart quotes?](#)
- [What's the deal with Table of Contents in MOBI files?](#)
- [The covers for my MOBI files have stopped showing up in Kindle for PC/Kindle for Android/iPad etc.](#)
- [How do I convert a collection of HTML files in a specific order?](#)
- [The EPUB I produced with calibre is not valid?](#)
- [How do I use some of the advanced features of the conversion tools?](#)

[What formats does calibre support conversion to/from?](#)

calibre supports the conversion of many input formats to many output formats. It can convert every input format in the following list, to every output format.

Input Formats: AZW, AZW3, AZW4, CBZ, CBR, CB7, CBC, CHM, DJVU, DOCX, EPUB, FB2, FBZ, HTML, HTMLZ, LIT, LRF, MOBI, ODT, PDF, PRC, PDB, PML, RB, RTF, SNB, TCR, TXT, TXTZ

Output Formats: AZW3, EPUB, DOCX, FB2, HTMLZ, OEB, LIT, LRF, MOBI, PDB, PMLZ, RB, PDF, RTF, SNB, TCR, TXT, TXTZ, ZIP

Note

PRC is a generic format, calibre supports PRC files with TextRead and MOBIBook headers. PDB is also a generic format. calibre supports eReader, Plucker (input only), PML and zTxt PDB files. DJVU support is only for converting DJVU files that contain embedded text. These are typically generated by OCR software. MOBI books can be of two types Mobi6 and KF8. calibre fully supports both. MOBI files often have .azw or .azw3 file extensions. DOCX files from Microsoft Word 2007 and newer are supported.

What are the best source formats to convert?

In order of decreasing preference: LIT, MOBI, AZW, EPUB, AZW3, FB2, FBZ, DOCX, HTML, PRC, ODT, RTF, PDB, TXT, PDF

I converted a PDF file, but the result has various problems?

PDF is a terrible format to convert from. For a list of the various issues you will encounter when converting PDF, see: [Convert PDF documents](#).

How do I convert my file containing non-English characters, or smart quotes?

There are two aspects to this problem:

1. Knowing the encoding of the source file: calibre tries to guess what character encoding your source files use, but often, this is impossible, so you need to tell it what encoding to use. This can be done in the GUI via the *Input character encoding* field in the *Look & feel* → *Text* section of the conversion dialog. The command-line tools have an [ebook-convert-txt-input](#) [--input-encoding](#) option.

2. When adding HTML files to calibre, you may need to tell calibre what encoding the files are in. To do this go to *Preferences → Advanced → Plugins → File type* and customize the *HTML to ZIP* plugin, telling it what encoding your HTML files are in. Now when you add HTML files to calibre they will be correctly processed. HTML files from different sources often have different encodings, so you may have to change this setting repeatedly. A common encoding for many files from the web is cp1252 and I would suggest you try that first. Note that when converting HTML files, leave the input encoding setting mentioned above blank. This is because the *HTML to ZIP* plugin automatically converts the HTML files to a standard encoding (UTF-8).

What's the deal with Table of Contents in MOBI files?

The first thing to realize is that most e-books have two tables of contents. One is the traditional Table of Contents, like the ToC you find in paper books. This Table of Contents is part of the main document flow and can be styled however you like. This ToC is called the *content ToC*.

Then there is the *metadata ToC*. A metadata ToC is a ToC that is not part of the book text and is typically accessed by some special button on a reader. For example, in the calibre E-book viewer, you use the Show Table of Contents button to see this ToC. This ToC cannot be styled by the book creator. How it is represented is up to the viewer program.

In the MOBI format, the situation is a little confused. This is because the MOBI format, alone amongst mainstream e-book formats, *does not* have decent support for a metadata ToC. A MOBI book simulates the presence of a metadata ToC by putting an *extra* content ToC at the end of the book. When you click Goto Table of Contents on your Kindle, it is to this extra content ToC that the Kindle takes you.

Now it might well seem to you that the MOBI book has two identical ToCs. Remember that one is semantically a content ToC and the other is a metadata ToC, even though both might have exactly the same entries and look the same. One can be accessed directly from the Kindle's menus, the other cannot.

When converting to MOBI, calibre detects the *metadata ToC* in the input document and generates an end-of-file ToC in the output MOBI file. You can turn this off by an option in the MOBI Output settings. You can also tell calibre whether to put it at the start or the end of the book via an option in the MOBI Output settings. Remember this ToC is semantically a *metadata ToC*, in any format other than MOBI it *cannot not be part of the text*. The fact that it is part of the text in MOBI is an accident caused by the limitations of MOBI. If you want a ToC at a particular location in your document text, create one by hand. So we strongly recommend that you leave the default as it is, i.e. with the metadata ToC at the end of the book. Also note that if you disable the generation of the end-of-file ToC the resulting MOBI

file may not function correctly on a Kindle, since the Kindle's use the metadata ToC for many things, including the Page Flip feature.

If you have a hand edited ToC in the input document, you can use the ToC detection options in calibre to automatically generate the metadata ToC from it. See the conversion section of the User Manual for more details on how to use these options.

Finally, I encourage you to ditch the content ToC and only have a metadata ToC in your e-books. Metadata ToCs will give the people reading your e-books a much superior navigation experience (except on the Kindle, where they are essentially the same as a content ToC).

Note

The newer AZW3 format has proper support for a metadata ToC. However, the Kindle firmware tends to malfunction if you disable the generation of the end-of-file inline ToC. So it is recommended that you leave the generated ToC alone. If you create an AZW3 file with a metadata ToC and no end-of-file generated ToC, some features on the Kindle will not work, such as the Page Flip feature.

[The covers for my MOBI files have stopped showing up in Kindle for PC/Kindle for Android/iPad etc.](#)

This is caused by a bug in the Amazon software. You can work around it by going to *Preferences* → *Conversion* → *Output Options* → *MOBI output* and setting the *Enable sharing of book content* option. If you are reconverting a previously converted book, you will also have to enable the option in the conversion dialog for that individual book (as per book conversion settings are saved and take precedence).

Note that doing this will mean that the generated MOBI will show up under personal documents instead of Books on the Kindle Fire and Amazon whispersync will not work, but the covers will. It's your choice which functionality is more important to you. I encourage you to contact Amazon and ask them to fix this bug.

The bug in Amazon's software is that when you put a MOBI file on a Kindle, unless the file is marked as a Personal document, Amazon assumes you bought the book from it and tries to download the cover thumbnail for it from its servers. When the download fails, it refuses to fallback to the cover defined in the MOBI file. This is likely deliberate on Amazon's part to try to force authors to sell only through them. In other words, the Kindle only displays covers for books marked as Personal Documents or books bought directly from Amazon.

If you send a MOBI file to an e-ink Kindle with calibre using a USB connection, calibre works around this Amazon bug by uploading a cover thumbnail itself. However, that workaround is only possible when using a USB connection and sending with calibre. Note that if you send using email, Amazon will automatically mark the MOBI file as a Personal Document and the cover will work, but the book will show up in Personal Documents.

How do I convert a collection of HTML files in a specific order?

In order to convert a collection of HTML files in a specific order, you have to create a table of contents file. That is, another HTML file that contains links to all the other files in the desired order. Such a file looks like:

```
<
html
>

<
body
>

<
h1
>
Table

of

Contents
</
h1
>

<
p

style
=
"text-indent:0pt"
>

<
a

href
=
```

```
"file1.html"
```

```
>
```

```
First
```

```
File
```

```
</
```

```
a
```

```
><
```

```
br
```

```
/>
```

```
<
```

```
a
```

```
href
```

```
=
```

```
"file2.html"
```

```
>
```

```
Second
```

```
File
```

```
</
```

```
a
```

```
><
```

```
br
```

```
/>
```

```
.
```

```
.
```

```
.
```

```
</
```

```
p
```

```
>
```

```
</  
body  
>  
  
</  
html  
>
```

Then, just add this HTML file to the GUI and use the *Convert* button to create your e-book. You can use the option in the Table of Contents section in the conversion dialog to control how the Table of Contents is generated.

Note

By default, when adding HTML files, calibre follows links in the files in *depth first* order. This means that if file A.html links to B.html and C.html and D.html, but B.html also links to D.html, then the files will be in the order A.html, B.html, D.html, C.html. If instead you want the order to be A.html, B.html, C.html, D.html then you must tell calibre to add your files in *breadth first* order. Do this by going to *Preferences → Advanced → Plugins → File type* and customizing the *HTML to ZIP* plugin.

[The EPUB I produced with calibre is not valid?](#)

calibre does not guarantee that an EPUB produced by it is valid. The only guarantee it makes is that if you feed it valid XHTML 1.1 + CSS 2.1 it will output a valid EPUB. calibre tries hard to ensure that EPUBs it produces actually work as intended on a wide variety of devices, a goal that is incompatible with producing valid EPUBs, and one that is far more important to the vast majority of its users. If you need a tool that always produces valid EPUBs, calibre is not for you. This means, that if you want to send a calibre produced EPUB to an online store that uses an EPUB validity checker, you have to make sure that the EPUB is valid yourself, calibre will not do it for you – in other words you must feed calibre valid XHTML + CSS as the input documents.

[How do I use some of the advanced features of the conversion tools?](#)

You can get help on any individual feature of the converters by mousing over it in the GUI or
running ebook-convert

dummy.html

.epub

-h

at a terminal. A good place to start is to look at the following demo file that demonstrates some of the advanced features [html-demo.zip](https://calibre-ebook.com/downloads/html-demo.zip) [https://calibre-ebook.com/downloads/html-demo.zip].

Device integration

Contents

- [What devices does calibre support?](#)
- [How can I help get my device supported in calibre?](#)
- [My device is not being detected by calibre?](#)
- [My device is non-standard or unusual. What can I do to connect to it?](#)
- [How do I use calibre with my iPad/iPhone/iPod touch?](#)
- [How do I use calibre with my Android phone/tablet or Kindle Fire HD?](#)
- [Can I access my calibre books using the web browser in my Kindle or other reading device?](#)
- [I cannot send emails using calibre?](#)
- [My device is getting mounted read-only in Linux, so calibre cannot connect to it?](#)
- [Why does calibre not support collections on the Kindle or shelves on the Nook?](#)
- [I am getting an error when I try to use calibre with my Kobo Touch/Glo/etc.?](#)
- [Covers for books I send to my e-ink Kindle show up momentarily and then are replaced by a generic cover?](#)
- [I transferred some books to my Kindle using calibre and they did not show up?](#)

What devices does calibre support?

calibre can directly connect to all the major (and most of the minor) e-book reading devices, smartphones, tablets, etc. In addition, using the *Connect to folder* function you can use it with any e-book reader that exports itself as a USB disk. Finally, you can connect wirelessly to any device that has a web browser using the calibre Content server.

How can I help get my device supported in calibre?

If your device appears as a USB disk to the operating system, adding support for it to calibre is very easy. We just need some information from you:

- Complete list of e-book formats that your device supports.
- Is there a special folder on the device in which all e-book files should be placed? Also does the device detect files placed in sub-folders?
- We also need information about your device that calibre will collect automatically. First, if your device supports SD cards, insert them. Then connect your device to the computer. In calibre go to *Preferences* → *Miscellaneous* and click the “Debug device detection” button. This will create some debug output. Copy it to a file and repeat the process, this time with your device disconnected from your computer.
- Send both the above outputs to us with the other information and we will write a device driver for your device.

Once you send us the output for a particular operating system, support for the device in that operating system will appear in the next release of calibre. To send us the output, open a bug report and attach the output to it. See [how to report bugs](https://calibre-ebook.com/bugs) [https://calibre-ebook.com/bugs].

[My device is not being detected by calibre?](#)

Follow these steps to find the problem:

- Make sure that you are connecting only a single device to your computer at a time. Do not have another calibre supported device like an iPhone/iPad etc. at the same time.
- If you are connecting an Apple iDevice (iPad, iPod Touch, iPhone), Apple no longer allows third party software to connect to their devices using a USB cable. Instead use a wireless connection, via the calibre Content server.
- If you are connecting a Kindle Fire or other Android device, read the note under [Using a USB cable](#).
- On macOS if you get permission errors when connecting a device to calibre, you can fix that by looking under *System Preferences* > *Security and Privacy* > *Privacy* > *Files and Folders*.
- Make sure you are running the latest version of calibre (currently 5.36.0). The latest version can always be downloaded from [the calibre website](https://calibre-ebook.com/download) [https://calibre-ebook.com/download]. You can tell what version of calibre you are currently running by looking at the bottom line of the main calibre window.

- Ensure your operating system is seeing the device. That is, the device should show up in Windows Explorer (in Windows) or Finder (in macOS).
- In calibre, go to *Preferences* → *Ignored Devices* and check that your device is not being ignored
- If all the above steps fail, go to *Preferences* → *Miscellaneous* and click *Debug device detection* with your device attached and post the output as a ticket on [the calibre bug tracker](https://bugs.launchpad.net/calibre) [https://bugs.launchpad.net/calibre].

My device is non-standard or unusual. What can I do to connect to it?

In addition to the *Connect to folder* function found under the *Connect/share* button, calibre provides a User

defined

device plugin that can be used to connect to any USB device that shows up as a disk drive in your operating system. Note: on Windows, the device must have a drive letter for calibre to use it. See the device plugin Preferences

->

Plugins

->

Device

plugins

->

User

defined
and

Preferences

->

Miscellaneous

->

Get

information

to

setup

the

user

defined

device

for more information. Note that if you are using the user defined plugin for a device normally detected by a builtin calibre plugin, you must disable the builtin plugin first, so that your user defined plugin is used instead.

[How do I use calibre with my iPad/iPhone/iPod touch?](#)

An easy way to browse your calibre collection from your Apple device is by using [The calibre Content server](#), which makes your collection available over the net. First perform the following steps in calibre

- Set the Preferred Output Format in calibre to EPUB (The output format can be set under *Preferences → Interface → Behavior*)
- Set the output profile to iPad (this will work for iPhone/iPods as well), under *Preferences → Conversion → Common options → Page setup*
- Convert the books you want to read on your iDevice to EPUB format by selecting them and clicking the *Convert* button.
- Turn on the Content server by clicking the *Connect/share* button and leave calibre running. You can also tell calibre to automatically start the Content server via *Preferences → Sharing → Sharing over the net*.

The Content server allows you to read books directly in Safari itself. In addition, there are many apps for your iDevice that can connect to the calibre Content server. Examples include: Marvin, Mapleread and iBooks itself.

Using the Content server

Start the Safari browser and type in the IP address and port of the computer running the calibre server, like this:

```
http
:
//
192.168.1.2
:
8080
/
```

Replace 192.168.1.2
with the local IP address of the computer running calibre. See [The calibre Content server](#) for details on running the server and finding out the right IP address to use.

You will see a list of books in Safari, tap on any book and you will be given the option to either download it, or read it in the browser itself. If you choose to download it, Safari will ask you if you want to open it with iBooks.

Many reading apps support browsing the calibre library directly via its [OPDS](#) support. In such apps you can go to the online catalog screen and add the IP address of the calibre server to browse and download books from your calibre library within the app.

[How do I use calibre with my Android phone/tablet or Kindle Fire HD?](#)

There are two ways that you can connect your Android device to calibre. Using a USB cable – or wirelessly, over the air. The first step to using an Android device is installing an e-book reading application on it. There are many free and paid e-book reading applications for Android: Some examples (in no particular order): [FBReader](#)
[<https://play.google.com/store/apps/details?id=org.geometerplus.zlibrary.ui.android&hl=en>], [Moon+](#)
[<https://play.google.com/store/apps/details?id=com.flyersoft.moonreader&hl=en>], [Mantano](#)
[<https://play.google.com/store/apps/details?id=com.mantano.reader.android.lite&hl=en>], [Aldiko](#)
[<https://play.google.com/store/apps/details?id=com.aldiko.android&hl=en>], [Kindle](#)
[https://play.google.com/store/apps/details?id=com.amazon.kindle&feature=related_apps].

Using a USB cable

Simply plug your device into the computer with a USB cable. calibre should automatically detect the device and then you can transfer books to it by clicking the *Send to device* button. Note that on macOS and Linux only a single program can connect to an Android device at a time, so make sure the device is not opened in the OS File manager, or the Android File Transfer utility, etc.

Note

With newer Android devices, you might have to jump through a few hoops to get the connection working, as Google really does not want you to be independent of its cloud. First, unlock the screen before plugging in the USB cable. When you plug in the USB cable you will get a popup notification. Make sure it says something like “Transferring Media files” or “MTP (Media Transfer mode)”. If it does not, tap the notification, and change the mode to Media Transfer (MTP). You may need to restart calibre at this point in order for your device to be recognized. Finally, you might get a popup on the device every time calibre or the operating system actually tries to connect to it, asking for permission, tap OK.

Note

With the Kindle Fire 8 or newer there is an icon that shows up when the USB cable is plugged in, showing that the device is charging. Tap that and switch the device to data transfer mode, and then start calibre, it should then be detected.

Over the air

calibre has a builtin web server, the [The calibre Content server](#). It makes your calibre collection available over the net. You can browse it on your device using a simple browser or a dedicated application. First perform the following steps in calibre:

- Set the *Preferred Output Format* in calibre to EPUB for normal Android devices or MOBI for Kindles (The output format can be set under *Preferences* → *Interface* → *Behavior*)
- Convert the books you want to read on your device to EPUB/MOBI format by selecting them and clicking the *Convert* button.
- Turn on the *Content server* in calibre’s preferences and leave calibre running.

Now on your Android device, open the browser and browse to

<http://192.168.1.2:8080/>

Replace 192.168.1.2
with the local IP address of the computer running calibre. See [The calibre Content server](#) for details on running the server and finding out the right IP address to use.

You can now browse your book collection and download books from calibre to your device to open with whatever e-book reading software you have on your Android device.

Many reading apps support browsing the calibre library directly via its [OPDS](#) support. In such apps you can go to the online catalog screen and add the IP address of the calibre server to browse and download books from your calibre library within the app.

Can I access my calibre books using the web browser in my Kindle or other reading device?

calibre has a *Content server* that exports the books in calibre as a web page. See [The calibre Content server](#) for details.

Some devices, like the Kindle (1/2/DX), do not allow you to access port 8080 (the default port on which the content server runs). In that case, change the port in the calibre Preferences to 80. (On some operating systems, you may not be able to run the server on a port number less than 1024 because of security settings. In this case the simplest solution is to adjust your router to forward requests on port 80 to port 8080).

Also some devices do not have browsers advanced enough to run the app-like interface used by the Content server. For such devices, you can simply add `/mobile` to the server URL to get a simplified, non-JavaScript interface.

I cannot send emails using calibre?

Because of the large amount of spam in email, sending email can be tricky, as different mail servers use different strategies to block email. The most common problem is if you are sending email directly (without a mail relay) in calibre. Many servers (for example, Amazon) block email that does not come from a well known relay. The most robust way to setup email sending in calibre is to do the following:

- Create a free GMX account at [GMX](https://www.gmx.com) [https://www.gmx.com].
- Goto *Preferences* → *Sharing* → *Sharing books by email* in calibre and click the *Use GMX* button and fill in the information asked for.
- Log into your GMX account on the website and enable SMTP sending (*Settings->POP3 & IMAP->Send and receive emails via external program*)
- calibre will then be able to use GMX to send the mail.
- If you are sending to your Kindle, remember to update the email preferences on your Amazon Kindle page to allow email sent from your GMX email address. Also note that Amazon does not allow email delivery of AZW3 and new style (KF8) MOBI files. Finally, Amazon recently started sending confirmation emails that you have to click on back to your GMX account before the book is actually delivered. They prefer you use Gmail to avoid the confirmation emails. See the note below for setting up Gmail to work with calibre.

Even after doing this, you may have problems. One common source of problems is that some poorly designed antivirus programs block calibre from opening a connection to send email. Try adding an exclusion for calibre in your antivirus program.

Note

Microsoft/Google/GMX can disable your account if you use it to send large amounts of email. So, when using these services to send mail calibre automatically restricts itself to sending one book every five minutes. If you don't mind risking your account being blocked you can reduce this wait interval by going to *Preferences* → *Advanced* → *Tweaks* in calibre.

Note

Google recently deliberately broke their email sending protocol (SMTP) support in an attempt to force everyone to use their web interface so they can show you more ads. They are trying to claim that SMTP is insecure, that is incorrect and simply an excuse. If you have trouble with Gmail you will need to [setup an app password](https://support.google.com/accounts/answer/185833) [https://support.google.com/accounts/answer/185833]. Use this app password as the password for Gmail in the calibre settings.

Note

If you are concerned about giving calibre access to your email account, simply create a new free email account with GMX or Outlook and use it only for calibre.

[My device is getting mounted read-only in Linux, so calibre cannot connect to it?](#)

Linux kernels mount devices read-only when their filesystems have errors. You can repair the filesystem with:

```
sudo  
  
fsck  
.  
vfat  
-  
y  
  
/  
dev  
/  
sdc
```

Replace `/dev/sdc` with the path to the device node of your device. You can find the device node of your device, which will always be under `/dev` by examining the output of:

mount

Why does calibre not support collections on the Kindle or shelves on the Nook?

Neither the Kindle nor the Nook provide any way to manipulate collections over a USB connection. If you really care about using collections, I would urge you to sell your Kindle/Nook and get a Kobo. Only Kobo seems to understand that life is too short to be entering collections one by one on an e-ink screen 😊

Note that in the case of the Kindle, there is a way to manipulate collections via USB, but it requires that the Kindle be rebooted *every time* it is disconnected from the computer, for the changes to the collections to be recognized. As such, it is unlikely that any calibre developers will ever feel motivated enough to support it. There is however, a calibre plugin that allows you to create collections on your Kindle from the calibre metadata. It is available [from here](https://www.mobileread.com/forums/showthread.php?t=244202) [https://www.mobileread.com/forums/showthread.php?t=244202].

Note

Amazon have removed the ability to manipulate collections completely in their newer models, like the Kindle Touch and Kindle Fire, making even the above plugin useless, unless you root your Kindle and install custom firmware.

I am getting an error when I try to use calibre with my Kobo Touch/Glo/etc.?

The Kobo has very buggy firmware. Connecting to it has been known to fail at random. Certain combinations of motherboard, USB ports/cables/hubs can exacerbate this tendency to fail. If you are getting an error when connecting to your touch with calibre try the following, each of which has solved the problem for *some* calibre users.

- Connect the Kobo directly to your computer, not via USB Hub
- Try a different USB cable and a different USB port on your computer
- Log out of the Kobo and log in again, this causes it to rebuild the database, fixing corrupted database errors.
- Try upgrading the firmware on your Kobo Touch to the latest
- Try resetting the Kobo (sometimes this cures the problem for a little while, but then it re-appears, in which case you have to reset again and again)

- Try only putting one or two books onto the Kobo at a time and do not keep large collections on the Kobo

Covers for books I send to my e-ink Kindle show up momentarily and then are replaced by a generic cover?

This happens because of an Amazon bug. They try to download a cover for the book from their servers and when that fails, they replace the existing cover that calibre created with a generic cover. For details see [this forum thread](https://www.mobileread.com/forums/showthread.php?t=329945) [https://www.mobileread.com/forums/showthread.php?t=329945]. As of version 4.17, calibre has a workaround, where if you connect the Kindle to calibre after the covers have been destroyed by Amazon, calibre will restore them automatically. So in order to see the covers on your Kindle, you have to:

1. Send the book to the Kindle with calibre
2. Disconnect the Kindle and wait for Amazon to destroy the cover
3. Reconnect the Kindle to calibre

Note that this workaround only works for books sent with calibre 4.17 or later. Alternately, simply keep your Kindle in airplane mode, you don't really want Amazon knowing every book you read anyway. I encourage you to contact Amazon customer support and complain loudly about this bug. Maybe Amazon will listen.

Note

If the workaround is not working for you make sure you Kindle firmware is at least version 5.12.5, released in April 2020.

I transferred some books to my Kindle using calibre and they did not show up?

Books sent to the Kindle only show up on the Kindle after they have been *indexed* by the Kindle. This can take some time. If the book still does not show up after some time, then it is likely that the Kindle indexer crashed. Sometimes a particular book can cause the indexer to crash. Unfortunately, Amazon has not provided any way to deduce which book is causing a crash on the Kindle. Your only recourse is to either reset the Kindle, or delete all files from its memory using Windows Explorer (or whatever file manager you use) and then send the books to it again, one by one, until you discover the problem book. Once you have found the problem book, delete it off the Kindle and do a MOBI to MOBI or MOBI to AZW3 conversion in calibre and then send it back. This will most likely take care of the problem.

Library management

Contents

- [Where are the book files stored?](#)
- [How does calibre manage author names and sorting?](#)
- [Why doesn't calibre let me store books in my own folder structure?](#)
- [Why doesn't calibre have a column for foo?](#)
- [Can I have a column showing the formats or the ISBN?](#)
- [How do I move my calibre data from one computer to another?](#)
- [The list of books in calibre is blank!](#)
- [I am getting errors with my calibre library on a networked drive/NAS?](#)

Where are the book files stored?

When you first run calibre, it will ask you for a folder in which to store your books. Whenever you add a book to calibre, it will copy the book into that folder. Books in the folder are nicely arranged into sub-folders by Author and Title. Note that the contents of this folder are automatically managed by calibre, **do not** add any files/folders manually to this folder, as they may be automatically deleted. If you want to add a file associated to a particular book, use the top right area of *Edit metadata* dialog to do so. Then, calibre will automatically put that file into the correct folder and move it around when the title/author changes.

Metadata about the books is stored in the file `metadata.db` at the top level of the library folder. This file is a sqlite database. When backing up your library make sure you copy the entire folder and all its sub-folders.

The library folder and all its contents make up what is called a calibre library. You can have multiple such libraries. To manage the libraries, click the calibre icon on the toolbar. You can create new libraries, remove/rename existing ones and switch between libraries easily.

You can copy or move books between different libraries (once you have more than one library setup) by right clicking on a book and selecting the *Copy to library* action.

How does calibre manage author names and sorting?

Author names are complex, especially across cultures, see [this note](https://www.w3.org/International/questions/qa-personal-names.en.php?changelang=en) [https://www.w3.org/International/questions/qa-personal-names.en.php?changelang=en] for some of the complexities. calibre has a very flexible strategy for managing author names. The first thing to understand is that books and authors are separate entities in calibre. A book can have more than one author, and an author can have more than one book. You can manage the authors of a book by the edit metadata dialog. You can manage individual authors by right clicking on the author in the Tag browser on the left of the main calibre window and selecting

Manage authors. Using this dialog you can change the name of an author and also how that name is sorted. This will automatically change the name of the author in all the books of that author. When a book has multiple authors, separate their names using the & character.

Now coming to author name sorting:

- When a new author is added to calibre (this happens whenever a book by a new author is added), calibre automatically computes a sort string for both the book and the author.
- Authors in the Tag browser are sorted by the sort value for the **authors**. Remember that this is different from the Author sort field for a book.
- By default, this sort algorithm assumes that the author name is in First name Last name format and generates a Last name, First name sort value.
- You can change this algorithm by going to *Preferences → Advanced → Tweaks* and setting the *author_sort_copy_method* tweak.
- You can force calibre to recalculate the author sort values for every author by right clicking on any author and selecting *Manage authors*, then pushing the *Recalculate all author sort values* button. Do this after you have set the *author_sort_copy_method* tweak to what you want.
- You can force calibre to recalculate the author sort values for all books by using the bulk metadata edit dialog (select all books and click edit metadata, check the *Automatically set author sort* checkbox, then press OK).
- When recalculating the author sort values for books, calibre uses the author sort values for each individual author. Therefore, ensure that the individual author sort values are correct before recalculating the books' author sort values.

- You can control whether the Tag browser display authors using their names or their sort values by setting the *categories_use_field_for_author_name* tweak in *Preferences → Advanced → Tweaks*

Note that you can set an individual author's sort value to whatever you want using *Manage authors*. This is useful when dealing with names that calibre will not get right, such as complex multi-part names like Miguel de Cervantes Saavedra or when dealing with Asian names like Sun Tzu.

With all this flexibility, it is possible to have calibre manage your author names however you like. For example, one common request is to have calibre display author names LN, FN. To do this, and if the note below does not apply to you, then:

- Set the `author_sort_copy_method` tweak to `copy` as described above.
- Restart calibre. Do not change any book metadata before doing the remaining steps.
- Change all author names to LN, FN using the Manage authors dialog.
- After you have changed all the authors, press the *Recalculate all author sort values* button.
- Press OK, at which point calibre will change the authors in all your books. This can take a while.

Note

When changing from FN LN to LN, FN, it is often the case that the values in `author_sort` are already in LN, FN format. If this is your case, then do the following:

- Set the `author_sort_copy_method` tweak to `copy` as described above.
- Restart calibre. Do not change any book metadata before doing the remaining steps.
- Open the Manage authors dialog. Press the copy all author

sort

values

to

author
button.

- Check through the authors to be sure you are happy. You can still press Cancel to abandon the changes. Once you press OK, there is no undo.
- Press OK, at which point calibre will change the authors in all your books. This can take a while.

[Why doesn't calibre let me store books in my own folder structure?](#)

The whole point of calibre's library management features is that they provide a search and sort based interface for locating books that is *much* more efficient than any possible folder scheme you could come up with for your collection. Indeed, once you become comfortable using calibre's interface to find, sort and browse your collection, you won't ever feel the need to hunt through the files on your disk to find a book again. By managing books in its own folder structure of Author -> Title -> Book files, calibre is able to achieve a high level of reliability and standardization. To illustrate why a search/tagging based interface is superior to folders, consider the following. Suppose your book collection is nicely sorted into folders with the following scheme:

Genre

->

Author

->

Series

->

ReadStatus

Now this makes it very easy to find for example all science fiction books by Isaac Asimov in the Foundation series. But suppose you want to find all unread science fiction books. There's

no easy way to do this with this folder scheme, you would instead need a folder scheme that looks like:

ReadStatus

->

Genre

->

Author

->

Series

In calibre, you would instead use tags to mark genre and read status and then just use a simple search query like `tag:scifi`

and

not

`tag:read`

. calibre even has a nice graphical interface, so you don't need to learn its search language instead you can just click on tags to include or exclude them from the search.

To those of you that claim that you need access to the filesystem, so that you can have access to your books over the network, calibre has an excellent Content server that gives you access to your calibre library over the net.

If you are worried that someday calibre will cease to be developed, leaving all your books marooned in its folder structure, explore the powerful *Save to disk* feature in calibre that lets you export all your files into a folder structure of arbitrary complexity based on their metadata.

Finally, the reason there are numbers at the end of every title folder, is for *robustness*. That number is the id number of the book record in the calibre database. The presence of the number allows you to have multiple records with the same title and author names. It is also part of what allows calibre to magically regenerate the database with all metadata if the database file gets corrupted. Given that calibre's mission is to get you to stop storing

metadata in filenames and stop using the filesystem to find things, the increased robustness afforded by the id numbers is well worth the uglier folder names.

If you are still not convinced, then I'm afraid calibre is not for you. Look elsewhere for your book cataloguing needs. Just so we're clear, **this is not going to change**. Kindly do not contact us in an attempt to get us to change this.

[Why doesn't calibre have a column for foo?](#)

calibre is designed to have columns for the most frequently and widely used fields. In addition, you can add any columns you like. Columns can be added via *Preferences* → *Interface* → *Add your own columns*. Watch the tutorial [UI Power tips](https://calibre-ebook.com/demo#tutorials) [https://calibre-ebook.com/demo#tutorials] to learn how to create your own columns, or read [this blog post](https://blog.calibre-ebook.com/calibre-custom-columns/) [https://blog.calibre-ebook.com/calibre-custom-columns/].

You can also create “virtual columns” that contain combinations of the metadata from other columns. In the add column dialog use the *Quick create* links to easily create columns to show the book ISBN or formats. You can use the powerful calibre template language to do much more with columns. For more details, see [The calibre template language](#).

[Can I have a column showing the formats or the ISBN?](#)

Yes, you can. Follow the instructions in the answer above for adding custom columns.

[How do I move my calibre data from one computer to another?](#)

You can export all calibre data (books, settings and plugins) and then import it on another computer. First let's see how to export the data:

- Right click the calibre icon in the main calibre toolbar and select *Export/import all calibre data*. Note that if there is currently a device connected, this menu option will not be available – so, disconnect any connected devices. Then click the button labelled *Export all your calibre data*. You will see a list of all your calibre libraries. Click OK and choose an empty folder somewhere on your computer. The exported data will be saved in this folder. Simply copy this folder to your new computer and follow the instructions below to import the data.
- Install calibre on your new computer and run through the *Welcome wizard*, it does not matter what you do there, as you will be importing your old settings in the next step. You will now have an empty calibre, with just the *Getting Started* guide in your library. Once again, right click the calibre button and choose *Export/import all calibre data*. Then click the button labelled *Import previously exported data*. Select the folder with the exported data that you copied over earlier. You will now have a list of libraries you can import. Go through the list one by one, and select the new location for each library (a location is just an empty folder somewhere on your computer). Click OK. After the import

completes, calibre will restart, with all your old libraries, settings and calibre plugins.

Note

This import/export functionality is only available from calibre version 2.47 onwards. If you have an older version of calibre, or if you encounter problems with the import/export, you can just copy over your calibre library folder manually, as described in the next paragraph.

Simply copy the calibre library folder from the old to the new computer. You can find out what the library folder is by clicking the calibre icon in the toolbar. Choose the *Switch/create calibre library* action and you will see the path to the current calibre library.

Now on the new computer, start calibre for the first time. It will run the *Welcome wizard* asking you for the location of the calibre library. Point it to the previously copied folder. If the computer you are transferring to already has a calibre installation, then the *Welcome wizard* won't run. In that case, right-click the calibre icon in the toolbar and point it to the newly copied folder. You will now have two calibre libraries on your computer and you can switch between them by clicking the calibre icon on the toolbar. Transferring your library in this manner preserves all your metadata, tags, custom columns, etc.

[The list of books in calibre is blank!](#)

In order to understand why that happened, you have to understand what a calibre library is. At the most basic level, a calibre library is just a folder. Whenever you add a book to calibre, that book's files are copied into this folder (arranged into sub folders by author and title). Inside the calibre library folder, at the top level, you will see a file called metadata.db. This file is where calibre stores the metadata like title/author/rating/tags etc. for *every* book in your calibre library. The list of books that calibre displays is created by reading the contents of this metadata.db file.

There can be two reasons why calibre is showing an empty list of books:

- Your calibre library folder changed its location. This can happen if it was on an external disk and the drive letter for that disk changed. Or if you accidentally moved the folder. In this case, calibre cannot find its library and so starts up with an empty library instead. To remedy this, do a right-click on the calibre icon in the calibre toolbar and select Switch/create library. Click the little blue icon to select the new location of your calibre library and click OK. If you don't know the new location search your computer for the file metadata.db.
- Your metadata.db file was deleted/corrupted. In this case, you can ask calibre to rebuild the metadata.db from its backups. Right click the calibre icon in the calibre toolbar and select Library maintenance->Restore database. calibre will automatically rebuild metadata.db.

[I am getting errors with my calibre library on a networked drive/NAS?](#)

Do not put your calibre library on a networked drive.

A filesystem is a complex beast. Most network filesystems lack various filesystem features that calibre uses. Some don't support file locking, some don't support hardlinking, some are just flaky. Additionally, calibre is a single user application, if you accidentally run two copies of calibre on the same networked library, bad things will happen. Finally, different OSes impose different limitations on filesystems, so if you share your networked drive across OSes, once again, bad things *will happen*.

Consider using the calibre Content server to make your books available on other computers. Run calibre on a single computer and access it via the Content server or a Remote Desktop solution.

If you must share the actual library, use a file syncing tool like DropBox or rsync instead of a networked drive. If you are using a file-syncing tool it is **essential** that you make sure that both calibre and the file syncing tool do not try to access the calibre library at the same time. In other words, **do not** run the file syncing tool and calibre at the same time.

Even with these tools there is danger of data corruption/loss, so only do this if you are willing to live with that risk. In particular, be aware that **Google Drive** is incompatible with calibre, if you put your calibre library in Google Drive, **you will suffer data loss**. See [this thread](https://www.mobileread.com/forums/showthread.php?t=205581) [https://www.mobileread.com/forums/showthread.php?t=205581] for details.

[Miscellaneous](#)

Contents

- [I want calibre to download news from my favorite news website.](#)
- [Why the name calibre?](#)
- [Why does calibre show only some of my fonts on macOS?](#)
- [calibre is not starting on Windows?](#)
- [calibre freezes/crashes occasionally?](#)
- [The calibre E-book viewer and Edit book tools do not work on Windows?](#)
- [Using the viewer or doing any conversions results in a permission denied error on Windows](#)
- [calibre is not starting/crashing on macOS?](#)

- [I downloaded the installer, but it is not working?](#)
- [My antivirus program claims calibre is a virus/trojan?](#)
- [How do I backup calibre?](#)
- [How do I use purchased EPUB books with calibre \(or what do I do with .acsm files\)?](#)
- [I am getting a "Permission Denied" error?](#)
- [Can I have the comment metadata show up on my reader?](#)
- [How do I get calibre to use my HTTP proxy?](#)
- [I want some feature added to calibre. What can I do?](#)
- [Why doesn't calibre have an automatic update?](#)
- [How is calibre licensed?](#)
- [How do I run calibre from my USB stick?](#)
- [How do I run parts of calibre like news download and the Content server on my own Linux server?](#)

I want calibre to download news from my favorite news website.

If you are reasonably proficient with computers, you can teach calibre to download news from any website of your choosing. To learn how to do this see [Adding your favorite news website.](#)

Otherwise, you can request a particular news site by posting in the [calibre Recipes forum](#) [<https://www.mobileread.com/forums/forumdisplay.php?f=228>].

Why the name calibre?

Take your pick:

- Converter And LIBRARY for E-books
- A high *calibre* product
- A tribute to the SONY Librie which was the first e-ink based e-book reader
- My wife chose it ;-)

calibre is pronounced as cal-i-ber *not* ca-li-bre. If you're wondering, calibre is the British/commonwealth spelling for caliber. Being Indian, that's the natural spelling for me.

Why does calibre show only some of my fonts on macOS?

calibre embeds fonts in e-book files it creates. E-book files support embedding only TrueType and OpenType (.ttf and .otf) fonts. Most fonts on macOS systems are in .dfont format, thus they cannot be embedded. calibre shows only TrueType and OpenType fonts found on your system. You can obtain many such fonts on the web. Simply download the .ttf/.otf files and add them to the Library/Fonts folder in your home folder.

calibre is not starting on Windows?

There can be several causes for this:

- If you get an error about calibre not being able to open a file because it is in use by another program, do the following:
 - Uninstall calibre
 - Reboot your computer
 - Re-install calibre. But do not start calibre from the installation wizard.
 - Temporarily disable your antivirus program (disconnect from the Internet before doing so, to be safe)
 - Look inside the folder you chose for your calibre library. If you see a file named metadata.db, delete it.
 - Start calibre
 - From now on you should be able to start calibre normally.
- If you get an error about a Python function terminating unexpectedly after upgrading calibre, first uninstall calibre, then delete the folders (if they exist) C:\Program

Files\Calibre
and

C:\Program

Files\Calibre2
. Now re-install and you should be fine.

- If you get an error in the *Welcome wizard* on an initial run of calibre, try choosing a folder like C:\library

as the calibre library (calibre sometimes has trouble with library locations if the path contains non-English characters, or only numbers, etc.)

- Try running it as administrator (Right click on the icon and select *Run as administrator*)

If it still won't launch, start a command prompt (press the Windows key and R; then type **cmd.exe** in the Run dialog that appears). At the command prompt type the following command and press Enter:

```
calibre
-
debug
-
g
```

Post any output you see in a help message on the [Forum](https://www.mobileread.com/forums/forumdisplay.php?f=166) [https://www.mobileread.com/forums/forumdisplay.php?f=166].

[calibre freezes/crashes occasionally?](#)

There are several possible things I know of, that can cause this:

- You recently connected an external monitor or TV to your computer. In this case, whenever calibre opens a new window like the edit metadata window or the conversion dialog, it appears on the second monitor where you don't notice it and so you think calibre has frozen. Disconnect your second monitor and restart calibre.
- The following programs have been reported to cause crashes in calibre: If you are running any of these, close them before starting calibre, or uninstall them: *RoboForm, Logitech SetPoint Settings, Constant Guard Protection by Xfinity, Spybot, Killer Network Manager, Nahimic UI Interface, Acronis True Image.*
- You are using a Wacom branded USB mouse/tablet. There is an incompatibility between Wacom drivers and the graphics toolkit calibre uses. Try using a non-Wacom mouse.
- On some 64 bit versions of Windows there are security software/settings that prevent 64-bit calibre from working properly. If you are using the 64-bit version of calibre try switching to the 32-bit version.

- If the crash happens when you are trying to copy text from the calibre E-book viewer, it is most likely caused by some clipboard monitoring/managing application you have running. Turn it off and you should be fine.
- If the crashes happen specifically when you are using a file dialog, like clicking on the *Add books* button or the *Save to Disk* button, then you have some software that has installed broken Shell extensions on your computer. Known culprits include: *SpiderOak*, *odrive sync* and *Dell Backup and Recovery* and *NetDrive*. If you have one of these, uninstall them and you will be fine. You can also use the [NirSoft Shell Extension Viewer](https://www.nirsoft.net/utils/shexview.html) [https://www.nirsoft.net/utils/shexview.html] to see what shell extensions are installed on your system and disable them individually, if you don't want to uninstall the full program. Remember to use "Restart Explorer" or reboot your computer after disabling the shell extensions.

If none of the above apply to you, then there is some other program on your computer that is interfering with calibre. First reboot your computer in safe mode, to have as few running programs as possible, and see if the crashes still happen. If they do not, then you know it is some program causing the problem. The most likely such culprit is a program that modifies other programs' behavior, such as an antivirus, a device driver, something like RoboForm (an automatic form filling app) or an assistive technology like Voice Control or a Screen Reader.

The only way to find the culprit is to eliminate the programs one by one and see which one is causing the issue. Basically, stop a program, run calibre, check for crashes. If they still happen, stop another program and repeat.

The calibre E-book viewer and Edit book tools do not work on Windows?

These two programs use hardware acceleration as they embed a version of the Chrome browser to render HTML. If they do not work it will be because of incompatibility with your system's GPU (graphics) drivers. Try updating these first, and reboot. If that does not fix it, you can set the QTWEBENGINE_CHROMIUM_FLAGS environment variable to the value --disable-gpu to turn off hardware acceleration. See [this page](https://doc.qt.io/qt-5/qtwebengine-debugging.html) [https://doc.qt.io/qt-5/qtwebengine-debugging.html] for details.

Using the viewer or doing any conversions results in a permission denied error on Windows

Something on your computer is preventing calibre from accessing its own temporary files. Most likely the permissions on your Temp folder are incorrect. Go to the folder file:C:\Users\USERNAME\AppData\Local in Windows Explorer and then right click on the file:Temp folder, select *Properties* and go to the *Security* tab. Make sure that your user account has full control for this folder.

Some users have reported that running the following command in an Administrator Command Prompt fixed their permissions. To get an Administrator Command Prompt search for cmd.exe in the start menu, then right click on the command prompt entry and select *Run as administrator*. At the command prompt type the following command and press Enter:

```
icacls  
  
"  
%a  
ppdata%..\Local\Temp"  
  
/  
reset  
  
/  
T
```

Alternately, you can run calibre as Administrator, but doing so will cause some functionality, such as drag and drop to not work.

Finally, some users have reported that disabling UAC fixes the problem.

[calibre is not starting/crashing on macOS?](#)

One common cause of failures on macOS is the use of accessibility technologies that are incompatible with the graphics toolkit calibre uses. Try turning off VoiceOver if you have it on. Also go to System Preferences->System->Universal Access and turn off the setting for enabling access for assistive devices in all the tabs. Another cause can be some third party apps that modify system behavior, such as Smart Scroll.

You can obtain debug output about why calibre is not starting by running *Console.app*. Debug output will be printed to it. If the debug output contains a line that looks like:

```
Qt  
:  
  
internal  
:  
  
-  
108  
:
```

Error

ATSUMeasureTextImage

text

/

qfontengine_mac

.

mm

then the problem is probably a corrupted font cache. You can clear the cache by following these [instructions](https://www.macworld.com/article/1139383/fontcache-clear.html) [https://www.macworld.com/article/1139383/fontcache-clear.html]. If that doesn't solve it, look for a corrupted font file on your system, in ~/Library/Fonts or the like. An easy way to check for corrupted fonts in macOS is to start the "Font Book" application, select all fonts and then in the File menu, choose "Validate fonts".

I downloaded the installer, but it is not working?

Downloading from the Internet can sometimes result in a corrupted download. If the calibre installer you downloaded is not opening, try downloading it again. If re-downloading it does not work, download it from [an alternate location](https://github.com/kovidgoyal/calibre/releases/latest) [https://github.com/kovidgoyal/calibre/releases/latest]. If the installer still doesn't work, then something on your computer is preventing it from running.

- Try temporarily disabling your antivirus program (Microsoft Security Essentials, or Kaspersky or Norton or McAfee or whatever). This is most likely the culprit if the upgrade process is hanging in the middle.
- Similarly, if the installer is failing/rolling and you have Microsoft PowerToys running, quit it.
- Try rebooting your computer and running a registry cleaner like [Wise registry cleaner](https://www.wisecleaner.com) [https://www.wisecleaner.com].
- Try a clean install. That is, uninstall calibre, delete C:\Program Files\Calibre2 (or wherever you previously chose to install calibre). Then re-install calibre. Note that uninstalling does not touch your books or settings.
- Try downloading the installer with an alternate browser. For example if you are using Microsoft Edge, try using Firefox or Chrome instead.

- If you get an error about a missing DLL on Windows, then most likely, the permissions on your temporary folder are incorrect. Go to the folder C:\Users\USERNAME\AppData\Local in Windows Explorer and then right click on the Temp folder and select *Properties* and go to the *Security* tab. Make sure that your user account has full control for this folder.

If you still cannot get the installer to work and you are on Windows, you can use the [calibre portable install](https://calibre-ebook.com/download_portable) [https://calibre-ebook.com/download_portable], which does not need an installer (it is just a ZIP file).

My antivirus program claims calibre is a virus/trojan?

The first thing to check is that you are downloading calibre from the official website: <https://calibre-ebook.com/<download>>. Make sure you are clicking the download links on the left, not the advertisements on the right. calibre is a very popular program and unscrupulous people try to setup websites offering it for download to fool the unwary.

If you have the official download and your antivirus program is still claiming calibre is a virus, then, your antivirus program is wrong. Antivirus programs use heuristics, patterns of code that “look suspicious” to detect viruses. It’s rather like racial profiling. calibre is a completely open source product. You can actually browse the source code yourself (or hire someone to do it for you) to verify that it is not a virus. Please report the false identification to whatever company you buy your antivirus software from. If the antivirus program is preventing you from downloading/installing calibre, disable it temporarily, install calibre and then re-enable it.

How do I backup calibre?

The most important thing to backup is the calibre library folder, that contains all your books and metadata. This is the folder you chose for your calibre library when you ran calibre for the first time. You can get the path to the library folder by clicking the calibre icon on the main toolbar. You must backup this complete folder with all its files and sub-folders.

You can switch calibre to using a backed up library folder by simply clicking the calibre icon on the toolbar and choosing your backup library folder. A backed up library folder backs up your custom columns and saved searches as well as all your books and metadata.

If you want to backup the calibre configuration/plugins, you have to backup the config folder. You can find this config folder via *Preferences* → *Miscellaneous*. Note that restoring configuration folders is not officially supported, but should work in most cases. Just copy the contents of the backup folder into the current configuration folder to restore.

How do I use purchased EPUB books with calibre (or what do I do with .acsm files)?

Most purchased EPUB books have [DRM](#). This prevents calibre from opening them. You can still use calibre to store and transfer them to your e-book reader. First, you must authorize your reader on a Windows machine with Adobe Digital Editions. Once this is done, EPUB books transferred with calibre will work fine on your reader. When you purchase an epub book from a website, you will get an “.acsm” file. This file should be opened with Adobe Digital Editions, which will then download the actual “.epub” e-book. The e-book file will be stored in the folder “My Digital Editions”, from where you can add it to calibre.

I am getting a “Permission Denied” error?

A permission denied error can occur because of many possible reasons, none of them having anything to do with calibre.

- You can get permission denied errors if you are using an SD card with write protect enabled.
- On macOS if you get permission errors when connecting a device to calibre, you can fix that by looking under *System Preferences > Security and Privacy > Privacy > Files and Folders*.
- If you, or some program you used changed the file permissions of the files in question to read only.
- If there is a filesystem error on the device which caused your operating system to mount the filesystem in read only mode or mark a particular file as read only pending recovery.
- If the files have their owner set to a user other than you.
- If your file is open in another program.
- If the file resides on a device, you may have reached the limit of a maximum of 256 files in the root of the device. In this case you need to reformat the device/sd card referred to in the error message with a FAT32 filesystem, or delete some files from the SD card/device memory.

You will need to fix the underlying cause of the permissions error before resuming to use calibre. Read the error message carefully, see what file it points to and fix the permissions on that file or its containing folders.

Can I have the comment metadata show up on my reader?

Most readers do not support this. You should complain to the manufacturer about it and hopefully if enough people complain, things will change. In the meantime, you can insert the metadata, including comments into a “Jacket page” at the start of the e-book, by using the option to “Insert metadata as page at start of book” during conversion. The option is found

in the *Structure detection* section of the conversion settings. Note that for this to have effect you have to *convert* the book. If your book is already in a format that does not need conversion, you can convert from that format to the same format.

Another alternative is to create a catalog in e-book form containing a listing of all the books in your calibre library, with their metadata. Click-and-hold the *Convert* button to access the catalog creation tool. And before you ask, no you cannot have the catalog “link directly to” books on your reader.

How do I get calibre to use my HTTP proxy?

By default, calibre uses whatever proxy settings are set in your OS. Sometimes these are incorrect, for example, on Windows if you don't use Microsoft Edge then the proxy settings may not be up to date. You can tell calibre to use a particular proxy server by setting the `http_proxy`

and `https_proxy` environment variables. The format of the variable is: `http://username:password@servername`

you should ask your network administrator to give you the correct value for this variable. Note that calibre only supports HTTP proxies not SOCKS proxies. You can see the current proxies used by calibre in Preferences->Miscellaneous.

I want some feature added to calibre. What can I do?

You have two choices:

1. Create a patch by hacking on calibre and send it to me for review and inclusion. See [Development](https://calibre-ebook.com/get-involved) [https://calibre-ebook.com/get-involved].
2. [Open a bug requesting the feature](https://calibre-ebook.com/bugs) [https://calibre-ebook.com/bugs]. Remember that while you may think your feature request is extremely important/essential, calibre developers might not agree. Fortunately, calibre is open source, which means you always have the option of implementing your feature yourself, or hiring someone to do it for you. Furthermore, calibre has a comprehensive plugin architecture, so you might be able to develop your feature as a plugin, see [Writing your own plugins to extend calibre's functionality](#).

Why doesn't calibre have an automatic update?

For many reasons:

- *There is no need to update every week.* If you are happy with how calibre works turn off the update notification and be on your merry way. Check back to see if you want to update once a year or so. There is a check box to turn off the update notification, on the update notification itself.

- calibre downloads currently use [about 150TB of bandwidth a month](https://calibre-ebook.comdynamic/downloads) [https://calibre-ebook.comdynamic/downloads]. Implementing automatic updates would greatly increase that and end up costing thousands of dollars a month, which someone has to pay.
- If I implement a dialog that downloads the update and launches it, instead of going to the website as it does now, that would save the most ardent calibre updater, *at most five clicks a week*. There are far higher priority things to do in calibre development.
- If you really, really hate downloading calibre every week but still want to be up to the latest, I encourage you to run from source, which makes updating trivial. Instructions are [available here](#).
- There are third party automatic updaters for calibre made by calibre users in the [calibre forum](https://www.mobileread.com/forums/forumdisplay.php?f=238) [https://www.mobileread.com/forums/forumdisplay.php?f=238].

How is calibre licensed?

calibre is licensed under the GNU General Public License v3 (an open source license). This means that you are free to redistribute calibre as long as you make the source code available. So if you want to put calibre on a CD with your product, you must also put the calibre source code on the CD. The source code is available [for download](https://download.calibre-ebook.com) [https://download.calibre-ebook.com]. You are free to use the results of conversions from calibre however you want. You cannot use either code or libraries from calibre in your software without making your software open source. For details, see [The GNU GPL v3](https://www.gnu.org/licenses/gpl.html) [https://www.gnu.org/licenses/gpl.html].

How do I run calibre from my USB stick?

A portable version of calibre is available [here](https://calibre-ebook.com/download_portable) [https://calibre-ebook.com/download_portable].

How do I run parts of calibre like news download and the Content server on my own Linux server?

First, you must install calibre onto your Linux server. If your server is using a modern Linux distribution, you should have no problems installing calibre onto it.

Note

calibre needs GLIBC >= 2.18 and libstdc++ >= 6.0.21. If you have an older server, you will either need to compile these from source, or use calibre 3.48 which requires GLIBC >= 2.17 or 2.85.1 which requires GLIBC >= 2.13 or calibre 1.48 which requires only GLIBC >= 2.10. In addition, although the calibre command line utilities do not need a running X server, some of them do require the X server libraries to be installed on your system. This is because of Qt, which is used for various image processing tasks, and links against these libraries. If you get an ImportError about some Qt modules, you are likely missing some X libraries.

You can run the calibre server via the command:

```
/
opt
/
calibre
/
calibre
-
server

/
path
/
to
/
the
/
library
/
you
/
want
/
to
/
share
```

You can download news and convert it into an e-book with the command:

```
/
opt
/
calibre
/
ebook
-
convert

"Title of news source.recipe"
```

```
outputfile
.  
epub
```

If you want to generate MOBI, use `outputfile.mobi` instead and use `--output-profile`
`kindle`
.

You can email downloaded news with the command:

```
/  
opt  
/  
calibre  
/  
calibre  
-  
smtp
```

I leave figuring out the exact command line as an exercise for the reader.

Finally, you can add downloaded news to the calibre library with:

```
/  
opt  
/  
calibre  
/  
calibredb  
  
add  
  
--  
with  
-  
library  
  
/  
path
```

```
/
to
/
library

outfile
.
epub
```

Remember to read the [Command Line Interface](#) section of the calibre User Manual to learn more about these, and other commands.

Tutorials

Here you will find tutorials to get you started using calibre's more advanced features, such as XPath and templates.

- [Adding your favorite news website](#)
- [Managing subgroups of books, for example "genre"](#)
- [XPath tutorial](#)
- [The calibre template language](#)
- [All about using regular expressions in calibre](#)
- [Writing your own plugins to extend calibre's functionality](#)
- [Typesetting mathematics in e-books](#)
- [Creating AZW3 • EPUB • MOBI catalogs](#)
- [Virtual libraries](#)

Managing subgroups of books, for example “genre”

Some people wish to organize the books in their library into subgroups, similar to subfolders. The most commonly provided reason is to create genre hierarchies, but there are many others. One user asked for a way to organize textbooks by subject and course number. Another wanted to keep track of gifts by subject and recipient. This tutorial will use the genre example for the rest of this post.

Before going on, please note that we are not talking about folders on the hard disk. Subgroups are not file folders. Books will not be copied anywhere. calibre’s library file structure is not affected. Instead, we are presenting a way to organize and display subgroups of books within a calibre library.

- [Setup](#)
- [Searching](#)
- [Restrictions](#)
- [Useful template functions](#)

The commonly-provided requirements for subgroups such as genres are:

- A subgroup (e.g., a genre) must contain (point to) books, not categories of books. This is what distinguishes subgroups from calibre user categories.
- A book can be in multiple subgroups (genres). This distinguishes subgroups from physical file folders.
- Subgroups (genres) must form a hierarchy; subgroups can contain subgroups.

Tags give you the first two. If you tag a book with the genre then you can use the Tag browser (or search) to find the books with that genre, giving you the first. Many books can have the same tag(s), giving you the second. The problem is that tags don’t satisfy the third requirement. They don’t provide a hierarchy.

The calibre hierarchy feature gives you the third – the ability to see the genres in a ‘tree’ and the ability to easily search for books in genre or sub-genre. For example, assume that your genre structure is similar to the following:



Genre

.

History

..

Japanese

..

Military

..

Roman

.

Mysteries

..

English

..

Vampire

.

Science

Fiction

..

Alternate

History

..

Military

..

Space

Opera

.

Thrillers

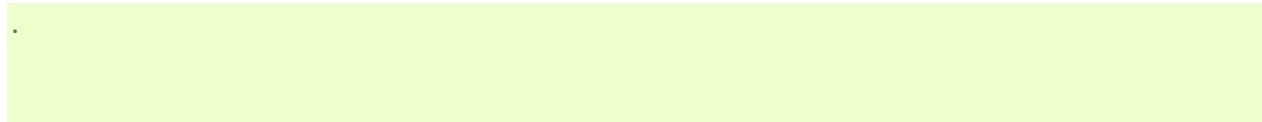
..

Crime

..

Horror

etc



By using the hierarchy feature, you can see these genres in the Tag browser in tree form, as shown in the screen image. In this example the outermost level (Genre) is a custom column that contains the genres. Genres containing sub-genres appear with a small triangle next to them. Clicking on that triangle will open the item and show the sub-genres, as you can see with History and Science Fiction.

Clicking on a genre can search for all books with that genre or children of that genre. For example, clicking on Science Fiction can give all three of the child genres, Alternate History, Military, and Space Opera. Clicking on Alternate History will give books in that genre, ignoring those in Military and Space Opera. Of course, a book can have multiple genres. If a book has both Space Opera and Military genres, then you will see that book if you click on either genre. Searching is discussed in more detail below.

Another thing you can see from the image is that the genre Military appears twice, once under History and once under Science Fiction. Because the genres are in a hierarchy, these are two separate genres. A book can be in one, the other, or (doubtfully in this case) both. For example, the books in Winston Churchill's "The Second World War" could be in "History.Military". David Weber's Honor Harrington books could be in "Science Fiction.Military", and for that matter also in "Science Fiction.Space Opera."

Once a genre exists, that is at least one book has that genre, you can easily apply it to other books by dragging the books from the library view onto the genre you want the books to have. You can also apply genres in the metadata editors; more on this below.

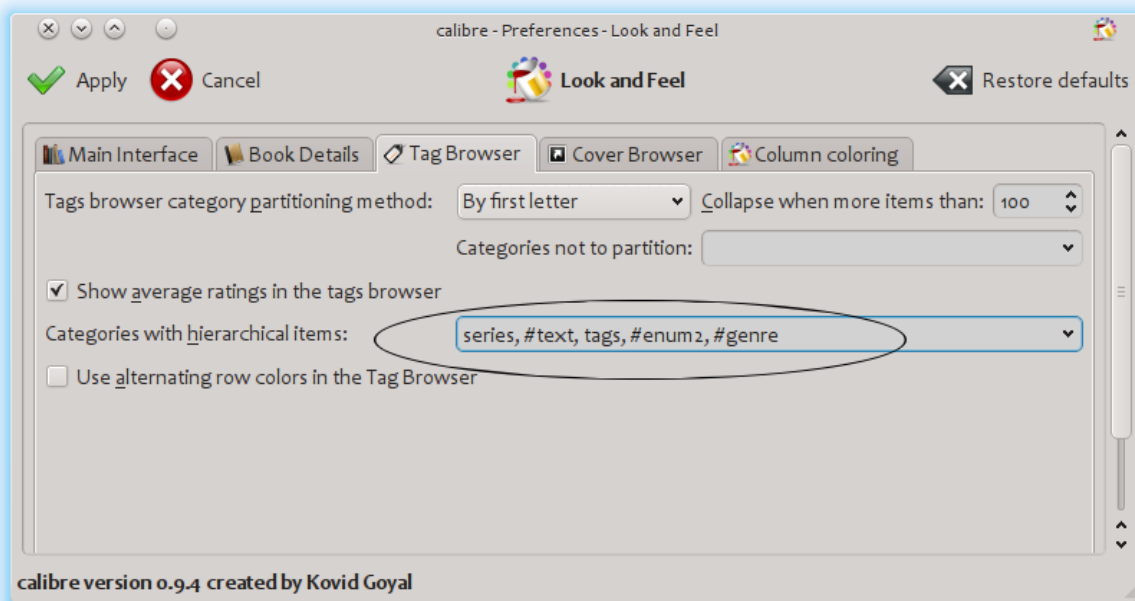
Setup

By now, your question might be "How was all of this setup?" There are three steps: 1) create the custom column, 2) tell calibre that the new column is to be treated as a hierarchy, and 3) add genres.

You create the custom column in the usual way, using Preferences -> Add your own columns. This example uses "#genre" as the lookup name and "Genre" as the column heading. The column type is "Comma-separated text, like tags, shown in the Tag browser."



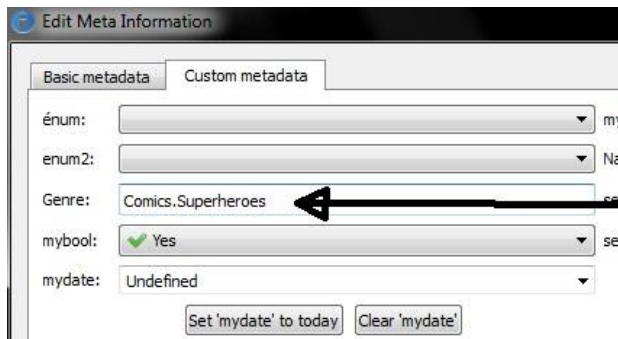
Then after restarting calibre, you must tell calibre that the column is to be treated as a hierarchy. Go to *Preferences* → *Look & feel* → *Tag browser* and enter the lookup name “#genre” into the “Categories with hierarchical items” box. Press *Apply*, and you are done with setting up.



At the point there are no genres in the column. We are left with the last step: how to apply a genre to a book. A genre does not exist in calibre until it appears on at least one book. To learn how to apply a genre for the first time, we must go into some detail about what a genre looks like in the metadata for a book.

A hierarchy of ‘things’ is built by creating an item consisting of phrases separated by periods. Continuing the genre example, these items would “History.Military”, “Mysteries.Vampire”, “Science Fiction.Space Opera”, etc. Thus to create a new genre, you pick a book that should have that genre, edit its metadata, and enter the new genre into the column you created. Continuing our example, if you want to assign a new genre “Comics” with a sub-genre

“Superheroes” to a book, you would ‘edit metadata’ for that (comic) book, choose the Custom metadata tab, and then enter “Comics.Superheroes” as shown in the following (ignore the other custom columns):



The screenshot shows the 'Edit Meta Information' dialog box with the 'Custom metadata' tab selected. The 'Genre' field is set to 'Comics.Superheroes', indicated by a black arrow pointing to it. Other fields include 'énum:' (my), 'enum2:' (Na), 'mybool:' (Yes), and 'mydate:' (Undefined). There are buttons for 'Set 'mydate' to today' and 'Clear 'mydate''.

After doing the above, you see in the Tag browser:



From here on, to apply this new genre to a book (a comic book, presumably), you can either drag the book onto the genre, or add it to the book using edit metadata in exactly the same way as done above.

Note

Hierarchical display only works if the Tag browser is set to sort items by name. This is the default and can be checked by clicking the *Configure* button at the bottom of the Tag browser.

Searching



The easiest way to search for genres is using the Tag browser, clicking on the genre you wish to see. Clicking on a genre with children will show you books with that genre and all child genres. However, this might bring up a question. Just because a genre has children doesn't mean that it isn't a genre in its own right. For example, a book can have the genre "History" but not "History.Military". How do you search for books with only "History"?

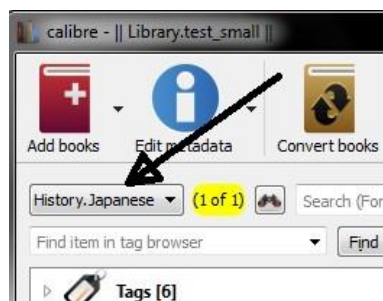
The Tag browser search mechanism knows if an item has children. If it does, clicking on the item cycles through 5 searches instead of the normal three. The first is the normal green plus, which shows you books with that genre only (e.g., History). The second is a doubled plus (shown above), which shows you books with that genre and all sub-genres (e.g., History and History.Military). The third is the normal red minus, which shows you books without that exact genre. The fourth is a doubled minus, which shows you books without that genre or sub-genres. The fifth is back to the beginning, no mark, meaning no search.

Restrictions

If you search for a genre then create a saved search for it, you can use the 'restrict to' box to create a Virtual library of books with that genre. This is useful if you want to do other searches within the genre or to manage/update metadata for books in the genre. Continuing our example, you can create a Saved search named 'History.Japanese' by first clicking on the genre Japanese in the Tag browser to get a search into the search field, entering History.Japanese into the saved search field, then pushing the "Save search" button (the green box with the white plus, on the right-hand side).



After creating the saved search, you can use it as a restriction.



Useful template functions

You might want to use the genre information in a template, such as with save to disk or send to device. The question might then be “How do I get the outermost genre name or names?” A calibre template function, `subitems`, is provided to make doing this easier.

For example, assume you want to add the outermost genre level to the save-to-disk template to make genre folders, as in “History/The Gathering Storm - Churchill, Winston”. To do this, you must extract the first level of the hierarchy and add it to the front along with a slash to indicate that it should make a folder. The template below accomplishes this:

```
{  
#genre:subitems(0,1) || /}{title} - {authors}
```

See [The template language](#) for more information about templates and the **`subitems()`** function.

XPath tutorial

In this tutorial, you will be given a gentle introduction to [XPath](https://en.wikipedia.org/wiki/XPath) [https://en.wikipedia.org/wiki/XPath], a query language that can be used to select arbitrary parts of [HTML](https://en.wikipedia.org/wiki/HTML) [https://en.wikipedia.org/wiki/HTML] documents in calibre. XPath is a widely used standard, and googling it will yield a ton of information. This tutorial, however, focuses on using XPath for e-book related tasks like finding chapter headings in an unstructured HTML document.

Contents

- [Selecting by tag name](#)
- [Selecting by attributes](#)
- [Selecting by tag content](#)
- [Sample e-book](#)
- [XPath built-in functions](#)

[Selecting by tag name](#)

The simplest form of selection is to select tags by name. For example, suppose you want to select all the `<h2>` tags in a document. The XPath query for this is simply:

```
//  
h  
:  
h2  
  
(  
Selects  
  
all  
  
<  
h2
```

```
>  
  
tags  
)
```

The prefix `//` means *search at any level of the document*. Now suppose you want to search for `<span>`

tags that are inside `<a>` tags. That can be achieved with:

```
//  
h  
:  
a  
/  
h  
:  
span  
  
(  
Selects
```

```
<  
span  
>
```

```
tags
```

```
inside
```

```
<  
a  
>
```

```
tags  
)
```

If you want to search for tags at a particular level in the document, change the prefix:

```
/
h
:
body
/
h
:
div
/
h
:
p

(
Selects
```

```
<
p
>
```

tags

that

are

children

of

```
<
div
>
```

tags

that

are

children

```
of  
  
the  
  
<  
body  
>  
  
tag  
)
```

This will match only <p>A

very

short

e-book

to

demonstrate

the

use

of

XPath.</p>

in the [Sample e-book](#) but not any of the other <p> tags. The h:

prefix in the above examples is needed to match XHTML tags. This is because internally, calibre represents all content as XHTML. In XHTML tags have a *namespace*, and h: is the namespace prefix for HTML tags.

Now suppose you want to select both <h1> and <h2>

tags. To do that, we need a XPath construct called *predicate*. A *predicate* is simply a test that is used to select tags. Tests can be arbitrarily powerful and as this tutorial progresses, you will see more powerful examples. A predicate is created by enclosing the test expression in square brackets:

```
//*
[
  name
()
=
'h1'

or

name
()
=
'h2'
]
```

There are several new features in this XPath expression. The first is the use of the wildcard *

. It means *match any tag*. Now look at the test expression `name()='h1'`

or

`name()='h2'`

. [name\(\)](#) is an example of a *built-in function*. It simply evaluates to the name of the tag. So by using it, we can select tags whose names are either *h1* or *h2*. Note that the [name\(\)](#) function ignores namespaces so that there is no need for the `h:` prefix. XPath has several useful built-in functions. A few more will be introduced in this tutorial.

Selecting by attributes

To select tags based on their attributes, the use of predicates is required:

```
//*
[
  @style
]

(
  Select

all
```

tags

that

have

a

style

attribute
)

/**

[

@class

=

"chapter"

]

(

Select

all

tags

that

have

class

=

"chapter"

)

//

h

:

h1

[

@class

=

```
"bookTitle"  
]  
  
(  
Select  
  
all  
  
h1  
  
tags  
  
that  
  
have  
  
class  
=  
"bookTitle"  
)
```

Here, the @ operator refers to the attributes of the tag. You can use some of the [XPath built-in functions](#) to perform more sophisticated matching on attribute values.

Selecting by tag content

Using XPath, you can even select tags based on the text they contain. The best way to do this is to use the power of *regular expressions* via the built-in function [re:test\(\)](#):

```
//  
h  
:  
h2  
[  
re  
:  
test  
(  
.  
,
```

```
'chapter|section'
```

```
,
```

```
'i'
```

```
)]
```

```
(  
Selects
```

```
<  
h2  
>
```

```
tags
```

```
that
```

```
contain
```

```
the
```

```
words
```

```
chapter
```

```
or
```

```
section  
)
```

Here the
operator refers to the contents of the tag, just as the @
operator referred to its attributes.

[Sample e-book](#)


```
<
html
>

<
head
>

<
title
>
A very short e-book
</
title
>

<
meta

name
=
"charset"

value
=
"utf-8"

/>

</
head
>

<
body
>
```

```
<
h1

class
=
"bookTitle"
>
A very short e-book
</
h1
>


<
p

style
=
"text-align:right"
>
Written by Kovid Goyal
</
p
>


<
div

class
=
"introduction"
>


<
p
>
A very short e-book to demonstrate the use of XPath.
</
p
>
```

```
</div>
```

```
<h2
```

```
class  
=  
"chapter"
```

```
>  
Chapter One  
</h2>
```

```
<p>
```

This is a truly fascinating chapter.

```
</p>
```

```
<h2
```

```
class  
=  
"chapter"
```

```
>  
Chapter Two  
</h2>
```

```
<
p
>
A worthy continuation of a fine tradition.
</
p
>

</
body
>

</
html
>
```

XPath built-in functions

name()

The name of the current tag.

contains()

contains(s1,

s2)

returns *true* if s1 contains s2.

re:test()

re:test(src,

pattern,

flags)

returns *true* if the string *src* matches the regular expression *pattern*. A particularly useful flag is `re.IGNORECASE`, it makes matching case insensitive. A good primer on the syntax for regular expressions can be found at [regex syntax](https://docs.python.org/library/re.html) [https://docs.python.org/library/re.html]

The calibre template language

The calibre template language is a calibre-specific language used throughout calibre for tasks such as specifying file paths, formatting values, and computing the value for user-specified columns. Examples:

- Specify the folder structure and file names when saving files from the calibre library to the disk or e-book reader.
- Define rules for adding icons and colors to the calibre book list.
- Define *virtual columns* that contain data from other columns.
- Advanced library searching.
- Advanced metadata search and replace.

The language is built around the notion of a *template*, which specifies which book metadata to use, computations on that metadata, and how it is to be formatted.

Basic templates

A basic template consists of one or more template

expressions

. A template

expression

consists of text and names in curly brackets (`{}`) that is replaced by the corresponding metadata from the book being processed. For example, the default template in calibre used for saving books to device has 4 template

expressions

:

```
{
author_sort
}
/
```

```
{
title
}
/
{
title
}

-

{
authors
}
```

For the book “The Foundation” by “Isaac Asimov” the will become:

```
Asimov
,
Isaac
/
The

Foundation
/
The

Foundation

-

Isaac

Asimov
```

The slashes are not template expressions

because they are in between in {}
. Such text is left where it appears. For example, if the template is:

```
{  
author_sort  
}
```

Some

Important

Text

```
{  
title  
}  
/  
{  
title  
}
```

-

```
{  
authors  
}
```

then for “The Foundation” the template produces:

Asimov

,

Isaac

Some

Important

Text

The

Foundation

/

The

Foundation

-

Isaac

Asimov

A template

expression

can access all the metadata available in calibre, including custom columns (columns you create yourself), by using a column's lookup

name

. To find the lookup name for a *column* (sometimes called *fields*), hover your mouse over the column header in calibre's book list. Lookup names for custom columns always begin with #
. For series type columns there is an additional field named #lookup

name_index

that is the series index for that book in the series. For example, if you have a custom series column named #myseries, there will also be a column named #myseries_index.
. The standard series column's index is named series_index.
.

In addition to the standard column based fields, you also can use:

- {formats}
- A list of formats available in the calibre library for a book
- {identifiers:select(isbn)}
- The ISBN of the book

If the metadata for the field for a given book is not defined then the field in the template is replaced by the empty string (""). For example, consider the following template:

```
{
author_sort
}
/
{
series
}
/
{
title
}

{
series_index
}
```

If Asimov's book "Second Foundation" is in the series "Foundation" then the template produces:

```
Asimov
,
Isaac
/
Foundation
/
Second
Foundation
3
```

If a series has not been entered for the book then the template produces:

```
Asimov
,
Isaac
/
Second
```

Foundation

The template processor automatically removes multiple slashes and leading or trailing spaces.

Advanced formatting

In addition to metadata substitution, templates can conditionally include additional text and control how substituted data is formatted.

Conditionally including text

Sometimes you want text to appear in the output only if a field is not empty. A common case is `{series}` and `{series_index}` where you want either nothing or the two values separated by a hyphen. calibre handles this case using a special template

expression
syntax.

For example and using the above Foundation example, assume you want the template to produce *Foundation - 3 - Second Foundation*. This template produces that output:

`{series}`

-

`{series_index}`

-

`{title}`

However, if a book has no series the template will produce - - *the title*, which is probably not what you want. Generally, people want the result be the title without the extraneous hyphens. You can accomplish this using the following template syntax:

`{field:|prefix_text|suffix_text}`

This

template

expression

says that if field has the value *XXXX* then the result will be *prefix_textXXXXsuffix_text*. If field is empty (has no value) then the result will be the empty string (nothing) because the prefix and suffix are ignored. The prefix and suffix can contain blanks.

Do not use subtemplates (`{ ... }`) or functions (see below) in the prefix or the suffix.

Using this syntax, we can solve the above no-series problem with the template:

```
{
series
}{
series_index
:
|
-
|
-
}{
title
}
```

The hyphens will be included only if the book has a series index, which it has only if it has a series. Continuing the Foundation example again, the template will produce *Foundation - 1 - Second Foundation*.

Notes:

- You must include the colon after the lookup name if you are using a prefix or a suffix.
- You must either use either no or both | characters. Using one, as in {field:|

-

}

, is not allowed.

- It is OK to provide no text for either the prefix or the suffix, such as in {series:||}

-

}

. The template {title:||}
is the same as {title}

Formatting

Suppose you want the series_index to be formatted as three digits with leading zeros. This does the trick:

```
{series_index:0>3s}
```

- Three digits with leading zeros

For trailing zeros, use:

```
{series_index:0<3s}
```

- Three digits with trailing zeros

If you use series indices with fractional values, e.g., 1.1, you might want the decimal points to line up. For example, you might want the indices 1 and 2.5 to appear as 01.00 and 02.50 so that they will sort correctly on a device that does lexical sorting. To do this, use:

```
{series_index:0>5.2f}
```

- Five characters consisting of two digits with leading zeros, a decimal point, then 2 digits after the decimal point.

If you want only the first two letters of the data, use:

```
{author_sort:.2}
```

- Only the first two letters of the author sort name

Much of the calibre template language formatting comes from Python. For more details on the syntax of these advanced formatting operations see the [Python documentation](https://docs.python.org/3/library/string.html#formatstrings) [https://docs.python.org/3/library/string.html#formatstrings].

Using templates to define custom columns

Templates can be used to display information that isn't in calibre metadata, or to display metadata differently from calibre's normal format. For example, you might want to show the ISBN

, a field that calibre does not display. You can accomplish this creating a custom column with the type *Column built from other columns* (hereafter called *composite columns*) and providing a template to generate the displayed text. The column will display the result of evaluating the template. For example, to display the ISBN, create the column and enter `{identifiers:select(isbn)}`

in the template box. To display a column containing the values of two series custom columns, separated by a comma, use `{#series1:| |,}{#series2}`

.

Composite columns can use any template option, including formatting.

Note: You cannot edit the data displayed in a composite column. Instead you edit the source columns. If you edit a composite column, for example by double-clicking it, calibre will open the template for editing, not the underlying data.

Using functions in templates - Single Function Mode

Suppose you want to display the value of a field in upper case when that field is normally in title case. You can do this using *template functions*. For example, to display the title in upper case

function, use the uppercase
as in `{title:uppercase()}`
To display it in title case, use `{title:titlecase()}`

.

Functions go into the format part of the template, after the :
and before the first |
or the closing }

if no prefix/suffix is used. If you have both a format and a function reference, the function comes after a second :

. Functions return the value of the column specified in the template, suitably modified.

The syntax for using functions is one of:

```
{
lookup_name
:
function
(
arguments
)}
```

```
{  
lookup_name  
:  
format  
:  
function  
(  
arguments  
)}
```

```
{  
lookup_name  
:  
function  
(  
arguments  
)  
|  
prefix  
|  
suffix  
}
```

```
{  
lookup_name  
:  
format  
:  
function  
(  
arguments  
)  
|  
prefix  
|  
suffix  
}
```

Function names must always be followed by opening and closing parentheses. Some functions require extra values (arguments), and these go inside the parentheses. Arguments

are separated by commas. Literal commas (commas as text, not argument separators) must be preceded by a backslash (\). The last (or only) argument cannot contain a textual closing parenthesis.

Functions are evaluated before format specifications and the prefix/suffix. See further down for an example of using both a format and a function.

Important: If you have programming experience, please note that the syntax in *Single Function Mode* is not what you expect. Strings are not quoted and spaces are significant. All arguments are considered to be constants; there are no expressions.

Do not use subtemplates (`{ ... }`) as function arguments. Instead, use [Template Program Mode](#) and [General Program Mode](#).

Some functions require regular expressions. In the template language regular expression matching is case-insensitive.

In the function documentation below, the notation `[something]*` means that something can be repeated zero or more times. The notation `[something]+` means that the something is repeated one or more times (must exist at least one time).

The functions intended for use in Single Function Mode are:

- `capitalize()`
 - returns the value with the first letter upper case and the rest lower case.
- `contains(pattern,`
`text`
`if`
`match,`
`text`
`if`
`not`
`match)`
 - checks if the value is matched by the regular expression pattern
 - . Returns text

if

match

if the pattern matches the value, otherwise returns text

if

no

match

.

- `count(separator)`
 - interprets the value as a list of items separated by separator and returns the number of items in the list. Most lists use a comma as the separator, but authors uses an ampersand (&). Examples: `{tags:count(,)}`
, `{authors:count(&)}`
.
Aliases: `count()`
, `list_count()`
- `format_number(template)`
 - interprets the value as a number and formats that number using a Python formatting template such as `{0:5.2f}`
or `{0:,d}`
or `$(0:5,.2f)`
. The formatting template must begin with `{0:`
and end with `}`
as in the above examples. Exception: you can leave off the leading “{0:” and trailing “}” if the format template contains only a format. See the template language and the [Python documentation](https://docs.python.org/3/library/string.html#formatstrings) [https://docs.python.org/3/library/string.html#formatstrings] for more examples. Returns the empty string if formatting fails.
- `human_readable()`
 - expects the value to be a number and returns a string representing that number in KB, MB, GB, etc.
- `ifempty(text)`

if

empty)

– if the value is not empty then return the value of the field, otherwise return *text if empty*.

- in_list(separator,

[

pattern,

found_val,

]*

not_found_val)

– interpret the value as a list of items separated by separator, checking the pattern against each item in the list. If the pattern matches an item then return found_val, otherwise return not_found_val. The pair pattern and found_value can be repeated as many times as desired, permitting returning different values depending on the item's value. The patterns are checked in order, and the first match is returned.

- language_strings(localize)

– return the [language names](https://www.loc.gov/standards/iso639-2/php/code_list.php) [https://www.loc.gov/standards/iso639-2/php/code_list.php] for the [language codes](https://www.loc.gov/standards/iso639-2/php/code_list.php) [https://www.loc.gov/standards/iso639-2/php/code_list.php] passed in as the value. Example: {languages:language_strings()}
. If localize is zero, return the strings in English. If localize is not zero, return the strings in the language of the current locale. Lang_codes is a comma-separated list.

- list_item(index,

separator)

– interpret the value as a list of items separated by separator, returning the 'index'th item. The first item is number zero. The last item has the index -1 as in list_item(-1,separator)
. If the item is not in the list, then the empty string is returned.

- `lookup([`

`pattern,`

`key,`

`]*`

`else_key)`
 - The patterns will be checked against the value in order. If a pattern matches then the value of the field named by `key` is returned. If no pattern matches then the value of the field named by `else_key` is returned. See ``switch`` (below).
- `lowercase()`
 - returns the value of the field in lower case.
- `rating_to_stars(use_half_stars)`
 - Returns the rating as string of star (★) characters. The value must be a number between 0 and 5. Set `use_half_stars` to 1 if you want half star characters for fractional numbers available with custom ratings columns.
- `re(pattern,`

`replacement)`
 - return the value after applying the regular expression. All instances of pattern in the value are replaced with replacement. The template language uses case insensitive [Python regular expressions](https://docs.python.org/3/library/re.html) [https://docs.python.org/3/library/re.html].
- `select(key)`
 - interpret the value as a comma-separated list of items with each item having the form id:value (the calibre identifier format). The function finds the first pair with the id equal to key and returns the corresponding value. If no id matches then the function returns the empty string.
- `shorten(left`

`chars,`

`middle`

text,

right

chars)

– Return a shortened version of the value, consisting of left

chars

characters from the beginning of the value, followed by middle

text

, followed by right

chars

characters from the end of the value. Left

chars

and right

chars

must be non-negative integers. Example: assume you want to display the title with a length of at most 15 characters in length. One template that does this is {title:shorten(9,-,5)}

. For a book with the title *Ancient English Laws in the Times of Ivanhoe* the result will be *Ancient E-anhoe*: the first 9 characters of the title, a -, then the last 5 characters. If the value's length is less than left

chars

+ right

chars

+ the length of middle

text

then the value will be returned unchanged. For example, the title *The Dome* would not be changed.

- str_in_list(separator,

[

string,

found_val,

]+

not_found_val)

– interpret the value as a list of items separated by separator then compare string against each value in the list. The string is not a regular expression. If string is equal to any item (ignoring case) then return the corresponding found_val. If string contains separators then it is also treated as a list and each subvalue is checked. The string and found_value pairs can be repeated as many times as desired, permitting returning different values depending on string's value. If none of the strings match then not_found_value is returned. The strings are checked in order. The first match is returned.

- subitems(start_index,

end_index)

– This function breaks apart lists of tag-like hierarchical items such as genres. It interprets the value as a comma-separated list of tag-like items, where each item is a period-separated list. It returns a new list made by extracting from each item the components from start_index to end_index, then merging the results back together. Duplicates are removed. The first subitem in a period-separated list has an index of zero. If an index is negative then it counts from the end of the list. As a special case, an end_index of zero is assumed to be the length of the list.

Examples:

- Assuming a #genre column containing *A.B.C*:
 - {#genre:subitems(0,1)}
returns "A"
 - {#genre:subitems(0,2)}
returns "A.B"
 - {#genre:subitems(1,0)}
returns "B.C"
- Assuming a #genre column containing "A.B.C, D.E":

- `{#genre:subitems(0,1)}`
returns "A, D"
- `{#genre:subitems(0,2)}`
returns "A.B, D.E"
- `sublist(start_index,`

`end_index,`

`separator)`
 - interpret the value as a list of items separated by separator, returning a new list made from the items from `start_index` to `end_index`. The first item is number zero. If an index is negative, then it counts from the end of the list. As a special case, an `end_index` of zero is assumed to be the length of the list.

Examples assuming that the tags column (which is comma-separated) contains "A, B,C":

- `{tags:sublist(0,1,\,)}`
returns "A"
- `{tags:sublist(-1,0,\,)}`
returns "C"
- `{tags:sublist(0,-1,\,)}`
returns "A, B"
- `swap_around_articles(separator)`
 - returns the value with articles moved to the end. The value can be a list, in which case each item in the list is processed. If the value is a list then you must provide the separator.
 - . If no separator is provided then the value is treated as being a single value, not a list. The *articles* are those used by calibre to generate the `title_sort`.
- `swap_around_comma()`
 - given a value of the form B, A, return A B

. This is most useful for converting names in LN, FN format to FN LN. If there is no comma in the value then the function returns the value unchanged.

- switch([pattern,

value,]+

else_value)

– for each pattern,

value

pair, checks if the value matches the regular expression pattern

and if so returns the associated value

. If no pattern

matches, then else_value

is returned. You can have as many pattern,

value

pairs as you wish. The first match is returned.

- test(text

if

not

empty,

text

if

empty)

– return text

if

not

empty

if the value is not empty, otherwise return text

if

empty

.

- `titlecase()`
 - returns the value of the field in title case.
- `transliterate()`
 - Return a string in a latin alphabet formed by approximating the sound of the words in the source field. For example, if the source field is Фёдор

Миха́йлович

Достоевский

this

function

returns

Fiodor

Mikhailovich

Dostoievskii

.

- `uppercase()`
 - returns the value of the field in upper case.

Using functions and formatting in the same template

Suppose you have an integer custom column `#myint` that you want displayed with leading zeros, as in `003`. One way to do this is to use a format of `0>3s`. However, by default if a number (integer or float) equals zero then the value is displayed as the empty string so zero values will produce the empty string, not `000`. If you want to see `000` values then you use both the format string and the `ifempty` function to change the empty value back to a zero. The template would be:

```
{  
#myint:0>3s:ifempty(0)}
```

Note that you can use the prefix and suffix as well. If you want the number to appear as `[003]` or `[000]`, then use the template:

```
{  
#myint:0>3s:ifempty(0)|[]|}
```


General Program Mode

General Program Mode (GPM) replaces *template expressions* with a program written in the *template language*. The syntax of the language is defined by the following grammar:

```
program
:
:=

'program:'

expression_list
expression_list
:
:=

top_expression
[
','
top_expression
]
*

top_expression
:
:=

or_expression
or_expression
```

```
:  
:=  
  
and_expression  
  
[  
  '||'  
  and_expression  
]  
*
```

```
and_expression  
  
:  
:=  
  
not_expression  
  
[  
  '&&'  
  not_expression  
]  
*
```

```
not_expression  
  
:  
:=  
  
[  
  '!'  
  not_expression  
]
```

*

|

compare_exp

compare_expr

:

:=

add_sub_expr

[

compare_op

add_sub_expr

]

compare_op

:

:=

'=='

|

'!='

|

'>='

|

'>'

|

'<='

|

'<'

|

'in'

|

'inlist'

|

'==#'

|

'!=#'

|

'>=#'

|

'>#'

|

'<=#'

|

'<#'

add_sub_expr

:

:=

times_div_expr

[

add_sub_op

times_div_expr

]

*

add_sub_op

:

:=

'+'

|

'_'

times_div_expr

:

:=

unary_op_expr

[

times_div_op

unary_op_expr

]

*

times_div_op

:

:=

/*

|

/'

unary_op_expr

:

:=

[

add_sub_op

unary_op_expr

]

*

|

expression

expression

:

:=

identifier

|

constant

|

function

|

assignment

|

field_reference

|

if_expr

|

for_expr

|

break_expr

|

continue_expr

|

'('

expression_list

)'

field_reference

:

:=

'\$'

[

'\$'

]

[

'#'

]

identifier

identifier

:

:=

id_start

[

id_rest

]

*

id_start

:

:=

letter

|

underscore

id_rest

:

:=

id_start

|

digit

constant

:

:=

" string "

|

' string '

|

number

function

:

:=

identifier

('

expression_list

[

','

expression_list

]

*

')'

assignment

:

:=

identifier

'='

top_expression

if_expr

:

:=

'if'

condition

'then'

expression_list

[

elif_expr

]

[

'else'

expression_list

]

'fi'

condition

:

:=

top_expression

elif_expr

:

:=

'elif'

condition

'then'

expression_list

elif_expr

|

"

for_expr

:

:=

'for'

identifier

'in'

list_expr

[

'separator'

separator_expr

]

','

expression_list

'rof'

list_expr

:
:=

top_expression

break_expr

:
:=

'break'

continue_expr

:
:=

'continue'

separator_expr

:
:=

top_expression

Notes:

- a top_expression always has a value. The value of an expression_list is the value of the last top_expression in the list. For example, the value of the expression list 1;2;'foobar';3 is 3.

- In a logical context, any non-empty value is True
- In a logical context, the empty value is False
- Strings and numbers can be used interchangeably. For example, 10 and '10' are the same thing.
- Comments are lines starting with a '#' character. Comments beginning later in a line are not supported.

Operator precedence

The operator precedence (order of evaluation) from highest (evaluated first) to lowest (evaluated last) is:

- Function calls, constants, parenthesized expressions, statement expressions, assignment expressions, field references.
- Unary $(+)$ and $(-)$ plus and minus. These operators evaluate right to left.

These and all the other arithmetic operators return integers if the expression results in a fractional part equal to zero. For example, if an expression returns 3.0 it is changed to 3.

- Multiply $(*)$ and divide $(/)$. These operators are associative and evaluate left to right. Use parentheses if you want to change the order of evaluation.
- Add $(+)$ and subtract $(-)$. These operators are associative and evaluate left to right.
- Numeric and string comparisons. These operators return '1' if the comparison succeeds, otherwise the empty string (''). Comparisons are not associative: a

<

b

<

c

is a syntax error.

- Unary logical not (!). This operator returns '1' if the expression is False (evaluates to the empty string), otherwise ''.
- Logical and (&&). This operator returns '1' if both the left-hand and right-hand expressions are True, or the empty string '' if either is False. It is associative, evaluates left to right, and does [short-circuiting](https://chortle.ccsu.edu/java5/Notes/chap40/ch40_2.html) [https://chortle.ccsu.edu/java5/Notes/chap40/ch40_2.html].
- Logical or (||). This operator returns '1' if either the left-hand or right-hand expression is True, or '' if both are False. It is associative, evaluates left to right, and does [short-circuiting](https://chortle.ccsu.edu/java5/Notes/chap40/ch40_2.html) [https://chortle.ccsu.edu/java5/Notes/chap40/ch40_2.html]. It is an *inclusive or*, returning '1' if both the left- and right-hand expressions are True.

Field references

A `field_reference` evaluates to the value of the metadata field named by lookup name that follows the \$ or \$\$. Using \$ is equivalent to using the field() function. Using \$\$ is equivalent to using the raw_field function. Examples:

```
* $authors ==> field('authors')
* $#genre ==> field('#genre')
* $$pubdate ==> raw_field('pubdate')
* $$#my_int ==> raw_field('#my_int')
```

If expressions

If expressions first evaluate the condition

. If the condition is True (a non-empty value) then the expression_list in the then clause is evaluated. If it is False then if present the expression_list in the elif or else clause is evaluated. The elif else and parts are optional. The words if , then , elif , else , and fi are reserved; you cannot use them as identifier names. You can put newlines and white space wherever they make sense. The condition is a top_expression not an expression_list ; semicolons are not allowed. The expression_lists are semicolon-separated sequences of top_expressions . An if expression returns the result of the last top_expression in the evaluated expression_list , or the empty string if no expression list was evaluated.

Examples:

```
*
program
:
if
field
(
'series'
)
then
'yes'
else
```

'no'

fi

*

program
:

if

field
(
'series'
)

then

a

=

'yes'
;

b

=

'no'

else

a

=

'no'


```
;

b

=

'yes'

fi

;

strcat
(
a
,
'_'
,
b
)
```

Nested
example:

if

```
program
:

if

field
(
'series'
)

then
```

```
if

check_yes_no
(
field
(
'#mybool'
),

"
,

"
,

'1'
)

then

'yes'

else

'no'

fi

else

'no series'

fi
```

As said above, an if produces a value. This means that all the following are equivalent:

```
*  
  
program  
:
```

```
if
```

```
field  
(  
  'series'  
)
```

```
then
```

```
'foo'
```

```
else
```

```
'bar'
```

```
fi
```

```
*
```

```
program  
:
```

```
if
```

```
field  
(  
  'series'  
)
```

```
then
```

```
a
```

```
=
```

'foo'

else

a

=

'bar'

fi

;

a

*

program

:

a

=

if

field

(

'series'

)

then

'foo'

else

'bar'

fi

;

a

As a last example, this program returns the value of the series column if the book has a series, otherwise the value of the title column:

```
program
:
field
(
if
field
(
'series'
)
then
'series'
else
'title'
fi
)
```

For expressions

The `for` expression iterates over a list of values, processing them one at a time. The `list_expression` must evaluate to either a metadata field lookup

name, `for` example `tags`, or a list of values. If the result is a valid lookup

name

then the field's value is fetched and the separator specified for that field type is used. If the result isn't a valid lookup name then it is assumed to be a list of values. The list is assumed to be separated by commas unless the optional keyword `separator` is supplied, in which case the list values must be separated by the result of evaluating the `separator_expr`

. Each value in the list is assigned to the specified variable then the `expression_list` is evaluated. You can use `break` to jump out of the loop, and `continue` to jump to the beginning of the loop for the next iteration.

Example: This template removes the first hierarchical name for each value in Genre (`#genre`), constructing a list with the new names:

```
program
:

new_tags

=

"
;

for
i
in
'#genre'
:

j

=

re
(
i
,
```

```
'^.*?\.(.*)$'
```

```
,
```

```
,
```

```
\1
```

```
,
```

```
);
```

```
new_tags
```

```
=
```

```
list_union
```

```
(
```

```
new_tags
```

```
,
```

```
j
```

```
,
```

```
''
```

```
;
```

```
)
```

```
rof
```

```
;
```

```
new_tags
```

If the original Genre is *History.Military, Science Fiction.Alternate History, ReadMe* then the template returns *Military, Alternate History, ReadMe*. You could use this template in calibre's *Edit metadata in bulk* → *Search & replace* with *Search for* set to template to strip off the first level of the hierarchy and assign the resulting value to Genre.

Note: the last line in the template, `new_tags`, isn't strictly necessary in this case because `for` returns the value of the last `top_expression` in the expression list. The value of an assignment is the value of its expression, so the value of the `for`

statement is what was assigned to new_tags
.

Relational operators

Relational operators return '1' if the comparison is true, otherwise the empty string ("").

There are two forms of relational operators: string comparisons and numeric comparisons.

String comparisons do case-insensitive string comparison using lexical order. The supported

string comparison operators	are	==
,		!=
,		<
,		<=
,		>
,		>=
,		in
,	and	inlist
.	For the	in

operator, the result of the left hand expression is interpreted as a regular expression pattern. The in operator is True if the value of left-hand regular expression matches the value of the right hand expression. The inlist operator is true if the left hand regular expression matches any one of the items in the right hand list where the items in the list are separated by commas. The matches are case-insensitive.

The numeric comparison operators	are	==#
,		!=#
,		<#
,		<=#
,		>#
,		>=#

. The left and right expressions must evaluate to numeric values with two exceptions: both the string value "None" (undefined field) and the empty string evaluate to the value zero.

Examples:

- program:

```
field('series')
```

```
==
```



```
'foo'
returns '1'
if the book's series is 'foo', otherwise "
```

- program:

```
'f.o'

in

field('series')
returns '1'
if the book's series matches the regular expression f.o
(e.g., foo, Off Onyx, etc.), otherwise "
```

- program:

```
'science'

inlist

field('#genre')
returns '1'
if any of the book's genres match the regular expression science
, e.g., Science, History of Science, Science Fiction etc.), otherwise "
```

- program:

```
'^science$'

inlist

field('#genre')
returns '1'
if any of the book's genres exactly match the regular expression ^science$
, e.g., Science. The genres History of Science and Science Fiction don't match. If
there isn't a match then returns "
```

- program:

```

if
  field('series')
  !=
  'foo'
then
  'bar'
else
  'mumble'
fi
returns the book's series is not 'bar'
if      . Otherwise it returns foo
.

```

- program:

```

if
  field('series')
  ==
  'foo'
||
  field('series')
  ==
  '1632'
then
  'yes'

```

```

else

'no'

fi
returns
if          series          is          either          'yes'
or          'foo'
,           '1632'
.           'no'
otherwise

```

- program:

```

if

'^(foo|1632)$'

in

field('series')

then

'yes'

else

'no'

fi
returns
if          series          is          either          'yes'
or          'foo'
,           '1632'
.           'no'
otherwise

```

- program:

```

if

11

>

```

2

then

'yes'

else

'no'

fi

returns

because

operator does a lexical comparison.

the

'no'

>

- program:

if

11

>#

2

then

'yes'

else

'no'

fi

returns

because

operator does a numeric comparison.

the

'yes'

>#

Additional available functions

The following functions are available in addition to those described in [Single Function Mode](#).

In *GPM* the functions described in *Single Function Mode* all require an additional first parameter specifying the value to operate upon. All parameters are expression_lists (see the grammar above).

- `add(x`

`,`

`y]*)`

– returns the sum of its arguments. Throws an exception if an argument is not a number. In most cases you can use the `+` operator instead of this function.

- `and(value`

`,`

`value]*)`

– returns the string “1” if all values are not empty, otherwise returns the empty string. You can have as many values as you want. In most cases you can use the `&&` operator instead of this function. One reason not to replace `and` with `&&` is if short-circuiting can change the results because of side effects. For example, `and(a="",b=5)` will always do both assignments, where the `&&` operator won’t do the second.

- `assign(id,`

`val)`

– assigns `val` to `id`, then returns `val`. `id` must be an identifier, not an expression. In most cases you can use the `=` operator instead of this function.

- `approximate_formats()`

– return a comma-separated list of formats associated with the book. There is no guarantee that the list is correct, although it probably is. This and other zero-parameter functions can be called in Template Program Mode (see below) using the `template` `{:'approximate_formats()'}`. Note that resulting format names are always uppercase, as in EPUB. The `approximate_formats()` function is significantly faster than the `formats_...` functions discussed below.

- `author_links(val_separator,`
`pair_separator)`
 – returns a string containing a list of authors and those authors' link values in the form:

```
author1
val_separator
author1_link
pair_separator
author2
val_separator
author2_link
etc
.
```

An author is separated from its link value by the `val_separator` string with no added spaces. author:linkvalue pairs are separated by the `pair_separator` string argument with no added spaces. It is up to you to choose separator strings that do not occur in author names or links. An author is included even if the author link is empty.

- `author_sorts(val_separator)`
 – returns a string containing a list of author's sort values for the authors of the book. The sort is the one in the author metadata information (different from the `author_sort` in books). The returned list has the form author

```
sort
```

```
1
```

```
val_separator
author
```

```
sort
```

2

etc. with no added spaces. The author sort values in this list are in the same order as the authors of the book. If you want spaces around val_separator then include them in the val_separator string.

- `booksize()`
 - returns the value of the calibre 'size' field. Returns "" if there are no formats.
- `check_yes_no(field_name,`

`is_undefined,`

`is_false,`

`is_true)`
 - checks if the value of the yes/no field named by the lookup name field_name is one of the values specified by the parameters, returning 'yes' if a match is found otherwise returning the empty string. Set the parameter `is_undefined`
, `is_false`
, `is_true`
to 1 (the number) to check that condition, otherwise set it to 0. Example:

```
check_yes_no("#bool",
```

```
1,
```

```
0,
```

```
1)
```

```
returns 'yes'  
if the yes/no field #bool  
is either True or undefined (neither True nor False).
```

```
More than one of is_undefined  
, is_false  
, or is_true  
can be set to 1.
```

- `ceiling(x)`
 - returns the smallest integer greater than or equal to x.
. Throws an exception if x
is not a number.

- `character(character_name)`
 - returns the character named by `character_name`. For example, `character('newline')` returns a newline character (`'\n'`).
 - The supported character names are newline, return, tab, and backslash.
- `cmp(x, y, lt, eq, gt)`
 - compares `x` and `y` after converting both to numbers. Returns `lt` if `x < y`, `eq` if `x == y`, and `gt` otherwise.
 - . This function can usually be replaced with one of the numeric compare operators (`==`, `<`, `>`, etc).
- `connected_device_name(storage_location_key)`
 - if a device is connected then return the device name, otherwise return the empty string. Each storage location on a device has its own device name. The `storage_location_key`

names are 'main'
, 'carda'
and 'cardb'

. This function works only in the GUI.

- `connected_device_uuid(storage_location_key)`
– if a device is connected then return the device uuid (unique id), otherwise return the empty string. Each storage location on a device has a different uuid. The storage_location_key location names are 'main', 'carda' and 'cardb'.
. This function works only in the GUI.

- `current_library_name()`
– return the last name on the path to the current calibre library.
- `current_library_path()`
– return the full path to the current calibre library.
- `current_virtual_library_name()`
– return the name of the current virtual library if there is one, otherwise the empty string. Library name case is preserved. Example: program:

`current_virtual_library_name()`
. This function works only in the GUI.

- `date_arithmetic(date, calc_spec, fmt)`
– Calculate a new date from date using calc_spec.
. Return the new date formatted according to optional fmt : if not supplied then the result will be in ISO format. The calc_spec is a string formed by concatenating pairs of vW (valueWhat where v is a possibly-negative number and W is one of the following letters:

- s : add v seconds to date

- m
: minutes add to v date
- h
: hours add to v date
- d
: days add to v date
- w
: weeks add to v date
- y
: years add to v date
 , where a year is 365 days.

Example: '1s3d-1m'
will add 1 second, add 3 days, and subtract 1 minute from date .

- days_between(date1,
date2)
– return the number of days between date1
and date2
• The number is positive if date1
is greater than date2
, otherwise negative. If either date1
or date2
are not dates, the function returns the empty string.
- divide(x,
y)
– returns x

/

y

. Throws an exception if either x or y are not numbers. This function can usually be replaced by the / operator.

- eval(string)
 - evaluates the string as a program, passing the local variables. This permits using the template processor to construct complex results from local variables. In [Template Program Mode](#), because the { and } characters are interpreted before the template is evaluated you must use [[for the { character and]] for the } character. They are converted automatically. Note also that prefixes and suffixes (the /*prefix/suffix* syntax) cannot be used in the argument to this function when using [Template Program Mode](#).
 - field(lookup_name)
 - returns the value of the metadata field with lookup name lookup_name
 - field_exists(field_name)
 - checks if a field (column) with the lookup name field_name exists, returning '1' if so and the empty string if not.
 - finish_formatting(val,

fmt,

prefix,

suffix)
 - apply the format, prefix, and suffix to a value in the same way as done in a template like {series_index:05.2f|
-
- |-
- }
- . This function is provided to ease conversion of complex single-function- or template-program-mode templates to *GPM* Templates. For example, the following program produces the same output as the above template:

```

program
:

finish_formatting
(
field
(
"series_index"
),

"05.2f"
,

" _ "
,

" _ "
)

```

Another example: for the template `{series:re(([^\\s])([^\\s]+(\\s|$),\\1)){series_index:0>2s|`

-

|

-

}{title}

use:

```

program
:

strcat
(

re
(
field

```

```

(
'series'
),

'([^\s])(^\s)+(\s|$)'
,

'
\1
'
),

finish_formatting
(
field
(
'series_index'
),

'0>2s'
,

'_'
,

'_'
),

field
(
'title'
)

)

```

- first_matching_cmp(val,

[

cmp,

result,

]*

else_result)

– compares val

<

cmp

in sequence, returning the associated result for the first comparison that succeeds. Returns else_result if no comparison succeeds. Example:

```
i
=
10
;

first_matching_cmp
(
i
,
5
,
"small"
,
10
,
"middle"
,
15
,
"large"
,
"giant"
)
```

returns "large"
. The same example with a first value of 16 returns "giant"
.

- `first_non_empty(value
[,
value]*)`
– returns the first value that is not empty. If all values are empty, then the empty string is returned. You can have as many values as you want.
- `floor(x)`
– returns the largest integer less than or equal to `x`. Throws an exception if `x` is not a number.
- `format_date(val,
format_string)`
– format the value, which must be a date string, using the `format_string`, returning a string. The formatting codes are:
 - `d`
:
the day as number without a leading zero (1 to 31)
 - `dd`
:
the day as number with a leading zero (01 to 31)
 - `ddd`
:
the abbreviated localized day name (e.g. “Mon” to “Sun”).
 - `dddd`
:
the long localized day name (e.g. “Monday” to “Sunday”).

- M
:
the month as number without a leading zero (1 to 12).
- MM
:
the month as number with a leading zero (01 to 12)
- MMM
:
the abbreviated localized month name (e.g. "Jan" to "Dec").
- MMMM
:
the long localized month name (e.g. "January" to "December").
- yy
:
the year as two digit number (00 to 99).
- yyyy
:
the year as four digit number.
- h
:
the hours without a leading 0 (0 to 11 or 0 to 23, depending on am/pm)
- hh
:
the hours with a leading 0 (00 to 11 or 00 to 23, depending on am/pm)
- m
:
the minutes without a leading 0 (0 to 59)

- mm
 - :
 - the minutes with a leading 0 (00 to 59)
- s
 - :
 - the seconds without a leading 0 (0 to 59)
- ss
 - :
 - the seconds with a leading 0 (00 to 59)
- ap
 - :
 - use a 12-hour clock instead of a 24-hour clock, with 'ap' replaced by the localized string for am or pm.
- AP
 - :
 - use a 12-hour clock instead of a 24-hour clock, with 'AP' replaced by the localized string for AM or PM.
- iso
 - :
 - the date with time and timezone. Must be the only format present.
- to_number
 - :
 - convert the date & time into a floating point number (a *timestamp*)
- from_number
 - :
 - convert a floating point number (a *timestamp*) into an iso formatted date. If you want a different date format then add the desired formatting string after from_number and a colon (:).
 - Example: from_number:MMM

dd

yyyy

You might get unexpected results if the date you are formatting contains localized month names, which can happen if you changed the date format tweaks to contain MMMM. In this case, instead of using the field() function as in:

```
format_date
(
  field
  (
    'pubdate'
  ),
  'yyyy'
)
```

use the raw_field() function as in:

```
format_date
(
  raw_field
  (
    'pubdate'
  ),
  'yyyy'
)
```

- `formats_modtimes(date_format_string)`
 - return a comma-separated list of colon-separated items `FMT:DATE` representing modification times for the formats of a book. The `date_format_string` parameter specifies how the date is to be formatted. See the `format_date()` function for details. You can use the `select`

function to get the modification time for a specific format. Note that format names are always uppercase, as in EPUB.

- `formats_paths()`
 - return a comma-separated list of colon-separated items `FMT:PATH` giving the full path to the formats of a book. You can use the `select` function to get the path for a specific format. Note that format names are always uppercase, as in EPUB.
- `formats_sizes()`
 - return a comma-separated list of colon-separated `FMT:SIZE` items giving the sizes in bytes of the formats of a book. You can use the `select` function to get the size for a specific format. Note that format names are always uppercase, as in EPUB.
- `fractional_part(x)`
 - returns the value after the decimal point. For example, `fractional_part(3.14)` returns 0.14.
Throws an exception if `x` is not a number.
- `has_cover()`
 - return 'Yes' if the book has a cover, otherwise the empty string.
- `is_marked()`
 - check whether the book is *marked* in calibre. If it is then return the value of the mark, either 'true' (lower case) or a comma-separated list of named marks. Returns '' (the empty string) if the book is not marked. This function works only in the GUI.
- `language_codes(lang_strings)`
 - return the [language codes](https://www.loc.gov/standards/iso639-2/php/code_list.php) [https://www.loc.gov/standards/iso639-2/php/code_list.php] for the language names passed in *lang_strings*. The strings must be in the language of the current locale. `Lang_strings` is a comma-separated list.
- `list_contains(value,`

`separator,`

`[`

`pattern,`

found_val,

]*

not_found_val)

– (Alias of in_list)
) Interpreting the value as a list of items separated by separator
, evaluate the pattern
against each value in the list. If the pattern
matches any value then return found_val
, otherwise return not_found_val
. The pattern
and found_value
can be repeated as many times as desired, permitting returning different values
depending on the search. The patterns are checked in order. The first match is
returned. Aliases: in_list()
, list_contains()

- list_count(value,

separator)

– interprets value
as a list of items separated by separator
, returning the count of items in the list. Aliases: count()
, list_count()

- list_count_matching(list,

pattern,

separator)

– interprets list
as a list of items separated by separator
, returning the number of items in the list that match the regular expression
pattern
. Aliases: list_count_matching()
, count_matching()

- list_difference(list1,

list2,

separator)
 – return a list made by removing from list1
 any item found in list2
 using a case-insensitive comparison. The items in list1
 and list2
 are separated by separator, as are the items in the returned list.

- list_equals(list1,

sep1,

list2,

sep2,

yes_val,

no_val)

– return yes_val
 if list1
 and list2 contain the same items, otherwise return no_val
 . The items are determined by splitting each list using the appropriate separator
 character (sep1
 or sep2
). The order of items in the lists is not relevant. The comparison is case-
 insensitive.

- list_intersection(list1,

list2,

separator)

– return a list made by removing from list1
 any item not found in list2
 , using a case-insensitive comparison. The items in list1
 and list2
 are separated by separator, as are the items in the returned list.

- list_re(src_list,

separator,

include_re,

opt_replace)

– Construct a list by first separating src_list into items using the separator character. For each item in the list, check if it matches include_re . If it does then add it to the list to be returned. If opt_replace is not the empty string then apply the replacement before adding the item to the returned list.

- list_re_group(src_list,

separator,

include_re,

search_re

[,

template_for_group]*)

– Like list_re except replacements are not optional. It uses re_group(item,

search_re,

template

...)

when doing the replacements.

- list_remove_duplicates(list,

separator)

– return a list made by removing duplicate items in list . If items differ only in case then the last is returned. The items in list are separated by separator , as are the items in the returned list.

- list_sort(list,

direction,

separator)

– return list sorted using a case-insensitive lexical sort. If direction

is zero, list
is sorted ascending, otherwise descending. The list items are separated by
separator
, as are the items in the returned list.

- `list_split(list_val,`

`sep,`

`id_prefix)`
– splits list_val
into separate values using sep
, then assigns the values to local variables named `id_prefix_N`
where N is the position of the value in the list. The first item has position 0 (zero).
The function returns the last element in the list.

Example:

```
list_split  
(  
  'one:two:foo'  
,  
  ':'  
,  
  'var'  
)
```

is equivalent to:

```
var_0  
  
=  
  
'one'  
;  
  
var_1  
  
=
```

```
'two'
;

var_2

=

'foo'
```

- `list_union(list1, list2, separator)`
 – return a list made by merging the items in list1 and list2, removing duplicate items using a case-insensitive comparison. If items differ in case, the one in list1 is used. The items in list1 and list2 are separated by separator, as are the items in the returned list. Aliases: `merge_lists()`, `list_union()`
- `mod(x, y)`
 – returns the remainder of x / y
 . Throws an exception if either x or y is not a number.
- `multiply(x, y)`

y]*)

– returns the product of its arguments. Throws an exception if any argument is not a number. This function can usually be replaced by the * operator.

- not(value)

– returns the string “1” if the value is empty, otherwise returns the empty string. This function can usually be replaced with the unary not (!) operator.

- ondevice()

– return the string 'Yes' if ondevice is set, otherwise return the empty string.

- or(value

,

value]*)

– returns the string '1' if any value is not empty, otherwise returns the empty string. You can have as many values as you want. This function can usually be replaced by the || operator. A reason it cannot be replaced is if short-circuiting will change the results because of side effects.

- print(a

,

b]*)

– prints the arguments to standard output. Unless you start calibre from the command line (calibre-debug

-g

), the output will go to a black hole. The print function always returns the empty string.

- raw_field(lookup_name

,

optional_default])

– returns the metadata field named by lookup_name without applying any formatting. It evaluates and returns the optional second

argument
if the field's value is undefined (None
optional_default
).

- `raw_list(lookup_name,`

`separator)`

– returns the metadata list named by `lookup_name` without applying any formatting or sorting, with the items separated by `separator`.

- `re_group(value,`

`pattern`

`[,`

`template_for_group]*)`

– return a string made by applying the regular expression pattern to `value` and replacing each matched instance with the the value returned by the corresponding template. In [Template Program Mode](#), like for the template and the eval functions, you use [[{ and]]] for

The following example looks for a series with more than one word and uppercases the first word:

```
program
:

re_group
(
field
(
'series'
),

"(\S* )(.*)"
,
```

```
"{$:uppercase()}"
,
"{$}"
)
'}
```

- `round(x)`
 - returns the nearest integer to x
 - Throws an exception if x is not a number.
- `series_sort()`
 - returns the series sort value.
- `strcat(a`
 `[,`
 `b]*)`
 - can take any number of arguments. Returns a string formed by concatenating all the arguments.
- `strcat_max(max,`
 `string1`
 `[,`
 `prefix2,`
 `string2]*)`
 - Returns a string formed by concatenating the arguments. The returned value is initialized to string1
 - Strings made from prefix,
 - string
 - pairs are added to the end of the value as long as the resulting string length is less than max
 - Prefixes can be empty. Returns string1
 - even if string1
 - is longer than max

. You can pass as many prefix, string pairs as you wish.

- `stricmp(x, y, lt, eq, gt)`
 – does a case-insensitive lexical comparison of x and y
 . Returns lt if x < y, eq if x == y, otherwise gt
 . This function can often be replaced by one of the lexical comparison operators (==, >, <, etc.)

- `strlen(value)`
 – Returns the length of the string value.

- `substr(str, start, end)`

– returns the start
 'th through the end
 'th characters of str
 . The first character in str
 is the zero'th character. If end
 is negative, then it indicates that many characters counting from the right. If end
 is zero, then it indicates the last character. For example, substr('12345',

1,

0)

returns

, and substr('12345',

1,

-1)

returns

'234'

.

- subtract(x,

y)

– returns x

-

y

. Throws an exception if either x
 or y
 are not numbers. This function can usually be replaced by the -
 operator.

- today()

– return a date+time string for today (now). This value is designed for use in
format_date or *days_between*, but can be manipulated like any other string. The
 date is in [ISO](https://en.wikipedia.org/wiki/ISO_8601) [https://en.wikipedia.org/wiki/ISO_8601] date/time format.

- template(x)

– evaluates x
 as a template. The evaluation is done in its own context, meaning that variables
 are not shared between the caller and the template evaluation.

More complex programs in template expressions - Template Program Mode

Template Program Mode (TPM) is a blend of [General Program Mode](#) and [Single Function Mode](#). *TPM* differs from *Single Function Mode* in that it permits writing template expressions that refer to other metadata fields, use nested functions, modify variables, and do arithmetic. It differs from *General Program Mode* in that the template is contained between { and } characters and doesn't begin with the word program: . The program portion of the template is a General Program Mode expression list.

Example: assume you want a template to show the series for a book if it has one, otherwise show the value of a custom field #genre. You cannot do this in the [Single Function Mode](#) because you cannot make reference to another metadata field within a template expression. In *TPM* you can, as the following expression demonstrates:

```
{  
#series:'ifempty($, field('#genre'))'}
```

The example shows several things:

- *TPM* is used if the expression begins with : ' and ends with ' . Anything else is assumed to be in [Single Function Mode](#).
- the variable \$ stands for the field named in the template: the expression is operating upon, #series in this case.
- functions must be given all their arguments. There is no default value. For example, the standard built-in functions must be given an additional initial parameter indicating the source field.
- white space is ignored and can be used anywhere within the expression.
- constant strings are enclosed in matching quotes, either ' or " .

All the functions listed under *Single Function Mode* and *General Program Mode* can be used in *TPM*.

In *TPM*, using `{` and `}` characters in string literals can lead to errors or unexpected results because they confuse the template processor. It tries to treat them as template expression boundaries, not characters. In some but not all cases you can replace a `{` with `[[` and `}` with `]]`. Generally, if your program contains `{` and `}` characters then you should use *General Program Mode*.

As with *General Program Mode*, for functions documented under [Single Function Mode](#) you must supply the value the function is to act upon as the first parameter in addition to the documented parameters. In *TPM* you can use `$` to access the value specified by the lookup name for the template expression.

Stored general program mode templates

[General Program Mode](#) supports saving templates and calling those templates from another template, much like calling stored functions. You save templates using *Preferences* → *Advanced* → *Template functions*. More information is provided in that dialog. You call a template the same way you call a function, passing positional arguments if desired. An argument can be any expression. Examples of calling a template, assuming the stored template is named `foo`:

- `foo()`
– call the template passing no arguments.
- `foo(a, b)`
call the template passing the values of the two variables `a` and `b`.
- `foo(if field('series')`
then

```

field('series_index')

else

0

fi)
-      if      the      book      has      a      series
  then      pass      the      series_index
,      otherwise      pass      the      value      0
.

```

You retrieve the arguments passed in the call to the stored template using the arguments function. It both declares and initializes local variables, effectively parameters. The variables are positional; they get the value of the value given in the call in the same position. If the corresponding parameter is not provided in the call then arguments assigns that variable the provided default value. If there is no default value then the variable is set to the empty string. For example, the following arguments function declares 2 variables, key alternate , :

```

arguments
(
key
,

alternate
=
'series'
)

```

Examples, again assuming the stored template is named foo :

- foo('#myseries')
 - argument key
 - is assigned the value 'myseries'
 - and the argument alternate
 - is assigned the default value 'series'
 - .

- `foo('series',
 '#genre')`
 the variable key
 is assigned the value 'series'
 and the variable alternate
 is assigned the value '#genre'
 .
- `foo()`
 – the variable key
 is assigned the empty string and the variable alternate
 is assigned the value 'series'
 .

An easy way to test stored templates is using the Template tester

dialog. For ease of access give it a keyboard shortcut in *Preferences* → *Advanced* → *Keyboard shortcuts* → *Template tester*. Giving the Stored

templates

dialog a shortcut will help switching more rapidly between the tester and editing the stored template's source code.

Providing additional information to templates

A developer can choose to pass additional information to the template processor, such as application-specific book metadata or information about what the processor is being asked to do. A template can access this information and use it during the evaluation.

Developer: how to pass additional information

The additional information is a Python dictionary containing pairs `variable_name:`

`variable_value`

where the values must be strings. The template can access the dict, creating template local variables named `variable_name` containing the value `variable_value`. The user cannot change the name so it is best to use names that won't collide with other template local variables, for example by prefixing the name with an underscore.

This dict is passed to the template processor (the formatter) using the named parameter `global_vars=your_dict`. The full method signature is:

```
def
safe_format
(
self
,

fmt
,

kwargs
,

error_value
,

book
,

column_name
=
None
,

template_cache
=
None
,

strip_results
=
True
,

template_functions
=
None
,

global_vars
```

```
=  
{})
```

Template writer: how to access the additional information

You access the additional information (the globals dict) in a template using the template function:

```
globals  
(  
id  
[  
=  
expression  
]  
  
[,  
  
id  
[  
=  
expression  
]]  
*)  
)
```

where `id` is any legal variable name. This function checks whether the additional information provided by the developer contains the name. If it does then the function assigns the provided value to a template local variable with that name. If the name is not in the additional information and if an expression is provided, the expression is evaluated and the result is assigned to the local variable. If neither a value nor an expression is provided, the function assigns the empty string ("") to the local variable.

A template can set a value in the globals dict using the template function:

```

set_globals
(
  id
  [
    =
    expression
  ]

  [,

  id
  [
    =
    expression
  ]]
  *
)

```

This function sets the globals dict key:value pair where the value of the template local variable id is the value of the local variable. If that local variable doesn't exist then the value is set to the result of evaluating expression.

Notes on the difference between modes

The three program modes, [Single Function Mode](#) (SFM), [Template Program Mode](#) (TPM), and [General Program Mode](#) (GPM), work differently. SFM is intended to be 'simple' so it hides a lot of programming language bits.

Differences:

- In SFM the value of the column is always passed as an 'invisible' first argument to a function included in the template.
- SFM doesn't support the difference between variables and strings; all values are strings.
- The following SFM template returns either the series name or the string "no series":

```
{
series
:
ifempty
(
no

series
)}
```

The equivalent template in *TPM* is

```
{
series
:
'ifempty($, '
no

series
')
}
```

The equivalent template in *GPM* is:

```
program
:

ifempty
(
field
(
'series'
),

'no series'
)
```

The first argument to ifempty
is the value of the field series
. The second argument is the string no

series

. In SFM the first argument, the value of the field, is automatically passed (the invisible argument).

- Several template functions, for example `booksize()` and `current_library_name()`, take no arguments. Because of the 'invisible argument' you cannot use these functions in SFM.
- Nested functions, where a function calls another function to compute an argument, cannot be used in SFM. For example this template, intended to return the first 5 characters of the series value uppercased, won't work in SFM:

```
{
series
:
uppercase
(
  substr
  (
    0
    ,
    5
  )
)}
```

- *TPM* and *GPM* support nested functions. The above template in *TPM* would be:

```
{
series
:
'uppercase(substr($, 0,5))'
}
```

In *GPM* it would be:

```
program
:
```

```
uppercase
(
  substr
  (
    field
    (
      'series'
    ),
    0
  ,
  5
))
```

- As noted in the above [Template Program Mode](#) section, using { and } characters in *TPM* string literals can lead to errors or unexpected results because they confuse the template processor. It tries to treat them as template boundaries, not characters. In some but not all cases you can replace a { with [[and a } with]]. Generally, if your program contains { and } characters then you should use *General Program Mode*.

User-defined Python template functions

You can add your own Python functions to the template processor. Such functions can be used in any of the three template programming modes. The functions are added by going to *Preferences → Advanced → Template functions*. Instructions are shown in that dialog.

Special notes for save/send templates

Special processing is applied when a template is used in a *save to disk* or *send to device* template. The values of the fields are cleaned, replacing characters that are special to file systems with underscores, including slashes. This means that field text cannot be used to create folders. However, slashes are not changed in prefix or suffix strings, so slashes in these strings will cause folders to be created. Because of this, you can create variable-depth folder structure.

For example, assume we want the folder structure *series/series_index - title*, with the caveat that if series does not exist, then the title should be in the top folder. The template to do this is:

```
{
series
:
||/
}{
series_index
:
||
-

}{
title
}
```

The slash and the hyphen appear only if series is not empty.

The lookup function lets us do even fancier processing. For example, assume that if a book has a series, then we want the folder structure *series/series index - title.fmt*. If the book does not have a series then we want the folder structure *genre/author_sort/title.fmt*. If the book has no genre then we want to use 'Unknown'. We want two completely different paths, depending on the value of series.

To accomplish this, we:

1. Create a composite field (give it lookup name #aa) containing
{series}/{series_index}
-
{title}
. If the series is not empty, then this template will produce *series/series_index - title*.
2. Create a composite field (give it lookup name #bb) containing
{#genre:ifempty(Unknown)}/{author_sort}/{title}
. This template produces *genre/author_sort/title*, where an empty genre is replaced with *Unknown*.

3. Set the save template to `{series:lookup(.,#aa,#bb)}`. This template chooses composite field `#aa` if series is not empty and composite field `#bb` if series is empty. We therefore have two completely different save paths, depending on whether or not *series* is empty.

Templates and plugboards

Plugboards are used for changing the metadata written into books during send-to-device and save-to-disk operations. A plugboard permits you to specify a template to provide the data to write into the book's metadata. You can use plugboards to modify the following fields: authors, author_sort, language, publisher, tags, title, title_sort. This feature helps people who want to use different metadata in books on devices to solve sorting or display issues.

When you create a plugboard, you specify the format and device for which the plugboard is to be used. A special device is provided, `save_to_disk`, that is used when saving formats (as opposed to sending them to a device). Once you have chosen the format and device, you choose the metadata fields to change, providing templates to supply the new values. These templates are *connected* to their destination fields, hence the name *plugboards*. You can of course use composite columns in these templates.

When a plugboard might apply (Content server, save to disk, or send to device), calibre searches the defined plugboards to choose the correct one for the given format and device. For example, to find the appropriate plugboard for an EPUB book being sent to an ANDROID device, calibre searches the plugboards using the following search order:

- a plugboard with an exact match on format and device, e.g., EPUB and ANDROID
- a plugboard with an exact match on format and the special any device choice, e.g., EPUB and any device
- a plugboard with the special any format choice and an exact match on device, e.g., any format

and

ANDROID

- a plugboard with any
format
and any
device

The tags and authors fields have special treatment, because both of these fields can hold more than one item. A book can have many tags and many authors. When you specify that one of these two fields is to be changed, the template's result is examined to see if more than one item is there. For tags, the result is cut apart wherever calibre finds a comma. For example, if the template produces the value Thriller,

Horror

, then the result will be two tags, Thriller
and Horror

. There is no way to put a comma in the middle of a tag.

The same thing happens for authors, but using a different character for the cut, a & (ampersand) instead of a comma. For example, if the template produces the value Blogs,

Joe&Posts,

Susan

, then the book will end up with two authors, Blogs,

Joe

and Posts,

Susan

. If the template produces the value Blogs,

Joe;Posts,

Susan

, then the book will have one author with a rather strange name.

Plugboards affect the metadata written into the book when it is saved to disk or written to the device. Plugboards do not affect the metadata used by save to

disk
and

send

to

device

to create the file names. Instead, file names are constructed using the templates entered on the appropriate preferences window.

Tips

- Use the Template Tester to test templates. Add the tester to the context menu for books in the library and/or give it a keyboard shortcut.
- Templates can use other templates by referencing composite columns built with the desired template. Alternatively, you can use Stored Templates.
- In a plugboard, you can set a field to empty (or whatever is equivalent to empty) by using the special template `{}`. This template will always evaluate to an empty string.
- The technique described above to show numbers even if they have a zero value works with the standard field `series_index`.

Function reference

- [Reference for all built-in template language functions](#)
 - [Arithmetic](#)
 - [add\(x \[,y\]*\)](#)
 - [ceiling\(x\)](#)
 - [divide\(x,y\)](#)
 - [floor\(x\)](#)
 - [fractional part\(x\)](#)
 - [mod\(x\)](#)
 - [multiply\(x \[,y\]*\)](#)

- [round\(x\)](#)
- [subtract\(x,y\)](#)
- [Boolean](#)
 - [and\(value \[, value\]*\)](#)
 - [not\(value\)](#)
 - [or\(value \[, value\]*\)](#)
- [Date functions](#)
 - [date arithmetic\(date, calc spec, fmt\)](#)
 - [days between\(date1, date2\)](#)
 - [today\(\)](#)
- [Formatting values](#)
 - [finish formatting\(val, fmt, prefix, suffix\)](#)
 - [format date\(val, format string\)](#)
 - [format number\(v, template\)](#)
 - [human readable\(v\)](#)
 - [rating to stars\(value, use half stars\)](#)
- [Get values from metadata](#)
 - [annotation count\(\)](#)
 - [approximate formats\(\)](#)
 - [author links\(val separator, pair separator\)](#)
 - [author sorts\(val separator\)](#)
 - [booksize\(\)](#)
 - [connected device name\(storage location\)](#)
 - [connected device uuid\(storage location\)](#)
 - [current library name\(\)](#)

- [current library path\(\)](#)
- [current virtual library name\(\)](#)
- [field\(lookup name\)](#)
- [formats modtimes\(date format\)](#)
- [formats paths\(\)](#)
- [formats sizes\(\)](#)
- [has cover\(\)](#)
- [is marked\(\)](#)
- [language codes\(lang strings\)](#)
- [language strings\(lang codes, localize\)](#)
- [ondevice\(\)](#)
- [raw field\(lookup name \[, optional default\]\)](#)
- [raw list\(lookup name, separator\)](#)
- [series sort\(\)](#)
- [user categories\(\)](#)
- [virtual libraries\(\)](#)
- [If-then-else](#)
 - [check yes no\(field name, is undefined, is false, is true\)](#)
 - [contains\(val, pattern, text if match, text if not match\)](#)
 - [field exists\(field name\)](#)
 - [ifempty\(val, text if empty\)](#)
 - [test\(val, text if not empty, text if empty\)](#)
- [Iterating over values](#)
 - [first non empty\(value \[, value\]*\)](#)
 - [lookup\(val, \[pattern, field,\]+ else field\)](#)

- switch(val, [pattern, value,]+ else value)
- List lookup
 - identifier in list(val, id, found val, not found val)
 - in list(val, separator, [pattern, found val,]+ not found val)
 - list item(val, index, separator)
 - select(val, key)
 - str in list(val, separator, [string, found val,]+ not found val)
- List manipulation
 - count(val, separator)
 - list count matching(list, pattern, separator)
 - list difference(list1, list2, separator)
 - list equals(list1, sep1, list2, sep2, yes val, no val)
 - list intersection(list1, list2, separator)
 - list re(src list, separator, include re, opt replace)
 - list re group(src list, separator, include re, search re [, group template]+)
 - list remove duplicates(list, separator)
 - list sort(list, direction, separator)
 - list split(list val, sep, id prefix)
 - list union(list1, list2, separator)
 - subitems(val, start index, end index)
 - sublist(val, start index, end index, separator)
- Other
 - assign(id, val)
 - print(a[, b]*)

- Recursion
 - eval(template)
 - template(x)
- Relational
 - cmp(x, y, lt, eq, gt)
 - first matching cmp(val, [cmp1, result1,]+, else result)
 - strcmp(x, y, lt, eq, gt)
- String case changes
 - capitalize(val)
 - lowercase(val)
 - titlecase(val)
 - uppercase(val)
- String manipulation
 - character(character name)
 - re(val, pattern, replacement)
 - re_group(val, pattern [, template for group]*)
 - shorten(val, left chars, middle text, right chars)
 - strcat(a [, b]*)
 - strcat max(max, string1 [, prefix2, string2]*)
 - strlen(a)
 - substr(str, start, end)
 - swap around articles(val, separator)
 - swap around comma(val)
 - transliterate(a)
- other

- [arguments\(id\[=expression\] \[, id\[=expression\]\]*\)](#)
- [globals\(id\[=expression\] \[, id\[=expression\]\]*\)](#)
- [API of the Metadata objects](#)

Reference for all built-in template language functions

Here, we document all the built-in functions available in the calibre template language. Every function is implemented as a class in python and you can click the source links to see the source code, in case the documentation is insufficient. The functions are arranged in logical groups by type.

- [Arithmetic](#)
 - [add\(x \[, y\]*\)](#)
 - [ceiling\(x\)](#)
 - [divide\(x, y\)](#)
 - [floor\(x\)](#)
 - [fractional part\(x\)](#)
 - [mod\(x\)](#)
 - [multiply\(x \[, y\]*\)](#)
 - [round\(x\)](#)
 - [subtract\(x, y\)](#)
- [Boolean](#)
 - [and\(value \[, value\]*\)](#)
 - [not\(value\)](#)
 - [or\(value \[, value\]*\)](#)
- [Date functions](#)

- [date arithmetic\(date, calc spec, fmt\)](#)
- [days between\(date1, date2\)](#)
- [today\(\)](#)
- [Formatting values](#)
 - [finish formatting\(val, fmt, prefix, suffix\)](#)
 - [format date\(val, format string\)](#)
 - [format number\(v, template\)](#)
 - [human readable\(v\)](#)
 - [rating to stars\(value, use half stars\)](#)
- [Get values from metadata](#)
 - [annotation count\(\)](#)
 - [approximate formats\(\)](#)
 - [author links\(val separator, pair separator\)](#)
 - [author sorts\(val separator\)](#)
 - [booksize\(\)](#)
 - [connected device name\(storage location\)](#)
 - [connected device uuid\(storage location\)](#)
 - [current library name\(\)](#)
 - [current library path\(\)](#)
 - [current virtual library name\(\)](#)
 - [field\(lookup name\)](#)
 - [formats modtimes\(date format\)](#)
 - [formats paths\(\)](#)
 - [formats sizes\(\)](#)

- [has cover\(\)](#)
- [is marked\(\)](#)
- [language codes\(lang strings\)](#)
- [language strings\(lang codes, localize\)](#)
- [ondevice\(\)](#)
- [raw field\(lookup name \[, optional default\]\)](#)
- [raw list\(lookup name, separator\)](#)
- [series sort\(\)](#)
- [user categories\(\)](#)
- [virtual libraries\(\)](#)
- [If-then-else](#)
 - [check yes no\(field name, is undefined, is false, is true\)](#)
 - [contains\(val, pattern, text if match, text if not match\)](#)
 - [field exists\(field name\)](#)
 - [ifempty\(val, text if empty\)](#)
 - [test\(val, text if not empty, text if empty\)](#)
- [Iterating over values](#)
 - [first non empty\(value \[, value\]*\)](#)
 - [lookup\(val, \[pattern, field,\]+ else field\)](#)
 - [switch\(val, \[pattern, value,\]+ else value\)](#)
- [List lookup](#)
 - [identifier in list\(val, id, found val, not found val\)](#)
 - [in list\(val, separator, \[pattern, found val, \]+ not found val\)](#)
 - [list item\(val, index, separator\)](#)

- [select\(val, key\)](#)
- [str in list\(val, separator, \[string, found val, \]+ not found val\)](#)
- [List manipulation](#)
 - [count\(val, separator\)](#)
 - [list count matching\(list, pattern, separator\)](#)
 - [list difference\(list1, list2, separator\)](#)
 - [list equals\(list1, sep1, list2, sep2, yes val, no val\)](#)
 - [list intersection\(list1, list2, separator\)](#)
 - [list re\(src list, separator, include re, opt replace\)](#)
 - [list re group\(src list, separator, include re, search re \[, group template\] +\)](#)
 - [list remove duplicates\(list, separator\)](#)
 - [list sort\(list, direction, separator\)](#)
 - [list split\(list val, sep, id prefix\)](#)
 - [list union\(list1, list2, separator\)](#)
 - [subitems\(val, start index, end index\)](#)
 - [sublist\(val, start index, end index, separator\)](#)
- [Other](#)
 - [assign\(id, val\)](#)
 - [print\(a\[, b\]*\)](#)
- [Recursion](#)
 - [eval\(template\)](#)
 - [template\(x\)](#)
- [Relational](#)
 - [cmp\(x, y, lt, eq, gt\)](#)

- [first matching cmp\(val, \[cmp1, result1,\]+, else result\)](#)
- [strcmp\(x, y, lt, eq, gt\)](#)
- [String case changes](#)
 - [capitalize\(val\)](#)
 - [lowercase\(val\)](#)
 - [titlecase\(val\)](#)
 - [uppercase\(val\)](#)
- [String manipulation](#)
 - [character\(character_name\)](#)
 - [re\(val, pattern, replacement\)](#)
 - [re group\(val, pattern \[, template for group\]*\)](#)
 - [shorten\(val, left chars, middle text, right chars\)](#)
 - [strcat\(a \[, b\]*\)](#)
 - [strcat_max\(max, string1 \[, prefix2, string2\]*\)](#)
 - [strlen\(a\)](#)
 - [substr\(str, start, end\)](#)
 - [swap_around_articles\(val, separator\)](#)
 - [swap_around_comma\(val\)](#)
 - [transliterate\(a\)](#)
- [other](#)
 - [arguments\(id\[=expression\] \[, id\[=expression\]\]*\)](#)
 - [globals\(id\[=expression\] \[, id\[=expression\]\]*\)](#)
- [API of the Metadata objects](#)

Arithmetic

add(x [, y]*)

class calibre.utils.formatter_functions.BuiltinAdd[\[source\]](#)

add(x [, y]*) – returns the sum of its arguments. Throws an exception if an argument is not a number. This function can often be replaced with the + operator.

ceiling(x)

class calibre.utils.formatter_functions.BuiltinCeiling[\[source\]](#)

ceiling(x) – returns the smallest integer greater than or equal to x. Throws an exception if x is not a number.

divide(x, y)

class calibre.utils.formatter_functions.BuiltinDivide[\[source\]](#)

divide(x, y) – returns x / y. Throws an exception if either x or y are not numbers. This function can often be replaced with the / operator.

floor(x)

class calibre.utils.formatter_functions.BuiltinFloor[\[source\]](#)

floor(x) – returns the largest integer less than or equal to x. Throws an exception if x is not a number.

fractional_part(x)

class calibre.utils.formatter_functions.BuiltinFractionalPart[\[source\]](#)

fractional_part(x) – returns the value after the decimal point. For example, fractional_part(3.14) returns 0.14. Throws an exception if x is not a number.

mod(x)

class calibre.utils.formatter_functions.BuiltinMod[\[source\]](#)

mod(x) – returns floor(remainder of x / y). Throws an exception if either x or y is not a number.

multiply(x [, y]*)

class calibre.utils.formatter_functions.BuiltinMultiply[\[source\]](#)

`multiply(x [, y]*)` – returns the product of its arguments. Throws an exception if any argument is not a number. This function can often be replaced with the `*` operator.

[round\(x\)](#)

`class calibre.utils.formatter_functions.BuiltinRound`[\[source\]](#)

`round(x)` – returns the nearest integer to x. Throws an exception if x is not a number.

[subtract\(x, y\)](#)

`class calibre.utils.formatter_functions.BuiltinSubtract`[\[source\]](#)

`subtract(x, y)` – returns `x - y`. Throws an exception if either x or y are not numbers. This function can often be replaced with the `-` operator.

[Boolean](#)

[and\(value \[, value\]*\)](#)

`class calibre.utils.formatter_functions.BuiltinAnd`[\[source\]](#)

`and(value [, value]*)` – returns the string “1” if all values are not empty, otherwise returns the empty string. This function works well with `test` or `first_non_empty`. You can have as many values as you want. In many cases the `&&` operator can replace this function.

[not\(value\)](#)

`class calibre.utils.formatter_functions.BuiltinNot`[\[source\]](#)

`not(value)` – returns the string “1” if the value is empty, otherwise returns the empty string. This function works well with `test` or `first_non_empty`. In many cases the `!` operator can replace this function.

[or\(value \[, value\]*\)](#)

`class calibre.utils.formatter_functions.BuiltinOr`[\[source\]](#)

`or(value [, value]*)` – returns the string “1” if any value is not empty, otherwise returns the empty string. This function works well with `test` or `first_non_empty`. You can have as many values as you want. In many cases the `||` operator can replace this function.

[Date functions](#)

[date_arithmetic\(date, calc_spec, fmt\)](#)

class calibre.utils.formatter_functions.BuiltinDateArithmetic[\[source\]](#)

`date_arithmetic(date, calc_spec, fmt)` – Calculate a new date from ‘date’ using ‘calc_spec’. Return the new date formatted according to optional ‘fmt’: if not supplied then the result will be in iso format. The calc_spec is a string formed by concatenating pairs of ‘vW’ (valueWhat) where ‘v’ is a possibly-negative number and W is one of the following letters: s: add ‘v’ seconds to ‘date’ m: add ‘v’ minutes to ‘date’ h: add ‘v’ hours to ‘date’ d: add ‘v’ days to ‘date’ w: add ‘v’ weeks to ‘date’ y: add ‘v’ years to ‘date’, where a year is 365 days. Example: ‘1s3d-1m’ will add 1 second, add 3 days, and subtract 1 minute from ‘date’.

[days_between\(date1, date2\)](#)

class calibre.utils.formatter_functions.BuiltinDaysBetween[\[source\]](#)

`days_between(date1, date2)` – return the number of days between date1 and date2. The number is positive if date1 is greater than date2, otherwise negative. If either date1 or date2 are not dates, the function returns the empty string.

[today\(\)](#)

class calibre.utils.formatter_functions.BuiltinToday[\[source\]](#)

`today()` – return a date string for today. This value is designed for use in `format_date` or `days_between`, but can be manipulated like any other string. The date is in ISO format.

[Formatting values](#)

[finish_formatting\(val, fmt, prefix, suffix\)](#)

class calibre.utils.formatter_functions.BuiltinFinishFormatting[\[source\]](#)

`finish_formatting(val, fmt, prefix, suffix)` – apply the format, prefix, and suffix to a value in the same way as done in a template like `{series_index:05.2f} - /- }`. For example, the following program produces the same output as the above template: `program: finish_formatting(field("series_index"), "05.2f", " - ", " - ")`

[format_date\(val, format_string\)](#)

class calibre.utils.formatter_functions.BuiltinFormatDate[\[source\]](#)

`format_date(val, format_string)` – format the value, which must be a date, using the `format_string`, returning a string. The formatting codes are: d : the day as number without a leading zero (1 to 31) dd : the day as number with a leading zero (01 to 31) ddd : the abbreviated localized day name (e.g. “Mon” to “Sun”). dddd : the long localized day name (e.g. “Monday” to “Sunday”). M : the month as number without a leading zero (1 to 12). MM : the month as number with a leading zero (01 to 12) MMM : the abbreviated localized month

name (e.g. “Jan” to “Dec”). MMMM : the long localized month name (e.g. “January” to “December”). yy : the year as two digit number (00 to 99). yyyy : the year as four digit number. h : the hours without a leading 0 (0 to 11 or 0 to 23, depending on am/pm) hh : the hours with a leading 0 (00 to 11 or 00 to 23, depending on am/pm) m : the minutes without a leading 0 (0 to 59) mm : the minutes with a leading 0 (00 to 59) s : the seconds without a leading 0 (0 to 59) ss : the seconds with a leading 0 (00 to 59) ap : use a 12-hour clock instead of a 24-hour clock, with “ap” replaced by the localized string for am or pm AP : use a 12-hour clock instead of a 24-hour clock, with “AP” replaced by the localized string for AM or PM iso : the date with time and timezone. Must be the only format present to_number: the date as a floating point number from_number[:fmt]: format the timestamp using fmt if present otherwise iso

[format_number\(v, template\)](#)

class calibre.utils.formatter_functions.BuiltinFormatNumber[\[source\]](#)

format_number(v, template) – format the number v using a Python formatting template such as “{0:5.2f}” or “{0:,d}” or “\${0:5.2f}”. The field_name part of the template must be a 0 (zero) (the “{0:” in the above examples). See the template language and Python documentation for more examples. You can leave off the leading “{0:” and trailing “}” if the template contains only a format. Returns the empty string if formatting fails.

[human_readable\(v\)](#)

class calibre.utils.formatter_functions.BuiltinHumanReadable[\[source\]](#)

human_readable(v) – return a string representing the number v in KB, MB, GB, etc.

[rating_to_stars\(value, use_half_stars\)](#)

class calibre.utils.formatter_functions.BuiltinRatingToStars[\[source\]](#)

rating_to_stars(value, use_half_stars) – Returns the rating as string of star characters. The value is a number between 0 and 5. Set use_half_stars to 1 if you want half star characters for custom ratings columns that support non-integer ratings, for example 2.5.

[Get values from metadata](#)

[annotation_count\(\)](#)

class calibre.utils.formatter_functions.BuiltinAnnotationCount[\[source\]](#)

annotation_count() – return the total number of annotations of all types attached to the current book. This function works only in the GUI.

[approximate_formats\(\)](#)

class calibre.utils.formatter_functions.BuiltinApproximateFormats[\[source\]](#)

`approximate_formats()` – return a comma-separated list of formats that at one point were associated with the book. There is no guarantee that this list is correct, although it probably is. This function can be called in template program mode using the template “{:’approximate_formats()}”. Note that format names are always uppercase, as in EPUB. This function works only in the GUI. If you want to use these values in save-to-disk or send-to-device templates then you must make a custom “Column built from other columns”, use the function in that column’s template, and use that column’s value in your save/send templates

author links(val separator, pair separator)

class calibre.utils.formatter_functions.BuiltinAuthorLinks[\[source\]](#)

`author_links(val_separator, pair_separator)` – returns a string containing a list of authors and that author’s link values in the form author1 val_separator author1link pair_separator author2 val_separator author2link etc. An author is separated from its link value by the val_separator string with no added spaces. author:linkvalue pairs are separated by the pair_separator string argument with no added spaces. It is up to you to choose separator strings that do not occur in author names or links. An author is included even if the author link is empty.

author sorts(val separator)

class calibre.utils.formatter_functions.BuiltinAuthorSorts[\[source\]](#)

`author_sorts(val_separator)` – returns a string containing a list of author’s sort values for the authors of the book. The sort is the one in the author metadata (different from the author_sort in books). The returned list has the form author sort 1 val_separator author sort 2 etc. The author sort values in this list are in the same order as the authors of the book. If you want spaces around val_separator then include them in the separator string

booksize()

class calibre.utils.formatter_functions.BuiltinBooksize[\[source\]](#)

`booksize()` – return value of the size field. This function works only in the GUI. If you want to use this value in save-to-disk or send-to-device templates then you must make a custom “Column built from other columns”, use the function in that column’s template, and use that column’s value in your save/send templates

connected device name(storage location)

class calibre.utils.formatter_functions.BuiltinConnectedDeviceName[\[source\]](#)

`connected_device_name(storage_location)` – if a device is connected then return the device name, otherwise return the empty string. Each storage location on a device can have a different name. The location names are ‘main’, ‘carda’ and ‘cardb’. This function works only in the GUI.

[connected_device_uuid\(storage_location\)](#)

class calibre.utils.formatter_functions.BuiltinConnectedDeviceUUID[\[source\]](#)

`connected_device_uuid(storage_location)` – if a device is connected then return the device uuid (unique id), otherwise return the empty string. Each storage location on a device has a different uuid. The location names are ‘main’, ‘carda’ and ‘cardb’. This function works only in the GUI.

[current_library_name\(\)](#)

class calibre.utils.formatter_functions.BuiltinCurrentLibraryName[\[source\]](#)

`current_library_name()` – return the last name on the path to the current calibre library. This function can be called in template program mode using the template “{:’current_library_name()}”.

[current_library_path\(\)](#)

class calibre.utils.formatter_functions.BuiltinCurrentLibraryPath[\[source\]](#)

`current_library_path()` – return the path to the current calibre library. This function can be called in template program mode using the template “{:’current_library_path()}”.

[current_virtual_library_name\(\)](#)

class calibre.utils.formatter_functions.BuiltinCurrentVirtualLibraryName[\[source\]](#)

`current_virtual_library_name()` – return the name of the current virtual library if there is one, otherwise the empty string. Library name case is preserved. Example: “program: current_virtual_library_name()”.

[field\(lookup_name\)](#)

class calibre.utils.formatter_functions.BuiltinField[\[source\]](#)

`field(lookup_name)` – returns the metadata field named by `lookup_name`

[formats_modtimes\(date_format\)](#)

class calibre.utils.formatter_functions.BuiltinFormatsModtimes[\[source\]](#)

`formats_modtimes(date_format)` – return a comma-separated list of colon-separated items representing modification times for the formats of a book. The `date_format` parameter specifies how the date is to be formatted. See the `format_date` function for details. You can use the `select` function to get the mod time for a specific format. Note that format names are always uppercase, as in EPUB.

[formats_paths\(\)](#)

class `calibre.utils.formatter_functions.BuiltinFormatsPaths`[\[source\]](#)

`formats_paths()` – return a comma-separated list of colon-separated items representing full path to the formats of a book. You can use the `select` function to get the path for a specific format. Note that format names are always uppercase, as in EPUB.

[formats_sizes\(\)](#)

class `calibre.utils.formatter_functions.BuiltinFormatsSizes`[\[source\]](#)

`formats_sizes()` – return a comma-separated list of colon-separated items representing sizes in bytes of the formats of a book. You can use the `select` function to get the size for a specific format. Note that format names are always uppercase, as in EPUB.

[has_cover\(\)](#)

class `calibre.utils.formatter_functions.BuiltinHasCover`[\[source\]](#)

`has_cover()` – return Yes if the book has a cover, otherwise return the empty string

[is_marked\(\)](#)

class `calibre.utils.formatter_functions.BuiltinIsMarked`[\[source\]](#)

`is_marked()` – check whether the book is ‘marked’ in calibre. If it is then return the value of the mark, either ‘true’ or the comma-separated list of named marks. Returns “ if the book is not marked.

[language_codes\(lang_strings\)](#)

class `calibre.utils.formatter_functions.BuiltinLanguageCodes`[\[source\]](#)

`language_codes(lang_strings)` – return the language codes for the strings passed in `lang_strings`. The strings must be in the language of the current locale. `Lang_strings` is a comma-separated list.

[language_strings\(lang_codes, localize\)](#)

class `calibre.utils.formatter_functions.BuiltinLanguageStrings`[\[source\]](#)

language_strings(lang_codes, localize) – return the strings for the language codes passed in lang_codes. If localize is zero, return the strings in English. If localize is not zero, return the strings in the language of the current locale. Lang_codes is a comma-separated list.

[ondevice\(\)](#)

class calibre.utils.formatter_functions.BuiltinOndevice[\[source\]](#)

ondevice() – return Yes if ondevice is set, otherwise return the empty string. This function works only in the GUI. If you want to use this value in save-to-disk or send-to-device templates then you must make a custom “Column built from other columns”, use the function in that column’s template, and use that column’s value in your save/send templates

[raw_field\(lookup_name \[, optional default\]\)](#)

class calibre.utils.formatter_functions.BuiltinRawField[\[source\]](#)

raw_field(lookup_name [, optional_default]) – returns the metadata field named by lookup_name without applying any formatting. It evaluates and returns the optional second argument ‘default’ if the field is undefined (‘None’).

[raw_list\(lookup_name, separator\)](#)

class calibre.utils.formatter_functions.BuiltinRawList[\[source\]](#)

raw_list(lookup_name, separator) – returns the metadata list named by lookup_name without applying any formatting or sorting and with items separated by separator.

[series_sort\(\)](#)

class calibre.utils.formatter_functions.BuiltinSeriesSort[\[source\]](#)

series_sort() – return the series sort value

[user_categories\(\)](#)

class calibre.utils.formatter_functions.BuiltinUserCategories[\[source\]](#)

user_categories() – return a comma-separated list of the user categories that contain this book. This function works only in the GUI. If you want to use these values in save-to-disk or send-to-device templates then you must make a custom “Column built from other columns”, use the function in that column’s template, and use that column’s value in your save/send templates

[virtual_libraries\(\)](#)

class calibre.utils.formatter_functions.BuiltinVirtualLibraries[\[source\]](#)

virtual_libraries() – return a comma-separated list of Virtual libraries that contain this book. This function works only in the GUI. If you want to use these values in save-to-disk or send-to-device templates then you must make a custom “Column built from other columns”, use the function in that column’s template, and use that column’s value in your save/send templates

If-then-else

check_yes_no(field_name, is_undefined, is_false, is_true)

class calibre.utils.formatter_functions.BuiltinCheckYesNo[\[source\]](#)

check_yes_no(field_name, is_undefined, is_false, is_true) – checks the value of the yes/no field named by the lookup key field_name for a value specified by the parameters, returning “yes” if a match is found, otherwise returning an empty string. Set the parameter is_undefined, is_false, or is_true to 1 (the number) to check that condition, otherwise set it to 0. Example: check_yes_no(“#bool”, 1, 0, 1) returns “yes” if the yes/no field “#bool” is either undefined (neither True nor False) or True. More than one of is_undefined, is_false, or is_true can be set to 1. This function is usually used by the test() or is_empty() functions.

contains(val, pattern, text if match, text if not match)

class calibre.utils.formatter_functions.BuiltinContains[\[source\]](#)

contains(val, pattern, text if match, text if not match) – checks if val contains matches for the regular expression *pattern*. Returns *text if match* if matches are found, otherwise it returns *text if no match*

field_exists(field_name)

class calibre.utils.formatter_functions.BuiltinFieldExists[\[source\]](#)

field_exists(field_name) – checks if a field (column) named field_name exists, returning ‘1’ if so and ‘’ if not.

ifempty(val, text if empty)

class calibre.utils.formatter_functions.BuiltinIfempty[\[source\]](#)

ifempty(val, text if empty) – return val if val is not empty, otherwise return *text if empty*

test(val, text if not empty, text if empty)

class calibre.utils.formatter_functions.BuiltinTest[\[source\]](#)

test(val, text if not empty, text if empty) – return *text if not empty* if val is not empty, otherwise return *text if empty*

Iterating over values

first_non_empty(value [, value]*)

`class calibre.utils.formatter_functions.BuiltinFirstNonEmpty`[\[source\]](#)

first_non_empty(value [, value]*) – returns the first value that is not empty. If all values are empty, then the empty string is returned. You can have as many values as you want.

lookup(val, [pattern, field,]+ else field)

`class calibre.utils.formatter_functions.BuiltinLookup`[\[source\]](#)

lookup(val, [pattern, field,]+ else_field) – like switch, except the arguments are field (metadata) names, not text. The value of the appropriate field will be fetched and used. Note that because composite columns are fields, you can use this function in one composite field to use the value of some other composite field. This is extremely useful when constructing variable save paths

switch(val, [pattern, value,]+ else value)

`class calibre.utils.formatter_functions.BuiltinSwitch`[\[source\]](#)

switch(val, [pattern, value,]+ else_value) – for each *pattern, value* pair, checks if *val* matches the regular expression *pattern* and if so, returns that *value*. If no pattern matches, then *else_value* is returned. You can have as many *pattern, value* pairs as you want

List lookup

identifier_in_list(val, id, found val, not found val)

`class calibre.utils.formatter_functions.BuiltinIdentifierInList`[\[source\]](#)

identifier_in_list(val, id, found_val, not_found_val) – treat val as a list of identifiers separated by commas, comparing the string against each value in the list. An identifier has the format “identifier:value”. The id parameter should be either “id” or “id:regexp”. The first case matches if there is any identifier with that id. The second case matches if the regexp matches the identifier’s value. If there is a match, return found_val, otherwise return not_found_val.

in_list(val, separator, [pattern, found val,]+ not found val)

`class calibre.utils.formatter_functions.BuiltinInList`[\[source\]](#)

`in_list(val, separator, [ pattern, found_val, ]+ not_found_val)` – treating `val` as a list of items separated by `separator`, if the pattern matches any of the list values then return `found_val`. If the pattern matches no list value then return `not_found_val`. The pattern and `found_value` pairs can be repeated as many times as desired. The patterns are checked in order. The `found_val` for the first match is returned. Aliases: `in_list()`, `list_contains()`

[list_item\(val, index, separator\)](#)

`class calibre.utils.formatter_functions.BuiltinListitem`[\[source\]](#)

`list_item(val, index, separator)` – interpret the value as a list of items separated by `separator`, returning the *index`th* item. *The first item is number zero. The last item can be returned using `list_item(-1,separator)*. If the item is not in the list, then the empty value is returned. The `separator` has the same meaning as in the `count` function.

[select\(val, key\)](#)

`class calibre.utils.formatter_functions.BuiltinSelect`[\[source\]](#)

`select(val, key)` – interpret the value as a comma-separated list of items, with the items being “id:value”. Find the pair with the id equal to `key`, and return the corresponding value. Returns the empty string if no match is found.

[str in list\(val, separator, \[string, found val, \]+ not found val\)](#)

`class calibre.utils.formatter_functions.BuiltinStrInList`[\[source\]](#)

`str_in_list(val, separator, [string, found_val, ]+ not_found_val)` – treating `val` as a list of items separated by `separator`, if the string matches any of the list values then return `found_val`. If the string matches no list value then return `not_found_val`. The comparison is exact match (not contains) and is case insensitive. The string and `found_value` pairs can be repeated as many times as desired. The patterns are checked in order. The `found_val` for the first match is returned.

[List manipulation](#)

[count\(val, separator\)](#)

`class calibre.utils.formatter_functions.BuiltinCount`[\[source\]](#)

`count(val, separator)` – interprets the value as a list of items separated by `separator`, returning the number of items in the list. Most lists use a comma as the separator, but authors uses an ampersand. Examples: `{tags:count(,)}`, `{authors:count(&)}`. Aliases: `count()`, `list_count()`

[list_count_matching\(list, pattern, separator\)](#)

class calibre.utils.formatter_functions.BuiltinListCountMatching[\[source\]](#)

list_count_matching(list, pattern, separator) – interprets ‘list’ as a list of items separated by ‘separator’, returning the number of items in the list that match the regular expression ‘pattern’. Aliases: list_count_matching(), count_matching()

list difference(list1, list2, separator)

class calibre.utils.formatter_functions.BuiltinListDifference[\[source\]](#)

list_difference(list1, list2, separator) – return a list made by removing from list1 any item found in list2, using a case-insensitive comparison. The items in list1 and list2 are separated by separator, as are the items in the returned list.

list equals(list1, sep1, list2, sep2, yes_val, no_val)

class calibre.utils.formatter_functions.BuiltinListEquals[\[source\]](#)

list_equals(list1, sep1, list2, sep2, yes_val, no_val) – return yes_val if list1 and list2 contain the same items, otherwise return no_val. The items are determined by splitting each list using the appropriate separator character (sep1 or sep2). The order of items in the lists is not relevant. The comparison is case insensitive.

list intersection(list1, list2, separator)

class calibre.utils.formatter_functions.BuiltinListIntersection[\[source\]](#)

list_intersection(list1, list2, separator) – return a list made by removing from list1 any item not found in list2, using a case-insensitive comparison. The items in list1 and list2 are separated by separator, as are the items in the returned list.

list re(src list, separator, include re, opt replace)

class calibre.utils.formatter_functions.BuiltinListRe[\[source\]](#)

list_re(src_list, separator, include_re, opt_replace) – Construct a list by first separating src_list into items using the separator character. For each item in the list, check if it matches include_re. If it does, then add it to the list to be returned. If opt_replace is not the empty string, then apply the replacement before adding the item to the returned list.

list re group(src list, separator, include re, search re [, group template]+)

class calibre.utils.formatter_functions.BuiltinListReGroup[\[source\]](#)

list_re_group(src_list, separator, include_re, search_re [, group_template]+) – Like list_re except replacements are not optional. It uses re_group(list_item, search_re, group_template, ...) when doing the replacements on the resulting list.

[list_remove_duplicates\(list, separator\)](#)

class calibre.utils.formatter_functions.BuiltinListRemoveDuplicates[\[source\]](#)

list_remove_duplicates(list, separator) – return a list made by removing duplicate items in the source list. If items differ only in case, the last of them is returned. The items in source list are separated by separator, as are the items in the returned list.

[list_sort\(list, direction, separator\)](#)

class calibre.utils.formatter_functions.BuiltinListSort[\[source\]](#)

list_sort(list, direction, separator) – return list sorted using a case-insensitive sort. If direction is zero, the list is sorted ascending, otherwise descending. The list items are separated by separator, as are the items in the returned list.

[list_split\(list_val, sep, id_prefix\)](#)

class calibre.utils.formatter_functions.BuiltinListSplit[\[source\]](#)

list_split(list_val, sep, id_prefix) – splits the list_val into separate values using ‘sep’, then assigns the values to variables named ‘id_prefix_N’ where N is the position of the value in the list. The first item has position 0 (zero). The function returns the last element in the list. Example: split(‘one:two:foo’, ‘:’, ‘var’) is equivalent to var_0 = ‘one’; var_1 = ‘two’; var_2 = ‘foo’.

[list_union\(list1, list2, separator\)](#)

class calibre.utils.formatter_functions.BuiltinListUnion[\[source\]](#)

list_union(list1, list2, separator) – return a list made by merging the items in list1 and list2, removing duplicate items using a case-insensitive comparison. If items differ in case, the one in list1 is used. The items in list1 and list2 are separated by separator, as are the items in the returned list. Aliases: list_union(), merge_lists()

[subitems\(val, start_index, end_index\)](#)

class calibre.utils.formatter_functions.BuiltinSubitems[\[source\]](#)

subitems(val, start_index, end_index) – This function is used to break apart lists of items such as genres. It interprets the value as a comma-separated list of items, where each item is a period-separated list. Returns a new list made by first finding all the period-separated items, then for each such item extracting the *start_index* to the *end_index* components, then combining the results back together. The first component in a period-separated list has an index of zero. If an index is negative, then it counts from the end of the list. As a special case, an end_index of zero is assumed to be the length of the list. Example using basic template

mode and assuming a `#genre` value of "A.B.C": `{#genre:subitems(0,1)}` returns "A". `{#genre:subitems(0,2)}` returns "A.B". `{#genre:subitems(1,0)}` returns "B.C". Assuming a `#genre` value of "A.B.C, D.E.F", `{#genre:subitems(0,1)}` returns "A, D". `{#genre:subitems(0,2)}` returns "A.B, D.E"

[sublist\(val, start_index, end_index, separator\)](#)

class calibre.utils.formatter_functions.BuiltinSublist[\[source\]](#)

`sublist(val, start_index, end_index, separator)` – interpret the value as a list of items separated by *separator*, returning a new list made from the *start_index* to the *end_index* item. The first item is number zero. If an index is negative, then it counts from the end of the list. As a special case, an *end_index* of zero is assumed to be the length of the list. Examples using basic template mode and assuming that the tags column (which is comma-separated) contains "A, B, C": `{tags:sublist(0,1,\,)}` returns "A". `{tags:sublist(-1,0,\,)}` returns "C". `{tags:sublist(0,-1,\,)}` returns "A, B".

[Other](#)

[assign\(id, val\)](#)

class calibre.utils.formatter_functions.BuiltinAssign[\[source\]](#)

`assign(id, val)` – assigns *val* to *id*, then returns *val*. *id* must be an identifier, not an expression. This function can often be replaced with the = operator.

[print\(a\[, b\]*\)](#)

class calibre.utils.formatter_functions.BuiltinPrint[\[source\]](#)

`print(a[, b]*)` – prints the arguments to standard output. Unless you start calibre from the command line (`calibre-debug -g`), the output will go to a black hole.

[Recursion](#)

[eval\(template\)](#)

class calibre.utils.formatter_functions.BuiltinEval[\[source\]](#)

`eval(template)` – evaluates the template, passing the local variables (those 'assign'ed to) instead of the book metadata. This permits using the template processor to construct complex results from local variables. Because the { and } characters are special, you must use `[[` for the { character and `]]` for the } character; they are converted automatically. Note also that prefixes and suffixes (the */prefix/suffix* syntax) cannot be used in the argument to this function when using template program mode.

[template\(x\)](#)

class calibre.utils.formatter_functions.BuiltinTemplate[\[source\]](#)

template(x) – evaluates x as a template. The evaluation is done in its own context, meaning that variables are not shared between the caller and the template evaluation. Because the { and } characters are special, you must use [[for the { character and]] for the } character; they are converted automatically. For example, template("[[title_sort]]") will evaluate the template {title_sort} and return its value. Note also that prefixes and suffixes (the /prefix/suffix syntax) cannot be used in the argument to this function when using template program mode.

[Relational](#)

[cmp\(x, y, lt, eq, gt\)](#)

class calibre.utils.formatter_functions.BuiltinCmp[\[source\]](#)

cmp(x, y, lt, eq, gt) – compares x and y after converting both to numbers. Returns lt if x < y. Returns eq if x == y. Otherwise returns gt. In many cases the numeric comparison operators (>#, <#, ==# etc) can replace this function.

[first_matching_cmp\(val, \[cmp1, result1,\]+, else_result\)](#)

class calibre.utils.formatter_functions.BuiltinFirstMatchingCmp[\[source\]](#)

first_matching_cmp(val, [cmp1, result1,]+, else_result) – compares “val < cmpN” in sequence, returning resultN for the first comparison that succeeds. Returns else_result if no comparison succeeds. Example:
first_matching_cmp(10,5,"small",10,"middle",15,"large","giant") returns “large”. The same example with a first value of 16 returns “giant”.

[strcmp\(x, y, lt, eq, gt\)](#)

class calibre.utils.formatter_functions.BuiltinStrcmp[\[source\]](#)

strcmp(x, y, lt, eq, gt) – does a case-insensitive comparison of x and y as strings. Returns lt if x < y. Returns eq if x == y. Otherwise returns gt. In many cases the lexical comparison operators (>, <, == etc) can replace this function.

[String case changes](#)

[capitalize\(val\)](#)

class calibre.utils.formatter_functions.BuiltinCapitalize[\[source\]](#)

capitalize(val) – return val capitalized

lowercase(val)

class calibre.utils.formatter_functions.BuiltinLowercase[\[source\]](#)

lowercase(val) – return val in lower case

titlecase(val)

class calibre.utils.formatter_functions.BuiltinTitlecase[\[source\]](#)

titlecase(val) – return val in title case

uppercase(val)

class calibre.utils.formatter_functions.BuiltinUppercase[\[source\]](#)

uppercase(val) – return val in upper case

String manipulation

character(character name)

class calibre.utils.formatter_functions.BuiltinCharacter[\[source\]](#)

character(character_name) – returns the character named by character_name. For example, character('newline') returns a newline character ('n'). The supported character names are 'newline', 'return', 'tab', and 'backslash'.

re(val, pattern, replacement)

class calibre.utils.formatter_functions.BuiltinRe[\[source\]](#)

re(val, pattern, replacement) – return val after applying the regular expression. All instances of *pattern* are replaced with *replacement*. As in all of calibre, these are Python-compatible regular expressions

re_group(val, pattern [, template for group]*)

class calibre.utils.formatter_functions.BuiltinReGroup[\[source\]](#)

re_group(val, pattern [, template_for_group]*) – return a string made by applying the regular expression pattern to the val and replacing each matched instance with the string computed by replacing each matched group by the value returned by the corresponding template. The original matched value for the group is available as \$. In template program mode, like for the template and the eval functions, you use [[for { and]] for }. The following example in

template program mode looks for series with more than one word and uppercases the first word: {series:'re_group(\$, "(S*)(.*)", "[[[:uppercase()]]", "[[[:\$]]")'}}

[shorten\(val, left chars, middle text, right chars\)](#)

`class calibre.utils.formatter_functions.BuiltinShorten`[\[source\]](#)

`shorten(val, left chars, middle text, right chars)` – Return a shortened version of `val`, consisting of *left chars* characters from the beginning of `val`, followed by *middle text*, followed by *right chars* characters from the end of the string. *Left chars* and *right chars* must be integers. For example, assume the title of the book is *Ancient English Laws in the Times of Ivanhoe*, and you want it to fit in a space of at most 15 characters. If you use {title:shorten(9,-,5)}, the result will be *Ancient E-anhoe*. If the field's length is less than `left chars + right chars` + the length of *middle text*, then the field will be used intact. For example, the title *The Dome* would not be changed.

[strcat\(a \[, b\]*\)](#)

`class calibre.utils.formatter_functions.BuiltinStrcat`[\[source\]](#)

`strcat(a [, b]*)` – can take any number of arguments. Returns the string formed by concatenating all the arguments

[strcat_max\(max, string1 \[, prefix2, string2\]*\)](#)

`class calibre.utils.formatter_functions.BuiltinStrcatMax`[\[source\]](#)

`strcat_max(max, string1 [, prefix2, string2]*)` – Returns a string formed by concatenating the arguments. The returned value is initialized to `string1`. *Prefix, string* pairs are added to the end of the value as long as the resulting string length is less than *max*. `string1` is returned even if `string1` is longer than *max*. You can pass as many *prefix, string* pairs as you wish.

[strlen\(a\)](#)

`class calibre.utils.formatter_functions.BuiltinStrlen`[\[source\]](#)

`strlen(a)` – Returns the length of the string passed as the argument

[substr\(str, start, end\)](#)

`class calibre.utils.formatter_functions.BuiltinSubstr`[\[source\]](#)

`substr(str, start, end)` – returns the start'th through the end'th characters of `str`. The first character in `str` is the zero'th character. If `end` is negative, then it indicates that many characters counting from the right. If `end` is zero, then it indicates the last character. For example, `substr('12345', 1, 0)` returns '2345', and `substr('12345', 1, -1)` returns '234'.

[swap_around_articles\(val, separator\)](#)

class calibre.utils.formatter_functions.BuiltinSwapAroundArticles[\[source\]](#)

swap_around_articles(val, separator) – returns the val with articles moved to the end. The value can be a list, in which case each member of the list is processed. If the value is a list then you must provide the list value separator. If no separator is provided then the value is treated as being a single value, not a list.

[swap_around_comma\(val\)](#)

class calibre.utils.formatter_functions.BuiltinSwapAroundComma[\[source\]](#)

swap_around_comma(val) – given a value of the form “B, A”, return “A B”. This is most useful for converting names in LN, FN format to FN LN. If there is no comma, the function returns val unchanged

[transliterate\(a\)](#)

class calibre.utils.formatter_functions.BuiltinTransliterate[\[source\]](#)

transliterate(a) – Returns a string in a latin alphabet formed by approximating the sound of the words in the source string. For example, if the source is “Фёдор Миха́йлович Достоевский” the function returns “Fiodor Mikhailovich Dostoevskii”.

[other](#)

[arguments\(id\[=expression\] \[, id\[=expression\]\]*\)](#)

class calibre.utils.formatter_functions.BuiltinArguments[\[source\]](#)

arguments(id[=expression] [, id[=expression]]*) – Used in a stored template to retrieve the arguments passed in the call. It both declares and initializes local variables, effectively parameters. The variables are positional; they get the value of the value given in the call in the same position. If the corresponding parameter is not provided in the call then arguments assigns that variable the provided default value. If there is no default value then the variable is set to the empty string.

[globals\(id\[=expression\] \[, id\[=expression\]\]*\)](#)

class calibre.utils.formatter_functions.BuiltinSetGlobals[\[source\]](#)

globals(id[=expression] [, id[=expression]]*) – Retrieves “global variables” that can be passed into the formatter. It both declares and initializes local variables with the names of the global variables passed in. If the corresponding variable is not provided in the passed-in

globals then it assigns that variable the provided default value. If there is no default value then the variable is set to the empty string.

API of the Metadata objects

The python implementation of the template functions is passed in a Metadata object. Knowing it's API is useful if you want to define your own template functions.

```
class calibre.ebooks.metadata.book.base.Metadata(title, authors=('Unknown'), other=None, template_cache=None, formatter=None)\[source\]
```

A class representing all the metadata for a book. The various standard metadata fields are available as attributes of this object. You can also stick arbitrary attributes onto this object.

Metadata from custom columns should be accessed via the `get()` method, passing in the lookup name for the column, for example: `"#mytags"`.

Use `is_null()` the `is_null()` method to test if a field is null.

This object also has functions to format fields into strings.

The list of standard metadata fields grows with time is in `STANDARD_METADATA_FIELDS`.

Please keep the method based API of this class to a minimum. Every method becomes a reserved field name.

```
is_null(field)\[source\]
```

Return True if the value of field is null in this object. 'null' means it is unknown or evaluates to False. So a title of `_(('Unknown'))` is null or a language of `'und'` is null.

Be careful with numeric fields since this will return True for zero as well as None.

Also returns True if the field does not exist.

```
deepcopy(class_generator=<function Metadata.<lambda>>)\[source\]
```

Do not use this method unless you know what you are doing, if you want to create a simple clone of this object, use `deepcopy_metadata()` instead. `Class_generator` must be a function that returns an instance of Metadata or a subclass of it.

`get_identifiers()`[\[source\]](#)

Return a copy of the identifiers dictionary. The dict is small, and the penalty for using a reference where a copy is needed is large. Also, we don't want any manipulations of the returned dict to show up in the book.

`set_identifiers(identifiers)`[\[source\]](#)

Set all identifiers. Note that if you previously set ISBN, calling this method will delete it.

`set_identifier(typ, val)`[\[source\]](#)

If *val* is empty, deletes identifier of type *typ*

`standard_field_keys()`[\[source\]](#)

return a list of all possible keys, even if this book doesn't have them

`custom_field_keys()`[\[source\]](#)

return a list of the custom fields in this book

`all_field_keys()`[\[source\]](#)

All field keys known by this instance, even if their value is None

`metadata_for_field(key)`[\[source\]](#)

return metadata describing a standard or custom field.

`all_non_none_fields()`[\[source\]](#)

Return a dictionary containing all non-None metadata fields, including the custom ones.

`get_standard_metadata(field, make_copy)`[\[source\]](#)

return field metadata from the field if it is there. Otherwise return None. *field* is the key name, not the label. Return a copy if requested, just in case the user wants to change values in the dict.

`get_all_standard_metadata(make_copy)`[\[source\]](#)

return a dict containing all the standard field metadata associated with the book.

`get_all_user_metadata(make_copy)`[\[source\]](#)

return a dict containing all the custom field metadata associated with the book.

`get_user_metadata(field, make_copy)[source]`

return field metadata from the object if it is there. Otherwise return None. field is the key name, not the label. Return a copy if requested, just in case the user wants to change values in the dict.

`set_all_user_metadata(metadata)[source]`

store custom field metadata into the object. Field is the key name not the label

`set_user_metadata(field, metadata)[source]`

store custom field metadata for one column into the object. Field is the key name not the label

`remove_stale_user_metadata(other_mi)[source]`

Remove user metadata keys (custom column keys) if they don't exist in 'other_mi', which must be a metadata object

`template_to_attribute(other, ops)[source]`

Takes a list [(src,dest), (src,dest)], evaluates the template in the context of other, then copies the result to self[dest]. This is on a best-efforts basis. Some assignments can make no sense.

`smart_update(other, replace_metadata=False)[source]`

Merge the information in *other* into self. In case of conflicts, the information in *other* takes precedence, unless the information in *other* is NULL.

`format_field(key, series_with_index=True)[source]`

Returns the tuple (display_name, formatted_value)

`to_html()[source]`

A HTML representation of this object.

`calibre.ebooks.metadata.book.base.STANDARD_METADATA_FIELDS`

The set of standard metadata fields.

'''

All fields must have a NULL value represented as None for simple types,

an empty list/dictionary for complex types and (None, None) for cover_data

'''

SOCIAL_METADATA_FIELDS

=

frozenset

((

'tags'

,

Ordered list

'rating'

,

A floating point number between 0 and 10

'comments'

,

A simple HTML enabled string

'series'

,

A simple string

'series_index'

,

A floating point number

```
# Of the form { scheme1:value1, scheme2:value2}
```

```
# For example: {'isbn':'123456789', 'doi':'xxx', ... }
```

```
'identifiers'
```

```
,
```

```
))
```

```
'''
```

```
The list of names that convert to identifiers when in get and set.
```

```
'''
```

```
TOP_LEVEL_IDENTIFIERS
```

```
=
```

```
frozenset
```

```
((
```

```
'isbn'
```

```
,
```

```
))
```

```
PUBLICATION_METADATA_FIELDS
```

```
=
```

```
frozenset
```

```
((
```

```
'title'
```

```
,  
  
# title must never be None. Should be _('Unknown')  
  
# Pseudo field that can be set, but if not set is auto generated  
  
# from title and languages  
  
'title_sort'  
,  
  
'authors'  
,  
  
# Ordered list. Must never be None, can be [_('Unknown')]  
  
'author_sort_map'  
,  
  
# Map of sort strings for each author  
  
# Pseudo field that can be set, but if not set is auto generated  
  
# from authors and languages  
  
'author_sort'  
,  
  
'book_producer'  
,  
  
'timestamp'  
,
```

Dates and times must be timezone aware

'pubdate'

,

'last_modified'

,

'rights'

,

So far only known publication type is periodical:calibre

If None, means book

'publication_type'

,

'uuid'

,

A UUID usually of type 4

'languages'

,

ordered list of languages in this publication

'publisher'

,

Simple string, no special semantics

```
# Absolute path to image file encoded in filesystem_encoding
```

```
'cover'
```

```
,
```

```
# Of the form (format, data) where format is, e.g. 'jpeg', 'png', 'gif'...
```

```
'cover_data'
```

```
,
```

```
# Either thumbnail data, or an object with the attribute
```

```
# image_path which is the path to an image file, encoded
```

```
# in filesystem_encoding
```

```
'thumbnail'
```

```
,
```

```
))
```

```
BOOK_STRUCTURE_FIELDS
```

```
=
```

```
frozenset
```

```
((
```

```
# These are used by code, Null values are None.
```

```
'toc'
```

```
,
```

```
'spine'
```

```
,
```

```
'guide'
```

```
,
```

```
'manifest'
```

```
,
```

```
))
```

```
USER_METADATA_FIELDS
```

```
=
```

```
frozenset
```

```
((
```

```
# A dict of dicts similar to field_metadata. Each field description dict
```

```
# also contains a value field with the key #value#.
```

```
'user_metadata'
```

```
,
```

```
))
```

```
DEVICE_METADATA_FIELDS
```

```
=
```

```
frozenset
```

```
((
```

```
'device_collections'
```

```
,
```



```
# Ordered list of strings
```

```
'lpath'
```

```
,
```

```
# Unicode, / separated
```

```
'size'
```

```
,
```

```
# In bytes
```

```
'mime'
```

```
,
```

```
# Mimetype of the book file being represented
```

```
))
```

```
CALIBRE_METADATA_FIELDS
```

```
=
```

```
frozenset
```

```
((
```

```
'application_id'
```

```
,
```

```
# An application id, currently set to the db_id.
```

```
'db_id'
```

```
,
```

```
# the calibre primary key of the item.
```

```
'formats'
',

# list of formats (extensions) for this book

# a dict of user category names, where the value is a list of item names

# from the book that are in that category

'user_categories'
',

# a dict of author to an associated hyperlink

'author_link_map'
',

))

ALL_METADATA_FIELDS

=

SOCIAL_METADATA_FIELDS
.
union
(

PUBLICATION_METADATA_FIELDS
)
.
union
(
```

```
BOOK_STRUCTURE_FIELDS
```

```
)
```

```
.
```

```
union
```

```
(
```

```
USER_METADATA_FIELDS
```

```
)
```

```
.
```

```
union
```

```
(
```

```
DEVICE_METADATA_FIELDS
```

```
)
```

```
.
```

```
union
```

```
(
```

```
CALIBRE_METADATA_FIELDS
```

```
)
```

```
# All fields except custom fields
```

```
STANDARD_METADATA_FIELDS
```

```
=
```

```
SOCIAL_METADATA_FIELDS
```

```
.
```

```
union
```

```
(
```

```
PUBLICATION_METADATA_FIELDS
```

```
)
```

```
.
```

```
union
```

```
(
```

```
BOOK_STRUCTURE_FIELDS
)
```

```
.
union
(
```

```
DEVICE_METADATA_FIELDS
)
```

```
.
union
(
```

```
CALIBRE_METADATA_FIELDS
)
```

```
# Metadata fields that smart update must do special processing to copy.
```

```
SC_FIELDS_NOT_COPIED
```

```
=
```

```
frozenset
```

```
((
```

```
'title'
```

```
,
```

```
'title_sort'
```

```
,
```

```
'authors'
```

```
,
```

```
'author_sort'
```

```
,
```

```
'author_sort_map'
```

```
,
```

```
'cover_data'
```

```
,
```

```
'tags'
```

```
,
```

```
'languages'
```

```
,
```

```
'identifiers'
```

```
))
```

```
# Metadata fields that smart update should copy only if the source is not None
```

```
SC_FIELDS_COPY_NOT_NULL
```

```
=
```

```
frozenset
```

```
((
```

```
'device_collections'
```

```
,
```

```
'lpath'
```

```
,
```

```
'size'
```

```
,
```

```
'comments'
```

```
,
```

```
'thumbnail'
```

```
))
```

```
# Metadata fields that smart update should copy without special handling
```

```
SC_COPYABLE_FIELDS
```

```
=  
  
SOCIAL_METADATA_FIELDS  
.  
union  
(  
  
PUBLICATION_METADATA_FIELDS  
)  
.  
union  
(  
  
BOOK_STRUCTURE_FIELDS  
)  
.  
union  
(  
  
DEVICE_METADATA_FIELDS  
)  
.  
union  
(  
  
CALIBRE_METADATA_FIELDS  
)  
-  
\  
  
SC_FIELDS_NOT_COPIED  
.  
union  
(  
  
SC_FIELDS_COPY_NOT_NULL  
)
```

SERIALIZABLE_FIELDS

=

SOCIAL_METADATA_FIELDS

.

union

(

USER_METADATA_FIELDS

)

.

union

(

PUBLICATION_METADATA_FIELDS

)

.

union

(

CALIBRE_METADATA_FIELDS

)

.

union

(

DEVICE_METADATA_FIELDS

)

-

\

frozenset

((

'device_collections'

,

```
'formats'
```

```
,
```

```
'cover_data'
```

```
))
```

```
# these are rebuilt when needed
```


All about using regular expressions in calibre

Regular expressions are features used in many places in calibre to perform sophisticated manipulation of e-book content and metadata. This tutorial is a gentle introduction to getting you started with using regular expressions in calibre.

Contents

- [First, a word of warning and a word of courage](#)
- [Where in calibre can you use regular expressions?](#)
- [What on earth *is* a regular expression?](#)
- [Care to explain?](#)
- [That doesn't sound too bad. What's next?](#)
- [Hey, neat! This is starting to make sense!](#)
- [Well, these special characters are very neat and all, but what if I wanted to match a dot or a question mark?](#)
- [So, what are the most useful sets?](#)
- [But if I had a few varying strings I wanted to match, things get complicated?](#)
- [You missed...](#)
- [In the beginning, you said there was a way to make a regular expression case insensitive?](#)
- [I think I'm beginning to understand these regular expressions now... how do I use them in calibre?](#)
 - [Conversions](#)
 - [Adding books](#)

- [Bulk editing metadata](#)
- [Quick reference](#)
- [Credits](#)

First, a word of warning and a word of courage

This is, inevitably, going to be somewhat technical- after all, regular expressions are a technical tool for doing technical stuff. I'm going to have to use some jargon and concepts that may seem complicated or convoluted. I'm going to try to explain those concepts as clearly as I can, but really can't do without using them at all. That being said, don't be discouraged by any jargon, as I've tried to explain everything new. And while regular expressions themselves may seem like an arcane, black magic (or, to be more prosaic, a random string of mumbo-jumbo letters and signs), I promise that they are not all that complicated. Even those who understand regular expressions really well have trouble reading the more complex ones, but writing them isn't as difficult- you construct the expression step by step. So, take a step and follow me into the rabbit hole.

Where in calibre can you use regular expressions?

There are a few places calibre uses regular expressions. There's the *Search & replace* in conversion options, metadata detection from filenames in the import settings and Search & replace when editing the metadata of books in bulk. The calibre book editor can also use regular expressions in its *Search and replace* feature. Finally, you can use regular expressions when searching the calibre book list and when searching inside the calibre E-book viewer.

What on earth is a regular expression?

A regular expression is a way to describe sets of strings. A single regular expression can *match* a number of different strings. This is what makes regular expression so powerful – they are a concise way of describing a potentially large number of variations.

Note

I'm using string here in the sense it is used in programming languages: a string of one or more characters, characters including actual characters, numbers, punctuation and so-called whitespace (linebreaks, tabulators etc.). Please note that generally, uppercase and lowercase characters are not considered the same, thus "a" being a different character from "A" and so forth. In calibre, regular expressions are case insensitive in the Search bar, but not in the conversion options. There's a way to make every regular expression case insensitive, but we'll discuss that later. It gets complicated because regular expressions allow for variations in the strings it matches, so one expression can match multiple strings, which is why people bother using them at all. More on that in a bit.

Care to explain?

Well, that's why we're here. First, this is the most important concept in regular expressions: *A string by itself is a regular expression that matches itself.* That is to say, if I wanted to match the string "Hello,

World!"

using a regular expression, the regular expression to use would be Hello,

World!

. And yes, it really is that simple. You'll notice, though, that this *only* matches the exact string "Hello,

World!"

, not e.g. "Hello,

wOrld!"

or "hello,

world!"

or any other such variation.

That doesn't sound too bad. What's next?

Next is the beginning of the really good stuff. Remember where I said that regular expressions can match multiple strings? This is where it gets a little more complicated. Say, as a somewhat more practical exercise, the e-book you wanted to convert had a nasty footer counting the pages, like "Page 5 of 423". Obviously the page number would rise from 1 to 423, thus you'd have to match 423 different strings, right? Wrong, actually: regular expressions allow you to define sets of characters that are matched: To define a set, you put all the characters you want to be in the set into square brackets. So, for example, the set [abc] would match either the character "a", "b" or "c". *Sets will always only match one of the characters in the set.* They "understand" character ranges, that is, if you wanted to match all the lower case characters, you'd use the set [a-z] for lower- and uppercase characters you'd use [a-zA-Z] and so on. Got the idea? So, obviously, using the expression Page

[0-9]

of

423

you'd be able to match the first 9 pages, thus reducing the expressions needed to three: The second expression Page

[0-9][0-9]

of

423

would match all two-digit page numbers, and I'm sure you can guess what the third expression would look like. Yes, go ahead. Write it down.

Hey, neat! This is starting to make sense!

I was hoping you'd say that. But brace yourself, now it gets even better! We just saw that using sets, we could match one of several characters at once. But you can even repeat a character or set, reducing the number of expressions needed to handle the above page number example to one. Yes, ONE! Excited? You should be! It works like this: Some so-called special characters, "+", "?" and "*", *repeat the single element preceding them*. (Element means either a single character, a character set, an escape sequence or a group (we'll learn about those last two later)- in short, any single entity in a regular expression). These characters are called wildcards or quantifiers. To be more precise, "?" matches *0 or 1* of the preceding element, "*" matches *0 or more* of the preceding element and "+" matches *1 or more* of the preceding element. A few examples: The expression `a?` would match either "" (which is the empty string, not strictly useful in this case) or "a", the expression `a*` would match "", "a", "aa" or any number of a's in a row, and, finally, the expression `a+` would match "a", "aa" or any number of a's in a row (Note: it wouldn't match the empty string!). Same deal for sets: The expression `[0-9]+` would match *every integer number there is!* I know what you're thinking, and you're right: If you use that in the above case of matching page numbers, wouldn't that be the single one expression to match all the page numbers? Yes, the expression

`[0-9]+`

of

423

would match every page number in that book!

Note

A note on these quantifiers: They generally try to match as much text as possible, so be careful when using them. This is called "greedy behaviour"- I'm sure you get why. It gets problematic when you, say, try to match a tag. Consider, for example, the string "<p

`class="calibre2">Title`

here</p>"

and let's say you'd want to match the opening tag (the part between the first pair of angle brackets, a little more on tags later). You'd think that the expression `<p.*>` would match that tag, but actually, it matches the whole string! (The character "." is another special character. It matches anything *except* linebreaks, so, basically, the expression `.*` would match any single line you can think of). Instead, try using `<p.*?>` which makes the quantifier "non-greedy. That expression would only match the first opening tag, as intended. There's actually another way to accomplish this: The expression `<p[^>]*>` will match that same opening tag- you'll see why after the next section. Just note that there quite frequently is more than one way to write a regular expression.

Well, these special characters are very neat and all, but what if I wanted to match a dot or a question mark?

You can of course do that: Just put a backslash in front of any special character and it is interpreted as the literal character, without any special meaning. This pair of a backslash followed by a single character is called an escape sequence, and the act of putting a backslash in front of a special character is called escaping that character. An escape sequence is interpreted as a single element. There are of course escape sequences that do more than just escaping special characters, for example `"\t"` means a tabulator. We'll get to some of the escape sequences later. Oh, and by the way, concerning those special characters: Consider any character we discuss in this introduction as having some function to be special and thus needing to be escaped if you want the literal character.

So, what are the most useful sets?

Knew you'd ask. Some useful sets are [0-9]
matching a single number, [a-z]
matching a single lowercase letter, [A-Z]
matching a single uppercase letter, [a-zA-Z]
matching a single letter and [a-zA-Z0-9]
matching a single letter or number. You can also use an escape sequence as shorthand:

`\d`

is equivalent to [0-9]

`\w`

is equivalent to [a-zA-Z0-9_]

\s

is equivalent to any whitespace

Note

“Whitespace” is a term for anything that won’t be printed. These characters include space, tabulator, line feed, form feed, carriage return, non-breaking spaces, etc.

As a last note on sets, you can also define a set as any character *but* those in the set. You do that by including the character “^” as the *very first character in the set*. Thus, [^a] would match any character excluding “a”. That’s called complementing the set. Those escape sequence shorthands we saw earlier can also be complemented: “\D” means any non-number character, thus being equivalent to [^0-9]. The other shorthands can be complemented by, you guessed it, using the respective uppercase letter instead of the lowercase one. So, going back to the example <p[^>]*> from the previous section, now you can see that the character set it’s using tries to match any character except for a closing angle bracket.

But if I had a few varying strings I wanted to match, things get complicated?

Fear not, life still is good and easy. Consider this example: The book you’re converting has “Title” written on every odd page and “Author” written on every even page. Looks great in print, right? But in e-books, it’s annoying. You can group whole expressions in normal parentheses, and the character “|” will let you match *either* the expression to its right *or* the one to its left. Combine those and you’re done. Too fast for you? Okay, first off, we group the expressions for odd and even pages, thus getting (Title)(Author) as our two needed expressions. Now we make things simpler by using the vertical bar (“|” is called the vertical bar character): If you use the expression (Title|Author) you’ll either get a match for “Title” (on the odd pages) or you’d match “Author” (on the even pages). Well, wasn’t that easy?

You can, of course, use the vertical bar without using grouping parentheses, as well. Remember when I said that quantifiers repeat the element preceding them? Well, the vertical bar works a little differently: The expression “Title|Author” will also match either the string “Title” or the string “Author”, just as the above example using grouping. *The vertical bar selects between the entire expression preceding and following it.* So, if you wanted to match the strings “Calibre” and “calibre” and wanted to select only between the upper- and

lowercase “c”, you’d have to use the expression (c|C)alibre, where the grouping ensures that only the “c” will be selected. If you were to use c|Calibre, you’d get a match on the string “c” or on the string “Calibre”, which isn’t what we wanted. In short: If in doubt, use grouping together with the vertical bar.

You missed...

... wait just a minute, there’s one last, really neat thing you can do with groups. If you have a group that you previously matched, you can use references to that group later in the expression: Groups are numbered starting with 1, and you reference them by escaping the number of the group you want to reference, thus, the fifth group would be referenced as \5.

So, if you searched for ([^

\1

in the string “Test Test”, you’d match the whole string!

In the beginning, you said there was a way to make a regular expression case insensitive?

Yes, I did, thanks for paying attention and reminding me. You can tell calibre how you want certain things handled by using something called flags. You include flags in your expression by using the special construct (?flags

go

here)

where, obviously, you’d replace “flags go here” with the specific flags you want. For ignoring case, the flag is i, thus you include (?i) in your expression. Thus, (?i)test would match “Test”, “tEst”, “TEst” and any case variation you could think of.

Another useful flag lets the dot match any character at all, *including* the newline, the flag s. If you want to use multiple flags in an expression, just put them in the same statement: (?is) would ignore case and make the dot match all. It doesn’t matter which flag you state first, (?si) would be equivalent to the above.

I think I’m beginning to understand these regular expressions now... how do I use them in calibre?

Conversions

Let's begin with the conversion settings, which is really neat. In the *Search & replace* part, you can input a regexp (short for regular expression) that describes the string that will be replaced during the conversion. The neat part is the wizard. Click on the wizard staff and you get a preview of what calibre "sees" during the conversion process. Scroll down to the string you want to remove, select and copy it, paste it into the regexp field on top of the window. If there are variable parts, like page numbers or so, use sets and quantifiers to cover those, and while you're at it, remember to escape special characters, if there are some. Hit the button labeled *Test* and calibre highlights the parts it would replace were you to use the regexp. Once you're satisfied, hit OK and convert. Be careful if your conversion source has tags like this example:

Maybe
,
but
the
cops
feel
like
you
do
,
Anita
.
What
's one more dead vampire?
New
laws
don
't change that. </p>


```
<
p
class
=
"calibre4"
>

<
b
class
=
"calibre2"
>
Generated

by

ABC

Amber

LIT

Conv

<
a
href
=
"http://www.processtext.com/abclit.html"
class
=
"calibre3"
>
erter
,

http
```

```
:  
//  
www  
.  
processtext  
.  
com  
/  
abclit  
.  
html  
</  
a  
></  
b  
></  
p  
>  
  
<  
p  
  
class  
=  
"calibre4"  
>  
  
It  
  
had  
  
only  
  
been  
  
two  
  
years  
  
since  
  
Addison
```

v
.
Clark
.
The
court
case
gave
us
a
revised
version
of
what
life
was

(shamelessly ripped out of [this thread](https://www.mobileread.com/forums/showthread.php?t=75594) [https://www.mobileread.com/forums/showthread.php?t=75594"]).
You'd have to remove some of the tags as well. In this example, I'd recommend beginning
with the tag <b

class="calibre2">

, now you have to end with the corresponding closing tag (opening tags are <tag>
, closing tags are </tag>
, which is simply the next
in this case. (Refer to a good HTML manual or ask in the forum if you are unclear on this
point). The opening tag can be described using <b.*?>
, the closing tag using

, thus we could remove everything between those tags using `<b.*?>..*?</b>`. But using this expression would be a bad idea, because it removes everything enclosed by `<b>`- tags (which, by the way, render the enclosed text in bold print), and it's a fair bet that we'll remove portions of the book in this way. Instead, include the beginning of the enclosed string as well, making the regular expression `<b.*?>\s*Generated\s+by\s+ABC\s+Amber\s+LIT.*?</b>`

The \s with quantifiers are included here instead of explicitly using the spaces as seen in the string to catch any variations of the string that might occur. Remember to check what calibre will remove to make sure you don't remove any portions you want to keep if you test a new expression. If you only check one occurrence, you might miss a mismatch somewhere else in the text. Also note that should you accidentally remove more or fewer tags than you actually wanted to, calibre tries to repair the damaged code after doing the removal.

[Adding books](#)

Another thing you can use regular expressions for is to extract metadata from filenames. You can find this feature in the "Adding books" part of the settings. There's a special feature here: You can use field names for metadata fields, for example `(?P<title>)` would indicate that calibre uses this part of the string as book title. The allowed field names are listed in the windows, together with another nice test field. An example: Say you want to import a whole bunch of files named like Classical

Texts:

The

Divine

Comedy

by

Dante

Alighieri.mobi

. (Obviously, this is already in your library, since we all love classical italian poetry) or Science

Fiction

epics:

The

Foundation

Trilogy

by

Isaac

Asimov.epub

. This is obviously a naming scheme that calibre won't extract any meaningful data out of - its standard expression for extracting metadata is (?P<title>.+)

-

(?P<author>[^_]+)

. A regular expression that works here would be [a-zA-Z]+:

(?P<title>.+)

by

(?P<author>.+)

. Please note that, inside the group for the metadata field, you need to use expressions to describe what the field actually matches. And also note that, when using the test field calibre provides, you need to add the file extension to your testing filename, otherwise you won't get any matches at all, despite using a working expression.

[Bulk editing metadata](#)

The last part is regular expression *Search and replace* in metadata fields. You can access this by selecting multiple books in the library and using bulk metadata edit. Be very careful when using this last feature, as it can do **Very Bad Things** to your library! Doublecheck that your expressions do what you want them to using the test fields, and only mark the books you really want to change! In the regular expression search mode, you can search in one field, replace the text with something and even write the result into another field. A practical example: Say your library contained the books of Frank Herbert's Dune series, named after the fashion Dune

1

-

Dune

,

Dune

2

-

Dune

Messiah

and so on. Now you want to get Dune
into the series field. You can do that by searching for (.*?)

\d+

-

.*

in the title field and replacing it with \1
in the series field. See what I did there? That's a reference to the first group you're replacing
the series field with. Now that you have the series all set, you only need to do another search
for .*?

-

in the title field and replace it with ""
(an empty string), again in the title field, and your metadata is all neat and tidy. Isn't that
great? By the way, instead of replacing the entire field, you can also append or prepend to
the field, so, if you *wanted* the book title to be prepended with series info, you could do that
as well. As you by now have undoubtedly noticed, there's a checkbox labeled *Case sensitive*,
so you won't have to use flags to select behaviour here.

Well, that just about concludes the very short introduction to regular expressions. Hopefully
I'll have shown you enough to at least get you started and to enable you to continue learning
by yourself- a good starting point would be the [Python documentation for regexps](https://docs.python.org/library/re.html)
[<https://docs.python.org/library/re.html>].

One last word of warning, though: Regexps are powerful, but also really easy to get wrong.
calibre provides really great testing possibilities to see if your expressions behave as you
expect them to. Use them. Try not to shoot yourself in the foot. (God, I love that expression...).
But should you, despite the warning, injure your foot (or any other body parts), try to learn
from it.

[Quick reference](#)

- [Quick reference for regexp syntax](#)
 - [Character classes](#)

- [Shorthand character classes](#)
- [The quantifiers](#)
- [Greed](#)
- [Alternation](#)
- [Exclusion](#)
- [Anchors](#)
- [Groups](#)
- [Lookarounds](#)
- [Recursion](#)
- [Special characters](#)
- [Meta-characters](#)
- [Modes](#)

Credits

Thanks for helping with tips, corrections and such:

- Idolse
- kovidgoyal
- chaley
- dwanthny
- kacir
- Starson17
- Orpheu

For more about regexps see [The Python User Manual](https://docs.python.org/library/re.html) [https://docs.python.org/library/re.html]. The actual regular expression library used by calibre is: [regex](https://bitbucket.org/mrabarnett/mrab-regex/src/hg/) [https://bitbucket.org/mrabarnett/mrab-regex/src/hg/] which supports several useful enhancements over the Python standard library one.

Quick reference for regexp syntax

This checklist summarizes the most commonly used/hard to remember parts of the regexp engine available in most parts of calibre.

Contents

- [Character classes](#)
- [Shorthand character classes](#)
- [The quantifiers](#)
- [Greed](#)
- [Alternation](#)
- [Exclusion](#)
- [Anchors](#)
- [Groups](#)
- [Lookarounds](#)
- [Recursion](#)
- [Special characters](#)
- [Meta-characters](#)
- [Modes](#)

[Character classes](#)

Character classes are useful to represent different groups of characters, succinctly.

Examples:

Representation Class

[a-z]	Lowercase letters. Does not include characters with accent mark and ligatures
[a-z0-9]	Lowercase letters from a to z or numbers from 0 to 9
[A-Za-z-]	Uppercase or lowercase letters, or a dash. To include the dash in a class, you must put it at the beginning or at the end so as not to confuse it with the hyphen that specifies a range of characters
[^0-9]	Any character except a digit. The caret (^) placed at the beginning of the class excludes the characters of the class (complemented class)
[[a-z--[aeiouy]]	The lowercase consonants. A class can be included in a class. The characters -- exclude what follows them
[\w--[d_]]	All letters (including foreign accented characters). Abbreviated classes can be used inside a class

Example:

```
<
[
^<>
]
+>

to

select

an

HTML
```

tag

Shorthand character classes

Representation Class

\d	A digit	(same as [0-9])

\D	Any non-numeric character	(same as [^0-9])

\w	An alphanumeric character	([a-zA-Z0-9_] including characters with accent mark and ligatures)

\W	Any “non-word” character

\s	Space, non-breaking space, tab, return line

\S	Any “non-whitespace” character

.	Any character except newline. Use the “dot all” checkbox or the (?s) regexp modifier to include the newline character.

The quantifiers

Quantifier Number of occurrences of the expression preceding the quantifier

?	0 or 1 occurrence of the expression. Same as {0,1}

+	1 or more occurrences of the expression. Same as {1,}

*	0, 1 or more occurrences of the expression. Same as {0,}

{n}	Exactly n occurrences of the expression

{min,max}	Number of occurrences between the minimum and maximum values included

{min,}	Number of occurrences between the minimum value included and the infinite

{,max}	Number of occurrences between 0 and the maximum value included

Greed

By default, with quantifiers, the regular expression engine is greedy: it extends the selection as much as possible. This often causes surprises, at first. ? follows a quantifier to make it lazy. Avoid putting two in the same expression, the result can be unpredictable.

Beware of nesting quantifiers, for example, the pattern (a*)*, as it exponentially increases processing time.

Alternation

The character in a regular expression is a logical OR |. It means that either the preceding or the following expression can match.

Exclusion

Method 1

pattern_to_exclude(*SKIP)(*FAIL)|pattern_to_select

Example:

"Blabla"(*SKIP)(*FAIL)|Blabla

selects Blabla, in the strings Blabla or "Blabla or Blabla", but not in "Blabla".

Method 2

pattern_to_exclude\K|(pattern_to_select)

"Blabla"\K|(Blabla)

selects Blabla, in the strings Blabla or "Blabla or Blabla", but not in "Blabla".

Anchors

An anchor is a way to match a logical location in a string, rather than a character. The most useful anchors for text processing are:

\b

Designates a word boundary, i.e. a transition from space to non-space character. For example,

you	can	use	\bsurd
to	match		the

surd		
but	not	absurd

.

^

Matches the start of a line (in multi-line mode, which is the default)

\$

Matches the end of a line (in multi-line mode, which is the default)

\K

Resets the start position of the selection to its position in the pattern. Some regexp engines (but not calibre) do not allow lookbehind of variable length, especially with quantifiers. When you can use \K with these engines, it also allows you to get rid of this limit by writing the equivalent of a positive lookbehind of variable length.

Groups

(expression)

Capturing group, which stores the selection and can be recalled later in the *search* or *replace* patterns with \n, where n is the sequence number of the capturing group (starting at 1 in reading order)

(?:expression)

Group that does not capture the selection

(?>expression)

Atomic Group: As soon as the expression is satisfied, the regexp engine passes, and if the rest of the pattern fails, it will not backtrack to try other combinations with the expression. Atomic groups do not capture.

(?|expression)

Branch reset group: the branches of the alternations included in the expression share the same group numbers

(?<name>expression)

Group named "name". The selection can be recalled later in the *search* pattern by (?P=name) and in the *replace* by \g<name>. Two different groups can use the same name.

Lookarounds

Lookaround Meaning

?= Positive lookahead (to be placed after the selection)

?! Negative lookahead (to be placed after the selection)

?<= Positive lookbehind (to be placed before the selection)

?<! Negative lookbehind (to be placed before the selection)

Lookaheads and lookbehinds do not consume characters, they are zero length and do not capture. They are atomic groups: as soon as the assertion is satisfied, the regexp engine passes, and if the rest of the pattern fails, it will not backtrack inside the lookahead to try other combinations.

When looking for multiple matches in a string, at the starting position of each match attempt, a lookbehind can inspect the characters before the current position. Therefore, on the string 123, the pattern (?<=\d)\d (a digit preceded by a digit) should, in theory, select 2 and 3. On the other hand, \d\K\d can only select 2, because the starting position after the first selection is immediately before 3, and there are not enough digits for a second match. Similarly, \d(\d) only captures 2. In calibre's regexp engine practice, the positive lookbehind behaves in the same way, and selects only 2, contrary to theory.

Groups can be placed inside lookarounds, but capture is rarely useful. Nevertheless, if it is useful, it will be necessary to be very careful in the use of a quantifier in a lookbehind: the greed associated with the absence of backtracking can give a surprising capture. For this reason, use \K rather than a positive lookbehind when you have a quantifier (or worse, several) in a capturing group of the positive lookbehind.

Example of negative lookahead:

```
(?![<>{}]*[>])
```

Placed at the end of the pattern prevents to select within a tag or a style embedded in the file.

Whenever possible, it is always better to “anchor” the lookarounds, to reduce the number of steps necessary to obtain the result.

Recursion

Representation Meaning

(?R)	Recursion of the entire pattern

(?1)	Recursion of the only pattern of the numbered capturing group, here group 1

Recursion is calling oneself. This is useful for balanced queries, such as quoted strings, which can contain embedded quoted strings. Thus, if during the processing of a string between double quotation marks, we encounter the beginning of a new string between double quotation marks, well we know how to do, and we call ourselves. Then we have a pattern like:

```
start-pattern(?>atomic sub-pattern|(?R))*end-pattern
```

To select a string between double quotation marks without stopping on an embedded string:

```
"((?>[^"]+|(?R))*[^"]+)"
```

This template can also be used to modify pairs of tags that can be embedded, such as <div> tags.

Special characters

Representation Character

\t	tabulation

<code>\n</code>	line break

<code>\x20</code>	(breakable) space

<code>\xa0</code>	no-break space

Meta-characters

Meta-characters are those that have a special meaning for the regexp engine. Of these, twelve must be preceded by an escape character, the backslash (`\`), to lose their special meaning and become a regular character again:

```
^ . [ ] $ ( ) * + ? | \
```

Seven other meta-characters do not need to be preceded by a backslash (but can be without any other consequence):

```
{ } ! < > = :
```

Special characters lose their status if they are used inside a class (between brackets `[]`). The closing bracket and the dash have a special status in a class. Outside the class, the dash is a simple literal, the closing bracket remains a meta-character.

The slash (`/`) and the number sign (or hash character) (`#`) are not meta-characters, they don't need to be escaped.

In some tools, like regex101.com with the Python engine, double quotes have the special status of separator, and must be escaped, or the options changed. This is not the case in the editor of calibre.

Modes

(`?s`)

Causes the dot (.) to match newline characters as well

(?m)

Makes the ^
and \$
anchors match the start and end of lines instead of the start and end of the entire string.

© Copyright Kovid Goyal. Last updated on Feb 04, 2022.

Writing your own plugins to extend calibre's functionality

calibre has a very modular design. Almost all functionality in calibre comes in the form of plugins. Plugins are used for conversion, for downloading news (though these are called recipes), for various components of the user interface, to connect to different devices, to process files when adding them to calibre and so on. You can get a complete list of all the built-in plugins in calibre by going to *Preferences → Advanced → Plugins*.

Here, we will teach you how to create your own plugins to add new features to calibre.

Contents

- [Anatomy of a calibre plugin](#)
- [A User Interface plugin](#)
 - [__init__.py](#)
 - [ui.py](#)
 - [main.py](#)
 - [Getting resources from the plugin ZIP file](#)
 - [Enabling user configuration of your plugin](#)
- [Edit book plugins](#)
 - [main.py](#)
- [Running User Interface plugins in a separate process](#)
- [Adding translations to your plugin](#)
- [The plugin API](#)
- [Debugging plugins](#)

- [More plugin examples](#)
- [Sharing your plugins with others](#)

Note

This only applies to calibre releases $\geq 0.8.60$

Anatomy of a calibre plugin

A calibre plugin is very simple, it's just a ZIP file that contains some Python code and any other resources like image files needed by the plugin. Without further ado, let's see a basic example.

Suppose you have an installation of calibre that you are using to self publish various e-documents in EPUB and MOBI formats. You would like all files generated by calibre to have their publisher set as "Hello world", here's how to do it. Create a file named `__init__.py` (this is a special name and must always be used for the main file of your plugin) and enter the following Python code into it:

```
from
calibre.customize

import

FileTypePlugin

class

HelloWorld
(
    FileTypePlugin
):

name

=

'Hello World Plugin'
```

```
# Name of the plugin

description

=

'Set the publisher to Hello World for all new conversions'

supported_platforms

=

[
'windows'
,
'osx'
,
'linux'
]

# Platforms this plugin will run on

author

=

'Acme Inc.'

# The author of this plugin

version

=

(
1
,
```

```
0
,
0
)

# The version number of this plugin

file_types

=

set
([
'epub'
,
'mobi'
])

# The file types that this plugin will be applied to

on_postprocess

=

True

# Run this plugin after conversion is complete

minimum_calibre_version

=

(
0
,
7
```

```
,  
  
53  
)  
  
def  
  
run  
(  
self  
,  
  
path_to_ebook  
):  
  
from  
  
calibre.ebooks.metadata.meta  
  
import  
  
get_metadata  
,  
  
set_metadata  
  
with  
  
open  
(  
path_to_ebook  
,  
  
'r+b'  
)  
  
as  
  
file
```

```
:  
  
ext  
  
=  
  
os  
.  
path  
.  
splitext  
(  
path_to_ebook  
)[  
-  
1  
][  
1  
:]  
.  
lower  
()  
  
mi  
  
=  
  
get_metadata  
(  
file  
,  
ext  
)  
  
mi  
.  
publisher  
  
=
```



```
'Hello World'
```

```
set_metadata  
(  
file  
,  
mi  
,  
ext  
)
```

```
return
```

```
path_to_ebook
```

That's all. To add this code to calibre as a plugin, simply run the following in the folder in which you created `__init__.py`:

```
calibre  
-  
customize  
  
-  
b  
  
.
```

Note

On macOS, the command line tools are inside the calibre bundle, for example, if you installed calibre in `/Applications`

the command line tools are in /Applications/calibre.app/Contents/MacOS/.

You can download the Hello World plugin from [helloworld_plugin.zip](https://calibre-ebook.com/downloads/helloworld_plugin.zip) [https://calibre-ebook.com/downloads/helloworld_plugin.zip].

Every time you use calibre to convert a book, the plugin's `run()` method will be called and the converted book will have its publisher set to "Hello World". This is a trivial plugin, lets move on to a more complex example that actually adds a component to the user interface.

A User Interface plugin

This plugin will be spread over a few files (to keep the code clean). It will show you how to get resources (images or data files) from the plugin ZIP file, allow users to configure your plugin, how to create elements in the calibre user interface and how to access and query the books database in calibre.

You can download this plugin from [interface_demo_plugin.zip](https://calibre-ebook.com/downloads/interface_demo_plugin.zip) [https://calibre-ebook.com/downloads/interface_demo_plugin.zip]

The first thing to note is that this ZIP file has a lot more files in it, explained below, pay particular attention to plugin-import-name-interface_demo.txt.

plugin-import-name-interface_demo.txt

An empty text file used to enable the multi-file plugin magic. This file must be present in all plugins that use more than one .py file. It should be empty and its filename must be of the form: plugin-import-name-****some_name****.txt. The presence of this file allows you to import code from the .py files present inside the ZIP file, using a statement like:

```
from
calibre_plugins.some_name.some_module
import
some_object
```

The must always prefix be present. calibre_plugins some_name

comes from the filename of the empty text file. `some_module` refers to `some_module.py` file inside the ZIP file. Note that this importing is just as powerful as regular Python imports. You can create packages and subpackages of .py modules inside the ZIP file, just like you would normally (by defining `__init__.py` in each sub-folder), and everything should “just work”.

The name you use for `some_name` enters a global namespace shared by all plugins, **so make it as unique as possible**. But remember that it must be a valid Python identifier (only alphabets, numbers and the underscore).

`__init__.py`

As before, the file that defines the plugin class

`main.py`

This file contains the actual code that does something useful

`ui.py`

This file defines the interface part of the plugin

`images/icon.png`

The icon for this plugin

`about.txt`

A text file with information about the plugin

`translations`

A folder containing .mo files with the translations of the user interface of your plugin into different languages. See below for details.

Now let’s look at the code.

`__init__.py`

First, the obligatory `__init__.py` to define the plugin metadata:

```
from  
calibre.customize
```

```
import
```

```
InterfaceActionBase
```

```
class
```

```
InterfacePluginDemo
```

```
(  
InterfaceActionBase  
):
```

```
'''
```

This class is a simple wrapper that provides information about the actual plugin class. The actual interface plugin class is called InterfacePlugin and is defined in the ui.py file, as specified in the actual_plugin field below.

The reason for having two classes is that it allows the command line calibre utilities to run without needing to load the GUI libraries.

```
'''
```

```
name
```

```
=
```

```
'Interface Plugin Demo'
```

```
description
```

```
=
```

```
'An advanced plugin demo'
```

```
supported_platforms
```

```
=
```

```
[
```

```
'windows'
```

```
,
```

```
'osx'
```

```
,
```

```
'linux'
```

```
]
```

```
author
```

```
=
```

```
'Kovid Goyal'
```

```
version
```

```
=
```

```
(
```

```
1
```

```
,
```

```
0
```

```
,
```

```
0
```

```
)
```

```
minimum_calibre_version
```

```
=
```

```
(  
0  
,  
7  
,  
53  
)
```

#: This field defines the GUI plugin class that contains all the code

#: that actually does something. Its format is module_path:class_name

#: The specified class must be defined in the specified module.

```
actual_plugin
```

```
=
```

```
'calibre_plugins.interface_demo.ui:InterfacePlugin'
```

```
def
```

```
is_customizable
```

```
(  
self  
):
```

```
'''
```

This method must return True to enable customization via

Preferences->Plugins

```
'''
```

```
return
```

```
True
```

```
def
```

```
config_widget
```

```
(
```

```
self
```

```
):
```

```
'''
```

Implement this method and :meth:`save\_settings` in your plugin to use a custom configuration dialog.

This method, if implemented, must return a QWidget. The widget can have an optional method validate() that takes no arguments and is called immediately after the user clicks OK. Changes are applied if and only if the method returns True.

If for some reason you cannot perform the configuration at this time, return a tuple of two strings (message, details), these will be displayed as a warning dialog to the user and the process will be aborted.

*The base class implementation of this method raises NotImplementedError
so by default no user configuration is possible.*

'''

It is important to put this import statement here rather than at the

top of the module as importing the config class will also cause the

GUI libraries to be loaded, which we do not want when using calibre

from the command line

from

calibre_plugins.interface_demo.config

import

ConfigWidget

return

ConfigWidget
()

def

save_settings
(
self
,

config_widget


```

):

'''
    Save the settings specified by the user with config_widget.

    :param config_widget: The widget returned by :meth:`config_widget`.
'''

config_widget
.
save_settings
()

# Apply the changes

ac
=

self
.
actual_plugin_

if
ac
is
not
None
:

```

```
ac
.  
apply_settings  
()
```

The only noteworthy feature is the field **actual_plugin**. Since calibre has both command line and GUI interfaces, GUI plugins like this one should not load any GUI libraries in `__init__.py`. The `actual_plugin` field does this for you, by telling calibre that the actual plugin is to be found in another file inside your ZIP archive, which will only be loaded in a GUI context.

Remember that for this to work, you must have a `plugin-import-name-some_name.txt` file in your plugin ZIP file, as discussed above.

Also there are a couple of methods for enabling user configuration of the plugin. These are discussed below.

[ui.py](#)

Now let's look at `ui.py` which defines the actual GUI plugin. The source code is heavily commented and should be self explanatory:

```
from  
calibre.gui2.actions  
  
import  
InterfaceAction  
  
from  
calibre_plugins.interface_demo.main  
  
import  
DemoDialog
```

```
class
```

```
InterfacePlugin
```

```
(  
InterfaceAction  
):
```

```
name
```

```
=
```

```
'Interface Plugin Demo'
```

```
# Declare the main action associated with this plugin
```

```
# The keyboard shortcut can be None if you dont want to use a keyboard
```

```
# shortcut. Remember that currently calibre has no central management for
```

```
# keyboard shortcuts, so try to use an unusual/unused shortcut.
```

```
action_spec
```

```
=
```

```
(  
'Interface Plugin Demo'
```

```
,
```

```
None
```

```
,
```

```
'Run the Interface Plugin Demo'
```

```
,
```

```
'Ctrl+Shift+F1'  
)
```

```
def
```

```
genesis  
(  
    self  
):
```

```
# This method is called once per plugin, do initial setup here
```

```
# Set the icon for this interface action
```

```
# The get_icons function is a builtin function defined for all your
```

```
# plugin code. It loads icons from the plugin zip file. It returns
```

```
# QIcon objects, if you want the actual data, use the analogous
```

```
# get_resources builtin function.
```

```
#
```

```
# Note that if you are loading more than one icon, for performance, you
```

```
# should pass a list of names to get_icons. In this case, get_icons
```

```
# will return a dictionary mapping names to QIcons. Names that
```

are not found in the zip file will result in null QIcons.

icon

=

```
get_icons  
(  
    'images/icon.png'  
)
```

The QAction is automatically created from the QActionSpec defined

above

self

.

qaction

.

setIcon

(

icon

)

self

.

qaction

.

triggered

.

connect

(

self

.

show_dialog

```
)

def
show_dialog
(
self
):

# The base plugin object defined in __init__.py

base_plugin_object

=

self
.
interface_action_base_plugin

# Show the config dialog

# The config dialog can also be shown from within

# Preferences->Plugins, which is why the do_user_config

# method is defined on the base plugin class

do_user_config

=

base_plugin_object
.
do_user_config
```

```
# self.gui is the main calibre GUI. It acts as the gateway to access
```

```
# all the elements of the calibre user interface, it should also be the
```

```
# parent of the dialog
```

```
d
```

```
=
```

```
DemoDialog
```

```
(  
self
```

```
.
```

```
gui
```

```
,
```

```
self
```

```
.
```

```
qaction
```

```
.
```

```
icon
```

```
()),
```

```
do_user_config
```

```
)
```

```
d
```

```
.
```

```
show
```

```
()
```

```
def
```

```
apply_settings
(
    self
):

    from
    calibre_plugins.interface_demo.config

    import
    prefs

    # In an actual non trivial plugin, you would probably need to

    # do something based on the settings in prefs

    prefs
```

[main.py](#)

The actual logic to implement the Interface Plugin Demo dialog.

```
from
calibre_plugins.interface_demo.config

import
prefs

class
```


DemoDialog

```
(  
QDialog  
):
```

```
def
```

```
__init__
```

```
(  
self  
,
```

```
gui  
,
```

```
icon  
,
```

```
do_user_config  
):
```

```
QDialog
```

```
.  
__init__
```

```
(  
self  
,
```

```
gui  
)
```

```
self
```

```
.  
gui
```

```
=
```

```
gui
```

```
self
.
do_user_config

=

do_user_config


# The current database shown in the GUI


# db is an instance of the class LibraryDatabase from db/legacy.py


# This class has many, many methods that allow you to do a lot of


# things. For most purposes you should use db.new_api, which has


# a much nicer interface from db/cache.py


self
.
db

=

gui
.
current_db


self
.
|

=
```

```
QVBoxLayout  
(
```

```
self
```

```
.
```

```
setLayout
```

```
(
```

```
self
```

```
.
```

```
|
```

```
)
```

```
self
```

```
.
```

```
label
```

```
=
```

```
QLabel
```

```
(
```

```
prefs
```

```
[
```

```
'hello_world_msg'
```

```
])
```

```
self
```

```
.
```

```
|
```

```
.
```

```
addWidget
```

```
(
```

```
self
```

```
.
```

```
label
```

```
)
```

```
self
.  
setTitle  
(  
    'Interface Plugin Demo'  
)
```

```
self
.  
setWindowIcon  
(  
    icon  
)
```

```
self
.  
about_button  
=  

```

```
QPushButton  
(  
    'About'  
,
```

```
self  
)
```

```
self
.  
about_button  
.  
clicked  
.  
connect  
(  
    self  
    .  
    about
```

```
)

self
.
|
.
addWidget
(
self
.
about_button
)

self
.
marked_button

=

QPushButton
(

'Show books with only one format in the calibre GUI'
,

self
)

self
.
marked_button
.
clicked
.
connect
(
self
.

```

```
marked
)

self
.
|
.
addWidget
(
self
.
marked_button
)

self
.
view_button
=
QPushButton
(

'View the most recently added book'
,

self
)

self
.
view_button
.
clicked
.
connect
(
self
```

```
.
view
)

self
.
|
.
addWidget
(
self
.
view_button
)

self
.
update_metadata_button

=

QPushButton
(

'Update metadata in a book
\'
s files'
,
self
)

self
.
update_metadata_button

.
clicked
.
```

```
connect
(
self
.
update_metadata
)

self
.
|
.
addWidget
(
self
.
update_metadata_button
)

self
.
conf_button
=
QPushButton
(
'Configure this plugin'
,
self
)

self
.
conf_button
.
clicked
```



```
.
connect
(
self
.
config
)

self
.
|
.
addWidget
(
self
.
conf_button
)

self
.
resize
(
self
.
sizeHint
())

def

about
(
self
):

# Get the about text from a file inside the plugin zip file
```

```
# The get_resources function is a builtin function defined for all your

# plugin code. It loads files from the plugin zip file. It returns

# the bytes from the specified file.

#

# Note that if you are loading more than one file, for performance, you

# should pass a list of names to get_resources. In this case,

# get_resources will return a dictionary mapping names to bytes. Names that

# are not found in the zip file will not be in the returned dictionary.

text

=

get_resources
(
'about.txt'
)

QMessageBox
.
about
(
self
,
'About the Interface Plugin Demo'
```

```
,  
  
text  
.  
decode  
(  
    'utf-8'  
)  
  
def  
  
    marked  
    (  
        self  
    ):  
  
        """ Show books with only one format """  
  
        db  
  
        =  
  
        self  
        .  
        db  
        .  
        new_api  
  
        matched_ids  
  
        =  
  
        {  
            book_id  
  
        }  
  
        for
```

```
book_id
```

```
in
```

```
db
```

```
.
```

```
all_book_ids
```

```
()
```

```
if
```

```
len
```

```
(
```

```
db
```

```
.
```

```
formats
```

```
(
```

```
book_id
```

```
))
```

```
==
```

```
1
```

```
}
```

```
# Mark the records with the matching ids
```

```
# new_api does not know anything about marked books, so we use the full
```

```
# db object
```

```
self
```

```
.
```

```
db
```

```
.
```

```
set_marked_ids
```

```
(
```

```
matched_ids
```

```
)
```

```
# Tell the GUI to search for all marked records
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
search
```

```
.
```

```
setEditText
```

```
(
```

```
'marked:true'
```

```
)
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
search
```

```
.
```

```
do_search
```

```
()
```

```
def
```

```
view
```

```
(
```

```
self
```

```
):
```

```
''' View the most recently added book '''
```

```
most_recent
```

```
=
```

```
most_recent_id
```

```
=
```

```
None
```

```
db
```

```
=
```

```
self
```

```
.
```

```
db
```

```
.
```

```
new_api
```

```
for
```

```
book_id
```

```
,
```

```
timestamp
```

```
in
```

```
db
```

```
.
```

```
all_field_for
```

```
(
```

```
'timestamp'
```

```
,
```

```
db
```

```
.
```

```
all_book_ids
```

```
())
```

```
.
```

```
items
```

```
():
```

if

most_recent

is

None

or

timestamp

>

most_recent

:

most_recent

=

timestamp

most_recent_id

=

book_id

if

most_recent_id

is

not

None

```
:  
  
# Get a reference to the View plugin  
  
view_plugin  
  
=  
  
self  
.  
gui  
.  
iactions  
[  
    'View'  
]  
  
  
# Ask the view plugin to launch the viewer for row_number  
  
view_plugin  
.  
_view_calibre_books  
([  
    most_recent_id  
])  
  
  
def  
  
update_metadata  
(  
    self  
):  
  
    """  
  
    Set the metadata in the files in the selected book's record to
```


match the current metadata in the database.

'''

from

calibre.ebooks.metadata.meta

import

set_metadata

from

calibre.gui2

import

error_dialog

,

info_dialog

Get currently selected books

rows

=

self

.

gui

.

library_view

.

selectionModel

()

```
.
selectedRows
()

if
not
rows
or
len
(
rows
)

==

0
:

return

error_dialog
(
self
.
gui
,
'Cannot update metadata'
,
'No books selected'
,
show
=
True
```

```
)  
  
# Map the rows to book ids  
  
ids  
  
=  
  
list  
(  
  map  
  (  
    self  
    .  
    gui  
    .  
    library_view  
    .  
    model  
    ()  
    .  
    id  
    ,  
    rows  
  ))  
  
db  
  
=  
  
self  
.  
db  
.  
new_api  
  
for
```

```
book_id

in

ids
:

# Get the current metadata for this book from the db

mi

=

db
.
get_metadata
(
book_id
,

get_cover
=
True
,

cover_as_data
=
True
)

fmts

=

db
.
formats
(
book_id
)
```

```
if
```

```
not
```

```
fmts
```

```
:
```

```
continue
```

```
for
```

```
fmt
```

```
in
```

```
fmts
```

```
:
```

```
fmt
```

```
=
```

```
fmt
```

```
.
```

```
lower
```

```
()
```

```
# Get a python file object for the format. This will be either
```

```
# an in memory file or a temporary on disk file
```

```
ffile
```

```
=
```

```
db
.
format
(
book_id
,

fmt
,

as_file
=
True
)
```

```
ffile
.
seek
(
0
)
```

Set metadata in the format

```
set_metadata
(
ffile
,

mi
,

fmt
)
```

```
ffile
.
seek
(
```

```
0
)

# Now replace the file in the calibre library with the updated

# file. We dont use add_format_with_hooks as the hooks were

# already run when the file was first added to calibre.

db
.
add_format
(
book_id
,

fmt
,

ffile
,

run_hooks
=
False
)

info_dialog
(
self
,

'Updated files'
,

'Updated the metadata in the files of
```

```
%d
book(s)'
%
len
(
ids
),

show
=
True
)

def

config
(
self
):

self
.
do_user_config
(
parent
=
self
)

# Apply the changes

self
.
label
.
setText
(
```



```
prefs
[
'hello_world_msg'
])
```

Getting resources from the plugin ZIP file

calibre's plugin loading system defines a couple of built-in functions that allow you to conveniently get files from the plugin ZIP file.

get_resources(name_or_list_of_names)

This function should be called with a list of paths to files inside the ZIP file. For example to access the file icon.png in the folder images in the ZIP file, you would use: images/icon.png . Always use a forward slash as the path separator, even on Windows. When you pass in a single name, the function will return the raw bytes of that file or None if the name was not found in the ZIP file. If you pass in more than one name then it returns a dict mapping the names to bytes. If a name is not found, it will not be present in the returned dict.

get_icons(name_or_list_of_names)

A convenience wrapper for get_resources() that creates QIcon objects from the raw bytes returned by get_resources. If a name is not found in the ZIP file the corresponding QIcon will be null.

Enabling user configuration of your plugin

To allow users to configure your plugin, you must define three methods in your base plugin class, **is_customizable**, **config_widget** and **save_settings** as shown below:

```
def
is_customizable
(
self
):
'''
```

This method must return True to enable customization via

Preferences->Plugins

'''

return

True

def

config_widget

(

self

):

'''

Implement this method and :meth:`save\_settings` in your plugin to use a custom configuration dialog.

This method, if implemented, must return a QWidget. The widget can have an optional method validate() that takes no arguments and is called immediately after the user clicks OK. Changes are applied if and only if the method returns True.

If for some reason you cannot perform the configuration at this time, return a tuple of two strings (message, details), these will be displayed as a warning dialog to the user and the process will be aborted.

*The base class implementation of this method raises NotImplementedError
so by default no user configuration is possible.*

'''

It is important to put this import statement here rather than at the

top of the module as importing the config class will also cause the

GUI libraries to be loaded, which we do not want when using calibre

from the command line

from

calibre_plugins.interface_demo.config

import

ConfigWidget

return

ConfigWidget

()

def

save_settings

(

self

```
,

config_widget
):

'''

    Save the settings specified by the user with config_widget.

    :param config_widget: The widget returned by :meth:`config_widget`.

'''

config_widget
.
save_settings
()

# Apply the changes

ac

=

self
.
actual_plugin_

if

ac

is

not
```

```
None
:

ac
.
apply_settings
()
```

calibre has many different ways to store configuration data (a legacy of its long history). The recommended way is to use the **JSONConfig** class, which stores your configuration information in a .json file.

The code to manage configuration data in the demo plugin is in config.py:

```
from
calibre.utils.config

import
JSONConfig

# This is where all preferences for this plugin will be stored

# Remember that this name (i.e. plugins/interface_demo) is also

# in a global namespace, so make it as unique as possible.

# You should always prefix your config file name with plugins/,

# so as to ensure you dont accidentally clobber a calibre config file

prefs
=
JSONConfig
(
```

```
'plugins/interface_demo'  
)
```

```
# Set defaults
```

```
prefs
```

```
.
```

```
defaults
```

```
[
```

```
'hello_world_msg'
```

```
]
```

```
=
```

```
'Hello, World!'
```

```
class
```

```
ConfigWidget
```

```
(
```

```
QWidget
```

```
):
```

```
def
```

```
__init__
```

```
(
```

```
self
```

```
):
```

```
QWidget
```

```
.
```

```
__init__
```

```
(
```

```
self
```

```
)
```

```
self
.
|

=

QHBoxLayout
()

self
.
setLayout
(
self
.
|
)

self
.
label

=

QLabel
(
'Hello world &message:'
)

self
.
|
.
addWidget
(
self
.
label
```

```
)

self
.
msg

=

QLineEdit
(
self
)

self
.
msg
.
setText
(
prefs
[
'hello_world_msg'
])

self
.
|
.
addWidget
(
self
.
msg
)

self
.
label
```



```
.
setBuddy
(
self
.
msg
)

def

save_settings
(
self
):

prefs
[
'hello_world_msg'
]

=

self
.
msg
.
text
()
```

The
object is now available throughout the plugin code by a simple:

prefs

```
from

calibre_plugins.interface_demo.config

import
```

prefs

You can see the prefs object being used in main.py:

```
def
config
(
self
):

self
.
do_user_config
(
parent
=
self
)

# Apply the changes

self
.
label
.
setText
(
prefs
[
'hello_world_msg'
])
```

Edit book plugins

Now let's change gears for a bit and look at creating a plugin to add tools to the calibre book editor. The plugin is available here: [editor demo plugin.zip](https://calibre-ebook.com/downloads/editor_demo_plugin.zip) [https://calibre-ebook.com/downloads/editor_demo_plugin.zip].

The first step, as for all plugins is to create the import name empty txt file, as described [above](#). We shall name the file plugin-import-name-editor_plugin_demo.txt.

Now we create the mandatory `__init__.py` file that contains metadata about the plugin – its name, author, version, etc.

```
from
calibre.customize

import
EditBookToolPlugin

class

DemoPlugin
(
EditBookToolPlugin
):

name

=

'Edit Book plugin demo'

version

=
```

```
(
1
,
0
,
0
)

author

=

'Kovid Goyal'

supported_platforms

=

[
'windows'
,
'osx'
,
'linux'
]

description

=

'A demonstration of the plugin interface for the ebook editor'

minimum_calibre_version

=
```

```
(
1
,
46
,
0
)
```

A single editor plugin can provide multiple tools each tool corresponds to a single button in the toolbar and entry in the *Plugins* menu in the editor. These can have sub-menus in case the tool has multiple related actions.

The tools must all be defined in the file `main.py` in your plugin. Every tool is a class that inherits from the [calibre.gui2.tweak_book.plugin.Tool](#) class. Let's look at `main.py` from the demo plugin, the source code is heavily commented and should be self-explanatory. Read the API documents of the [calibre.gui2.tweak_book.plugin.Tool](#) class for more details.

[main.py](#)

Here we will see the definition of a single tool that will multiply all font sizes in the book by a number provided by the user. This tool demonstrates various important concepts that you will need in developing your own plugins, so you should read the (heavily commented) source code carefully.

```
import
re
from
qt.core
import
QAction
,
```

QInputDialog

from

[css_parser.css](#)

import

CSSRule

The base class that all tools must inherit from

from

[calibre.gui2.tweak_book.plugin](#)

import

Tool

from

[calibre](#)

import

force_unicode

from

[calibre.gui2](#)

import

error_dialog

from

[calibre.ebooks.oeb.polish.container](#)

```
import
```

```
OEB_DOCS
```

```
,
```

```
OEB_STYLES
```

```
,
```

```
serialize
```

```
class
```

```
DemoTool
```

```
(
```

```
Tool
```

```
):
```

```
#: Set this to a unique name it will be used as a key
```

```
name
```

```
=
```

```
'demo-tool'
```

```
#: If True the user can choose to place this tool in the plugins toolbar
```

```
allowed_in_toolbar
```

```
=
```

```
True
```

```
#: If True the user can choose to place this tool in the plugins menu
```

```
allowed_in_menu
```

```
=
```

```
True
```

```
def
```

```
create_action
```

```
(
```

```
self
```

```
,
```

```
for_toolbar
```

```
=
```

```
True
```

```
):
```

```
# Create an action, this will be added to the plugins toolbar and
```

```
# the plugins menu
```

```
ac
```

```
=
```

```
QAction
```

```
(
```

```
get_icons
```

```
(
```

```
'images/icon.png'
```

```
),
```

```
'Magnify fonts'
```

```
,
```



```
self
.
gui
)

# noqa

if
not
for_toolbar
:

# Register a keyboard shortcut for this toolbar action. We only

# register it for the action created for the menu, not the toolbar,

# to avoid a double trigger

self
.
register_shortcut
(
ac
,
'magnify-fonts-tool'
,
default_keys
=
(
'Ctrl+Shift+Alt+D'
,))
```

```
ac
.
triggered
.
connect
(
self
.
ask_user
)

return

ac

def

ask_user
(
self
):

# Ask the user for a factor by which to multiply all font sizes

factor
,

ok

=

QInputDialog
.
getDouble
(

self
```

```
.
gui
,

'Enter a magnification factor'
,

'Allow font sizes in the book will be multiplied by the specified factor'
,

value
=
2
,

min
=
0.1
,

max
=
4

)

if

ok
:

# Ensure any in progress editing the user is doing is present in the container

self
.
boss
.
commit_all_editors_to_container
```

```
()

try
:

self
.
magnify_fonts
(
factor
)

except

Exception
:

# Something bad happened report the error to the user

import

traceback

error_dialog
(
self
.
gui
,

—
(
'Failed to magnify fonts'
),

—
(
```

```
'Failed to magnify fonts, click "Show details" for more info'  
,
```

```
det_msg  
=  
traceback  
.  
format_exc  
(),
```

```
show  
=  
True  
)
```

```
# Revert to the saved restore point
```

```
self  
.  
boss  
.  
revert_requested  
(  
self  
.  
boss  
.  
global_undo  
.  
previous_container  
)
```

```
else  
:
```

```
# Show the user what changes we have made, allowing her to
```

```
# revert them if necessary
```

```
self
```

```
.
```

```
boss
```

```
.
```

```
show_current_diff
```

```
()
```

```
# Update the editor UI to take into account all the changes we
```

```
# have made
```

```
self
```

```
.
```

```
boss
```

```
.
```

```
apply_container_update_to_gui
```

```
()
```

```
def
```

```
magnify_fonts
```

```
(
```

```
self
```

```
,
```

```
factor
```

```
):
```

```
# Magnify all font sizes defined in the book by the specified factor
```

```
# First we create a restore point so that the user can undo all changes
```

```
# we make.
```

```
self
```

```
.
```

```
boss
```

```
.
```

```
add_savepoint
```

```
(
```

```
'Before: Magnify fonts'
```

```
)
```

```
container
```

```
=
```

```
self
```

```
.
```

```
current_container
```

```
# The book being edited as a container object
```

```
# Iterate over all style declarations in the book, this means css
```

```
# stylesheets, <style> tags and style="" attributes
```

```
for
```

```
name
```

```
,
```

```
media_type
```

```
in
```

```
container
.
mime_map
.
items
():

if

media_type

in

OEB_STYLES
:

# A stylesheet. Parsed stylesheets are css_parser CSSStyleSheet

# objects.

self
.
magnify_stylesheet
(
container
.
parsed
(
name
),
factor
)

container
.
dirty
(
```



```
name
)

# Tell the container that we have changed the stylesheet

elif

media_type

in

OEB_DOCS
:

# A HTML file. Parsed HTML files are lxml elements

for

style_tag

in

container
.
parsed
(
name
)
.
xpath
(
'//*[local-name="style"]'
):

if

style_tag
.
```

```
text
```

```
and
```

```
style_tag
```

```
.
```

```
get
```

```
(
```

```
'type'
```

```
,
```

```
None
```

```
)
```

```
in
```

```
{
```

```
None
```

```
,
```

```
'text/css'
```

```
};
```

```
# We have an inline CSS <style> tag, parse it into a
```

```
# stylesheet object
```

```
sheet
```

```
=
```

```
container
```

```
.
```

```
parse_css
```

```
(
```

```
style_tag
```

```
.
```

```
text
```

```
)
```

```
self
.
magnify_stylesheet
(
sheet
,

factor
)

style_tag
.
text

=

serialize
(
sheet
,

'text/css'
,

pretty_print
=
True
)

container
.
dirty
(
name
)

# Tell the container that we have changed the stylesheet

for
```

```
elem
```

```
in
```

```
container
```

```
.
```

```
parsed
```

```
(
```

```
name
```

```
)
```

```
.
```

```
xpath
```

```
(
```

```
'//*[@style]'
```

```
):
```

```
# Process inline style attributes
```

```
block
```

```
=
```

```
container
```

```
.
```

```
parse_css
```

```
(
```

```
elem
```

```
.
```

```
get
```

```
(
```

```
'style'
```

```
),
```

```
is_declaration
```

```
=
```

```
True
```

```
)
```

```
self
```

```
.  
magnify_declaration  
(  
    block  
,  
  
    factor  
)
```

```
elem
```

```
.  
set  
(  
    'style'  
,  
  
    force_unicode  
(  
        block  
        .  
        getCssText  
(  
            separator  
            =  
            ' '  
        ),  
  
        'utf-8'  
    ))
```

```
def
```

```
magnify_stylesheet  
(  
    self  
,  
  
    sheet  
,
```

```
factor
):

# Magnify all fonts in the specified stylesheet by the specified

# factor.

for
rule
in
sheet
.
cssRules
.
rulesOfType
(
    CSSRule
    .
    STYLE_RULE
):

self
.
magnify_declaration
(
    rule
    .
    style
    ,
    factor
)

def
```

```
magnify_declaration
```

```
(  
  self
```

```
,
```

```
  style
```

```
,
```

```
  factor
```

```
):
```

```
# Magnify all fonts in the specified style declaration by the specified
```

```
# factor
```

```
val
```

```
=
```

```
style
```

```
.
```

```
getPropertyValue
```

```
(
```

```
  'font-size'
```

```
)
```

```
if
```

```
not
```

```
val
```

```
:
```

```
return
```

```
# see if the font-size contains a number
```

```
num
```

```
=
```

```
re
```

```
.
```

```
search
```

```
(
```

```
r
```

```
'[0-9.]+'
```

```
,
```

```
val
```

```
)
```

```
if
```

```
num
```

```
is
```

```
not
```

```
None
```

```
:
```

```
num
```

```
=
```

```
num
```

```
.
```

```
group
```

```
()
```

```
val
```

```
=
```



```
val
.
replace
(
  num
  ,
  '
%f
'

%

(
  float
  (
    num
  )

  *

factor
))

style
.
setProperty
(
  'font-size'
  ,
  val
)
)
```

We should also be dealing with the font shorthand property and

font sizes specified as non numbers, but those are left as exercises

for the reader

Let's break down `main.py`. We see that it defines a single tool, named *Magnify fonts*. This tool will ask the user for a number and multiply all font sizes in the book by that number.

The first important thing is the tool name which you must set to some relatively unique string as it will be used as the key for this tool.

The next important entry point is the `calibre.gui2.tweak_book.plugin.Tool.create_action()`. This method creates the QAction objects that appear in the plugins toolbar and plugin menu. It also, optionally, assigns a keyboard shortcut that the user can customize. The triggered signal from the QAction is connected to the `ask_user()` method that asks the user for the font size multiplier, and then runs the magnification code.

The magnification code is well commented and fairly simple. The main things to note are that you get a reference to the editor window as `self.gui` and the editor *Boss* as `self.boss`. The *Boss* is the object that controls the editor user interface. It has many useful methods, that are documented in the `calibre.gui2.tweak_book.boss.Boss` class.

Finally, there is `self.current_container` which is a reference to the book being edited as a `calibre.ebooks.oeb.polish.container.Container` object. This represents the book as a collection of its constituent HTML/CSS/image files and has convenience methods for doing many useful things. The container object and various useful utility functions that can be reused in your plugin code are documented in [API documentation for the e-book editing tools](#).

Running User Interface plugins in a separate process

If you are writing a user interface plugin that needs to make use of Qt WebEngine, it cannot be run in the main calibre process as it is not possible to use WebEngine there. Instead you can copy the data your plugin needs to a temporary folder and run the plugin with that data in a separate process. A simple example plugin follows that shows how to do this.

You can download the plugin from [webengine_demo_plugin.zip](https://calibre-ebook.com/downloads/webengine_demo_plugin.zip) [https://calibre-ebook.com/downloads/webengine_demo_plugin.zip].

The important part of the plugin is in two functions:

```
def

show_dialog
(
self
):

# Ask the user for a URL

url
,

ok

=

QInputDialog
.
getText
(
self
.
gui
,

'Enter a URL'
,

'Enter a URL to browse below'
,

text
=
'https://calibre-ebook.com'
)

if
```

not

ok

or

not

url

:

return

Launch a separate process to view the URL in WebEngine

self

.

gui

.

job_manager

.

launch_gui_app

(

'webengine-dialog'

,

kwargs

=

{

'module'

:

'calibre_plugins.webengine_demo.main'

,

'url'

:

url

})

```
def
```

```
main
```

```
(  
url  
):
```

```
# This function is run in a separate process and can do anything it likes,
```

```
# including use QWebEngine. Here it simply opens the passed in URL
```

```
# in a QWebEngineView
```

```
app
```

```
=
```

```
Application  
([])
```

```
w
```

```
=
```

```
QWebEngineView  
()
```

```
w
```

```
.
```

```
setUrl
```

```
(  
QUrl  
(  
url
```

```

))

w
.
show
()

w
.
raise_
()

app
.
exec
()

```

The `show_demo()` function asks the user for a URL and then runs the `main()` function passing it that URL. The `main()` function displays the URL in a `QWebEngineView`.

[Adding translations to your plugin](#)

You can have all the user interface strings in your plugin translated and displayed in whatever language is set for the main calibre user interface.

The first step is to go through your plugin's source code and mark all user visible strings as translatable, by surrounding them in `_()`. For example:

```

action_spec

=

(
-
(
'My plugin'

```

```

),

None
,

-
(
'My plugin is cool'
),

None
)

```

Then use some program to generate .po files from your plugin source code. There should be one .po file for every language you want to translate into. For example: de.po for German, fr.po for French and so on. You can use the [Poedit](https://poedit.net/) [https://poedit.net/] program for this.

Send these .po files to your translators. Once you get them back, compile them into .mo files. You can again use Poedit for that, or just do:

```

calibre
-
debug

-
c

"from calibre.translations.msgfmt import main; main()"

filename
.
po

```

Put the .mo files into the translations folder in your plugin.

The last step is to simply call the function `load_translations()` at the top of your plugin's .py files. For performance reasons you should only call this function in those .py files that actually have translatable strings. So in a typical User Interface plugin you would call it at the top of ui.py

but

not

__init__.py

.

You can test the translations of your plugins by changing the user interface language in calibre under *Preferences* → *Interface* → *Look & feel* or by running calibre like this:

```
CALIBRE_OVERRIDE_LANG
=
de
calibre
```

Replace

de

with the language code of the language you want to test.

[The plugin API](#)

As you may have noticed above, a plugin in calibre is a class. There are different classes for the different types of plugins in calibre. Details on each class, including the base class of all plugins can be found in [API documentation for plugins](#).

Your plugin is almost certainly going to use code from calibre. To learn how to find various bits of functionality in the calibre code base, read the section on the calibre [Code layout](#).

[Debugging plugins](#)

The first, most important step is to run calibre in debug mode. You can do this from the command line with:

```
calibre
-
debug
-
g
```

Or from within calibre by right-clicking the *Preferences* button or using the Ctrl+Shift+R keyboard shortcut.

When running from the command line, debug output will be printed to the console, when running from within calibre the output will go to a txt file.

You can insert print statements anywhere in your plugin code, they will be output in debug mode. Remember, this is Python, you really shouldn't need anything more than print statements to debug ;) I developed all of calibre using just this debugging technique.

You can quickly test changes to your plugin by using the following command line:

```
calibre
-
debug

-
s
;

calibre
-
customize

-
b

/
path
/
to
/
your
/
plugin
/
folder
;

calibre
```

This will shutdown a running calibre, wait for the shutdown to complete, then update your plugin in calibre and relaunch calibre.

[More plugin examples](#)

You can find a list of many sophisticated calibre plugins [here](https://www.mobileread.com/forums/showthread.php?t=118764) [https://www.mobileread.com/forums/showthread.php?t=118764].

Sharing your plugins with others

If you would like to share the plugins you have created with other users of calibre, post your plugin in a new thread in the [calibre plugins forum](https://www.mobileread.com/forums/forumdisplay.php?f=237) [https://www.mobileread.com/forums/forumdisplay.php?f=237].

Typesetting mathematics in e-books

The calibre E-book viewer has the ability to display mathematics embedded in e-books (EPUB and HTML files). You can typeset the mathematics directly with TeX or MathML or AsciiMath. The calibre E-book viewer uses the excellent [MathJax](https://www.mathjax.org) library to do this. This is a brief tutorial on creating e-books with mathematics in them that work well with the calibre E-book viewer.

A simple HTML file with mathematics

You can write mathematics inline inside a simple HTML file and the calibre E-book viewer will render it into properly typeset mathematics. In the example below, we use TeX notation for mathematics. You will see that you can use normal TeX commands, with the small caveat that ampersands and less than and greater than signs have to be written as `&`, `<` and `>`; respectively.

The first step is to tell calibre that this will contains mathematics. You do this by adding the following snippet of code to the `<head>` section of the HTML file:

```
<
script
type
=
"text/x-mathjax-config"
></
script
>
```

That's it, now you can type mathematics just as you would in a .tex file. For example, here are Lorentz's equations:

```
<
h2
>
The
```

Lorenz

Equations

</

h2

>

<

p

>

\

begin

{

align

}

\

dot

{

x

}

&

amp

;

=

\

sigma

(

y

-

x

)

\\

\

dot

{

y

}

&
amp
;

=
\
rho

x

-

y

-

xz
\\
\
dot
{
z
}

&
amp
;

=

-

\
beta

z

+

xy

\
end

```
{
align
}

</
p
>
```

This snippet looks like the following screen shot in the calibre E-book viewer.

$$\begin{aligned}\dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - y - xz \\ \dot{z} &= -\beta z + xy\end{aligned}$$

The Lorenz Equations

The complete HTML file, with more equations and inline mathematics is reproduced below. You can convert this HTML file to EPUB in calibre to end up with an e-book you can distribute easily to other people.

```
<!DOCTYPE html>

<
html
>

<!-- Copyright (c) 2012 Design Science, Inc. -->

<
head
>

<
title
>
Math Test Page
</
title
>

<
```

meta

http-equiv

=

"content-type"

content

=

"text/html; charset=UTF-8"

/>

<!-- This script tag is needed to make calibre's ebook-viewer recognize that this file needs math typesetting -->

<

script

type

=

"text/x-mathjax-config"

>

// This line adds numbers to all equations automatically, unless explicitly suppressed.

MathJax

.

tex

=

{

tags

:

'all'

};

</

script

```
>

<
style
>

h1

{
text-align
:
center
}

h2

{

font-weight
:

bold
;

background-color
:

#DDDDDD
;

padding
:

.2
em

.5
em
;
```


margin-top

:

1.5

em

;

border-top

:

3

px

solid

#666666

;

border-bottom

:

2

px

solid

#999999

;

}

</

style

>

</

head

>

```
<
body
>
```

```
<
h1
>
Sample Equations
</
h1
>
```

```
<
h2
>
The Lorenz Equations
</
h2
>
```

```
<
p
>
```

```
\begin{align}
\dot{x}
&= \sigma(y-x) \label{lorenz} \\
\dot{y}
&= \rho x - y - xz \\
\dot{z}
&= -\beta z + xy
\end{align}
</
p
>
```

<

h2

>

The Cauchy-Schwarz Inequality

</

h2

>

<

p

>

\[

$\left(\sum_{k=1}^n a_k b_k \right)^2 \leq$

$\left(\sum_{k=1}^n a_k^2 \right) \left(\sum_{k=1}^n b_k^2 \right)$

\]

</

p

>

<

h2

>

A Cross Product Formula

</

h2

>

<

p

>

\[

$\mathbf{V}_1 \times \mathbf{V}_2 =$

$\begin{vmatrix}$

$\mathbf{i}$

&

$\mathbf{j}$

&

$\mathbf{k} \begin{vmatrix}$

$\frac{\partial X}{\partial u}$

&

$\frac{\partial Y}{\partial u}$

&

0 \

$\frac{\partial X}{\partial v}$

&

$\frac{\partial Y}{\partial v}$

&

0 \

$\end{vmatrix}$

\]

</

p

>

<

h2

>

The probability of getting (k) heads when flipping (n) coins is:

</

h2

>

<

p

>

$[P(E) = \{n \text{ choose } k\} p^k (1-p)^{n-k}]$

</

p

>

<

h2

>

An Identity of Ramanujan

</

h2

>

<

p
 >

$$\frac{1}{(\sqrt{\phi \sqrt{5}} - \phi) e^{\frac{25}{\pi}} + \frac{e^{-2\pi}}{1 + \frac{e^{-4\pi}}{1 + \frac{e^{-6\pi}}{1 + \frac{e^{-8\pi}}{1 + \dots}}}}}$$
 </p>

p
 >
 <
h2
 >
 A Rogers-Ramanujan Identity
 </p>

h2
 >
 <
p
 >

$$1 + \frac{q^2}{(1-q)} + \frac{q^6}{(1-q)(1-q^2)} + \dots = \prod_{j=0}^{\infty} \frac{1}{(1-q^{5j+2})(1-q^{5j+3})},$$

$$\quad \quad \quad \text{for } |q|$$

<
 1\$}.

</p>

p
 >
 <
h2
 >
 Maxwell's Equations
 </p>

h2
 >

<

p

>

\begin{align}

\nabla \times \vec{\mathbf{B}} - \frac{1}{c} \frac{\partial \vec{\mathbf{E}}}{\partial t}

&

= \frac{4\pi}{c} \vec{\mathbf{j}} \nabla \cdot \vec{\mathbf{E}}

&

&

= 4\pi \rho \nabla \times \vec{\mathbf{E}} + \frac{1}{c} \frac{\partial \vec{\mathbf{B}}}{\partial t}

&

= \vec{\mathbf{0}} \nabla \cdot \vec{\mathbf{B}}

&

&

= 0

\end{align}

</

p

>

<

h2

>

In-line Mathematics

</

h2

>

<

p

>

While display equations look good for a page of samples, the ability to mix math and text in a paragraph is also important. This expression $\sqrt{3x-1}+(1+x)^2$ is an example of an inline equation. As you see, equations can be used this way as well, without unduly disturbing the spacing between lines.

</

```
p
>

<
h2
>
References to equations
</
h2
>

<
p
>
Here is a reference to the Lorenz Equations ( $\ref{lorenz}$ ). Clicking on the equation number will
take you back to the equation.
</
p
>

</
body
>

</
html
>
```

More information

Since the calibre E-book viewer uses the MathJax library to render mathematics, the best place to find out more about mathematics in e-books and get help is the [MathJax website](https://www.mathjax.org) [https://www.mathjax.org].

Creating AZW3 • EPUB • MOBI catalogs

calibre's Create catalog feature enables you to create a catalog of your library in a variety of formats. This help file describes cataloging options when generating a catalog in AZW3, EPUB and MOBI formats.

- [Selecting books to catalog](#)
- [Included sections](#)
- [Prefixes](#)
- [Excluded books](#)
- [Excluded genres](#)
- [Other options](#)
- [Custom catalog covers](#)
- [Additional help resources](#)

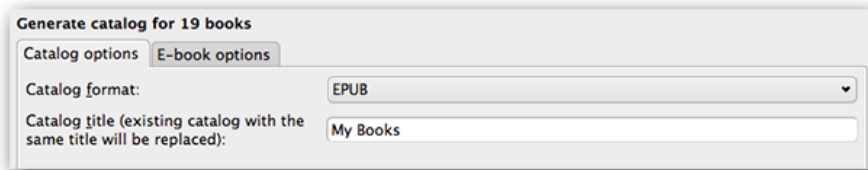
[Selecting books to catalog](#)

If you want *all* of your library cataloged, remove any search or filtering criteria in the main window. With a single book selected, all books in your library will be candidates for inclusion in the generated catalog. Individual books may be excluded by various criteria; see the [Excluded genres](#) section below for more information.

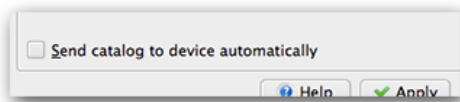
If you want only *some* of your library cataloged, you have two options:

- Create a multiple selection of the books you want cataloged. With more than one book selected in calibre's main window, only the selected books will be cataloged.
- Use the Search field or the Tag browser to filter the displayed books. Only the displayed books will be cataloged.

To begin catalog generation, select the menu item *Convert books > Create a catalog of the books in your calibre library*. You may also add a *Create catalog* button to a toolbar in *Preferences > Interface > Toolbars & menus* for easier access to the Generate catalog dialog.

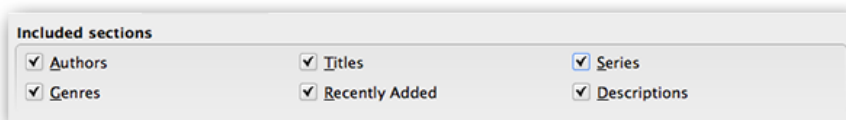


In *Catalog options*, select **AZW3**, **EPUB** or **MOBI** as the Catalog format. In the *Catalog title* field, provide a name that will be used for the generated catalog. If a catalog of the same name and format already exists, it will be replaced with the newly-generated catalog.



Enabling *Send catalog to device automatically* will download the generated catalog to a connected device upon completion.

Included sections



Sections enabled by a checkmark will be included in the generated catalog:

- *Authors* - all books, sorted by author, presented in a list format. Non-series books are listed before series books.
- *Titles* - all books, sorted by title, presented in a list format.
- *Series* - all books that are part of a series, sorted by series, presented in a list format.
- *Genres* - individual genres presented in a list, sorted by Author and Series.
- *Recently Added* - all books, sorted in reverse chronological order. List includes books added in the last 30 days, then a month-by-month listing of added books.
- *Descriptions* - detailed description page for each book, including a cover thumbnail and comments. Sorted by author, with non-series books listed before series books.

Prefixes

	Name	Prefix	Field	Value
1	☒ Read book	✓	Last Read	any date
2	☒ Wishlist item	×	Tags	Wishlist
3	☒ Library books	↔	Available in Library	True

Prefix rules allow you to add a prefix to book listings when certain criteria are met. For example, you might want to mark books you've read with a checkmark, or books on your wishlist with an X.

The checkbox in the first column enables the rule. *Name* is a rule name that you provide. *Field* is either *Tags* or a custom column from your library. *Value* is the content of *Field* to match. When a prefix rule is satisfied, the book will be marked with the selected *Prefix*.

Three prefix rules have been specified in the example above:

1. *Read book* specifies that a book with any date in a custom column named *Last read* will be prefixed with a checkmark symbol.
2. *Wishlist item* specifies that any book with a *Wishlist* tag will be prefixed with an X symbol.
3. *Library books* specifies that any book with a value of True (or Yes) in a custom column *Available in Library* will be prefixed with a double arrow symbol.

The first matching rule supplies the prefix. Disabled or incomplete rules are ignored.

Excluded books

	Name	Field	Value
1	☒ Catalogs	Tags	Catalog
2	☒ Archived Books	Status	Archived

Exclusion rules allow you to specify books that will not be cataloged.

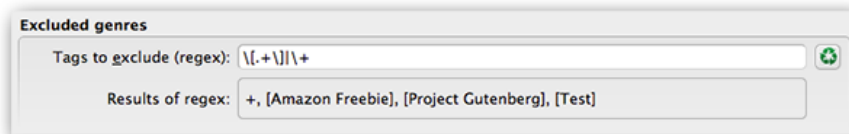
The checkbox in the first column enables the rule. *Name* is a rule name that you provide. *Field* is either *Tags* or a custom column in your library. *Value* is the content of *Field* to match. When an exclusion rule is satisfied, the book will be excluded from the generated catalog.

Two exclusion rules have been specified in the example above:

1. The *Catalogs* rule specifies that any book with a *Catalog* tag will be excluded from the generated catalog.
2. The *Archived Books* rule specifies that any book with a value of *Archived* in the custom column *Status* will be excluded from the generated catalog.

All rules are evaluated for every book. Disabled or incomplete rules are ignored.

Excluded genres



When the catalog is generated, tags in your database are used as genres. For example, you may use the tags Fiction and Nonfiction. These tags become genres in the generated catalog, with books listed under their respective genre lists based on their assigned tags. A book will be listed in every genre section for which it has a corresponding tag.

You may be using certain tags for other purposes, perhaps a + to indicate a read book, or a bracketed tag like [Amazon

Freebie]

to indicate a book's source. The *Excluded genres* regex allows you to specify tags that you don't want used as genres in the generated catalog. The default exclusion regex pattern `[.+\\]\\+`

excludes any tags of the form [tag], as well as excluding +, the default tag for read books, from being used as genres in the generated catalog.

You can also use an exact tag name in a regex. For example, [Amazon

Freebie]

or [Project

Gutenberg]

. If you want to list multiple exact tags for exclusion, put a pipe (vertical bar) character between them: [Amazon

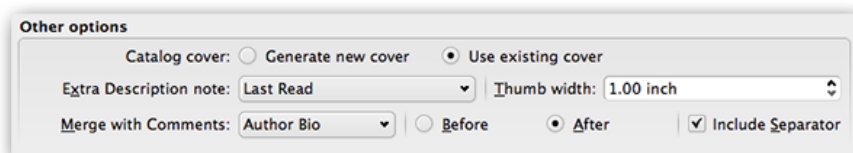
Freebie]|[Project

Gutenberg]

.

Results of regex shows you which tags will be excluded when the catalog is built, based on the tags in your database and the regex pattern you enter. The results are updated as you modify the regex pattern.

Other options



Catalog cover specifies whether to generate a new cover or use an existing cover. It is possible to create a custom cover for your catalogs - see [Custom catalog covers](#) for more information. If you have created a custom cover that you want to reuse, select *Use existing cover*. Otherwise, select *Generate new cover*.

Extra Description note specifies a custom column's contents to be inserted into the Description page, next to the cover thumbnail. For example, you might want to display the date you last read a book using a *Last Read* custom column. For advanced use of the Description note feature, see [this post in the calibre forum](https://www.mobileread.com/forums/showpost.php?p=1335767&postcount=395) [https://www.mobileread.com/forums/showpost.php?p=1335767&postcount=395].

Thumb width specifies a width preference for cover thumbnails included with Descriptions pages. Thumbnails are cached to improve performance. To experiment with different widths, try generating a catalog with just a few books until you've determined your preferred width, then generate your full catalog. The first time a catalog is generated with a new thumbnail width, performance will be slower, but subsequent builds of the catalog will take advantage of the thumbnail cache.

Merge with comments specifies a custom column whose content will be non-destructively merged with the comments metadata during catalog generation. For example, you might have a custom column *Author bio* that you'd like to append to the comments metadata. You can choose to insert the custom column contents *before or after* the comments section, and optionally separate the appended content with a horizontal rule separator. Eligible custom column types include text,

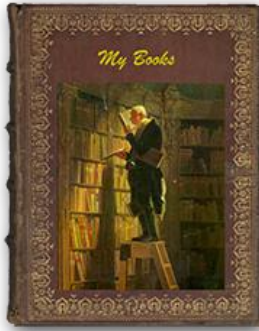
comments,

and

composite

.

Custom catalog covers



With the [Generate Cover plugin](https://www.mobileread.com/forums/showthread.php?t=124219) [https://www.mobileread.com/forums/showthread.php?t=124219] installed, you can create custom covers for your catalog. To install the plugin, go to *Preferences > Advanced > Plugins > Get new plugins*.

Additional help resources

For more information on calibre's Catalog feature, see the MobileRead forum sticky [Creating Catalogs - Start here](https://www.mobileread.com/forums/showthread.php?t=118556) [https://www.mobileread.com/forums/showthread.php?t=118556], where you can find information on how to customize the catalog templates, and how to submit a bug report.

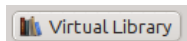
To ask questions or discuss calibre's Catalog feature with other users, visit the MobileRead forum [Library Management](https://www.mobileread.com/forums/forumdisplay.php?f=236) [https://www.mobileread.com/forums/forumdisplay.php?f=236].

Virtual libraries

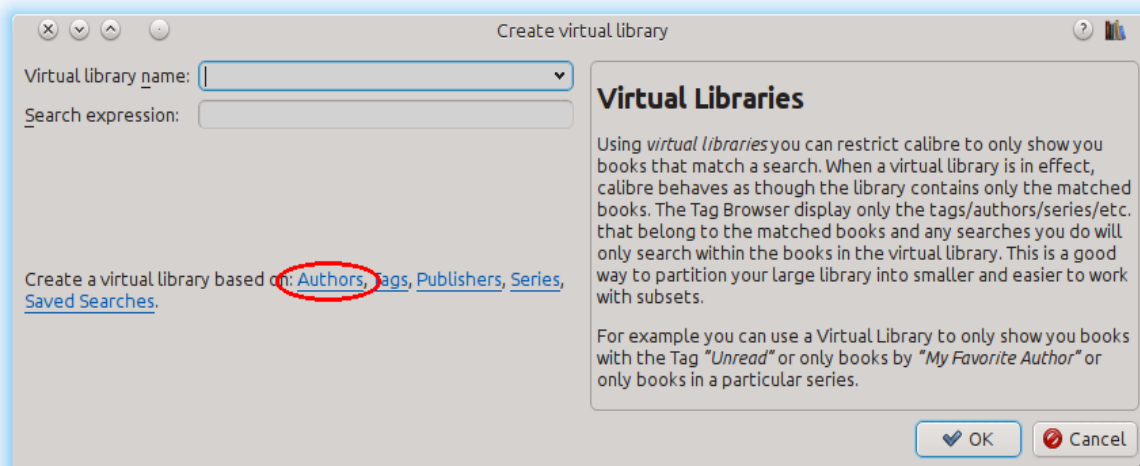
In calibre, a Virtual library is a way to tell calibre to open only a subset of a normal library. For example, you might want to only work with books by a certain author, or books having only a certain tag. Using Virtual libraries is the preferred way of partitioning your large book collection into smaller sub collections. It is superior to splitting up your library into multiple smaller libraries as, when you want to search through your entire collection, you can simply go back to the full library. There is no way to search through multiple separate libraries simultaneously in calibre.

A Virtual library is different from a simple search. A search will only restrict the list of books shown in the book list. A Virtual library does that, and in addition it also restricts the entries shown in the *Tag browser* to the left. The Tag browser will only show tags, authors, series, publishers, etc. that come from the books in the Virtual library. A Virtual library thus behaves as though the actual library contains only the restricted set of books.

Creating Virtual libraries



Virtual library click the *Virtual library* button located to the left of the Search bar and select the *Create Virtual library* option. As a first example, let's create a Virtual library that shows us only the books by a particular author. Click the *Authors* link as shown in the image below and choose the author you want to use and click OK.



The Create Virtual library dialog has been filled in for you. Click OK and you will see that a new Virtual library has been created, and automatically switched to, that displays only the books by the selected author. As far as calibre is concerned, it is as if your library contains only the books by the selected author.

You can switch back to the full library at any time by once again clicking the *Virtual library* and selecting the entry named *<None>*.

Virtual libraries are based on *searches*. You can use any search as the basis of a Virtual library. The Virtual library will contain only the books matched by that search. First, type in the search you want to use in the Search bar or build a search using the *Tag browser*. When you are happy with the returned results, click the *Virtual library* button, choose *Create library* and enter a name for the new Virtual library. The Virtual library will then be created based on the search you just typed in. Searches are very powerful, for examples of the kinds of things you can do with them, see [The search interface](#).

Examples of useful Virtual libraries

- Books added to calibre in the last day::

date:>1daysago

- Books added to calibre in the last month::

date:>30daysago

- Books with a rating of 5 stars::

rating:5

- Books with a rating of at least 4 stars::

rating:>=4

- Books with no rating::

rating:false

- Periodicals downloaded by the Fetch News function in calibre::

tags:=News and author:=calibre

- Books with no tags::

tags:false

- Books with no covers::

cover:false

Working with Virtual libraries

You can edit a previously created Virtual library or remove it, by clicking the *Virtual library* and choosing the appropriate action.

You can tell calibre that you always want to apply a particular Virtual library when the current library is opened, by going to *Preferences → Interface → Behavior*.

You can quickly use the current search as a temporary Virtual library by clicking the *Virtual library* button and choosing the **current search* entry.

You can display all available Virtual libraries as tabs above the book list. This is particularly handy if you like switching between Virtual libraries very often. Click the *Virtual library* button and select *Show Virtual libraries as tabs*. You can re-arrange the tabs by drag and drop and close ones you do not want to see. Closed tabs can be restored by right-clicking on the tab bar.

Using Virtual libraries in searches

You can search for books that are in a Virtual library using the `vl:` prefix. For example, `vl:Read`

will find all the books in the *Read* Virtual library. The search `vl:Read`

and

`vl:"Science`

`Fiction"`

will find all the books that are in both the *Read* and *Science Fiction* Virtual libraries.

The value following `vl:` must be the name of a Virtual library. If the Virtual library name contains spaces then surround it with quotes.

One use for a Virtual library search is in the Content server. In *Preferences → Sharing over the net → Require username and password* you can limit the calibre libraries visible to a user. For each visible library you can specify a search expression to further limit which books are seen. Use `vl:"Virtual`

library

`name"`

to limit the books to those in a Virtual library.

Using additional restrictions

You can further restrict the books shown in a Virtual library by using *Additional restrictions*. An additional restriction is saved search you previously created that can be applied to the current Virtual library to further restrict the books shown in a Virtual library. For example, say you have a Virtual library for books tagged as *Historical Fiction* and a saved search that shows you unread books, you can click the *Virtual Library* button and choose the *Additional restriction* option to show only unread Historical Fiction books. To learn about saved searches, see [Saving searches](#).

The calibre:// URL scheme

calibre registers itself as the handler program for calibre:// URLs. So you can use these to perform actions like opening books, searching for books, etc from other programs/documents or via the command line. For example, running the following at the command line:

```
calibre  
  
calibre  
:  
//  
switch  
-  
library  
/  
Some_Library
```

Will open calibre with the library named Some

Library

. Library names are the folder name of the library folder with spaces replaced by underscores. The special value _ means the current library. The various types of URLs are documented below.

You can even place these links inside HTML files or Word documents or similar and the operating system will automatically run calibre to perform the specified action.

- [Switch to a specific library](#)
- [Show a specific book in calibre](#)
- [Open a specific book in the E-book viewer at a specific position](#)
- [Searching for books](#)
- [Hex encoding of URL parameters](#)

[Switch to a specific library](#)

The URL syntax is:

```
calibre
:
//
switch
-
library
/
Library_Name
```

Library names are the folder name of the library with spaces replaced by underscores. The special value `-` means the current library. You can also use [hex encoding](#) for the library names, useful if the library names have special characters that would otherwise require URL encoding. Hex encoded library names look like:

```
_hex_
-
AD23F4BC
```

Where the part after the `_hex_` prefix is the library name encoded as UTF-8 and every byte represented by two hexadecimal characters.

[Show a specific book in calibre](#)

The URL syntax is:

```
calibre
:
//
show
-
book
/
```

```
Library_Name  
/  
book_id
```

This will show the book with book_id (a number) in calibre. The ids for books can be seen in the calibre interface by hovering over the *Click to open* link in the *Book details* panel, it is the number in brackets at the end of the path to the book folder.

You can copy a link to the current book displayed in calibre by right clicking the *Book details* panel and choosing *Copy link to book*.

[Open a specific book in the E-book viewer at a specific position](#)

The URL syntax is:

```
calibre://view-book/Library_Name/book_id/book_format?open_at=location
```

Here, book_format is the format of the book, for example, EPUB or MOBI and the location is an optional location inside the book. The easiest way to get these links is to open a book in the viewer, then in the viewer controls select *Go to* → *Location* and there such a link will be given that you can copy/paste elsewhere.

[Searching for books](#)

The URL syntax is:

```
calibre://search/Library_Name?q=query  
calibre://search/Library_Name?eq=hex_encoded_query
```

Here query is any valid [search expression](#). If the search expression is complicated, [encode it as a hex string](#) and use eq instead. Leaving out the query will cause the current search to be cleared.

By default, if a Virtual library is selected, calibre will clear it before doing the search to ensure all books are found. If you want to preserve the Virtual library, use:

```
calibre://search/Library_Name?q=query&virtual_library=_
```

If you want to switch to a particular Virtual library, use:

```
calibre://search/Library_Name?virtual_library=Library%20Name
```

or

```
calibre://search/Library_Name?encoded_virtual_library=hex_encoded_virtual_library_name
```

replacing spaces in the Virtual library name by %20.

If you perform a search in calibre and want to generate a link for it you can do so by right clicking the search bar and choosing *Copy search as URL*.

Hex encoding of URL parameters

Hex encoding of URL parameters is done by first encoding the parameter as UTF-8 bytes, and then replacing each byte by two hexadecimal characters representing the byte. For example, the string abc is the bytes 0x61

0x62

and

0x63

in UTF-8, so the encoded version is the string: 616263.

Customizing calibre

calibre has a highly modular design. Various parts of it can be customized. Here, you will learn:

- how to use environment variables and *tweaks* to customize calibre's behavior,
- how to specify your own static resources like icons and templates to override the defaults
- how to use *plugins* to add functionality to calibre.
- how to share icon themes and plugins with other calibre users.
- to see how to create *recipes* to add new sources of online content to calibre visit the Section [Adding your favorite news website](#).

Note

Note that although icon themes and plugins are indexed and downloadable via calibre's builtin updater, they are not part of calibre, and their canonical locations for support and source code are on the [Mobileread forums](https://www.mobileread.com/forums/forumdisplay.php?f=166) [https://www.mobileread.com/forums/forumdisplay.php?f=166] in their support threads.

- [Environment variables](#)
- [Tweaks](#)
- [Overriding icons, templates, et cetera](#)
- [Creating your own icon theme for calibre](#)
- [Customizing calibre with plugins](#)

Environment variables

- CALIBRE_CONFIG_DIRECTORY
- sets the folder where configuration files are stored/read.

- CALIBRE_TEMP_DIR
- sets the temporary folder used by calibre
- CALIBRE_CACHE_DIRECTORY
- sets the folder calibre uses to cache persistent data between sessions
- CALIBRE_OVERRIDE_DATABASE_PATH
- allows you to specify the full path to metadata.db. Using this variable you can have metadata.db be in a location other than the library folder. Useful if your library folder is on a networked drive that does not support file locking.
- CALIBRE_DEVELOP_FROM
- used to run from a calibre development environment. See [Setting up a calibre development environment](#).
- CALIBRE_OVERRIDE_LANG
- used to force the language used by the interface (ISO 639 language code)
- CALIBRE_TEST_TRANSLATION
- used to test a translation .po file (should be the path to the .po file)
- CALIBRE_NO_NATIVE_FILEDIALOGS
- causes calibre to not use native file dialogs for selecting files/folders.
- CALIBRE_NO_NATIVE_MENUBAR
- causes calibre to not create a native (global) menu on Ubuntu Unity and similar linux desktop environments. The menu is instead placed inside the window, as is traditional.
- CALIBRE_USE_SYSTEM_THEME
- by default, on Linux, calibre uses its own builtin Qt style. This is to avoid crashes and hangs caused by incompatibilities between the version of Qt calibre is built against and the system Qt. The downside is that calibre may not follow the system look and feel. If you set this environment variable on Linux, it will cause calibre to use the system theme – beware of crashes and hangs.
- CALIBRE_SHOW_DEPRECATION_WARNINGS
- causes calibre to print deprecation warnings to stdout. Useful for calibre developers.
- CALIBRE_NO_DEFAULT_PROGRAMS
- prevent calibre from automatically registering the filetypes it is capable of handling with Windows.
- CALIBRE_USE_DARK_PALETTE
- set it to 1

to have calibre use dark colors and 0 for normal colors (ignored on macOS). On Windows 10 in the absence of this variable, the Windows system preference for dark colors is used.

- `SYSFS_PATH`
 - Use if sysfs is mounted somewhere other than `/sys`
- `http_proxy`
 - used on Linux to specify an HTTP(S) proxy

`https_proxy`

See [How to set environment variables in Windows](https://www.computerhope.com/issues/ch000549.htm) [https://www.computerhope.com/issues/ch000549.htm]. If you are on macOS you can set environment variables by creating the `~/Library/Preferences/calibre/macos-env.txt` and putting the environment variables one per line in it, for example:

```
CALIBRE_DEVELOP_FROM=$HOME/calibre-src/src
CALIBRE_NO_NATIVE_FILEDIALOGS=1
CALIBRE_CONFIG_DIRECTORY=~/.config/calibre
```

Tweaks

Tweaks are small changes that you can specify to control various aspects of calibre's behavior. You can change them by going to Preferences->Advanced->Tweaks. The default values for the tweaks are reproduced below

```
#!/usr/bin/env python

# vim:fileencoding=UTF-8:ts=4:sw=4:sta:et:sts=4:ai

# License: GPLv3 Copyright: 2010, Kovid Goyal <kovid at kovidgoyal.net>

# Contains various tweaks that affect calibre behavior. Only edit this file if
# you know what you are doing. If you delete this file, it will be recreated from
# defaults.

#: Auto increment series index
```

```
# The algorithm used to assign a book added to an existing series a series number.

# New series numbers assigned using this tweak are always integer values, except

# if a constant non-integer is specified.

# Possible values are:

# next - First available integer larger than the largest existing number

# first_free - First available integer larger than 0

# next_free - First available integer larger than the smallest existing number

# last_free - First available integer smaller than the largest existing number. Return largest
existing + 1 if no free number is found

# const - Assign the number 1 always

# no_change - Do not change the series index

# a number - Assign that number always. The number is not in quotes. Note that 0.0 can be
used here.

# Examples:

# series_index_auto_increment = 'next'

# series_index_auto_increment = 'next_free'

# series_index_auto_increment = 16.5

#

# Set the use_series_auto_increment_tweak_when_importing tweak to True to

# use the above values when importing/adding books. If this tweak is set to

# False (the default) then the series number will be set to 1 if it is not

# explicitly set during the import. If set to True, then the

# series index will be set according to the series_index_auto_increment setting.
```

Note that the use_series_auto_increment_tweak_when_importing tweak is used

only when a value is not provided during import. If the importing regular

expression produces a value for series_index, or if you are reading metadata

from books and the import plugin produces a value, than that value will

be used irrespective of the setting of the tweak.

series_index_auto_increment

=

'next'

use_series_auto_increment_tweak_when_importing

=

False

#: Add separator after completing an author name

Should the completion separator be append

to the end of the completed text to

automatically begin a new completion operation

for authors.

Can be either True or False

authors_completer_append_separator

=

False

```
#: Author sort name algorithm

# The algorithm used to copy author to author_sort.

# Possible values are:

# invert: use "fn ln" -> "ln, fn"

# copy : copy author to author_sort without modification

# comma : use 'copy' if there is a ',' in the name, otherwise use 'invert'

# nocomma : "fn ln" -> "ln fn" (without the comma)

# When this tweak is changed, the author_sort values stored with each author

# must be recomputed by right-clicking on an author in the left-hand tags

# panel, selecting 'manage authors', and pressing

# 'Recalculate all author sort values'.

#

# The author_name_suffixes are words that are ignored when they occur at the

# end of an author name. The case of the suffix is ignored and trailing

# periods are automatically handled.

#

# The same is true for author_name_prefixes.

#

# The author_name_copywords are a set of words which, if they occur in an

# author name, cause the automatically generated author sort string to be

# identical to the author name. This means that the sort for a string like

# "Acme Inc." will be "Acme Inc." instead of "Inc., Acme".
```

```
#
```

```
# If author_use_surname_prefixes is enabled, any of the words in
```

```
# author_surname_prefixes will be treated as a prefix to the surname, if they
```

```
# occur before the surname. So for example, "John von Neumann" would be sorted
```

```
# as "von Neumann, John" and not "Neumann, John von".
```

```
author_sort_copy_method
```

```
=
```

```
'comma'
```

```
author_name_suffixes
```

```
=
```

```
(
```

```
'Jr'
```

```
,
```

```
'Sr'
```

```
,
```

```
'Inc'
```

```
,
```

```
'Ph.D'
```

```
,
```

```
'Phd'
```

```
,
```

```
'MD'
```

```
,
```

```
'M.D'
```

```
,
```

'I'

,

'II'

,

'III'

,

'IV'

,

'Junior'

,

'Senior'

)

author_name_prefixes

=

(

'Mr'

,

'Mrs'

,

'Ms'

,

'Dr'

,

'Prof'

)

author_name_copywords

=

```
(  
'Agency'  
,  
'Corporation'  
,  
'Company'  
,  
'Co.'  
,  
'Council'  
,  
'Committee'  
,  
'Inc.'  
,  
'Institute'  
,  
'National'  
,  
'Society'  
,  
'Club'  
,  
'Team'  
)
```

author_use_surname_prefixes

=

False

author_surname_prefixes

=

```
(  
'da'  
,  
'de'  
,  
'di'  
,  
'la'  
,  
'le'  
,  
'van'  
,  
'von'  
)
```

#: Splitting multiple author names

By default, calibre splits a string containing multiple author names on

ampersands and the words "and" and "with". You can customize the splitting

by changing the regular expression below. Strings are split on whatever the

specified regular expression matches, in addition to ampersands.

Default: r'(?i),?\s+(and|with)\s+'

authors_split_regex

=

r

'(?i),?\s+(and|with)\s+'

#: Use author sort in Tag browser

Set which author field to display in the Tag browser (the list of authors,

series, publishers etc on the left hand side). The choices are author and

author_sort. This tweak affects only what is displayed under the authors

category in the Tag browser and Content server. Please note that if you set this

to author_sort, it is very possible to see duplicate names in the list because

although it is guaranteed that author names are unique, there is no such

guarantee for author_sort values. Showing duplicates won't break anything, but

it could lead to some confusion. When using 'author_sort', the tooltip will

show the author's name.

Examples:

categories_use_field_for_author_name = 'author'

categories_use_field_for_author_name = 'author_sort'

`categories_use_field_for_author_name`

=

`'author'`

#: Control partitioning of Tag browser

When partitioning the Tag browser, the format of the subcategory label is

*# controlled by a template: categories_collapsed_name_template if sorting by
name, categories_collapsed_rating_template if sorting by average rating, and
categories_collapsed_popularity_template if sorting by popularity. There are
two variables available to the template: first and last. The variable 'first'
is the initial item in the subcategory, and the variable 'last' is the final
item in the subcategory. Both variables are 'objects'; they each have multiple
values that are obtained by using a suffix. For example, first.name for an
author category will be the name of the author. The sub-values available are:
name: the printable name of the item
count: the number of books that references this item
avg_rating: the average rating of all the books referencing this item
sort: the sort value. For authors, this is the author_sort for that author
category: the category (e.g., authors, series) that the item is in.
Note that the "r" in front of the { is necessary if there are backslashes
(\ characters) in the template. It doesn't hurt anything to leave it there
even if there aren't any backslashes.*

categories_collapsed_name_template

=

r

'{first.sort:shorten(4,,0)} - {last.sort:shorten(4,,0)}'

categories_collapsed_rating_template

=

```
r
'{first.avg_rating:4.2f:ifempty(0)} - {last.avg_rating:4.2f:ifempty(0)}'
```

categories_collapsed_popularity_template

=

```
r
'
{first.count:d}
-
{last.count:d}
'
```

#: Control order of categories in the Tag browser

Change the following dict to change the order that categories are displayed in

the Tag browser. Items are named using their lookup name, and will be sorted

using the number supplied. The lookup name '' stands for all names that*

otherwise do not appear. Two names with the same value will be sorted

using the default order, the one specified by tag_browser_category_default_sort.

Example:

```
# tag_browser_category_order = {'series':1, 'tags':2, '*':3}
```

#

results in the order series, tags, then everything else in default order.

The tweak tag_browser_category_default_sort specifies the sort order before

applying the category order from the dict. The allowed values are:

```
# tag_browser_category_default_sort = 'default' # The calibre default order
```

```
# tag_browser_category_default_sort = 'display_name' # Sort by the display name of the
```

category

tag_browser_category_default_sort = 'lookup_name' # Sort by the lookup name of the category

#

In addition and if the category default sort is not 'default' you can specify

whether the sort is ascending or descending. This is ignored if the sort is 'default'.

tag_browser_category_default_sort_direction = 'ascending'

tag_browser_category_default_sort_direction = 'descending'

tag_browser_category_order

=

```
{  
  '*':  
  :  
  1  
}
```

tag_browser_category_default_sort

=

'default'

tag_browser_category_default_sort_direction

=

'ascending'

#: Specify columns to sort the booklist by on startup

Provide a set of columns to be sorted on when calibre starts.

The argument is None if saved sort history is to be used

otherwise it is a list of column,order pairs. Column is the

lookup/search name, found using the tooltip for the column

Order is 0 for ascending, 1 for descending.

For example, set it to [('authors',0),('title',0)] to sort by

title within authors.

sort_columns_at_startup

=

None

#: Control how dates are displayed

Format to be used for publication date and the timestamp (date).

A string controlling how the publication date is displayed in the GUI

d the day as number without a leading zero (1 to 31)

dd the day as number with a leading zero (01 to 31)

ddd the abbreviated localized day name (e.g. 'Mon' to 'Sun').

dddd the long localized day name (e.g. 'Monday' to 'Sunday').

M the month as number without a leading zero (1-12)

MM the month as number with a leading zero (01-12)

MMM the abbreviated localized month name (e.g. 'Jan' to 'Dec').

MMMM the long localized month name (e.g. 'January' to 'December').

yy the year as two digit number (00-99)

```
# yyyy the year as four digit number

# h the hours without a leading 0 (0 to 11 or 0 to 23, depending on am/pm) '

# hh the hours with a leading 0 (00 to 11 or 00 to 23, depending on am/pm) '

# m the minutes without a leading 0 (0 to 59) '

# mm the minutes with a leading 0 (00 to 59) '

# s the seconds without a leading 0 (0 to 59) '

# ss the seconds with a leading 0 (00 to 59) '

# ap use a 12-hour clock instead of a 24-hour clock, with "ap" replaced by the localized string
for am or pm

# AP use a 12-hour clock instead of a 24-hour clock, with "AP" replaced by the localized string
for AM or PM

# iso the date with time and timezone. Must be the only format present

# For example, given the date of 9 Jan 2010, the following formats show

# MMM yyyy ==> Jan 2010 yyyy ==> 2010 dd MMM yyyy ==> 09 Jan 2010

# MM/yyyy ==> 01/2010 d/M/yy ==> 9/1/10 yy ==> 10

#

# publication default if not set: MMM yyyy

# timestamp default if not set: dd MMM yyyy

# last_modified_display_format if not set: dd MMM yyyy

gui_pubdate_display_format

=

'MMM yyyy'

gui_timestamp_display_format
```

=

'dd MMM yyyy'

gui_last_modified_display_format

=

'dd MMM yyyy'

#: Control sorting of titles and series in the library display

Control title and series sorting in the library view. If set to

'library_order', the title sort field will be used instead of the title.

Unless you have manually edited the title sort field, leading articles such as

The and A will be ignored. If set to 'strictly_alphabetic', the titles will be

sorted as-is (sort by title instead of title sort). For example, with

library_order, The Client will sort under 'C'. With strictly_alphabetic, the

book will sort under 'T'.

This flag affects calibre's library display. It has no effect on devices. In

addition, titles for books added before changing the flag will retain their

order until the title is edited. Editing a title and hitting Enter

without changing anything is sufficient to change the sort. Or you can use

the 'Update title sort' action in the Bulk metadata edit dialog to update

it for many books at once.

title_series_sorting

=

'library_order'

#: Control formatting of title and series when used in templates

Control how title and series names are formatted when saving to disk/sending

to device. The behavior depends on the field being processed. If processing

title, then if this tweak is set to 'library_order', the title will be

replaced with title_sort. If it is set to 'strictly_alphabetic', then the

title will not be changed. If processing series, then if set to

'library_order', articles such as 'The' and 'An' will be moved to the end. If

set to 'strictly_alphabetic', the series will be sent without change.

For example, if the tweak is set to library_order, "The Lord of the Rings"

will become "Lord of the Rings, The". If the tweak is set to

strictly_alphabetic, it would remain "The Lord of the Rings". Note that the

formatter function raw_field will return the base value for title and

series regardless of the setting of this tweak.

save_template_title_series_sorting

=

'library_order'

#: Set the list of words considered to be "articles" for sort strings

Set the list of words that are to be considered 'articles' when computing the

title sort strings. The articles differ by language. By default, calibre uses

*# a combination of articles from English and whatever language the calibre user
interface is set to. In addition, in some contexts where the book language is
available, the language of the book is used. You can change the list of
articles for a given language or add a new language by editing
per_language_title_sort_articles. To tell calibre to use a language other
than the user interface language, set, default_language_for_title_sort. For
example, to use German, set it to 'deu'. A value of None means the user
interface language is used. The setting title_sort_articles is ignored
(present only for legacy reasons).*

per_language_title_sort_articles

=

{

English

'eng'

:

(

r

'A\s+'

,

r

'The\s+'

,

r

'An\s+'

),

Esperanto

'epo'

:

(

r

'La\s+'

,

r

"L'"

,

'L''

),

Spanish

'spa'

:

(

r

'El\s+'

,

r

'La\s+'

,

r

'Lo\s+'

,

r

'Los\s+'

,

r

'Las\s+'

,

r

'Un\s+'

,

r

'Una\s+'

,

r

'Unos\s+'

,

r

'Unas\s+'

),

French

'fra'

:

(

r

'Le\s+'

,

r

'La\s+'

,

r

"L'"

,

u
'L'

,

u
'L''

,

r
'Les\s+'

,

r
'Un\s+'

,

r
'Une\s+'

,

r
'Des\s+'

,

r
'De\s+La\s+'

,

r
'De\s+'

,

r
"D'"

,

u
'D'

,

u
'L'
,

Polish

'pol'
:
(),

Italian

'ita'
:

(
'Lo
\\
s+'
,

'Il
\\
s+'
,

"L"
,

'L'
,

'La
\\
s+'
,

'Gli

\\
s+'
,

'I
\\
s+'
,

'Le
\\
s+'
,

'Uno
\\
s+'
,

'Un
\\
s+'
,

'Una
\\
s+'
,

"Un"
,

'Un'
,

'Dei
\\
s+'
,

'Degli

\\
s+'
,

'Delle

\\
s+'
,

'Del

\\
s+'
,

'Della

\\
s+'
,

'Dello

\\
s+'
,

"Dell'"

,

'Dell''

),

Portuguese

'por'

:

(

r

'A\s+'
,

```
r
'O\s+'
,
```

```
r
'Os\s+'
,
```

```
r
'As\s+'
,
```

```
r
'Um\s+'
,
```

```
r
'Uns\s+'
,
```

```
r
'Uma\s+'
,
```

```
r
'Umas\s+'
,
```

```
),
```

```
# Romanian
```

```
'ron'
```

```
:
```

```
(
r
'Un\s+'
,
```


,

r

'O\s+'

,

r

'Nişte\s+'

,

),

German

'deu'

:

(

r

'Der\s+'

,

r

'Die\s+'

,

r

'Das\s+'

,

r

'Den\s+'

,

r

'Ein\s+'

,

r

```
'Eine\s+'  
,  
  
r  
'Einen\s+'  
,  
  
r  
'Dem\s+'  
,  
  
r  
'Des\s+'  
,  
  
r  
'Einem\s+'  
,  
  
r  
'Eines\s+'  
)
```

Dutch

```
'nld'  
  
:  
  
(  
r  
'De\s+'  
,  
  
r  
'Het\s+'  
,  
  
r  
'Een\s+'  
)
```

,

r
""n\s+""

,

r
""s\s+""

,

r
'Ene\s+'

,

r
'Ener\s+'

,

r
'Enes\s+'

,

r
'Den\s+'

,

r
'Der\s+'

,

r
'Des\s+'

,

r
""t\s+""

),

Swedish

'swe'

:

(

r

'En\s+'

,

r

'Ett\s+'

,

r

'Det\s+'

,

r

'Den\s+'

,

r

'De\s+'

,

),

Turkish

'tur'

:

(

r

'Bir\s+'

,

),

```
# Afrikaans
```

```
'afr'
```

```
:
```

```
(
```

```
r
```

```
"n\s+"
```

```
,
```

```
r
```

```
'Die\s+'
```

```
,
```

```
),
```

```
# Greek
```

```
'ell'
```

```
:
```

```
(
```

```
r
```

```
'O\s+'
```

```
,
```

```
r
```

```
'I\s+'
```

```
,
```

```
r
```

```
'To\s+'
```

```
,
```

```
r
```

```
'Ta\s+'
```

```
,
```

```
r
'Tus\s+'
,

r
'Tis\s+'
,

r
'''Enas\s+'''
,

r
'''Mia\s+'''
,

r
'''Ena\s+'''
,

r
'''Enan\s+'''
,

),
```

Hungarian

'hun'

:

```
(
r
'A\s+'
,
```

```
r
'Az\s+'
```

```
,  
  
r  
'Egy\s+'  
,),  
  
}
```

default_language_for_title_sort

=

None

title_sort_articles

=

```
r  
'^(A|The|An)\s+'
```

#: Specify a folder calibre should connect to at startup

Specify a folder that calibre should connect to at startup using

connect_to_folder. This must be a full path to the folder. If the folder does

not exist when calibre starts, it is ignored.

Example for Windows:

auto_connect_to_folder = 'C:/Users/someone/Desktop/testlib'

Example for other operating systems:

auto_connect_to_folder = '/home/dropbox/My Dropbox/someone/library'

auto_connect_to_folder

=

''

#: Specify renaming rules for SONY collections

*# Specify renaming rules for SONY collections. This tweak is only applicable if
metadata management is set to automatic. Collections on SONYs are named
depending upon whether the field is standard or custom. A collection derived
from a standard field is named for the value in that field.*

#

*# For example, if the standard 'series' column contains the value 'Darkover', then the
collection name is 'Darkover'. A collection derived from a custom field will
have the name of the field added to the value. For example, if a custom series
column named 'My Series' contains the name 'Darkover', then the collection
will by default be named 'Darkover (My Series)'. For purposes of this
documentation, 'Darkover' is called the value and 'My Series' is called the
category. If two books have fields that generate the same collection name,
then both books will be in that collection.*

#

*# This set of tweaks lets you specify for a standard or custom field how
the collections are to be named. You can use it to add a description to a
standard field, for example 'Foo (Tag)' instead of the 'Foo'. You can also use
it to force multiple fields to end up in the same collection.*

#

*# For example, you could force the values in 'series', '#my_series_1', and
'#my_series_2' to appear in collections named 'some_value (Series)', thereby*


```
# merging all of the fields into one set of collections.

#

# There are two related tweaks. The first determines the category name to use

# for a metadata field. The second is a template, used to determines how the

# value and category are combined to create the collection name.

# The syntax of the first tweak, sony_collection_renaming_rules, is:

# {'field_lookup_name':'category_name_to_use', 'lookup_name':'name', ...}

#

# The second tweak, sony_collection_name_template, is a template. It uses the

# same template language as plugboards and save templates. This tweak controls

# how the value and category are combined together to make the collection name.

# The only two fields available are {category} and {value}. The {value} field is

# never empty. The {category} field can be empty. The default is to put the

# value first, then the category enclosed in parentheses, it isn't empty:

# '{value} {category:|(|)}'

#

# Examples: The first three examples assume that the second tweak

# has not been changed.

#

# 1) I want three series columns to be merged into one set of collections. The

# column lookup names are 'series', '#series_1' and '#series_2'. I want nothing
```

in the parenthesis. The value to use in the tweak value would be:

```
# sony_collection_renaming_rules={'series':", '#series_1':", '#series_2':"}
```

#

2) I want the word '(Series)' to appear on collections made from series, and

the word '(Tag)' to appear on collections made from tags. Use:

```
# sony_collection_renaming_rules={'series':'Series', 'tags':'Tag'}
```

#

3) I want 'series' and '#myseries' to be merged, and for the collection name

to have '(Series)' appended. The renaming rule is:

```
# sony_collection_renaming_rules={'series':'Series', '#myseries':'Series'}
```

#

4) Same as example 2, but instead of having the category name in parentheses

and appended to the value, I want it prepended and separated by a colon, such

as in Series: Darkover. I must change the template used to format the category name

#

The resulting two tweaks are:

```
# sony_collection_renaming_rules={'series':'Series', 'tags':'Tag'}
```

```
# sony_collection_name_template='{category:| |: }{{value}}'
```

```
sony_collection_renaming_rules
```

```
=
```

```
{}
```

```
sony_collection_name_template
```

```
=
```

```
'
```

```
{value}  
{category:| (|)}
```

#: Specify how SONY collections are sorted

Specify how SONY collections are sorted. This tweak is only applicable if

metadata management is set to automatic. You can indicate which metadata is to

be used to sort on a collection-by-collection basis. The format of the tweak

is a list of metadata fields from which collections are made, followed by the

name of the metadata field containing the sort value.

Example: The following indicates that collections built from pubdate and tags

are to be sorted by the value in the custom column '#mydate', that collections

built from 'series' are to be sorted by 'series_index', and that all other

collections are to be sorted by title. If a collection metadata field is not

named, then if it is a series- based collection it is sorted by series order,

otherwise it is sorted by title order.

[(['pubdate', 'tags'], '#mydate'), (['series'], 'series_index'), ([''], 'title')]*

Note that the bracketing and parentheses are required. The syntax is

[([list of fields], sort field) , ([list of fields] , sort field)]

Default: empty (no rules), so no collection attributes are named.

sony_collection_sorting_rules

=

[]

#: Control how tags are applied when copying books to another library

Set this to True to ensure that tags in 'Tags to add when adding

a book' are added when copying books to another library

`add_new_book_tags_when_importing_books`

`=`

False

#: Set the maximum number of sort 'levels'

Set the maximum number of sort 'levels' that calibre will use to resort the

library after certain operations such as searches or device insertion. Each

sort level adds a performance penalty. If the database is large (thousands of

books) the penalty might be noticeable. If you are not concerned about multi-

level sorts, and if you are seeing a slowdown, reduce the value of this tweak.

`maximum_resort_levels`

`=`

`5`

#: Choose whether dates are sorted using visible fields

Date values contain both a date and a time. When sorted, all the fields are

used, regardless of what is displayed. Set this tweak to True to use only

the fields that are being displayed.

`sort_dates_using_visible_fields`

`=`

False

#: Fuzz value for trimming covers

The value used for the fuzz distance when trimming a cover.

Colors within this distance are considered equal.

The distance is in absolute intensity units.

cover_trim_fuzz_value

=

10

#: Control behavior of the book list

You can control the behavior of double clicks and pressing Enter on the books

list. Choices: open_viewer, do_nothing, show_book_details, edit_cell,

edit_metadata. Selecting anything other than open_viewer or show_book_details

has the side effect of disabling editing a field using a single click.

Default: open_viewer.

Example: doubleclick_on_library_view = 'do_nothing'

You can also control whether the book list scrolls per item or

per pixel. Default is per item.

doubleclick_on_library_view

=

'open_viewer'

enter_key_behavior

=

'do_nothing'

horizontal_scrolling_per_column

=

False

vertical_scrolling_per_row

=

False

#: Language to use when sorting

Setting this tweak will force sorting to use the

collating order for the specified language. This might be useful if you run

calibre in English but want sorting to work in the language where you live.

Set the tweak to the desired ISO 639-1 language code, in lower case.

You can find the list of supported locales at

https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes

Default: locale_for_sorting = '' -- use the language calibre displays in

Example: locale_for_sorting = 'fr' -- sort using French rules.

Example: locale_for_sorting = 'nb' -- sort using Norwegian rules.

locale_for_sorting

=

11

#: Number of columns for custom metadata in the edit metadata dialog

Set whether to use one or two columns for custom metadata when editing

metadata one book at a time. If True, then the fields are laid out using two

columns. If False, one column is used.

metadata_single_use_2_cols_for_custom_fields

=

True

#: Edit metadata custom column label width and elision point

Set the width of custom column labels shown in the edit metadata dialogs.

If metadata_edit_elide_labels is True then labels wider than the width

will be elided, otherwise they will be word wrapped. The maximum width is

computed by multiplying the average width of characters in the font by the

appropriate number.

Set the elision point to 'middle' to put the ellipsis (...) in the middle of

the label, 'right' to put it at the right end of the label, and 'left' to

put it at the left end.

metadata_edit_elide_labels

=

True

metadata_edit_bulk_cc_label_length

=

25

metadata_edit_single_cc_label_length

=

12

metadata_edit_elision_point

=

'right'

#: The number of seconds to wait before sending emails

The number of seconds to wait before sending emails when using a

public email server like GMX/Hotmail/Gmail. Default is: 5 minutes

Setting it to lower may cause the server's SPAM controls to kick in,

making email sending fail. Changes will take effect only after a restart of

calibre. You can also change the list of hosts that calibre considers

to be public relays here. Any relay host ending with one of the suffixes

in the list below will be considered a public email server.

public_smtp_relay_delay

=

301

public_smtp_relay_host_suffixes

=


```
[  
'gmail.com'  
,  
'live.com'  
,  
'gmx.com'  
,  
'outlook.com'  
]
```

#: The maximum width and height for covers saved in the calibre library

All covers in the calibre library will be resized, preserving aspect ratio,

to fit within this size. This is to prevent slowdowns caused by extremely

large covers

maximum_cover_size

=

```
(  
1650  
,  
2200  
)
```

#: Where to send downloaded news

When automatically sending downloaded news to a connected device, calibre

will by default send it to the main memory. By changing this tweak, you can

control where it is sent. Valid values are "main", "carda", "cardb". Note

*# that if there isn't enough free space available on the location you choose,
the files will be sent to the location with the most free space.*

send_news_to_device_location

=

"main"

#: Unified toolbar on macOS

If you enable this option and restart calibre, the toolbar will be 'unified'

with the titlebar as is normal for macOS applications. However, doing this has

various bugs, for instance the minimum width of the toolbar becomes twice

what it should be and it causes other random bugs on some systems, so turn it

on at your own risk!

unified_title_toolbar_on_osx

=

False

#: Save original file when converting/polishing from same format to same format

When calibre does a conversion from the same format to the same format, for

example, from EPUB to EPUB, the original file is saved, so that in case the

conversion is poor, you can tweak the settings and run it again. By setting

this to False you can prevent calibre from saving the original file.

Similarly, by setting save_original_format_when_polishing to False you can

prevent calibre from saving the original file when polishing.

save_original_format

=

True

save_original_format_when_polishing

=

True

#: Number of recently viewed books to show

Right-clicking the "View" button shows a list of recently viewed books. Control

how many should be shown, here.

gui_view_history_size

=

15

#: Change the font size of the Book details panel in the interface

Change the font size at which book details are rendered in the side panel and

comments are rendered in the metadata edit dialog. Set it to a positive or

negative number to increase or decrease the font size.

change_book_details_font_size_by

=

0

#: What format to default to when using the "Unpack book" feature

The "Unpack book" feature of calibre allows direct editing of a book format.

If multiple formats are available, calibre will offer you a choice

of formats, defaulting to your preferred output format if it is available.

Set this tweak to a specific value of 'EPUB' or 'AZW3' to always default

to that format rather than your output format preference.

Set to a value of 'remember' to use whichever format you chose last time you

used the "Unpack book" feature.

Examples:

default_tweak_format = None (Use output format)

default_tweak_format = 'EPUB'

default_tweak_format = 'remember'

default_tweak_format

=

None

#: Do not preselect a completion when editing authors/tags/series/etc.

This means that you can make changes and press Enter and your changes will

not be overwritten by a matching completion. However, if you wish to use the

completions you will now have to press Tab to select one before pressing

Enter. Which technique you prefer will depend on the state of metadata in

your library and your personal editing style.

preselect_first_completion

=

False

#: Completion mode when editing authors/tags/series/etc.

By default, when completing items, calibre will show you all the candidates

that start with the text you have already typed. You can instead have it show

all candidates that contain the text you have already typed. To do this, set

completion_mode to 'contains'. For example, if you type asi it will match both

Asimov and Quasimodo, whereas the default behavior would match only Asimov.

completion_mode

=

'prefix'

#: Sort the list of libraries alphabetically

The list of libraries in the Copy to library and Quick switch menus are

normally sorted by most used. However, if there are more than a certain

number of such libraries, the sorting becomes alphabetic. You can set that

number here. The default is ten libraries.

many_libraries

=

10

#: Choose available output formats for conversion

Restrict the list of available output formats in the conversion dialogs.

For example, if you only want to convert to EPUB and AZW3, change this to

restrict_output_formats = ['EPUB', 'AZW3']. The default value of None causes

all available output formats to be present.

`restrict_output_formats`

`=`

None

#: Set the thumbnail image quality used by the Content server

The quality of a thumbnail is largely controlled by the compression quality

used when creating it. Set this to a larger number to improve the quality.

Note that the thumbnails get much larger with larger compression quality

numbers.

The value can be between 50 and 99

`content_server_thumbnail_compression_quality`

`=`

75

#: Image file types to treat as e-books when dropping onto the "Book details" panel

Normally, if you drop any image file in a format known to calibre onto the

"Book details" panel, it will be used to set the cover. If you want to store

some image types as e-books instead, you can set this tweak.

Examples:

cover_drop_exclude = {'tiff', 'webp'}

cover_drop_exclude

=

()

#: Show the Saved searches box in the Search bar

In newer versions of calibre, only a single button that allows you to add a

new Saved search is shown in the Search bar. If you would like to have the

old Saved searches box with its two buttons back, set this tweak to True.

show_saved_search_box

=

False

#: Exclude fields when copy/pasting metadata

You can ask calibre to not paste some metadata fields when using the

Edit metadata->Copy metadata/Paste metadata actions. For example,

exclude_fields_on_paste = ['cover', 'timestamp', '#mycolumn']

to prevent pasting of the cover, Date and custom column, mycolumn.

You can also add a shortcut in Preferences->Shortcuts->Edit metadata

to paste metadata ignoring this tweak.

exclude_fields_on_paste

=

[]

#: Skip internet connected check

Skip checking whether the internet is available before downloading news.

Useful if for some reason your operating systems network checking

facilities are not reliable (for example NetworkManager on Linux).

skip_network_check

=

False

#: Tab stop width in the template editor

Sets the width of the tab stop in the template editor in "average characters".

For example, a value of 1 results in a space with the width of one average character.

template_editor_tab_stop_width

=

4

#: Value for undefined numbers when sorting

Sets the value to use for undefined numbers when sorting.

For example, the value -10 sorts undefined numbers as if they were set to -10.

Use 'maximum' for the largest possible number. Use 'minimum' for the smallest

possible number. Quotes are optional if entering a number.

Examples:


```
# value_for_undefined_numbers_when_sorting = -100

# value_for_undefined_numbers_when_sorting = '2'

# value_for_undefined_numbers_when_sorting = -0.01

# value_for_undefined_numbers_when_sorting = 'minimum'

# value_for_undefined_numbers_when_sorting = 'maximum'

value_for_undefined_numbers_when_sorting

=

0
```

Overriding icons, templates, et cetera

Note

calibre has direct support for icon themes, there are several icon themes available for calibre, that you can use by going to *Preferences* → *Interface* → *Look & Feel* → *Change Icon theme*. The icon themes use the same mechanism as described below for overriding static resources.

calibre allows you to override the static resources, like icons, JavaScript and templates for the metadata jacket, catalogs, etc. with customized versions that you like. All static resources are stored in the resources sub-folder of the calibre install location. On Windows, this is usually C:\Program

Files\Calibre2\app\resources

. On macOS, /Applications/calibre.app/Contents/Resources/resources/

. On Linux, if you are using the binary installer from the calibre website it will be /opt/calibre/resources

. These paths can change depending on where you choose to install calibre.

You should not change the files in this resources folder, as your changes will get overwritten the next time you update calibre. Instead, go to *Preferences* → *Advanced* → *Miscellaneous* and click *Open calibre configuration folder*. In this configuration folder, create a sub-folder called resources and place the files you want to override in it. Place the files in the appropriate sub folders, for example place images in resources/images

, etc. calibre will automatically use your custom file in preference to the built-in one the next time it is started.

For example, if you wanted to change the icon for the *Remove books* action, you would first look in the built-in resources folder and see that the relevant file is `resources/images/remove_books.png`

. Assuming you have an alternate icon in PNG format called `my_remove_books.png` you would save it in the configuration folder as `resources/images/remove_books.png`. All the icons used by the calibre user interface are in `resources/images` and its sub-folders.

[Creating your own icon theme for calibre](#)

If you have created a beautiful set of icons and wish to share them with other calibre users via calibre's builtin icon theme support, you can easily package up your icons into a theme. To do so, go to *Preferences* → *Miscellaneous* → *Create icon theme*, select the folder where you have put your icons (usually the `resources/images` folder in the calibre config folder, as described above). Then fill up the theme metadata and click OK. This will result in a ZIP file containing the theme icons. You can upload that to the calibre forum at [Mobileread](https://www.mobileread.com/forums/forumdisplay.php?f=166) [https://www.mobileread.com/forums/forumdisplay.php?f=166] and then I will make your theme available via calibre's builtin icon theme system.

[Customizing calibre with plugins](#)

calibre has a very modular design. Almost all functionality in calibre comes in the form of plugins. Plugins are used for conversion, for downloading news (though these are called recipes), for various components of the user interface, to connect to different devices, to process files when adding them to calibre and so on. You can get a complete list of all the built-in plugins in calibre by going to *Preferences* → *Advanced* → *Plugins*.

You can write your own plugins to customize and extend the behavior of calibre. The plugin architecture in calibre is very simple, see the tutorial [Writing your own plugins to extend calibre's functionality](#).

Once you have written a plugin, you can upload that to the calibre plugins forum at [Mobileread](https://www.mobileread.com/forums/forumdisplay.php?f=237) [https://www.mobileread.com/forums/forumdisplay.php?f=237] and it will be made available via calibre's builtin plugin updater.

API documentation for plugins

Defines various abstract base classes that can be subclassed to create powerful plugins. The useful classes are:

- [Plugin](#)
- [FileTypePlugin](#)
- [Metadata plugins](#)
- [Catalog plugins](#)
- [Metadata download plugins](#)
- [Conversion plugins](#)
- [Device drivers](#)
- [User interface actions](#)
- [Preferences plugins](#)

Plugin

`class calibre.customize.Plugin(plugin_path)`[\[source\]](#)

A calibre plugin. Useful members include:

- `self.installation_type`
: Stores how the plugin was installed.
- `self.plugin_path`
: Stores path to the ZIP file that contains

this plugin or None if it is a builtin plugin

- `self.site_customization`
: Stores a customization string entered

by the user.

Methods that should be overridden in sub classes:

- `initialize()`
- `customization_help()`

Useful methods:

- `temporary_file()`
- `__enter__()`
- `load_resources()`

```
supported_platforms = []
```

List of platforms this plugin works on. For example: `['windows', 'osx', 'linux']`

```
name = 'Trivial Plugin'
```

The name of this plugin. You must set it something other than Trivial Plugin for it to work.

```
version = (1, 0, 0)
```

The version of this plugin as a 3-tuple (major, minor, revision)

```
description = 'Does absolutely nothing'
```

A short string describing what this plugin does

```
author = 'Unknown'
```

The author of this plugin

`priority = 1`

When more than one plugin exists for a filetype, the plugins are run in order of decreasing priority. Plugins with higher priority will be run first. The highest possible priority is `sys.maxsize`

. Default priority is 1.

`minimum_calibre_version = (0, 4, 118)`

The earliest version of calibre this plugin requires

`installation_type = None`

The way this plugin is installed

`can_be_disabled = True`

If False, the user will not be able to disable this plugin. Use with care.

`type = 'Base'`

The type of this plugin. Used for categorizing plugins in the GUI

`initialize()`[\[source\]](#)

Called once when calibre plugins are initialized. Plugins are re-initialized every time a new plugin is added. Also note that if the plugin is run in a worker process, such as for adding books, then the plugin will be initialized for every new worker process.

Perform any plugin specific initialization here, such as extracting resources from the plugin ZIP file. The path to the ZIP file is available as `self.plugin_path`.

Note that `self.site_customization` is **not** available at this point.

`config_widget()`[\[source\]](#)

Implement this method and [save_settings\(\)](#) in your plugin to use a custom configuration dialog, rather than relying on the simple string based default customization.

This method, if implemented, must return a `QWidget`. The widget can have an optional method `validate()` that takes no arguments and is called immediately after the user clicks OK. Changes are applied if and only if the method returns `True`.

If for some reason you cannot perform the configuration at this time, return a tuple of two strings (message, details), these will be displayed as a warning dialog to the user and the process will be aborted.

`save_settings(config_widget)`[\[source\]](#)

Save the settings specified by the user with `config_widget`.

Parameters **`config_widget`** – The widget returned by [code `config\_widget\(\)`](#)

`do_user_config(parent=None)`[\[source\]](#)

This method shows a configuration dialog for this plugin. It returns True if the user clicks OK, False otherwise. The changes are automatically applied.

`load_resources(names)`[\[source\]](#)

If this plugin comes in a ZIP file (user added plugin), this method will allow you to load resources from the ZIP file.

For example to load an image:

```
 pixmap
=
 QPixmap
()
 pixmap
.
 loadFromData
(
 self
.
 load_resources
([
 'images/icon.png'
 ])[
 'images/icon.png'
 ])
 icon
```

```
=
QIcon
(
pixmap
)
```

Parameters **names** – List of paths to resources in the ZIP file using / as separator

Returns A dictionary of the form {name:
 file_contents}
 . Any names that were not found in the ZIP file will not be present in the dictionary.

customization_help(*gui=False*)[\[source\]](#)

Return a string giving help on how to customize this plugin. By default raise a **NotImplementedError**, which indicates that the plugin does not require customization.

If you re-implement this method in your subclass, the user will be asked to enter a string as customization for this plugin. The customization string will be available as `self.site_customization`.

Site customization could be anything, for example, the path to a needed binary on the user's computer.

Parameters **gui** – If True return HTML help, otherwise return plain text help.

temporary_file(*suffix*)[\[source\]](#)

Return a file-like object that is a temporary file on the file system. This file will remain available even after being closed and will only be removed on interpreter shutdown. Use the `name`

member of the returned object to access the full path to the created temporary file.

Parameters **suffix** – The suffix that the temporary file will have.

cli_main(*args*)[\[source\]](#)

This method is the main entry point for your plugins command line interface. It is called when the user does: `calibre-debug -r "Plugin Name"`. Any arguments passed are present in the `args` variable.

[FileTypePlugin](#)

`class calibre.customize.FileTypePlugin(plugin_path)`[\[source\]](#)

Bases:

[calibre.customize.Plugin](#)

A plugin that is associated with a particular set of file types.

`file_types = {}`

Set of file types for which this plugin should be run. Use '*' for all file types. For example: {'lit',

'mobi',

'prc'}

`on_import = False`

If True, this plugin is run when books are added to the database

`on_postimport = False`

If True, this plugin is run after books are added to the database. In this case the `postimport` and `postadd` methods of the plugin are called.

`on_preprocess = False`

If True, this plugin is run just before a conversion

`on_postprocess = False`

If True, this plugin is run after conversion on the final file produced by the conversion output plugin.

`run(path_to_ebook)`[\[source\]](#)

Run the plugin. Must be implemented in subclasses. It should perform whatever modifications are required on the e-book and return the absolute path to the modified e-book. If no modifications are needed, it should return the path to the original e-book. If an

error is encountered it should raise an Exception. The default implementation simply return the path to the original e-book. Note that the path to the original file (before any file type plugins are run, is available as `self.original_path_to_file`).

The modified e-book file should be created with the **`temporary_file()`** method.

Parameters **`path_to_ebook`** – Absolute path to the e-book.

Returns Absolute path to the modified e-book.

`postimport(book_id, book_format, db)`[\[source\]](#)

Called post import, i.e., after the book file has been added to the database. Note that this is different from **`postadd()`** which is called when the book record is created for the first time. This method is called whenever a new file is added to a book record. It is useful for modifying the book record based on the contents of the newly added file.

Parameters

- **`book_id`** – Database id of the added book.
- **`book_format`** – The file type of the book that was added.
- **`db`** – Library database.

`postadd(book_id, fmt_map, db)`[\[source\]](#)

Called post add, i.e. after a book has been added to the db. Note that this is different from **`postimport()`**, which is called after a single book file has been added to a book. `postadd()` is called only when an entire book record with possibly more than one book file has been created for the first time. This is useful if you wish to modify the book record in the database when the book is first added to calibre.

Parameters

- **`book_id`** – Database id of the added book.
- **`fmt_map`** – Map of file format to path from which the file format was added. Note that this might or might not point to an actual existing file, as sometimes files are added as streams. In which case it might be a dummy value or a non-existent path.
- **`db`** – Library database

Metadata plugins

`class calibre.customize.MetadataReaderPlugin(*args, **kwargs)`[\[source\]](#)

Bases: [calibre.customize.Plugin](#)

A plugin that implements reading metadata from a set of file types.

`file_types = {}`

Set of file types for which this plugin should be run. For example: `set(['lit', 'mobi', 'prc'])`

`get_metadata(stream, type)`[\[source\]](#)

Return metadata for the file represented by stream (a file like object that supports reading). Raise an exception when there is an error with the input data.

Parameters **type** – The type of file. Guaranteed to be one of the entries in [file_types](#).

Returns A [calibre.ebooks.metadata.book.Metadata](#) object

`class calibre.customize.MetadataWriterPlugin(*args, **kwargs)`[\[source\]](#)

Bases: [calibre.customize.Plugin](#)

A plugin that implements reading metadata from a set of file types.

`file_types = {}`

Set of file types for which this plugin should be run. For example: `set(['lit', 'mobi',`

'prc'])

`set_metadata(stream, mi, type)`[\[source\]](#)

Set metadata for the file represented by stream (a file like object that supports reading). Raise an exception when there is an error with the input data.

Parameters

- **type** – The type of file. Guaranteed to be one of the entries in [file_types](#).
- **mi** – A `calibre.ebooks.metadata.book.Metadata` object

Catalog plugins

`class calibre.customize.CatalogPlugin(plugin_path)`[\[source\]](#)

Bases: [calibre.customize.Plugin](#)

A plugin that implements a catalog generator.

`file_types = {}`

Output file type for which this plugin should be run. For example: 'epub' or 'xml'

`cli_options = []`

CLI parser options specific to this plugin, declared as *namedtuple Option*:

```
from collections import namedtuple
Option = namedtuple('Option', 'option, default, dest, help')
cli_options = [Option('-catalog-title', default = 'My Catalog', dest = 'catalog_title', help =
_["Title of generated catalog. nDefault:"] + " '" + '%default' + "'"))]
cli_options parsed in
calibre.db.cli.cmd_catalog:option_parser()
```

`initialize()`[\[source\]](#)

If plugin is not a built-in, copy the plugin's .ui and .py files from the ZIP file to \$TMPDIR. Tab will be dynamically generated and added to the Catalog Options dialog in `calibre.gui2.dialogs.catalog.py:Catalog`

`run(path_to_output, opts, db, ids, notification=None)`[\[source\]](#)

Run the plugin. Must be implemented in subclasses. It should generate the catalog in the format specified in `file_types`, returning the absolute path to the generated catalog file. If an error is encountered it should raise an Exception.

The generated catalog file should be created with the `temporary_file()` method.

Parameters

- **path_to_output** – Absolute path to the generated catalog file.
- **opts** – A dictionary of keyword arguments
- **db** – A LibraryDatabase2 object

Metadata download plugins

`class calibre.ebooks.metadata.sources.base.Source(*args, **kwargs)`[\[source\]](#)

Bases: [calibre.customize.Plugin](#)

`capabilities = frozenset({})`

Set of capabilities supported by this plugin. Useful capabilities are: 'identify', 'cover'

`touched_fields = frozenset({})`

List of metadata fields that can potentially be download by this plugin during the identify phase

`has_html_comments = False`

Set this to True if your plugin returns HTML formatted comments

`supports_gzip_transfer_encoding = False`

Setting this to True means that the browser object will indicate that it supports gzip transfer encoding. This can speedup downloads but make sure that the source actually supports gzip transfer encoding correctly first

`ignore_ssl_errors = False`

Set this to True to ignore HTTPS certificate errors when connecting to this source.

`cached_cover_url_is_reliable = True`

Cached cover URLs can sometimes be unreliable (i.e. the download could fail or the returned image could be bogus). If that is often the case with this source, set to False

`options = []`

A list of **Option** objects. They will be used to automatically construct the configuration widget for this plugin

`config_help_message = None`

A string that is displayed at the top of the config widget for this plugin

`can_get_multiple_covers = False`

If True this source can return multiple covers for a given query

`auto_trim_covers = False`

If set to True covers downloaded by this plugin are automatically trimmed.

`prefer_results_with_isbn = True`

If set to True, and this source returns multiple results for a query, some of which have ISBNs and some of which do not, the results without ISBNs will be ignored

`is_configured()`[\[source\]](#)

Return False if your plugin needs to be configured before it can be used. For example, it might need a username/password/API key.

`customization_help()`[\[source\]](#)

Return a string giving help on how to customize this plugin. By default raise a **NotImplementedError**, which indicates that the plugin does not require customization.

If you re-implement this method in your subclass, the user will be asked to enter a string as customization for this plugin. The customization string will be available as `self.site_customization`.

Site customization could be anything, for example, the path to a needed binary on the user's computer.

Parameters **gui** – If True return HTML help, otherwise return plain text help.

`config_widget()`[\[source\]](#)

Implement this method and [save_settings\(\)](#) in your plugin to use a custom configuration dialog, rather than relying on the simple string based default customization.

This method, if implemented, must return a QWidget. The widget can have an optional method `validate()` that takes no arguments and is called immediately after the user clicks OK. Changes are applied if and only if the method returns True.

If for some reason you cannot perform the configuration at this time, return a tuple of two strings (message, details), these will be displayed as a warning dialog to the user and the process will be aborted.

`save_settings(config_widget)`[\[source\]](#)

Save the settings specified by the user with `config_widget`.

Parameters **config_widget** – The widget returned by [config_widget\(\)](#)

`get_author_tokens(authors, only_first_author=True)`[\[source\]](#)

Take a list of authors and return a list of tokens useful for an AND search query. This function tries to return tokens in first name middle names last name order, by assuming that if a comma is in the author name, the name is in lastname, other names form.

`get_title_tokens(title, strip_joiners=True, strip_subtitle=False)`[\[source\]](#)

Take a title and return a list of tokens useful for an AND search query. Excludes connectives(optionally) and punctuation.

`split_jobs(jobs, num)`[\[source\]](#)

Split a list of jobs into at most num groups, as evenly as possible

`test_fields(mi)`[\[source\]](#)

Return the first field from `self.touched_fields` that is null on the `mi` object

`clean_downloaded_metadata(mi)`[\[source\]](#)

Call this method in your plugin's identify method to normalize metadata before putting the Metadata object into `result_queue`. You can of course, use a custom algorithm suited to your metadata source.

`get_book_url(identifiers)`[\[source\]](#)

Return a 3-tuple or None. The 3-tuple is of the form: (*identifier_type*, *identifier_value*, URL). The URL is the URL for the book identified by *identifiers* at this source. *identifier_type*, *identifier_value* specify the identifier corresponding to the URL. This URL must be browsable to by a human using a browser. It is meant to provide a clickable link for the user to easily visit the books page at this source. If no URL is found, return None. This method must be quick, and consistent, so only implement it if it is possible to construct the URL from a known scheme given *identifiers*.

`get_book_url_name(idtype, idval, url)`[\[source\]](#)

Return a human readable name from the return value of `get_book_url()`.

`get_book_urls(identifiers)`[\[source\]](#)

Override this method if you would like to return multiple urls for this book. Return a list of 3-tuples. By default this method simply calls `get_book_url()`.

`get_cached_cover_url(identifiers)`[\[source\]](#)

Return cached cover URL for the book identified by the *identifiers* dict or None if no such URL exists.

Note that this method must only return validated URLs, i.e. not URLs that could result in a generic cover image or a not found error.

`id_from_url(url)`[\[source\]](#)

Parse a URL and return a tuple of the form: (*identifier_type*, *identifier_value*). If the URL does not match the pattern for the metadata source, return None.

`identify_results_keygen(title=None, authors=None, identifiers={})`[\[source\]](#)

Return a function that is used to generate a key that can sort Metadata objects by their relevance given a search query (*title*, *authors*, *identifiers*).

These keys are used to sort the results of a call to `identify()`.

For details on the default algorithm see [InternalMetadataCompareKeyGen](#). Re-implement this function in your plugin if the default algorithm is not suitable.

`identify(log, result_queue, abort, title=None, authors=None, identifiers={}, timeout=30)`[\[source\]](#)

Identify a book by its Title/Author/ISBN/etc.

If identifiers(s) are specified and no match is found and this metadata source does not store all related identifiers (for example, all ISBNs of a book), this method should retry with just the title and author (assuming they were specified).

If this metadata source also provides covers, the URL to the cover should be cached so that a subsequent call to the get covers API with the same ISBN/special identifier does not need to get the cover URL again. Use the caching API for this.

Every Metadata object put into `result_queue` by this method must have a *source_relevance* attribute that is an integer indicating the order in which the results were returned by the metadata source for this query. This integer will be used by **`compare_identify_results()`**. If the order is unimportant, set it to zero for every result.

Make sure that any cover/ISBN mapping information is cached before the Metadata object is put into `result_queue`.

Parameters

- **log** – A log object, use it to output debugging information/errors
- **result_queue** – A result Queue, results should be put into it. Each result is a Metadata object
- **abort** – If `abort.is_set()` returns True, abort further processing and return as soon as possible
- **title** – The title of the book, can be None
- **authors** – A list of authors of the book, can be None
- **identifiers** – A dictionary of other identifiers, most commonly `{‘isbn’:‘1234...’}`
- **timeout** – Timeout in seconds, no network request should hang for longer than timeout.

Returns None if no errors occurred, otherwise a unicode representation of the error suitable for showing to the user

`download_cover(log, result_queue, abort, title=None, authors=None, identifiers={}, timeout=30, get_best_cover=False)`[\[source\]](#)

Download a cover and put it into `result_queue`. The parameters all have the same meaning as `for` `identify()`. Put `(self, cover_data)` into `result_queue`.

This method should use cached cover URLs for efficiency whenever possible. When cached data is not present, most plugins simply call `identify` and use its results.

If the parameter `get_best_cover` is `True` and this plugin can get multiple covers, it should only get the “best” one.

class calibre.ebooks.metadata.sources.base.InternalMetadataCompareKeyGen(mi, source_plugin, title, authors, identifiers) [\[source\]](#)

Generate a sort key for comparison of the relevance of Metadata objects, given a search query. This is used only to compare results from the same metadata source, not across different sources.

The sort key ensures that an ascending order sort is a sort by order of decreasing relevance.

The algorithm is:

- Prefer results that have at least one identifier the same as for the query
- Prefer results with a cached cover URL
- Prefer results with all available fields filled in
- Prefer results with the same language as the current user interface language
- Prefer results that are an exact title match to the query
- Prefer results with longer comments (greater than 10% longer)
- Use the relevance of the result as reported by the metadata source’s search

engine

[Conversion plugins](#)

*class calibre.customize.conversion.InputFormatPlugin(*args)* [\[source\]](#)

Bases:

[calibre.customize.Plugin](#)

InputFormatPlugins are responsible for converting a document into HTML+OPF+CSS+etc. The results of the conversion *must* be encoded in UTF-8. The main action happens in [convert\(\)](#)

.

`file_types = {}`

Set of file types for which this plugin should be run For example: `set(['azw',`

`'mobi',`

`'prc'])`

`is_image_collection = False`

If True, this input plugin generates a collection of images, one per HTML file. This can be set dynamically, in the convert method if the input files can be both image collections and non-image collections. If you set this to True, you must implement the `get_images()` method that returns a list of images.

`core_usage = 1`

Number of CPU cores used by this plugin. A value of -1 means that it uses all available cores

`for_viewer = False`

If set to True, the input plugin will perform special processing to make its output suitable for viewing

`output_encoding = 'utf-8'`

The encoding that this input plugin creates files in. A value of None means that the encoding is undefined and must be detected individually

`common_options = {<calibre.customize.conversion.OptionRecommendation object>}`

Options shared by all Input format plugins. Do not override in sub-classes. Use [options](#) instead. Every option must be an instance of **OptionRecommendation**

.

`options = {}`

Options to customize the behavior of this plugin. Every option must be an instance of **OptionRecommendation**

.

recommendations = {}

A set of 3-tuples of the form (option_name, recommended_value, recommendation_level)

get_images()[\[source\]](#)

Return a list of absolute paths to the images, if this input plugin represents an image collection. The list of images is in the same order as the spine and the TOC.

convert(*stream*, *options*, *file_ext*, *log*, *accelerators*)[\[source\]](#)

This method must be implemented in sub-classes. It must return the path to the created OPF file or an **OEBBook** instance. All output should be contained in the current folder. If this plugin creates files outside the current folder they must be deleted/marked for deletion before this method returns.

Parameters

- **stream** – A file like object that contains the input file.
- **options** – Options to customize the conversion process. Guaranteed to have attributes corresponding to all the options declared by this plugin. In addition, it will have a verbose attribute that takes integral values from zero upwards. Higher numbers mean be more verbose. Another useful attribute is input_profile that is an instance of **calibre.customize.profiles.InputProfile**.
- **file_ext** – The extension (without the .) of the input file. It is guaranteed to be one of the *file_types* supported by this plugin.
- **log** – A **calibre.utils.logging.Log** object. All output should use this object.
- **accelarators** – A dictionary of various information that the input plugin can get easily that would speed up the subsequent stages of the conversion.

postprocess_book(*oeb*, *opts*, *log*)[\[source\]](#)

Called to allow the input plugin to perform postprocessing after the book has been parsed.

specialize(*oeb*, *opts*, *log*, *output_fmt*)[\[source\]](#)

Called to allow the input plugin to specialize the parsed book for a particular output format. Called after postprocess_book and before any transforms are performed on the parsed book.

`gui_configuration_widget(parent, get_option_by_name, get_option_help, db, book_id=None)`[\[source\]](#)

Called to create the widget used for configuring this plugin in the calibre GUI. The widget must be an instance of the `PluginWidget` class. See the builtin input plugins for examples.

`class calibre.customize.conversion.OutputFormatPlugin(*args)`[\[source\]](#)

Bases:

[calibre.customize.Plugin](#)

`OutputFormatPlugins` are responsible for converting an OEB document (OPF+HTML) into an output e-book.

The OEB document can be assumed to be encoded in UTF-8. The main action happens in [convert\(\)](#)

.

`file_type = None`

The file type (extension without leading period) that this plugin outputs

`common_options = {<calibre.customize.conversion.OptionRecommendation object>}`

Options shared by all Input format plugins. Do not override in sub-classes. Use [options](#) instead. Every option must be an instance of **OptionRecommendation**

.

`options = {}`

Options to customize the behavior of this plugin. Every option must be an instance of **OptionRecommendation**

.

`recommendations = {}`

A set of 3-tuples of the form (option_name, recommended_value, recommendation_level)

property description

`str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str`

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of `object.__str__()` (if defined) or `repr(object)`. encoding defaults to `sys.getdefaultencoding()`. errors defaults to 'strict'.

`convert(oeb_book, output, input_plugin, opts, log)`[\[source\]](#)

Render the contents of *oeb_book* (which is an instance of **calibre.ebooks.oeb.OEBBook**) to the file specified by *output*.

Parameters

- **output** – Either a file like object or a string. If it is a string it is the path to a folder that may or may not exist. The output plugin should write its output into that folder. If it is a file like object, the output plugin should write its output into the file.
- **input_plugin** – The input plugin that was used at the beginning of the conversion pipeline.
- **opts** – Conversion options. Guaranteed to have attributes corresponding to the OptionRecommendations of this plugin.
- **log** – The logger. Print debug/info messages etc. using this.

`specialize_options(log, opts, input_fmt)`[\[source\]](#)

Can be used to change the values of conversion options, as used by the conversion pipeline.

`specialize_css_for_output(log, opts, item, stylizer)`[\[source\]](#)

Can be used to make changes to the css during the CSS flattening process.

Parameters

- **item** – The item (HTML file) being processed
- **stylizer** – A Stylizer object containing the flattened styles for item. You can get the style for any element by `stylizer.style(element)`.

`gui_configuration_widget(parent, get_option_by_name, get_option_help, db, book_id=None)`[\[source\]](#)

Called to create the widget used for configuring this plugin in the calibre GUI. The widget must be an instance of the PluginWidget class. See the builtin output plugins for examples.

Device drivers

The base class for all device drivers is [DevicePlugin](#). However, if your device exposes itself as a USBMS drive to the operating system, you should use the USBMS class instead as it implements all the logic needed to support these kinds of devices.

`class calibre.devices.interface.DevicePlugin(plugin_path)`[\[source\]](#)

Bases:

[calibre.customize.Plugin](#)

Defines the interface that should be implemented by backends that communicate with an e-book reader.

`FORMATS = ['lrf', 'rtf', 'pdf', 'txt']`

Ordered list of supported formats

`VENDOR_ID = 0`

VENDOR_ID can be either an integer, a list of integers or a dictionary. If it is a dictionary, it must be a dictionary of dictionaries, of the form:

```
{  
  
integer_vendor_id  
  
:  
  
{  
  
product_id  
  
:  
  
[  
list  
  
of  
  
BCDs  
],
```

```
...
```

```
},
```

```
...
```

```
}
```

`PRODUCT_ID = 0`

An integer or a list of integers

`BCD = None`

BCD can be either None to not distinguish between devices based on BCD, or it can be a list of the BCD numbers of all devices supported by this driver.

`THUMBNAIL_HEIGHT = 68`

Height for thumbnails on the device

`THUMBNAIL_COMPRESSION_QUALITY = 75`

Compression quality for thumbnails. Set this closer to 100 to have better quality thumbnails with fewer compression artifacts. Of course, the thumbnails get larger as well.

`WANTS_UPDATED_THUMBNAILS = False`

Set this to True if the device supports updating cover thumbnails during sync_booklists. Setting it to true will ask device.py to refresh the cover thumbnails during book matching

`CAN_SET_METADATA = ['title', 'authors', 'collections']`

Whether the metadata on books can be set via the GUI.

`CAN_DO_DEVICE_DB_PLUGBOARD = False`

Whether the device can handle device_db metadata plugboards

`path_sep = '/'`

Path separator for paths to books on device

`icon = '/home/kovid/work/calibre/resources/images/reader.png'`

Icon for this device

UserAnnotation

alias of **calibre.devices.interface.Annotation**

OPEN_FEEDBACK_MESSAGE = *None*

GUI displays this as a message if not None. Useful if opening can take a long time

VIRTUAL_BOOK_EXTENSIONS = *frozenset({})*

Set of extensions that are “virtual books” on the device and therefore cannot be viewed/saved/added to library. For example: *frozenset(['kobo'])*

VIRTUAL_BOOK_EXTENSION_MESSAGE = *None*

Message to display to user for virtual book extensions.

NUKE_COMMENTS = *None*

Whether to nuke comments in the copy of the book sent to the device. If not None this should be short string that the comments will be replaced by.

MANAGES_DEVICE_PRESENCE = *False*

If True indicates that this driver completely manages device detection, ejecting and so forth. If you set this to True, you *must* implement the `detect_managed_devices` and `debug_managed_device_detection` methods. A driver with this set to true is responsible for detection of devices, managing a blacklist of devices, a list of ejected devices and so forth. calibre will periodically call the `detect_managed_devices()` method and if it returns a detected device, calibre will call `open()`. `open()` will be called every time a device is returned even if previous calls to `open()` failed, therefore the driver must maintain its own blacklist of failed devices. Similarly, when ejecting, calibre will call `eject()` and then assuming the next call to `detect_managed_devices()` returns None, it will call `post_yank_cleanup()`.

SLOW_DRIVEINFO = *False*

If set the True, calibre will call the `get_driveinfo()` method after the books lists have been loaded to get the driveinfo.

ASK_TO_ALLOW_CONNECT = *False*

If set to True, calibre will ask the user if they want to manage the device with calibre, the first time it is detected. If you set this to True you must implement `get_device_uid()`

and
and
and

[ignore_connected_device\(\)](#)
[get_user_blacklisted_devices\(\)](#)
[set_user_blacklisted_devices\(\)](#)

`user_feedback_after_callback = None`

Set this to a dictionary of the form `{'title':title, 'msg':msg, 'det_msg':detailed_msg}` to have calibre popup a message to the user after some callbacks are run (currently only `upload_books`). Be careful to not spam the user with too many messages. This variable is checked after *every* callback, so only set it when you really need to.

`is_usb_connected(devices_on_system, debug=False, only_presence=False)`[\[source\]](#)

Return True, `device_info` if a device handled by this plugin is currently connected.

Parameters **devices_on_system** – List of devices currently connected

`detect_managed_devices(devices_on_system, force_refresh=False)`[\[source\]](#)

Called only if `MANAGES_DEVICE_PRESENCE` is True.

Scan for devices that this driver can handle. Should return a device object if a device is found. This object will be passed to the `open()` method as the `connected_device`. If no device is found, return None. The returned object can be anything, calibre does not use it, it is only passed to `open()`.

This method is called periodically by the GUI, so make sure it is not too resource intensive. Use a cache to avoid repeatedly scanning the system.

Parameters

- **devices_on_system** – Set of USB devices found on the system.
- **force_refresh** – If True and the driver uses a cache to prevent repeated scanning, the cache must be flushed.

`debug_managed_device_detection(devices_on_system, output)`[\[source\]](#)

Called only if `MANAGES_DEVICE_PRESENCE` is True.

Should write information about the devices detected on the system to `output`, which is a file like object.

Should return True if a device was detected and successfully opened, otherwise False.

`reset(key='-1', log_packets=False, report_progress=None, detected_device=None)`[\[source\]](#)

Parameters

- **key** – The key to unlock the device
- **log_packets** – If true the packet stream to/from the device is logged
- **report_progress** – Function that is called with a % progress (number between 0 and 100) for various tasks. If it is called with -1 that means that the task does not have any progress information
- **detected_device** – Device information from the device scanner

`can_handle_windows(usbdevice, debug=False)`[\[source\]](#)

Optional method to perform further checks on a device to see if this driver is capable of handling it. If it is not it should return False. This method is only called after the vendor, product ids and the bcd have matched, so it can do some relatively time intensive checks. The default implementation returns True. This method is called only on Windows. See also [can_handle\(\)](#)

.

Note that for devices based on USBMS this method by default delegates to [can_handle\(\)](#). So you only need to override [can_handle\(\)](#) in your subclass of USBMS.

Parameters **usbdevice** – A usbdevice as returned by `calibre.devices.winusb.scan_usb_devices()`

`can_handle(device_info, debug=False)`[\[source\]](#)

Unix version of [can_handle_windows\(\)](#)

.

Parameters **device_info** – Is a tuple of (vid, pid, bcd, manufacturer, product, serial number)

`open(connected_device, library_uuid)`[\[source\]](#)

Perform any device specific initialization. Called after the device is detected but before any other functions that communicate with the device. For example: For devices that present themselves as USB Mass storage devices, this method would be responsible for mounting the

device or if the device has been automounted, for finding out where it has been mounted. The `calibre.devices.usbms.device.Device.open()` method has an implementation of this function that should serve as a good example for USB Mass storage devices.

This method can raise an `OpenFeedback` exception to display a message to the user.

Parameters

- **connected_device** – The device that we are trying to open. It is a tuple of (vendor id, product id, bcd, manufacturer name, product name, device serial number). However, some devices have no serial number and on Windows only the first three fields are present, the rest are `None`.
- **library_uuid** – The UUID of the current calibre library. Can be `None` if there is no library (for example when used from the command line).

`eject()`[\[source\]](#)

Un-mount / eject the device from the OS. This does not check if there are pending GUI jobs that need to communicate with the device.

NOTE: That this method may not be called on the same thread as the rest of the device methods.

`post_yank_cleanup()`[\[source\]](#)

Called if the user yanks the device without ejecting it first.

`set_progress_reporter(report_progress)`[\[source\]](#)

Set a function to report progress information.

Parameters **report_progress** – Function that is called with a % progress (number between 0 and 100) for various tasks. If it is called with -1 that means that the task does not have any progress information

`get_device_information(end_session=True)`[\[source\]](#)

Ask device for device information. See `L{DeviceInfoQuery}`.

Returns (device name, device version, software version on device, MIME type)
The tuple can optionally have a fifth element, which is a drive information dictionary. See `usbms.driver` for an example.

`get_driveinfo()`[\[source\]](#)

Return the driveinfo dictionary. Usually called from `get_device_information()`, but if loading the driveinfo is slow for this driver, then it should set `SLOW_DRIVEINFO`. In this case, this method will be called by calibre after the book lists have been loaded. Note that it is not called on the device thread, so the driver should cache the drive info in the `books()` method and this function should return the cached data.

`card_prefix(end_session=True)`[\[source\]](#)

Return a 2 element list of the prefix to paths on the cards. If no card is present None is set for the card's prefix. E.G. ('/place', '/place2') (None, 'place2') ('place', None) (None, None)

`total_space(end_session=True)`[\[source\]](#)

Get total space available on the mountpoints:

1. Main memory
2. Memory Card A
3. Memory Card B

Returns A 3 element list with total space in bytes of (1, 2, 3). If a particular device doesn't have any of these locations it should return 0.

`free_space(end_session=True)`[\[source\]](#)

Get free space available on the mountpoints:

1. Main memory
2. Card A
3. Card B

Returns A 3 element list with free space in bytes of (1, 2, 3). If a particular device doesn't have any of these locations it should return -1.

`books(oncard=None, end_session=True)`[\[source\]](#)

Return a list of e-books on the device.

Parameters **oncard** – If 'carda' or 'cardb' return a list of e-books on the specific storage card, otherwise return list of e-books in main memory of device. If a card is specified and no books are on the card return empty list.

Returns A BookList.

`upload_books(files, names, on_card=None, end_session=True, metadata=None)`[\[source\]](#)

Upload a list of books to the device. If a file already exists on the device, it should be replaced. This method should raise a **FreeSpaceError** if there is not enough free space on the device. The text of the **FreeSpaceError** must contain the word “card” if `on_card` is not `None` otherwise it must contain the word “memory”.

Parameters

- **files** – A list of paths
- **names** – A list of file names that the books should have once uploaded to the device. `len(names) == len(files)`
- **metadata** – If not `None`, it is a list of **Metadata** objects. The idea is to use the metadata to determine where on the device to put the book. `len(metadata) == len(files)`. Apart from the regular cover (path to cover), there may also be a thumbnail attribute, which should be used in preference. The thumbnail attribute is of the form (width, height, cover_data as jpeg).

Returns A list of 3-element tuples. The list is meant to be passed to [add_books_to_metadata\(\)](#)

classmethod `add_books_to_metadata(locations, metadata, booklists)`[\[source\]](#)

Add locations to the booklists. This function must not communicate with the device.

Parameters

- **locations** – Result of a call to `L{upload_books}`
- **metadata** – List of **Metadata** objects, same as for [upload_books\(\)](#).
- **booklists** – A tuple containing the result of calls to `(books(ocard=None)(), books(ocard='carda')(), :meth`books(ocard='cardb`'))`.

`delete_books(paths, end_session=True)`[\[source\]](#)

Delete books at paths on device.

classmethod `remove_books_from_metadata(paths, booklists)`[\[source\]](#)

Remove books from the metadata list. This function must not communicate with the device.

Parameters

- **paths** – paths to books on the device.
- **booklists** – A tuple containing the result of calls to `(books(ocard=None)(), books(ocard='carda')(), :meth`books(ocard='cardb')`)`.

`sync_booklists(booklists, end_session=True)`[\[source\]](#)

Update metadata on device.

Parameters **booklists** – A tuple containing the result of calls to `(books(ocard=None)(), books(ocard='carda')(), :meth`books(ocard='cardb')`)`.

`get_file(path, outfile, end_session=True)`[\[source\]](#)

Read the file at path on the device and write it to outfile.

Parameters **outfile** – file object like `sys.stdout` or the result of an [open\(\)](#) call.

classmethod `config_widget()`[\[source\]](#)

Should return a QWidget. The QWidget contains the settings for the device interface

classmethod `save_settings(settings_widget)`[\[source\]](#)

Should save settings to disk. Takes the widget created in [config_widget\(\)](#) and saves all settings to disk.

`classmethod settings()`[\[source\]](#)

Should return an opts object. The opts object should have at least one attribute *format_map* which is an ordered list of formats for the device.

`set_plugboards(plugboards, pb_func)`[\[source\]](#)

provide the driver the current set of plugboards and a function to select a specific plugboard. This method is called immediately before `add_books` and `sync_booklists`.

`pb_func` is a callable with the following signature::

```
def pb_func(device_name, format, plugboards)
```

You give it the current device name (either the class name or `DEVICE_PLUGBOARD_NAME`), the format you are interested in (a ‘real’ format or ‘device_db’), and the plugboards (you were given those by `set_plugboards`, the same place you got this method).

Returns None or a single plugboard instance.

`set_driveinfo_name(location_code, name)`[\[source\]](#)

Set the device name in the driveinfo file to ‘name’. This setting will persist until the file is recreated or the name is changed again.

Non-disk devices should implement this method based on the location codes returned by the `get_device_information()` method.

`prepare_addable_books(paths)`[\[source\]](#)

Given a list of paths, returns another list of paths. These paths point to addable versions of the books.

If there is an error preparing a book, then instead of a path, the position in the returned list for that book should be a three tuple: (original_path, the exception instance, traceback)

`startup()`[\[source\]](#)

Called when calibre is starting the device. Do any initialization required. Note that multiple instances of the class can be instantiated, and thus `__init__` can be called multiple times, but only one instance will have this method called. This method is called on the device thread, not the GUI thread.

`shutdown()`[\[source\]](#)

Called when calibre is shutting down, either for good or in preparation to restart. Do any cleanup required. This method is called on the device thread, not the GUI thread.

`get_device_uid()`[\[source\]](#)

Must return a unique id for the currently connected device (this is called immediately after a successful call to `open()`). You must implement this method if you set `ASK_TO_ALLOW_CONNECT = True`

`ignore_connected_device(uid)`[\[source\]](#)

Should ignore the device identified by *uid* (the result of a call to `get_device_uid()`) in the future. You must implement this method if you set `ASK_TO_ALLOW_CONNECT = True`. Note that this function is called immediately after `open()`, so if `open()` caches some state, the driver should reset that state.

`get_user_blacklisted_devices()`[\[source\]](#)

Return map of device uid to friendly name for all devices that the user has asked to be ignored.

`set_user_blacklisted_devices(devices)`[\[source\]](#)

Set the list of device uids that should be ignored by this driver.

`specialize_global_preferences(device_prefs)`[\[source\]](#)

Implement this method if your device wants to override a particular preference. You must ensure that all call sites that want a preference that can be overridden use `device_prefs['something']` instead of `prefs['something']`. Your method should call `device_prefs.set_overrides(pref=val, pref=val, ...)`. Currently used for: metadata management (`prefs['manage_device_metadata']`)

`set_library_info(library_name, library_uuid, field_metadata)`[\[source\]](#)

Implement this method if you want information about the current calibre library. This method is called at startup and when the calibre library changes while connected.

`is_dynamically_controllable()`[\[source\]](#)

Called by the device manager when starting plugins. If this method returns a string, then a) it supports the device manager's dynamic control interface, and b) that name is to be used when talking to the plugin.

This method can be called on the GUI thread. A driver that implements this method must be thread safe.

`start_plugin()`[\[source\]](#)

This method is called to start the plugin. The plugin should begin to accept device connections however it does that. If the plugin is already accepting connections, then do nothing.

This method can be called on the GUI thread. A driver that implements this method must be thread safe.

`stop_plugin()`[\[source\]](#)

This method is called to stop the plugin. The plugin should no longer accept connections, and should cleanup behind itself. It is likely that this method should call shutdown. If the plugin is already not accepting connections, then do nothing.

This method can be called on the GUI thread. A driver that implements this method must be thread safe.

`get_option(opt_string, default=None)`[\[source\]](#)

Return the value of the option indicated by `opt_string`. This method can be called when the plugin is not started. Return `None` if the option does not exist.

This method can be called on the GUI thread. A driver that implements this method must be thread safe.

`set_option(opt_string, opt_value)`[\[source\]](#)

Set the value of the option indicated by `opt_string`. This method can be called when the plugin is not started.

This method can be called on the GUI thread. A driver that implements this method must be thread safe.

`is_running()`[\[source\]](#)

Return `True` if the plugin is started, otherwise `false`

This method can be called on the GUI thread. A driver that implements this method must be thread safe.

`synchronize_with_db(db, book_id, book_metadata, first_call)`[\[source\]](#)

Called during book matching when a book on the device is matched with a book in calibre's db. The method is responsible for synchronizing data from the device to calibre's db (if needed).

The method must return a two-value tuple. The first value is a set of calibre book ids changed if calibre's database was changed or `None` if the database was not changed. If the first value

is an empty set then the metadata for the book on the device is updated with calibre's metadata and given back to the device, but no GUI refresh of that book is done. This is useful when the calibre data is correct but must be sent to the device.

The second value is itself a 2-value tuple. The first value in the tuple specifies whether a book format should be sent to the device. The intent is to permit verifying that the book on the device is the same as the book in calibre. This value must be None if no book is to be sent, otherwise return the base file name on the device (a string like foobar.epub). Be sure to include the extension in the name. The device subsystem will construct a send_books job for all books with not- None returned values. Note: other than to later retrieve the extension, the name is ignored in cases where the device uses a template to generate the file name, which most do. The second value in the returned tuple indicated whether the format is future-dated. Return True if it is, otherwise return False. calibre will display a dialog to the user listing all future dated books.

Extremely important: this method is called on the GUI thread. It must be threadsafe with respect to the device manager's thread.

book_id: the calibre id for the book in the database. book_metadata: the Metadata object for the book coming from the device. first_call: True if this is the first call during a sync, False otherwise

`class calibre.devices.interface.BookList(oncard, prefix, settings)`[\[source\]](#)

Bases:

list

A list of books. Each Book object must have the fields

1. title
2. authors
3. size (file size of the book)
4. datetime (a UTC time tuple)
5. path (path on the device to the book)
6. thumbnail (can be None) thumbnail is either a str/bytes object with the image data or it should have an attribute image_path that stores an absolute (platform native) path to the image
7. tags (a list of strings, can be empty).

`supports_collections()`[\[source\]](#)

Return True if the device supports collections for this book list.

`add_book(book, replace_metadata)`[\[source\]](#)

Add the book to the booklist. Intent is to maintain any device-internal metadata. Return True if booklists must be sync'ed

`remove_book(book)`[\[source\]](#)

Remove a book from the booklist. Correct any device metadata at the same time

`get_collections(collection_attributes)`[\[source\]](#)

Return a dictionary of collections created from `collection_attributes`. Each entry in the dictionary is of the form collection name:[list of books]

The list of books is sorted by book title, except for collections created from series, in which case `series_index` is used.

Parameters **collection_attributes** – A list of attributes of the Book object

USB Mass Storage based devices

The base class for such devices is [calibre.devices.usbms.driver.USBMS](#). This class in turn inherits some of its functionality from its bases, documented below. A typical basic USBMS based driver looks like this:

```
from
calibre.devices.usbms.driver

import
USBMS

class
PDNOVEL
(
USBMS
):

name
```

=

'Pandigital Novel device interface'

gui_name

=

'PD Novel'

description

=

—
(

'Communicate with the Pandigital Novel'

)

author

=

'Kovid Goyal'

supported_platforms

=

[

'windows'

,

'linux'

,

'osx'

]

FORMATS

=

```
[  
  'epub'  
,  
  'pdf'  
]
```

VENDOR_ID

=

```
[  
  0x18d1  
]
```

PRODUCT_ID

=

```
[  
  0xb004  
]
```

BCD

=

```
[  
  0x224  
]
```

THUMBNAIL_HEIGHT

=

144

EBOOK_DIR_MAIN

=

'eBooks'

SUPPORTS_SUB_DIRS

=

False

def

upload_cover

(

self

,

path

,

filename

,

metadata

):

coverdata

=

```
getattr
(
metadata
,
'thumbnail'
,
None
)
```

```
if
```

```
coverdata
```

```
and
```

```
coverdata
[
2
]:
```

```
with
```

```
open
(
'
```

```
%s
.jpg'
```

```
%
```

```
os
```

```
.
path
```

```
.
join
```

```
(
path
```

```
,
```

```

filename
),

'wb'
)

as

coverfile
:

coverfile
.
write
(
coverdata
[
2
])

```

`class calibre.devices.usbms.device.Device(plugin_path)`[\[source\]](#)

Bases: `calibre.devices.usbms.deviceconfig.DeviceConfig`
`calibre.devices.interface.DevicePlugin`

This class provides logic common to all drivers for devices that export themselves as USB Mass Storage devices. Provides implementations for mounting/ejecting of USBMS devices on all platforms.

`WINDOWS_MAIN_MEM = None`

String identifying the main memory of the device in the Windows PnP id strings This can be None, string, list of strings or compiled regex

`WINDOWS_CARD_A_MEM = None`

String identifying the first card of the device in the Windows PnP id strings This can be None, string, list of strings or compiled regex

`WINDOWS_CARD_B_MEM = None`

String identifying the second card of the device in the Windows PnP id strings This can be None, string, list of strings or compiled regex

OSX_MAIN_MEM_VOL_PAT = *None*

Used by the new driver detection to disambiguate main memory from storage cards. Should be a regular expression that matches the main memory mount point assigned by macOS

BACKLOADING_ERROR_MESSAGE = *None*

MAX_PATH_LEN = 250

The maximum length of paths created on the device

NEWS_IN_FOLDER = *True*

Put news in its own folder

`reset(key='-1', log_packets=False, report_progress=None, detected_device=None)`[\[source\]](#)

Parameters

- **key** – The key to unlock the device
- **log_packets** – If true the packet stream to/from the device is logged
- **report_progress** – Function that is called with a % progress (number between 0 and 100) for various tasks. If it is called with -1 that means that the task does not have any progress information
- **detected_device** – Device information from the device scanner

`set_progress_reporter(report_progress)`[\[source\]](#)

Set a function to report progress information.

Parameters **report_progress** – Function that is called with a % progress (number between 0 and 100) for various tasks. If it is called with -1 that means that the task does not have any progress information

`card_prefix(end_session=True)`[\[source\]](#)

Return a 2 element list of the prefix to paths on the cards. If no card is present None is set for the card's prefix. E.G. ('/place', '/place2') (None, 'place2') ('place', None) (None, None)

`total_space(end_session=True)`[\[source\]](#)

Get total space available on the mountpoints:

1. Main memory
2. Memory Card A
3. Memory Card B

Returns A 3 element list with total space in bytes of (1, 2, 3). If a particular device doesn't have any of these locations it should return 0.

`free_space(end_session=True)`[\[source\]](#)

Get free space available on the mountpoints:

1. Main memory
2. Card A
3. Card B

Returns A 3 element list with free space in bytes of (1, 2, 3). If a particular device doesn't have any of these locations it should return -1.

`windows_sort_drives(drives)`[\[source\]](#)

Called to disambiguate main memory and storage card for devices that do not distinguish between them on the basis of *WINDOWS_CARD_NAME*. For example: The EB600

`can_handle_windows(usbdevice, debug=False)`[\[source\]](#)

Optional method to perform further checks on a device to see if this driver is capable of handling it. If it is not it should return False. This method is only called after the vendor, product ids and the bcd have matched, so it can do some relatively time intensive checks. The default implementation returns True. This method is called only on Windows. See also **can_handle()**

.

Note that for devices based on USBMS this method by default delegates to **can_handle()**. So you only need to override **can_handle()** in your subclass of USBMS.

Parameters **usbdevice** – A usbdevice as returned by **calibre.devices.winusb.scan_usb_devices()**

`open(connected_device, library_uuid)`[\[source\]](#)

Perform any device specific initialization. Called after the device is detected but before any other functions that communicate with the device. For example: For devices that present themselves as USB Mass storage devices, this method would be responsible for mounting the device or if the device has been automounted, for finding out where it has been mounted. The `calibre.devices.usbms.device.Device.open()` method has an implementation of this function that should serve as a good example for USB Mass storage devices.

This method can raise an `OpenFeedback` exception to display a message to the user.

Parameters

- **connected_device** – The device that we are trying to open. It is a tuple of (vendor id, product id, bcd, manufacturer name, product name, device serial number). However, some devices have no serial number and on Windows only the first three fields are present, the rest are `None`.
- **library_uuid** – The UUID of the current calibre library. Can be `None` if there is no library (for example when used from the command line).

`eject()`[\[source\]](#)

Un-mount / eject the device from the OS. This does not check if there are pending GUI jobs that need to communicate with the device.

NOTE: That this method may not be called on the same thread as the rest of the device methods.

`post_yank_cleanup()`[\[source\]](#)

Called if the user yanks the device without ejecting it first.

`sanitize_callback(path)`[\[source\]](#)

Callback to allow individual device drivers to override the path sanitization used by `create_upload_path()`

.

`filename_callback(default, mi)`[\[source\]](#)

Callback to allow drivers to change the default file name set by `create_upload_path()`

.

`sanitize_path_components(components)`[\[source\]](#)

Perform any device specific sanitization on the path components for files to be uploaded to the device

`get_annotations(path_map)`[\[source\]](#)

Resolve `path_map` to `annotation_map` of files found on the device

`add_annotation_to_library(db, db_id, annotation)`[\[source\]](#)

Add an annotation to the calibre library

`class calibre.devices.usbms.cli.CLI`[\[source\]](#)

`class calibre.devices.usbms.driver.USBMS(plugin_path)`[\[source\]](#)

Bases:

,

[calibre.devices.usbms.cli.CLI](#)
[calibre.devices.usbms.device.Device](#)

The base class for all USBMS devices. Implements the logic for sending/getting/updating metadata/caching metadata/etc.

`booklist_class`

alias of [calibre.devices.usbms.books.BookList](#)

`book_class`

alias of [calibre.devices.usbms.books.Book](#)

`get_device_information(end_session=True)`[\[source\]](#)

Ask device for device information. See `L{DeviceInfoQuery}`.

Returns (device name, device version, software version on device, MIME type)
The tuple can optionally have a fifth element, which is a drive information dictionary. See `usbms.driver` for an example.

`set_driveinfo_name(location_code, name)`[\[source\]](#)

Set the device name in the driveinfo file to 'name'. This setting will persist until the file is re-created or the name is changed again.

Non-disk devices should implement this method based on the location codes returned by the `get_device_information()` method.

`books(ocard=None, end_session=True)`[\[source\]](#)

Return a list of e-books on the device.

Parameters **ocard** – If 'carda' or 'cardb' return a list of e-books on the specific storage card, otherwise return list of e-books in main memory of device. If a card is specified and no books are on the card return empty list.

Returns A BookList.

`upload_books(files, names, on_card=None, end_session=True, metadata=None)`[\[source\]](#)

Upload a list of books to the device. If a file already exists on the device, it should be replaced. This method should raise a **FreeSpaceError** if there is not enough free space on the device. The text of the FreeSpaceError must contain the word "card" if `on_card` is not None otherwise it must contain the word "memory".

Parameters

- **files** – A list of paths
- **names** – A list of file names that the books should have once uploaded to the device. `len(names) == len(files)`
- **metadata** – If not None, it is a list of **Metadata** objects. The idea is to use the metadata to determine where on the device to put the book. `len(metadata) == len(files)`. Apart from the regular cover (path to cover), there may also be a thumbnail attribute, which should be used in preference. The thumbnail attribute is of the form (width, height, cover_data as jpeg).

Returns A list of 3-element tuples. The list is meant to be passed to [add_books_to_metadata\(\)](#)

`upload_cover(path, filename, metadata, filepath)`[\[source\]](#)

Upload book cover to the device. Default implementation does nothing.

Parameters

- **path** – The full path to the folder where the associated book is located.
- **filename** – The name of the book file without the extension.
- **metadata** – metadata belonging to the book. Use metadata.thumbnail for cover
- **filepath** – The full path to the e-book file

`add_books_to_metadata(locations, metadata, booklists)`[\[source\]](#)

Add locations to the booklists. This function must not communicate with the device.

Parameters

- **locations** – Result of a call to `L{upload_books}`
- **metadata** – List of **Metadata** objects, same as for [upload_books\(\)](#).
- **booklists** – A tuple containing the result of calls to `(books(ocard=None)(), books(ocard='carda')(), :meth`books(ocard='cardb')`)`.

`delete_books(paths, end_session=True)`[\[source\]](#)

Delete books at paths on device.

`remove_books_from_metadata(paths, booklists)`[\[source\]](#)

Remove books from the metadata list. This function must not communicate with the device.

Parameters

- **paths** – paths to books on the device.
- **booklists** – A tuple containing the result of calls to `(books(ocard=None)(), books(ocard='carda')(), :meth`books(ocard='cardb')`)`.

Sub-classes should implement the `genesis()`, `library_changed()`, `location_selected()`, `shutting_down()`, `initialization_complete()` and `tag_browser_context_action()` methods.

Once initialized, this plugin has access to the main calibre GUI via the **gui** member. You can access other plugins by name, for example:

```
self
.
gui
.
iactions
[
'Save To Disk'
]
```

To access the actual plugin, use the **interface_action_base_plugin** attribute, this attribute only becomes available after the plugin has been initialized. Useful if you want to use methods from the plugin class like `do_user_config()`.

The QAction specified by **action_spec** is automatically create and made available as `self.qaction`.

`name = 'Implement me'`

The plugin name. If two plugins with the same name are present, the one with higher priority takes precedence.

`priority = 1`

The plugin priority. If two plugins with the same name are present, the one with higher priority takes precedence.

`popup_type = 1`

The menu popup type for when this plugin is added to a toolbar

`auto_repeat = False`

Whether this action should be auto repeated when its shortcut key is held down.

`action_spec = ('text', 'icon', None, None)`

Of the form: (text, icon_path, tooltip, keyboard shortcut) icon, tooltip and keyboard shortcut can be None shortcut must be a string, None or tuple of shortcuts. If None, a keyboard shortcut corresponding to the action is not registered. If you pass an empty tuple, then the shortcut is registered with no default key binding.

`action_add_menu = False`

If True, a menu is automatically created and added to `self.qaction`

`action_menu_clone_qaction = False`

If True, a clone of `self.qaction` is added to the menu of `self.qaction`. If you want the text of this action to be different from that of `self.qaction`, set this variable to the new text

`dont_add_to = frozenset({})`

Set of locations to which this action must not be added. See **`all_locations`** for a list of possible locations

`dont_remove_from = frozenset({})`

Set of locations from which this action must not be removed. See **`all_locations`** for a list of possible locations

`action_type = 'global'`

Type of action 'current' means acts on the current view 'global' means an action that does not act on the current view, but rather on calibre as a whole

`accepts_drops = False`

If True, then this `InterfaceAction` will have the opportunity to interact with drag and drop events. See the methods, **`accept_enter_event()`**, **`accept_drag_move_event()`**, **`drop_event()`** for details.

`accept_enter_event(event, mime_data)`[\[source\]](#)

This method should return True iff this interface action is capable of handling the drag event. Do not call `accept/ignore` on the event, that will be taken care of by the calibre UI.

`accept_drag_move_event(event, mime_data)`[\[source\]](#)

This method should return True iff this interface action is capable of handling the drag event. Do not call `accept/ignore` on the event, that will be taken care of by the calibre UI.

`drop_event(event, mime_data)`[\[source\]](#)

This method should perform some useful action and return True iff this interface action is capable of handling the drop event. Do not call `accept/ignore` on the event, that will be taken care of by the calibre UI. You should not perform blocking/long operations in this function.

Instead emit a signal or use `QTimer.singleShot` and return quickly. See the builtin actions for examples.

`create_menu_action(menu, unique_name, text, icon=None, shortcut=None, description=None, triggered=None, shortcut_name=None, persist_shortcut=False)`[\[source\]](#)

Convenience method to easily add actions to a `QMenu`. Returns the created `QAction`. This action has one extra attribute `calibre_shortcut_unique_name` which if not `None` refers to the unique name under which this action is registered with the keyboard manager.

Parameters

- **menu** – The `QMenu` the newly created action will be added to
- **unique_name** – A unique name for this action, this must be globally unique, so make it as descriptive as possible. If in doubt, add an UUID to it.
- **text** – The text of the action.
- **icon** – Either a `QIcon` or a file name. The file name is passed to the `IQ` builtin, so you do not need to pass the full path to the images folder.
- **shortcut** – A string, a list of strings, `None` or `False`. If `False`, no keyboard shortcut is registered for this action. If `None`, a keyboard shortcut with no default keybinding is registered. String and list of strings register a shortcut with default keybinding as specified.
- **description** – A description for this action. Used to set tooltips.
- **triggered** – A callable which is connected to the triggered signal of the created action.
- **shortcut_name** – The text displayed to the user when customizing the keyboard shortcuts for this action. By default it is set to the value of `text`.
- **persist_shortcut** – Shortcuts for actions that don't always appear, or are library dependent, may disappear when other keyboard shortcuts are edited unless `'persist_shortcut'` is set `True`.

`load_resources(names)`[\[source\]](#)

If this plugin comes in a ZIP file (user added plugin), this method will allow you to load resources from the ZIP file.

For example to load an image:

```
 pixmap
=
 QPixmap
()
 pixmap
.
loadFromData
(
 tuple
(
 self
.
load_resources
([
'images/icon.png'
])
.
values
())[
0
])
 icon
=
 QIcon
(
 pixmap
)
```

Parameters **names** – List of paths to resources in the ZIP file using / as separator

Returns

A dictionary of the form {name
:

file_contents}

. Any names that were not found in the ZIP file will not be present in the dictionary.

`genesis()`[\[source\]](#)

Setup this plugin. Only called once during initialization. `self.gui` is available. The action specified by [action_spec](#) is available as `self.qaction`.

`location_selected(loc)`[\[source\]](#)

Called whenever the book list being displayed in calibre changes. Currently values for `loc` are:

main,

card

and

cardb

.

This method should enable/disable this action and its sub actions as appropriate for the location.

`library_changed(db)`[\[source\]](#)

Called whenever the current library is changed.

Parameters **db** – The `LibraryDatabase` corresponding to the current library.

`gui_layout_complete()`[\[source\]](#)

Called once per action when the layout of the main GUI is completed. If your action needs to make changes to the layout, they should be done here, rather than in [initialization_complete\(\)](#).

`initialization_complete()`[\[source\]](#)

Called once per action when the initialization of the main GUI is completed.

`tag_browser_context_action(index)`[\[source\]](#)

Called when displaying the context menu in the Tag browser. *index* is the QModelIndex that points to the Tag browser item that was right clicked. Test it for validity with `index.valid()` and get the underlying TagTreeItem object with `index.data(Qt.ItemDataRole.UserRole)`. Any action objects yielded by this method will be added to the context menu.

`shutting_down()`[\[source\]](#)

Called once per plugin when the main GUI is in the process of shutting down. Release any used resources, but try not to block the shutdown for long periods of time.

`class calibre.customize.InterfaceActionBase(*args, **kwargs)`[\[source\]](#)

Bases: [calibre.customize.Plugin](#)

`load_actual_plugin(gui)`[\[source\]](#)

This method must return the actual interface action plugin object.

Preferences plugins

`class calibre.customize.PreferencesPlugin(plugin_path)`[\[source\]](#)

Bases: [calibre.customize.Plugin](#)

A plugin representing a widget displayed in the Preferences dialog.

This plugin has only one important method [create_widget\(\)](#). The various fields of the plugin control how it is categorized in the UI.

`config_widget = None`

Import path to module that contains a class named ConfigWidget which implements the ConfigWidgetInterface. Used by [create_widget\(\)](#).

`category_order = 100`

Where in the list of categories the [category](#) of this plugin should be.

`name_order = 100`

Where in the list of names in a category, the `gui_name` of this plugin should be

`category = None`

The category this plugin should be in

`gui_category = None`

The category name displayed to the user for this plugin

`gui_name = None`

The name displayed to the user for this plugin

`icon = None`

The icon for this plugin, should be an absolute path

`description = None`

The description used for tooltips and the like

`create_widget(parent=None)`[\[source\]](#)

Create and return the actual Qt widget used for setting this group of preferences. The widget must implement the `calibre.gui2.preferences.ConfigWidgetInterface`.

The default implementation uses `config_widget` to instantiate the widget.

`class calibre.gui2.preferences.ConfigWidgetInterface`[\[source\]](#)

This class defines the interface that all widgets displayed in the Preferences dialog must implement. See `ConfigWidgetBase`

for a base class that implements this interface and defines various convenience methods as well.

`changed_signal = None`

This signal must be emitted whenever the user changes a value in this widget

`supports_restoring_to_defaults = True`

Set `restart_critical` to `True` iff the `restore_to_defaults()` method is implemented.

`restore_defaults_desc = 'Restore settings to default values. You have to click Apply to actually save the default settings.'`

The tooltip for the “Restore defaults” button

`restart_critical = False`

If `True` the Preferences dialog will not allow the user to set any more preferences. Only has effect if `commit()` returns `True`.

`genesis(gui)`[\[source\]](#)

Called once before the widget is displayed, should perform any necessary setup.

Parameters `gui` – The main calibre graphical user interface

`initialize()`[\[source\]](#)

Should set all config values to their initial values (the values stored in the config files). A “return” statement is optional. Return `False` if the dialog is not to be shown.

`restore_defaults()`[\[source\]](#)

Should set all config values to their defaults.

`commit()`[\[source\]](#)

Save any changed settings. Return `True` if the changes require a restart, `False` otherwise. Raise `AbortCommit` an exception to indicate that an error occurred. You are responsible for giving the user feedback about what the error is and how to correct it.

`refresh_gui(gui)`[\[source\]](#)

Called once after this widget is committed. Responsible for causing the gui to reread any changed settings. Note that by default the GUI re-initializes various elements anyway, so most widgets won’t need to use this method.

`class calibre.gui2.preferences.ConfigWidgetBase(parent=None)`[\[source\]](#)

Base class that contains code to easily add standard config widgets like checkboxes, combo boxes, text fields and so on. See the `register()` method.

This class automatically handles change notification, resetting to default, translation between gui objects and config objects, etc. for registered settings.

If your config widget inherits from this class but includes setting that are not registered, you should override the [ConfigWidgetInterface](#) methods and call the base class methods inside the overrides.

`register(name, config_obj, gui_name=None, choices=None, restart_required=False, empty_string_is_None=True, setting=<class 'calibre.gui2.preferences.Setting'>)` [\[source\]](#)

Register a setting.

Parameters

- **name** – The setting name
- **config** – The config object that reads/writes the setting
- **gui_name** – The name of the GUI object that presents an interface to change the setting. By default it is assumed to be 'opt_'

+

name
.

• **choices** – If this setting is a multiple choice (combobox) based setting, the list of choices. The list is a list of two element tuples of the form: [(gui

name,

value),

...]
- **setting** – The class responsible for managing this setting. The default class handles almost all cases, so this param is rarely used.

`initialize()` [\[source\]](#)

Should set all config values to their initial values (the values stored in the config files). A “return” statement is optional. Return False if the dialog is not to be shown.

`commit(*args)`[\[source\]](#)

Save any changed settings. Return True if the changes require a restart, False otherwise.

Raise `AbortCommit` an

exception to indicate that an error occurred. You are responsible for giving the user feedback about what the error is and how to correct it.

`restore_defaults(*args)`[\[source\]](#)

Should set all config values to their defaults.

Command Line Interface

```
kovid giskard ~/work/libprs500/src/libprs500/manual $
```

Note

On macOS, the command line tools are inside the calibre bundle, for example, if you installed calibre in /Applications the command line tools are in /Applications/calibre.app/Contents/MacOS/.

Documented commands

- [calibre](#)
- [calibre-customize](#)
- [calibre-debug](#)
- [calibre-server](#)
- [calibre-smtp](#)
- [calibredb](#)
- [ebook-convert](#)
- [ebook-edit](#)
- [ebook-meta](#)

- `ebook-polish`
- `ebook-viewer`
- `fetch-ebook-metadata`
- `lrf2lrs`
- `lrfviewer`
- `lrs2lrf`
- `web2disk`

Undocumented commands

- `ebook-device`
- `markdown-calibre`

You can see usage for undocumented commands by executing them without arguments in a terminal.

calibre

```
calibre [options] [path_to_ebook or calibre url ...]
```

Launch the main **calibre** Graphical User Interface and optionally add the e-book at path_to_ebook to the database. You can also specify **calibre** URLs to perform various different actions, than just adding books. For example:

calibre://view-book/test_library/1842/epub

Will open the book with id 1842 in the EPUB format from the library “test_library” in the **calibre** E-book viewer. Library names are the folder names of the libraries with spaces replaced by underscores. A full description of the various URL based actions is in the User Manual.

Whenever you pass arguments to **calibre** that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

[options]

--detach

Detach from the controlling terminal, if any (Linux only)

--help, -h

show this help message and exit

--ignore-plugins

Ignore custom plugins, useful if you installed a plugin that is preventing calibre from starting

--no-update-check

Do not check for updates

--shutdown-running-calibre, -s

Cause a running calibre instance, if any, to be shutdown. Note that if there are running jobs, they will be silently aborted, so use with care.

--start-in-tray

Start minimized to system tray.

--verbose, -v

Ignored, do not use. Present only for legacy reasons

--version

show program'
s version number and exit

--with-library

Use the library located at the specified path.

calibre-customize

calibre-customize options

Customize calibre by loading external plugins.

Whenever you pass arguments to **calibre-customize** that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

[options]

--add-plugin, -a

Add a plugin by specifying the path to the ZIP file containing it.

--build-plugin, -b

For plugin developers: Path to the folder where you are developing the plugin. This command will automatically zip up the plugin and update it in calibre.

--customize-plugin

Customize plugin. Specify name of plugin and customization string separated by a comma.

--disable-plugin

Disable the named plugin

--enable-plugin

Enable the named plugin

--help, -h

show this help message and exit

--list-plugins, -l

List all installed plugins

--remove-plugin, -r

Remove a custom plugin by name. Has no effect on builtin plugins

--version

show program'
s version number and exit

calibre-debug

calibre-debug [options]

Various command line interfaces useful for debugging calibre. With no options, this command starts an embedded Python interpreter. You can also run the main calibre GUI, the calibre E-book viewer and the calibre editor in debug mode.

It also contains interfaces to various bits of calibre that do not have dedicated command line tools, such as font subsetting, the E-book diff tool and so on.

You can also use **calibre-debug** to run standalone scripts. To do that use it like this:

```
calibre-debug                                myscript.py                --  
--option1  
--option2  
file1 file2 ...
```

Everything after the --
is passed to the script.

Whenever you pass arguments to **calibre-debug** that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

[options]

--add-simple-plugin

Add a simple plugin (i.e. a plugin that consists of only a .py file), by specifying the path to the py file containing the plugin code.

--command, -c

Run Python code.

--debug-device-driver, -d

Debug device detection

--default-programs

(Un)register calibre from Windows Default Programs. **--default-programs**
(register|unregister)

--diff

Run the calibre diff tool. For example: calibre-debug **--diff**
file1 file2

--edit-book, -t

Launch the calibre "Edit book"
tool in debug mode.

--exec-file, -e

Run the Python code in file.

--explode-book, -x

Explode the book into the specified folder. Usage: -x file.epub output_dir Exports the book as a collection of HTML files and metadata, which you can edit using standard HTML editing tools. Works with EPUB, AZW3, HTMLZ and DOCX files.

--export-all-calibre-data

Export all calibre data (books/settings/plugins). Normally, you will be asked for the export folder and the libraries to export. You can also specify them as command line arguments to skip the questions. Use absolute paths for the export folder and libraries. The special keyword "all" can be used to export all libraries.

--fix-multiprocessing

For internal use

--gui, -g

Run the GUI with debugging enabled. Debug output is printed to stdout and stderr.

--gui-debug

Run the GUI with a debug console, logging to the specified path. For internal use only, use the -g option to run the GUI in debug mode

--help, -h

show this help message and exit

--implode-book, -i

Implode a previously exploded book. Usage: -i output_dir file.epub Imports the book from the files in output_dir which must have been created by a previous call to **--explode-book**. Be sure to specify the same file type as was used when exploding.

--import-calibre-data

Import previously exported calibre data

--inspect-mobi, -m

Inspect the MOBI file(s) at the specified path(s)

--paths

Output the paths necessary to setup the calibre environment

--reinitialize-db

Re-initialize the sqlite calibre database at the specified path. Useful to recover from db corruption.

--run-plugin, -r

Run a plugin that provides a command line interface. For example: calibre-debug -r "Add Books"

--

file1 --option1

Everything after the -- will be passed to the plugin as arguments.

--shutdown-running-calibre, -s

Cause a running calibre instance, if any, to be shutdown. Note that if there are running jobs, they will be silently aborted, so use with care.

--subset-font, -f

Subset the specified font. Use --
after this option to pass option to the font subsetting program.

--test-build

Test binary modules in build

--version

show program'
s version number and exit

--viewer, -w

Run the E-book viewer in debug mode

calibre-server

```
calibre-server [options] [path to library folder...]
```

Start the calibre Content server. The calibre Content server exposes your calibre libraries over the internet. You can specify the path to the library folders as arguments to **calibre-server**. If you do not specify any paths, all the libraries that the main calibre program knows about will be used.

Whenever you pass arguments to **calibre-server** that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

[options]

--access-log

Path to the access log file. This log contains information about clients connecting to the server and making requests. By default no access logging is done.

--ajax-timeout

Time (in seconds) to wait for a response from the server when making queries.

--auth-mode

Choose the type of authentication used. Set the HTTP authentication mode used by the server.

Set to "

basic"

if you are putting this server behind an SSL proxy. Otherwise, leave it as "

auto"

, which will use "

basic"

if SSL is configured otherwise it will use "

digest"

.

--auto-reload

Automatically reload server when source code changes. Useful for development. You should also specify a small value for the shutdown timeout.

`--ban-after`

Number of login failures for ban. The number of login failures after which an IP address is banned

`--ban-for`

Ban IP addresses that have repeated login failures. Temporarily bans access for IP addresses that have repeated login failures for the specified number of minutes. Useful to prevent attempts at guessing passwords. If set to zero, no banning is done.

`--book-list-mode`

Choose the default book list mode. Set the default book list mode that will be used for new users. Individual users can override the default in their own settings. The default is to use a cover grid.

`--compress-min-size`

Minimum size for which responses use data compression (in bytes).

`--custom-list-template`

Path to a JSON file containing a template for the custom book list mode. The easiest way to create such a template file is to go to Preferences-> Sharing over the net-> Book list template in calibre, create the template and export it.

`--daemonize`

Run process in background as a daemon (Linux only).

`--displayed-fields`

Restrict displayed user-defined fields. Comma separated list of user-defined metadata fields that will be displayed by the Content server in the /opds and /mobile views. If you specify this option, any fields not in this list will not be displayed. For example: my_rating,my_tags

`--enable-allow-socket-preallocation, --disable-allow-socket-preallocation`

Socket pre-allocation, for example, with systemd socket activation. By default, this option is enabled.

`--enable-auth, --disable-auth`

Password based authentication to access the server. Normally, the server is unrestricted, allowing anyone to access it. You can restrict access to predefined users with this option. By default, this option is disabled.

--enable-fallback-to-detected-interface, --disable-fallback-to-detected-interface

Fallback to auto-detected interface. If for some reason the server is unable to bind to the interface specified in the listen_on option, then it will try to detect an interface that connects to the outside world and bind to that. By default, this option is enabled.

--enable-local-write, --disable-local-write

Allow un-authenticated local connections to make changes. Normally, if you do not turn on authentication, the server operates in read-only mode, so as to not allow anonymous users to make changes to your calibre libraries. This option allows anybody connecting from the same computer as the server is running on to make changes. This is useful if you want to run the server without authentication but still use calibredb to make changes to your calibre libraries. Note that turning on this option means any program running on the computer can make changes to your calibre libraries. By default, this option is disabled.

--enable-log-not-found, --disable-log-not-found

Log HTTP 404 (Not Found) requests. Normally, the server logs all HTTP requests for resources that are not found. This can generate a lot of log spam, if your server is targeted by bots. Use this option to turn it off. By default, this option is enabled.

--enable-use-bonjour, --disable-use-bonjour

Advertise OPDS feeds via Bonjour. Advertise the OPDS feeds via the Bonjour service, so that OPDS based reading apps can detect and connect to the server automatically. By default, this option is enabled.

--enable-use-sendfile, --disable-use-sendfile

Zero copy file transfers for increased performance. This will use zero-copy in-kernel transfers when sending files over the network, increasing performance. However, it can cause corrupted file transfers on some broken filesystems. If you experience corrupted file transfers, turn it off. By default, this option is enabled.

--help, -h

show this help message and exit

--ignored-fields

Ignored user-defined metadata fields. Comma separated list of user-defined metadata fields that will not be displayed by the Content server in the /opds and /mobile views. For example: my_rating,my_tags

--listen-on

The interface on which to listen for connections. The default is to listen on all available IPv4 interfaces. You can change this to, for example, "127.0.0.1" to only listen for connections from the local machine, or to "::" to listen to all incoming IPv6 and IPv4 connections.

--log

Path to log file for server log. This log contains server information and errors, not access logs. By default it is written to stdout.

--manage-users

Manage the database of users allowed to connect to this server. You can use it in automated mode by adding a `-`. See `calibre-server --manage-users --help` for details. See also the `--userdb` option.

--max-header-line-size

Max. size of single HTTP header (in KB).

--max-job-time

Maximum time for worker processes. Maximum amount of time worker processes are allowed to run (in minutes). Set to zero for no limit.

--max-jobs

Maximum number of worker processes. Worker processes are launched as needed and used for large jobs such as preparing a book for viewing, adding books, converting, etc. Normally, the max. number of such processes is based on the number of CPU cores. You can control it by this setting.

--max-log-size

Max. log file size (in MB). The maximum size of log files, generated by the server. When the log becomes larger than this size, it is automatically rotated. Set to zero to disable log rotation.

`--max-opds-items`

Maximum number of books in OPDS feeds. The maximum number of books that the server will return in a single OPDS acquisition feed.

`--max-opds-ungrouped-items`

Maximum number of ungrouped items in OPDS feeds. Group items in categories such as author/tags by first letter when there are more than this number of items. Set to zero to disable.

`--max-request-body-size`

Max. allowed size for files uploaded to the server (in MB).

`--num-per-page`

Number of books to show in a single page. The number of books to show in a single page in the browser.

`--pidfile`

Write process PID to the specified file

`--port`

The port on which to listen for connections.

`--search-the-net-urls`

Path to a JSON file containing URLs for the "Search the internet" feature. The easiest way to create such a file is to go to Preferences-> Sharing over the net->Search the internet in calibre, create the URLs and export them.

`--shutdown-timeout`

Total time in seconds to wait for clean shutdown.

`--ssl-certfile`

Path to the SSL certificate file.

`--ssl-keyfile`

Path to the SSL private key file.

`--timeout`

Time (in seconds) after which an idle connection is closed.

`--trusted-ips`

Allow un-authenticated connections from specific IP addresses to make changes. Normally, if you do not turn on authentication, the server operates in read-only mode, so as to not allow anonymous users to make changes to your calibre libraries. This option allows anybody connecting from the specified IP addresses to make changes. Must be a comma separated list of address or network specifications. This is useful if you want to run the server without authentication but still use calibredb to make changes to your calibre libraries. Note that turning on this option means anyone connecting from the specified IP addresses can make changes to your calibre libraries.

`--url-prefix`

A prefix to prepend to all URLs. Useful if you wish to run this server behind a reverse proxy. For example use, /calibre as the URL prefix.

`--userdb`

Path to the user database to use for authentication. The database is a SQLite file. To create it use

`--manage-users`
. You can read more about managing users at: <https://manual.calibre-ebook.com/server.html#managing-user-accounts-from-the-command-line-only>

`--version`

show program'
s version number and exit

`--worker-count`

Number of worker threads used to process requests.

calibre-smtp

calibre-smtp [options] [from to text]

Send mail using the SMTP protocol. **calibre-smtp** has two modes of operation. In the compose mode you specify from to and text and these are used to build and send an email message. In the filter mode, **calibre-smtp** reads a complete email message from STDIN and sends it.

text is the body of the email message. If text is not specified, a complete email message is read from STDIN. from is the email address of the sender and to is the email address of the recipient. When a complete email is read from STDIN, from and to are only used in the SMTP negotiation, the message headers are not modified.

Whenever you pass arguments to **calibre-smtp** that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

[options]

--fork, -f

Fork and deliver message in background. If you use this option, you should also use **--outbox** to handle delivery failures.

--help, -h

show this help message and exit

--localhost, -l

Host name of localhost. Used when connecting to SMTP server.

--outbox, -o

Path to maildir folder to store failed email messages in.

--timeout, -t

Timeout for connection

--verbose, -v

Be more verbose

--version

show

s version number and exit

program'

COMPOSE MAIL

Options to compose an email. Ignored if text is not specified

--attachment, -a

File to attach to the email

--subject, -s

Subject of the email

SMTP RELAY

Options to use an SMTP relay server to send mail. calibre will try to send the email directly unless -relay is specified.

--cafile

Path to a file of concatenated CA certificates in PEM format, used to verify the server certificate when using TLS. By default, the system CA certificates are used.

--dont-verify-server-certificate

Do not verify the server certificate when connecting using TLS. This used to be the default behavior in calibre versions before 3.27. If you are using a relay with a self-signed or otherwise invalid certificate, you can use this option to restore the pre 3.27 behavior

--encryption-method, -e

Encryption method to use when connecting to relay. Choices are TLS, SSL and NONE. Default is TLS. WARNING: Choosing NONE is highly insecure

--password, -p

Password for relay

`--port`

Port to connect to on relay server. Default is to use 465 if encryption method is SSL and 25 otherwise.

`--relay, -r`

An SMTP relay server to use to send mail.

`--username, -u`

Username for relay

calibredb

calibredb command [options] [arguments]

calibredb is the command line interface to the calibre database. It has several sub-commands, documented below.

calibredb can be used to manipulate either a calibre database specified by path or a calibre *Content server* running either on the local machine or over the internet. You can start a calibre *Content server* using either the **calibre-server** program or in the main calibre program click *Connect/share → Start Content server*. Since **calibredb** can make changes to your calibre libraries, you must setup authentication on the server first. There are two ways to do that:

- If you plan to connect only to a server running on the same computer, you can simply use the `--enable-local-write` option of the Content server, to allow any program, including calibredb, running on the local computer to make changes to your calibre data. When running the server from the main calibre program, this option is in *Preferences → Sharing over the net → Advanced*.
- If you want to enable access over the internet, then you should setup user accounts on the server and use the `--username` and `--password` options to **calibredb** to give it access. You can setup user authentication for **calibre-server** by using the `--enable-auth` option and using `--manage-users` to create the user accounts. If you are running the server from the main calibre program, use *Preferences → Sharing over the net → Require username/password*.

To connect to a running Content server, pass the URL of the server to the `--with-library` option, see the documentation of that option for details and examples.

- [Global Options](#)
- [list](#)

- [add](#)
 - [Adding From Folders](#)
- [remove](#)
- [add format](#)
- [remove_format](#)
- [show metadata](#)
- [set metadata](#)
- [export](#)
- [catalog](#)
 - [Epub Options](#)
- [saved searches](#)
- [add custom column](#)
- [custom columns](#)
- [remove custom column](#)
- [set custom](#)
- [restore database](#)
- [check library](#)
- [list categories](#)
- [backup metadata](#)
- [clone](#)
- [embed_metadata](#)
- [search](#)

Global Options

--help, -h

show this help message and exit

--library-path, --with-library

Path to the calibre library. Default is to use the path stored in the settings. You can also connect to a calibre Content server to perform actions on remote libraries. To do so use a URL of the form: http://hostname:port/#library_id for example, <http://localhost:8080/#mylibrary>. library_id is the library id of the library you want to connect to on the Content server. You can use the special library_id value of - to get a list of library ids available on the server. For details on how to setup access via a Content server, see <https://manual.calibre-ebook.com/generated/en/calibredb.html>.

--password

Password for connecting to a calibre Content server. To read the password from standard input, use the special value: <stdin>. To read the password from a file, use: <f:/path/to/file> (i.e. <f: followed by the full path to the file and a trailing >). The angle brackets in the above are required, remember to escape them or use quotes for your shell.

--timeout

The timeout, in seconds, when connecting to a calibre library over the network. The default is two minutes.

--username

Username for connecting to a calibre Content server

--version

show
s version number and exit

program'

list

calibredb list [options]

List the books available in the calibre database.

Whenever you pass arguments to calibredb that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

--ascending

Sort results in ascending order

--fields, -f

The fields to display when listing books in the database. Should be a comma separated list of fields. Available fields: author_sort, authors, comments, cover, formats, identifiers, isbn, languages, last_modified, pubdate, publisher, rating, series, series_index, size, tags, timestamp, title, uuid Default: title,authors. The special field "all"

can be used to select all fields. In addition to the builtin fields above, custom fields are also available as *field_name, for example, for a custom field #rating, use the name: *rating

--for-machine

Generate output in JSON format, which is more suitable for machine parsing. Causes the line width and separator options to be ignored.

--limit

The maximum number of results to display. Default: all

--line-width, -w

The maximum width of a single line in the output. Defaults to detecting screen size.

--prefix

The prefix for all file paths. Default is the absolute path to the library folder.

--search, -s

Filter the results by the search query. For the format of the search query, please see the search related documentation in the User Manual. Default is to do no filtering.

--separator

The string used to separate fields. Default is a space.

--sort-by

The field by which to sort the results. Available fields: author_sort, authors, comments, cover, formats, identifiers, isbn, languages, last_modified, pubdate, publisher, rating, series, series_index, size, tags, timestamp, title, uuid Default: id

[**add**](#)


```
calibredb add [options] file1 file2 file3 ...
```

Add the specified files as books to the database. You can also specify folders, see the folder related options below.

Whenever you pass arguments to calibredb that have spaces in them, enclose the arguments in quotation marks. For example: `"/some path/with spaces"`

`--authors, -a`

Set the authors of the added book(s)

`--automerge, -m`

If books with similar titles and authors are found, merge the incoming formats (files) automatically into existing book records. A value of `"ignore"`

means duplicate formats are discarded. A value of `"overwrite"`

means duplicate formats in the library are overwritten with the newly added files. A value of `"new_record"`

means duplicate formats are placed into a new book record.

`--cover, -c`

Path to the cover to use for the added book

`--duplicates, -d`

Add books to database even if they already exist. Comparison is done based on book titles and authors. Note that the `--automerge` option takes precedence.

`--empty, -e`

Add an empty book (a book with no formats)

`--identifier, -I`

Set the identifiers for this book, e.g. `-I asin:XXX -I isbn:YYY`

`--isbn, -i`

Set the ISBN of the added book(s)

--languages, -l

A comma separated list of languages (best to use ISO639 language codes, though some language names may also be recognized)

--series, -s

Set the series of the added book(s)

--series-index, -S

Set the series number of the added book(s)

--tags, -T

Set the tags of the added book(s)

--title, -t

Set the title of the added book(s)

Adding From Folders

Options to control the adding of books from folders. By default only files that have extensions of known e-book file types are added.

--add

A filename (glob) pattern, files matching this pattern will be added when scanning folders for files, even if they are not of a known e-book file type. Can be specified multiple times for multiple patterns.

--ignore

A filename (glob) pattern, files matching this pattern will be ignored when scanning folders for files. Can be specified multiple times for multiple patterns. For example: *.pdf will ignore all PDF files

--one-book-per-directory, -1

Assume that each folder has only a single logical book and that all files in it are different e-book formats of that book

--recurse, -r

Process folders recursively

[remove](#)

```
calibredb remove ids
```

Remove the books identified by ids from the database. ids should be a comma separated list of id numbers (you can get id numbers by using the search command). For example, 23,34,57-85 (when specifying a range, the last number in the range is not included).

Whenever you pass arguments to calibredb that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

--permanent

Do not use the Recycle Bin

[add format](#)

```
calibredb add_format [options] id ebook_file
```

Add the e-book in ebook_file to the available formats for the logical book identified by id. You can get id by using the search command. If the format already exists, it is replaced, unless the do not replace option is specified.

Whenever you pass arguments to calibredb that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

--dont-replace

Do not replace the format if it already exists

[remove format](#)

```
calibredb remove_format [options] id fmt
```

Remove the format fmt from the logical book identified by id. You can get id by using the search command. fmt should be a file extension like LRF or TXT or EPUB. If the logical book does not have fmt available, do nothing.

Whenever you pass arguments to calibredb that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

show metadata

```
calibredb show_metadata [options] id
```

Show the metadata stored in the calibre database for the book identified by id. id is an id number from the search command.

Whenever you pass arguments to calibredb that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

--as-opf

Print metadata in OPF form (XML)

set metadata

```
calibredb set_metadata [options] id [/path/to/metadata.opf]
```

Set the metadata stored in the calibre database for the book identified by id from the OPF file metadata.opf. id is an id number from the search command. You can get a quick feel for the OPF format by using the --as-opf switch to the show_metadata command. You can also set the metadata of individual fields with the --field option. If you use the --field option, there is no need to specify an OPF file.

Whenever you pass arguments to calibredb that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

--field, -f

The field to set. Format is field_name:value, for example: **--field tags:tag1,tag2.** Use **--list-fields** to get a list of all field names. You can specify this option multiple times to set multiple fields. Note: For languages you must use the ISO639 language codes (e.g. en for English, fr for French and so on). For identifiers, the syntax is **--field identifiers:isbn:XXXX,doi:YYYYY.** For boolean (yes/no) fields use true and false or yes and no.

--list-fields, -l

List the metadata field names that can be used with the **--field** option

export

```
calibredb export [options] ids
```

Export the books specified by ids (a comma separated list) to the filesystem. The **export** operation saves all formats of the book, its cover and metadata (in an opf file). You can get id numbers from the search command.

Whenever you pass arguments to calibredb that have spaces in them, enclose the arguments in quotation marks. For example: `"/some path/with spaces"`

--all

Export all books in database, ignoring the list of ids.

--dont-asciiize

Have calibre convert all non English characters into English equivalents for the file names. This is useful if saving to a legacy filesystem without full support for Unicode filenames. Specifying this switch will turn this behavior off.

--dont-save-cover

Normally, calibre will save the cover in a separate file along with the actual e-book files. Specifying this switch will turn this behavior off.

--dont-update-metadata

Normally, calibre will update the metadata in the saved files from what is in the calibre library. Makes saving to disk slower. Specifying this switch will turn this behavior off.

--dont-write-opf

Normally, calibre will write the metadata into a separate OPF file along with the actual e-book files. Specifying this switch will turn this behavior off.

--formats

Comma separated list of formats to save for each book. By default all available formats are saved.

--progress

Report progress

--replace-whitespace

Replace whitespace with underscores.

--single-dir

Export all books into a single folder

--template

The template to control the filename and folder structure of the saved files. Default is "{author_sort}/{title}/{title} - {authors}" which will save books into a per-author subfolder with filenames containing title and author. Available controls are: {author_sort, authors, id, isbn, languages, last_modified, pubdate, publisher, rating, series, series_index, tags, timestamp, title}

--timefmt

The format in which to display dates. %d - day, %b - month, %m - month number, %Y - year. Default is: %b, %Y

--to-dir

Export books to the specified folder. Default is .

--to-lowercase

Convert paths to lowercase.

[catalog](#)

```
calibredb catalog /path/to/destination.(csv|epub|mobi|xml...) [options]
```

Export a **catalog** in format specified by path/to/destination extension. Options control how entries are displayed in the generated **catalog** output. Note that different **catalog** formats support different sets of options. To see the different options, specify the name of the output file and then the -help option.

Whenever you pass arguments to calibredb that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

--ids, -i

Comma-separated list of database IDs to catalog. If declared, [--search](#) is ignored. Default: all

--search, -s

Filter the results by the search query. For the format of the search query, please see the search-related documentation in the User Manual. Default: no filtering

--verbose, -v

Show detailed output information. Useful for debugging

[Epub Options](#)

--catalog-title

Title of generated catalog used as title in metadata. Default: 'My Books'
Applies to: AZW3, EPUB, MOBI output formats

--cross-reference-authors

Create cross-references in Authors section for books with multiple authors. Default: 'False'
Applies to: AZW3, EPUB, MOBI output formats

--debug-pipeline

Save the output from different stages of the conversion pipeline to the specified folder. Useful if you are unsure at which stage of the conversion process a bug is occurring. Default: 'None'
Applies to: AZW3, EPUB, MOBI output formats

--exclude-genre

Regex describing tags to exclude as genres. Default: '[.+]|^+\$'
excludes bracketed tags, e.g. '[Project Gutenberg]'
, and
+',
, the default tag for read books. Applies to: AZW3, EPUB, MOBI output formats

--exclusion-rules

Specifies the rules used to exclude books from the generated catalog. The model for an

exclusion rule is either ('

<rule
,

Tags'
,

<comma-separated list of tags>
) or ('

<rule
,

<custom column>
,

<pattern>
)

For example: (('

Archived
,

#status'
,

Archived'
,) will exclude a book with a value of '

Archived'
in the custom column '

status'
. When multiple rules are defined, all rules will be applied. Default: "

('

Catalogs'
,

Tags'
,

Catalog'
,)"

Applies to: AZW3, EPUB, MOBI output formats

--generate-authors

Include
Authors'

section in catalog. Default: '

False'
Applies to: AZW3, EPUB, MOBI output formats

--generate-descriptions

--header-note-source-field

Custom field containing note text to insert in Description header. Default: ''

Applies to: AZW3, EPUB, MOBI output formats

--merge-comments-rule

#<custom field>:[before|after]:[True|False] specifying: <custom field> Custom field containing notes to merge with comments [before|after] Placement of notes with respect to comments [True|False] - A horizontal rule is inserted between notes and comments Default: ''

::'

Applies to: AZW3, EPUB, MOBI output formats

--output-profile

Specifies the output profile. In some cases, an output profile is required to optimize the catalog for the device. For example, 'kindle'

or

kindle_dx'

creates a structured Table of Contents with Sections and Articles. Default: 'None'

Applies to: AZW3, EPUB, MOBI output formats

--prefix-rules

Specifies the rules used to include prefixes indicating read books, wishlist items and other user-specified prefixes. The model for a prefix rule is ('<rule name>'

,

<source field>'

,

<pattern>'

,

<prefix>'

). When multiple rules are defined, the first matching rule will be used. Default: ''

((

Read books'

,

tags'

,

+

,

```

✓'
),('
Wishlist
item'
,
tags'
,
Wishlist'
,
x'
))"

```

Applies to: AZW3, EPUB, MOBI output formats

--preset

Use a named preset created with the GUI catalog builder. A preset specifies all settings for building a catalog. Default: 'None'

Applies to: AZW3, EPUB, MOBI output formats

--thumb-width

Size hint (in inches) for book covers in catalog. Range: 1.0 - 2.0 Default: '1.0'

Applies to: AZW3, EPUB, MOBI output formats

--use-existing-cover

Replace existing cover when generating the catalog. Default: 'False'

Applies to: AZW3, EPUB, MOBI output formats

[saved searches](#)

```
calibredb saved_searches [options] (list|add|remove)
```

Manage the saved searches stored in this database. If you try to add a query with a name that already exists, it will be replaced.

Syntax for adding:

```
calibredb saved_searches add search_name search_expression
```

Syntax for removing:

calibredb **saved_searches** remove search_name

Whenever you pass arguments to calibredb that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

[add custom column](#)

```
calibredb add_custom_column [options] label name datatype
```

Create a custom column. label is the machine friendly name of the column. Should not contain spaces or colons. name is the human friendly name of the column. datatype is one of: bool, comments, composite, datetime, enumeration, float, int, rating, series, text

Whenever you pass arguments to calibredb that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

--display

A dictionary of options to customize how the data in this column will be interpreted. This is a JSON string. For enumeration columns, use **--display**

```
{\
enum_values\
:[\
val1\
,\
val2\
]}\
"
```

There are many options that can go into the display variable. The options by column type are: composite: composite_template, composite_sort, make_category, contains_html, use_decorations datetime: date_format enumeration: enum_values, enum_colors, use_decorations int, float: number_format text: is_names, use_decorations The best way to find legal combinations is to create a custom column of the appropriate type in the GUI then look at the backup OPF for a book (ensure that a new OPF has been created since the column was added). You will see the JSON for the "display"

for the new column in the OPF.

--is-multiple

This column stores tag like data (i.e. multiple comma separated values). Only applies if datatype is text.

custom columns

```
calibredb custom_columns [options]
```

List available custom columns. Shows column labels and ids.

Whenever you pass arguments to calibredb that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

--details, -d

Show details for each column.

remove custom column

```
calibredb remove_custom_column [options] label
```

Remove the custom column identified by label. You can see available columns with the custom_columns command.

Whenever you pass arguments to calibredb that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

--force, -f

Do not ask for confirmation

set custom

```
calibredb set_custom [options] column id value
```

Set the value of a custom column for the book identified by id. You can get a list of ids using the search command. You can get a list of custom column names using the custom_columns command.

Whenever you pass arguments to calibredb that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

--append, -a

If the column stores multiple values, append the specified values to the existing ones, instead of replacing them.

restore database

```
calibredb restore_database [options]
```

Restore this database from the metadata stored in OPF files in each folder of the calibre library. This is useful if your metadata.db file has been corrupted.

WARNING: This command completely regenerates your database. You will lose all saved searches, user categories, plugboards, stored per-book conversion settings, and custom recipes. Restored metadata will only be as accurate as what is found in the OPF files.

Whenever you pass arguments to calibredb that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

--really-do-it, -r

Really do the recovery. The command will not run unless this option is specified.

check library

```
calibredb check_library [options]
```

Perform some checks on the filesystem representing a library. Reports are invalid_titles, extra_titles, invalid_authors, extra_authors, missing_formats, extra_formats, extra_files, missing_covers, extra_covers, failed_folders

Whenever you pass arguments to calibredb that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

--csv, -c

Output in CSV

--ignore_extensions, -e

Comma-separated list of extensions to ignore. Default: all

--ignore_names, -n

Comma-separated list of names to ignore. Default: all

--report, -r

Comma-separated list of reports. Default: all

list categories

```
calibredb list_categories [options]
```

Produce a report of the category information in the database. The information is the equivalent of what is shown in the Tag browser.

Whenever you pass arguments to calibredb that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

--categories, -r

Comma-separated list of category lookup names. Default: all

--csv, -c

Output in CSV

--dialect

The type of CSV file to produce. Choices: excel, excel-tab, unix

--item_count, -i

Output only the number of items in a category instead of the counts per item within the category

--width, -w

The maximum width of a single line in the output. Defaults to detecting screen size.

backup metadata

```
calibredb backup_metadata [options]
```

Backup the metadata stored in the database into individual OPF files in each books folder. This normally happens automatically, but you can run this command to force re-generation of the OPF files, with the `--all` option.

Note that there is normally no need to do this, as the OPF files are backed up automatically, every time metadata is changed.

Whenever you pass arguments to `calibredb` that have spaces in them, enclose the arguments in quotation marks. For example: `"/some path/with spaces"`

`--all`

Normally, this command only operates on books that have out of date OPF files. This option makes it operate on all books.

[clone](#)

```
calibredb clone path/to/new/library
```

Create a **clone** of the current library. This creates a new, empty library that has all the same custom columns, Virtual libraries and other settings as the current library.

The cloned library will contain no books. If you want to create a full duplicate, including all books, then simply use your filesystem tools to copy the library folder.

Whenever you pass arguments to `calibredb` that have spaces in them, enclose the arguments in quotation marks. For example: `"/some path/with spaces"`

[embed metadata](#)

```
calibredb embed_metadata [options] book_id
```

Update the metadata in the actual book files stored in the calibre library from the metadata in the calibre database. Normally, metadata is updated only when exporting files from calibre, this command is useful if you want the files to be updated in place. Note that different file formats support different amounts of metadata. You can use the special value 'all' for `book_id` to update metadata in all books. You can also specify many book ids separated by

spaces and id ranges separated by hyphens. For example: `calibredb embed_metadata 1 2 10-15 23`

Whenever you pass arguments to `calibredb` that have spaces in them, enclose the arguments in quotation marks. For example: `"/some path/with spaces"`

`--only-formats, -f`

Only update metadata in files of the specified format. Specify it multiple times for multiple formats. By default, all formats are updated.

[search](#)

```
calibredb search [options] search expression
```

Search the library for the specified **search** term, returning a comma separated list of book ids matching the **search** expression. The output format is useful to feed into other commands that accept a list of ids as input.

The **search** expression can be anything from calibre's powerful **search** query language, for example: `calibredb search author:asimov 'title:"i robot"'`

Whenever you pass arguments to `calibredb` that have spaces in them, enclose the arguments in quotation marks. For example: `"/some path/with spaces"`

`--limit, -l`

The maximum number of results to return. Default is all results.

ebook-convert

```
ebook-convert input_file output_file [options]
```

Convert an e-book from one format to another.

input_file is the input and output_file is the output. Both must be specified as the first two arguments to the command.

The output e-book format is guessed from the file extension of output_file. output_file can also be of the special format .EXT where EXT is the output file extension. In this case, the name of the output file is derived from the name of the input file. Note that the filenames must not start with a hyphen. Finally, if output_file has no extension, then it is treated as a folder and an “open e-book” (OEB) consisting of HTML files is written to that folder. These files are the files that would normally have been passed to the output plugin.

After specifying the input and output file you can customize the conversion by specifying various options. The available options depend on the input and output file types. To get help on them specify the input and output file and then use the -h option.

For full documentation of the conversion system see [E-book conversion](#)

Whenever you pass arguments to **ebook-convert** that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

The options and default values for the options change depending on both the input and output formats, so you should always check with:

```
ebook
-
convert

myfile
.
input_format

myfile
```

```
.  
output_format
```

```
-
```

```
h
```

Below are the options that are common to all conversion, followed by the options specific to every input and output format.

- [Look and Feel](#)
- [Heuristic Processing](#)
- [Search and Replace](#)
- [Structure Detection](#)
- [Table of Contents](#)
- [Metadata](#)
- [Debug](#)
- [AZW4 Input Options](#)
- [CHM Input Options](#)
- [Comic Input Options](#)
- [DJVU Input Options](#)
- [DOCX Input Options](#)
- [EPUB Input Options](#)
- [FB2 Input Options](#)
- [HTLZ Input Options](#)
- [HTML Input Options](#)
- [LIT Input Options](#)
- [LRF Input Options](#)

- [MOBI Input Options](#)
- [ODT Input Options](#)
- [PDB Input Options](#)
- [PDF Input Options](#)
- [PML Input Options](#)
- [RB Input Options](#)
- [RTF Input Options](#)
- [Recipe Input Options](#)
- [SNB Input Options](#)
- [TCR Input Options](#)
- [TXT Input Options](#)
- [AZW3 Output Options](#)
- [DOCX Output Options](#)
- [EPUB Output Options](#)
- [FB2 Output Options](#)
- [HTML Output Options](#)
- [HTMLZ Output Options](#)
- [LIT Output Options](#)
- [LRF Output Options](#)
- [MOBI Output Options](#)
- [OEB Output Options](#)
- [PDB Output Options](#)
- [PDF Output Options](#)
- [PML Output Options](#)

- [RB Output Options](#)
- [RTF Output Options](#)
- [SNB Output Options](#)
- [TCR Output Options](#)
- [TXT Output Options](#)
- [TXTZ Output Options](#)

--help, -h

show this help message and exit

--input-profile

Specify the input profile. The input profile gives the conversion system information on how to interpret various information in the input document. For example resolution dependent lengths (i.e. lengths in pixels). Choices are: cybookg3, cybook_opus, default, hanlinv3, hanlinv5, illiad, irexdr1000, irexdr800, kindle, msreader, mobipocket, nook, sony, sony300, sony900

--list-recipes

List builtin recipe names. You can create an e-book from a builtin recipe like this: ebook-convert "Recipe" Name.recipe" output.epub

--output-profile

Specify the output profile. The output profile tells the conversion system how to optimize the created document for the specified device (such as by resizing images for the device screen size). In some cases, an output profile can be used to optimize the output for a particular device, but this is rarely necessary. Choices are: cybookg3, cybook_opus, default, generic_eink, generic_eink_hd, generic_eink_large, hanlinv3, hanlinv5, illiad, ipad, ipad3, irexdr1000, irexdr800, jetbook5, kindle, kindle_dx, kindle_fire, kindle_oasis, kindle_pw, kindle_pw3, kindle_voyage, kobo, msreader, mobipocket, nook, nook_color, nook_hd_plus, pocketbook_inkpad3, pocketbook_lux, pocketbook_hd, pocketbook_900, pocketbook_pro_912, galaxy, sony, sony300, sony900, sony-landscape, sonyt3, tablet

--version

show program's version number and exit

Look and Feel

Options to control the look and feel of the output

--asciize

Transliterate Unicode characters to an ASCII representation. Use with care because this will replace Unicode characters with ASCII. For instance it will replace "Pelé" with "Pele".

. Also, note that in cases where there are multiple representations of a character (characters shared by Chinese and Japanese for instance) the representation based on the current calibre interface language will be used.

--base-font-size

The base font size in pts. All font sizes in the produced book will be rescaled based on this size. By choosing a larger size you can make the fonts in the output bigger and vice versa. By default, when the value is zero, the base font size is chosen based on the output profile you chose.

--change-justification

Change text justification. A value of "left"

converts all justified text in the source to left aligned (i.e. unjustified) text. A value of "justify"

converts all unjustified text to justified. A value of "original"

(the default) does not change justification in the source file. Note that only some output formats support justification.

--disable-font-rescaling

Disable all rescaling of font sizes.

--embed-all-fonts

Embed every font that is referenced in the input document but not already embedded. This will search your system for the fonts, and if found, they will be embedded. Embedding will only work if the format you are converting to supports embedded fonts, such as EPUB, AZW3, DOCX or PDF. Please ensure that you have the proper license for embedding the fonts used in this document.

--embed-font-family

Embed the specified font family into the book. This specifies the "base"

font used for the book. If the input document specifies its own fonts, they may override this base font. You can use the filter style information option to remove fonts from the input document. Note that font embedding only works with some output formats, principally EPUB, AZW3 and DOCX.

--expand-css

By default, calibre will use the shorthand form for various CSS properties such as margin, padding, border, etc. This option will cause it to use the full expanded form instead. Note that CSS is always expanded when generating EPUB files with the output profile set to one of the Nook profiles as the Nook cannot handle shorthand CSS.

--extra-css

Either the path to a CSS stylesheet or raw CSS. This CSS will be appended to the style rules from the source file, so it can be used to override those rules.

--filter-css

A comma separated list of CSS properties that will be removed from all CSS style rules. This is useful if the presence of some style information prevents it from being overridden on your device. For example: font-family,color,margin-left,margin-right

--font-size-mapping

Mapping from CSS font names to font sizes in pts. An example setting is 12,12,14,16,18,20,22,24. These are the mappings for the sizes xx-small to xx-large, with the final size being for huge fonts. The font rescaling algorithm uses these sizes to intelligently rescale fonts. The default is to use a mapping based on the output profile you chose.

--insert-blank-line

Insert a blank line between paragraphs. Will not work if the source file does not use paragraphs (<p> or <div> tags).

--insert-blank-line-size

Set the height of the inserted blank lines (in em). The height of the lines between paragraphs will be twice the value set here.

--keep-ligatures

Preserve ligatures present in the input document. A ligature is a special rendering of a pair of characters like ff, fi, fl et cetera. Most readers do not have support for ligatures in their

default fonts, so they are unlikely to render correctly. By default, calibre will turn a ligature into the corresponding pair of normal characters. This option will preserve them instead.

`--line-height`

The line height in pts. Controls spacing between consecutive lines of text. Only applies to elements that do not define their own line height. In most cases, the minimum line height option is more useful. By default no line height manipulation is performed.

`--linearize-tables`

Some badly designed documents use tables to control the layout of text on the page. When converted these documents often have text that runs off the page and other artifacts. This option will extract the content from the tables and present it in a linear fashion.

`--margin-bottom`

Set the bottom margin in pts. Default is 5.0. Setting this to less than zero will cause no margin to be set (the margin setting in the original document will be preserved). Note: Page oriented formats such as PDF and DOCX have their own margin settings that take precedence.

`--margin-left`

Set the left margin in pts. Default is 5.0. Setting this to less than zero will cause no margin to be set (the margin setting in the original document will be preserved). Note: Page oriented formats such as PDF and DOCX have their own margin settings that take precedence.

`--margin-right`

Set the right margin in pts. Default is 5.0. Setting this to less than zero will cause no margin to be set (the margin setting in the original document will be preserved). Note: Page oriented formats such as PDF and DOCX have their own margin settings that take precedence.

`--margin-top`

Set the top margin in pts. Default is 5.0. Setting this to less than zero will cause no margin to be set (the margin setting in the original document will be preserved). Note: Page oriented formats such as PDF and DOCX have their own margin settings that take precedence.

`--minimum-line-height`

The minimum line height, as a percentage of the element's calculated font size. calibre will ensure that every element has a line height of at least this setting, irrespective of what the input document specifies. Set to zero to disable. Default is 120%. Use this setting in preference to the direct line height specification, unless you know what you are doing. For example, you can achieve "

doublespaced"
text by setting this to 240.

--remove-paragraph-spacing

Remove spacing between paragraphs. Also sets an indent on paragraphs of 1.5em. Spacing removal will not work if the source file does not use paragraphs (<p> or <div> tags).

--remove-paragraph-spacing-indent-size

When calibre removes blank lines between paragraphs, it automatically sets a paragraph indent, to ensure that paragraphs can be easily distinguished. This option controls the width of that indent (in em). If you set this value negative, then the indent specified in the input document is used, that is, calibre does not change the indentation.

--smarten-punctuation

Convert plain quotes, dashes and ellipsis to their typographically correct equivalents. For details, see <https://daringfireball.net/projects/smarty-pants>.

--subset-embedded-fonts

Subset all embedded fonts. Every embedded font is reduced to contain only the glyphs used in this document. This decreases the size of the font files. Useful if you are embedding a particularly large font with lots of unused glyphs.

--transform-css-rules

Path to a file containing rules to transform the CSS styles in this book. The easiest way to create such a file is to use the wizard for creating rules in the calibre GUI. Access it in the "Look & feel->Transform styles" section of the conversion dialog. Once you create the rules, you can use the "Export" button to save them to a file.

--transform-html-rules

Path to a file containing rules to transform the HTML in this book. The easiest way to create such a file is to use the wizard for creating rules in the calibre GUI. Access it in the "Look & feel->Transform HTML" section of the conversion dialog. Once you create the rules, you can use the "Export" button to save them to a file.

--unsmarten-punctuation

Convert fancy quotes, dashes and ellipsis to their plain equivalents.

Heuristic Processing

Modify the document text and structure using common patterns. Disabled by default. Use `--enable-heuristics` to enable. Individual actions can be disabled with the `--disable-*` options.

`--disable-dehyphenate`

Analyze hyphenated words throughout the document. The document itself is used as a dictionary to determine whether hyphens should be retained or removed.

`--disable-delete-blank-paragraphs`

Remove empty paragraphs from the document when they exist between every other paragraph

`--disable-fix-indents`

Turn indentation created from multiple non-breaking space entities into CSS indents.

`--disable-format-scene-breaks`

Left aligned scene break markers are center aligned. Replace soft scene breaks that use multiple blank lines with horizontal rules.

`--disable-italicize-common-cases`

Look for common words and patterns that denote italics and italicize them.

`--disable-markup-chapter-headings`

Detect unformatted chapter headings and sub headings. Change them to h2 and h3 tags. This setting will not create a TOC, but can be used in conjunction with structure detection to create one.

`--disable-renumber-headings`

Looks for occurrences of sequential `<h1>` or `<h2>` tags. The tags are renumbered to prevent splitting in the middle of chapter headings.

`--disable-unwrap-lines`

Unwrap lines using punctuation and other formatting clues.

`--enable-heuristics`

Enable heuristic processing. This option must be set for any heuristic processing to take place.

--html-unwrap-factor

Scale used to determine the length at which a line should be unwrapped. Valid values are a decimal between 0 and 1. The default is 0.4, just below the median line length. If only a few lines in the document require unwrapping this value should be reduced

--replace-scene-breaks

Replace scene breaks with the specified text. By default, the text from the input document is used.

Search and Replace

Modify the document text and structure using user defined patterns.

--search-replace

Path to a file containing search and replace regular expressions. The file must contain alternating lines of regular expression followed by replacement pattern (which can be an empty line). The regular expression must be in the Python regex syntax and the file must be UTF-8 encoded.

--sr1-replace

Replacement to replace the text found with sr1-search.

--sr1-search

Search pattern (regular expression) to be replaced with sr1-replace.

--sr2-replace

Replacement to replace the text found with sr2-search.

--sr2-search

Search pattern (regular expression) to be replaced with sr2-replace.

--sr3-replace

Replacement to replace the text found with sr3-search.

--sr3-search

Search pattern (regular expression) to be replaced with sr3-replace.

Structure Detection

Control auto-detection of document structure.

--chapter

An XPath expression to detect chapter titles. The default is to consider <h1> or <h2> tags that contain the words "chapter", "book", "section", "prologue", "epilogue", or "part" as chapter titles as well as any tags that have class="chapter". The expression used must evaluate to a list of elements. To disable chapter detection, use the expression "/"

. See the XPath Tutorial in the calibre User Manual for further help on using this feature.

--chapter-mark

Specify how to mark detected chapters. A value of "pagebreak" will insert page breaks before chapters. A value of "rule" will insert a line before chapters. A value of "none" will disable chapter marking and a value of "both" will use both page breaks and lines to mark chapters.

--disable-remove-fake-margins

Some documents specify page margins by specifying a left and right margin on each individual paragraph. calibre will try to detect and remove these margins. Sometimes, this can cause the removal of margins that should not have been removed. In this case you can disable the removal.

--insert-metadata

Insert the book metadata at the start of the book. This is useful if your e-book reader does not support displaying/searching metadata directly.

--page-breaks-before

An XPath expression. Page breaks are inserted before the specified elements. To disable use the expression: /

--prefer-metadata-cover

Use the cover detected from the source file in preference to the specified cover.

--remove-first-image

Remove the first image from the input e-book. Useful if the input document has a cover image that is not identified as a cover. In this case, if you set a cover in calibre, the output document will end up with two cover images if you do not specify this option.

--start-reading-at

An XPath expression to detect the location in the document at which to start reading. Some e-book reading programs (most prominently the Kindle) use this location as the position at which to open the book. See the XPath tutorial in the calibre User Manual for further help using this feature.

Table of Contents

Control the automatic generation of a Table of Contents. By default, if the source file has a Table of Contents, it will be used in preference to the automatically generated one.

--duplicate-links-in-toc

When creating a TOC from links in the input document, allow duplicate entries, i.e. allow more than one entry with the same text, provided that they point to a different location.

--level1-toc

XPath expression that specifies all tags that should be added to the Table of Contents at level one. If this is specified, it takes precedence over other forms of auto-detection. See the XPath Tutorial in the calibre User Manual for examples.

--level2-toc

XPath expression that specifies all tags that should be added to the Table of Contents at level two. Each entry is added under the previous level one entry. See the XPath Tutorial in the calibre User Manual for examples.

--level3-toc

XPath expression that specifies all tags that should be added to the Table of Contents at level three. Each entry is added under the previous level two entry. See the XPath Tutorial in the calibre User Manual for examples.

--max-toc-links

Maximum number of links to insert into the TOC. Set to 0 to disable. Default is: 50. Links are only added to the TOC if less than the threshold number of chapters were detected.

--no-chapters-in-toc

Don't add auto-detected chapters to the Table of Contents.

--toc-filter

Remove entries from the Table of Contents whose titles match the specified regular expression. Matching entries and all their children are removed.

--toc-threshold

If fewer than this number of chapters is detected, then links are added to the Table of Contents. Default: 6

--use-auto-toc

Normally, if the source file already has a Table of Contents, it is used in preference to the auto-generated one. With this option, the auto-generated one is always used.

Metadata

Options to set metadata in the output

--author-sort

String to be used when sorting by author.

--authors

Set the authors. Multiple authors should be separated by ampersands.

--book-producer

Set the book producer.

--comments

Set the e-book description.

--cover

Set the cover to the specified file or URL

--isbn

Set the ISBN of the book.

--language

Set the language.

--pubdate

Set the publication date (assumed to be in the local timezone, unless the timezone is explicitly specified)

--publisher

Set the e-book publisher.

--rating

Set the rating. Should be a number between 1 and 5.

--read-metadata-from-opf, --from-opf, -m

Read metadata from the specified OPF file. Metadata read from this file will override any metadata in the source file.

--series

Set the series this e-book belongs to.

--series-index

Set the index of the book in this series.

--tags

Set the tags for the book. Should be a comma separated list.

--timestamp

Set the book timestamp (no longer used anywhere)

--title

Set the title.

--title-sort

The version of the title to be used for sorting.

Debug

Options to help with debugging the conversion

--debug-pipeline, -d

Save the output from different stages of the conversion pipeline to the specified folder. Useful if you are unsure at which stage of the conversion process a bug is occurring.

--verbose, -v

Level of verbosity. Specify multiple times for greater verbosity. Specifying it twice will result in full verbosity, once medium verbosity and zero times least verbosity.

AZW4 Input Options

--input-encoding

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

CHM Input Options

--input-encoding

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

Comic Input Options

--colors

Reduce the number of colors used in the image. This works only if you choose the PNG output format. It is useful to reduce file sizes. Set to zero to turn off. Maximum value is 256. It is off by default.

`--comic-image-size`

Specify the image size as width x height pixels, for example: 123x321. Normally, an image size is automatically calculated from the output profile, this option overrides it.

`--despeckle`

Enable Despeckle. Reduces speckle noise. May greatly increase processing time.

`--disable-trim`

Disable trimming of comic pages. For some comics, trimming might remove content as well as borders.

`--dont-add-comic-pages-to-toc`

When converting a CBC do not add links to each page to the TOC. Note this only applies if the TOC has more than one section

`--dont-grayscale`

Do not convert the image to grayscale (black and white)

`--dont-normalize`

Disable normalize (improve contrast) color range for pictures. Default: False

`--dont-sharpen`

Disable sharpening.

`--input-encoding`

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

`--keep-aspect-ratio`

Maintain picture aspect ratio. Default is to fill the screen.

`--landscape`

Don't split landscape images into two portrait images

--no-process

Apply no processing to the image

--no-sort

Don't sort the files found in the comic alphabetically by name. Instead use the order they were added to the comic.

--output-format

The format that images in the created e-book are converted to. You can experiment to see which format gives you optimal size and look on your device.

--right2left

Used for right-to-left publications like manga. Causes landscape pages to be split into portrait pages from right to left.

--wide

Keep aspect ratio and scale image using screen height as image width for viewing in landscape mode.

DJVU Input Options

--input-encoding

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

DOCX Input Options

--docx-inline-subsup

Render superscripts and subscripts so that they do not affect the line height

--docx-no-cover

Normally, if a large image is present at the start of the document that looks like a cover, it will be removed from the document and used as the cover for created e-book. This option turns off that behavior.

`--docx-no-pagebreaks-between-notes`

Do not insert a page break after every endnote.

`--input-encoding`

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

EPUB Input Options

`--input-encoding`

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

FB2 Input Options

`--input-encoding`

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

`--no-inline-fb2-toc`

Do not insert a Table of Contents at the beginning of the book

HTLZ Input Options

`--input-encoding`

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

HTML Input Options

`--breadth-first`

Traverse links in HTML files breadth first. Normally, they are traversed depth first.

`--dont-package`

Normally this input plugin re-arranges all the input files into a standard folder hierarchy. Only use this option if you know what you are doing as it can result in various nasty side effects in the rest of the conversion pipeline.

`--input-encoding`

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

`--max-levels`

Maximum levels of recursion when following links in HTML files. Must be non-negative. 0 implies that no links in the root HTML file are followed. Default is 5.

LIT Input Options

`--input-encoding`

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

LRF Input Options

`--input-encoding`

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

MOBI Input Options

`--input-encoding`

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

ODT Input Options

`--input-encoding`

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

PDB Input Options

--input-encoding

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

PDF Input Options

--input-encoding

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

--new-pdf-engine

Use the new PDF conversion engine. Currently not operational.

--no-images

Do not extract images from the document

--unwrap-factor

Scale used to determine the length at which a line should be unwrapped. Valid values are a decimal between 0 and 1. The default is 0.45, just below the median line length.

PML Input Options

--input-encoding

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

RB Input Options

--input-encoding

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

RTF Input Options

`--ignore-wmf`

Ignore WMF images instead of replacing them with a placeholder image.

`--input-encoding`

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

Recipe Input Options

`--dont-download-recipe`

Do not download latest version of builtin recipes from the calibre server

`--input-encoding`

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

`--lrf`

Optimize fetching for subsequent conversion to LRF.

`--password`

Password for sites that require a login to access content.

`--test`

Useful for recipe development. Forces `max_articles_per_feed` to 2 and downloads at most 2 feeds. You can change the number of feeds and articles by supplying optional arguments. For example:

3 1 will download at most 3 feeds and only 1 article per feed.

`--test`

`--username`

Username for sites that require a login to access content.

SNB Input Options

`--input-encoding`

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

TCR Input Options

--input-encoding

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

TXT Input Options

--formatting-type

Formatting used within the document. * auto: Automatically decide which formatting processor to use * plain: No formatting * heuristic: Use heuristics to determine chapter headings, italics, etc. * textile: Use the Textile markup language * markdown: Use the Markdown markup language To learn more about markdown see <https://daringfireball.net/projects/markdown/>

--input-encoding

Specify the character encoding of the input document. If set this option will override any encoding declared by the document itself. Particularly useful for documents that do not declare an encoding or that have erroneous encoding declarations.

--markdown-extensions

Enable extensions to Markdown syntax. Extensions are formatting that is not part of the standard Markdown format. The extensions enabled by default: footnotes, tables, toc. To learn more about Markdown extensions, see <https://python-markdown.github.io/extensions/> This should be a comma separated list of extensions to enable: * abbr: Abbreviations * admonition: Support admonitions * attr_list: Add attribute to HTML tags * codehilite: Add code highlighting via Pygments * def_list: Definition lists * extra: Enables various common extensions * fenced_code: Alternative code block syntax * footnotes: Footnotes * legacy_attrs: Use legacy element attributes * legacy_em: Use legacy underscore handling for connected words * meta: Metadata in the document * nl2br: Treat newlines as hard breaks * sane_lists: Do not allow mixing list types * smarty: Use markdown's internal smartypants parser * tables: Support tables * toc: Generate a table of contents * wikilinks: Wiki style links

--paragraph-type

Paragraph structure to assume. The value of "

off"
is useful for formatted documents such as Markdown or Textile. Choices are: * auto: Try to auto detect paragraph type * block: Treat a blank line as a paragraph break * single: Assume every line is a paragraph * print: Assume every line starting with 2+ spaces or a tab starts a paragraph * unformatted: Most lines have hard line breaks, few/no blank lines or indents * off: Don'

t modify the paragraph structure

--preserve-spaces

Normally extra spaces are condensed into a single space. With this option all spaces will be displayed.

--txt-in-remove-indent

Normally extra space at the beginning of lines is retained. With this option they will be removed.

[AZW3 Output Options](#)

--dont-compress

Disable compression of the file contents.

--extract-to

Extract the contents of the generated AZW3 file to the specified folder. The contents of the folder are first deleted, so be careful.

--mobi-toc-at-start

When adding the Table of Contents to the book, add it at the start of the book instead of the end. Not recommended.

--no-inline-toc

Don'

t add Table of Contents to the book. Useful if the book has its own table of contents.

--prefer-author-sort

When present, use author sort field as author.

--pretty-print

If specified, the output plugin will try to create output that is as human readable as possible. May not have any effect for some output plugins.

--share-not-sync

Enable sharing of book content via Facebook etc. on the Kindle. WARNING: Using this feature means that the book will not auto sync its last read position on multiple devices. Complain to Amazon.

--toc-title

Title for any generated in-line table of contents.

DOCX Output Options

--docx-custom-page-size

Custom size of the document. Use the form width x height, for example: *123x321* to specify the width and height (in pts). This overrides any specified page-size.

--docx-no-cover

Do not insert the book cover as an image at the start of the document. If you use this option, the book cover will be discarded.

--docx-no-toc

Do not insert the table of contents as a page at the start of the document.

--docx-page-margin-bottom

The size of the bottom page margin, in pts. Default is 72pt. Overrides the common bottom page margin setting, unless set to zero.

--docx-page-margin-left

The size of the left page margin, in pts. Default is 72pt. Overrides the common left page margin setting.

--docx-page-margin-right

The size of the right page margin, in pts. Default is 72pt. Overrides the common right page margin setting, unless set to zero.

--docx-page-margin-top

The size of the top page margin, in pts. Default is 72pt. Overrides the common top page margin setting, unless set to zero.

--docx-page-size

The size of the page. Default is letter. Choices are ['
a0',
,
a1',
,
a2',
,
a3',
,
a4',
,
a5',
,
a6',
,
b0',
,
b1',
,
b2',
,
b3',
,
b4',
,
b5',
,
b6',
,
legal',
,
letter',
]

--extract-to

Extract the contents of the generated DOCX file to the specified folder. The contents of the folder are first deleted, so be careful.

--preserve-cover-aspect-ratio

Preserve the aspect ratio of the cover image instead of stretching it out to cover the entire page.

`--pretty-print`

If specified, the output plugin will try to create output that is as human readable as possible. May not have any effect for some output plugins.

EPUB Output Options

`--dont-split-on-page-breaks`

Turn off splitting at page breaks. Normally, input files are automatically split at every page break into two files. This gives an output e-book that can be parsed faster and with less resources. However, splitting is slow and if your source file contains a very large number of page breaks, you should turn off splitting on page breaks.

`--epub-flatten`

This option is needed only if you intend to use the EPUB with FBReaderJ. It will flatten the file system inside the EPUB, putting all files into the top level.

`--epub-inline-toc`

Insert an inline Table of Contents that will appear as part of the main book content.

`--epub-toc-at-end`

Put the inserted inline Table of Contents at the end of the book instead of the start.

`--epub-version`

The version of the EPUB file to generate. EPUB 2 is the most widely compatible, only use EPUB 3 if you know you actually need it.

`--extract-to`

Extract the contents of the generated EPUB file to the specified folder. The contents of the folder are first deleted, so be careful.

`--flow-size`

Split all HTML files larger than this size (in KB). This is necessary as most EPUB readers cannot handle large file sizes. The default of 260KB is the size required for Adobe Digital Editions. Set to 0 to disable size based splitting.

`--no-default-epub-cover`

Normally, if the input file has no cover and you don't specify one, a default cover is generated with the title, authors, etc. This option disables the generation of this cover.

`--no-svg-cover`

Do not use SVG for the book cover. Use this option if your EPUB is going to be used on a device that does not support SVG, like the iPhone or the JetBook Lite. Without this option, such devices will display the cover as a blank page.

`--preserve-cover-aspect-ratio`

When using an SVG cover, this option will cause the cover to scale to cover the available screen area, but still preserve its aspect ratio (ratio of width to height). That means there may be white borders at the sides or top and bottom of the image, but the image will never be distorted. Without this option the image may be slightly distorted, but there will be no borders.

`--pretty-print`

If specified, the output plugin will try to create output that is as human readable as possible. May not have any effect for some output plugins.

`--toc-title`

Title for any generated in-line table of contents.

FB2 Output Options

`--fb2-genre`

Genre for the book. Choices: sf_history, sf_action, sf_epic, sf_heroic, sf_detective, sf_cyberpunk, sf_space, sf_social, sf_horror, sf_humor, sf_fantasy, sf, det_classic, det_police, det_action, det_irony, det_history, det_espionage, det_crime, det_political, det_maniac, det_hard, thriller, detective, prose_classic, prose_history, prose_contemporary, prose_counter, prose_rus_classic, prose_su_classics, love_contemporary, love_history, love_detective, love_short, love_erotica, adv_western, adv_history, adv_indian, adv_maritime, adv_geo, adv_animal, adventure, child_tale, child_verse, child_prose, child_sf, child_det, child_adv, child_education, children, poetry, dramaturgy, antique_ant, antique_european, antique_russian, antique_east, antique_myths, antique, sci_history, sci_psychology, sci_culture, sci_religion, sci_philosophy, sci_politics, sci_business, sci_juris, sci_linguistic, sci_medicine, sci_phys, sci_math, sci_chem, sci_biology, sci_tech, science, comp_www, comp_programming, comp_hard, comp_soft, comp_db, comp_osnet, computers, ref_encyc, ref_dict, ref_ref, ref_guide, reference, nonf_biography, nonf_publicism, nonf_criticism, design, nonfiction, religion_rel, religion_esoterics, religion_self, religion, humor_anecdote, humor_prose, humor_verse, humor, home_cooking, home_pets, home_crafts,

home_entertain, home_health, home_garden, home_diy, home_sport, home_sex, home See: <http://www.fictionbook.org/index.php/Eng:FictionBook 2.1 genres> for a complete list with descriptions.

--pretty-print

If specified, the output plugin will try to create output that is as human readable as possible. May not have any effect for some output plugins.

--sectionize

Specify how sections are created: * nothing: A single section * files: Section per file * toc: Section per entry in the ToC If ToC based generation fails, adjust the "Structure detection" and/or "Table of Contents" settings (turn on "Force use of auto-generated Table of Contents").

HTML Output Options

--extract-to

Extract the contents of the generated ZIP file to the specified folder. WARNING: The contents of the folder will be deleted.

--pretty-print

If specified, the output plugin will try to create output that is as human readable as possible. May not have any effect for some output plugins.

--template-css

CSS file used for the output instead of the default file

--template-html

Template used for the generation of the HTML contents of the book instead of the default file

--template-html-index

Template used for generation of the HTML index file instead of the default file

HTMLZ Output Options

--htmlz-class-style

How to handle the CSS when using `css-type = 'class'`

. Default is external. external: Use an external CSS file inline: Use a `<style>` tag in the HTML file

`--htmlz-css-type`

Specify the handling of CSS. Default is class. class: Use CSS classes inline: Use the style attribute tag: Use HTML tags wherever possible

`--htmlz-title-filename`

If set this option causes the file name of the HTML file inside the HTMLZ archive to be based on the book title.

`--pretty-print`

If specified, the output plugin will try to create output that is as human readable as possible. May not have any effect for some output plugins.

LIT Output Options

`--pretty-print`

If specified, the output plugin will try to create output that is as human readable as possible. May not have any effect for some output plugins.

LRF Output Options

`--enable-autorotation`

Enable auto-rotation of images that are wider than the screen width.

`--header`

Add a header to all the pages with title and author.

`--header-format`

Set the format of the header. %a is replaced by the author and %t by the title. Default is %t by %a

`--header-separation`

Add extra spacing below the header. Default is 0 pt.

`--minimum-indent`

Minimum paragraph indent (the indent of the first line of a paragraph) in pts. Default: 0

--mono-family

The monospace family of fonts to embed

--pretty-print

If specified, the output plugin will try to create output that is as human readable as possible. May not have any effect for some output plugins.

--render-tables-as-images

This option has no effect

--sans-family

The sans-serif family of fonts to embed

--serif-family

The serif family of fonts to embed

--text-size-multiplier-for-rendered-tables

Multiply the size of text in rendered tables by this factor. Default is 1.0

--wordspace

Set the space between words in pts. Default is 2.5

MOBI Output Options

--dont-compress

Disable compression of the file contents.

--extract-to

Extract the contents of the generated MOBI file to the specified folder. The contents of the folder are first deleted, so be careful.

--mobi-file-type

By default calibre generates MOBI files that contain the old MOBI 6 format. This format is compatible with all devices. However, by changing this setting, you can tell calibre to generate MOBI files that contain both MOBI 6 and the new KF8 format, or only the new KF8

format. KF8 has more features than MOBI 6, but only works with newer Kindles. Allowed values: old, both, new

--mobi-ignore-margins

Ignore margins in the input document. If False, then the MOBI output plugin will try to convert margins specified in the input document, otherwise it will ignore them.

--mobi-keep-original-images

By default calibre converts all images to JPEG format in the output MOBI file. This is for maximum compatibility as some older MOBI viewers have problems with other image formats. This option tells calibre not to do this. Useful if your document contains lots of GIF/PNG images that become very large when converted to JPEG.

--mobi-toc-at-start

When adding the Table of Contents to the book, add it at the start of the book instead of the end. Not recommended.

--no-inline-toc

Don't

add Table of Contents to the book. Useful if the book has its own table of contents.

--personal-doc

Tag for MOBI files to be marked as personal documents. This option has no effect on the conversion. It is used only when sending MOBI files to a device. If the file being sent has the specified tag, it will be marked as a personal document when sent to the Kindle.

--prefer-author-sort

When present, use author sort field as author.

--pretty-print

If specified, the output plugin will try to create output that is as human readable as possible. May not have any effect for some output plugins.

--share-not-sync

Enable sharing of book content via Facebook etc. on the Kindle. WARNING: Using this feature means that the book will not auto sync its last read position on multiple devices. Complain to Amazon.

--toc-title

Title for any generated in-line table of contents.

OEB Output Options

--pretty-print

If specified, the output plugin will try to create output that is as human readable as possible. May not have any effect for some output plugins.

PDB Output Options

--format, -f

Format to use inside the PDB container. Choices are: ['doc', 'ereader', 'ztxt']

--inline-toc

Add Table of Contents to beginning of the book.

--pdb-output-encoding

Specify the character encoding of the output document. The default is cp1252. Note: This option is not honored by all formats.

--pretty-print

If specified, the output plugin will try to create output that is as human readable as possible. May not have any effect for some output plugins.

PDF Output Options

--custom-size

Custom size of the document. Use the form width x height e.g. *123x321* to specify the width and height. This overrides any specified paper-size.

--paper-size

The size of the paper. This size will be overridden when a non default output profile is used. Default is letter. Choices are a0, a1, a2, a3, a4, a5, a6, b0, b1, b2, b3, b4, b5, b6, legal, letter

`--pdf-add-toc`

Add a Table of Contents at the end of the PDF that lists page numbers. Useful if you want to print out the PDF. If this PDF is intended for electronic use, use the PDF Outline instead.

`--pdf-default-font-size`

The default font size (in pixels)

`--pdf-footer-template`

An HTML template used to generate footers on every page. The strings `_PAGENUM_`, `_TITLE_`, `_AUTHOR_` and `_SECTION_` will be replaced by their current values.

`--pdf-header-template`

An HTML template used to generate headers on every page. The strings `_PAGENUM_`, `_TITLE_`, `_AUTHOR_` and `_SECTION_` will be replaced by their current values.

`--pdf-hyphenate`

Break long words at the end of lines. This can give the text at the right margin a more even appearance. Note that depending on the fonts used this option can break the copying of text from the PDF file.

`--pdf-mark-links`

Surround all links with a red box, useful for debugging.

`--pdf-mono-family`

The font family used to render monospace fonts. Will work only if the font is available system-wide.

`--pdf-mono-font-size`

The default font size for monospaced text (in pixels)

`--pdf-odd-even-offset`

Shift the text horizontally by the specified offset (in pts). On odd numbered pages, it is shifted to the right and on even numbered pages to the left. Use negative numbers for the opposite effect. Note that this setting is ignored on pages where the margins are smaller than the specified offset. Shifting is done by setting the PDF CropBox, not all software respects the CropBox.

`--pdf-page-margin-bottom`

The size of the bottom page margin, in pts. Default is 72pt. Overrides the common bottom page margin setting, unless set to zero.

`--pdf-page-margin-left`

The size of the left page margin, in pts. Default is 72pt. Overrides the common left page margin setting.

`--pdf-page-margin-right`

The size of the right page margin, in pts. Default is 72pt. Overrides the common right page margin setting, unless set to zero.

`--pdf-page-margin-top`

The size of the top page margin, in pts. Default is 72pt. Overrides the common top page margin setting, unless set to zero.

`--pdf-page-number-map`

Adjust page numbers, as needed. Syntax is a JavaScript expression for the page number. For example,

```
if (n < 3) 0; else n - 3;
```

, where n is current page number.

`--pdf-page-numbers`

Add page numbers to the bottom of every page in the generated PDF file. If you specify a footer template, it will take precedence over this option.

`--pdf-sans-family`

The font family used to render sans-serif fonts. Will work only if the font is available system-wide.

`--pdf-serif-family`

The font family used to render serif fonts. Will work only if the font is available system-wide.

`--pdf-standard-font`

The font family used to render monospace fonts

`--pdf-use-document-margins`

Use the page margins specified in the input document via @page CSS rules. This will cause the margins specified in the conversion settings to be ignored. If the document does not specify page margins, the conversion settings will be used as a fallback.

--preserve-cover-aspect-ratio

Preserve the aspect ratio of the cover, instead of stretching it to fill the full first page of the generated PDF.

--pretty-print

If specified, the output plugin will try to create output that is as human readable as possible. May not have any effect for some output plugins.

--toc-title

Title for generated table of contents.

--uncompressed-pdf

Generate an uncompressed PDF, useful for debugging.

--unit, -u

The unit of measure for page sizes. Default is inch. Choices are millimeter, centimeter, point, inch, pica, didot, cicero, devicepixel Note: This does not override the unit for margins!

--use-profile-size

Instead of using the paper size specified in the PDF Output options, use a paper size corresponding to the current output profile. Useful if you want to generate a PDF for viewing on a specific device.

PML Output Options

--full-image-depth

Do not reduce the size or bit depth of images. Images have their size and depth reduced by default to accommodate applications that can not convert images on their own such as Dropbox.

--inline-toc

Add Table of Contents to beginning of the book.

--pml-output-encoding

Specify the character encoding of the output document. The default is cp1252.

--pretty-print

If specified, the output plugin will try to create output that is as human readable as possible. May not have any effect for some output plugins.

RB Output Options

--inline-toc

Add Table of Contents to beginning of the book.

--pretty-print

If specified, the output plugin will try to create output that is as human readable as possible. May not have any effect for some output plugins.

RTF Output Options

--pretty-print

If specified, the output plugin will try to create output that is as human readable as possible. May not have any effect for some output plugins.

SNB Output Options

--pretty-print

If specified, the output plugin will try to create output that is as human readable as possible. May not have any effect for some output plugins.

--snb-dont-indent-first-line

Specify whether or not to insert two space characters to indent the first line of each paragraph.

--snb-full-screen

Resize all the images for full screen mode.

--snb-hide-chapter-name

Specify whether or not to hide the chapter title for each chapter. Useful for image-only output (eg. comics).

--snb-insert-empty-line

Specify whether or not to insert an empty line between two paragraphs.

--snb-max-line-length

The maximum number of characters per line. This splits on the first space before the specified value. If no space is found the line will be broken at the space after and will exceed the specified value. Also, there is a minimum of 25 characters. Use 0 to disable line splitting.

--snb-output-encoding

Specify the character encoding of the output document. The default is utf-8.

TCR Output Options

--pretty-print

If specified, the output plugin will try to create output that is as human readable as possible. May not have any effect for some output plugins.

--tcr-output-encoding

Specify the character encoding of the output document. The default is utf-8.

TXT Output Options

--force-max-line-length

Force splitting on the max-line-length value when no space is present. Also allows max-line-length to be below the minimum

--inline-toc

Add Table of Contents to beginning of the book.

--keep-color

Do not remove font color from output. This is only useful when TXT output formatting is set to textile. Textile is the only formatting that supports setting font color. If this option is not specified font color will not be set and default to the color displayed by the reader (generally this is black).

--keep-image-references

Do not remove image references within the document. This is only useful when paired with a TXT output formatting option that is not none because links are always removed with plain text output.

--keep-links

Do not remove links within the document. This is only useful when paired with a TXT output formatting option that is not none because links are always removed with plain text output.

--max-line-length

The maximum number of characters per line. This splits on the first space before the specified value. If no space is found the line will be broken at the space after and will exceed the specified value. Also, there is a minimum of 25 characters. Use 0 to disable line splitting.

--newline, -n

Type of newline to use. Options are ['old_mac', 'system', 'unix', 'windows']. Default is 'system'. Use 'old_mac' for compatibility with Mac OS 9 and earlier. For macOS use 'unix'. 'system' will default to the newline type used by this OS.

--pretty-print

If specified, the output plugin will try to create output that is as human readable as possible. May not have any effect for some output plugins.

--txt-output-encoding

Specify the character encoding of the output document. The default is utf-8.

--txt-output-formatting

Formatting used within the document. * plain: Plain text * markdown: Markdown formatted text * textile: Textile formatted text

[TXTZ Output Options](#)

--force-max-line-length

Force splitting on the max-line-length value when no space is present. Also allows max-line-length to be below the minimum

--inline-toc

Add Table of Contents to beginning of the book.

--keep-color

Do not remove font color from output. This is only useful when TXT output formatting is set to textile. Textile is the only formatting that supports setting font color. If this option is not specified font color will not be set and default to the color displayed by the reader (generally this is black).

--keep-image-references

Do not remove image references within the document. This is only useful when paired with a TXT output formatting option that is not none because links are always removed with plain text output.

--keep-links

Do not remove links within the document. This is only useful when paired with a TXT output formatting option that is not none because links are always removed with plain text output.

--max-line-length

The maximum number of characters per line. This splits on the first space before the specified value. If no space is found the line will be broken at the space after and will exceed the specified value. Also, there is a minimum of 25 characters. Use 0 to disable line splitting.

--newline, -n

Type	of	newline	to	use.	Options	are	['
old_mac'							,
,							,
system'							,
,							,
unix'							,
,							,
windows'							,
].		Default			is		,
system'							,
.				Use			,

old_mac'
for compatibility with Mac OS 9 and earlier. For macOS use '
unix'
,
system'
will default to the newline type used by this OS.

--pretty-print

If specified, the output plugin will try to create output that is as human readable as possible.
May not have any effect for some output plugins.

--txt-output-encoding

Specify the character encoding of the output document. The default is utf-8.

--txt-output-formatting

Formatting used within the document. * plain: Plain text * markdown: Markdown formatted text * textile: Textile formatted text

ebook-edit

```
ebook-edit [opts] [path_to_ebook] [name_of_file_inside_book ...]
```

Launch the calibre Edit book tool. You can optionally also specify the names of files inside the book which will be opened for editing automatically.

Whenever you pass arguments to **ebook-edit** that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

[options]

--detach

Detach from the controlling terminal, if any (Linux only)

--help, -h

show this help message and exit

--select-text

The text to select in the book when it is opened for editing

--version

show

s version number and exit

program'

ebook-meta

`ebook-meta ebook_file [options]`

Read/Write metadata from/to e-book files.

Supported formats for reading metadata: azw, azw1, azw3, azw4, cb7, cbr, cbz, chm, docx, epub, fb2, fbz, html, htmlz, imp, lit, lrf, lrx, mobi, odt, oebzip, opf, pdb, pdf, pml, pmlz, pobi, prc, rar, rb, rtf, snb, tpz, txt, txtz, updb, zip

Supported formats for writing metadata: azw, azw1, azw3, azw4, docx, epub, fb2, fbz, htmlz, lrf, mobi, odt, pdb, pdf, prc, rtf, tpz, txtz

Different file types support different kinds of metadata. If you try to set some metadata on a file type that does not support it, the metadata will be silently ignored.

Whenever you pass arguments to **ebook-meta** that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

[options]

`--author-sort`

String to be used when sorting by author. If unspecified, and the author(s) are specified, it will be auto-generated from the author(s).

`--authors, -a`

Set the authors. Multiple authors should be separated by the & character. Author names should be in the order Firstname Lastname.

`--book-producer, -k`

Set the book producer.

`--category`

Set the book category.

--comments, -c

Set the e-book description.

--cover

Set the cover to the specified file.

--date, -d

Set the published date.

--from-opf

Read metadata from the specified OPF file and use it to set metadata in the e-book. Metadata specified on the command line will override metadata read from the OPF file

--get-cover

Get the cover from the e-book and save it at as the specified file.

--help, -h

show this help message and exit

--identifier

Set the identifiers for the book, can be specified multiple times. For example: **--identifier** uri:https://acme.com **--identifier** isbn:12345 To remove an identifier, specify no value, **--identifier** isbn: Note that for EPUB files, an identifier marked as the package identifier cannot be removed.

--index, -i

Set the index of the book in this series.

--isbn

Set the ISBN of the book.

--language, -l

Set the language.

--lrf-bookid

Set the BookID in LRF files

--publisher, -p

Set the e-book publisher.

--rating, -r

Set the rating. Should be a number between 1 and 5.

--series, -s

Set the series this e-book belongs to.

--tags

Set the tags for the book. Should be a comma separated list.

--title, -t

Set the title.

--title-sort

The version of the title to be used for sorting. If unspecified, and the title is specified, it will be auto-generated from the title.

--to-opf

Specify the name of an OPF file. The metadata will be written to the OPF file.

--version

show

program'

s version number and exit

ebook-polish

```
ebook-polish [options] input_file [output_file]
```

Polishing books is all about putting the shine of perfection onto your carefully crafted e-books.

Polishing tries to minimize the changes to the internal code of your e-book. Unlike conversion, it does not flatten CSS, rename files, change font sizes, adjust margins, etc. Every action performs only the minimum set of changes needed for the desired effect.

You should use this tool as the last step in your e-book creation process.

Note that polishing only works on files in the AZW3 or EPUB formats.

Whenever you pass arguments to **ebook-polish** that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

[options]

--add-soft-hyphens, -H

Add soft hyphens to all words in the book. This allows the book to be rendered better when the text is justified, in readers that do not support hyphenation.

--compress-images, -i

Losslessly compress images in the book, to reduce the filesize, without affecting image quality.

--cover, -c

Path to a cover image. Changes the cover specified in the e-book. If no cover is present, or the cover is not properly identified, inserts a new cover.

--embed-fonts, -e

Embed all fonts that are referenced in the document and are not already embedded. This will scan your computer for the fonts, and if they are found, they will be embedded into the document. Please ensure that you have the proper license for embedding the fonts used in this document.

--help, -h

show this help message and exit

--jacket, -j

Insert a "book jacket" page at the start of the book that contains all the book metadata such as title, tags, authors, series, comments, etc. Any previous book jacket will be replaced.

--opf, -o

Path to an OPF file. The metadata in the book is updated from the OPF file.

--remove-jacket

Remove a previous inserted book jacket page.

--remove-soft-hyphens

Remove soft hyphens from all text in the book.

--remove-unused-css, -u

Remove all unused CSS rules from stylesheets and <style> tags. Some books created from production templates can have a large number of extra CSS rules that don't match any actual content. These extra rules can slow down readers that need to parse them all.

--smarten-punctuation, -p

Convert plain text dashes, ellipsis, quotes, multiple hyphens, etc. into their typographically correct equivalents. Note that the algorithm can sometimes generate incorrect results, especially when single quotes at the start of contractions are involved.

--subset-fonts, -f

Subsetting fonts means reducing an embedded font to contain only the characters used from that font in the book. This greatly reduces the size of the font files (halving the font file sizes is common). For example, if the book uses a specific font for headers, then subsetting will reduce that font to contain only the characters present in the actual headers in the book. Or

if the book embeds the bold and italic versions of a font, but bold and italic text is relatively rare, or absent altogether, then the bold and italic fonts can either be reduced to only a few characters or completely removed. The only downside to subsetting fonts is that if, at a later date you decide to add more text to your books, the newly added text might not be covered by the subset font.

--upgrade-book, -U

Upgrade the internal structures of the book, if possible. For instance, upgrades EPUB 2 books to EPUB 3 books.

--verbose

Produce more verbose output, useful for debugging.

--version

show

program'

s version number and exit

ebook-viewer

```
ebook-viewer [options] file
```

View an e-book.

Whenever you pass arguments to **ebook-viewer** that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

[options]

--continue

Continue reading the last opened book

--detach

Detach from the controlling terminal, if any (Linux only)

--force-reload

Force reload of all opened books

--full-screen, --fullscreen, -f

If specified, the E-book viewer window will try to open full screen when started.

--help, -h

show this help message and exit

--open-at

The position at which to open the specified book. The position is a location or position you can get by using the Go to->Location action in the viewer controls. Alternately, you can use the form toc:something and it will open at the location of the first Table of Contents entry that contains the string something"

. The form `toc-href:something` will match the `href` (internal link destination) of `toc` nodes. The matching is exact. If you want to match a substring, use the form `toc-href-contains:something`. The form `ref:something` will use Reference mode references.

`--raise-window`

If specified, the E-book viewer window will try to come to the front when started.

`--version`

`show`

`program'`

`s version number and exit`

fetch-ebook-metadata

fetch-ebook-metadata [options]

Fetch book metadata from online sources. You must specify at least one of title, authors or ISBN.

Whenever you pass arguments to **fetch-ebook-metadata** that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

[options]

--allowed-plugin, -p

Specify the name of a metadata download plugin to use. By default, all metadata plugins will be used. Can be specified multiple times for multiple plugins. All plugin names: Google, Google Images, Amazon.com, Edelweiss, Open Library, Big Book Search

--authors, -a

Book author(s)

--cover, -c

Specify a filename. The cover, if available, will be saved to it. Without this option, no cover will be downloaded.

--help, -h

show this help message and exit

--identifier, -I

Identifiers such as ASIN/Goodreads id etc. Can be specified multiple times for multiple identifiers. For example: **--identifier**

asin:B0082BAJA0

--isbn, -i

Book ISBN

--opf, -o

Output the metadata in OPF format instead of human readable text.

--timeout, -d

Timeout in seconds. Default is 30

--title, -t

Book title

--verbose, -v

Print the log to the console (stderr)

--version

show

s version number and exit

program'

lrf2lrs

lrf2lrs book.lrf

Convert an LRF file into an LRS (XML UTF-8 encoded) file

Whenever you pass arguments to **lrf2lrs** that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

[options]

--dont-output-resources

Do not save embedded image and font files to disk

--help, -h

show this help message and exit

--output, -o

Output LRS file

--verbose

Be more verbose

--version

show program'
s version number and exit

lrfviewer

```
lrfviewer [options] book.lrf
```

Read the LRF e-book book.lrf

Whenever you pass arguments to **lrfviewer** that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

[options]

--disable-hyphenation

Disable hyphenation. Should significantly speed up rendering.

--help, -h

show this help message and exit

--profile

Profile the LRF renderer

--verbose

Print more information about the rendering process

--version

show program'
s version number and exit

--visual-debug

Turn on visual aids to debugging the rendering engine

--white-background

By default the background is off white as I find this easier on the eyes. Use this option to make the background pure white.

lrs2lrf

```
lrs2lrf [options] file.lrs
```

Compile an LRS file into an LRF file.

Whenever you pass arguments to **lrs2lrf** that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

[options]

--help, -h

show this help message and exit

--lrs

Convert LRS to LRS, useful for debugging.

--output, -o

Path to output file

--verbose

Verbose processing

--version

show program'
s version number and exit

web2disk

web2disk URL

Where URL is for example <https://google.com>

Whenever you pass arguments to **web2disk** that have spaces in them, enclose the arguments in quotation marks. For example: “/some path/with spaces”

[options]

--base-dir, -d

Base folder into which URL is saved. Default is .

--delay

Minimum interval in seconds between consecutive fetches. Default is 0 s

--dont-download-stylesheets

Do not download CSS stylesheets.

--encoding

The character encoding for the websites you are trying to download. The default is to try and guess the encoding.

--filter-regexp

Any link that matches this regular expression will be ignored. This option can be specified multiple times, in which case as long as any regexp matches a link, it will be ignored. By default, no links are ignored. If both filter regexp and match regexp are specified, then filter regexp is applied first.

--help, -h

show this help message and exit

`--match-regex`

Only links that match this regular expression will be followed. This option can be specified multiple times, in which case as long as a link matches any one regexp, it will be followed. By default all links are followed.

`--max-files, -n`

The maximum number of files to download. This only applies to files from `<a href>` tags. Default is 9223372036854775807

`--max-recursions, -r`

Maximum number of levels to recurse i.e. depth of links to follow. Default 1

`--timeout, -t`

Timeout in seconds to wait for a response from the server. Default: 10.0 s

`--verbose`

Show detailed output information. Useful for debugging

`--version`

show

s version number and exit

program'

Setting up a calibre development environment

calibre is completely open source, licensed under the [GNU GPL v3](https://www.gnu.org/licenses/gpl.html) [https://www.gnu.org/licenses/gpl.html]. This means that you are free to download and modify the program to your heart's content. In this section, you will learn how to get a calibre development environment set up on the operating system of your choice. calibre is written primarily in [Python](https://www.python.org) [https://www.python.org] with some C/C++ code for speed and system interfacing. Note that calibre requires at least Python 3.8.

Contents

- [Design philosophy](#)
 - [Code layout](#)
- [Getting the code](#)
 - [Submitting your changes to be included](#)
- [Windows development environment](#)
- [macOS development environment](#)
- [Linux development environment](#)
- [Having separate “normal” and “development” calibre installs on the same computer](#)
- [Debugging tips](#)
 - [Using print statements](#)
 - [Using an interactive Python interpreter](#)
 - [Using the Python debugger as a remote debugger](#)
 - [Using the debugger in your favorite Python IDE](#)

- [Executing arbitrary scripts in the calibre Python environment](#)
- [Using calibre in your projects](#)
 - [Binary install of calibre](#)
 - [Source install on Linux](#)
- [API documentation for various parts of calibre](#)

Design philosophy

calibre has its roots in the Unix world, which means that its design is highly modular. The modules interact with each other via well defined interfaces. This makes adding new features and fixing bugs in calibre very easy, resulting in a frenetic pace of development. Because of its roots, calibre has a comprehensive command line interface for all its functions, documented in [Command Line Interface](#).

The modular design of calibre is expressed via Plugins. There is a [tutorial](#) on writing calibre plugins. For example, adding support for a new device to calibre typically involves writing less than a 100 lines of code in the form of a device driver plugin. You can browse the [built-in drivers](#) [<https://github.com/kovidgoyal/calibre/tree/master/src/calibre/devices>]. Similarly, adding support for new conversion formats involves writing input/output format plugins. Another example of the modular design is the [recipe system](#) for fetching news. For more examples of plugins designed to add features to calibre, see the [Index of plugins](#) [<https://www.mobileread.com/forums/showthread.php?p=1362767#post1362767>].

Code layout

All the calibre python code is in the calibre package. This package contains the following main sub-packages

- devices - All the device drivers. Just look through some of the built-in drivers to get an idea for how they work.
 - For details, see: `devices.interface` which defines the interface supported by device drivers and `devices.usbms` which defines a generic driver that connects to a USBMS device. All USBMS based drivers in calibre inherit from it.
- e-books - All the e-book conversion/metadata code. A good starting point is `calibre.ebooks.conversion.cli` which is the module powering the **ebook-convert** command. The conversion

process is controlled via `conversion.plumber`.
 The format independent code is all in `ebooks.oeb`
 and the format dependent code is in `ebooks.format_name`.

- Metadata reading, writing, and downloading is all in `ebooks.metadata`
- Conversion happens in a pipeline, for the structure of the pipeline, see [Introduction](#). The pipeline consists of an input plugin, various transforms and an output plugin. The code that constructs and drives the pipeline is in `plumber.py`. The pipeline works on a representation of an e-book that is like an unzipped epub, with manifest, spine, toc, guide, html content, etc. The class that manages this representation is `OEBBook` in `ebooks.oeb.base`. The various transformations that are applied to the book during conversions live in `oeb/transforms/*.py`. And the input and output plugins live in `conversion/plugins/*.py`.
- E-book editing happens using a different container object. It is documented in [API documentation for the e-book editing tools](#).
- db - The database back-end. See [API documentation for the database interface](#) for the interface to the calibre library.
- Content server: `srv`
is the calibre Content server.
- gui2 - The Graphical User Interface. GUI initialization happens in `gui2.main` and `gui2.ui`. The e-book-viewer is in `gui2.viewer`. The e-book editor is in `gui2.tweak_book`.

If you want to locate the entry points for all the various calibre executables, look at the `entry_points`

structure in [linux.py](#) [<https://github.com/kovidgoyal/calibre/blob/master/src/calibre/linux.py>].

If you need help understanding the code, post in the [development forum](#) [<https://www.mobileread.com/forums/forumdisplay.php?f=240>] and you will most likely get help from one of calibre's many developers.

[Getting the code](#)

You can get the calibre source code in two ways, using a version control system or directly downloading a [tarball](https://calibre-ebook.comdist/src) [https://calibre-ebook.comdist/src].

calibre uses [Git](https://www.git-scm.com/) [https://www.git-scm.com/], a distributed version control system. Git is available on all the platforms calibre supports. After installing Git, you can get the calibre source code with the command:

```
git
clone

git
:
//
github
.
com
/
kovidgoyal
/
calibre
.
git
```

On Windows you will need the complete path name, that will be something like C:\Program

Files\Git\git.exe

.

calibre is a very large project with a very long source control history, so the above can take a while (10 mins to an hour depending on your internet speed).

If you want to get the code faster, the source code for the latest release is always available as an [archive](https://calibre-ebook.comdist/src) [https://calibre-ebook.comdist/src].

To update a branch to the latest code, use the command:

```
git
pull

--
no
```



```
-  
edit
```

You can also browse the code at [GitHub](https://github.com/kovidgoyal/calibre) [https://github.com/kovidgoyal/calibre].

Submitting your changes to be included

If you only plan to make a few small changes, you can make your changes and create a “merge directive” which you can then attach to a ticket in the calibre [bug tracker](https://bugs.launchpad.net/calibre) [https://bugs.launchpad.net/calibre]. To do this, make your changes, then run:

```
git  
  
commit  
  
-  
am  
  
"Comment describing your changes"  
  
git  
  
format  
-  
patch  
  
origin  
/  
master  
  
--  
stdout  
  
>  
  
my  
-  
changes
```

This will create a my-changes file in the current folder, simply attach that to a ticket on the calibre [bug tracker](https://bugs.launchpad.net/calibre) [https://bugs.launchpad.net/calibre]. Note that this will include *all* the commits you have made. If you only want to send some commits, you have to change origin/master above. To send only the last commit, use:

```
git
format
-
patch
HEAD
~
1
--
stdout
>
my
-
changes
```

To send the last n commits, replace *1* with n , for example, for the last 3 commits:

```
git
format
-
patch
HEAD
~
3
--
stdout
>
```

```
my
-
changes
```

Be careful to not include merges when using HEAD~n.

If you plan to do a lot of development on calibre, then the best method is to create a [GitHub](https://github.com) [https://github.com] account. Below is a basic guide to setting up your own fork of calibre in a way that will allow you to submit pull requests for inclusion into the main calibre repository:

- Setup git on your machine as described in this article: [Setup Git](https://help.github.com/articles/set-up-git) [https://help.github.com/articles/set-up-git]
- Setup ssh keys for authentication to GitHub, as described here: [Generating SSH keys](https://help.github.com/articles/generating-ssh-keys) [https://help.github.com/articles/generating-ssh-keys]
- Go to <https://github.com/kovidgoyal/calibre> and click the *Fork* button.
- In a Terminal do:

```
git
clone

git
@github
.
com
:
<
username
>/
calibre
.
git

git

remote

add
```

```
upstream  
  
https  
:  
//  
github  
.  
com  
/  
kovidgoyal  
/  
calibre  
.  
git
```

Replace <username> above with your GitHub username. That will get your fork checked out locally.

- You can make changes and commit them whenever you like. When you are ready to have your work merged, do a:

```
git  
  
push
```

and go to <https://github.com/<username>/calibre> and click the *Pull Request* button to generate a pull request that can be merged.

- You can update your local copy with code from the main repo at any time by doing:

```
git  
  
pull  
  
upstream
```

You should also keep an eye on the calibre [development forum](https://www.mobileread.com/forums/forumdisplay.php?f=240) [https://www.mobileread.com/forums/forumdisplay.php?f=240]. Before making major changes, you should discuss them in the forum or contact Kovid directly (his email address is all over the source code).

Windows development environment

Note

You must also get the calibre source code separately as described above.

Install calibre normally, using the Windows installer. Then open a Command Prompt and change to the previously checked out calibre code folder. For example:

```
cd  
  
C  
:  
\  
Users  
\  
kovid  
\  
work  
\  
calibre
```

calibre is the folder that contains the src and resources sub-folders.

The next step is to set the environment variable CALIBRE_DEVELOP_FROM to the absolute path of the src folder. So, following the example above, it would be C:\Users\kovid\work\calibre\src

. [Here is a short guide](https://docs.python.org/using/windows.html#excursus-setting-environment-variables) [https://docs.python.org/using/windows.html#excursus-setting-environment-variables] to setting environment variables on Windows.

Once you have set the environment variable, open a new command prompt and check that it was correctly set by using the command:

```
echo  
  
%  
CALIBRE_DEVELOP_FROM  
%
```

Setting this environment variable means that calibre will now load all its Python code from the specified location.

That's it! You are now ready to start hacking on the calibre code. For example, open the file `src\calibre\__init__.py` in your favorite editor and add the line:

```
print  
  
(  
"Hello, world!"  
)
```

near the top of the file. Now run the command **calibredb**. The very first line of output should be

world!

.

You can also setup a calibre development environment inside the free Microsoft Visual Studio, if you like, following the instructions [here](https://www.mobileread.com/forums/showthread.php?t=251201) [https://www.mobileread.com/forums/showthread.php?t=251201].

[macOS development environment](#)

Note

You must also get the calibre source code separately as described above.

Install calibre normally using the provided .dmg. Then open a Terminal and change to the previously checked out calibre code folder, for example:

```
cd  
  
/  
Users  
/  
kovid  
/  
work
```

```
/
calibre
```

calibre is the folder that contains the src and resources sub-folders. The calibre command line tools are found inside the calibre app bundle, in /Applications/calibre.app/Contents/MacOS
you should add this folder to your PATH environment variable, if you want to run the command line tools easily.

The next step is to create a bash script that will set the environment variable CALIBRE_DEVELOP_FROM to the absolute path of the src folder when running calibre in debug mode.

Create a plain text file:

```
#!/bin/sh

export
CALIBRE_DEVELOP_FROM
=
"/Users/kovid/work/calibre/src"

calibre
-
debug
-
g
```

Save this file as /usr/local/bin/calibre-develop , then set its permissions so that it can be executed:

```
chmod
+
x
/
usr
```

```
/
local
/
bin
/
calibre
-
develop
```

Once you have done this, run:

```
calibre
-
develop
```

You should see some diagnostic information in the Terminal window as calibre starts up, and you should see an asterisk after the version number in the GUI window, indicating that you are running from source.

Linux development environment

Note

You must also get the calibre source code separately as described above.

calibre is primarily developed on Linux. You have two choices in setting up the development environment. You can install the calibre binary as normal and use that as a runtime environment to do your development. This approach is similar to that used in Windows and macOS. Alternatively, you can install calibre from source. Instructions for setting up a development environment from source are in the INSTALL file in the source tree. Here we will address using the binary as a runtime, which is the recommended method.

Install calibre using the binary installer. Then open a terminal and change to the previously checked out calibre code folder, for example:

```
cd

/
home
/
kovid
```



```
/
work
/
calibre
```

calibre is the folder that contains the src and resources sub-folders.

The next step is to set the environment variable `CALIBRE_DEVELOP_FROM` to the absolute path of the src folder. So, following the example above, it would be `/home/kovid/work/calibre/src`

. How to set environment variables depends on your Linux distribution and what shell you are using.

Once you have set the environment variable, open a new terminal and check that it was correctly set by using the command:

```
echo $CALIBRE_DEVELOP_FROM
```

Setting this environment variable means that calibre will now load all its Python code from the specified location.

That's it! You are now ready to start hacking on the calibre code. For example, open the file `src/calibre/__init__.py`

in your favorite editor and add the line:

```
print
(
"Hello, world!"
)
```

near the top of the file. Now run the command **calibredb**. The very first line of output should be

Hello,

world!

.

[Having separate “normal” and “development” calibre installs on the same computer](#)

The calibre source tree is very stable and rarely breaks, but if you feel the need to run from source on a separate test library and run the released calibre version with your everyday library, you can achieve this easily using .bat files or shell scripts to launch calibre. The example below shows how to do this on Windows using .bat files (the instructions for other platforms are the same, just use a shell script instead of a .bat file)

To launch the release version of calibre with your everyday library:

calibre-normal.bat:

```
calibre
.
exe

"--with-library=C:\path
\t
o\everyday\library folder"
```

calibre-dev.bat:

```
set

CALIBRE_DEVELOP_FROM
=
C
:
\
path
\
to
\
calibre
\
checkout
\
src

calibre
.
exe

"--with-library=C:\path
```

```
\t
o
\t
est\library folder"
```

Debugging tips

Python is a dynamically typed language with excellent facilities for introspection. Kovid wrote the core calibre code without once using a debugger. There are many strategies to debug calibre code:

Using print statements

This is Kovid's favorite way to debug. Simply insert print statements at points of interest and run your program in the terminal. For example, you can start the GUI from the terminal as:

```
calibre
-
debug
-
g
```

Similarly, you can start the E-book viewer as:

```
calibre
-
debug
-
w

/
path
/
to
/
file
/
to
```

```
/
be
/
viewed
```

The e-book-editor can be started as:

```
calibre
-
debug
-
t
/
path
/
to
/
be
/
edited
```

[Using an interactive Python interpreter](#)

You can insert the following two lines of code to start an interactive Python session at that point:

```
from
calibre
import
ipython
ipython
(
locals
())
```

When running from the command line, this will start an interactive Python interpreter with access to all locally defined variables (variables in the local scope). The interactive prompt even has TAB completion for object properties and you can use the various Python facilities for introspection, such as `dir()`, `type()`, `repr()`, etc.

[Using the Python debugger as a remote debugger](#)

You can use the builtin Python debugger (pdb) as a remote debugger from the command line. First, start the remote debugger at the point in the calibre code you are interested in, like this:

```
from
calibre.rpdb

import

set_trace

set_trace
()
```

Then run calibre, either as normal, or using one of the calibre-debug commands described in the previous section. Once the above point in the code is reached, calibre will freeze, waiting for the debugger to connect.

Now open a terminal or command prompt and use the following command to start the debugging session:

```
calibre
-
debug
-
c

"from calibre.rpdb import cli; cli()"
```

You can read about how to use the Python debugger in the [Python stdlib docs for the pdb module](https://docs.python.org/library/pdb.html#debugger-commands) [https://docs.python.org/library/pdb.html#debugger-commands].

Note

By default, the remote debugger will try to connect on port 4444. You can change it, by passing the port parameter to both the `set_trace()` and the `cli()` functions above, like this:

```
set_trace(port=1234)
and
cli(port=1234)
.
```

Note

The Python debugger cannot handle multiple threads, so you have to call `set_trace` once per thread, each time with a different port number.

Using the debugger in your favorite Python IDE

It is possible to use the builtin debugger in your favorite Python IDE, if it supports remote debugging. The first step is to add the calibre src checkout to the PYTHONPATH in your IDE. In other words, the folder you set as `CALIBRE_DEVELOP_FROM` above, must also be in the PYTHONPATH of your IDE.

Then place the IDE's remote debugger module into the src sub-folder of the calibre source code checkout. Add whatever code is needed to launch the remote debugger to calibre at the point of interest, for example in the main function. Then run calibre as normal. Your IDE should now be able to connect to the remote debugger running inside calibre.

Executing arbitrary scripts in the calibre Python environment

The **calibre-debug** command provides a couple of handy switches to execute your own code, with access to the calibre modules:

```
calibre
-
debug
-
c

"some Python code"
```

is great for testing a little snippet of code on the command line. It works in the same way as the -c switch to the Python interpreter:

```
calibre
-
debug

myscript
.
py
```

can be used to execute your own Python script. It works in the same way as passing the script to the Python interpreter, except that the calibre environment is fully initialized, so you can use all the calibre code in your script. To use command line arguments with your script, use the form:

```
calibre
-
debug

myscript
.
py

--

--
option1

arg1
```

The `--` causes all subsequent arguments to be passed to your script.

[Using calibre in your projects](#)

It is possible to directly use calibre functions/code in your Python project. Two ways exist to do this:

Binary install of calibre

If you have a binary install of calibre, you can use the Python interpreter bundled with calibre, like this:

```
calibre
-
debug

/
path
/
to
/
your
/
python
/
script
.
py

--

arguments

to

your

script
```

Source install on Linux

In addition to using the above technique, if you do a source install on Linux, you can also directly import calibre, as follows:

```
import
```



```
init_calibre  
  
import  
  
calibre  
  
print  
  
calibre  
.  
__version__
```

It is essential that you import the `init_calibre` module before any other calibre modules/packages as it sets up the interpreter to run calibre code.

[API documentation for various parts of calibre](#)

- [API documentation for recipes](#)
- [API documentation for plugins](#)
- [API documentation for the database interface](#)
- [API documentation for the e-book editing tools](#)

API documentation for the database interface

This API is thread safe (it uses a multiple reader, single writer locking scheme). You can access this API like this:

```
from
calibre.library

import

db

db

=

db
(
'Path to calibre library folder'
)
.
new_api
```

If you are in a calibre plugin that is part of the main calibre GUI, you get access to it like this instead:

```
db

=

self
```

```

.
gui
.
current_db
.
new_api

```

`class calibre.db.cache.Cache(backend)`[\[source\]](#)

An in-memory cache of the metadata.db file from a calibre library. This class also serves as a threadsafe API for accessing the database. The in-memory cache is maintained in normal form for maximum performance.

SQLITE is simply used as a way to read and write from metadata.db robustly. All table reading/sorting/searching/caching logic is re-implemented. This was necessary for maximum performance and flexibility.

`class EventType(value)`

An enumeration.

`add_books(books, add_duplicates=True, apply_import_tags=True, preserve_uuid=False, run_hooks=True, dbapi=None)`[\[source\]](#)

Add the specified books to the library. Books should be an iterable of 2-tuples, each 2-tuple of the form (mi,

format_map)

where mi is a Metadata object and format_map is a dictionary of the form {fmt:

path_or_stream}

, for example: {'EPUB':

'/path/to/file.epub'}

.

Returns a pair of lists: ids,

duplicates

. ids

contains the book ids for all newly created books in the database. duplicates contains the (mi,

format_map)

for all books that already exist in the database as per the simple duplicate detection heuristic
used by [has_book\(\)](#)

.

`add_custom_book_data(name, val_map, delete_first=False)` [\[source\]](#)

Add data for name where val_map is a map of book_ids to values. If delete_first is True, all previously stored data for name will be removed.

`add_format(book_id, fmt, stream_or_path, replace=True, run_hooks=True, dbapi=None)` [\[source\]](#)

Add a format to the specified book. Return True if the format was added successfully.

Parameters

- **replace** – If True replace existing format, otherwise if the format already exists, return False.
- **run_hooks** – If True, file type plugins are run on the format before and after being added.
- **dbapi** – Internal use only.

`add_listener(event_callback_function)` [\[source\]](#)

Register a callback function that will be called after certain actions are taken on this database. The function must take three arguments: ([EventType](#), library_id, event_type_specific_data)

`all_book_ids(type=<class 'frozenset'>)` [\[source\]](#)

Frozen set of all known book ids.

`all_field_for(field, book_ids, default_value=None)` [\[source\]](#)

Same as field_for, except that it operates on multiple books at once

`all_field_ids(name)` [\[source\]](#)

Frozen set of ids for all values in the field name
.

`all_field_names(field)` [\[source\]](#)

Frozen set of all fields names (should only be used for many-one and many-many fields)

`author_data(author_ids=None)`[\[source\]](#)

Return author data as a dictionary with keys: name, sort, link

If no authors with the specified ids are found an empty dictionary is returned. If *author_ids* is *None*, data for all authors is returned.

`author_sort_from_authors(authors, key_func=<function make_change_case_func.<locals>.change_case>)`[\[source\]](#)

Given a list of authors, return the *author_sort* string for the authors, preferring the author sort associated with the author over the computed string.

`books_for_field(name, item_id)`[\[source\]](#)

Return all the books associated with the item identified by *item_id*, where the item belongs to the field name.

Returned value is a set of book ids, or the empty set if the item or the field does not exist.

`books_in_virtual_library(vl, search_restriction=None, virtual_fields=None)`[\[source\]](#)

Return the set of books in the specified virtual library

`compress_covers(book_ids, jpeg_quality=100, progress_callback=None)`[\[source\]](#)

Compress the cover images for the specified books. A compression quality of 100 will perform lossless compression, otherwise lossy compression.

The progress callback will be called with the *book_id* and the old and new sizes for each book that has been processed. If an error occurs, the new size will be a string with the error details.

`copy_cover_to(book_id, dest, use_hardlink=False, report_file_size=None)`[\[source\]](#)

Copy the cover to the file like object *dest*. Returns *False* if no cover exists or *dest* is the same file as the current cover. *dest* can also be a path in which case the cover is copied to it if and only if the path is different from the current path (taking case sensitivity into account).

`copy_format_to(book_id, fmt, dest, use_hardlink=False, report_file_size=None)`[\[source\]](#)

Copy the format *fmt* to the file like object *dest*

. If the specified format does not exist, raises **NoSuchFormat** error. `dest` can also be a path (to a file), in which case the format is copied to it, iff the path is different from the current path (taking case sensitivity into account).

`cover(book_id, as_file=False, as_image=False, as_path=False)`[\[source\]](#)

Return the cover image or None. By default, returns the cover as a bytestring.

WARNING: Using `as_path` will copy the cover to a temp file and return the path to the temp file. You should delete the temp file when you are done with it.

Parameters

- **as_file** – If True return the image as an open file object (a `SpooledTemporaryFile`)
- **as_image** – If True return the image as a `QImage` object
- **as_path** – If True return the image as a path pointing to a temporary file

`data_for_find_identical_books()`[\[source\]](#)

Return data that can be used to implement [find_identical_books\(\)](#) in a worker process without access to the db. See `db.utils` for an implementation.

`data_for_has_book()`[\[source\]](#)

Return data suitable for use in [has_book\(\)](#). This can be used for an implementation of [has_book\(\)](#) in a worker process without access to the db.

`delete_custom_book_data(name, book_ids=())`[\[source\]](#)

Delete data for `name`. By default deletes all data, if you only want to delete data for some book ids, pass in a list of book ids.

`embed_metadata(book_ids, only_fmts=None, report_error=None, report_progress=None)`[\[source\]](#)

Update metadata in all formats of the specified `book_ids` to current metadata in the database.

`fast_field_for(field_obj, book_id, default_value=None)`[\[source\]](#)

Same as `field_for`, except that it avoids the extra lookup to get the field object

`field_for(name, book_id, default_value=None)`[\[source\]](#)

Return the value of the field name for the book identified by `book_id`. If no such book exists or it has no defined value for the field name or no such field exists, then `default_value` is returned.

`default_value`

is not used for `title`, `title_sort`, `authors`, `author_sort` and `series_index`. This is because these always have values in the db. `default_value` is used for all custom columns.

The returned value for `is_multiple` fields are always tuples, even when no values are found (in other words, `default_value` is ignored). The exception is identifiers for which the returned value is always a dict. The returned tuples are always in link order, that is, the order in which they were created.

`field_ids_for(name, book_id)`[\[source\]](#)

Return the ids (as a tuple) for the values that the field name has on the book identified by `book_id`. If there are no values, or no such book, or no such field, an empty tuple is returned.

`find_identical_books(mi, search_restriction="", book_ids=None)`[\[source\]](#)

Finds books that have a superset of the authors in `mi` and the same title (title is fuzzy matched). See also [data_for_find_identical_books\(\)](#).

`format(book_id, fmt, as_file=False, as_path=False, preserve_filename=False)`[\[source\]](#)

Return the e-book format as a bytestring or *None* if the format doesn't exist, or we don't have permission to write to the e-book file.

Parameters

- **as_file** – If True the e-book format is returned as a file object. Note that the file object is a `SpooledTemporaryFile`, so if what you want to do is copy the format to another file, use [copy_format_to\(\)](#) instead for performance.
- **as_path** – Copies the format file to a temp file and returns the path to the temp file
- **preserve_filename** – If True and returning a path the filename is the same as that used in the library. Note that using this means that repeated calls yield the same temp file (which is re-created each time)

`format_abspath(book_id, fmt)`[\[source\]](#)

Return absolute path to the e-book file of format *format*. You should almost never use this, as it breaks the threadsafe promise of this API. Instead use, [copy_format_to\(\)](#).

Currently used only in calibredb list, the viewer, edit book, compare_format to original format, open with, bulk metadata edit and the catalogs (via `get_data_as_dict()`).

Apart from the viewer, open with and edit book, I don't believe any of the others do any file write I/O with the results of this call.

`format_hash(book_id, fmt)`[\[source\]](#)

Return the hash of the specified format for the specified book. The kind of hash is backend dependent, but is usually SHA-256.

`format_metadata(book_id, fmt, allow_cache=True, update_db=False)`[\[source\]](#)

Return the path, size and mtime for the specified format for the specified book. You should not use path unless you absolutely have to, since accessing it directly breaks the threadsafe guarantees of this API. Instead use the [copy_format_to\(\)](#) method.

Parameters

- **allow_cache** – If True
cached values are used, otherwise a slow filesystem access is done. The cache values could be out of date if access was performed to the filesystem outside of this API.
- **update_db** – If True
The max_size field of the database is updated for this book.

`formats(book_id, verify_formats=True)`[\[source\]](#)

Return tuple of all formats for the specified book. If *verify_formats* is True, verifies that the files exist on disk.

`get_categories(sort='name', book_ids=None, already_fixed=None, first_letter_sort=False)`[\[source\]](#)

Used internally to implement the Tag Browser

`get_custom_book_data(name, book_ids=(), default=None)`[\[source\]](#)

Get data for name. By default returns data for all book_ids, pass in a list of book ids if you only want some data. Returns a map of book_id to values. If a particular value could not be decoded, uses default for it.

`get_id_map(field)`[\[source\]](#)

Return a mapping of id numbers to values for the specified field. The field must be a many-one or many-many field, otherwise a ValueError is raised.

`get_ids_for_custom_book_data(name)`[\[source\]](#)

Return the set of book ids for which name has data.

`get_item_id(field, item_name)`[\[source\]](#)

Return the item id for item_name (case-insensitive)

`get_item_ids(field, item_names)`[\[source\]](#)

Return the item id for item_name (case-insensitive)

`get_item_name(field, item_id)`[\[source\]](#)

Return the item name for the item specified by item_id in the specified field. See also [get_id_map\(\)](#).

`get_metadata(book_id, get_cover=False, get_user_categories=True, cover_as_data=False)`[\[source\]](#)

Return metadata for the book identified by book_id as a [calibre.ebooks.metadata.book.base.Metadata](#) object. Note that the list of formats is not verified. If get_cover is True, the cover is returned, either a path to temp file as mi.cover or if cover_as_data is True then as mi.cover_data.

`get_next_series_num_for(series, field='series', current_indices=False)`[\[source\]](#)

Return the next series index for the specified series, taking into account the various preferences that control next series number generation.

Parameters

- **field** – The series-like field (defaults to the builtin series column)

- **current_indices** – If True, returns a mapping of book_id to current series_index value instead.

`get_proxy_metadata(book_id)[source]`

Like [get_metadata\(\)](#) except that it returns a ProxyMetadata object that only reads values from the database on demand. This is much faster than `get_metadata` when only a small number of fields need to be accessed from the returned metadata object.

`get_usage_count_by_id(field)[source]`

Return a mapping of id to usage count for all values of the specified field, which must be a many-one or many-many field.

`has_book(mi)[source]`

Return True iff the database contains an entry with the same title as the passed in Metadata object. The comparison is case-insensitive. See also [data_for_has_book\(\)](#).

`has_format(book_id, fmt)[source]`

Return True iff the format exists on disk

`has_id(book_id)[source]`

Return True iff the specified book_id exists in the db

`init()[source]`

Initialize this cache with data from the backend.

`multisort(fields, ids_to_sort=None, virtual_fields=None)[source]`

Return a list of sorted book ids. If *ids_to_sort* is None, all book ids are returned.

fields must be a list of 2-tuples of the form (field_name, ascending=True or False). The most significant field is the first 2-tuple.

`pref(name, default=None, namespace=None)[source]`

Return the value for the specified preference or the value specified as default if the preference is not set.

`read_backup(book_id)`[\[source\]](#)

Return the OPF metadata backup for the book as a bytestring or None if no such backup exists.

`remove_books(book_ids, permanent=False)`[\[source\]](#)

Remove the books specified by the *book_ids* from the database and delete their format files. If *permanent* is False, then the format files are placed in the recycle bin.

`remove_formats(formats_map, db_only=False)`[\[source\]](#)

Remove the specified formats from the specified books.

Parameters

- **formats_map** – A mapping of *book_id* to a list of formats to be removed from the book.
- **db_only** – If True, only remove the record for the format from the db, do not delete the actual format file from the filesystem.

`remove_items(field, item_ids, restrict_to_book_ids=None)`[\[source\]](#)

Delete all items in the specified field with the specified ids. Returns the set of affected book ids. *restrict_to_book_ids* is an optional set of books ids. If specified the items will only be removed from those books.

`rename_items(field, item_id_to_new_name_map, change_index=True, restrict_to_book_ids=None)`[\[source\]](#)

Rename items from a many-one or many-many field such as tags or series.

Parameters

- **change_index** – When renaming in a series-like field also change the *series_index* values.
- **restrict_to_book_ids** – An optional set of book ids for which the rename is to be performed, defaults to all books.

`restore_book(book_id, mi, last_modified, path, formats, annotations=())`[\[source\]](#)

Restore the book entry in the database for a book that already exists on the filesystem

`restore_original_format(book_id, original_fmt)`[\[source\]](#)

Restore the specified format from the previously saved ORIGINAL_FORMAT, if any. Return True on success. The ORIGINAL_FORMAT is deleted after a successful restore.

property `safe_read_lock`

A safe read lock is a lock that does nothing if the thread already has a write lock, otherwise it acquires a read lock. This is necessary to prevent DowngradeLockErrors, which can happen when updating the search cache in the presence of composite columns. Updating the search cache holds an exclusive lock, but searching a composite column involves reading field values via ProxyMetadata which tries to get a shared lock. There may be other scenarios that trigger this as well.

This property returns a new lock object on every access. This lock object is not recursive (for performance) and must only be used in a `with` statement as with

`cache.safe_read_lock:`
otherwise bad things will happen.

`save_original_format(book_id, fmt)`[\[source\]](#)

Save a copy of the specified format as ORIGINAL_FORMAT, overwriting any existing ORIGINAL_FORMAT.

`search(query, restriction='', virtual_fields=None, book_ids=None)`[\[source\]](#)

Search the database for the specified query, returning a set of matched book ids.

Parameters

- **restriction** – A restriction that is ANDed to the specified query. Note that restrictions are cached, therefore the search for a AND b will be slower than a with restriction b.
- **virtual_fields** – Used internally (virtual fields such as `on_device` to search over).
- **book_ids** – If not None, a set of book ids for which books will be searched instead of searching all books.

`set_conversion_options(options, fmt='PIPE')`[\[source\]](#)

`options` must be a map of the form `{book_id:conversion_options}`

`set_cover(book_id_data_map)`[\[source\]](#)

Set the cover for this book. The data can be either a QImage, QPixmap, file object or bytestring. It can also be None, in which case any existing cover is removed.

`set_field(name, book_id_to_val_map, allow_case_change=True, do_path_update=True)`[\[source\]](#)

Set the values of the field specified by name. Returns the set of all book ids that were affected by the change.

Parameters

- **book_id_to_val_map** – Mapping of book_ids to values that should be applied.
- **allow_case_change** – If True, the case of many-one or many-many fields will be changed. For example, if a book has the tag tag1 and you set the tag for another book to Tag1 then the both books will have the tag Tag1 if allow_case_change is True, otherwise they will both have the tag tag1.
- **do_path_update** – Used internally, you should never change it.

`set_metadata(book_id, mi, ignore_errors=False, force_changes=False, set_title=True, set_authors=True, allow_case_change=False)`[\[source\]](#)

Set metadata for the book *id* from the *Metadata* object *mi*

Setting force_changes=True will force set_metadata to update fields even if mi contains empty values. In this case, 'None' is distinguished from 'empty'. If mi.XXX is None, the XXX is not replaced, otherwise it is. The tags, identifiers, and cover attributes are special cases. Tags and identifiers cannot be set to None so they will always be replaced if force_changes is true. You must ensure that mi contains the values you want the book to have. Covers are always changed if a new cover is provided, but are never deleted. Also note that force_changes has no effect on setting title or authors.

`set_pref(name, val, namespace=None)`[\[source\]](#)

Set the specified preference to the specified value. See also [pref\(\)](#).

`tags_older_than(tag, delta=None, must_have_tag=None, must_have_authors=None)`[\[source\]](#)

Return the ids of all books having the tag tag that are older than the specified time. tag comparison is case insensitive.

Parameters

- **delta** – A timedelta object or None. If None, then all ids with the tag are returned.
- **must_have_tag** – If not None the list of matches will be restricted to books that have this tag
- **must_have_authors** – A list of authors. If not None the list of matches will be restricted to books that have these authors (case insensitive).

`user_categories_for_books(book_ids, proxy_metadata_map=None)`[\[source\]](#)

Return the user categories for the specified books. `proxy_metadata_map` is optional and is useful for a performance boost, in contexts where a `ProxyMetadata` object for the books already exists. It should be a mapping of `book_ids` to their corresponding `ProxyMetadata` objects.



[modules](#) [next](#) | [previous](#) | [Start](#) » [Setting up a calibre development environment](#) » [API documentation for the e-book editing tools](#)

API documentation for the e-book editing tools

The e-book editing tools consist of a [calibre.ebooks.oeb.polish.container.Container](#) object that represents a book as a collection of HTML + resource files, and various tools that can be used to perform operations on the container. All the tools are in the form of module level functions in the various `calibre.ebooks.oeb.polish.*` modules.

You obtain a container object for a book at a path like this:

```
from
calibre.ebooks.oeb.polish.container

import

get_container

container

=

get_container
(
'Path to book file'
,

tweak_mode
=
True
)
```

If you are writing a plugin for the E-book editor, you get the current container for the book being edited like this:

```
from
```



```
calibre.gui2.tweak_book
```

```
import
```

```
current_container
```

```
container
```

```
=
```

```
current_container  
( )
```

```
if
```

```
container
```

```
is
```

```
None
```

```
:
```

```
report_error
```

```
# No book has been opened yet
```

The Container object

```
class calibre.ebooks.oeb.polish.container.Container(rootpath, opfpath, log,  
clone_data=None)\[source\]
```

A container represents an open e-book as a folder full of files and an OPF file. There are two important concepts:

- The root folder. This is the base of the e-book. All the e-books files are inside this folder or in its sub-folders.
- Names: These are paths to the books' files relative to the root folder. They always contain POSIX separators and are unquoted. They can be thought of as canonical identifiers for files in the book. Most methods on the container object work with names. Names are always in the NFC Unicode normal form.

- Clones: the container object supports efficient on-disk cloning, which is used to implement checkpoints in the e-book editor. In order to make this work, you should never access files on the filesystem directly. Instead, use [raw_data\(\)](#) or [open\(\)](#) to read/write to component files in the book.

When converting between hrefs and names use the methods provided by this class, they assume all hrefs are quoted.

`abspath_to_name(fullpath, root=None)`[\[source\]](#)

Convert an absolute path to a canonical name relative to **root**

Parameters **root** – The base folder. By default the root for this container object is used.

`add_file(name, data, media_type=None, spine_index=None, modify_name_if_needed=False, process_manifest_item=None)`[\[source\]](#)

Add a file to this container. Entries for the file are automatically created in the OPF manifest and spine (if the file is a text document)

`add_name_to_manifest(name, process_manifest_item=None)`[\[source\]](#)

Add an entry to the manifest for a file with the specified name. Returns the manifest id.

`add_properties(name, *properties)`[\[source\]](#)

Add the specified properties to the manifest item identified by name.

`apply_unique_properties(name, *properties)`[\[source\]](#)

Ensure that the specified properties are set on only the manifest item identified by name. You can pass None as the name to remove the property from all items.

`book_type = 'oeb'`

The type of book (epub for EPUB files and azw3 for AZW3 files)

`commit(outpath=None, keep_parsed=False)`[\[source\]](#)

Commit all dirtied parsed objects to the filesystem and write out the e-book file at outpath.

Parameters

- **output** – The path to write the saved e-book file to. If None, the path of the original book file is used.
- **keep_parsed** – If True the parsed representations of committed items are kept in the cache.

`commit_item(name, keep_parsed=False)`[\[source\]](#)

Commit a parsed object to disk (it is serialized and written to the underlying file). If `keep_parsed` is True the parsed representation is retained in the cache. See also: [parsed\(\)](#)

`dirty(name)`[\[source\]](#)

Mark the parsed object corresponding to `name` as dirty. See also: [parsed\(\)](#)

`exists(name)`[\[source\]](#)

True iff a file/folder corresponding to the canonical name exists. Note that this function suffers from the limitations of the underlying OS filesystem, in particular case (in)sensitivity. So on a case insensitive filesystem this will return True even if the case of name is different from the case of the underlying filesystem file. See also [has_name\(\)](#)

`filesize(name)`[\[source\]](#)

Return the size in bytes of the file represented by the specified canonical name. Automatically handles dirtied parsed objects. See also: [parsed\(\)](#)

`generate_item(name, id_prefix=None, media_type=None, unique_href=True)`[\[source\]](#)

Add an item to the manifest with href derived from the given name. Ensures uniqueness of href and id automatically. Returns generated item.

`get_file_path_for_processing(name, allow_modification=True)`[\[source\]](#)

Similar to `open()` except that it returns a file path, instead of an open file object.

property `guide_type_map`

Mapping of guide type to canonical name

`has_name(name)`[\[source\]](#)

Return True iff a file with the same canonical name as that specified exists. Unlike `exists()` this method is always case-sensitive.

`href_to_name(href, base=None)`[\[source\]](#)

Convert an href (relative to base) to a name. base must be a name or None, in which case self.root is used.

`insert_into_xml(parent, item, index=None)`[\[source\]](#)

Insert item into parent (or append if index is None), fixing indentation. Only works with self closing items.

`is_dir = False`

If this container represents an unzipped book (a directory)

`iterlinks(name, get_line_numbers=True)`[\[source\]](#)

Iterate over all links in name. If *get_line_numbers* is True the yields results of the form (link, line_number, offset). Where line_number is the line_number at which the link occurs and offset is the number of characters from the start of the line. Note that offset could actually encompass several lines if not zero.

`make_name_unique(name)`[\[source\]](#)

Ensure that *name* does not already exist in this book. If it does, return a modified version that does not exist.

`manifest_has_name(name)`[\[source\]](#)

Return True if the manifest has an entry corresponding to name

property manifest_id_map

Mapping of manifest id to canonical names

`manifest_items_of_type(predicate)`[\[source\]](#)

The names of all manifest items whose media-type matches predicate. *predicate* can be a set, a list, a string or a function taking a single argument, which will be called with the media-type.

`manifest_items_with_property(property_name)`[\[source\]](#)

All manifest items that have the specified property

property manifest_type_map

Mapping of manifest media-type to list of canonical names of that media-type

property mi

The metadata of this book as a Metadata object. Note that this object is constructed on the fly every time this property is requested, so use it sparingly.

name_to_abspath(*name*)[\[source\]](#)

Convert a canonical name to an absolute OS dependent path

name_to_href(*name*, *base=None*)[\[source\]](#)

Convert a name to a href relative to base, which must be a name or None in which case self.root is used as the base

property names_that_must_not_be_changed

Set of names that must never be renamed. Depends on the e-book file format.

property names_that_must_not_be_removed

Set of names that must never be deleted from the container. Depends on the e-book file format.

property names_that_need_not_be_manifested

Set of names that are allowed to be missing from the manifest. Depends on the e-book file format.

open(*name*, *mode='rb'*)[\[source\]](#)

Open the file pointed to by name for direct read/write. Note that this will commit the file if it is dirtied and remove it from the parse cache. You must finish with this file before accessing the parsed version of it again, or bad things will happen.

property opf

The parsed OPF file

opf_get_or_create(*name*)[\[source\]](#)

Convenience method to either return the first XML element with the specified name or create it under the opf:package element and then return it, if it does not already exist.

property opf_version

The version set on the OPF's <package> element

property opf_version_parsed

The version set on the OPF's <package> element as a tuple of integers

opf_xpath(*expr*)[[source](#)]

Convenience method to evaluate an XPath expression on the OPF file, has the opf: and dc: namespace prefixes pre-defined.

parsed(*name*)[[source](#)]

Return a parsed representation of the file specified by name. For HTML and XML files an lxml tree is returned. For CSS files a css_parser stylesheet is returned. Note that parsed objects are cached for performance. If you make any changes to the parsed object, you must call [dirty\(\)](#)

so that the container knows to update the cache. See also [replace\(\)](#).

raw_data(*name*, *decode=True*, *normalize_to_nfc=True*)[[source](#)]

Return the raw data corresponding to the file specified by name

Parameters

- **decode** – If True and the file has a text based MIME type, decode it and return a unicode object instead of raw bytes.
- **normalize_to_nfc** – If True the returned unicode object is normalized to the NFC normal form as is required for the EPUB and AZW3 file formats.

relpath(*path*, *base=None*)[[source](#)]

Convert an absolute path (with os separators) to a path relative to base (defaults to self.root). The relative path is *not* a name. Use [abspath_to_name\(\)](#) for that.

remove_from_spine(*spine_items*, *remove_if_no_longer_in_spine=True*)[[source](#)]

Remove the specified items (by canonical name) from the spine. If `remove_if_no_longer_in_spine` is True, the items are also deleted from the book, not just from the spine.

`remove_from_xml(item)`[\[source\]](#)

Removes item from parent, fixing indentation (works only with self closing items)

`remove_item(name, remove_from_guide=True)`[\[source\]](#)

Remove the item identified by name from this container. This removes all references to the item in the OPF manifest, guide and spine as well as from any internal caches.

`rename(current_name, new_name)`[\[source\]](#)

Renames a file from `current_name` to `new_name`. It automatically rebases all links inside the file if the folder the file is in changes. Note however, that links are not updated in the other files that could reference this file. This is for performance, such updates should be done once, in bulk.

`replace(name, obj)`[\[source\]](#)

Replace the parsed object corresponding to `name` with `obj`, which must be a similar object, i.e. an lxml tree for HTML/XML or a css_parser stylesheet for a CSS file.

`replace_links(name, replace_func)`[\[source\]](#)

Replace all links in `name` using `replace_func`, which must be a callable that accepts a URL and returns the replaced URL. It must also have a 'replaced' attribute that is set to True if any actual replacement is done. Convenient ways of creating such callables are using the

LinkReplacer

and
classes.

LinkRebaser

`serialize_item(name)`[\[source\]](#)

Convert a parsed object (identified by canonical name) into a bytestring. See [parsed\(\)](#).

`set_spine(spine_items)`[\[source\]](#)

Set the spine to be `spine_items` where `spine_items` is an iterable of the form (name, linear). Will raise an error if one of the names is not present in the manifest.

property spine_items

An iterator yielding the path for every item in the books' spine. See also: [spine_iter](#) and [spine_items](#)

.

property spine_iter

An iterator that yields item, name is_linear for every item in the books' spine. item is the lxml element, name is the canonical file name and is_linear is True if the item is linear. See also:

[spine_names](#)

and

[spine_items](#)

.

property spine_names

An iterator yielding name and is_linear for every item in the books' spine. See also: [spine_iter](#) and [spine_items](#)

.

Managing component files in a container

calibre.ebooks.oeb.polish.replace.replace_links(*container*, *link_map*, *frag_map*=<function <lambda>>, *replace_in_opf*=False)[\[source\]](#)

Replace links to files in the container. Will iterate over all files in the container and change the specified links in them.

Parameters

- **link_map** – A mapping of old canonical name to new canonical name. For example:

```
{'images/old.png':  
    'images/new.png'}
```
- **frag_map** – A callable that takes two arguments (name, anchor) and returns a new anchor. This is useful if you need to change the anchors in HTML files. By default, it does nothing.
- **replace_in_opf** – If False, links are not replaced in the OPF file.

calibre.ebooks.oeb.polish.replace.rename_files(*container, file_map*)[\[source\]](#)

Rename files in the container, automatically updating all links to them.

Parameters

- file_map** – A mapping of old canonical name to new canonical name, for example: `{'text/chapter1.html': 'chapter1.html'}`.

calibre.ebooks.oeb.polish.replace.get_recommended_folders(*container, names*)[\[source\]](#)

Return the folders that are recommended for the given filenames. The recommendation is based on where the majority of files of the same type are located in the container. If no files of a particular type are present, the recommended folder is assumed to be the folder containing the OPF file.

Pretty printing and auto fixing parse errors

```
calibre.ebooks.oeb.polish.pretty.fix_html(container, raw)\[source\]
```

Fix any parsing errors in the HTML represented as a string in raw. Fixing is done using the HTML5 parsing algorithm.

calibre.ebooks.oeb.polish.pretty.fix_all_html(*container*)[\[source\]](#)

Fix any parsing errors in all HTML files in the container. Fixing is done using the HTML5 parsing algorithm.

```
calibre.ebooks.oeb.polish.pretty.pretty_html(container, name, raw)[source]
```

Pretty print the HTML represented as a string in raw

calibre.ebooks.oeb.polish.pretty.pretty_css(*container, name, raw*) [\[source\]](#)

Pretty print the CSS represented as a string in raw

calibre.ebooks.oeb.polish.pretty.pretty_xml(*container, name, raw*)[\[source\]](#)

Pretty print the XML represented as a string in raw. If name is the name of the OPF, extra OPF-specific prettying is performed.

```
calibre.ebooks.oeb.polish.pretty.pretty_all(container)[source]
```

Pretty print all HTML/CSS/XML files in the container

Managing book jackets

`calibre.ebooks.oeb.polish.jacket.remove_jacket(container)`[\[source\]](#)

Remove an existing jacket, if any. Returns False if no existing jacket was found.

`calibre.ebooks.oeb.polish.jacket.add_or_replace_jacket(container)`[\[source\]](#)

Either create a new jacket from the book's metadata or replace an existing jacket. Returns True if an existing jacket was replaced.

Splitting and merging of files

`calibre.ebooks.oeb.polish.split.split(container, name, loc_or_xpath, before=True, totals=None)`[\[source\]](#)

Split the file specified by name at the position specified by *loc_or_xpath*. Splitting automatically migrates all links and references to the affected files.

Parameters

- **loc_or_xpath** – Should be an XPath expression such as `//h:div[@id="split_here"]`. Can also be a *loc* which is used internally to implement splitting in the preview panel.
- **before** – If True the split occurs before the identified element otherwise after it.
- **totals** – Used internally

`calibre.ebooks.oeb.polish.split.multisplit(container, name, xpath, before=True)`[\[source\]](#)

Split the specified file at multiple locations (all tags that match the specified XPath expression). See also: [split\(\)](#). Splitting automatically migrates all links and references to the affected files.

Parameters **before** – If True the splits occur before the identified element otherwise after it.

`calibre.ebooks.oeb.polish.split.merge(container, category, names, master)`[\[source\]](#)

Merge the specified files into a single file, automatically migrating all links and references to the affected files. The file must all either be HTML or CSS files.

Parameters

- **category** – Must be either 'text' for HTML files or 'styles' for CSS files
- **names** – The list of files to be merged
- **master** – Which of the merged files is the *master* file, that is, the file that will remain after merging.

Managing covers

`calibre.ebooks.oeb.polish.cover.set_cover(container, cover_path, report=None, options=None)` [\[source\]](#)

Set the cover of the book to the image pointed to by `cover_path`.

Parameters

- **cover_path** – Either the absolute path to an image file or the canonical name of an image in the book. When using an image in the book, you must also set options, see below.
- **report** – An optional callable that takes a single argument. It will be called with information about the tasks being processed.
- **options** – None or a dictionary that controls how the cover is set. The dictionary can have entries: **keep_aspect**: True or False (Preserve aspect ratio of covers in EPUB) **no_svg**: True or False (Use an SVG cover wrapper in the EPUB titlepage) **existing**: True or False (cover_path refers to an existing image in the book)

`calibre.ebooks.oeb.polish.cover.mark_as_cover(container, name)` [\[source\]](#)

Mark the specified image as the cover image.

`calibre.ebooks.oeb.polish.cover.mark_as_titlepage(container, name, move_to_start=True)` [\[source\]](#)

Mark the specified HTML file as the titlepage of the EPUB.

Parameters **move_to_start** – If True the HTML file is moved to the start of the spine

Working with CSS

`calibre.ebooks.oeb.polish.fonts.change_font(container, old_name, new_name=None)`[\[source\]](#)

Change a font family from `old_name` to `new_name`. Changes all occurrences of the font family in stylesheets, style tags and style attributes. If the `old_name` refers to an embedded font, it is removed. You can set `new_name` to `None` to remove the font family instead of changing it.

`calibre.ebooks.oeb.polish.css.remove_unused_css(container, report=None, remove_unused_classes=False, merge_rules=False, merge_rules_with_identical_properties=False, remove_unreferenced_sheets=False)`[\[source\]](#)

Remove all unused CSS rules from the book. An unused CSS rule is one that does not match any actual content.

Parameters

- **report** – An optional callable that takes a single argument. It is called with information about the operations being performed.
- **remove_unused_classes** – If True, class attributes in the HTML that do not match any CSS rules are also removed.
- **merge_rules** – If True, rules with identical selectors are merged.
- **merge_rules_with_identical_properties** – If True, rules with identical properties are merged.
- **remove_unreferenced_sheets** – If True, stylesheets that are not referenced by any content are removed

`calibre.ebooks.oeb.polish.css.filter_css(container, properties, names=())`[\[source\]](#)

Remove the specified CSS properties from all CSS rules in the book.

Parameters

- **properties** – Set of properties to remove. For example: {'font-family',

```
'color'}
```

```
.
```

- **names** – The files from which to remove the properties. Defaults to all HTML and CSS files in the book.

Working with the Table of Contents

`calibre.ebooks.oeb.polish.toc.from_xpaths(container, xpaths)`[\[source\]](#)

Generate a Table of Contents from a list of XPath expressions. Each expression in the list corresponds to a level of the generate ToC. For example: `['//h:h1',`

`//h:h2',`

`//h:h3']`

will generate a three level Table of Contents from the `<h1>`
, `<h2>`
and `<h3>`
tags.

`calibre.ebooks.oeb.polish.toc.from_links(container)`[\[source\]](#)

Generate a Table of Contents from links in the book.

`calibre.ebooks.oeb.polish.toc.from_files(container)`[\[source\]](#)

Generate a Table of Contents from files in the book.

`calibre.ebooks.oeb.polish.toc.create_inline_toc(container, title=None)`[\[source\]](#)

Create an inline (HTML) Table of Contents from an existing NCX Table of Contents.

Parameters **title** – The title for this table of contents.

Edit book tool

`class calibre.gui2.tweak_book.plugin.Tool`[\[source\]](#)

Bases:

object

The base class for individual tools in an Edit Book plugin. Useful members include:

- `self.plugin`
: A reference to the [calibre.customize.Plugin](#) object to which this tool belongs.
- `self.` [boss](#)
- `self.` [gui](#)

Methods that must be overridden in sub classes:

- [create_action\(\)](#)
- [register_shortcut\(\)](#)

`name = None`

Set this to a unique name it will be used as a key

`allowed_in_toolbar = True`

If True the user can choose to place this tool in the plugins toolbar

`allowed_in_menu = True`

If True the user can choose to place this tool in the plugins menu

`toolbar_button_popup_mode = 'delayed'`

The popup mode for the menu (if any) of the toolbar button. Possible values are 'delayed', 'instant', 'button'

property boss

The [calibre.gui2.tweak_book.boss.Boss](#) object. Used to control the user interface.

property gui

The main window of the user interface

property `current_container`

Return the current [calibre.ebooks.oeb.polish.container.Container](#) object that represents the book being edited.

`register_shortcut(qaction, unique_name, default_keys=(), short_text=None, description=None, **extra_data)` [\[source\]](#)

Register a keyboard shortcut that will trigger the specified `qaction`. This keyboard shortcut will become automatically customizable by the user in the Keyboard shortcuts section of the editor preferences.

Parameters

- **qaction** – A `QAction` object, it will be triggered when the configured key combination is pressed by the user.
- **unique_name** – A unique name for this shortcut/action. It will be used internally, it must not be shared by any other actions in this plugin.
- **default_keys** – A list of the default keyboard shortcuts. If not specified no default shortcuts will be set. If the shortcuts specified here conflict with either builtin shortcuts or shortcuts from user configuration/other plugins, they will be ignored. In that case, users will have to configure the shortcuts manually via Preferences. For example: `default_keys=('Ctrl+J', 'F9')`.
- **short_text** – An optional short description of this action. If not specified the text from the `QAction` will be used.
- **description** – An optional longer description of this action, it will be used in the preferences entry for this shortcut.

`create_action(for_toolbar=True)` [\[source\]](#)

Create a `QAction` that will be added to either the plugins toolbar or the plugins menu depending on `for_toolbar`. For example:


```
def
create_action
(
self
,

for_toolbar
=
True
):

ac
=
QAction
(
get_icons
(
'myicon.png'
),

'Do something'
)

if

for_toolbar
:

# We want the toolbar button to have a popup menu

menu
=
QMenu
()
```

```
ac
.
setMenu
(
menu
)

menu
.
addAction
(
'Do something else'
)

subaction
=

menu
.
addAction
(
'And another'
)

# Register a keyboard shortcut for this toolbar action be

# careful to do this for only one of the toolbar action or

# the menu action, not both.

self
.
register_shortcut
```

```
(
ac
,

'some-unique-name'
,

default_keys
=
(
'Ctrl+K'
,))

return

ac
```

See also

Method

[register_shortcut\(\)](#)

.

Controlling the editor's user interface

The e-book editor's user interface is controlled by a single global *Boss* object. This has many useful methods that can be used in plugin code to perform common tasks.

`class calibre.gui2.tweak_book.boss.Boss(parent, notify=None)`[\[source\]](#)

`add_savepoint(msg)`[\[source\]](#)

Create a restore checkpoint with the name specified as `msg`

`apply_container_update_to_gui(mark_as_modified=True)`[\[source\]](#)

Update all the components of the user interface to reflect the latest data in the current book container.

Parameters

mark_as_modified – If True, the book will be marked as modified, so the user will be prompted to save it when quitting.

`close_editor(name)`[\[source\]](#)

Close the editor that is editing the file specified by name

`commit_all_editors_to_container()`[\[source\]](#)

Commit any changes that the user has made to files open in editors to the container. You should call this method before performing any actions on the current container

property `currently_editing`

Return the name of the file being edited currently or None if no file is being edited

`edit_file(name, syntax=None, use_template=None)`[\[source\]](#)

Open the file specified by name in an editor

Parameters

- **syntax** – The media type of the file, for example, 'text/html'. If not specified it is guessed from the file extension.
- **use_template** – A template to initialize the opened editor with

`open_book(path=None, edit_file=None, clear_notify_data=True, open_folder=False, search_text=None)`[\[source\]](#)

Open the e-book at path for editing. Will show an error if the e-book is not in a supported format or the current book has unsaved changes.

Parameters **edit_file** – The name of a file inside the newly opened book to start editing. Can also be a list of names.

`rewind_savepoint()`[\[source\]](#)

Undo the previous creation of a restore checkpoint, useful if you create a checkpoint, then abort the operation with no changes

`save_book()`[\[source\]](#)

Save the book. Saving is performed in the background

`set_modified()`[\[source\]](#)

Mark the book as having been modified

`show_current_diff(allow_revert=True, to_container=None)`[\[source\]](#)

Show the changes to the book from its last checkpointed state

Parameters

- **allow_revert** – If True the diff dialog will have a button to allow the user to revert all changes
- **to_container** – A container object to compare the current container to. If None, the previously checkpointed container is used

`show_editor(name)`[\[source\]](#)

Show the editor that is editing the file specified by name

`sync_preview_to_editor()`[\[source\]](#)

Sync the position of the preview panel to the current cursor position in the current editor

Digital Rights Management (DRM)

Digital Rights Management (DRM) is a generic term for access control technologies that can be used by hardware manufacturers, publishers, copyright holders and individuals to try to impose limitations on the usage of digital content and devices. It is also, sometimes, disparagingly described as Digital Restrictions Management. The term is used to describe any technology which inhibits uses (legitimate or otherwise) of digital content that were not desired or foreseen by the content provider. The term generally doesn't refer to other forms of copy protection which can be circumvented without modifying the file or device, such as serial numbers or key-files. It can also refer to restrictions associated with specific instances of digital works or devices. DRM technologies attempt to control use of digital media by preventing access, copying or conversion to other formats by end users. See [Wikipedia](https://en.wikipedia.org/wiki/Digital_rights_management) [https://en.wikipedia.org/wiki/Digital_rights_management].

What does DRM imply for me personally?

When you buy an e-book with DRM you don't really own it but have purchased the permission to use it in a manner dictated to you by the seller. DRM limits what you can do with e-books you have "bought". Often people who buy books with DRM are unaware of the extent of these restrictions. These restrictions prevent you from reformatting the e-book to your liking, including making stylistic changes like adjusting the font sizes, although there is software that empowers you to do such things for non DRM books. People are often surprised that an e-book they have bought in a particular format cannot be converted to another format if the e-book has DRM. So if you have an Amazon Kindle and buy a book sold by Barnes and Nobles, you should know that if that e-book has DRM you will not be able to read it on your Kindle. Notice that I am talking about a book you buy, not steal or pirate but BUY.

What does DRM do for authors?

Publishers of DRMed e-books argue that the DRM is all for the sake of authors and to protect their artistic integrity and prevent piracy. But DRM does NOT prevent piracy. People who want to pirate content or use pirated content still do it and succeed. The three major DRM schemes for e-books today are run by Amazon, Adobe and Barnes and Noble and all three DRM schemes have been cracked. All DRM does is inconvenience legitimate users. It can be argued that it actually harms authors as people who would have bought the book choose to find a pirated version as they are not willing to put up with DRM. Those that would pirate in the absence of DRM do so in its presence as well. To reiterate, the key point is that DRM *does*

not prevent piracy. So DRM is not only pointless and harmful to buyers of e-books but also a waste of money.

DRM and freedom

Although digital content can be used to make information as well as creative works easily available to everyone and empower humanity, this is not in the interests of some publishers who want to steer people away from this possibility of freedom simply to maintain their relevance in world developing so fast that they can't keep up.

Why does calibre not support DRM?

calibre is open source software while DRM by its very nature is closed. If calibre were to support opening or viewing DRM files it could be trivially modified to be used as a tool for DRM removal which is illegal under today's laws. Open source software and DRM are a clash of principles. While DRM is all about controlling the user, open source software is about empowering the user. The two simply can not coexist.

What is calibre's view on content providers?

We firmly believe that authors and other content providers should be compensated for their efforts, but DRM is not the way to go about it. We are developing this database of DRM-free e-books from various sources to help you find DRM-free alternatives and to help independent authors and publishers of DRM-free e-books publicize their content. We hope you will find this useful and we request that you do not pirate the content made available to you here.

How can I help fight DRM?

As somebody who reads and buys e-books you can help fight DRM. Do not buy e-books with DRM. There are some publishers who publish DRM-free e-books. Make an effort to see if they carry the e-book you are looking for. If you like books by certain independent authors that sell DRM-free e-books and you can afford it make donations to them. This is money well spent as their e-books tend to be cheaper (there may be exceptions) than the ones you would buy from publishers of DRMed books and would probably work on all devices you own in the future saving you the cost of buying the e-book again. Do not discourage publishers and authors of DRM-free e-books by pirating their content. Content providers deserve compensation for their efforts. Do not punish them for trying to make your reading experience better by making available DRM-free e-books. In the long run this is detrimental to you. If you have bought books from sellers that carry both DRMed as well as DRM-free books, not knowing if they carry DRM or not make it a point to leave a comment or review on the website informing future buyers of its DRM status. Many sellers do not think it important to clearly indicate to their buyers if an e-book carries DRM or not. [Here](https://www.defectivebydesign.org/guide/ebooks) [https://www.defectivebydesign.org/guide/ebooks] you will find a guide to DRM-free living.

Glossary

RSS

RSS (*Really Simple Syndication*) is a web feed format that is used to publish frequently updated content, like news articles, blog posts, etc. It is a format that is particularly suited to being read by computers, and is therefore the preferred way of getting content from the web into an e-book. There are many other feed formats in use on the Internet, and calibre understands most of them. In particular, it has good support for the *ATOM* format, which is commonly used for blogs.

recipe

A recipe is a set of instructions that teach calibre how to convert an online news source, such as a magazine or a blog, into an e-book. A recipe is essentially [Python](https://www.python.org) code. As such, it is capable of converting arbitrarily complex news sources into e-books. At the simplest level, it is just a set of variables, such as URLs, that give calibre enough information to go out onto the Internet and download the news.

HTML

HTML (*Hyper Text Mark-Up Language*), a subset of Standard Generalized Mark-Up Language (SGML) for electronic publishing, is the specific standard used for the World Wide Web.

CSS

CSS (*Cascading Style Sheets*) is a language used to describe how an [HTML](#) document should be rendered (visual styling).

API

API (*Application Programming Interface*) is a source code interface that a library provides to support requests for services to be made of it by computer programs.

LRF

LRF The e-book format that is read by the SONY e-book readers.

URL

URL (*Uniform Resource Locator*) for example: `http://example.com`

regexp

Regular expressions provide a concise and flexible means for identifying strings of text of interest, such as particular characters, words, or patterns of characters. See [the tutorial](#) for an introduction to regular expressions.

Python Module Index

[c](#)

c

calibre

[calibre.customize](#)

Defines various abstract base classes that can be subclassed to create plugins.

[calibre.customize.conversion](#)

[calibre.db.cache](#)

The API accessing and manipulating a calibre library.

[calibre.devices.interface](#)

[calibre.ebooks.metadata.book.base](#)

[calibre.ebooks.metadata.sources.base](#)

[calibre.ebooks.oeb.polish.container](#)

The container object used to represent a book as a collection of its constituent HTML files.

[calibre.ebooks.oeb.polish.cover](#)

[calibre.ebooks.oeb.polish.css](#)

[calibre.ebooks.oeb.polish.jacket](#)

[calibre.ebooks.oeb.polish.pretty](#)

[calibre.ebooks.oeb.polish.replace](#)

[calibre.ebooks.oeb.polish.split](#)

[calibre.ebooks.oeb.polish.toc](#)

[calibre.gui2.tweak_book.boss](#)

[calibre.utils.formatter_functions](#)

[calibre.web.feeds.news](#)

*The API for writing recipes is defined by the
:class: `BasicNewsRecipe`*

All modules for which code is available

- [PyQt5.QtWidgets](#)
- [calibre.customize](#)
 - [calibre.customize.conversion](#)
- [calibre.db.cache](#)
- [calibre.devices.interface](#)
- [calibre.devices.usbms.cli](#)
- [calibre.devices.usbms.device](#)
- [calibre.devices.usbms.driver](#)
- [calibre.ebooks.metadata.book.base](#)
- [calibre.ebooks.metadata.sources.base](#)
- [calibre.ebooks.oeb.polish.container](#)
- [calibre.ebooks.oeb.polish.cover](#)
- [calibre.ebooks.oeb.polish.css](#)
- [calibre.ebooks.oeb.polish.fonts](#)
- [calibre.ebooks.oeb.polish.jacket](#)
- [calibre.ebooks.oeb.polish.pretty](#)
- [calibre.ebooks.oeb.polish.replace](#)
- [calibre.ebooks.oeb.polish.split](#)
- [calibre.ebooks.oeb.polish.toc](#)
- [calibre.gui2.actions](#)
- [calibre.gui2.preferences](#)

- [calibre.gui2.tweak book.boss](#)
- [calibre.gui2.tweak book.plugin](#)
- [calibre.utils.formatter functions](#)
- [calibre.web.feeds.news](#)

Source code for calibre.customize

```
__license__

=

'GPL v3'

__copyright__

=

'2008, Kovid Goyal <kovid at kovidgoyal.net>'


import

os

,

sys

,

zipfile

,

importlib

,

enum


from

calibre.constants

import
```

```
numeric_version
',

iswindows
',

ismacos

from

calibre.ptempfile

import

PersistentTemporaryFile

if

iswindows
:

platform

=

'windows'

elif

ismacos
:

platform

=

'osx'

else
```



```
:  
  
platform  
  
=  
  
'linux'  
  
class  
  
PluginNotFound  
(  
    ValueError  
):  
  
    pass  
  
class  
  
InvalidPlugin  
(  
    ValueError  
):  
  
    pass  
  
class  
  
PluginInstallationType  
(  
    enum  
    .  
    IntEnum  
):
```

EXTERNAL

=

1

SYSTEM

=

2

BUILTIN

=

3

[\[docs\]](#)

class

Plugin

:

{{{

'''

A calibre plugin. Useful members include:

** ``self.installation\_type``: Stores how the plugin was installed.*

** ``self.plugin\_path``: Stores path to the ZIP file that contains*

this plugin or None if it is a builtin

plugin

** ``self.site\_customization``: Stores a customization string entered
by the user.*

Methods that should be overridden in sub classes:

** :meth:``initialize``*

** :meth:``customization\_help``*

Useful methods:

** :meth:``temporary\_file``*

** :meth:``\_\_enter\_\_``*

** :meth:``load\_resources``*

'''

#: List of platforms this plugin works on.

#: For example: ``['windows', 'osx', 'linux']``

supported_platforms

=

[]

#: The name of this plugin. You must set it something other

#: than Trivial Plugin for it to work.

name

=

'Trivial Plugin'

#: The version of this plugin as a 3-tuple (major, minor, revision)

version

=

(

1

,

0

,

0

)

#: A short string describing what this plugin does

description

=

—

```
(  
'Does absolutely nothing'  
)
```

#: The author of this plugin

author

=

```
—  
(  
'Unknown'  
)
```

#: When more than one plugin exists for a filetype,

#: the plugins are run in order of decreasing priority.

#: Plugins with higher priority will be run first.

#: The highest possible priority is ``sys.maxsize``.

#: Default priority is 1.

priority

=

1

#: The earliest version of calibre this plugin requires

minimum_calibre_version

=

(

0

,

4

,

118

)

#: The way this plugin is installed

installation_type

=

None

#: If False, the user will not be able to disable this plugin. Use with

#: care.

can_be_disabled

=

True

#: The type of this plugin. Used for categorizing plugins in the

#: GUI

type

=

-

(
'Base'
)

def

__init__

(

self

,

plugin_path

):

self

.

plugin_path

=

plugin_path

self

.

site_customization

=

None

[\[docs\]](#)

def

initialize

(
self
):

'''

Called once when calibre plugins are initialized. Plugins are re-initialized every time a new plugin is added. Also note that if the plugin is run in a worker process, such as for adding books, then the plugin will be initialized for every new worker process.

Perform any plugin specific initialization here, such as extracting resources from the plugin ZIP file. The path to the ZIP file is available as ``self.plugin\_path``.

*Note that ``self.site\_customization`` is ****not**** available at this point.*

'''

pass

[\[docs\]](#)

def

config_widget

(

self

):

'''

Implement this method and :meth:`save\_settings` in your plugin to use a custom configuration dialog, rather than relying on the simple string based default customization.

This method, if implemented, must return a QWidget. The widget can have an optional method validate() that takes no arguments and is called immediately after the user clicks OK. Changes are applied if and only if the method returns True.

If for some reason you cannot perform the configuration at this time, return a tuple of two strings (message, details), these will be displayed as a warning dialog to the user and the process will be aborted.

'''

raise

NotImplementedError

```
()
```

[\[docs\]](#)

```
def
```

```
    save_settings
```

```
    (
```

```
    self
```

```
    ,
```

```
    config_widget
```

```
):
```

```
    """
```

```
        Save the settings specified by the user with config_widget.
```

```
        :param config_widget: The widget returned by :meth:`config_widget`.
```

```
    """
```

```
raise
```

```
NotImplementedError
```

```
()
```

[\[docs\]](#)

```
def
```

```
    do_user_config
```

```
    (
```

```
    self
```

```
    ,
```

```
    parent
```

```
=
None
):

'''
    This method shows a configuration dialog for this plugin. It returns
    True if the user clicks OK, False otherwise. The changes are
    automatically applied.
'''

from
qt.core
import
QDialog
,
QDialogButtonBox
,
QVBoxLayout
,
\
QLabel
,
Qt
,
QLineEdit

from
```

calibre.gui2

import

gprefs

prefname

=

'plugin config dialog:'

+

self

.

type

+

':'

+

self

.

name

geom

=

gprefs

.

get

(

prefname

,

None

)

```
config_dialog
```

```
=
```

```
QDialog
```

```
(
```

```
parent
```

```
)
```

```
button_box
```

```
=
```

```
QDialogButtonBox
```

```
(
```

```
QDialogButtonBox
```

```
.
```

```
StandardButton
```

```
.
```

```
Ok
```

```
|
```

```
QDialogButtonBox
```

```
.
```

```
StandardButton
```

```
.
```

```
Cancel
```

```
)
```

```
v
```

```
=
```

```
QVBoxLayout
```

```
(
```

```
config_dialog
```

```
)
```

```
def
size_dialog
():

if
geom

is
None
:

config_dialog
.
resize
(
config_dialog
.
sizeHint
())

else
:

from
qt.core

import
QApplication

QApplication
.
```

```
instance
(  
.  
safe_restore_geometry  
(  
config_dialog  
,  
geom  
)
```

```
button_box  
.  
accepted  
.  
connect  
(  
config_dialog  
.  
accept  
)
```

```
button_box  
.  
rejected  
.  
connect  
(  
config_dialog  
.  
reject  
)
```

```
config_dialog  
.  
setWindowTitle  
(  
—  
(
```

```
'Customize'
)

+

''

+

self
.
name
)

try
:

config_widget

=

self
.
config_widget
()

except

NotImplementedError
:

config_widget

=

None
```



```
if
    isinstance
    (
        config_widget
        ,
        tuple
    ):

    from
        calibre.gui2

    import
        warning_dialog

    warning_dialog
    (
        parent
        ,
        _
    (
        'Cannot configure'
    ),
    config_widget
    [
        0
    ],

    det_msg
    =
    config_widget
    [
        1
    ],
```

```
show
=
True
)

return

False

if

config_widget

is

not

None
:

v
.
addWidget
(
config_widget
)

v
.
addWidget
(
button_box
)

size_dialog
()
```

```
config_dialog
```

```
.
```

```
exec
```

```
()
```

```
if
```

```
config_dialog
```

```
.
```

```
result
```

```
()
```

```
==
```

```
QDialog
```

```
.
```

```
DialogCode
```

```
.
```

```
Accepted
```

```
:
```

```
if
```

```
hasattr
```

```
(
```

```
config_widget
```

```
,
```

```
'validate'
```

```
):
```

```
if
```

```
config_widget
```

```
.
```

```
validate
```

```
():
```

```
self
.  
save_settings  
(  
config_widget  
)
```

```
else  
:
```

```
self  
.  
save_settings  
(  
config_widget  
)
```

```
else  
:
```

```
from
```

```
calibre.customize.ui
```

```
import
```

```
plugin_customization
```

```
,  
\
```

```
customize_plugin
```

```
help_text
```

```
=
```

```
self
```

```
.
customization_help
(
gui
=
True
)

help_text

=

QLabel
(
help_text
,

config_dialog
)

help_text
.
setWordWrap
(
True
)

help_text
.
setTextInteractionFlags
(
Qt
.
TextInteractionFlag
.
LinksAccessibleByMouse
|

Qt
```

```
.  
TextInteractionFlag  
.br/>LinksAccessibleByKeyboard  
)
```

help_text

```
.  
setOpenExternalLinks  
(  
  True  
)
```

v

```
.  
addWidget  
(  
  help_text  
)
```

sc

=

```
plugin_customization  
(  
  self  
)
```

if

not

```
sc  
:
```

sc

```
=  
  
"  
  
sc  
  
=  
  
sc  
.  
strip  
(  
  
sc  
  
=  
  
QLineEdit  
(  
sc  
,  
config_dialog  
)  
  
v  
.  
addWidget  
(  
sc  
)  
  
v  
.  
addWidget  
(  
button_box  
)
```

```
size_dialog  
()
```

```
config_dialog  
.  
exec  
()
```

```
if
```

```
config_dialog  
.  
result  
()
```

```
==
```

```
QDialog  
.  
DialogCode  
.  
Accepted  
:
```

```
sc
```

```
=
```

```
str  
(  
sc  
.  
text  
())  
.  
strip  
()
```



```
customize_plugin
```

```
(
```

```
self
```

```
,
```

```
sc
```

```
)
```

```
geom
```

```
=
```

```
bytearray
```

```
(
```

```
config_dialog
```

```
.
```

```
saveGeometry
```

```
())
```

```
gprefs
```

```
[
```

```
prefname
```

```
]
```

```
=
```

```
geom
```

```
return
```

```
config_dialog
```

```
.
```

```
result
```

```
()
```

[\[docs\]](#)

def

load_resources

(
self
,

names
):

'''

*If this plugin comes in a ZIP file (user added plugin), this method
will allow you to load resources from the ZIP file.*

For example to load an image::

pixmap = QPixmap()

pixmap.loadFromData(self.load_resources(['images/icon.png'])['images/icon.png'])

icon = QIcon(pixmap)

:param names: List of paths to resources in the ZIP file using / as separator

:return: A dictionary of the form ``{name: file\_contents}``. Any names

that were not found in the ZIP file will not be present in the

dictionary.

'''

```
if
self
.
plugin_path

is

None
:

raise

ValueError
(
'This plugin was not loaded from a ZIP file'
)

ans

=

{}

with

zipfile
.
ZipFile
(
self
.
plugin_path
,
'r'
)
as
```

```
zf
:

for

candidate

in

zf
.
namelist
():

if

candidate

in

names
:

ans
[
candidate
]

=

zf
.
read
(
candidate
)

return
```

ans

[\[docs\]](#)

def

customization_help

(

self

,

gui

=

False

):

'''

Return a string giving help on how to customize this plugin.

By default raise a :class:`NotImplementedError`, which indicates that the plugin does not require customization.

If you re-implement this method in your subclass, the user will be asked to enter a string as customization for this plugin.

The customization string will be available as

```self.site_customization```.

Site customization could be anything, for example, the path to a needed binary on the user's computer.

:param gui: If True return HTML help, otherwise return plain text help.

```
'''

raise

NotImplementedError
()

\[docs\]

def

temporary_file
(
    self
,
    suffix
):

'''

Return a file-like object that is a temporary file on the file system.

This file will remain available even after being closed and will only
be removed on interpreter shutdown. Use the ``name`` member of the
returned object to access the full path to the created temporary file.

:param suffix: The suffix that the temporary file will have.

'''

return

PersistentTemporaryFile
(
```

```
suffix
)

def

is_customizable
(
self
):

try
:

self
.
customization_help
()

return

True

except

NotImplementedError
:

return

False

def

__enter__
(
```

```
self
',
*
args
):

'''

    Add this plugin to the python path so that it's contents become directly importable.

    Useful when bundling large python libraries into the plugin. Use it like this::

        with plugin:

            import something

'''

if
self
.
plugin_path

is

not

None
:

from

calibre.utils.zipfile

import

ZipFile
```



```
from
importlib.machinery

import
EXTENSION_SUFFIXES

with
ZipFile
(
self
.
plugin_path
)

as

zf
:

extensions

=

{
x
.
lower
()

for
x

in
EXTENSION_SUFFIXES
}
```

```
zip_safe
```

```
=
```

```
True
```

```
for
```

```
name
```

```
in
```

```
zf
```

```
.
```

```
namelist
```

```
():
```

```
for
```

```
q
```

```
in
```

```
extensions
```

```
:
```

```
if
```

```
name
```

```
.
```

```
endswith
```

```
(
```

```
q
```

```
):
```

```
zip_safe
```

=

False

break

if

not

zip_safe
:

break

if

zip_safe
:

sys

·
path

·
append

(
self

·
plugin_path
)

self

·
sys_insertion_path

=

```
self
    .
    plugin_path

else
:

from
calibre.ptempfile

import
TemporaryDirectory

self
    .
    _sys_insertion_tdir

=

TemporaryDirectory
(
    'plugin_unzip'
)

self
    .
    sys_insertion_path

=

self
    .
    _sys_insertion_tdir
    .
    __enter__
(
    *
```

```
args
)

zf
.
extractall
(
self
.
sys_insertion_path
)

sys
.
path
.
append
(
self
.
sys_insertion_path
)

def
__exit__
(
self
,
*
args
):

ip
,

it
```

```
=  
  
getattr  
(  
    self  
,  
  
    'sys_insertion_path'  
,  
  
    None  
)  
  
getattr  
(  
    self  
,  
  
    '_sys_insertion_tdir'  
,  
  
    None  
)  
  
if  
  
ip  
  
in  
  
sys  
.  
path  
:  
  
sys  
.  
path  
.
```

```
remove
```

```
(  
ip  
)
```

```
if
```

```
hasattr
```

```
(  
it  
,
```

```
'__exit__'  
):
```

```
it
```

```
.  
__exit__  
(  
*  
args  
)
```

```
\[docs\]
```

```
def
```

```
cli_main
```

```
(  
self  
,
```

```
args  
):
```

```
'''
```

This method is the main entry point for your plugins command line interface. It is called when the user does: calibre-debug -r "Plugin Name". Any arguments passed are present in the args variable.

'''

raise

NotImplementedError

(
'The
%s
plugin has no command line interface'

%
self
.
name
)

}}

[\[docs\]](#)

class

FileTypePlugin

(
Plugin
):

{{{

'''

A plugin that is associated with a particular set of file types.

'''

#: Set of file types for which this plugin should be run.

#: Use '' for all file types.*

#: For example: ``{'lit', 'mobi', 'prc'}``

file_types

=

set
(

#: If True, this plugin is run when books are added

#: to the database

on_import

=

False

#: If True, this plugin is run after books are added

#: to the database. In this case the postimport and postadd

#: methods of the plugin are called.

on_postimport

=

False

#: If True, this plugin is run just before a conversion

on_preprocess

=

False

#: If True, this plugin is run after conversion

#: on the final file produced by the conversion output plugin.

on_postprocess

=

False

type

=

—

```
(  
'File type'  
)
```

[\[docs\]](#)

```
def
```

```
run
```

```
(  
self
```

```
,
```

```
path_to_ebook  
):
```

```
'''
```

Run the plugin. Must be implemented in subclasses.

It should perform whatever modifications are required

on the e-book and return the absolute path to the

modified e-book. If no modifications are needed, it should

return the path to the original e-book. If an error is encountered

it should raise an Exception. The default implementation

simply return the path to the original e-book. Note that the path to

the original file (before any file type plugins are run, is available as

self.original_path_to_file).

The modified e-book file should be created with the

:meth:`temporary\_file` method.

:param path_to_ebook: Absolute path to the e-book.

:return: Absolute path to the modified e-book.

'''

Default implementation does nothing

return

path_to_ebook

[\[docs\]](#)

def

postimport

(

self

,

book_id

,

book_format

,

db

):

'''

Called post import, i.e., after the book file has been added to the database. Note that

this is different from :meth:`postadd` which is called when the book record is created for

the first time. This method is called whenever a new file is added to a book record. It is useful for modifying the book record based on the contents of the newly added file.

:param book_id: Database id of the added book.

:param book_format: The file type of the book that was added.

:param db: Library database.

'''

pass

Default implementation does nothing

[\[docs\]](#)

def

postadd

(
self

,

book_id

,

fmt_map

,

db

):

'''

Called post add, i.e. after a book has been added to the db. Note that

this is different from :meth:`postimport`, which is called after a single book file has been added to a book. postadd() is called only when an entire book record with possibly more than one book file has been created for the first time. This is useful if you wish to modify the book record in the database when the book is first added to calibre.

:param book_id: Database id of the added book.

:param fmt_map: Map of file format to path from which the file format was added. Note that this might or might not point to an actual existing file, as sometimes files are added as streams. In which case it might be a dummy value or a non-existent path.

:param db: Library database

'''

pass

Default implementation does nothing

}}}

[\[docs\]](#)

class

MetadataReaderPlugin

(

Plugin

```
):  
  
# {{{  
  
'''  
  
    A plugin that implements reading metadata from a set of file types.  
  
    '''  
  
#: Set of file types for which this plugin should be run.  
  
#: For example: ``set(['lit', 'mobi', 'prc'])``  
  
file_types  
  
=  
  
set  
(  
  
supported_platforms  
  
=  
  
[  
    'windows'  
,  
  
    'osx'  
,  
  
    'linux'  
]  
  
version
```

```
=
```

```
numeric_version
```

```
author
```

```
=
```

```
'Kovid Goyal'
```

```
type
```

```
=
```

```
-
```

```
(  
'Metadata reader'  
)
```

```
def
```

```
__init__
```

```
(
```

```
self
```

```
,
```

```
*
```

```
args
```

```
,
```

```
**
```

```
kwargs
```

```
):
```

```
Plugin
```

```
.
```



```
__init__
```

```
(  
self
```

```
,
```

```
*
```

```
args
```

```
,
```

```
**
```

```
kwargs
```

```
)
```

```
self
```

```
.
```

```
quick
```

```
=
```

```
False
```

```
\[docs\]
```

```
def
```

```
get_metadata
```

```
(
```

```
self
```

```
,
```

```
stream
```

```
,
```

```
type
```

```
):
```

```
'''
```

Return metadata for the file represented by stream (a file like object that supports reading). Raise an exception when there is an error with the input data.

:param type: The type of file. Guaranteed to be one of the entries in :attr:`file\_types`.

:return: A :class:`calibre.ebooks.metadata.book.Metadata` object

'''

return

None

}}}}

[\[docs\]](#)

class

MetadataWriterPlugin

(
Plugin
):

{{{

'''

A plugin that implements reading metadata from a set of file types.

'''

#: Set of file types for which this plugin should be run.

#: For example: ``set(['lit', 'mobi', 'prc'])``

file_types

=

set
()

supported_platforms

=

[
'windows'
,
'osx'
,
'linux'
]

version

=

numeric_version

author

=

```
'Kovid Goyal'
```

```
type
```

```
=
```

```
—  
(
```

```
'Metadata writer'  
)
```

```
def
```

```
__init__
```

```
(
```

```
self
```

```
,
```

```
*
```

```
args
```

```
,
```

```
**
```

```
kwargs
```

```
):
```

```
Plugin
```

```
.
```

```
__init__
```

```
(
```

```
self
```

```
,
```

```
*
```

```
args
```

```
,
```

```
**
```

```
kwargs
)

self
.
apply_null
```

```
=
```

```
False
```

```
\[docs\]
```

```
def
```

```
set_metadata
(
self
,

stream
,

mi
,

type
):
```

```
'''
```

Set metadata for the file represented by stream (a file like object that supports reading). Raise an exception when there is an error with the input data.

:param type: The type of file. Guaranteed to be one of the entries

in :attr:`file\_types`.

:param mi: A :class:`calibre.ebooks.metadata.book.Metadata` object

'''

pass

}}}}

[\[docs\]](#)

class

CatalogPlugin

(
Plugin
):

{{{

'''

A plugin that implements a catalog generator.

'''

resources_path

=

None

```
#: Output file type for which this plugin should be run.
```

```
#: For example: 'epub' or 'xml'
```

```
file_types
```

```
=
```

```
set  
(
```

```
type
```

```
=
```

```
—  
(
```

```
'Catalog generator'  
)
```

```
#: CLI parser options specific to this plugin, declared as `namedtuple` `Option`:
```

```
#: 
```

```
#: from collections import namedtuple
```

```
#: Option = namedtuple('Option', 'option, default, dest, help')
```

```
#: cli_options = [Option('--catalog-title', default = 'My Catalog',
```

```
#: dest = 'catalog_title', help = (_('Title of generated catalog. \nDefault:') + " '" + '%default' +
"""))]
```

```
#: cli_options parsed in calibre.db.cli.cmd_catalog:option_parser()
```

```
#:
```

```
cli_options
```

```
=
```

```
[]
```

```
def
```

```
_field_sorter
```

```
(
```

```
self
```

```
,
```

```
key
```

```
):
```

```
'''
```

```
Custom fields sort after standard fields
```

```
'''
```

```
if
```

```
key
```

```
.
```

```
startswith
```

```
(
```

```
'#'
```



```
):  
  
    return  
  
    '~  
    %s  
    '  
  
    %  
  
    key  
    [  
    1  
    :]  
  
    else  
    :  
  
    return  
  
    key  
  
    def  
    search_sort_db  
    (  
    self  
    ,  
  
    db  
    ,  
  
    opts  
    ):  
  
    db
```

```
.  
search  
(  
  opts  
.  
  search_text  
)
```

```
if
```

```
getattr  
(  
  opts  
  ,
```

```
'sort_by'  
  ,
```

```
None  
):
```

```
# 2nd arg = ascending
```

```
db
```

```
.  
sort  
(  
  opts
```

```
.  
  sort_by  
  ,
```

```
True  
)
```

```
return
```

```
db
```

```
.  
get_data_as_dict  
(  
ids  
=  
opts  
.  
ids  
)
```

```
def
```

```
get_output_fields  
(  
self  
,  
  
db  
,  
  
opts  
):
```

```
# Return a list of requested fields
```

```
all_std_fields  
  
=  
  
{  
'author_sort'  
,  
'authors'  
,  
'comments'  
,  
'cover'  
,  
'formats'
```

```
,  
  
'id'  
,  
'isbn'  
,  
'library_name'  
,  
'ondevice'  
,  
'pubdate'  
,  
'publisher'  
,  
  
'rating'  
,  
'series_index'  
,  
'series'  
,  
'size'  
,  
'tags'  
,  
'timestamp'  
,  
  
'title_sort'  
,  
'title'  
,  
'uuid'  
,  
'languages'  
,  
'identifiers'  
}
```

```
all_custom_fields
```

```
=
```

```
set
```

```
(
```

```
db
```

```
.
```

```
custom_field_keys
```

```
())
```

```
for
```

```
field
```

```
in
```

```
list
```

```
(
```

```
all_custom_fields
```

```
):
```

```
fm
```

```
=
```

```
db
```

```
.
```

```
field_metadata
```

```
[
```

```
field
```

```
]
```

```
if
```

```
fm
```

```
[
```

```
'datatype'
```

```
]
```

```
==

'series'
:

all_custom_fields
.
add
(
field
+
'_index'
)

all_fields

=

all_std_fields
.
union
(
all_custom_fields
)

if

getattr
(
opts
,

'fields'
,

'all'
)

!=
```

```
'all'
:

# Make a list from opts.fields

of

=

[
x
.
strip
()

for

x

in

opts
.
fields
.
split
(
' '
,
)]

requested_fields

=

set
(
of
)
```

```
# Validate requested_fields
```

```
if
```

```
requested_fields
```

```
-
```

```
all_fields
```

```
:
```

```
from
```

```
calibre.library
```

```
import
```

```
current_library_name
```

```
invalid_fields
```

```
=
```

```
sorted
```

```
(
```

```
list
```

```
(
```

```
requested_fields
```

```
-
```

```
all_fields
```

```
))
```

```
print
```

```
(
```

```
"invalid --fields specified:
```



```
%s
"

%

','
.
join
(
invalid_fields
))

print
(
"available fields in "
%s
':
%s
"

%

(
current_library_name
()),

','
.
join
(
sorted
(
list
(
all_fields
))))

raise
ValueError
```

```
(  
"unable to generate catalog with specified fields"  
)
```

```
fields
```

```
=
```

```
[  
x
```

```
for
```

```
x
```

```
in
```

```
of
```

```
if
```

```
x
```

```
in
```

```
all_fields  
]
```

```
else
```

```
:
```

```
fields
```

```
=
```

```
sorted
```

```
(  
all_fields
```

```
,
```

```
key
=
self
.
_field_sorter
)

if
not
opts
.
connected_device
[
'is_device_connected'
]
and
'ondevice'
in
fields
:

fields
.
pop
(
int
(
fields
.
index
(
'ondevice'
)))
```

return

fields

[\[docs\]](#)

def

initialize

(
self
):

'''

*If plugin is not a built-in, copy the plugin's .ui and .py files from
the ZIP file to \$TMPDIR.*

*Tab will be dynamically generated and added to the Catalog Options dialog in
calibre.gui2.dialogs.catalog.py:Catalog*

'''

from

calibre.customize.builtins

import

plugins

as

```
builtin_plugins
```

```
from
```

```
calibre.customize.ui
```

```
import
```

```
config
```

```
from
```

```
calibre.ptempfile
```

```
import
```

```
PersistentTemporaryDirectory
```

```
if
```

```
not
```

```
type
```

```
(
```

```
self
```

```
)
```

```
in
```

```
builtin_plugins
```

```
and
```

```
self
```

```
.
```

```
name
```

```
not
```

in

config

```
[  
'disabled_plugins'  
]:
```

files_to_copy

=

```
[  
f  
"  
{  
self  
.  
name  
.  
lower  
()  
}  
.  
{  
ext  
}  
"
```

for

ext

in

```
[  
"ui"  
,  
"py"  
]]
```

resources

```
=

zipfile
.
ZipFile
(
self
.
plugin_path
,
'r'
)

if

self
.
resources_path

is

None
:

self
.
resources_path

=

PersistentTemporaryDirectory
(
'_plugin_resources'
,

prefix
=
"
)
)
```

```
for
file

in

files_to_copy
:

try
:

resources
.
extract
(
file
,

self
.
resources_path
)

except
:

print
(
f
"customize:__init__.initialize():
{
file
}
not found in
{
```



```
os
.  
path  
.  
basename  
(  
self  
.  
plugin_path  
)  
}  
"  
)
```

continue

```
resources  
.  
close  
()
```

[\[docs\]](#)

def

```
run  
(  
self  
,  
path_to_output  
,  
opts  
,  
db  
,  
ids
```

```

',
notification
=
None
):

'''
    Run the plugin. Must be implemented in subclasses.

    It should generate the catalog in the format specified
    in file_types, returning the absolute path to the
    generated catalog file. If an error is encountered
    it should raise an Exception.

    The generated catalog file should be created with the
    :meth:`temporary_file` method.

    :param path_to_output: Absolute path to the generated catalog file.

    :param opts: A dictionary of keyword arguments

    :param db: A LibraryDatabase2 object
'''

# Default implementation does nothing

raise
NotImplementedError
(
'CatalogPlugin.generate_catalog() default '

```

```
'method, should be overridden in subclass'  
)
```

```
# }}}
```

```
\[docs\]  
class
```

```
InterfaceActionBase
```

```
(  
    Plugin  
):
```

```
# {{{
```

```
supported_platforms
```

```
=
```

```
[  
    'windows'  
,  
    'osx'  
,  
    'linux'  
]
```

```
author
```

```
=
```

```
'Kovid Goyal'
```

```
type
```

```
=
```

```
-
```

```
(  
'User interface action'  
)
```

```
can_be_disabled
```

```
=
```

```
False
```

```
actual_plugin
```

```
=
```

```
None
```

```
def
```

```
__init__
```

```
(
```

```
self
```

```
,
```

```
*
```

```
args
```

```
,
```

```
**
```

```
kwargs
```

```
):
```

Plugin

```
·  
__init__
```

```
(  
self
```

```
,
```

```
*
```

args

```
,
```

```
**
```

kwargs

```
)
```

```
self
```

```
·
```

actual_plugin_

```
=
```

None

[\[docs\]](#)

def

load_actual_plugin

```
(
```

```
self
```

```
,
```

gui

```
):
```

```
'''
```

This method must return the actual interface action plugin object.

```
'''  
  
ac  
  
=  
  
self  
.  
actual_plugin_  
  
if  
  
ac  
  
is  
  
None  
:  
  
mod  
,  
  
cls  
  
=  
  
self  
.  
actual_plugin  
.  
split  
(  
    '.',  
)  
  
ac
```

```
=  
  
getattr  
(  
importlib  
.  
import_module  
(  
mod  
),  
  
cls  
)(  
gui  
,  
  
self  
.  
site_customization  
)  
  
self  
.  
actual_plugin_  
  
=  
  
ac  
  
return  
  
ac  
  
# }}}}
```

[\[docs\]](#)

class

PreferencesPlugin

(
Plugin
):

{{{

'''

A plugin representing a widget displayed in the Preferences dialog.

This plugin has only one important method :meth:`create\_widget`. The

various fields of the plugin control how it is categorized in the UI.

'''

supported_platforms

=

[
'windows'
,
'osx'
,
'linux'
]

author

=


```
'Kovid Goyal'
```

```
type
```

```
=
```

```
—
```

```
(  
'Preferences'  
)
```

```
can_be_disabled
```

```
=
```

```
False
```

```
#: Import path to module that contains a class named ConfigWidget
```

```
#: which implements the ConfigWidgetInterface. Used by
```

```
#: :meth:`create_widget`.
```

```
config_widget
```

```
=
```

```
None
```

```
#: Where in the list of categories the :attr:`category` of this plugin should be.
```

```
category_order
```

=

100

#: Where in the list of names in a category, the :attr:`gui\_name` of this

#: plugin should be

name_order

=

100

#: The category this plugin should be in

category

=

None

#: The category name displayed to the user for this plugin

gui_category

=

None

#: The name displayed to the user for this plugin

gui_name

=

None

#: The icon for this plugin, should be an absolute path

icon

=

None

#: The description used for tooltips and the like

description

=

None

[\[docs\]](#)

def

create_widget

(

self

,

```
parent
=
None
):

'''
    Create and return the actual Qt widget used for setting this group of
    preferences. The widget must implement the
    :class:`calibre.gui2.preferences.ConfigWidgetInterface`.

    The default implementation uses :attr:`config_widget` to instantiate
    the widget.
'''

base
,
_
,
wc
=
self
.
config_widget
.
partition
(
    '!'
)

if
```

not

WC

:

WC

=

'ConfigWidget'

base

=

importlib

.

import_module

(

base

)

widget

=

getattr

(

base

,

WC

)

return

widget

(

```
parent
)
```

```
# }}}
```

```
class
```

```
StoreBase
```

```
(
Plugin
):
```

```
# {{{
```

```
supported_platforms
```

```
=
```

```
[
'windows'
,
'osx'
,
'linux'
]
```

```
author
```

```
=
```

```
'John Schember'
```

```
type
```

```
=  
  
_  
(  
'Store'  
)  
  
# Information about the store. Should be in the primary language  
  
# of the store. This should not be translatable when set by  
  
# a subclass.  
  
description  
  
=  
  
_  
(  
'An e-book store.'  
)  
  
minimum_calibre_version  
  
=  
  
(  
0  
,  
8  
,  
0  
)  
  
version
```

=

(
1
,
0
,
1
)

actual_plugin

=

None

Does the store only distribute e-books without DRM.

drm_free_only

=

False

This is the 2 letter country code for the corporate

headquarters of the store.

headquarters

=


```
"""

# All formats the store distributes e-books in.

formats

=

[]

# Is this store on an affiliate program?

affiliate

=

False


def

load_actual_plugin
(
    self
,
    gui
):

"""

    This method must return the actual interface action plugin object.

    """

mod
```

```
,

cls

=

self
.
actual_plugin
.
split
(
    '\.'
)

self
.
actual_plugin_object

=

getattr
(
    importlib
.
import_module
(
    mod
),

cls
)(
    gui
,

self
.
name
)

return
```

```
self
.  
actual_plugin_object
```

```
def
```

```
customization_help
```

```
(  
self  
,
```

```
gui
```

```
=
```

```
False
```

```
):
```

```
if
```

```
getattr
```

```
(  
self  
,
```

```
'actual_plugin_object'
```

```
,
```

```
None
```

```
)
```

```
is
```

```
not
```

```
None
```

```
:
```

```
return
```

```
self
.  
actual_plugin_object  
.  
customization_help  
(  
gui  
)
```

raise

```
NotImplementedError  
(
```

def

```
config_widget  
(  
self  
):
```

if

```
getattr  
(  
self  
,  
  
'actual_plugin_object'  
,
```

```
None  
)
```

is

not

None

```
:  
  
    return  
  
    self  
    .  
    actual_plugin_object  
    .  
    config_widget  
    ()  
  
    raise  
  
    NotImplementedError  
    ()  
  
    def  
  
    save_settings  
    (  
    self  
    ,  
  
    config_widget  
    ):  
  
    if  
  
    getattr  
    (  
    self  
    ,  
  
    'actual_plugin_object'  
    ,  
  
    None  
    )
```

is

not

None

:

return

self

.

actual_plugin_object

.

save_settings

(

config_widget

)

raise

NotImplementedError

()

}}}}

class

EditBookToolPlugin

(

Plugin

):

{{{

type

```
=
```

```
-  
(  
'Edit book tool'  
)
```

```
minimum_calibre_version
```

```
=
```

```
(  
1  
,  
46  
,  
0  
)
```

```
# }}}  
  

```

```
class
```

```
LibraryClosedPlugin
```

```
(  
Plugin  
):
```

```
# {{{
```

```
'''
```

*LibraryClosedPlugins are run when a library is closed, either at shutdown,
when the library is changed, or when a library is used in some other way.*

```

'''
    At the moment these plugins won't be called by the CLI functions.

'''

type
    =
    (
        'Library closed'
    )

# minimum version 2.54 because that is when support was added

minimum_calibre_version
    =
    (
        2
        ,
        54
        ,
        0
    )

def
run
(
    self
    ,

```



```

db
):

'''
    The db will be a reference to the new_api (db.cache.py).

    The plugin must run to completion. It must not use the GUI, threads, or
    any signals.
'''

raise

NotImplementedError
(
'LibraryClosedPlugin '

'run method must be overridden in subclass'
)

# }}}

```

Source code for calibre.customize.conversion

```
'''  
  
Defines the plugin system for conversions.  
  
'''  
  
import  
  
re  
,  
  
os  
,  
  
shutil  
,  
  
numbers  
  
from  
  
calibre  
  
import  
  
CurrentDir  
  
from  
  
calibre.customize  
  
import
```

Plugin

class

ConversionOption

:

'''

Class representing conversion options

'''

def

__init__

(

self

,

name

=

None

,

help

=

None

,

long_switch

=

None

,

```
short_switch
```

```
=
```

```
None
```

```
,
```

```
choices
```

```
=
```

```
None
```

```
):
```

```
self
```

```
.
```

```
name
```

```
=
```

```
name
```

```
self
```

```
.
```

```
help
```

```
=
```

```
help
```

```
self
```

```
.
```

```
long_switch
```

```
=
```

```
long_switch
```

```
self
```

```
.
```

```
short_switch
```

```
=
```

```
short_switch
```

```
self
```

```
.
```

```
choices
```

```
=
```

```
choices
```

```
if
```

```
self
```

```
.
```

```
long_switch
```

```
is
```

```
None
```

```
:
```

```
self
```

```
.
```

```
long_switch
```

```
=
```

```
self
```

```
.
```

```
name
```

```
.
```

```
replace
```

```
(
```

```
'_'
```

```
—
```

```
,
```

```
'_'
```

```
)
```

```
self
·
validate_parameters
()

def

validate_parameters
(
self
):

'''

    Validate the parameters passed to :meth:``__init__``.

'''

if

re
·
match
(
r
'[a-zA-Z_]([a-zA-Z0-9_]*)'
,

self
·
name
)

is

None
```

```
:  
  
raise  
  
ValueError  
(  
    self  
    .  
    name  
    +  
    ' is not a valid Python identifier'  
)  
  
if  
  
not  
  
self  
.  
help  
:  
  
raise  
  
ValueError  
(  
    'You must set the help text'  
)  
  
def  
  
__hash__  
(  
    self  
):
```

return

hash

(
self
.
name
)

def

__eq__

(
self
,

other
):

return

self
.
name

==

getattr

(
other
,

'name'
,

other
)


```
def
```

```
clone
```

```
(
```

```
self
```

```
):
```

```
return
```

```
ConversionOption
```

```
(
```

```
name
```

```
=
```

```
self
```

```
.
```

```
name
```

```
,
```

```
help
```

```
=
```

```
self
```

```
.
```

```
help
```

```
,
```

```
long_switch
```

```
=
```

```
self
```

```
.
```

```
long_switch
```

```
,
```

```
short_switch
```

```
=
```

```
self
```

```
.
```

```
short_switch
```

```
,
```

```
choices
```

```
=
```

```
self
```

```
.
```

```
choices
```

```
)
```

```
class
```

```
OptionRecommendation
```

```
:
```

```
LOW
```

```
=
```

```
1
```

```
MED
```

```
=
```

```
2
```

```
HIGH
```

```
=
```

```
3
```

```
def
```

```
__init__
```

```
(
```

```
self
```

```
,
```

```
recommended_value
=
None
,

level
=
LOW
,

**
kwargs
):

'''
    An option recommendation. That is, an option as well as its recommended
    value and the level of the recommendation.
'''

self
.
level
=
level

self
.
recommended_value
=
recommended_value
```

```
self
.  
option  
  
=  
  
kwargs  
.  
pop  
(  
    'option'  
    ,  
    None  
)
```

```
if  
  
self  
.  
option  
  
is  
  
None  
:
```

```
self  
.  
option  
  
=  
  
ConversionOption  
(  
    **  
    kwargs  
)
```

```
self
.
validate_parameters
()

@property

def

help
(
self
):

return

self
.
option
.
help

def

clone
(
self
):

return

OptionRecommendation
(
recommended_value
=
self
```

```
.  
recommended_value  
,
```

```
level  
=  
self
```

```
.  
level  
,
```

```
option  
=  
self
```

```
.  
option  
.  
clone  
())
```

```
def
```

```
validate_parameters  
(  
self  
):
```

```
if
```

```
self
```

```
.  
option
```

```
.  
choices
```

```
and
```

```
self
```

```
.
```

```
recommended_value

not

in
\

self
.
option
.
choices
:

raise

ValueError
(
'OpRec:
%s
: Recommended value not in choices'
%

self
.
option
.
name
)

if

not

(
isinstance
(
self
.
recommended_value
```

```
,  
  
(  
  numbers  
  .  
  Number  
,  
  
  bytes  
,  
  
  str  
)  
  
or  
  
self  
.  
recommended_value  
  
is  
  
None  
):  
  
raise  
  
ValueError  
(  
  'OpRec:  
  %s  
  :'  
  %  
  self  
  .  
  option  
  .  
  name  
  +  
  repr
```



```
(  
  
    self  
    .  
    recommended_value  
)  
  
+  
  
' is not a string or a number'  
)
```

class

DummyReporter

:

def

__init__

(

self

):

self

.

cancel_requested

=

False

def

__call__

```
(
self
,

percent
,

msg
=
"
):

pass


def

gui_configuration_widget
(
name
,

parent
,

get_option_by_name
,

get_option_help
,

db
,

book_id
,

for_output
=
True
```

```
):
```

```
import
```

```
importlib
```

```
def
```

```
widget_factory
```

```
(
```

```
cls
```

```
):
```

```
return
```

```
cls
```

```
(
```

```
parent
```

```
,
```

```
get_option_by_name
```

```
,
```

```
get_option_help
```

```
,
```

```
db
```

```
,
```

```
book_id
```

```
)
```

```
if
```

```
for_output
```

```
:
```

```
try
:

output_widget

=

importlib
.
import_module
(

'calibre.gui2.convert.'
+
name
)

pw

=

output_widget
.
PluginWidget

pw
.
ICON

=

|
(
'back.png'
)
```

```
pw
.  
HELP  
  
=  
  
_  
(  
'Options specific to the output format.'  
)  
  
return  
  
widget_factory  
(  
pw  
)  
  
except  
  
ImportError  
:  
  
pass  
  
else  
:  
  
try  
:  
  
input_widget  
  
=  
  
importlib  
.
```

```
import_module
(

'calibre.gui2.convert.'
+
name
)

pw

=

input_widget
.
PluginWidget

pw
.
ICON

=

I
(
'forward.png'
)

pw
.
HELP

=

-
(
'Options specific to the input format.'
)
```

```
return

widget_factory
(
pw
)
```

```
except
```

```
ImportError
:
```

```
pass
```

```
return
```

```
None
```

```
\[docs\]
```

```
class
```

```
InputFormatPlugin
```

```
(
Plugin
):
```

```
'''
```

InputFormatPlugins are responsible for converting a document into

HTML+OPF+CSS+etc.

*The results of the conversion ***must*** be encoded in UTF-8.*

The main action happens in :meth:`convert`.

```
'''
    """
    type
    =
    -
    (
    'Conversion input'
    )

    can_be_disabled

    =

    False

    supported_platforms

    =

    [
    'windows'
    ,

    'osx'
    ,

    'linux'
    ]

    commit_name

    =

    None
```


unique name under which options for this plugin are saved

ui_data

=

None

#: Set of file types for which this plugin should be run

#: For example: ``set(['azw', 'mobi', 'prc'])``

file_types

=

set
()

#: If True, this input plugin generates a collection of images,

#: one per HTML file. This can be set dynamically, in the convert method

#: if the input files can be both image collections and non-image collections.

#: If you set this to True, you must implement the get_images() method that returns

#: a list of images.

is_image_collection

=

False

#: Number of CPU cores used by this plugin.

#: A value of -1 means that it uses all available cores

core_usage

=

1

#: If set to True, the input plugin will perform special processing

#: to make its output suitable for viewing

for_viewer

=

False

#: The encoding that this input plugin creates files in. A value of

#: None means that the encoding is undefined and must be

#: detected individually

```
output_encoding
```

```
=
```

```
'utf-8'
```

```
#: Options shared by all Input format plugins. Do not override
```

```
#: in sub-classes. Use :attr:`options` instead. Every option must be an
```

```
#: instance of :class:`OptionRecommendation`.
```

```
common_options
```

```
=
```

```
{
```

```
OptionRecommendation
```

```
(
```

```
name
```

```
=
```

```
'input_encoding'
```

```
,
```

```
recommended_value
```

```
=
```

```
None
```

```
,
```

```
level
```

```
=
```

```
OptionRecommendation
```

```
.
```

```
LOW
```

```
,
```

```
help
```

```
=
```

```
—
```

```
(
```

```
'Specify the character encoding of the input document. If '
```

```
'set this option will override any encoding declared by the '
```

```
'document itself. Particularly useful for documents that '
```

```
'do not declare an encoding or that have erroneous '
```

```
'encoding declarations.'
```

```
)
```

```
}}
```

```
#: Options to customize the behavior of this plugin. Every option must be an
```

```
#: instance of :class:`OptionRecommendation`.
```

```
options
```

```
=
```

```
set
```

```
()
```

```
#: A set of 3-tuples of the form
```

```
#: (option_name, recommended_value, recommendation_level)
```

```
recommendations
```

```
=
```

```
set
```

```
()
```

```
def
```

```
    __init__
```

```
(
```

```
    self
```

```
,
```

```
    *
```

```
    args
```

```
):
```

```
    Plugin
```

```
    .
```

```
        __init__
```

```
(
```

```
    self
```

```
,
```

```
    *
```

```
    args
```

```
)
```

```
    self
```

```
    .
```

```
    report_progress
```

```
=
```

```
DummyReporter  
(
```

```
\[docs\]
```

```
def
```

```
get_images  
(  
    self  
):
```

```
'''
```

```
    Return a list of absolute paths to the images, if this input plugin
```

```
    represents an image collection. The list of images is in the same order
```

```
    as the spine and the TOC.
```

```
'''
```

```
raise
```

```
NotImplementedError  
(
```

```
\[docs\]
```

```
def
```

```
convert  
(  
    self  
    ,
```

```
stream
',
options
',
file_ext
',
log
',
accelerators
):
```

```
'''
```

This method must be implemented in sub-classes. It must return the path to the created OPF file or an :class:`OEBBook` instance.

All output should be contained in the current folder.

If this plugin creates files outside the current folder they must be deleted/marked for deletion before this method returns.

:param stream: A file like object that contains the input file.

:param options: Options to customize the conversion process.

Guaranteed to have attributes corresponding to all the options declared by this plugin. In addition, it will have a verbose attribute that takes integral values from zero upwards. Higher numbers

```

        mean be more verbose. Another useful attribute is

        ``input_profile`` that is an instance of

        :class:`calibre.customize.profiles.InputProfile`.

    :param file_ext: The extension (without the .) of the input file. It
        is guaranteed to be one of the `file_types` supported
        by this plugin.

    :param log: A :class:`calibre.utils.logging.Log` object. All output
        should use this object.

    :param accelerators: A dictionary of various information that the input
        plugin can get easily that would speed up the
        subsequent stages of the conversion.

    """

    raise
    NotImplementedError

    def
    __call__
    (
    self
    ,
    stream
    ,
    options

```



```
,  
  
file_ext  
,  
  
log  
,  
  
accelerators  
,  
  
output_dir  
):  
  
try  
:  
  
log  
(  
    'InputFormatPlugin:  
    %s  
    running'  
    %  
    self  
    .  
    name  
)  
  
if  
  
hasattr  
(  
    stream  
    ,  
  
    'name'  
):
```

```
log
(
'on'
,

stream
.
name
)

except
:

# In case stdout is broken

pass

with

CurrentDir
(
output_dir
):

for

x

in

os
.
listdir
(
'.'
):
```

```
shutil
```

```
.
```

```
rmtree
```

```
(
```

```
x
```

```
)
```

```
if
```

```
os
```

```
.
```

```
path
```

```
.
```

```
isdir
```

```
(
```

```
x
```

```
)
```

```
else
```

```
os
```

```
.
```

```
remove
```

```
(
```

```
x
```

```
)
```

```
ret
```

```
=
```

```
self
```

```
.
```

```
convert
```

```
(
```

```
stream
```

```
,
```

```
options
```

```
,
```

```
file_ext
,

log
,

accelerators
)
```

```
return
```

```
ret
```

```
\[docs\]
```

```
def
```

```
postprocess_book
```

```
(  
self
```

```
,
```

```
oeb
```

```
,
```

```
opts
```

```
,
```

```
log
```

```
):
```

```
'''
```

Called to allow the input plugin to perform postprocessing after

the book has been parsed.

'''

pass

[\[docs\]](#)

def

specialize

(

self

,

oeb

,

opts

,

log

,

output_fmt

):

'''

Called to allow the input plugin to specialize the parsed book

for a particular output format. Called after postprocess_book

and before any transforms are performed on the parsed book.

'''

pass

[\[docs\]](#)

def

gui_configuration_widget

(

self

,

parent

,

get_option_by_name

,

get_option_help

,

db

,

book_id

=

None

):

"""

Called to create the widget used for configuring this plugin in the

calibre GUI. The widget must be an instance of the PluginWidget class.

See the builtin input plugins for examples.

"""

name

=

```
self
    .
    name
    .
    lower
    ()
    .
    replace
    (
        ' '
    ,
        ' _ '
    )

return

gui_configuration_widget
(
    name
    ,

    parent
    ,

    get_option_by_name
    ,

    get_option_help
    ,

    db
    ,

    book_id
    ,

    for_output
    =
False
```

```
)
```

```
\[docs\]
```

```
class
```

```
OutputFormatPlugin
```

```
(  
Plugin  
):
```

```
'''
```

```
OutputFormatPlugins are responsible for converting an OEB document  
(OPF+HTML) into an output e-book.
```

```
The OEB document can be assumed to be encoded in UTF-8.
```

```
The main action happens in :meth:`convert`.
```

```
'''
```

```
type
```

```
=
```

```
—  
(  
'Conversion output'  
)
```

```
can_be_disabled
```

```
=
```

```
False
```


supported_platforms

=

```
[  
  'windows'  
,  
  'osx'  
,  
  'linux'  
]
```

commit_name

=

None

unique name under which options for this plugin are saved

ui_data

=

None

#: The file type (extension without leading period) that this

#: plugin outputs

file_type

=

None

#: Options shared by all Input format plugins. Do not override

#: in sub-classes. Use :attr:`options` instead. Every option must be an

#: instance of :class:`OptionRecommendation`.

common_options

=

{

OptionRecommendation

(

name

=

'pretty_print'

,

recommended_value

=

False

,

level

=

OptionRecommendation

.

LOW

,

help

```
=
-
(
'If specified, the output plugin will try to create output '

'that is as human readable as possible. May not have any effect '

'for some output plugins.'
)

}}
```

#: Options to customize the behavior of this plugin. Every option must be an

#: instance of :class:`OptionRecommendation`.

options

=

set
()

#: A set of 3-tuples of the form

#: (option_name, recommended_value, recommendation_level)

recommendations

=

set

```
()

@property

def
description
(
    self
):

    return

_
(
    'Convert e-books to the
    %s
    format'
)
%
self
.
file_type
.
upper
()

def
__init__
(
    self
,

    *
args
):
```

Plugin

```
·  
__init__
```

```
(  
self
```

```
,
```

```
*
```

```
args
```

```
)
```

```
self
```

```
·  
report_progress
```

```
=
```

```
DummyReporter  
()
```

```
\[docs\]
```

```
def
```

```
convert
```

```
(  
self
```

```
,
```

```
oeb_book
```

```
,
```

```
output
```

```
,
```

```
input_plugin
```

```
,
```

```
opts
```

```
,
```

```
log
```

```
):
```

```
'''
```

*Render the contents of `oeb\_book` (which is an instance of
:class:`calibre.ebooks.oeb.OEBBook`) to the file specified by output.*

*:param output: Either a file like object or a string. If it is a string
it is the path to a folder that may or may not exist. The output
plugin should write its output into that folder. If it is a file like
object, the output plugin should write its output into the file.*

*:param input_plugin: The input plugin that was used at the beginning of
the conversion pipeline.*

*:param opts: Conversion options. Guaranteed to have attributes
corresponding to the OptionRecommendations of this plugin.*

:param log: The logger. Print debug/info messages etc. using this.

```
'''
```

```
raise
```

```
NotImplementedError
```

```
()
```

```
@property
```

```
def
```

```
is_periodical
```

```
(
```

```
self
```

```
):
```

```
return
```

```
self
```

```
.
```

```
oeb
```

```
.
```

```
metadata
```

```
.
```

```
publication_type
```

```
and
```

```
\
```

```
str
```

```
(
```

```
self
```

```
.
```

```
oeb
```

```
.
```

```
metadata
```

```
.
```

```
publication_type
```

```
[
```

```
0
```

```
])
```

```
.
```

```
startswith
```

```
(
```

```
'periodical:'
```

```
)
```

[\[docs\]](#)

def

specialize_options

(
self

,

log

,

opts

,

input_fmt

):

'''

*Can be used to change the values of conversion options, as used by the
conversion pipeline.*

'''

pass

[\[docs\]](#)

def

specialize_css_for_output

(

self

,


```
log
```

```
,
```

```
opts
```

```
,
```

```
item
```

```
,
```

```
stylizer
```

```
):
```

```
'''
```

Can be used to make changes to the css during the CSS flattening process.

:param item: The item (HTML file) being processed

:param stylizer: A Stylizer object containing the flattened styles for

item. You can get the style for any element by

stylizer.style(element).

```
'''
```

```
pass
```

```
\[docs\]
```

```
def
```

```
gui_configuration_widget
```

```
(
```

```
self
```

```
,
```

```

parent
,

get_option_by_name
,

get_option_help
,

db
,

book_id
=
None
):

"""
    Called to create the widget used for configuring this plugin in the
    calibre GUI. The widget must be an instance of the PluginWidget class.

    See the builtin output plugins for examples.
    """

name
=

self
.
name
.
lower
()
.
replace

```

```
(  
    '  
,  
    '  
    )  
  
    return  
  
    gui_configuration_widget  
    (  
        name  
,  
  
        parent  
,  
  
        get_option_by_name  
,  
  
        get_option_help  
,  
  
        db  
,  
  
        book_id  
,  
  
        for_output  
        =  
        True  
    )
```

Source code for calibre.db.cache

```
#!/usr/bin/env python
```

```
__license__
```

```
=
```

```
'GPL v3'
```

```
__copyright__
```

```
=
```

```
'2011, Kovid Goyal <kovid@kovidgoyal.net>'
```

```
__docformat__
```

```
=
```

```
'restructuredtext en'
```

```
import
```

```
operator
```

```
import
```

```
os
```

```
import
```

```
random
```

import

shutil

import

sys

import

traceback

from

collections

import

defaultdict

from

collections.abc

import

MutableSet

,

Set

from

functools

import

partial

,

wraps

from

io

import

BytesIO

from

threading

import

Lock

from

time

import

time

from

calibre

import

as_unicode

,

isbytestring

from

calibre.constants

import

iswindows

,

```
preferred_encoding

from
calibre.customize.ui

import

(

run_plugins_on_import
,

run_plugins_on_postadd
,

run_plugins_on_postimport
)

from
calibre.db

import

SPOOL_SIZE
,

_get_next_series_num_for_list

from
calibre.db.annotations

import

merge_annotations

from
```

calibre.db.categories

import

get_categories

from

calibre.db.errors

import

NoSuchBook

,

NoSuchFormat

from

calibre.db.fields

import

IDENTITY

,

InvalidLinkTable

,

create_field

from

calibre.db.lazy

import

FormatMetadata

,

FormatsList

,

ProxyMetadata

from

calibre.db.listeners

import

EventDispatcher

,

EventType

from

calibre.db.locking

import

(

DowngradeLockError

,

LockingError

,

SafeReadLock

,

create_locks

,

try_lock

)

from

calibre.db.search

import

Search

from

calibre.db.tables

import

VirtualTable

from

calibre.db.utils

import

type_safe_sort_key_function

from

calibre.db.write

import

get_series_values

,

uniq

from

calibre.ebooks

import

check_ebook_format

from

calibre.ebooks.metadata

import

```
author_to_author_sort
,
string_to_authors

from
calibre.ebooks.metadata.book.base

import
Metadata

from
calibre.ebooks.metadata.opf2

import
metadata_to_opf

from
calibre.ptempfile

import
PersistentTemporaryFile
,
SpooledTemporaryFile
,
base_dir

from
calibre.utils.config

import
prefs
```

```
,  
  
tweaks  
  
from  
  
calibre.utils.date  
  
import  
  
UNDEFINED_DATE  
,  
  
now  
  
as  
  
nowf  
,  
  
utcnow  
  
from  
  
calibre.utils.icu  
  
import  
  
sort_key  
  
from  
  
calibre.utils.localization  
  
import  
  
canonicalize_lang  
  
from  
  
polyglot.builtins  
  
import
```

```
cmp
,

iteritems
,

itervalues
,

string_or_bytes
```

```
def
```

```
api
(
f
):
```

```
f
.
is_cache_api
```

```
=
```

```
True
```

```
return
```

```
f
```

```
def
```

```
read_api
(
f
):
```

```
f
=
api
(
f
)

f
.
is_read_api
=
True

return

f

def
write_api
(
f
):

f
=
api
(
f
)
```

```
f
.  
is_read_api
```

```
=
```

```
False
```

```
return
```

```
f
```

```
def
```

```
wrap_simple
```

```
(  
lock  
,
```

```
func  
):
```

```
@wraps
```

```
(  
func  
)
```

```
def
```

```
call_func_with_lock
```

```
(  
*  
args  
,
```

```
**
```

```
kwargs
):

try
:

with

lock
:

return

func
(
*
args
,
**
kwargs
)

except

DowngradeLockError
:

# We already have an exclusive lock, no need to acquire a shared

# lock. See the safe_read_lock properties' documentation for why

# this is necessary.

return
```



```
func
```

```
(
```

```
*
```

```
args
```

```
,
```

```
**
```

```
kwargs
```

```
)
```

```
return
```

```
call_func_with_lock
```

```
def
```

```
run_import_plugins
```

```
(
```

```
path_or_stream
```

```
,
```

```
fmt
```

```
):
```

```
fmt
```

```
=
```

```
fmt
```

```
.
```

```
lower
```

```
()
```

```
if
```

```
hasattr
```

```
(
```

```
path_or_stream
```

```
,
```

```
'seek'
```

```
):
```

```
path_or_stream
```

```
.
```

```
seek
```

```
(
```

```
0
```

```
)
```

```
pt
```

```
=
```

```
PersistentTemporaryFile
```

```
(
```

```
'_import_plugin.'
```

```
+
```

```
fmt
```

```
)
```

```
shutil
```

```
.
```

```
copyfileobj
```

```
(
```

```
path_or_stream
```

```
,
```

```
pt
```

```
,
```

```
1024
```

```
**
```

```
2
```

```
)
```

```
pt
.
close
()

path

=

pt
.
name

else
:

path

=

path_or_stream

return

run_plugins_on_import
(
path
,
fmt
)

def

_add_newbook_tag
(
mi
```

```
):  
  
tags  
  
=  
  
prefs  
[  
  'new_book_tags'  
]  
  
if  
  
tags  
:  
  
    for  
  
        tag  
  
    in  
  
        [  
            t  
            .  
            strip  
            ()  
  
        for  
  
            t  
  
        in  
  
            tags  
        ]:  
  
        if
```

tag
:

if

not

mi
.
tags
:

mi
.
tags

=

[
tag
]

elif

tag

not

in

mi
.
tags
:

mi
.
tags
.

```
append
(
tag
)

def

_add_default_custom_column_values
(
mi
,
fm
):

cols

=

fm
.
custom_field_metadata
(
include_composites
=
False
)

for

cc
,
col

in

iteritems
(
cols
```

```
):  
  
dv  
  
=  
  
col  
[  
  'display'  
]  
.  
get  
(  
  'default_value'  
,  
  None  
)  
  
try  
:  
  
  if  
  
    dv  
  
    is  
  
    not  
  
    None  
    :  
  
    if  
  
    not  
  
    mi  
    .
```

```
get_user_metadata
```

```
(
```

```
cc
```

```
,
```

```
make_copy
```

```
=
```

```
False
```

```
):
```

```
mi
```

```
.
```

```
set_user_metadata
```

```
(
```

```
cc
```

```
,
```

```
col
```

```
)
```

```
dt
```

```
=
```

```
col
```

```
[
```

```
'datatype'
```

```
]
```

```
if
```

```
dt
```

```
==
```

```
'datetime'
```

```
and
```

```
icu_lower
```



```
(  
dv  
)
```

```
==
```

```
'now'  
:
```

```
dv
```

```
=
```

```
nowf  
()
```

```
mi
```

```
.  
set  
(  
cc  
,
```

```
dv  
)
```

```
except  
:
```

```
traceback  
.  
print_exc  
()
```

```
dynamic_category_preferences
```

```
=
```

```
frozenset
({
'grouped_search_make_user_categories'
,
'grouped_search_terms'
,
'user_categories'
})
```

[\[docs\]](#)

class

Cache

:

'''

An in-memory cache of the metadata.db file from a calibre library.

This class also serves as a threadsafe API for accessing the database.

The in-memory cache is maintained in normal form for maximum performance.

SQLITE is simply used as a way to read and write from metadata.db robustly.

All table reading/sorting/searching/caching logic is re-implemented. This

was necessary for maximum performance and flexibility.

'''

EventType

```
=
```

```
EventType
```

```
def
```

```
    __init__
```

```
(
```

```
    self
```

```
,
```

```
    backend
```

```
):
```

```
    self
```

```
        .
```

```
        backend
```

```
=
```

```
    backend
```

```
    self
```

```
        .
```

```
        event_dispatcher
```

```
=
```

```
    EventDispatcher
```

```
()
```

```
    self
```

```
        .
```

```
        fields
```

```
=
```

```
{  
  
    self  
    .  
    composites  
  
    =  
  
    {  
  
        self  
        .  
        read_lock  
        ,  
  
        self  
        .  
        write_lock  
  
        =  
  
        create_locks  
        ()  
  
        self  
        .  
        format_metadata_cache  
  
        =  
  
        defaultdict  
        (  
            dict  
        )  
  
        self  
        .  
        formatter_template_cache
```

```
=  
  
{  
  
self  
.  
dirtied_cache  
  
=  
  
}  
  
self  
.  
vls_for_books_cache  
  
=  
  
None  
  
self  
.  
vls_for_books_lib_in_process  
  
=  
  
None  
  
self  
.  
vls_cache_lock  
  
=  
  
Lock  
(  
  
self
```

```
.
dirtyed_sequence

=

0

self
.
cover_caches

=

set
()

self
.
clear_search_cache_count

=

0


# Implement locking for all simple read/write API methods


# An unlocked version of the method is stored with the name starting


# with a leading underscore. Use the unlocked versions when the lock


# has already been acquired.


for
name
```

in

dir

(
self
):

func

=

getattr

(
self
,

name
)

ira

=

getattr

(
func
,

'is_read_api'

,

None

)

if

ira

is

not

None

:

Save original function

setattr

(

self

,

' '

+

name

,

func

)

Wrap it in a lock

lock

=

self

.

read_lock

if

ira

else

self

.


```
write_lock

setattr
(
self
,

name
,

wrap_simple
(
lock
,

func
))

self
.
_search_api
=

Search
(
self
,

'saved_searches'
,

self
.
field_metadata
.
get_search_terms
())
```

```
self
.  
initialize_dynamic  
()
```

```
@property
```

```
def
```

```
new_api  
(  
self  
):
```

```
return
```

```
self
```

```
@property
```

```
def
```

```
library_id  
(  
self  
):
```

```
return
```

```
self
```

```
.  
backend
```

```
.  
library_id
```

```
@property
```

```
def
```

```
dbpath
```

```
(  
    self  
):
```

```
return
```

```
self
```

```
.  
backend
```

```
.  
dbpath
```

```
@property
```

```
def
```

```
safe_read_lock
```

```
(  
    self  
):
```

```
""" A safe read lock is a lock that does nothing if the thread already
```

```
has a write lock, otherwise it acquires a read lock. This is necessary
```

```
to prevent DowngradeLockErrors, which can happen when updating the
```

```
search cache in the presence of composite columns. Updating the search
```

```
cache holds an exclusive lock, but searching a composite column
```

involves reading field values via ProxyMetadata which tries to get a shared lock. There may be other scenarios that trigger this as well.

This property returns a new lock object on every access. This lock object is not recursive (for performance) and must only be used in a with statement as ``with cache.safe\_read\_lock:`` otherwise bad things will happen."""

return

```
SafeReadLock  
(  
    self  
    .  
    read_lock  
)
```

@write_api

def

```
ensure_has_search_category  
(  
    self  
    ,  
    fail_on_existing  
    =  
    True  
):
```

if

```
len
(
self
.
_search_api
.
saved_searches
.
names
())

>

0
:

self
.
field_metadata
.
add_search_category
(
label
=
'search'
,
name
=
-
(
'Saved searches'
),
fail_on_existing
=
fail_on_existing
)
```

```
def
_initialize_dynamic_categories
(
self
):

# Reconstruct the user categories, putting them into field_metadata

fm
=

self
.field_metadata

fm
.remove_dynamic_categories
()

for
user_cat

in
sorted
(
self
.field_metadata
.remove_dynamic_categories
(),
key=lambda x: x[0]
),
```

```
key
=
sort_key
):

cat_name

=

'@'

+

user_cat

# add the '@' to avoid name collision

while

cat_name
:

try
:

fm
.
add_user_category
(
label
=
cat_name
,

name
=
user_cat
)
```

```
except
```

```
ValueError
```

```
:
```

```
break
```

```
# Can happen since we are removing dots and adding parent categories ourselves
```

```
cat_name
```

```
=
```

```
cat_name
```

```
.
```

```
rpartition
```

```
(
```

```
''
```

```
)[
```

```
0
```

```
]
```

```
# add grouped search term user categories
```

```
muc
```

```
=
```

```
frozenset
```

```
(
```

```
self
```

```
.
```

```
_pref
```

```
(
```

```
'grouped_search_make_user_categories'
```

```
,
```



```
[[]))

for
cat

in

sorted
(
self
.
 _pref
(
'grouped_search_terms'
,

{}),

key
=
sort_key
):

if

cat

in

muc
:

# There is a chance that these can be duplicates of an existing

# user category. Print the exception and continue.

try
```

```
:  
  
self  
.  
field_metadata  
.  
add_user_category  
(  
label  
=  
'@'  
+  
  
cat  
,  
  
name  
=  
cat  
)  
  
except  
  
ValueError  
:  
  
traceback  
.  
print_exc  
()  
  
self  
.  
_ensure_has_search_category  
()
```

```
self
.
field_metadata
.
add_grouped_search_terms
(

self
.
.
 _pref
(
'grouped_search_terms'
,

{}})

self
.
.
 _refresh_search_locations
()

@write_api

def

initialize_dynamic
(
self
):

self
.
.
 backend
.
.
 dirty_books_with_dirtied_annotations
()
```

```
self
    .
    dirtied_cache

=

{
x
:
i

for

i
,

x

in

enumerate
(
self
    .
    backend
    .
    dirtied_books
    ())}

if

self
    .
    dirtied_cache
:

self
    .
    dirtied_sequence
```

```
=  
  
max  
(  
    itervalues  
(  
        self  
        .  
        dirtied_cache  
    ))  
+  
1  
  
self  
.  
_initialize_dynamic_categories  
()
```

@write_api

def

```
initialize_template_cache  
(  
    self  
):
```

```
self  
.  
formatter_template_cache
```

```
=
```

```
{}
```

@write_api

def

set_user_template_functions

(
self

,

user_template_functions

):

self

.

backend

.

set_user_template_functions

(
user_template_functions
)

@write_api

def

clear_composite_caches

(
self

,

book_ids

=

None

):

for

field

in

```
intervalvalues  
(  
    self  
    .  
    composites  
):
```

```
field  
.  
clear_caches  
(  
    book_ids  
    =  
    book_ids  
)
```

@write_api

def

```
clear_search_caches  
(  
    self  
    ,  
    book_ids  
    =  
    None  
):
```

```
self  
.  
clear_search_cache_count
```

+=

```
1
```

```
self
```

```
.
```

```
_search_api
```

```
.
```

```
update_or_clear
```

```
(
```

```
self
```

```
,
```

```
book_ids
```

```
)
```

```
self
```

```
.
```

```
vls_for_books_cache
```

```
=
```

```
None
```

```
self
```

```
.
```

```
vls_for_books_lib_in_process
```

```
=
```

```
None
```

```
@read_api
```

```
def
```

```
last_modified
```

```
(
```



```
self
):

return

self
.
backend
.
last_modified
()

@write_api

def

clear_caches
(
self
,

book_ids
=
None
,

template_cache
=
True
,

search_cache
=
True
):

if
```

```
template_cache
:

self
.
_initialize_template_cache
()

# Clear the formatter template cache

for
field
in
itervalues
(
self
.
fields
):

if
hasattr
(
field
,
'clear_caches'
):

field
.
clear_caches
(
book_ids
=
```

```
book_ids
)

# Clear the composite cache and ondevice caches

if

book_ids
:

for

book_id

in

book_ids
:

self
.
format_metadata_cache
.
pop
(
book_id
,
None
)

else
:

self
.
format_metadata_cache
.
```

```
clear
()

if

search_cache
:

self
.
_clear_search_caches
(
book_ids
)
```

```
@write_api
```

```
def

reload_from_db
(
self
,

clear_caches
=
True
):
```

```
if

clear_caches
:

self
.
```

```
_clear_caches
()

with

self
.
backend
.
conn
:

# Prevent other processes, such as calibredb from interrupting the reload by locking the db

self
.
backend
.
prefs
.
load_from_db
()

self
.
_search_api
.
saved_searches
.
load_from_db
()

for

in

intervalvalues
```

```
(
self
.
fields
):

if

hasattr
(
field
,
'table'
):

field
.
table
.
read
(
self
.
backend
)

# Reread data from metadata.db

@property

def
field_metadata
(
self
):
```

```
return
```

```
self
```

```
.
```

```
backend
```

```
.
```

```
field_metadata
```

```
def
```

```
    _get_metadata
```

```
(
```

```
    self
```

```
,
```

```
    book_id
```

```
,
```

```
    get_user_categories
```

```
=
```

```
    True
```

```
):
```

```
    # {{{
```

```
    mi
```

```
=
```

```
    Metadata
```

```
(
```

```
    None
```

```
,
```

```
    template_cache
```

```
=
```

```
    self
```

```
.
```

```
    formatter_template_cache
```

```
)  
  
mi  
.  
_proxy_metadata
```

```
=  
  
ProxyMetadata  
(  
self  
,  
  
book_id  
,  
  
formatter  
=  
mi  
.  
formatter  
)
```

```
author_ids  
  
=  
  
self  
.  
_field_ids_for  
(  
'authors'  
,  
  
book_id  
)
```

```
adata
```



```
=  
  
self  
.  
_author_data  
(  
author_ids  
)
```

```
aut_list
```

```
=  
  
[  
adata  
[  
i  
]  
]
```

```
for
```

```
i
```

```
in
```

```
author_ids  
]
```

```
aum
```

```
=
```

```
[]
```

```
aus
```

```
=
```

```
{}
```

```
aul
```

```
=
```

```
{}
```

```
for
```

```
rec
```

```
in
```

```
aut_list
```

```
:
```

```
aut
```

```
=
```

```
rec
```

```
[
```

```
'name'
```

```
]
```

```
aum
```

```
.
```

```
append
```

```
(
```

```
aut
```

```
)
```

```
aus
```

```
[
```

```
aut
```

```
]
```

```
=
```

```
rec
[
'sort'
]
```

```
aul
[
aut
]
```

```
=
```

```
rec
[
'link'
]
```

```
mi
.
title
```

```
=
```

```
self
.
_field_for
(
'title'
,
book_id
,
```

```
default_value
=
-
(
'Unknown'
))
```

```
mi
.
authors

=

aum

mi
.
author_sort

=

self
.
_field_for
(
'author_sort'
,
book_id
,

default_value
=
_
(
'Unknown'
))

mi
.
author_sort_map

=

aus
```

```
mi
.
author_link_map
```

```
=
```

```
aul
```

```
mi
.
comments
```

```
=
```

```
self
.
_field_for
(
'comments'
,
book_id
)
```

```
mi
.
publisher
```

```
=
```

```
self
.
_field_for
(
'publisher'
,
book_id
)
```

```
n
=
utcnow
()

mi
.
timestamp
=
self
.
_field_for
(
'timestamp'
,
book_id
,
default_value
=
n
)

mi
.
pubdate
=
self
.
_field_for
(
'pubdate'
```

```
,  
  
book_id  
,  
  
default_value  
=  
n  
)
```

```
mi  
.  
uuid  
  
=  
  
self  
.  
_field_for  
(  
    'uuid'  
,  
  
    book_id  
,
```

```
default_value  
=  
'dummy'  
)
```

```
mi  
.  
title_sort  
  
=  
  
self  
.  
_field_for
```

```
(
'sort'
,

book_id
,

default_value
=
_
(
'Unknown'
))

mi
.
last_modified

=

self
.
_field_for
(
'last_modified'
,

book_id
,

default_value
=
n
)

formats

=
```



```
self
.
_field_for
(
'formats'
,

book_id
)

mi
.
format_metadata

=

{}

mi
.
languages

=

list
(
self
.
_field_for
(
'languages'
,

book_id
))

if

not
```

```
formats
:

good_formats

=

None

else
:

mi
.
format_metadata

=

FormatMetadata
(
self
,
book_id
,
formats
)

good_formats

=

FormatsList
(
sorted
(
formats
),
```

```
mi
.  
format_metadata  
)
```

```
# These three attributes are returned by the db2 get_metadata(),
```

```
# however, we dont actually use them anywhere other than templates, so
```

```
# they have been removed, to avoid unnecessary overhead. The templates
```

```
# all use _proxy_metadata.
```

```
# mi.book_size = self._field_for('size', book_id, default_value=0)
```

```
# mi.ondesign_col = self._field_for('ondesign', book_id, default_value="")
```

```
# mi.db_approx_formats = formats
```

```
mi  
.  
formats
```

```
=
```

```
good_formats
```

```
mi  
.  
has_cover
```

```
=
```

```
-
(
'Yes'
)

if

self

.
_field_for
(
'cover'
,

book_id
,

default_value
=
False
)

else

"

mi

.
tags

=

list
(
self

.
_field_for
(
'tags'
,
```

```
book_id
,
default_value
=
()))
```

```
mi
.
series
```

```
=
```

```
self
.
_field_for
(
'series'
,
book_id
)
```

```
if
```

```
mi
.
series
:
```

```
mi
.
series_index
```

```
=
```

```
self
.
_field_for
(
```

```
'series_index'
```

```
,
```

```
book_id
```

```
,
```

```
default_value
```

```
=
```

```
1.0
```

```
)
```

```
mi
```

```
.
```

```
rating
```

```
=
```

```
self
```

```
.
```

```
_field_for
```

```
(
```

```
'rating'
```

```
,
```

```
book_id
```

```
)
```

```
mi
```

```
.
```

```
set_identifiers
```

```
(
```

```
self
```

```
.
```

```
_field_for
```

```
(
```

```
'identifiers'
```

```
,
```

```
book_id
```

```
,
```

```
default_value  
=  
{})
```

```
mi  
.  
application_id  
=  

```

```
book_id
```

```
mi  
.  
id  
=  

```

```
book_id
```

```
composites  
=  

```

```
[]
```

```
for
```

```
key  
,
```

```
meta
```

```
in
```

```
self  
.
```

```
field_metadata
.
custom_iteritems
():

mi
.
set_user_metadata
(
key
,

meta
)

if

meta
[
'datatype'
]

==

'composite'
:

composites
.
append
(
key
)

else
:

val
```



```
=  
  
self  
.  
_field_for  
(  
key  
,  
  
book_id  
)  
  
if  
  
isinstance  
(  
val  
,  
  
tuple  
):  
  
val  
  
=  
  
list  
(  
val  
)  
  
extra  
  
=  
  
self  
.  
_field_for  
(
```

```
key
+
'_index'
,

book_id
)

mi
.
set
(
key
,

val
=
val
,

extra
=
extra
)

for

key

in

composites
:

mi
.
set
(
key
,
```

```
val
=
self
.
_composite_for
(
key
,
book_id
,
mi
))

user_cat_vals
=
{}

if
get_user_categories
:

user_cats
=

self
.
_pref
(
'user_categories'
,
{})
```

for

ucat

in

user_cats

:

res

=

[]

for

name

,

cat

,

ign

in

user_cats

[

ucat

]:

v

=

mi

.

get

(

```
cat
,
None
)
```

```
if
isinstance
(
v
,
list
):
```

```
if
name
in
v
:
```

```
res
.
append
([
name
,
cat
])
```

```
elif
name
```

```
==
```

```
v
:

res
.
append
([
name
,
cat
])

user_cat_vals
[
ucat
]

=

res

mi
.
user_categories

=

user_cat_vals

return

mi

# }}}}
```

[\[docs\]](#)

@api

def

init

(
self
):

'''

Initialize this cache with data from the backend.

'''

with

self

.
write_lock
:

self

.
backend
.
read_tables
()

bools_are_tristate

=

```
self
.  
backend  
.  
prefs  
[  
    'bools_are_tristate'  
]
```

for

```
field  
,
```

```
table
```

in

```
iteritems  
(  
    self  
    .  
    backend  
    .  
    tables  
):
```

```
self
```

```
.  
fields  
[  
    field  
]
```

```
=
```

```
create_field  
(  
    field  
,
```



```
table
,

bools_are_tristate
,

self
.
backend
.
get_template_functions
)

if

table
.
metadata
[
'datatype'
]

==

'composite'
:

self
.
composites
[
field
]

=

self
.
fields
```

```
[
field
]

self
.
fields
[
'ondevice'
]

=

create_field
(
'ondevice'
,

VirtualTable
(
'ondevice'
),

bools_are_tristate
,

self
.
backend
.
get_template_functions
)

for
name
,
```

```
field
```

```
in
```

```
iteritems
```

```
(
```

```
self
```

```
.
```

```
fields
```

```
):
```

```
if
```

```
name
```

```
[
```

```
0
```

```
]
```

```
==
```

```
'#'
```

```
and
```

```
name
```

```
.
```

```
endswith
```

```
(
```

```
'_index'
```

```
):
```

```
field
```

```
.
```

```
series_field
```

```
=
```

```
self
```

```
.
```

```
fields
```

```
[
    name
    [:
    -
    len
    (
    '_index'
    )]]

self
.
fields
[
    name
    [:
    -
    len
    (
    '_index'
    )]]
.
index_field

=

field

elif

name

==

'series_index'
:

field
.
series_field
```

```
=  
  
self  
.  
fields  
[  
    'series'  
]  
  
self  
.  
fields  
[  
    'series'  
]  
.  
index_field
```

```
=  
  
field  
  
elif  
  
name  
  
==  
  
'authors'  
:  
  
field  
.  
author_sort_field  
  
=  
  
self  
.  
fields
```

```
[
    'author_sort'
]

elif

name

==

'title'
:

field
.
title_sort_field

=

self
.
fields
[
    'sort'
]

if

self
.
backend
.
prefs
[
    'update_all_last_mod_dates_on_start'
]:

self
.
```

```
update_last_modified
(
    self
    .
    all_book_ids
    ())

self
    .
    backend
    .
    prefs
    .
    set
    (
        'update_all_last_mod_dates_on_start'
        ,
        False
    )

# Cache Layer API {{{

\[docs\]

@write_api

def

add_listener
(
    self
    ,

    event_callback_function
):
```

```
'''
    Register a callback function that will be called after certain actions are
    taken on this database. The function must take three arguments:

    (:class:`EventType`, library_id, event_type_specific_data)
'''

self
.
event_dispatcher
.
library_id
=

getattr
(
self
,
'server_library_id'
,
self
.
library_id
)

self
.
event_dispatcher
.
add_listener
(
event_callback_function
)
```


@write_api

def

remove_listener

(
self
,

event_callback_function
):

self

.
event_dispatcher

.
remove_listener
(
event_callback_function
)

[\[docs\]](#)

@read_api

def

field_for

(
self
,

name
,

book_id
,

```
default_value
```

```
=
```

```
None
```

```
):
```

```
'''
```

Return the value of the field ``name`` for the book identified

by ``book\_id``. If no such book exists or it has no defined

value for the field ``name`` or no such field exists, then

``default\_value`` is returned.

``default\_value`` is not used for title, title_sort, authors, author_sort

and series_index. This is because these always have values in the db.

``default\_value`` is used for all custom columns.

The returned value for is_multiple fields are always tuples, even when

no values are found (in other words, default_value is ignored). The

exception is identifiers for which the returned value is always a dict.

The returned tuples are always in link order, that is, the order in

which they were created.

```
'''
```

```
if
```

```
self
```

```
.
```

```
composites
```

and

name

in

self

.

composites

:

return

self

.

composite_for

(

name

,

book_id

,

default_value

=

default_value

)

try

:

field

=

self

.

fields

```
[
    name
]

except

KeyError
:

return

default_value

if

field
.
is_multiple
:

default_value

=

field
.
default_value

try
:

return

field
.
for_book
(
```

```
book_id
,

default_value
=
default_value
)
```

```
except
```

```
(
    KeyError
,
    IndexError
):
```

```
return
```

```
default_value
```

```
\[docs\]
```

```
@read_api
```

```
def
```

```
fast_field_for
(
    self
,
    field_obj
,
    book_id
,
    default_value
```

```
=
None
):

' Same as field_for, except that it avoids the extra lookup to get the field object '

if

field_obj
.
is_composite
:

return

field_obj
.
get_value_with_cache
(
book_id
,
self
.
_get_proxy_metadata
)

if

field_obj
.
is_multiple
:

default_value

=
```

```
field_obj
.  
default_value
```

```
try  
:
```

```
return
```

```
field_obj  
.  
for_book  
(  
book_id  
,  
  
default_value  
=  
default_value  
)
```

```
except
```

```
(  
KeyError  
,  
  
IndexError  
):
```

```
return
```

```
default_value
```

```
\[docs\]
```

```
@read_api
```

```
def
all_field_for
(
self
,
field
,
book_ids
,
default_value
=
None
):

' Same as field_for, except that it operates on multiple books at once '

field_obj
=

self
.
fields
[
field
]

return

{
book_id
:
self
.
_fast_field_for
```



```
(  
    field_obj  
,  
  
    book_id  
,  
  
    default_value  
    =  
    default_value  
)
```

for

book_id

in

```
book_ids  
}
```

@read_api

def

composite_for

```
(  
    self
```

,

name

,

book_id

,

mi

=

None

,

```
default_value
```

```
=
```

```
"
```

```
):
```

```
try
```

```
:
```

```
f
```

```
=
```

```
self
```

```
.
```

```
fields
```

```
[
```

```
name
```

```
]
```

```
except
```

```
KeyError
```

```
:
```

```
return
```

```
default_value
```

```
if
```

```
mi
```

```
is
```

```
None
```

```
:
```

return

f

.

get_value_with_cache

(

book_id

,

self

.

_get_proxy_metadata

)

else

:

return

f

.

_render_composite_with_cache

(

book_id

,

mi

,

mi

.

formatter

,

mi

.

template_cache

)

[\[docs\]](#)

@read_api

def

field_ids_for

(
self
,

name

,

book_id

):

'''

Return the ids (as a tuple) for the values that the field ``name`` has on the book

identified by ``book\_id``. If there are no values, or no such book, or

no such field, an empty tuple is returned.

'''

try

:

return

self

.

fields

[

```
name
]
.
ids_for_book
(
book_id
)
```

except

```
(
KeyError
,
IndexError
):
```

return

```
()
```

[\[docs\]](#)

@read_api

def

```
books_for_field
(
self
,
name
,
item_id
):
```

```
"""  
  
    Return all the books associated with the item identified by  
  
    ``item_id``, where the item belongs to the field ``name``.  
  
    Returned value is a set of book ids, or the empty set if the item  
  
    or the field does not exist.  
  
    """
```

```
try  
:  
  
return  
  
self  
.fields  
[  
    name  
]  
.books_for  
(  
    item_id  
)
```

```
except  
  
(  
    KeyError  
,  
  
    IndexError  
):
```

```
return
```

```
set  
()
```

```
\[docs\]
```

```
@read_api
```

```
def
```

```
all_book_ids
```

```
(  
    self  
,
```

```
    type
```

```
    =
```

```
    frozenset
```

```
):
```

```
'''
```

```
    Frozen set of all known book ids.
```

```
'''
```

```
return
```

```
type
```

```
(
```

```
    self
```

```
    .
```

```
    fields
```

```
    [
```

```
        'uuid'
```

```
    ]
```

```
    .
```

```
    table
```

```
.  
book_col_map  
)
```

[\[docs\]](#)

```
@read_api
```

```
def
```

```
all_field_ids
```

```
(  
self  
,
```

```
name  
):
```

```
"""
```

```
    Frozen set of ids for all values in the field ``name``.
```

```
    """
```

```
return
```

```
frozenset
```

```
(  
iter  
(  
self
```

```
.  
fields  
[  
name  
]))
```


[\[docs\]](#)

@read_api

def

all_field_names

(

self

,

field

):

""" Frozen set of all fields names (should only be used for many-one and many-many fields) """

if

field

==

'formats'

:

return

frozenset

(

self

.

fields

[

field

]

.

table

.

col_book_map

```
)

try
:

return

frozenset
(
self
.
fields
[
field
]
.
table
.
id_map
.
values
())

except

AttributeError
:

raise

ValueError
(
'
%s
is not a many-one or many-many field'

%
```

```
field
)
```

```
\[docs\]
```

```
@read_api
```

```
def
```

```
get_usage_count_by_id
(
    self
    ,
```

```
field
):
```

```
""" Return a mapping of id to usage count for all values of the specified
        field, which must be a many-one or many-many field. """
```

```
try
:
```

```
return
```

```
{
    k
    :
    len
    (
        v
    )
}
```

```
for
```

```
k
,
```

```
v
```

```
in
```

```
iteritems
```

```
(
```

```
self
```

```
.
```

```
fields
```

```
[
```

```
field
```

```
]
```

```
.
```

```
table
```

```
.
```

```
col_book_map
```

```
)}
```

```
except
```

```
AttributeError
```

```
:
```

```
raise
```

```
ValueError
```

```
(
```

```
,
```

```
%s
```

```
is not a many-one or many-many field'
```

```
%
```

```
field
```

```
)
```

```
\[docs\]
```

```
@read_api
```

```

def
get_id_map
(
self
,
field
):

""" Return a mapping of id numbers to values for the specified field.

        The field must be a many-one or many-many field, otherwise a ValueError
        is raised. """

try
:

return

self
.
fields
[
field
]
.
table
.
id_map
.
copy
()

except

```

```
AttributeError
:

if

field

==

'title'
:

return

self
.
fields
[
field
]
.
table
.
book_col_map
.
copy
()

raise

ValueError
(
'
%s
is not a many-one or many-many field'

%

field
```

```
)
```

[\[docs\]](#)

```
@read_api
```

```
def
```

```
get_item_name
```

```
(
```

```
self
```

```
,
```

```
field
```

```
,
```

```
item_id
```

```
):
```

```
""" Return the item name for the item specified by item_id in the  
specified field. See also :meth:`get_id_map`."""
```

```
return
```

```
self
```

```
.
```

```
fields
```

```
[
```

```
field
```

```
]
```

```
.
```

```
table
```

```
.
```

```
id_map
```

```
[
```

```
item_id
```

```
]
```

[\[docs\]](#)

@read_api

def

get_item_id

(

self

,

field

,

item_name

):

' Return the item id for item_name (case-insensitive) '

rmap

=

{

icu_lower

(

v

)

if

isinstance

(

v

,

str

)

else


```
v
:
k

for

k
,
v

in

iteritems
(
self
.
fields
[
field
]
.
table
.
id_map
)}

return

rmap
.
get
(
icu_lower
(
item_name
)

if

isinstance
```

```
(
    item_name
    ,
    str
)

else

    item_name
    ,

    None
)

\[docs\]

@read_api

def

get_item_ids
(
    self
    ,

    field
    ,

    item_names
):

    ' Return the item id for item_name (case-insensitive) '

    rmap

    =

    {
```

```
icu_lower
```

```
(
```

```
v
```

```
)
```

```
if
```

```
isinstance
```

```
(
```

```
v
```

```
,
```

```
str
```

```
)
```

```
else
```

```
v
```

```
:
```

```
k
```

```
for
```

```
k
```

```
,
```

```
v
```

```
in
```

```
iteritems
```

```
(
```

```
self
```

```
.
```

```
fields
```

```
[
```

```
field
```

```
]
```

```
.
```

```
table
```

```
.
```

```
id_map
```

```
)}  
}
```

return

```
{  
name  
:  
rmap  
.  
get  
(  
icu_lower  
(  
name  
)  
)
```

if

```
isinstance  
(  
name  
,  
  
str  
)
```

else

```
name  
,
```

None

```
)
```

for

```
name
```

in

```
item_names  
}
```

[\[docs\]](#)

@read_api

def

author_data

(

self

,

author_ids

=

None

):

'''

Return author data as a dictionary with keys: name, sort, link

If no authors with the specified ids are found an empty dictionary is

returned. If author_ids is None, data for all authors is returned.

'''

af

=

self

.

fields

[

'authors'

]

if

```
author_ids
```

```
is
```

```
None
```

```
:
```

```
return
```

```
{
```

```
aid
```

```
:
```

```
af
```

```
.
```

```
author_data
```

```
(
```

```
aid
```

```
)
```

```
for
```

```
aid
```

```
in
```

```
af
```

```
.
```

```
table
```

```
.
```

```
id_map
```

```
}
```

```
return
```

```
{
```

```
aid
```

```
:
```

```
af
```

```
.
```

```
author_data
```

```
(
aid
)

for

aid

in

author_ids

if

aid

in

af
.
table
.
id_map
}
```

[\[docs\]](#)

@read_api

```
def

format_hash
(
self
,

book_id
,

fmt
):
```

```
""" Return the hash of the specified format for the specified book. The  
kind of hash is backend dependent, but is usually SHA-256. """
```

```
try
```

```
:
```

```
name
```

```
=
```

```
self
```

```
.
```

```
fields
```

```
[
```

```
'formats'
```

```
]
```

```
.
```

```
format_fname
```

```
(
```

```
book_id
```

```
,
```

```
fmt
```

```
)
```

```
path
```

```
=
```

```
self
```

```
.
```

```
_field_for
```

```
(
```

```
'path'
```

```
,
```

```
book_id
```



```
)
.
replace
(
    '/'
,

os
.
sep
)

except
:

raise

NoSuchFormat
(
    'Record
    %d
    has no fmt:
    %s
    '
    %
    (
        book_id
    ,

    fmt
    ))

return

self
.
backend
.
format_hash
(
```

```
book_id
```

```
,
```

```
fmt
```

```
,
```

```
name
```

```
,
```

```
path
```

```
)
```

```
\[docs\]
```

```
@api
```

```
def
```

```
format_metadata
```

```
(
```

```
self
```

```
,
```

```
book_id
```

```
,
```

```
fmt
```

```
,
```

```
allow_cache
```

```
=
```

```
True
```

```
,
```

```
update_db
```

```
=
```

```
False
```

```
):
```

```
'''
```

Return the path, size and mtime for the specified format for the specified book.

You should not use path unless you absolutely have to,

since accessing it directly breaks the threadsafe guarantees of this API. Instead use

the :meth:`copy\_format\_to` method.

:param allow_cache: If ``True`` cached values are used, otherwise a

slow filesystem access is done. The cache values could be out of date

if access was performed to the filesystem outside of this API.

:param update_db: If ``True`` The max_size field of the database is updated for this book.

'''

if

not

fmt

:

return

{}

fmt

=

fmt

.

upper

()

allow_cache and update_db are mutually exclusive. Give priority to update_db

if

allow_cache

and

not

update_db

:

x

=

self

.

format_metadata_cache

[

book_id

]

.

get

(

fmt

,

None

)

if

x

is

not

None

:

return

x

with

self

.

safe_read_lock

:

try

:

name

=

self

.

fields

[

'formats'

]

.

format_fname

(

book_id

,

fmt

)

```
path

=

self
.
_field_for
(
'path'
,

book_id
)
.
replace
(
'/'
,

os
.
sep
)

except
:

return

{}

ans

=

{}

if
```

```
path
```

```
and
```

```
name
```

```
:
```

```
ans
```

```
=
```

```
self
```

```
.
```

```
backend
```

```
.
```

```
format_metadata
```

```
(
```

```
book_id
```

```
,
```

```
fmt
```

```
,
```

```
name
```

```
,
```

```
path
```

```
)
```

```
self
```

```
.
```

```
format_metadata_cache
```

```
[
```

```
book_id
```

```
][
```

```
fmt
```

```
]
```

```
=
```

```
ans

if
    update_db

and
    'size'

in
    ans
    :

with
    self
    .
    write_lock
    :

    max_size

    =

    self
    .
    fields
    [
        'formats'
    ]
    .
    table
    .
    update_fmt
    (
        book_id
    ,
```



```
fmt
,

name
,

ans
[
'size'
],

self
.
backend
)

self
.
fields
[
'size'
]
.
table
.
update_sizes
({
book_id
:

max_size
})

return

ans

@read_api
```

```
def
format_files
(
self
,
book_id
):

field

=

self
.
fields
[
'formats'
]

fmts

=

field
.
table
.
book_col_map
.
get
(
book_id
,

())

return
```

```
{
  fmt
  :
  field
  .
  format_fname
  (
  book_id
  ,

  fmt
  )

  for

  fmt

  in

  fmts
}
```

@read_api

```
def

format_db_size
(
  self
  ,

  book_id
  ,

  fmt
  ):

  field
```

```
=
```

```
self
```

```
.
```

```
fields
```

```
[
```

```
'formats'
```

```
]
```

```
return
```

```
field
```

```
.
```

```
format_size
```

```
(
```

```
book_id
```

```
,
```

```
fmt
```

```
)
```

```
\[docs\]
```

```
@read_api
```

```
def
```

```
pref
```

```
(
```

```
self
```

```
,
```

```
name
```

```
,
```

```
default
```

```
=
```

None

,

namespace

=

None

):

' Return the value for the specified preference or the value specified as ``default`` if the preference is not set. '

if

namespace

is

not

None

:

return

self

.

backend

.

prefs

.

get_namespaced

(

namespace

,

name

,

default

)

return

self

.

backend

.

prefs

.

get

(

name

,

default

)

[\[docs\]](#)

@write_api

def

set_pref

(

self

,

name

,

val

,

namespace

=

None

):

' Set the specified preference to the specified value. See also :meth:`pref`. '

if

namespace

is

not

None

:

self

.

backend

.

prefs

.

set_namespaced

(

namespace

,

name

,

val

)

return

self

.

backend

.

prefs

.

set

```
(
    name
    ,
    val
)

if

name

in

(
    'grouped_search_terms'
    ,
    'virtual_libraries'
):

    self
        .
        _clear_search_caches
        ()

    if

    name

    in

    dynamic_category_preferences
    :

    self
        .
        _initialize_dynamic_categories
        ()
```


[\[docs\]](#)

@api

def

get_metadata

(

self

,

book_id

,

get_cover

=

False

,

get_user_categories

=

True

,

cover_as_data

=

False

):

'''

*Return metadata for the book identified by book_id as a
:class:`calibre.ebooks.metadata.book.base.Metadata` object.*

Note that the list of formats is not verified. If get_cover is True,

the cover is returned, either a path to temp file as mi.cover or if

cover_as_data is True then as mi.cover_data.

```
'''

# Check if virtual_libraries_for_books rebuilt its cache. If it did then

# we must clear the composite caches so the new data can be taken into

# account. Clearing the caches requires getting a write lock, so it must

# be done outside of the closure of _get_metadata().

composite_cache_needs_to_be_cleared

=

False

with

self

.

safe_read_lock

:

vl_cache_was_none

=

self

.

vls_for_books_cache

is

None
```

```
mi
=
self
.
_get_metadata
(
book_id
,
get_user_categories
=
get_user_categories
)

if
vl_cache_was_none
and
self
.
vls_for_books_cache
is
not
None
:

composite_cache_needs_to_be_cleared
=
True
```

```
if
    composite_cache_needs_to_be_cleared
    :

    try
    :

        self
        .
        clear_composite_caches
        ()

    except
        LockingError
        :

        # We can't clear the composite caches because a read lock is set.

        # As a consequence the value of a composite column that calls

        # virtual_libraries() might be wrong. Oh well. Log and keep running.

        print
        (
        "Couldn't get write lock after vls_for_books_cache was loaded"
        ,

        file
        =
        sys
        .
        stderr
        )
```

```
traceback
```

```
.
```

```
print_exc
```

```
()
```

```
if
```

```
get_cover
```

```
:
```

```
if
```

```
cover_as_data
```

```
:
```

```
cdata
```

```
=
```

```
self
```

```
.
```

```
cover
```

```
(
```

```
book_id
```

```
)
```

```
if
```

```
cdata
```

```
:
```

```
mi
```

```
.
```

```
cover_data
```

```
=
```

```
(  
'jpeg'  
,
```

```
cdata  
)
```

```
else  
:
```

```
mi  
.  
cover  
  
=
```

```
self  
.  
cover  
(  
book_id  
,  
  
as_path  
=  
True  
)
```

```
return
```

```
mi
```

```
\[docs\]
```

```
@read_api
```

```
def
get_proxy_metadata
(
self
,
book_id
):

""" Like :meth:`get_metadata` except that it returns a ProxyMetadata
    object that only reads values from the database on demand. This is much
    faster than get_metadata when only a small number of fields need to be
    accessed from the returned metadata object. """

return

ProxyMetadata
(
self
,
book_id
)

\[docs\]

@api

def
cover
(
self
,
```

```

book_id
,

as_file
=
False
,

as_image
=
False
,

as_path
=
False
):

'''

    Return the cover image or None. By default, returns the cover as a
    bytestring.

    WARNING: Using as_path will copy the cover to a temp file and return
    the path to the temp file. You should delete the temp file when you are
    done with it.

    :param as_file: If True return the image as an open file object (a SpooledTemporaryFile)

    :param as_image: If True return the image as a QImage object

    :param as_path: If True return the image as a path pointing to a
                    temporary file

'''

```


if

as_file

:

ret

=

SpoiledTemporaryFile

(

SPOOL_SIZE

)

if

not

self

.

copy_cover_to

(

book_id

,

ret

):

return

ret

.

seek

(

0

)

elif

as_path
:

pt

=

PersistentTemporaryFile
(
'_dbcover.jpg'
)

with

pt
:

if

not

self

.

copy_cover_to
(
book_id
,

pt
):

return

ret

```
=

pt
.
name

else
:

buf

=

BytesIO
()

if
not
self
.
copy_cover_to
(
book_id
,
buf
):

return

ret

=

buf
.
```

```
getvalue
()

if

as_image
:

from

qt.core

import

QImage

i

=

QImage
()

i
.
loadFromData
(
ret
)

ret

=

i

return
```

```
ret
```

```
@read_api
```

```
def
```

```
cover_or_cache
```

```
(  
    self
```

```
,
```

```
    book_id
```

```
,
```

```
    timestamp
```

```
):
```

```
try
```

```
:
```

```
    path
```

```
=
```

```
    self
```

```
    .
```

```
    _field_for
```

```
(
```

```
    'path'
```

```
,
```

```
    book_id
```

```
)
```

```
    .
```

```
    replace
```

```
(
```

```
    '/'
```

```
,
```

```
os
.
sep
)

except

AttributeError
:

return

False
,

None
,

None

return

self
.
backend
.
cover_or_cache
(
path
,

timestamp
)

@read_api
```

```
def
cover_last_modified
(
self
,
book_id
):

try
:

path

=

self
.
_field_for
(
'path'
,
book_id
)
.
replace
(
'/'
,
os
.
sep
)

except
AttributeError
```

```
:  
  
    return  
  
    return  
  
    self  
    .  
    backend  
    .  
    cover_last_modified  
    (  
    path  
    )
```

[\[docs\]](#)

@read_api

```
def  
  
    copy_cover_to  
    (  
    self  
    ,  
  
    book_id  
    ,  
  
    dest  
    ,  
  
    use_hardlink  
    =  
    False  
    ,
```



```
report_file_size
=
None
):

'''
    Copy the cover to the file like object ``dest``. Returns False
    if no cover exists or dest is the same file as the current cover.

    dest can also be a path in which case the cover is
    copied to it if and only if the path is different from the current path (taking
    case sensitivity into account).
'''

try
:

path

=

self
.
_field_for
(
'path'
,
book_id
)
.
replace
(
'/'
,
```

```
os
.
sep
)

except

AttributeError
:

return

False

return

self
.
backend
.
copy_cover_to
(
path
,

dest
,

use_hardlink
=
use_hardlink
,

report_file_size
=
report_file_size
```

```
)
```

```
\[docs\]
```

```
@write_api
```

```
def
```

```
compress_covers
```

```
(
```

```
self
```

```
,
```

```
book_ids
```

```
,
```

```
jpeg_quality
```

```
=
```

```
100
```

```
,
```

```
progress_callback
```

```
=
```

```
None
```

```
):
```

```
'''
```

Compress the cover images for the specified books. A compression quality of 100 will perform lossless compression, otherwise lossy compression.

The progress callback will be called with the book_id and the old and new sizes for each book that has been processed. If an error occurs, the new size will be a string with the error details.

```
'''
```

```
jpeg_quality
```

```
=
```

```
max
```

```
(
```

```
10
```

```
,
```

```
min
```

```
(
```

```
jpeg_quality
```

```
,
```

```
100
```

```
))
```

```
path_map
```

```
=
```

```
{}
```

```
for
```

```
book_id
```

```
in
```

```
book_ids
```

```
:
```

```
try
```

```
:
```

```
path_map
```

```
[
```

```
book_id
]

=

self
.
_field_for
(
'path'
,

book_id
)
.
replace
(
'/'
,

os
.
sep
)

except

AttributeError
:

continue

self
.
backend
.
compress_covers
(
path_map
,
```

```
jpeg_quality
',

progress_callback
)
```

[\[docs\]](#)

@read_api

def

copy_format_to

(
self

,

book_id

,

fmt

,

dest

,

use_hardlink

=

False

,

report_file_size

=

None

):

'''

Copy the format ``fmt`` to the file like object ``dest``. If the

specified format does not exist, raises :class:`NoSuchFormat` error.

dest can also be a path (to a file), in which case the format is copied to it, iff

the path is different from the current path (taking case sensitivity

into account).

'''

fmt

=

(

fmt

or

"

)

.

upper

()

try

:

name

=

self

.

fields

[

'formats'

]

.

```
format_fname
```

```
(
```

```
book_id
```

```
,
```

```
fmt
```

```
)
```

```
path
```

```
=
```

```
self
```

```
.
```

```
_field_for
```

```
(
```

```
'path'
```

```
,
```

```
book_id
```

```
)
```

```
.
```

```
replace
```

```
(
```

```
'/'
```

```
,
```

```
os
```

```
.
```

```
sep
```

```
)
```

```
except
```

```
(
```

```
KeyError
```

```
,
```

```
AttributeError
```

```
):
```


raise

NoSuchFormat

```
(  
    'Record  
    %d  
    has no  
    %s  
    file'  
    %  
    (  
    book_id  
    ,  
  
    fmt  
    ))
```

return

```
self  
.  
backend  
.  
copy_format_to  
(  
    book_id  
    ,  
  
    fmt  
    ,  
  
    name  
    ,  
  
    path  
    ,  
  
    dest  
    ,
```

```
use_hardlink
=
use_hardlink
,

report_file_size
=
report_file_size
)
```

[\[docs\]](#)

@read_api

def

```
format_abspath
(
self
,

book_id
,

fmt
):
```

"""

Return absolute path to the e-book file of format `format`. You should

almost never use this, as it breaks the threadsafe promise of this API.

Instead use, :meth:`copy\_format\_to`.

Currently used only in calibredb list, the viewer, edit book,

compare_format to original format, open with, bulk metadata edit and

the catalogs (via `get_data_as_dict()`).

Apart from the viewer, open with and edit book, I don't believe any of

the others do any file write I/O with the results of this call.

'''

fmt

=

(

fmt

or

"

)

.

upper

()

try

:

path

=

self

.

_field_for

(

'path'

,

book_id

```
)  
.  
replace  
(  
    '/'  
    ,  
  
    os  
    .  
    sep  
    )  
  
except  
:  
  
return  
  
None  
  
if  
  
path  
:  
  
if  
  
fmt  
  
==  
  
    '__COVER_INTERNAL__'  
    :  
  
return  
  
self  
.  
backend
```

```
.  
cover_abspath  
(  
    book_id  
,  
  
    path  
)
```

```
else  
:
```

```
try  
:
```

```
name  
  
=
```

```
self  
.fields  
[  
    'formats'  
]
```

```
.format_fname  
(  
    book_id  
,  
  
    fmt  
)
```

```
except  
:
```

```
return
```

None

if

name
:

return

self

.
backend

.
format_abspath
(
book_id

,

fmt
,

name
,

path
)

[\[docs\]](#)

@read_api

def

has_format
(
self
,

```
book_id
,

fmt
):

'Return True iff the format exists on disk'

fmt

=

(
fmt

or

"
)
.
upper
()

try
:

name

=

self
.
fields
[
'formats'
]
.
format_fname
(
```

```
book_id
,

fmt
)

path

=

self
.
_field_for
(
'path'
,
book_id
)
.
replace
(
'/'
,
os
.
sep
)

except
:

return

False

return
```



```
self
.  
backend  
.  
has_format  
(  
book_id  
,  
  
fmt  
,  
  
name  
,  
  
path  
)
```

[\[docs\]](#)

@api

def

```
save_original_format  
(  
self  
,  
  
book_id  
,  
  
fmt  
):
```

' Save a copy of the specified format as ORIGINAL_FORMAT, overwriting any existing ORIGINAL_FORMAT. '

```
fmt
```

```
=
```

```
fmt
```

```
.
```

```
upper
```

```
()
```

```
if
```

```
'ORIGINAL'
```

```
in
```

```
fmt
```

```
:
```

```
raise
```

```
ValueError
```

```
(
```

```
'Cannot save original of an original fmt'
```

```
)
```

```
fmtfile
```

```
=
```

```
self
```

```
.
```

```
format
```

```
(
```

```
book_id
```

```
,
```

```
fmt
```

```
,
```

```
as_file
```

```
=
```

```
True
)

if
fmtfile

is

None
:

return

False

with

fmtfile
:

nfmt

=

'ORIGINAL_'
+
fmt

return

self
.
add_format
(
book_id
,
```

```
nfmt
',

fmtfile
',

run_hooks
=
False
)
```

[\[docs\]](#)

@write_api

def

```
restore_original_format
(
self
',

book_id
',

original_fmt
):
```

""" Restore the specified format from the previously saved

ORIGINAL_FORMAT, if any. Return True on success. The ORIGINAL_FORMAT is

deleted after a successful restore. """

```
original_fmt
```

```
=
```

```
original_fmt
```

```
.
upper
()

fmt

=

original_fmt

.
partition
(
    ' '
    _
)[
    2
]

try
:

ofmt_name

=

self

.
fields
[
    'formats'
]

.
format_fname
(
    book_id
    ,
    original_fmt
)
```

```
path

=

self
.
_field_for
(
'path'
,

book_id
)
.
replace
(
'/'
,

os
.
sep
)

except

Exception
:

return

False

if

self
.
backend
.
is_format_accessible
```

```
(
    book_id
    ,
    original_fmt
    ,
    ofmt_name
    ,
    path
):
```

```
self
.
add_format
(
    book_id
    ,
    fmt
    ,
    BytesIO
    (),
    run_hooks
    =
    False
)
```

```
fmt_name

=

self
.
fields
[
    'formats'
]
```

```
.
format_fname
(
book_id
,

fmt
)

file_size

=

self
.
backend
.
rename_format_file
(
book_id
,

ofmt_name
,

original_fmt
,

fmt_name
,

fmt
,

path
)

self
.
fields
[
```



```
'formats'
]
.
table
.
update_fmt
(
book_id
,

fmt
,

fmt_name
,

file_size
,

self
.
backend
)

self
.
_remove_formats
({
book_id
:({
original_fmt
,}))

return

True

return
```

False

[\[docs\]](#)

@read_api

def

formats

(

self

,

book_id

,

verify_formats

=

True

):

'''

*Return tuple of all formats for the specified book. If verify_formats
is True, verifies that the files exist on disk.*

'''

ans

=

self

.

field_for

(

'formats'

,

```
book_id
)

if
verify_formats

and

ans
:

try
:

path

=

self
.
_field_for
(
'path'
,
book_id
)
.
replace
(
'/'
,
os
.
sep
)
```

```
except
```

```
:
```

```
return
```

```
()
```

```
def
```

```
verify
```

```
(
```

```
fmt
```

```
):
```

```
try
```

```
:
```

```
name
```

```
=
```

```
self
```

```
.
```

```
fields
```

```
[
```

```
'formats'
```

```
]
```

```
.
```

```
format_fname
```

```
(
```

```
book_id
```

```
,
```

```
fmt
```

```
)
```

```
except
:
```

```
return
```

```
False
```

```
return
```

```
self
```

```
.  
backend
```

```
.  
has_format  
(  
book_id
```

```
,  
  
fmt
```

```
,  
  
name
```

```
,  
  
path  
)
```

```
ans
```

```
=
```

```
tuple
```

```
(  
x
```

```
for
```

```
x
```

in

ans

if

verify

(
x
)

return

ans

[\[docs\]](#)

@api

def

format

(
self
,

book_id

,

fmt

,

as_file

=

False

,

as_path

=

False

```

',
preserve_filename
=
False
):

'''
    Return the e-book format as a bytestring or `None` if the format doesn't exist,
    or we don't have permission to write to the e-book file.

    :param as_file: If True the e-book format is returned as a file object. Note
        that the file object is a SpooledTemporaryFile, so if what you want to
        do is copy the format to another file, use :meth:`copy_format_to`
        instead for performance.

    :param as_path: Copies the format file to a temp file and returns the
        path to the temp file

    :param preserve_filename: If True and returning a path the filename is
        the same as that used in the library. Note that using
        this means that repeated calls yield the same
        temp file (which is re-created each time)
'''

fmt
=
(

```

```
fmt
```

```
or
```

```
"
```

```
)
```

```
.
```

```
upper
```

```
()
```

```
ext
```

```
=
```

```
(
```

```
' '
```

```
+
```

```
fmt
```

```
.
```

```
lower
```

```
())
```

```
if
```

```
fmt
```

```
else
```

```
"
```

```
if
```

```
as_path
```

```
:
```

```
if
```

```
preserve_filename
```

```
:
```


with

self

.

safe_read_lock

:

try

:

fname

=

self

.

fields

[

'formats'

]

.

format_fname

(

book_id

,

fmt

)

except

:

return

None

fname

```
+=
```

```
ext
```

```
bd
```

```
=
```

```
base_dir  
(
```

```
d
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
join
```

```
(
```

```
bd
```

```
,
```

```
'format_abspath'
```

```
)
```

```
try
```

```
:
```

```
os
```

```
.
```

```
makedirs
```

```
(
```

```
d
```

```
)
```

```
except
```

```
:
```

```
pass
```

```
ret
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
join
```

```
(
```

```
d
```

```
,
```

```
fname
```

```
)
```

```
try
```

```
:
```

```
self
```

```
.
```

```
copy_format_to
```

```
(
```

```
book_id
```

```
,
```

```
fmt
```

```
,
```

```
ret
```

```
)
```

except

NoSuchFormat

:

return

None

else

:

with

PersistentTemporaryFile

(

ext

)

as

pt

:

try

:

self

.

copy_format_to

(

book_id

,

fmt

,

pt

```
)

except

NoSuchFormat
:

return

None

ret

=

pt
.
name

elif

as_file
:

with

self
.
safe_read_lock
:

try
:

fname
```

```
=  
  
self  
.  
fields  
[  
    'formats'  
]  
.  
format_fname  
(  
    book_id  
    ,  
  
    fmt  
)
```

```
except  
:
```

```
return
```

```
None
```

```
fname
```

```
+=
```

```
ext
```

```
ret
```

```
=
```

```
SpooledTemporaryFile  
(  
    SPOOL_SIZE  
)
```

```
try
```

```
:
```

```
self
```

```
.
```

```
copy_format_to
```

```
(
```

```
book_id
```

```
,
```

```
fmt
```

```
,
```

```
ret
```

```
)
```

```
except
```

```
NoSuchFormat
```

```
:
```

```
return
```

```
None
```

```
ret
```

```
.
```

```
seek
```

```
(
```

```
0
```

```
)
```

```
# Various bits of code try to use the name as the default
```

```
# title when reading metadata, so set it
```

```
ret
.
name

=

fname

else
:

buf

=

BytesIO
()

try
:

self
.
copy_format_to
(
book_id
,
fmt
,
buf
)

except
```


NoSuchFormat

:

return

None

ret

=

buf

.

getvalue

()

return

ret

[\[docs\]](#)

@read_api

def

multisort

(

self

,

fields

,

ids_to_sort

=

None

,

virtual_fields

=

None

):

'''

*Return a list of sorted book ids. If ids_to_sort is None, all book ids
are returned.*

*fields must be a list of 2-tuples of the form (field_name,
ascending=True or False). The most significant field is the first
2-tuple.*

'''

ids_to_sort

=

self

.

_all_book_ids

()

if

ids_to_sort

is

None

else

```
ids_to_sort
```

```
get_metadata
```

```
=
```

```
self
```

```
.
```

```
_get_proxy_metadata
```

```
lang_map
```

```
=
```

```
self
```

```
.
```

```
fields
```

```
[
```

```
'languages'
```

```
]
```

```
.
```

```
book_value_map
```

```
virtual_fields
```

```
=
```

```
virtual_fields
```

```
or
```

```
{}
```

```
fm
```

```
=
```

```
{  
  'title'  
  :  
  'sort'  
  ,  
  
  'authors'  
  :  
  'author_sort'  
}
```

def

```
sort_key_func  
(  
  field  
):
```

'Handle series type fields, virtual fields and the id field'

idx

=

field

+

'_index'

is_series

=

idx

in

```
self
.
fields

try
:

func

=

self
.
fields
[
fm
.
get
(
field
,
field
)]
.
sort_keys_for_books
(
get_metadata
,
lang_map
)

except
KeyError
:

if
```

```
field
```

```
==
```

```
'id'
```

```
:
```

```
return
```

```
IDENTITY
```

```
else
```

```
:
```

```
return
```

```
virtual_fields
```

```
[
```

```
fm
```

```
.
```

```
get
```

```
(
```

```
field
```

```
,
```

```
field
```

```
)]
```

```
.
```

```
sort_keys_for_books
```

```
(
```

```
get_metadata
```

```
,
```

```
lang_map
```

```
)
```

```
if
```

```
is_series
:

idx_func

=

self
.
fields
[
idx
]
.
sort_keys_for_books
(
get_metadata
,
lang_map
)

def

skf
(
book_id
):

return

(
func
(
book_id
),
idx_func
(
```

```
book_id
))
```

```
return
```

```
skf
```

```
return
```

```
func
```

```
# Sort only once on any given field
```

```
fields
```

```
=
```

```
uniq  
(  
fields  
,
```

```
operator
```

```
.
```

```
itemgetter  
(  
0  
)
```

```
if
```

```
len  
(  
fields  
)
```



```
==

1
:

keyfunc

=

sort_key_func
(
fields
[
0
][
0
])

reverse

=

not

fields
[
0
][
1
]

try
:

return

sorted
(
ids_to_sort
```

```
,  
  
key  
=  
keyfunc  
,  
  
reverse  
=  
reverse  
)  
  
except  
  
Exception  
  
as  
  
err  
:  
  
print  
(  
    'Failed to sort database on field:'  
,  
  
    fields  
    [  
        0  
    ]  
    [  
        0  
    ],  
  
    'with error:'  
,  
  
    err  
,  
  
    file  
    =
```

```
sys
.
stderr
)

try
:

return

sorted
(
ids_to_sort
,
key
=
type_safe_sort_key_function
(
keyfunc
),
reverse
=
reverse
)

except
Exception

as

err
:

print
(
'Failed to type-safe sort database on field:'
```

```
,  
  
fields  
[  
0  
],  
  
'with error:',  
  
,  
  
err  
,  
  
file  
=  
sys  
.  
stderr  
)  
  
return  
  
sorted  
(  
ids_to_sort  
,  
  
reverse  
=  
reverse  
)  
  
sort_key_funcs  
  
=  
  
tuple  
(  
sort_key_func
```

```
(  
field  
)
```

```
for
```

```
field  
,
```

```
order
```

```
in
```

```
fields  
)
```

```
orders
```

```
=
```

```
tuple  
(  
1
```

```
if
```

```
order
```

```
else
```

```
-  
1
```

```
for
```

```
-  
,
```

```
order
```

```
in
```

```
fields  
)
```

Lazy

=

```
object  
(
```

Lazy load the sort keys for sub-sort fields

class

SortKey

:

__slots__

=

'book_id'

,

'sort_key'

def

__init__

(

self

,

book_id

):

```
self
.  
book_id  
  
=  
  
book_id
```

Calculate only the first sub-sort key since that will always be used

```
self
.  
sort_key  
  
=  
  
[  
key  
(  
book_id  
)
```

```
if
```

```
i
```

```
==
```

```
0
```

```
else
```

```
Lazy
```

```
for
```

```
i
```

```
,
```

```
key
```

in

```
enumerate  
(  
    sort_key_funcs  
)]
```

def

```
compare_to_other  
(  
    self  
,  
  
    other  
):
```

for

```
i  
,  
  
(  
    order  
,  
  
    self_key  
,  
  
    other_key  
)
```

in

```
enumerate  
(  
    zip  
(  
        orders
```



```
,  
  
self  
.  
sort_key  
,  
  
other  
.  
sort_key  
)):  
  
if  
  
self_key  
  
is  
  
Lazy  
:  
  
self_key  
  
=  
  
self  
.  
sort_key  
[  
i  
]  
  
=  
  
sort_key_funcs  
[  
i  
](  
self  
.  
book_id
```

```
)  
  
if  
    other_key  
  
    is  
    Lazy  
    :  
  
    other_key  
  
    =  
    other  
    .  
    sort_key  
    [  
    i  
    ]  
  
    =  
    sort_key_funcs  
    [  
    i  
    ](  
    other  
    .  
    book_id  
    )  
  
    ans  
  
    =  
  
    cmp  
    (  
    self_key
```

```
,  
  
other_key  
)
```

```
if
```

```
ans
```

```
!=
```

```
0
```

```
:
```

```
return
```

```
ans
```

```
*
```

```
order
```

```
return
```

```
0
```

```
def
```

```
__eq__
```

```
(  
self
```

```
,
```

```
other
```

```
):
```

```
return
```

```
self
.  
compare_to_other  
(  
other  
)
```

```
==
```

```
0
```

```
def
```

```
__ne__
```

```
(  
self  
,  
other  
):
```

```
return
```

```
self  
.  
compare_to_other  
(  
other  
)
```

```
!=
```

```
0
```

```
def
```

```
__lt__
```

```
(
self
,
other
):

return

self
.
compare_to_other
(
other
)

<

0


def
__le__
(
self
,
other
):

return

self
.
compare_to_other
(
other
)
```

<=

0

def

__gt__

(
self
,

other
):

return

self

.
compare_to_other
(
other
)

>

0

def

__ge__

(
self
,

other
):

```
return
```

```
self
```

```
.
```

```
compare_to_other
```

```
(
```

```
other
```

```
)
```

```
>=
```

```
0
```

```
return
```

```
sorted
```

```
(
```

```
ids_to_sort
```

```
,
```

```
key
```

```
=
```

```
SortKey
```

```
)
```

```
\[docs\]
```

```
@read_api
```

```
def
```

```
search
```

```
(
```

```
self
```

```
,
```

```
query
```

```
,
```

```
restriction
=
"
,
virtual_fields
=
None
,
book_ids
=
None
):
```

```
'''
```

Search the database for the specified query, returning a set of matched book ids.

:param restriction: A restriction that is ANDed to the specified query. Note that

restrictions are cached, therefore the search for a AND b will be slower than a with restriction b.

:param virtual_fields: Used internally (virtual fields such as on_device to search over).

:param book_ids: If not None, a set of book ids for which books will

be searched instead of searching all books.

```
'''
```

```
return
```

```
self
```

```
._search_api
(
```



```
self
,
query
,
restriction
,
virtual_fields
=
virtual_fields
,
book_ids
=
book_ids
)
```

[\[docs\]](#)

@read_api

def

books_in_virtual_library

(
self

,

vl

,

search_restriction

=

None

,

virtual_fields

=

None

```
):
```

```
' Return the set of books in the specified virtual library '
```

```
vl
```

```
=
```

```
self
```

```
.
```

```
_pref
```

```
(
```

```
'virtual_libraries'
```

```
,
```

```
{})
```

```
.
```

```
get
```

```
(
```

```
vl
```

```
)
```

```
if
```

```
vl
```

```
else
```

```
None
```

```
if
```

```
not
```

```
vl
```

```
and
```

```
not
```

```
search_restriction
:

return

self
.
all_book_ids
()

# We utilize the search restriction cache to speed this up

srch
=

partial
(
self
.
_search
,
virtual_fields
=
virtual_fields
)

if

vl
:

if

search_restriction
:
```

return

frozenset

(
srch

(
"

,

vl
)

&

srch
(
"

,

search_restriction
)

return

frozenset

(
srch

(
"

,

vl
)

return

frozenset

(
srch

(

```
"
,

search_restriction
))

@read_api

def

number_of_books_in_virtual_library
(
self
,

vl
=
None
,

search_restriction
=
None
):

if

not

vl

and

not

search_restriction
:

return
```

```
len
(
self
.
fields
[
'uuid'
]
.
table
.
book_col_map
)
```

return

```
len
(
self
.
books_in_virtual_library
(
vl
,
search_restriction
))
```

[\[docs\]](#)

@api

def

```
get_categories
(
self
```

```
,

sort
=
'name'
,

book_ids
=
None
,

already_fixed
=
None
,

first_letter_sort
=
False
):

' Used internally to implement the Tag Browser '

try
:

with

self
.
safe_read_lock
:

return

get_categories
(
```

```
self
,

sort
=
sort
,

book_ids
=
book_ids
,

first_letter_sort
=
first_letter_sort
)
```

except

InvalidLinkTable

as

err
:

bad_field

=

err
.
field_name

if

bad_field


```
==

already_fixed
:

raise

with

self
.
write_lock
:

self
.
fields
[
bad_field
]
.
table
.
fix_link_table
(
self
.
backend
)

return

self
.
get_categories
(
sort
=
sort
```

```
,  
  
book_ids  
=  
book_ids  
,  
  
already_fixed  
=  
bad_field  
)  
  
@write_api  
  
def  
  
update_last_modified  
(  
self  
,  
  
book_ids  
,  
  
now  
=  
None  
):  
  
if  
  
book_ids  
:  
  
if  
  
now  
  
is
```

None

:

now

=

nowf

()

f

=

self

.

fields

[

'last_modified'

]

f

.

writer

.

set_books

{

book_id

:

now

for

book_id

in

book_ids

},

```
self
    .
    backend
)

if

self
    .
    composites
:

self
    .
    _clear_composite_caches
(
    book_ids
)

self
    .
    _clear_search_caches
(
    book_ids
)

@write_api

def

mark_as_dirty
(
    self
    ,
    book_ids
```

```
);

self
.
_update_last_modified
(
book_ids
)

already_dirtied

=

set
(
self
.
dirtied_cache
)
.
intersection
(
book_ids
)

new_dirtied

=

book_ids

-

already_dirtied

already_dirtied

=
```

```
{
book_id
:
self
.
dirtied_sequence
+
i

for

i
,

book_id

in

enumerate
(
already_dirtied
)}

if

already_dirtied
:

self
.
dirtied_sequence

=

max
(
intervalues
(
already_dirtied
))
```

```
+  
  
1  
  
self  
.  
dirtied_cache  
.  
update  
(  
already_dirtied  
)  
  
if  
  
new_dirtied  
:  
  
self  
.  
backend  
.  
dirty_books  
(  
new_dirtied  
)  
  
new_dirtied  
  
=  
  
{  
book_id  
:  
self  
.  
dirtied_sequence  
+  
i
```

```
for
i
,
book_id

in

enumerate
(
new_dirtied
)}

self
.
dirtied_sequence

=

max
(
itervalues
(
new_dirtied
))

+

1

self
.
dirtied_cache
.
update
(
new_dirtied
)
```



```
@write_api
```

```
def
```

```
commit_dirty_cache
```

```
(  
    self  
):
```

```
if
```

```
self
```

```
.  
dirtied_cache  
:
```

```
self
```

```
.  
backend
```

```
.  
dirty_books  
(  
    self  
.  
dirtied_cache  
)
```

```
@write_api
```

```
def
```

```
check_dirtied_annotations
```

```
(  
    self  
):
```

```
if
not
self
.
backend
.
dirty_books_with_dirtied_annotations
():
```

```
return
```

```
book_ids
```

```
=
```

```
set
(
self
.
backend
.
dirtied_books
())
```

```
new_dirtied
```

```
=
```

```
book_ids
```

```
-
```

```
set
(
self
```

```
.
```

```
dirtied_cache
)

if

new_dirtied
:

new_dirtied

=

{
book_id
:
self
.
dirtied_sequence
+
i

for

i
,

book_id

in

enumerate
(
new_dirtied
)}

self
.
dirtied_sequence

=
```

```
max
(
intervalvalues
(
new_dirtied
))
```

```
+
```

```
1
```

```
self
.
dirtied_cache
.
update
(
new_dirtied
)
```

[\[docs\]](#)

```
@write_api
```

```
def
```

```
set_field
(
self
,
name
,
book_id_to_val_map
,
```

```
allow_case_change
```

```
=
```

```
True
```

```
,
```

```
do_path_update
```

```
=
```

```
True
```

```
):
```

```
'''
```

Set the values of the field specified by ``name``. Returns the set of all book ids that were affected by the change.

:param book_id_to_val_map: Mapping of book_ids to values that should be applied.

:param allow_case_change: If True, the case of many-one or many-many fields will be changed.

For example, if a book has the tag ``tag1`` and you set the tag for another book to ``Tag1``

then the both books will have the tag ``Tag1`` if allow_case_change is True, otherwise they will

both have the tag ``tag1``.

:param do_path_update: Used internally, you should never change it.

```
'''
```

```
f
```

```
=
```

```
self
```

```
.
```

```
fields
```

```
[
```

```
name
]

is_series

=

f
.
metadata
[
'datatype'
]

==

'series'

update_path

=

name

in

{
'title'
,
'authors'
}

if

update_path

and

iswindows
```

```
:  
  
paths  
  
=  
  
(  
x  
  
for  
  
x  
  
in  
  
(  
self  
.  
_field_for  
(  
'path'  
,  
book_id  
)  
for  
book_id  
in  
book_id_to_val_map  
)  
if  
x  
)  
  
self
```

```
.  
backend  
.  
windows_check_if_files_in_use  
(  
paths  
)
```

```
if
```

```
is_series  
:
```

```
bimap
```

```
,
```

```
simap
```

```
=
```

```
{},
```

```
{}
```

```
sfield
```

```
=
```

```
self
```

```
.  
fields
```

```
[  
name
```

```
+
```

```
'_index'  
]
```



```
for
k
,
v
in
iteritems
(
book_id_to_val_map
):

if
isinstance
(
v
,
string_or_bytes
):

v
,
sid
=
get_series_values
(
v
)

else
:
```

v

=

sid

=

None

if

sid

is

None

and

name

.

startswith

(

'#'

):

sid

=

self

.

_fast_field_for

(

sfield

,

k

)

sid

=

1.0

if

sid

is

None

else

sid

The value to be set the db link table

bimap

[

k

]

=

v

if

sid

is

not

None

:

```
simap
[
k
]

=

sid

book_id_to_val_map

=

bimap


dirtied

=

f
.
writer
.
set_books
(

book_id_to_val_map
,

self
.
backend
,

allow_case_change
=
allow_case_change
```

```
)  
  
if  
is_series  
and  
simap  
:  
  
sf  
=  
self  
.  
fields  
[  
f  
.  
name  
+  
'_index'  
]  
  
dirtied  
|=  
sf  
.  
writer  
.  
set_books  
(  
simap  
,
```

```
self
.
backend
,

allow_case_change
=
False
)
```

```
if
```

```
dirtyed
:
```

```
if
```

```
update_path
```

```
and
```

```
do_path_update
:
```

```
self
```

```
.
_update_path
(
dirtyed
,

mark_as_dirtyed
=
False
)
```

```
self
```

```
.
```

```
_mark_as_dirty
(
    dirtied
)

self
.
event_dispatcher
(
    EventType
.
    metadata_changed
,
    name
,
    dirtied
)

return

dirtied

@write_api

def

update_path
(
    self
,
    book_ids
,
    mark_as_dirtied
=
    True
```

```
):  
  
for  
    book_id  
  
    in  
        book_ids  
        :  
  
        title  
  
        =  
  
        self  
        .  
        _field_for  
        (  
            'title'  
        ,  
  
        book_id  
        ,  
  
        default_value  
        =  
        _  
        (  
            'Unknown'  
        ))  
  
try  
:  
  
author  
  
=
```



```
self
    .
    _field_for
    (
        'authors'
    ,

    book_id
    ,

    default_value
    =
    (
        -
        (
            'Unknown'
        ),))[
    0
]
```

except

```
IndexError
:
```

```
author
```

```
=
```

```
-
(
    'Unknown'
)
```

```
self
```

```
.
backend
```

```
.
update_path
(
```

```
book_id
',

title
',

author
',

self
.
fields
[
'path'
],

self
.
fields
[
'formats'
])

self
.
format_metadata_cache
.
pop
(
book_id
,

None
)

if

mark_as_dirtied
:
```

```
self
.
_mark_as_dirty
(
book_ids
)

@read_api

def

get_a_dirtied_book
(
self
):

if

self
.
dirtied_cache
:

return

random
.
choice
(
tuple
(
self
.
dirtied_cache
))

return
```

None

@read_api

def

get_metadata_for_dump

(

self

,

book_id

):

mi

=

None

get the current sequence number for this book to pass back to the

backup thread. This will avoid double calls in the case where the

thread has not done the work between the put and the get_metadata

sequence

=

self

.

dirtied_cache

.

```
get
(
book_id
,
None
)

if
sequence
is
not
None
:

try
:

# While a book is being created, the path is empty. Don't bother to

# try to write the opf, because it will go to the wrong folder.

if
self
.
_field_for
(
'path'
,
book_id
):
```

```
mi
=
self
.
_get_metadata
(
book_id
)

# Always set cover to cover.jpg. Even if cover doesn't exist,

# no harm done. This way no need to call dirtied when

# cover is set/removed

mi
.
cover
=
'cover.jpg'

mi
.
all_annotations
=
self
.
_all_annotations_for_book
(
book_id
)
```

```
except
```

```
:
```

```
# This almost certainly means that the book has been deleted while
```

```
# the backup operation sat in the queue.
```

```
import
```

```
traceback
```

```
traceback
```

```
.
```

```
print_exc
```

```
()
```

```
return
```

```
mi
```

```
,
```

```
sequence
```

```
@write_api
```

```
def
```

```
clear_dirtied
```

```
(
```

```
self
```

```
,
```

```
book_id
```

```
,  
  
sequence  
):  
  
# Clear the dirtied indicator for the books. This is used when fetching  
  
# metadata, creating an OPF, and writing a file are separated into steps.  
  
# The last step is clearing the indicator  
  
dc_sequence  
  
=  
  
self  
.  
dirtied_cache  
.  
get  
(  
    book_id  
,  
    None  
)  
  
if  
  
dc_sequence  
  
is  
  
None  
  
or  
  
sequence
```


is

None

or

dc_sequence

==

sequence

:

self

.

backend

.

mark_book_as_clean

(

book_id

)

self

.

dirtied_cache

.

pop

(

book_id

,

None

)

@write_api

def

```
write_backup
```

```
(
```

```
self
```

```
,
```

```
book_id
```

```
,
```

```
raw
```

```
):
```

```
try
```

```
:
```

```
path
```

```
=
```

```
self
```

```
.
```

```
_field_for
```

```
(
```

```
'path'
```

```
,
```

```
book_id
```

```
)
```

```
.
```

```
replace
```

```
(
```

```
'/'
```

```
,
```

```
os
```

```
.
```

```
sep
```

```
)
```

```
except
```

```
:  
  
return  
  
self  
.  
backend  
.  
write_backup  
(  
path  
,  
raw  
)  
  
@read_api  
  
def  
dirty_queue_length  
(  
self  
):  
  
return  
  
len  
(  
self  
.  
dirty_cache  
)
```

[\[docs\]](#)

@read_api

def

read_backup

(

self

,

book_id

):

""" Return the OPF metadata backup for the book as a bytestring or None

if no such backup exists. """

try

:

path

=

self

.

_field_for

(

'path'

,

book_id

)

.

replace

(

'/'

,

```
os
.
sep
)

except
:

return

try
:

return

self
.
backend
.
read_backup
(
path
)

except

OSError
:

return

None
```

```
@write_api
```

```
def
```

```
    dump_metadata
```

```
    (
```

```
        self
```

```
    ,
```

```
        book_ids
```

```
    =
```

```
        None
```

```
    ,
```

```
        remove_from_dirtied
```

```
    =
```

```
        True
```

```
    ,
```

```
        callback
```

```
    =
```

```
        None
```

```
    ):
```

```
        # Write metadata for each record to an individual OPF file. If callback
```

```
        # is not None, it is called once at the start with the number of book_ids
```

```
        # being processed. And once for every book_id, with arguments (book_id,
```

```
        # mi, ok).
```

```
    if
```

```
        book_ids
```

```
is
None
:

book_ids

=

set
(
self
.
dirtied_cache
)

if
callback

is
not
None
:

callback
(
len
(
book_ids
),

True
,

False
)
```

for

book_id

in

book_ids

:

if

self

.

_field_for

(

'path'

,

book_id

)

is

None

:

if

callback

is

not

None

:


```
callback
(
    book_id
    ,
    None
    ,
    False
)

continue

mi
,

sequence

=

self
.
_get_metadata_for_dump
(
    book_id
)

if

mi

is

None
:

if

callback
```

is

not

None

:

callback

(

book_id

,

mi

,

False

)

continue

try

:

raw

=

metadata_to_opf

(

mi

)

self

.

_write_backup

(

book_id

```
,  
  
raw  
)  
  
if  
  
remove_from_dirtied  
:  
  
self  
.  
_clear_dirtied  
(  
book_id  
,  
  
sequence  
)  
  
except  
:  
  
pass  
  
if  
  
callback  
  
is  
  
not  
  
None  
:  
  
callback
```

```
(  
    book_id  
,  
    mi  
,  
    True  
)
```

[\[docs\]](#)

@write_api

def

```
    set_cover  
(  
        self  
,  
        book_id_data_map  
    ):
```

```
    """ Set the cover for this book. The data can be either a QImage,  
        QPixmap, file object or bytestring. It can also be None, in which  
        case any existing cover is removed. """
```

for

```
    book_id  
,  
    data
```

in

```
iteritems  
(  
    book_id_data_map  
):
```

try

:

path

=

self

.

_field_for

(

'path'

,

book_id

)

.

replace

(

'/'

,

os

.

sep

)

except

AttributeError

:

```
self
.  
_update_path  
((  
book_id  
,))
```

```
path
```

```
=
```

```
self  
.  
_field_for  
(  
'path'  
,  
book_id  
)  
.  
replace  
(  
'/'  
,  
os  
.  
sep  
)
```

```
self  
.  
backend  
.  
set_cover  
(  
book_id  
,
```

```
path
,
data
)

for

cc

in

self
.
cover_caches
:

cc
.
invalidate
(
book_id_data_map
)

return

self
.
_set_field
(
'cover'
,
{

book_id
:(
0
```

```
if
data
is
None
else
1
)
for
book_id
,
data
in
iteritems
(
book_id_data_map
))

@write_api

def
add_cover_cache
(
self
,
cover_cache
):
```



```
if
    not
        callable
        (
            cover_cache
            .
            invalidate
        ):

    raise
        ValueError
        (
            'Cover caches must have an invalidate method'
        )

    self
    .
    cover_caches
    .
    add
    (
        cover_cache
    )

@write_api

def
    remove_cover_cache
    (
        self
    ,
        cover_cache
    ):
    pass
```

```
self
.  
cover_caches  
.  
discard  
(  
cover_cache  
)
```

[\[docs\]](#)

@write_api

```
def  
  
set_metadata  
(  
self  
,  
  
book_id  
,  
  
mi  
,  
  
ignore_errors  
=  
False  
,  
  
force_changes  
=  
False  
,
```

```

set_title
=
True
,

set_authors
=
True
,

allow_case_change
=
False
):

'''
    Set metadata for the book `id` from the `Metadata` object `mi`

    Setting force_changes=True will force set_metadata to update fields even
    if mi contains empty values. In this case, 'None' is distinguished from
    'empty'. If mi.XXX is None, the XXX is not replaced, otherwise it is.

    The tags, identifiers, and cover attributes are special cases. Tags and
    identifiers cannot be set to None so they will always be replaced if
    force_changes is true. You must ensure that mi contains the values you
    want the book to have. Covers are always changed if a new cover is
    provided, but are never deleted. Also note that force_changes has no
    effect on setting title or authors.
'''

```

dirtied

```
set
()
```

```
# Handle code passing in an OPF object instead of a Metadata object
```

$$=$$

```
to_book_metadata
```

```
(
AttributeError
,
```

pass

```
set_field
```

```
name
,

val
):

dirtied
.
update
(
self
.
_set_field
(
name
,

{
book_id
:
val
},

do_path_update
=
False
,

allow_case_change
=
allow_case_change
))

path_changed

=

False
```

```
if
    set_title

    and

    mi
    .
    title
    :

    path_changed

    =

    True

    set_field
    (
    'title'
    ,
    mi
    .
    title
    )

    authors_changed

    =

    False

    if

    set_authors
    :
```

```
path_changed
```

```
=
```

```
True
```

```
if
```

```
not
```

```
mi
```

```
.  
authors  
:
```

```
mi
```

```
.  
authors
```

```
=
```

```
[
```

```
_  
(  
'Unknown'  
)]
```

```
authors
```

```
=
```

```
[]
```

```
for
```

```
a
```

```
in
```

```
mi
.
authors
:

authors

+=

string_to_authors
(
a
)

set_field
(
'authors'
,
authors
)

authors_changed

=

True

if

path_changed
:

self

.
_update_path
({
```



```
book_id
})

def

protected_set_field
(
name
,

val
):

try
:

set_field
(
name
,

val
)

except
:

if

ignore_errors
:

traceback
.
print_exc
()
```

```
else
```

```
:
```

```
raise
```

```
# force_changes has no effect on cover manipulation
```

```
try
```

```
:
```

```
cdata
```

```
=
```

```
mi
```

```
.
```

```
cover_data
```

```
[
```

```
1
```

```
]
```

```
if
```

```
cdata
```

```
is
```

```
None
```

```
and
```

```
isinstance
```

```
(
```

```
mi
```

```
.
```

```
cover
,
string_or_bytes
)
```

and

```
mi
.
cover
```

and

```
os
.
access
(
mi
.
cover
,
os
.
R_OK
):
```

with

```
lopen
(
mi
.
cover
,
'rb'
)
```

as

```
f
:

cdata

=

f
.
read
()

or

None

if

cdata

is

not

None
:

self
.
_set_cover
({
book_id
:

cdata
})

except
:
```

if

ignore_errors

:

traceback

.

print_exc

()

else

:

raise

try

:

with

self

.

backend

.

conn

:

Speed up set_metadata by not operating in autocommit mode

for

field

in

```
(  
    'rating'  
,  
  
    'series_index'  
,  
  
    'timestamp'  
):
```

```
val
```

```
=
```

```
getattr  
(  
    mi  
,  
  
    field  
)
```

```
if
```

```
val
```

```
is
```

```
not
```

```
None  
:
```

```
protected_set_field  
(  
    field  
,
```

```
val
```

```
)  
  
val  
  
=  
  
mi  
.  
get  
(  
  'author_sort'  
,  
  None  
)
```

```
if  
  
authors_changed  
  
and  
  
(  
  not  
  val  
  or  
  mi  
  .  
  is_null  
  (  
    'author_sort'  
  )):
```

```
val  
  
=
```

```
self
.  
_author_sort_from_authors  
(  
mi  
.  
authors  
)
```

```
if
```

```
authors_changed
```

```
or
```

```
(  
force_changes
```

```
and
```

```
val
```

```
is
```

```
not
```

```
None
```

```
)
```

```
or
```

```
not
```

```
mi
```

```
.  
is_null
```

```
(  
'author_sort'  
):
```



```
protected_set_field
```

```
(
```

```
'author_sort'
```

```
,
```

```
val
```

```
)
```

```
for
```

```
field
```

```
in
```

```
(
```

```
'publisher'
```

```
,
```

```
'series'
```

```
,
```

```
'tags'
```

```
,
```

```
'comments'
```

```
,
```

```
'languages'
```

```
,
```

```
'pubdate'
```

```
):
```

```
val
```

```
=
```

```
mi
```

```
.
```

```
get
(
field
,
None
)

if

(
force_changes

and

val

is

not

None
)

or

not

mi
.
is_null
(
field
):

protected_set_field
(
field
,
val
```

```
)

val
=
mi
.
get
(
'title_sort'
,
None
)

if
(
force_changes
and
val
is
not
None
)
or
not
mi
.
is_null
(
```

```
'title_sort'
):

protected_set_field
(
'sort'
,
val
)

# identifiers will always be replaced if force_changes is True

mi_idents

=

mi
.
get_identifiers
()

if

force_changes
:

protected_set_field
(
'identifiers'
,
mi_idents
)

elif
```

```
mi_idents
:

identifiers

=

self
.
_field_for
(
'identifiers'
,
book_id
,
default_value
=
{})
```

```
for
```

```
key
,
```

```
val
```

```
in
```

```
iteritems
(
mi_idents
):
```

```
if
```

```
val
```

and

val

.

strip

():

Don't delete an existing identifier

identifiers

[

icu_lower

(

key

)]

=

val

protected_set_field

(

'identifiers'

,

identifiers

)

user_mi

=

mi

.

get_all_user_metadata

(

make_copy

=

False

```
)  
  
fm  
  
=  
  
self  
.  
field_metadata
```

```
for
```

```
key
```

```
in
```

```
user_mi  
:
```

```
if
```

```
(  
key
```

```
in
```

```
fm
```

```
and
```

```
user_mi  
[  
key  
][  
'datatype'  
]
```

```
==
```

```
fm
```

```
[
key
][
'datatype'
]

and

(

user_mi
[
key
][
'datatype'
]

!=

'text'

or

(

user_mi
[
key
][
'is_multiple'
]

==

fm
[
key
][
'is_multiple'
]]):
```



```
val
=
mi
.
get
(
key
,
None
)

if
force_changes
or
val
is
not
None
:

protected_set_field
(
key
,
val
)

idx
```

```
=  
  
key  
  
+  
  
'_index'  
  
if  
  
idx  
  
in  
  
self  
.  
fields  
:  
  
extra  
  
=  
  
mi  
.  
get_extra  
(  
key  
)  
  
if  
  
extra  
  
is  
  
not  
  
None
```

or

force_changes

:

protected_set_field

(

idx

,

extra

)

except

:

sqlite will rollback the entire transaction, thanks to the with

statement, so we have to re-read everything from the db to ensure

the db and Cache are in sync

self

.

_reload_from_db

()

raise

return

dirtied

```
def
    _do_add_format
    (
        self
        ,

        book_id
        ,

        fmt
        ,

        stream
        ,

        name
        =
        None
        ,

        mtime
        =
        None
    ):

    path

    =

    self
    .
    _field_for
    (
        'path'
        ,

        book_id
    )

    if
```

path

is

None

:

Theoretically, this should never happen, but apparently it

does: <https://www.mobileread.com/forums/showthread.php?t=233353>

self

.

_update_path

{

book_id

},

mark_as_dirtied

=

False

)

path

=

self

.

_field_for

(

'path'

,

book_id

)

```
path
=
path
.
replace
(
    '/',
    os
    .
    sep
)

title
=
self
.
    _field_for
    (
        'title'
    ,
        book_id
    ,
        default_value
        =
        _
        (
            'Unknown'
        ))

try
:
```

```
author
=
self
.
_field_for
(
'authors'
,
book_id
,
default_value
=
(
-
(
'Unknown'
),))[
0
]
```

```
except
IndexError
:
```

```
author
=
-
(
'Unknown'
)
```

```
size
,

fname

=

self
.
backend
.
add_format
(
book_id
,

fmt
,

stream
,

title
,

author
,

path
,

name
,

mtime
=
mtime
)

return

size
```



```
,  
  
fname  
  
\[docs\]  
  
@api  
  
def  
  
add_format  
(  
    self  
,  
  
    book_id  
,  
  
    fmt  
,  
  
    stream_or_path  
,  
  
    replace  
    =  
    True  
,  
  
    run_hooks  
    =  
    True  
,  
  
    dbapi  
    =  
    None  
):
```

```
'''
```

```
    Add a format to the specified book. Return True if the format was added successfully.
```

```
    :param replace: If True replace existing format, otherwise if the format already exists,  
return False.
```

```
    :param run_hooks: If True, file type plugins are run on the format before and after being  
added.
```

```
    :param dbapi: Internal use only.
```

```
'''
```

```
needs_close
```

```
=
```

```
False
```

```
if
```

```
run_hooks
```

```
:
```

```
    # Run import plugins, the write lock is not held to cater for
```

```
    # broken plugins that might spin the event loop by popping up a
```

```
    # message in the GUI during the processing.
```

```
npath
```

```
=
```

```
run_import_plugins
```

```
(
```

```
stream_or_path
```

```
,
```

```
fmt
```

```
)
```

```
fmt
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
splitext
```

```
(
```

```
npath
```

```
)[
```

```
-
```

```
1
```

```
]
```

```
.
```

```
lower
```

```
()
```

```
.
```

```
replace
```

```
(
```

```
'.'
```

```
,
```

```
''
```

```
)
```

```
.
```

```
upper
```

```
()
```

```
stream_or_path
```

```
=
```

```
    lopen
    (
    npath
    ,
    'rb'
    )

    needs_close

    =

    True

    fmt

    =

    check_ebook_format
    (
    stream_or_path
    ,
    fmt
    )

    with

    self
    .
    write_lock
    :

    if

    not
```

```
self
.
_has_id
(
book_id
):

raise

NoSuchBook
(
book_id
)

fmt

=

(
fmt

or

"
)
.
upper
()

self
.
format_metadata_cache
[
book_id
]
.
pop
(
fmt
,
```

None

)

try

:

name

=

self

.

fields

[

'formats'

]

.

format_fname

(

book_id

,

fmt

)

except

Exception

:

name

=

None

if

name

and

not

replace

:

return

False

if

hasattr

(

stream_or_path

,

'read'

):

stream

=

stream_or_path

else

:

stream

=

```
lopen  
(  
    stream_or_path  
,  
    'rb'  
)
```

```
needs_close
```

```
=
```

```
True
```

```
try  
:
```

```
    stream
```

```
=
```

```
    stream_or_path
```

```
if
```

```
    hasattr  
    (  
        stream_or_path  
,  
        'read'  
    )
```

```
else
```

```
    lopen  
    (  
        stream_or_path  
,
```



```
'rb'
)

size
,

fname

=

self
.
    _do_add_format
(
    book_id
,

    fmt
,

    stream
,

    name
)

finally
:

if

needs_close
:

stream
.
close
()
```

```
del
```

```
stream
```

```
max_size
```

```
=
```

```
self
```

```
.
```

```
fields
```

```
[
```

```
'formats'
```

```
]
```

```
.
```

```
table
```

```
.
```

```
update_fmt
```

```
(
```

```
book_id
```

```
,
```

```
fmt
```

```
,
```

```
fname
```

```
,
```

```
size
```

```
,
```

```
self
```

```
.
```

```
backend
```

```
)
```

```
self
```

```
.
```

```
fields
[
'size'
]
.
table
.
update_sizes
({
book_id
:

max_size
})

self
.
_update_last_modified
((
book_id
,))

self
.
event_dispatcher
(
EventType
.
format_added
,

book_id
,

fmt
)

if
```

```
run_hooks
:

# Run post import plugins, the write lock is released so the plugin

# can call api without a locking violation.

run_plugins_on_postimport
(
dbapi

or

self
,

book_id
,

fmt
)

stream_or_path
.
close
()

return

True

\[docs\]

@write_api
```

```

def
remove_formats
(
self
,

formats_map
,

db_only
=
False
):

'''

    Remove the specified formats from the specified books.

    :param formats_map: A mapping of book_id to a list of formats to be removed from the
    book.

    :param db_only: If True, only remove the record for the format from the db, do not delete
    the actual format file from the filesystem.

    '''

table

=

self
.
fields
[
'formats'
]
.
table

```

```
formats_map
```

```
=
```

```
{  
    book_id  
    :  
    frozenset  
    ((  
        f
```

```
    or
```

```
    "
```

```
)
```

```
    .  
    upper  
    ()
```

```
for
```

```
f
```

```
in
```

```
fmts  
)
```

```
for
```

```
book_id  
,
```

```
fmts
```

```
in
```

```
iteritems  
(  
    formats_map  
)}
```

for

book_id

,

fmts

in

iteritems

(

formats_map

):

for

fmt

in

fmts

:

self

.

format_metadata_cache

[

book_id

]

.

pop

(

fmt

,

None

)

```
if
not
db_only
:

removes

=

defaultdict
(
set
)

for
book_id
,
fmts

in

iteritems
(
formats_map
):

try
:

path

=

self
```



```
.  
_field_for  
(
```

```
'path'
```

```
,
```

```
book_id
```

```
)
```

```
.
```

```
replace
```

```
(
```

```
'/'
```

```
,
```

```
os
```

```
.
```

```
sep
```

```
)
```

```
except
```

```
:
```

```
continue
```

```
for
```

```
fmt
```

```
in
```

```
fmts
```

```
:
```

```
try
```

```
:
```

```
name
```

```
=  
  
self  
.fields  
[  
    'formats'  
]  
.format_fname  
(  
    book_id  
,  
  
    fmt  
)  
  
except  
:  
  
    continue  
  
if  
  
    name  
  
    and  
  
    path  
:  
  
    removes  
    [  
        book_id  
    ]  
    .add  
    ((  
        fmt
```

```
,  
  
name  
,  
  
path  
)  
  
if  
  
removes  
:  
  
self  
.  
backend  
.  
remove_formats  
(  
removes  
)  
  
size_map  
  
=  
  
table  
.  
remove_formats  
(  
formats_map  
,  
self  
.  
backend  
)
```

```
self
.  
fields  
[  
    'size'  
]  
.  
table  
.  
update_sizes  
(  
    size_map  
)
```

```
self  
.  
_update_last_modified  
(  
    tuple  
(  
        formats_map  
    ))
```

```
self  
.  
event_dispatcher  
(  
    EventType  
.  
formats_removed  
,  
  
    formats_map  
)
```

[\[docs\]](#)

@read_api

```

def
get_next_series_num_for
(
self
,
series
,
field
=
'series'
,
current_indices
=
False
):
'''
    Return the next series index for the specified series, taking into account the various
    preferences that
    control next series number generation.
    :param field: The series-like field (defaults to the builtin series column)
    :param current_indices: If True, returns a mapping of book_id to current series_index value
    instead.
    '''

books
=
()

```

```
sf
=
self
.
fields
[
field
]

if
series
:

q
=

icu_lower
(
series
)

for
val
,
book_ids
in
sf
.
iter_searchable_values
(
self
```

```
._get_proxy_metadata
```

```
,
```

```
frozenset
```

```
(
```

```
self
```

```
.
```

```
_all_book_ids
```

```
())):
```

```
if
```

```
q
```

```
==
```

```
icu_lower
```

```
(
```

```
val
```

```
):
```

```
books
```

```
=
```

```
book_ids
```

```
break
```

```
idf
```

```
=
```

```
sf
```

```
.
```

```
index_field
```

```
index_map

=

{
book_id
:
self
.
_fast_field_for
(
idf
,
book_id
,
default_value
=
1.0
)
for
book_id
in
books
}

if
current_indices
:

return
index_map
```



```
series_indices

=

sorted
(
intervalvalues
(
index_map
))

return

_get_next_series_num_for_list
(
tuple
(
series_indices
),

unwrap
=
False
)
```

[\[docs\]](#)

@read_api

```
def

author_sort_from_authors
(
self
,

authors
,

key_func
```

```
=
icu_lower
):

"""Given a list of authors, return the author_sort string for the authors,

    preferring the author sort associated with the author over the computed

    string. """
```

```
table
```

```
=
```

```
self
```

```
.
```

```
fields
```

```
[
```

```
'authors'
```

```
]
```

```
.
```

```
table
```

```
result
```

```
=
```

```
[]
```

```
rmap
```

```
=
```

```
{
```

```
key_func
```

```
(
```

```
v
```

```
):
```

```
k
```

```
for
```

```
k
```

```
,
```

```
v
```

```
in
```

```
iteritems
```

```
(
```

```
table
```

```
.
```

```
id_map
```

```
)}
```

```
for
```

```
aut
```

```
in
```

```
authors
```

```
:
```

```
aid
```

```
=
```

```
rmap
```

```
.
```

```
get
```

```
(
```

```
key_func
```

```
(
```

```
aut
```

```
),
```

```
None
```

```
)
```

```
result
.  
append  
(  
  author_to_author_sort  
(  
    aut  
  )  
)
```

```
if
```

```
aid
```

```
is
```

```
None
```

```
else
```

```
table
```

```
.  
asort_map  
[  
  aid  
)
```

```
return
```

```
' & '
```

```
.  
join  
(  
  _f
```

```
for
```

```
_f
```

```
in
```

```
result
```

```
if
```

```
    _f  
)
```

```
\[docs\]
```

```
@read_api
```

```
def
```

```
    data_for_has_book  
    (  
    self  
    ):
```

```
    """ Return data suitable for use in :meth:`has_book`. This can be used for an  
        implementation of :meth:`has_book` in a worker process without access to the  
        db. """
```

```
try  
:
```

```
return
```

```
{  
icu_lower  
(  
title  
)
```

```
for
```

```
title
```

in

intervalues

(
self

.
fields

[
'title'
]

.
table

.
book_col_map
)}

except

TypeError
:

Some non-unicode titles in the db

return

{
icu_lower
(
as_unicode
(
title
))
}

for

title

in

```
intervalvalues
(
self
.
fields
[
'title'
]
.
table
.
book_col_map
)}
```

[\[docs\]](#)

@read_api

def

has_book

```
(
self
,
```

```
mi
):
```

""" Return True iff the database contains an entry with the same title

as the passed in Metadata object. The comparison is case-insensitive.

See also :meth:`data\_for\_has\_book`. """

title

=

mi

```
.
title

if

title
:

if

isbytestring
(
title
):

title

=

title
.
decode
(
preferred_encoding
,
'replace'
)

q

=

icu_lower
(
title
)
.
strip
```



```
()

for

title

in

intervalues
(
self
.
fields
[
'title'
]
.
table
.
book_col_map
):

if

q

==

icu_lower
(
title
):

return

True

return
```

False

[\[docs\]](#)

@read_api

def

has_id

(
self
,

book_id
):

' Return True iff the specified book_id exists in the db ''

return

book_id

in

self

.
fields

[
'title'
]

.
table

.
book_col_map

@write_api

```
def
create_book_entry
(
self
,

mi
,

cover
=
None
,

add_duplicates
=
True
,

force_id
=
None
,

apply_import_tags
=
True
,

preserve_uuid
=
False
):

if

mi
.
tags
:
```

```
mi
.
tags

=

list
(
mi
.
tags
)

if

apply_import_tags
:

_add_newbook_tag
(
mi
)

_add_default_custom_column_values
(
mi
,

self
.
field_metadata
)

if

not

add_duplicates
```

and

self

.

_has_book

(

mi

):

return

series_index

=

(

self

.

_get_next_series_num_for

(

mi

.

series

)

if

mi

.

series_index

is

None

else

mi

.

series_index

```
)

try
:

series_index

=

float
(
series_index
)

except

Exception
:

try
:

series_index

=

float
(
self
.
_get_next_series_num_for
(
mi
.
series
))
```

```
except
Exception
:

series_index
```

```
=

1.0
```

```
if
```

```
not
```

```
mi
.
authors
:
```

```
mi
.
authors
```

```
=
```

```
(
-
(
'Unknown'
),)
```

```
aus
```

```
=
```

```
mi
.
author_sort
```

```
if
not
mi
.
is_null
(
'author_sort'
)

else
self
.
 _author_sort_from_authors
(
mi
.
authors
)

mi
.
title
=
mi
.
title

or

(
'Unknown'
)

if
```



```
isbytestring
```

```
(
```

```
  aus
```

```
):
```

```
  aus
```

```
  =
```

```
  aus
```

```
  .
```

```
  decode
```

```
  (
```

```
    preferred_encoding
```

```
  ,
```

```
    'replace'
```

```
  )
```

```
if
```

```
isbytestring
```

```
(
```

```
  mi
```

```
  .
```

```
  title
```

```
):
```

```
  mi
```

```
  .
```

```
  title
```

```
  =
```

```
  mi
```

```
  .
```

```
  title
```

```
  .
```

```
  decode
```

```
(
preferred_encoding
,
'replace'
)

if
force_id

is
None
:

self
.
backend
.
execute
(
'INSERT INTO books(title, series_index, author_sort) VALUES (?, ?, ?)'
,

(
mi
.
title
,
series_index
,
aus
))

else
:
```

```
self
.
backend
.
execute
(
'INSERT INTO books(id, title, series_index, author_sort) VALUES (?, ?, ?, ?)'
,

(
force_id
,

mi
.
title
,

series_index
,

aus
))

book_id

=

self
.
backend
.
last_insert_rowid
()

self
.
event_dispatcher
```

```
(
    EventType
    .
    book_created
    ,
    book_id
)
```

```
mi
.
timestamp
```

```
=
```

```
utcnow
()
```

```
if
```

```
mi
.
timestamp
```

```
is
```

```
None
```

```
else
```

```
mi
.
timestamp
```

```
mi
.
pubdate
```

```
=
```

UNDEFINED_DATE

if

mi

.

pubdate

is

None

else

mi

.

pubdate

if

cover

is

not

None

:

mi

.

cover

,

mi

.

cover_data

=

None

```
,  
  
(  
    None  
,  
  
    cover  
)  
  
self  
.   
_set_metadata  
(  
    book_id  
,  
  
    mi  
,  
  
    ignore_errors  
    =  
    True  
)  
  
if  
    preserve_uuid  
  
    and  
  
    mi  
    .  
    uuid  
    :  
  
    self  
    .  
    _set_field  
    (  
        'uuid'
```

```
,  
  
{  
  book_id  
  :  
  mi  
  .  
  uuid  
})  
  
# Update the caches for fields from the books table
```

```
self  
.  
fields  
[  
  'size'  
]  
.  
table  
.  
book_col_map  
[  
  book_id  
]  
=  
  
0  
  
row  
  
=  
  
next  
(  
  self  
  .  
  backend  
  .
```

```
execute
(
'SELECT sort, series_index, author_sort, uuid, has_cover FROM books WHERE id=?'
,

(
book_id
,)))

for

field
,

val

in

zip
((
'sort'
,

'series_index'
,

'author_sort'
,

'uuid'
,

'cover'
),

row
):

if

field
```



```
==
```

```
'cover'
```

```
:
```

```
val
```

```
=
```

```
bool
```

```
(
```

```
val
```

```
)
```

```
elif
```

```
field
```

```
==
```

```
'uuid'
```

```
:
```

```
self
```

```
.
```

```
fields
```

```
[
```

```
field
```

```
]
```

```
.
```

```
table
```

```
.
```

```
uuid_to_id_map
```

```
[
```

```
val
```

```
]
```

```
=
```

```
book_id

self
.
fields
[
field
]
.
table
.
book_col_map
[
book_id
]

=

val
```

```
return
```

```
book_id
```

```
\[docs\]
```

```
@api
```

```
def
```

```
add_books
```

```
(
self
```

```
,
```

```
books
```

```
,
add_duplicates
=
True
,
apply_import_tags
=
True
,
preserve_uuid
=
False
,
run_hooks
=
True
,
dbapi
=
None
):
```

```
'''
```

Add the specified books to the library. Books should be an iterable of 2-tuples, each 2-tuple of the form `(mi, format_map)` where `mi` is a Metadata object and `format_map` is a dictionary of the form `{fmt: path_or_stream}`, for example: `{'EPUB': '/path/to/file.epub}'`.

Returns a pair of lists: `ids, duplicates`. `ids` contains the book ids for all newly created books in the

database. `duplicates` contains the `(mi, format_map)` for all books that already

exist in the database

as per the simple duplicate detection heuristic used by :meth:`has\_book`.

'''

duplicates

,

ids

=

[],

[]

fmt_map

=

{}

for

mi

,

format_map

in

books

:

book_id

=

```
self
.  
create_book_entry  
(  
mi  
,  
  
add_duplicates  
=  
add_duplicates  
,  
  
apply_import_tags  
=  
apply_import_tags  
,  
  
preserve_uuid  
=  
preserve_uuid  
)
```

if

book_id

is

None

:

duplicates

```
.  
append  
((  
mi  
,  
  
format_map  
))
```

```
else
```

```
:
```

```
ids
```

```
.
```

```
append
```

```
(
```

```
book_id
```

```
)
```

```
for
```

```
fmt
```

```
,
```

```
stream_or_path
```

```
in
```

```
iteritems
```

```
(
```

```
format_map
```

```
):
```

```
if
```

```
self
```

```
.
```

```
add_format
```

```
(
```

```
book_id
```

```
,
```

```
fmt
```

```
,
```

```
stream_or_path
```

```
,
```

```
dbapi
=
dbapi
,

run_hooks
=
run_hooks
):

fmt_map
[
fmt
.
lower
()]

=

getattr
(
stream_or_path
,
'<name>'
,
stream_or_path
)

or

'<stream>'

run_plugins_on_postadd
(
dbapi

or

self
```

```
,  
  
book_id  
,  
  
fmt_map  
)  
  
return  
  
ids  
,  
  
duplicates  
  
\[docs\]  
  
@write_api  
  
def  
  
remove_books  
(  
    self  
,  
  
    book_ids  
,  
  
    permanent  
    =  
    False  
):  
  
    """ Remove the books specified by the book_ids from the database and delete  
        their format files. If ``permanent`` is False, then the format files  
        are placed in the recycle bin. """
```



```
path_map
```

```
=
```

```
{}
```

```
for
```

```
book_id
```

```
in
```

```
book_ids
```

```
:
```

```
try
```

```
:
```

```
path
```

```
=
```

```
self
```

```
.
```

```
_field_for
```

```
(
```

```
'path'
```

```
,
```

```
book_id
```

```
)
```

```
.
```

```
replace
```

```
(
```

```
'/'
```

```
,
```

```
os
```

```
.
sep
)

except
:

path

=

None

path_map
[
book_id
]

=

path

if

iswindows
:

paths

=

(
x
.
replace
(
os
.

```

```
sep
,
'/'
)

for

x

in

itervalues
(
path_map
)

if

x
)

self
.
backend
.
windows_check_if_files_in_use
(
paths
)

self
.
backend
.
remove_books
(
path_map
,
```

```
permanent
=
permanent
)
```

```
for
```

```
field
```

```
in
```

```
intervalues
(
self
.
fields
):
```

```
try
:
```

```
table
```

```
=
```

```
field
.
table
```

```
except
```

```
AttributeError
:
```

```
continue
```

```
# Some fields like ondevice do not have tables
```

```
else
:

table
.
remove_books
(
book_ids
,

self
.
backend
)

self
.
_search_api
.
discard_books
(
book_ids
)

self
.
_clear_caches
(
book_ids
=
book_ids
,

template_cache
=
False
,

search_cache
```

```
=
False
)

for

cc

in

self
.
cover_caches
:

cc
.
invalidate
(
book_ids
)

self
.
event_dispatcher
(
EventType
.
books_removed
,
book_ids
)

@read_api

def
```

```
author_sort_strings_for_books
```

```
(  
    self
```

```
,
```

```
    book_ids
```

```
):
```

```
    val_map
```

```
=
```

```
{}
```

```
    for
```

```
        book_id
```

```
    in
```

```
        book_ids
```

```
:
```

```
        authors
```

```
=
```

```
        self
```

```
.
```

```
        _field_ids_for
```

```
(
```

```
        'authors'
```

```
,
```

```
        book_id
```

```
)
```

```
adata
```

```
=  
  
self  
.  
_author_data  
(  
authors  
)
```

```
val_map  
[  
book_id  
]
```

```
=  
  
tuple  
(  
adata  
[  
aid  
][  
'sort'  
]
```

```
for
```

```
aid
```

```
in
```

```
authors  
)
```

```
return
```

```
val_map
```


[\[docs\]](#)

@write_api

def

rename_items

(

self

,

field

,

item_id_to_new_name_map

,

change_index

=

True

,

restrict_to_book_ids

=

None

):

'''

Rename items from a many-one or many-many field such as tags or series.

:param change_index: When renaming in a series-like field also change the series_index values.

:param restrict_to_book_ids: An optional set of book ids for which the rename is to be performed, defaults to all books.

'''

```
f
=

self
.
fields
[
field
]

affected_books

=

set
()

try
:

sv

=

f
.
metadata
[
'is_multiple'
][
'ui_to_list'
]

except

(
TypeError
```

```
,
KeyError
',
AttributeError
):

sv

=

None

if

restrict_to_book_ids

is

not

None
:

# We have a VL. Only change the item name for those books

if

not

isinstance
(
restrict_to_book_ids
,
(
Set
```

```
,  
  
MutableSet  
)):  
  
restrict_to_book_ids  
  
=  
  
frozenset  
(  
restrict_to_book_ids  
)  
  
id_map  
  
=  
  
{}  
  
default_process_map  
  
=  
  
{}  
  
for  
  
old_id  
,  
  
new_name  
  
in  
  
iteritems  
(  
item_id_to_new_name_map  
):
```

```
new_names
```

```
=
```

```
tuple
```

```
(
```

```
x
```

```
.
```

```
strip
```

```
()
```

```
for
```

```
x
```

```
in
```

```
new_name
```

```
.
```

```
split
```

```
(
```

```
sv
```

```
))
```

```
if
```

```
sv
```

```
else
```

```
(
```

```
new_name
```

```
,)
```

```
# Get a list of books in the VL with the item
```

```
books_with_id
```

```
=
```

```
f
.
books_for
(
old_id
)

books_to_process

=

books_with_id

&

restrict_to_book_ids
```

```
if

len
(
books_with_id
)

==

len
(
books_to_process
):
```

All the books with the ID are in the VL, so we can use

the normal processing

```
default_process_map
[
```

```
old_id
]

=

new_name

elif

books_to_process
:

affected_books
.
update
(
books_to_process
)

newvals

=

{}

for

book_id

in

books_to_process
:

# Get the current values, remove the one being renamed, then add

# the new value(s) back.
```

```
vals

=

self
.
_field_for
(
field
,
book_id
)

# Check for is_multiple

if

isinstance
(
vals
,
tuple
):

# We must preserve order.

vals

=

list
(
vals
)
```



```
# Don't need to worry about case here because we
```

```
# are fetching its one-true spelling. But lets be
```

```
# careful anyway
```

```
try
```

```
:
```

```
dex
```

```
=
```

```
vals
```

```
.
```

```
index
```

```
(
```

```
self
```

```
.
```

```
_get_item_name
```

```
(
```

```
field
```

```
,
```

```
old_id
```

```
))
```

```
# This can put the name back with a different case
```

```
vals
```

```
[
```

```
dex
```

```
]
```

```
=
```

```
new_names
[
0
]

# now add any other items if they aren't already there

if

len
(
new_names
)

>

1
:

set_vals

=

{
icu_lower
(
x
)

for

x

in

vals
}

for
```

```
v
```

```
in
```

```
new_names
```

```
[
```

```
1
```

```
:]:
```

```
lv
```

```
=
```

```
icu_lower
```

```
(
```

```
v
```

```
)
```

```
if
```

```
lv
```

```
not
```

```
in
```

```
set_vals
```

```
:
```

```
vals
```

```
.
```

```
append
```

```
(
```

```
v
```

```
)
```

```
set_vals
```

```
.
```

```
add
(
lv
)

newvals
[
book_id
]

=

vals

except

Exception
:

traceback
.
print_exc
()

else
:

newvals
[
book_id
]

=

new_names
[
0
]
```

```
# Allow case changes
```

```
self
```

```
.
```

```
_set_field
```

```
(
```

```
field
```

```
,
```

```
newvals
```

```
)
```

```
id_map
```

```
[
```

```
old_id
```

```
]
```

```
=
```

```
self
```

```
.
```

```
_get_item_id
```

```
(
```

```
field
```

```
,
```

```
new_names
```

```
[
```

```
0
```

```
])
```

```
if
```

```
default_process_map
```

```
:
```

```
ab
```

```
,  
  
idm  
  
=  
  
self  
.  
_rename_items  
(  
field  
,  
  
default_process_map  
,  
  
change_index  
=  
change_index  
)  
  
affected_books  
.  
update  
(  
ab  
)  
  
id_map  
.  
update  
(  
idm  
)  
  
self  
.  
event_dispatcher  
(  
EventType
```

```
.  
items_renamed  
,  
  
field  
,  
  
affected_books  
,  
  
id_map  
)
```

```
return
```

```
affected_books  
,  
  
id_map
```

```
try
```

```
:
```

```
func
```

```
=
```

```
f
```

```
.
```

```
table
```

```
.
```

```
rename_item
```

```
except
```

```
AttributeError
```

```
:
```

```
raise
```

```
ValueError
```

```
(  
'Cannot rename items for one-one fields:  
%s'  
,
```

```
%
```

```
field
```

```
)
```

```
moved_books
```

```
=
```

```
set
```

```
()
```

```
id_map
```

```
=
```

```
{}
```

```
for
```

```
item_id
```

```
,
```

```
new_name
```

```
in
```

```
iteritems
```

```
(  
item_id_to_new_name_map  
):
```



```
new_names
```

```
=
```

```
tuple
```

```
(
```

```
x
```

```
.
```

```
strip
```

```
()
```

```
for
```

```
x
```

```
in
```

```
new_name
```

```
.
```

```
split
```

```
(
```

```
sv
```

```
))
```

```
if
```

```
sv
```

```
else
```

```
(
```

```
new_name
```

```
,)
```

```
books
```

```
,
```

```
new_id
```

```
=
```

```
func
(
item_id
,

new_names
[
0
],

self
.
backend
)

affected_books
.
update
(
books
)

id_map
[
item_id
]

=

new_id

if

new_id

!=

item_id
:
```

```
moved_books
```

```
.
```

```
update
```

```
(
```

```
books
```

```
)
```

```
if
```

```
len
```

```
(
```

```
new_names
```

```
)
```

```
>
```

```
1
```

```
:
```

```
# Add the extra items to the books
```

```
extra
```

```
=
```

```
new_names
```

```
[
```

```
1
```

```
:]
```

```
self
```

```
.
```

```
_set_field
```

```
(
```

```
field
```

```
,
```

```
{
book_id
:
self
.
_fast_field_for
(
f
,
book_id
)
+

extra

for

book_id

in

books
})

if

affected_books
:

if

field

==

'authors'
:
```

```

self
.
  _set_field
(
  'author_sort'
,

{
k
:
' & '
.
join
(
v
)

for

k
,

v

in

iteritems
(
self
.
  _author_sort_strings_for_books
(
affected_books
)))

self
.
  _update_path
(
affected_books

```

```
,  
  
mark_as_dirtied  
=  
False  
)  
  
elif  
  
change_index  
  
and  
  
hasattr  
(  
f  
,  
  
'index_field'  
)  
  
and  
  
tweaks  
[  
'series_index_auto_increment'  
]  
  
!=  
  
'no_change'  
:  
  
for  
  
book_id  
  
in  
  
moved_books  
:
```

```
self
.
_set_field
(
f
.
index_field
.
name
,

{
book_id
:
self
.
_get_next_series_num_for
(
self
.
_fast_field_for
(
f
,

book_id
),

field
=
field
))

self
.
_mark_as_dirty
(
affected_books
)
```

```
self
    .
    event_dispatcher
    (
        EventType
    .
    items_renamed
    ,

    field
    ,

    affected_books
    ,

    id_map
    )
```

return

```
affected_books
,

id_map
```

[\[docs\]](#)

@write_api

def

```
remove_items
(
    self
    ,

    field
    ,
```



```
item_ids
```

```
,
```

```
restrict_to_book_ids
```

```
=
```

```
None
```

```
):
```

```
""" Delete all items in the specified field with the specified ids.
```

```
    Returns the set of affected book ids. ``restrict_to_book_ids`` is an
```

```
    optional set of books ids. If specified the items will only be removed
```

```
    from those books. """
```

```
field
```

```
=
```

```
self
```

```
.
```

```
fields
```

```
[
```

```
field
```

```
]
```

```
if
```

```
restrict_to_book_ids
```

```
is
```

```
not
```

```
None
```

```
and
```

```
not
```

```
isinstance
(
    restrict_to_book_ids
    ,

    (
        MutableSet
        ,

        Set
    )):
```

```
restrict_to_book_ids

=
```

```
frozenset
(
    restrict_to_book_ids
)
```

```
affected_books
```

```
=
```

```
field
.
table
.
remove_items
(
    item_ids
    ,

    self
    .
    backend
    ,
```

```
restrict_to_book_ids  
=  
restrict_to_book_ids  
)
```

```
if
```

```
affected_books  
:
```

```
if
```

```
hasattr  
(  
field  
,  
  
'index_field'  
):
```

```
self  
.  
_set_field  
(  
field  
.  
index_field  
.  
name  
,
```

```
{  
bid  
:  
1.0
```

```
for
```

```
bid
```

in

```
affected_books  
})
```

else

:

self

.

```
_mark_as_dirty  
(  
affected_books  
)
```

self

.

```
event_dispatcher  
(  
EventType
```

.

```
items_removed  
,
```

field

,

```
affected_books  
,
```

```
item_ids  
)
```

return

```
affected_books
```

[\[docs\]](#)

@write_api

def

add_custom_book_data

(

self

,

name

,

val_map

,

delete_first

=

False

):

""" Add data for name where val_map is a map of book_ids to values. If

delete_first is True, all previously stored data for name will be

removed. """

missing

=

frozenset

(

val_map

)

-

self

```
.
_all_book_ids
()

if

missing
:

raise

ValueError
(
'add_custom_book_data: no such book_ids:
%d
'
%
missing
)

self

.
backend

.
add_custom_data
(
name
,

val_map
,

delete_first
)

\[docs\]

@read_api
```

def

get_custom_book_data

(
self

,

name

,

book_ids

=

()),

default

=

None

):

''' Get data for name. By default returns data for all book_ids, pass

in a list of book ids if you only want some data. Returns a map of

book_id to values. If a particular value could not be decoded, uses

default for it. '''

return

self

.

backend

.

get_custom_book_data

(

name

,

book_ids

,

```
default
)
```

[\[docs\]](#)

```
@write_api
```

```
def
```

```
delete_custom_book_data
```

```
(
    self
,
```

```
    name
,
```

```
    book_ids
=
()):
```

```
""" Delete data for name. By default deletes all data, if you only want
    to delete data for some book ids, pass in a list of book ids. """
```

```
self
```

```
·
backend
```

```
·
delete_custom_book_data
(
    name
,
    book_ids
)
```


[\[docs\]](#)

@read_api

def

get_ids_for_custom_book_data

(
self

,

name

):

""" Return the set of book ids for which name has data. """

return

self

.

backend

.

get_ids_for_custom_book_data

(
name
)

@read_api

def

conversion_options

(
self

,

book_id

,

```
fmt
=
'PIPE'
):
```

```
return
```

```
self
.
backend
.
conversion_options
(
book_id
,
fmt
)
```

```
@read_api
```

```
def
```

```
has_conversion_options
(
self
,
ids
,
fmt
=
'PIPE'
):
```

```
return
```

```
self
.
backend
.
has_conversion_options
(
ids
,
fmt
)

@write_api

def
delete_conversion_options
(
self
,
book_ids
,
fmt
=
'PIPE'
):

return

self
.
backend
.
delete_conversion_options
(
book_ids
```

```
,
```

```
fmt
```

```
)
```

```
\[docs\]
```

```
@write_api
```

```
def
```

```
set_conversion_options
```

```
(
```

```
self
```

```
,
```

```
options
```

```
,
```

```
fmt
```

```
=
```

```
'PIPE'
```

```
):
```

```
""" options must be a map of the form {book_id:conversion_options} """
```

```
return
```

```
self
```

```
.
```

```
backend
```

```
.
```

```
set_conversion_options
```

```
(
```

```
options
```

```
,
```

```
fmt
)
```

```
@write_api
```

```
def
```

```
refresh_format_cache
(
self
):
```

```
self
```

```
.
```

```
fields
```

```
[
```

```
'formats'
```

```
]
```

```
.
```

```
table
```

```
.
```

```
read
```

```
(
```

```
self
```

```
.
```

```
backend
```

```
)
```

```
self
```

```
.
```

```
format_metadata_cache
```

```
.
```

```
clear
```

```
()
```

```
@write_api
```

def

refresh_onddevice

(
self
):

self

.

fields

[
'onddevice'
]

.

clear_caches

()

self

.

clear_search_caches

()

self

.

clear_composite_caches

()

@read_api

def

books_matching_device_book

(
self
,

```
lpath
):

ans

=

set
()

for

book_id
,

(
-
,

-
,

-
,

-
,

-
,

lpaths
)

in

self
.
fields
[
'ondevice'
]
.
cache
```

```
.
items
():

if

lpath

in

lpaths
:

ans
.
add
(
book_id
)
```

```
return
```

```
ans
```

```
\[docs\]
```

```
@read_api
```

```
def
```

```
tags_older_than
```

```
(
self
,
```

```
tag
```



```

,

delta
=
None
,

must_have_tag
=
None
,

must_have_authors
=
None
):

'''

    Return the ids of all books having the tag ``tag`` that are older than

    the specified time. tag comparison is case insensitive.

    :param delta: A timedelta object or None. If None, then all ids with

        the tag are returned.

    :param must_have_tag: If not None the list of matches will be

        restricted to books that have this tag

    :param must_have_authors: A list of authors. If not None the list of

        matches will be restricted to books that have these authors (case

        insensitive).

'''

```

```
tag_map
=
{
icu_lower
(
v
):
k

for

k
,

v

in

iteritems
(
self
.
_get_id_map
(
(
'tags'
)))

tag

=

icu_lower
(
tag
.
strip
())
```

```
mht
=
icu_lower
(
must_have_tag
.
strip
())
if
must_have_tag
else
None

tag_id
,
mht_id
=
tag_map
.
get
(
tag
,
None
),
tag_map
.
get
(
mht
```

```
,
```

```
None
```

```
)
```

```
ans
```

```
=
```

```
set
```

```
()
```

```
if
```

```
mht_id
```

```
is
```

```
None
```

```
and
```

```
mht
```

```
:
```

```
return
```

```
ans
```

```
if
```

```
tag_id
```

```
is
```

```
not
```

```
None
```

```
:
```

```
tagged_books
=
self
.
_books_for_field
(
'tags'
,
tag_id
)
```

```
if
```

```
mht_id
```

```
is
```

```
not
```

```
None
```

```
and
```

```
tagged_books
:
```

```
tagged_books
```

```
=
```

```
tagged_books
```

```
.
intersection
(
self
```

```
.
```

```
_books_for_field
```

```
(
```

```
'tags'
```

```
,
```

```
mht_id
```

```
))
```

```
if
```

```
tagged_books
```

```
:
```

```
if
```

```
must_have_authors
```

```
is
```

```
not
```

```
None
```

```
:
```

```
amap
```

```
=
```

```
{
```

```
icu_lower
```

```
(
```

```
v
```

```
):
```

```
k
```

```
for
```

```
k
```

```
,
```

```
v
in
iteritems
(
self
.
_get_id_map
(
'authors'
))}

books

=

None

for

author

in

must_have_authors
:

abooks

=

self
.
_books_for_field
(
'authors'
,
amap
```

```
.  
get  
(  
icu_lower  
(  
author  
),  
None  
))
```

```
books
```

```
=
```

```
abooks
```

```
if
```

```
books
```

```
is
```

```
None
```

```
else
```

```
books
```

```
.  
intersection  
(  
abooks  
)
```

```
if
```

```
not
```

```
books
```

```
:
```


break

tagged_books

=

tagged_books

.

intersection

(

books

or

set

())

if

delta

is

None

:

ans

=

tagged_books

else

:

now

```
nowf
()
```

book_id

```
tagged_books
:
```

```

    _field_for

```

```
_field_for
(
'timestamp'
,
```

```
book_id
)
```

(now

```
delta
```

```
:
```

```
ans
```

```
.
```

```
add
```

```
(
```

```
book_id
```

```
)
```

```
return
```

```
ans
```

```
@write_api
```

```
def
```

```
set_sort_for_authors
```

```
(
```

```
self
```

```
,
```

```
author_id_to_sort_map
```

```
,
```

```
update_books
```

```
=
```

```
True
```

```
):
```

```
sort_map
```

```
=
```

```
self
```

```
.
```

```
fields
```

```
[
'authors'
]
.
table
.
set_sort_names
(
author_id_to_sort_map
,

self
.
backend
)

changed_books

=

set
()

if

update_books
:

val_map

=

{}

for

author_id

in
```

```
sort_map
:

books

=

self
.
  _books_for_field
(
  'authors'
,
  author_id
)

changed_books

|=

books

for
book_id

in
books
:

authors

=

self
.
```

```
_field_ids_for  
(  
    'authors'  
,  
    book_id  
)
```

```
adata
```

```
=
```

```
self  
.  
_author_data  
(  
    authors  
)
```

```
sorts
```

```
=
```

```
[  
    adata  
    [  
        x  
    ](  
        'sort'  
    )  
]
```

```
for
```

```
x
```

```
in
```

```
authors  
]
```

```
val_map
[
  book_id
]

=

' & '
.
join
(
  sorts
)

if

val_map
:

self
.
  _set_field
(
  'author_sort'
,
  val_map
)

if

changed_books
:

self
.
  _mark_as_dirty
(
  changed_books
```

```
)

return

changed_books

@write_api

def

set_link_for_authors
(
self
,

author_id_to_link_map
):

link_map

=

self
.
fields
[
'authors'
]
.
table
.
set_links
(
author_id_to_link_map
,
self
.
```



```
backend
)

changed_books

=

set
()

for

author_id

in

link_map
:

changed_books

|=

self
.
  _books_for_field
(
  'authors'
,
  author_id
)

if

changed_books
:
```

```
self
.  
_mark_as_dirty  
(  
changed_books  
)
```

```
return
```

```
changed_books
```

```
@read_api
```

```
def
```

```
lookup_by_uuid  
(  
self  
,  
  
uuid  
):
```

```
return
```

```
self  
.  
fields  
[  
'uuid'  
]  
.  
table  
.  
lookup_by_uuid  
(  
uuid  
)
```

```
@write_api
```

```
def
```

```
delete_custom_column
```

```
(  
self
```

```
,
```

```
label
```

```
=
```

```
None
```

```
,
```

```
num
```

```
=
```

```
None
```

```
):
```

```
self
```

```
.
```

```
backend
```

```
.
```

```
delete_custom_column
```

```
(  
label
```

```
,
```

```
num
```

```
)
```

```
@write_api
```

```
def
```

```
create_custom_column
(
    self
    ,

    label
    ,

    name
    ,

    datatype
    ,

    is_multiple
    ,

    editable
    =
    True
    ,

    display
    =
    {}):

return

self
.
backend
.
create_custom_column
(
    label
    ,

    name
    ,

    datatype
    ,
```

```
is_multiple
```

```
,
```

```
editable
```

```
=
```

```
editable
```

```
,
```

```
display
```

```
=
```

```
display
```

```
)
```

```
@write_api
```

```
def
```

```
set_custom_column_metadata
```

```
(
```

```
self
```

```
,
```

```
num
```

```
,
```

```
name
```

```
=
```

```
None
```

```
,
```

```
label
```

```
=
```

```
None
```

```
,
```

```
is_editable
```

```
=
```

```
None
```

```
,
```

```
display
=
None
,

update_last_modified
=
False
):

changed

=

self
.
backend
.
set_custom_column_metadata
(
num
,

name
=
name
,

label
=
label
,

is_editable
=
is_editable
,

display
=
```

```
display
)

if
changed
:

if
update_last_modified
:

self
.
_update_last_modified
(
self
.
_all_book_ids
())

else
:

self
.
backend
.
prefs
.
set
(
'update_all_last_mod_dates_on_start'
,
True
)
```

```
return
```

```
changed
```

```
@read_api
```

```
def
```

```
get_books_for_category
```

```
(  
    self
```

```
,
```

```
category
```

```
,
```

```
item_id_or_composite_value  
):
```

```
f
```

```
=
```

```
self
```

```
.
```

```
fields
```

```
[  
    category  
]
```

```
if
```

```
hasattr
```

```
(
```

```
f
```

```
,
```



```
'get_books_for_val'
):

# Composite field

return

f
.
get_books_for_val
(
item_id_or_composite_value
,

self
.
_get_proxy_metadata
,

self
.
_all_book_ids
())

return

self
.
_books_for_field
(
f
.
name
,

int
(
item_id_or_composite_value
))
```

[\[docs\]](#)

@read_api

def

data_for_find_identical_books

(

self

):

""" Return data that can be used to implement

:meth:`find\_identical\_books` in a worker process without access to the

db. See db.utils for an implementation. """

at

=

self

.

fields

[

'authors'

]

.

table

author_map

=

defaultdict

```
(
set
)

for

aid
,

author

in

iteritems
(
at
.
id_map
):

author_map
[
icu_lower
(
author
)]
.
add
(
aid
)

return

(
author_map
,

at
.
```

```
col_book_map
.  
copy  
(  
  
self  
.  
fields  
[  
    'title'  
]  
.  
table  
.  
book_col_map  
.  
copy  
(  
  
self  
.  
fields  
[  
    'languages'  
]  
.  
book_value_map  
.  
copy  
(  
  
@read_api  
  
def  
  
update_data_for_find_identical_books  
(  
    self  
    ,  
  
    book_id
```

```
,  
  
data  
):  
  
author_map  
,  
  
author_book_map  
,  
  
title_map  
,  
  
lang_map  
  
=  
  
data  
  
title_map  
[  
book_id  
]  
  
=  
  
self  
.  
_field_for  
(  
    'title'  
,  
  
    book_id  
)  
  
lang_map  
[  
    book_id
```

```
]

=

self
.
_field_for
(
'languages'
,

book_id
)

at

=

self
.
fields
[
'authors'
]
.
table

for

aid

in

at

.
book_col_map
.
get
(
book_id
,
```

```
()):
```

```
author_map
```

```
[
```

```
icu_lower
```

```
(
```

```
at
```

```
.
```

```
id_map
```

```
[
```

```
aid
```

```
]])
```

```
.
```

```
add
```

```
(
```

```
aid
```

```
)
```

```
try
```

```
:
```

```
author_book_map
```

```
[
```

```
aid
```

```
]
```

```
.
```

```
add
```

```
(
```

```
book_id
```

```
)
```

```
except
```

```
KeyError
```

```
:
```

```
author_book_map
```

```
[  
aid  
]
```

```
=
```

```
{  
book_id  
}
```

[\[docs\]](#)

@read_api

def

find_identical_books

```
(  
self  
,
```

mi

```
,
```

search_restriction

```
=  
"
```

```
,
```

book_ids

```
=
```

None

```
):
```

""" Finds books that have a superset of the authors in mi and the same

title (title is fuzzy matched). See also :meth:`data\_for\_find\_identical\_books`. """


```
from
calibre.db.utils

import

fuzzy_title

identical_book_ids

=

set
()

langq

=

tuple
(
x

for
x

in
map
(
canonicalize_lang
,
mi
.
languages

or
```

```
()
```

```
if
```

```
x
```

```
and
```

```
x
```

```
!=
```

```
'und'
```

```
)
```

```
if
```

```
mi
```

```
.
```

```
authors
```

```
:
```

```
try
```

```
:
```

```
quathors
```

```
=
```

```
mi
```

```
.
```

```
authors
```

```
[:
```

```
20
```

```
]
```

```
# Too many authors causes parsing of the search expression to fail
```

```
query
```

```
=  
  
' and '  
,  
join  
(  
  'authors:="  
  %s  
  "'  
  %  
  (  
  a  
  ,  
  "  
  ))  
  
for  
a  
in  
quathors  
)  
  
qauthors  
  
=  
  
mi  
.  
authors  
[  
  20  
:]
```

```
except
```

```
ValueError
```

```
:
```

```
return
```

```
identical_book_ids
```

```
try
```

```
:
```

```
book_ids
```

```
=
```

```
self
```

```
.
```

```
_search
```

```
(
```

```
query
```

```
,
```

```
restriction
```

```
=
```

```
search_restriction
```

```
,
```

```
book_ids
```

```
=
```

```
book_ids
```

```
)
```

```
except
```

```
:
```

```
traceback
```

```
.
print_exc
()

return

identical_book_ids

if

qauthors

and

book_ids
:

matches

=

set
()

qauthors

=

{
icu_lower
(
x
)
}

for

x

in
```

```
qauthors
}
```

```
for
```

```
book_id
```

```
in
```

```
book_ids
:
```

```
aut
```

```
=
```

```
self
```

```
._field_for
```

```
(  
'authors'  
,
```

```
book_id  
)
```

```
if
```

```
aut  
:
```

```
aut
```

```
=
```

```
{  
icu_lower  
(
```

```
x
)

for

x

in

aut
}

if

aut
.
issuperset
(
qauthors
):

matches
.
add
(
book_id
)

book_ids

=

matches

for

book_id
```

in

book_ids

:

fbook_title

=

self

.

_field_for

(

'title'

,

book_id

)

fbook_title

=

fuzzy_title

(

fbook_title

)

mbook_title

=

fuzzy_title

(

mi

.

title

)


```
if
fbook_title
==
mbook_title
:
bl
=
self
.
_field_for
(
'languages'
,
book_id
)

if
not
langq
or
not
bl
or
bl
==
```

```
langq
:

identical_book_ids
.
add
(
book_id
)

return

identical_book_ids

@read_api

def

get_top_level_move_items
(
self
):

all_paths

=

{
self
.
_field_for
(
'path'
,
book_id
)
.
}
```

```
partition
(
    '/'
)[
    0
]

for

book_id

in

self
.
_all_book_ids
()}

return

self
.
backend
.
get_top_level_move_items
(
    all_paths
)

@write_api

def

move_library_to
(
    self
,

newloc
```

```
,  
  
progress  
=  
None  
,  
  
abort  
=  
None  
):  
  
def  
  
progress_callback  
(  
item_name  
,  
  
item_count  
,  
  
total  
):  
  
try  
:  
  
if  
  
progress  
  
is  
  
not  
  
None  
:
```

```
progress
(
item_name
,
item_count
,
total
)
```

```
except
```

```
Exception
:
```

```
import
```

```
traceback
```

```
traceback
.
print_exc
()
```

```
all_paths
```

```
=
```

```
{
self
.
_field_for
(
'path'
,
book_id
```

```
)  
.  
partition  
(  
    '/'  
)[  
    0  
]  
  
for  
  
    book_id  
  
    in  
  
        self  
        .  
        _all_book_ids  
        ()}  
  
self  
.  
backend  
.  
move_library_to  
(  
    all_paths  
    ,  
  
    newloc  
    ,  
  
    progress  
    =  
    progress_callback  
    ,  
  
    abort  
    =  
    abort  
)
```

```
@read_api
```

```
def
```

```
saved_search_names
```

```
(  
    self  
):
```

```
return
```

```
self
```

```
·  
_search_api
```

```
·  
saved_searches
```

```
·  
names  
()
```

```
@read_api
```

```
def
```

```
saved_search_lookup
```

```
(  
    self  
    ,
```

```
    name  
):
```

```
return
```

```
self
```

```
.
_search_api
.
saved_searches
.
lookup
(
name
)

@write_api

def
saved_search_set_all
(
self
,

smap
):

self
.
_search_api
.
saved_searches
.
set_all
(
smap
)

self
.
_clear_search_caches
()
```



```
@write_api
```

```
def
```

```
saved_search_delete
```

```
(  
self
```

```
,
```

```
name
```

```
):
```

```
self
```

```
.
```

```
_search_api
```

```
.
```

```
saved_searches
```

```
.
```

```
delete
```

```
(
```

```
name
```

```
)
```

```
self
```

```
.
```

```
_clear_search_caches
```

```
()
```

```
@write_api
```

```
def
```

```
saved_search_add
```

```
(
```

```
self
```

```
,  
  
name  
,  
  
val  
):  
  
self  
.  
_search_api  
.  
saved_searches  
.  
add  
(  
name  
,  
  
val  
)
```

```
@write_api
```

```
def  
  
saved_search_rename  
(  
self  
,  
  
old_name  
,  
  
new_name  
):  
  
self
```

```
.
_search_api
.
saved_searches
.
rename
(
old_name
,
new_name
)

self
.
_clear_search_caches
()
```

@write_api

def

```
change_search_locations
(
self
,
newlocs
):
```

```
self
.
_search_api
.
change_locations
(
newlocs
)
```

```
@write_api
```

```
def
```

```
refresh_search_locations
```

```
(  
    self  
):
```

```
self
```

```
._search_api
```

```
change_locations
```

```
(  
    self
```

```
._field_metadata
```

```
._get_search_terms  
())
```

```
@write_api
```

```
def
```

```
dump_and_restore
```

```
(  
    self
```

```
,
```

```
callback
```

```
=
```

```
None
```

```
,
```

```
sql
=
None
):

return

self
.
backend
.
dump_and_restore
(
callback
=
callback
,

sql
=
sql
)
```

@write_api

```
def

vacuum
(
self
):

self
.
backend
.
vacuum
```

```
()
```

```
@write_api
```

```
def
```

```
close
```

```
(  
    self  
):
```

```
self
```

```
.  
event_dispatcher
```

```
.  
close  
()
```

```
from
```

```
calibre.customize.ui
```

```
import
```

```
available_library_closed_plugins
```

```
for
```

```
plugin
```

```
in
```

```
available_library_closed_plugins  
():
```

```
try
```

```
:  
  
plugin  
.  
run  
(  
    self  
)  
  
except  
Exception  
:  
  
import  
traceback  
  
traceback  
.  
print_exc  
()  
  
self  
.  
backend  
.  
close  
()  
  
@property  
  
def  
is_closed
```

```
(
    self
):

    return

    self
    .
    backend
    .
    is_closed

\[docs\]

@write_api

def
restore_book
(
    self
    ,
    book_id
    ,
    mi
    ,
    last_modified
    ,
    path
    ,
    formats
    ,
```



```
annotations
=
():

""" Restore the book entry in the database for a book that already exists on the filesystem """

cover

=

mi
.
cover

mi
.
cover

=

None

self
.
_create_book_entry
(
mi
,
add_duplicates
=
True
,
force_id
=
book_id
,
```

```
apply_import_tags
=
False
,

preserve_uuid
=
True
)

self
.
_update_last_modified
((
book_id
,)),

last_modified
)

if

cover

and

os
.
path
.
exists
(
cover
):

self
.
_set_field
(
```

```
'cover'
,

{
book_id
:
1
})

self
.
backend
.
restore_book
(
book_id
,

path
,

formats
)

if

annotations
:

self
.
_restore_annotations
(
book_id
,

annotations
)
```

```
@read_api
```

```
def
```

```
virtual_libraries_for_books
```

```
(
```

```
self
```

```
,
```

```
book_ids
```

```
,
```

```
virtual_fields
```

```
=
```

```
None
```

```
):
```

```
# use a primitive lock to ensure that only one thread is updating
```

```
# the cache and that recursive calls don't do the update. This
```

```
# method can recurse via self._search()
```

```
with
```

```
try_lock
```

```
(
```

```
self
```

```
.
```

```
vls_cache_lock
```

```
)
```

```
as
```

```
got_lock
```

```
:
```

Using a list is slightly faster than a set.

c

=

defaultdict

(
list
)

if

not

got_lock
:

We get here if resolving the books in a VL triggers another VL

cache calculation. This can be 'real' recursion, for example a

VL expression using a template that calls virtual_libraries(),

or a search using a location of 'all' that causes evaluation

of a composite that uses virtual_libraries(). The first case

is an error and the exception message should appear somewhere.

However, the error can seem nondeterministic. It might not be

raised if the use is via a composite and that composite is

evaluated before it is used in the search. The second case is

also an error but if the composite isn't used in a VL then the

eventual answer will be correct because get_metadata() will

clear the caches.

raise

ValueError

(

—
(

'Recursion detected while processing Virtual library "

%s

""

)

%

self

.

vls_for_books_lib_in_process

)

if

self

.

vls_for_books_cache

is

None

```
:  
  
libraries  
  
=  
  
self  
.  
_pref  
(  
    'virtual_libraries'  
,  
    {})  
  
for  
  
lib  
,  
  
expr  
  
in  
  
libraries  
.  
items  
():  
  
    book  
  
    =  
  
    None  
  
self  
.  
vls_for_books_lib_in_process
```

```
=  
  
lib  
  
try  
:  
  
for  
  
book  
  
in  
  
self  
.  
_search  
(  
expr  
,  
  
virtual_fields  
=  
virtual_fields  
):  
  
c  
[  
book  
]  
.  
append  
(  
lib  
)  
  
except  
  
Exception
```



```
as

e
:

if

book
:

c
[
book
]
.
append
(
    -
    (
        '[Error in Virtual library
{0}
:
{1}
]
'
    )
    .
format
(
    lib
    ,
    str
    (
        e
    )))

self

.
vls_for_books_cache
```

```
=  
  
{  
b  
:  
tuple  
(  
sorted  
(  
libs  
,  
  
key  
=  
sort_key  
))  
  
for  
  
b  
,  
  
libs  
  
in  
  
c  
.  
items  
())  
  
if  
  
not  
  
book_ids  
:  
  
book_ids  
  
=
```

```
self
    .
    _all_book_ids
    ()

# book_ids is usually 1 long. The loop will be faster than a comprehension

r
=
{}

default
=
()

for
    b
    in
        book_ids
        :

    r
    [
        b
    ]
    =
    self
    .
```

```
vls_for_books_cache
```

```
.
```

```
get
```

```
(
```

```
b
```

```
,
```

```
default
```

```
)
```

```
return
```

```
r
```

```
\[docs\]
```

```
@read_api
```

```
def
```

```
user_categories_for_books
```

```
(
```

```
self
```

```
,
```

```
book_ids
```

```
,
```

```
proxy_metadata_map
```

```
=
```

```
None
```

```
):
```

```
""" Return the user categories for the specified books.
```

```
    proxy_metadata_map is optional and is useful for a performance boost,
```

in contexts where a ProxyMetadata object for the books already exists.

It should be a mapping of book_ids to their corresponding ProxyMetadata objects.

'''

```
user_cats
=
self
.
 _pref
(
'user_categories'
,
{})

pmm
=
proxy_metadata_map
or
{}

ans
=
{}

for
```

```
book_id
```

```
in
```

```
book_ids
```

```
:
```

```
proxy_metadata
```

```
=
```

```
pmm
```

```
.
```

```
get
```

```
(
```

```
book_id
```

```
)
```

```
or
```

```
self
```

```
.
```

```
_get_proxy_metadata
```

```
(
```

```
book_id
```

```
)
```

```
user_cat_vals
```

```
=
```

```
ans
```

```
[
```

```
book_id
```

```
]
```

```
=
```

```
{}
```

for

ucat

,

categories

in

iteritems

(

user_cats

):

user_cat_vals

[

ucat

]

=

res

=

[]

for

name

,

cat

,

ign

in

categories

```
:  
  
try  
:  
  
    field_obj  
  
    =  
  
    self  
    .  
    fields  
    [  
    cat  
    ]  
  
except  
  
    KeyError  
    :  
  
        continue  
  
  
if  
  
    field_obj  
    .  
    is_composite  
    :  
  
        v  
  
        =  
  
        field_obj  
        .
```



```
get_value_with_cache
```

```
(  
    book_id  
    ,
```

```
    lambda
```

```
        x  
        :  
        proxy_metadata  
    )
```

```
    else
```

```
        :
```

```
        v
```

```
        =
```

```
        self
```

```
        .  
        _fast_field_for  
        (  
            field_obj  
            ,  
            book_id  
        )
```

```
    if
```

```
        isinstance
```

```
        (  
            v  
            ,
```

```
            (  
                list
```

```
            ,
```

```
tuple
)):
```

```
if
```

```
name
```

```
in
```

```
v
:
```

```
res
```

```
.
append
([
name
,
cat
])
```

```
elif
```

```
name
```

```
==
```

```
v
:
```

```
res
```

```
.
append
([
name
,
```

```
cat  
])
```

```
return
```

```
ans
```

```
\[docs\]
```

```
@write_api
```

```
def
```

```
embed_metadata
```

```
(  
self  
,
```

```
book_ids  
,
```

```
only_fmts  
=
```

```
None
```

```
,
```

```
report_error  
=
```

```
None
```

```
,
```

```
report_progress  
=
```

```
None
```

```
):
```

```
""" Update metadata in all formats of the specified book_ids to current metadata in the  
database. """
```

```
field
```

```
=
```

```
self
```

```
.
```

```
fields
```

```
[
```

```
'formats'
```

```
]
```

```
from
```

```
calibre.customize.ui
```

```
import
```

```
apply_null_metadata
```

```
from
```

```
calibre.ebooks.metadata.meta
```

```
import
```

```
set_metadata
```

```
from
```

```
calibre.ebooks.metadata.opf2
```

```
import
```

```
pretty_print
```

```
if
```

```
only_fmts
```

```
:  
  
only_fmts  
  
=  
  
{  
f  
.  
lower  
()  
  
for  
  
f  
  
in  
  
only_fmts  
}  
  
  
  
def  
  
doit  
(  
fmt  
,  
  
mi  
,  
  
stream  
):  
  
with  
  
apply_null_metadata  
,
```

```
pretty_print  
:
```

```
set_metadata  
(  
stream  
,
```

```
mi  
,
```

```
stream_type  
=  
fmt  
,
```

```
report_error  
=  
report_error  
)
```

```
stream
```

```
.  
seek  
(  
0  
,
```

```
os  
.  
SEEK_END  
)
```

```
return
```

```
stream  
.  
tell  
(
```

for

i

,

book_id

in

enumerate

(

book_ids

):

fmts

=

field

.

table

.

book_col_map

.

get

(

book_id

,

()))

if

not

fmts

:

continue

mi

=

self

.

get_metadata

(

book_id

,

get_cover

=

True

,

cover_as_data

=

True

)

try

:

path

=

self

.

_field_for

(

'path'

,

book_id

)

.


```
replace
(
    '/'
,
    os
.
sep
)

except
:

continue

for

fmt

in

fmts
:

if

only_fmts

is

not

None

and

fmt
.
lower
```

```
()

not

in

only_fmts
:

continue

try
:

name

=

self
.
fields
[
'formats'
]
.
format_fname
(
book_id
,
fmt
)

except
:

continue
```

```
if
    name
    and
    path
    :

    new_size

    =

    self
    .
    backend
    .
    apply_to_format
    (
    book_id
    ,
    path
    ,
    name
    ,
    fmt
    ,
    partial
    (
    doit
    ,
    fmt
    ,
    mi
    ))
```

if

new_size

is

not

None

:

self

.

format_metadata_cache

[

book_id

]

.

get

(

fmt

,

{}[

'size'

]

=

new_size

max_size

=

self

.

fields

[

```
'formats'
]
.
table
.
update_fmt
(
book_id
,

fmt
,

name
,

new_size
,

self
.
backend
)

self
.
fields
[
'size'
]
.
table
.
update_sizes
({
book_id
:

max_size
})
```

```
if
    report_progress
    is
    not
    None
:

report_progress
(
    i
    +
    1
    ,
    len
    (
        book_ids
    ),
    mi
)

@read_api

def
get_last_read_positions
(
    self
    ,
    book_id
    ,
    fmt
    ,
```

```
user  
):
```

```
fmt
```

```
=
```

```
fmt
```

```
.
```

```
upper  
()
```

```
ans
```

```
=
```

```
[]
```

```
for
```

```
device
```

```
,
```

```
cfi
```

```
,
```

```
epoch
```

```
,
```

```
pos_frac
```

```
in
```

```
self
```

```
.
```

```
backend
```

```
.
```

```
execute
```

```
(
```

```
'SELECT device,cfi,epoch,pos_frac FROM last_read_positions WHERE book=? AND format=?  
AND user=?'
```

```
,
```

```
(  
book_id
```

```
,
```

```
fmt
```

```
,
```

```
user
```

```
)):
```

```
ans
```

```
.
```

```
append
```

```
{{
```

```
'device'
```

```
:
```

```
device
```

```
,
```

```
'cfi'
```

```
:
```

```
cfi
```

```
,
```

```
'epoch'
```

```
:
```

```
epoch
```

```
,
```

```
'pos_frac'
```

```
:
```

```
pos_frac
```

```
}}
```



```
return
```

```
ans
```

```
@write_api
```

```
def
```

```
set_last_read_position
```

```
(  
    self
```

```
,
```

```
    book_id
```

```
,
```

```
    fmt
```

```
,
```

```
    user
```

```
    =
```

```
    ' '
```

```
    _
```

```
,
```

```
    device
```

```
    =
```

```
    ' '
```

```
    _
```

```
,
```

```
    cfi
```

```
    =
```

```
    None
```

```
,
```

```
    epoch
```

```
    =
```

```
    None
```

```
,
```

```
pos_frac
```

```
=
```

```
0
```

```
):
```

```
fmt
```

```
=
```

```
fmt
```

```
.
```

```
upper
```

```
()
```

```
device
```

```
=
```

```
device
```

```
or
```

```
' '
```

```
—
```

```
user
```

```
=
```

```
user
```

```
or
```

```
' '
```

```
—
```

```
if
```

```
not
```

```
cfi
```

```
:

self
.
backend
.
execute
(

'DELETE FROM last_read_positions WHERE book=? AND format=? AND user=? AND device=?'
,

(
book_id
,

fmt
,

user
,

device
))

else
:

self
.
backend
.
execute
(

'INSERT OR REPLACE INTO last_read_positions(book,format,user,device,cfi,epoch,pos_frac)
VALUES (?, ?, ?, ?, ?, ?, ?)'
```

```
,
```

```
(  
book_id
```

```
,
```

```
fmt
```

```
,
```

```
user
```

```
,
```

```
device
```

```
,
```

```
cfi
```

```
,
```

```
epoch
```

```
or
```

```
time
```

```
()),
```

```
pos_frac
```

```
))
```

```
@read_api
```

```
def
```

```
export_library
```

```
(
```

```
self
```

```
,
```

```
library_key
```

```
,
```

```
exporter
',

progress
=
None
',

abort
=
None
):

from
polyglot.binary

import

as_hex_unicode

key_prefix

=

as_hex_unicode
(
library_key
)

book_ids

=

self
.
_all_book_ids
()
```

```
total
```

```
=
```

```
len
```

```
(  
    book_ids  
)
```

```
+
```

```
1
```

```
format_metadata
```

```
=
```

```
{}
```

```
if
```

```
    progress
```

```
is
```

```
not
```

```
None
```

```
:
```

```
    progress
```

```
(  
    'metadata.db'  
,
```

```
    0
```

```
,
```

```
total
```

```
)

pt

=

PersistentTemporaryFile
(
'-export.db'
)

pt
.
close
()

self
.
backend
.
backup_database
(
pt
.
name
)

dbkey

=

key_prefix

+

'...'

+
```

```
'metadata.db'
```

```
with
```

```
lopen
```

```
(
```

```
pt
```

```
.
```

```
name
```

```
,
```

```
'rb'
```

```
)
```

```
as
```

```
f
```

```
:
```

```
exporter
```

```
.
```

```
add_file
```

```
(
```

```
f
```

```
,
```

```
dbkey
```

```
)
```

```
os
```

```
.
```

```
remove
```

```
(
```

```
pt
```

```
.
```

```
name
```

```
)
```

```
metadata
```



```
=  
  
{  
  'format_data'  
  :  
  format_metadata  
  ,  
  
  'metadata.db'  
  :  
  dbkey  
  ,  
  
  'total'  
  :  
  total  
}
```

for

```
i  
,  
  
book_id
```

in

```
enumerate  
(  
  book_ids  
):
```

if

abort

is

not

None

and

abort

.

is_set

():

return

if

progress

is

not

None

:

progress

(

self

.

_field_for

(

'title'

,

book_id

),

i

+

1

,

```
total  
)
```

```
format_metadata  
[  
book_id  
]
```

```
=
```

```
{}
```

```
for
```

```
fmt
```

```
in
```

```
self
```

```
.
```

```
_formats
```

```
(
```

```
book_id
```

```
):
```

```
mdata
```

```
=
```

```
self
```

```
.
```

```
format_metadata
```

```
(
```

```
book_id
```

```
,
```

```
fmt
```

```
)
```

key

=

f
,

{
key_prefix
}

:

{
book_id
}

:

{
fmt
}
,

format_metadata

[
book_id
][
fmt
]

=

key

with

exporter

.

start_file

(

key

,

mtime

```
=
mdata
.
get
(
'mtime'
))

as

dest
:

self
.
_copy_format_to
(
book_id
,
fmt
,
dest
,
report_file_size
=
dest
.
ensure_space
)

cover_key

=
'
{}
:
{}

```

```
:  
{  
.  
format  
(  
key_prefix  
,  
  
book_id  
,  
  
'cover'  
)
```

with

```
exporter  
.  
start_file  
(  
cover_key  
)
```

as

```
dest  
:
```

if

not

```
self  
.  
copy_cover_to  
(  
book_id  
,  
  
dest
```

```
,  
  
report_file_size  
=  
dest  
.  
ensure_space  
):  
  
dest  
.  
discard  
()  
  
else  
:  
  
format_metadata  
[  
book_id  
][  
'cover'  
]  
  
=  
  
cover_key  
  
exporter  
.  
set_metadata  
(  
library_key  
,  
  
metadata  
)
```

if

progress

is

not

None

:

progress

(

—
(

'Completed'

),

total

,

total

)

@read_api

def

annotations_map_for_book

(

self

,

book_id

,

fmt

,


```
user_type
=
'local'
,

user
=
'viewer'
):

ans

=

{}

for

annot

in

self
.
backend
.
annotations_for_book
(
book_id
,

fmt
,

user_type
,

user
):
```

```
ans
.  
setdefault  
(  
annot  
[  
'type'  
],  
  
[])  
.  
append  
(  
annot  
)
```

```
return
```

```
ans
```

```
@read_api
```

```
def
```

```
all_annotations_for_book  
(  
self  
,  
  
book_id  
):
```

```
return
```

```
tuple  
(  
self  
.  

```

```
backend
.  
all_annotations_for_book  
(  
book_id  
))
```

```
@read_api
```

```
def
```

```
annotation_count_for_book  
(  
self  
,  
  
book_id  
):
```

```
return
```

```
self  
.  
backend  
.  
annotation_count_for_book  
(  
book_id  
)
```

```
@read_api
```

```
def
```

```
all_annotation_users  
(
```

```
self
):

return

tuple
(
self
.
backend
.
all_annotation_users
())
```

@read_api

```
def

all_annotation_types
(
self
):

return

tuple
(
self
.
backend
.
all_annotation_types
())
```

@read_api

```
def
all_annotations
(
self
,
restrict_to_user
=
None
,
limit
=
None
,
annotation_type
=
None
,
ignore_removed
=
False
,
restrict_to_book_ids
=
None
):

return

tuple
(
self
.
backend
.
all_annotations
```

```
(
    restrict_to_user
    ,

    limit
    ,

    annotation_type
    ,

    ignore_removed
    ,

    restrict_to_book_ids
))
```

@read_api

def

search_annotations

(

self

,

fts_engine_query

,

use_stemming

=

True

,

highlight_start

=

```
None
,

highlight_end
=
None
,

snippet_size
=
None
,

annotation_type
=
None
,

restrict_to_book_ids
=
None
,

restrict_to_user
=
None
,

ignore_removed
=
False

):

return
```

```
tuple
(
  self
  .
  backend
  .
  search_annotations
  (
    fts_engine_query
    ,
    use_stemming
    ,
    highlight_start
    ,
    highlight_end
    ,
    snippet_size
    ,
    annotation_type
    ,
    restrict_to_book_ids
    ,
    restrict_to_user
    ,
    ignore_removed
  )
))
```



```
@write_api
```

```
def
```

```
delete_annotations
```

```
(  
self
```

```
,
```

```
annot_ids
```

```
):
```

```
self
```

```
.
```

```
backend
```

```
.
```

```
delete_annotations
```

```
(
```

```
annot_ids
```

```
)
```

```
@write_api
```

```
def
```

```
update_annotations
```

```
(
```

```
self
```

```
,
```

```
annot_id_map
```

```
):
```

```
self
```

```
.
```

```
backend
```

```
.  
update_annotations  
(  
    annot_id_map  
)
```

```
@write_api
```

```
def
```

```
restore_annotations  
(  
    self  
,  
  
    book_id  
,  
  
    annotations  
):
```

```
from
```

```
calibre.utils.date
```

```
import
```

```
EPOCH
```

```
from
```

```
calibre.utils.iso8601
```

```
import
```

```
parse_iso8601
```

```
umap

=

defaultdict
(
list
)

for

adata

in

annotations
:

key

=

adata
[
'user_type'
],

adata
[
'user'
],

adata
[
'format'
]

a

=
```

```
adata
[
    'annotation'
]

ts

=

(
    parse_iso8601
    (
        a
        [
            'timestamp'
        ])
    -
    EPOCH
)
.
total_seconds
()

umap
[
    key
]
.
append
((
    a
    ,
    ts
))

for
```

```
(
    user_type
    ,
    user
    ,
    fmt
),

annots_list

in

iteritems
(
    umap
):

    self
    .
    _set_annotations_for_book
    (
        book_id
        ,
        fmt
        ,
        annots_list
        ,
        user_type
        =
        user_type
        ,
        user
        =
        user
    )
```

```
@write_api
```

```
def
```

```
set_annotations_for_book
```

```
(  
    self
```

```
,
```

```
    book_id
```

```
,
```

```
    fmt
```

```
,
```

```
    annots_list
```

```
,
```

```
    user_type
```

```
=
```

```
    'local'
```

```
,
```

```
    user
```

```
=
```

```
    'viewer'
```

```
):
```

```
    self
```

```
.
```

```
    backend
```

```
.
```

```
    set_annotations_for_book
```

```
(
```

```
    book_id
```

```
,
```

```
    fmt
```

```
,  
  
annots_list  
,  
  
user_type  
,  
  
user  
)
```

@write_api

```
def  
  
merge_annotations_for_book  
(  
    self  
,  
  
    book_id  
,  
  
    fmt  
,  
  
    annots_list  
,  
  
    user_type  
    =  
    'local'  
,  
  
    user  
    =  
    'viewer'  
):
```

```
from
```

```
calibre.utils.date
```

```
import
```

```
EPOCH
```

```
from
```

```
calibre.utils.iso8601
```

```
import
```

```
parse_iso8601
```

```
amap
```

```
=
```

```
self
```

```
.
```

```
_annotations_map_for_book
```

```
(
```

```
book_id
```

```
,
```

```
fmt
```

```
,
```

```
user_type
```

```
=
```

```
user_type
```

```
,
```

```
user
```

```
=
```

```
user
```

```
)
```



```
merge_annotations
```

```
(
```

```
  annots_list
```

```
,
```

```
  amap
```

```
)
```

```
alist
```

```
=
```

```
[]
```

```
for
```

```
val
```

```
in
```

```
  itervalues
```

```
  (
```

```
    amap
```

```
  ):
```

```
for
```

```
annot
```

```
in
```

```
val
```

```
:
```

```
ts
```

```
=
```

```
(
```

```
parse_iso8601
(
annot
[
'timestamp'
])

-

EPOCH
)

.
total_seconds
()

alist
.
append
((
annot
,

ts
))

self
.
_set_annotations_for_book
(
book_id
,

fmt
,

alist
,

user_type
=
user_type
```

```
,  
  
user  
=  
user  
)  
  
@write_api  
  
def  
  
reindex_annotations  
(  
    self  
):  
  
    self  
    .  
    backend  
    .  
    reindex_annotations  
    ()  
  
def  
  
import_library  
(  
    library_key  
    ,  
  
    importer  
    ,  
  
    library_path  
    ,  
  
    progress  
    =
```

None

,

abort

=

None

):

from

calibre.db.backend

import

DB

metadata

=

importer

.

metadata

[

library_key

]

total

=

metadata

[

'total'

]

if

progress

is

not

None

:

progress

(

'metadata.db'

,

0

,

total

)

if

abort

is

not

None

and

abort

.

is_set

():

return

with

```
open
(
os
.
path
.
join
(
library_path
,

'metadata.db'
),

'wb'
)

as

f
:

src

=

importer
.
start_file
(
metadata
[
'metadata.db'
],

'metadata.db for '

+

library_path
)
```

```
shutil
```

```
.
```

```
copyfileobj
```

```
(
```

```
src
```

```
,
```

```
f
```

```
)
```

```
src
```

```
.
```

```
close
```

```
()
```

```
cache
```

```
=
```

```
Cache
```

```
(
```

```
DB
```

```
(
```

```
library_path
```

```
,
```

```
load_user_formatter_functions
```

```
=
```

```
False
```

```
))
```

```
cache
```

```
.
```

```
init
```

```
()
```

```
format_data
```

```
=  
  
{  
int  
(  
book_id  
):  
data  
  
for  
  
book_id  
,  
  
data  
  
in  
  
iteritems  
(  
metadata  
[  
'format_data'  
])}  
  
for  
  
i  
,  
  
(  
book_id  
,  
  
fmt_key_map  
)  
  
in  
  
enumerate  
(
```



```
iteritems
(
format_data
)):
```

```
if
```

```
abort
```

```
is
```

```
not
```

```
None
```

```
and
```

```
abort
```

```
.
```

```
is_set
```

```
():
```

```
return
```

```
title
```

```
=
```

```
cache
```

```
.
```

```
_field_for
```

```
(
```

```
'title'
```

```
,
```

```
book_id
```

```
)
```

```
if
```

```
progress
```

```
is
```

```
not
```

```
None
```

```
:
```

```
progress
```

```
(
```

```
title
```

```
,
```

```
i
```

```
+
```

```
1
```

```
,
```

```
total
```

```
)
```

```
cache
```

```
.
```

```
_update_path
```

```
((
```

```
book_id
```

```
),
```

```
mark_as_dirtied
```

```
=
```

```
False
```

```
)
```

```
for
```

```
fmt
```

```
,  
  
fmtkey  
  
in  
  
iteritems  
(  
    fmt_key_map  
):
```

```
if  
  
    fmt  
  
    ==  
  
    '.cover'  
    :
```

```
stream  
  
=  
  
importer  
.  
start_file  
(  
    fmtkey  
    ,
```

```
—  
(  
    'Cover for  
    %s  
    '  
)  
  
%
```

```
title
```

```
)

path

=

cache

.
_field_for
(
'path'
,

book_id
)

.
replace
(
'/'
,

os

.
sep
)

cache

.
backend

.
set_cover
(
book_id
,

path
,

stream
,
```

```
no_processing
```

```
=
```

```
True
```

```
)
```

```
else
```

```
:
```

```
stream
```

```
=
```

```
importer
```

```
.
```

```
start_file
```

```
(
```

```
fmtkey
```

```
,
```

```
_
```

```
(
```

```
'
```

```
{0}
```

```
format for
```

```
{1}
```

```
'
```

```
)
```

```
.
```

```
format
```

```
(
```

```
fmt
```

```
.
```

```
upper
```

```
()),
```

```
title
```

```
))
```

```
size
```

```
,
```

```
fname
=
cache
.
  _do_add_format
  (
    book_id
    ,

    fmt
    ,

    stream
    ,

    mtime
    =
    stream
    .
    mtime
  )

cache
.
  fields
  [
    'formats'
  ]
.
  table
.
  update_fmt
  (
    book_id
    ,

    fmt
    ,
```

```
fname
,

size
,

cache
.
backend
)

stream
.
close
()

cache
.
dump_metadata
({
book_id
})

if

progress

is

not

None
:

progress
(
_
(
'Completed'
```

```
),  
total  
,  
total  
)  
  
return  
  
cache  
  
# }}}
```


Source code for calibre.devices.interface

```
__license__  
  
=  
'GPL v3'  
  
__copyright__  
  
=  
'2008, Kovid Goyal <kovid at kovidgoyal.net>'  
  
import  
  
os  
  
from  
  
collections  
  
import  
  
namedtuple  
  
  
  
from  
  
calibre  
  
import  
  
prints  
  
from
```

```
calibre.constants
```

```
import
```

```
iswindows
```

```
from
```

```
calibre.customize
```

```
import
```

```
Plugin
```

```
\[docs\]
```

```
class
```

```
DevicePlugin
```

```
(  
    Plugin  
):
```

```
"""
```

```
Defines the interface that should be implemented by backends that  
communicate with an e-book reader.
```

```
"""
```

```
type
```

```
=
```

```
-
```

```
(  
    'Device interface'
```

)

#: Ordered list of supported formats

FORMATS

=

```
[  
"lrf"  
,  
"rtf"  
,  
"pdf"  
,  
"txt"  
]
```

If True, the config dialog will not show the formats box

HIDE_FORMATS_CONFIG_BOX

=

False

#: VENDOR_ID can be either an integer, a list of integers or a dictionary

#: If it is a dictionary, it must be a dictionary of dictionaries,

#: of the form::

#:

#: {

#: integer_vendor_id : { product_id : [list of BCDs], ... },

#: ...

#: }

#:

VENDOR_ID

=

0x0000

#: An integer or a list of integers

PRODUCT_ID

=

0x0000

#: BCD can be either None to not distinguish between devices based on BCD, or

#: it can be a list of the BCD numbers of all devices supported by this driver.

BCD

=

None

#: Height for thumbnails on the device

THUMBNAIL_HEIGHT

=

68

#: Compression quality for thumbnails. Set this closer to 100 to have better

#: quality thumbnails with fewer compression artifacts. Of course, the

#: thumbnails get larger as well.

THUMBNAIL_COMPRESSION_QUALITY

=

75

#: Set this to True if the device supports updating cover thumbnails during

#: sync_booklists. Setting it to true will ask device.py to refresh the

#: cover thumbnails during book matching

WANTS_UPDATED_THUMBNAILS

=

False

#: Whether the metadata on books can be set via the GUI.

CAN_SET_METADATA

=

```
[  
'title'  
,  
'authors'  
,  
'collections'  
]
```

#: Whether the device can handle device_db metadata plugboards

CAN_DO_DEVICE_DB_PLUGBOARD

=

False

Set this to None if the books on the device are files that the GUI can

access in order to add the books from the device to the library

BACKLOADING_ERROR_MESSAGE

=

—
(

'Cannot get files from this device'

)

#: Path separator for paths to books on device

path_sep

=

os

.

sep

#: Icon for this device

icon

=

|

(

'reader.png'

)

Encapsulates an annotation fetched from the device

UserAnnotation

=

namedtuple

(

'Annotation'

,

'type, value'

)

#: GUI displays this as a message if not None. Useful if opening can take a

#: long time

OPEN_FEEDBACK_MESSAGE

=

None

#: Set of extensions that are "virtual books" on the device

#: and therefore cannot be viewed/saved/added to library.

#: For example: ``frozenset(['kobo'])``

VIRTUAL_BOOK_EXTENSIONS

=

frozenset

()

#: Message to display to user for virtual book extensions.

VIRTUAL_BOOK_EXTENSION_MESSAGE

=

None

#: Whether to nuke comments in the copy of the book sent to the device. If

#: not None this should be short string that the comments will be replaced

#: by.

NUKE_COMMENTS

=

None

#: If True indicates that this driver completely manages device detection,

*#: ejecting and so forth. If you set this to True, you **must** implement the*

#: detect_managed_devices and debug_managed_device_detection methods.

#: A driver with this set to true is responsible for detection of devices,

#: managing a blacklist of devices, a list of ejected devices and so forth.

#: calibre will periodically call the detect_managed_devices() method and

#: if it returns a detected device, calibre will call open(). open() will

#: be called every time a device is returned even if previous calls to open()

#: failed, therefore the driver must maintain its own blacklist of failed

#: devices. Similarly, when ejecting, calibre will call eject() and then

#: assuming the next call to detect_managed_devices() returns None, it will

#: call post_yank_cleanup().

MANAGES_DEVICE_PRESENCE

=

False

#: If set the True, calibre will call the :meth:`get\_driveinfo()` method

#: after the books lists have been loaded to get the driveinfo.

SLOW_DRIVEINFO

=

False

#: If set to True, calibre will ask the user if they want to manage the

#: device with calibre, the first time it is detected. If you set this to

#: True you must implement :meth:`get\_device\_uid()` and

#: :meth:`ignore\_connected\_device()` and

#: :meth:`get\_user\_blacklisted\_devices` and

#: :meth:`set\_user\_blacklisted\_devices`

ASK_TO_ALLOW_CONNECT

=

False

#: Set this to a dictionary of the form {'title':title, 'msg':msg, 'det_msg':detailed_msg} to have calibre popup

#: a message to the user after some callbacks are run (currently only upload_books).

*#: Be careful to not spam the user with too many messages. This variable is checked after *every* callback,*

#: so only set it when you really need to.

```
user_feedback_after_callback
```

```
=
```

```
None
```

```
@classmethod
```

```
def
```

```
get_gui_name
```

```
(
```

```
cls
```

```
):
```

```
if
```

```
hasattr
```

```
(
```

```
cls
```

```
,
```

```
'gui_name'
```

```
):
```

```
return
```

```
cls
```

```
.
```

```
gui_name
```

```
if
```

```
hasattr
```

```
(
```

```
cls
```

```
,  
  
'__name__'  
):
```

```
return
```

```
cls  
.  
__name__
```

```
return
```

```
cls  
.  
name
```

```
# Device detection {{{
```

```
def
```

```
test_bcd  
(  
self  
,  
  
bcdDevice  
,  
  
bcd  
):
```

```
if
```

```
bcd
```

```
is
```

None

or

len
(
bcd
)

==

0
:

return

True

for

c

in

bcd
:

if

c

==

bcdDevice
:

return

True

return

False

[\[docs\]](#)

def

is_usb_connected

(

self

,

devices_on_system

,

debug

=

False

,

only_presence

=

False

):

'''

Return True, device_info if a device handled by this plugin is currently connected.

:param devices_on_system: List of devices currently connected

'''

```
vendors_on_system
```

```
=
```

```
{
```

```
x
```

```
[
```

```
0
```

```
]
```

```
for
```

```
x
```

```
in
```

```
devices_on_system
```

```
}
```

```
vendors
```

```
=
```

```
set
```

```
(
```

```
self
```

```
.
```

```
VENDOR_ID
```

```
)
```

```
if
```

```
hasattr
```

```
(
```

```
self
```

```
.
```

```
VENDOR_ID
```

```
,
```

```
'__len__'
```

```
)
```



```
else
```

```
{
```

```
self
```

```
.
```

```
VENDOR_ID
```

```
}
```

```
if
```

```
hasattr
```

```
(
```

```
self
```

```
.
```

```
VENDOR_ID
```

```
,
```

```
'keys'
```

```
):
```

```
products
```

```
=
```

```
[]
```

```
for
```

```
ven
```

```
in
```

```
self
```

```
.
```

```
VENDOR_ID
```

```
:
```

```
products
```

```
.  
extend  
(  
self  
.  
VENDOR_ID  
[  
ven  
]  
.  
keys  
())
```

```
else  
:
```

```
products
```

```
=
```

```
self
```

```
.  
PRODUCT_ID
```

```
if
```

```
hasattr
```

```
(  
self
```

```
.  
PRODUCT_ID
```

```
,
```

```
'__len__'
```

```
)
```

```
else
```

```
[  
self
```

```
.
```

```
PRODUCT_ID  
]
```

```
ch
```

```
=
```

```
self
```

```
.
```

```
can_handle_windows
```

```
if
```

```
iswindows
```

```
else
```

```
self
```

```
.
```

```
can_handle
```

```
for
```

```
vid
```

```
in
```

```
vendors_on_system
```

```
.
```

```
intersection
```

```
(
```

```
vendors
```

```
):
```

```
for
```

```
dev
```

```
in
```

```
devices_on_system
```

```
:
```

```
  cvid
```

```
,
```

```
  pid
```

```
,
```

```
  bcd
```

```
=
```

```
  dev
```

```
[:
```

```
3
```

```
]
```

```
if
```

```
  cvid
```

```
==
```

```
  vid
```

```
:
```

```
if
```

```
  pid
```

```
in
```

```
  products
```

```
:
```

```
if
```

```
hasattr
(
self
.
VENDOR_ID
,

'keys'
):

try
:

cbcd

=

self
.
VENDOR_ID
[
vid
][
pid
]

except

KeyError
:

# Vendor vid does not have product pid, pid

# exists for some other vendor in this

# device
```

continue

else

:

cbcd

=

self

.

BCD

if

self

.

test_bcd

(

bcd

,

cbcd

):

if

debug

:

prints

(

dev

)

if

```
ch
(  
dev
```

```
,
```

```
debug  
=  
debug  
):
```

```
return
```

```
True
```

```
,
```

```
dev
```

```
return
```

```
False
```

```
,
```

```
None
```

```
\[docs\]
```

```
def
```

```
detect_managed_devices
```

```
(
```

```
self
```

```
,
```

```
devices_on_system
```

```
,
```

```
force_refresh
```

```
=
```

```
False
```

```
);
```

```
'''
```

Called only if MANAGES_DEVICE_PRESENCE is True.

Scan for devices that this driver can handle. Should return a device object if a device is found. This object will be passed to the open() method as the connected_device. If no device is found, return None. The returned object can be anything, calibre does not use it, it is only passed to open().

This method is called periodically by the GUI, so make sure it is not too resource intensive. Use a cache to avoid repeatedly scanning the system.

:param devices_on_system: Set of USB devices found on the system.

:param force_refresh: If True and the driver uses a cache to prevent repeated scanning, the cache must be flushed.

```
'''
```

```
raise
```

```
NotImplementedError
```

```
()
```


[\[docs\]](#)

def

debug_managed_device_detection

(
self

,

devices_on_system

,

output

):

'''

Called only if MANAGES_DEVICE_PRESENCE is True.

Should write information about the devices detected on the system to

output, which is a file like object.

Should return True if a device was detected and successfully opened,

otherwise False.

'''

raise

NotImplementedError

()

}}}}

[\[docs\]](#)

def

reset

(

self

,

key

=

'-1'

,

log_packets

=

False

,

report_progress

=

None

,

detected_device

=

None

):

"""

:param key: The key to unlock the device

:param log_packets: If true the packet stream to/from the device is logged

:param report_progress: Function that is called with a % progress

(number between 0 and 100) for various tasks.

If it is called with -1 that means that the

task does not have any progress information

:param detected_device: Device information from the device scanner

"""

raise

NotImplementedError

()

[\[docs\]](#)

def

can_handle_windows

(

self

,

usbdevice

,

debug

=

False

):

"""

Optional method to perform further checks on a device to see if this driver

is capable of handling it. If it is not it should return False. This method

is only called after the vendor, product ids and the bcd have matched, so

it can do some relatively time intensive checks. The default implementation

returns True. This method is called only on Windows. See also

:meth:`can\_handle`.

Note that for devices based on USBMS this method by default delegates

to :meth:`can\_handle`. So you only need to override :meth:`can\_handle`

in your subclass of USBMS.

*:param usbdevice: A usbdevice as returned by
:func:`calibre.devices.winusb.scan\_usb\_devices`*

'''

return

True

[\[docs\]](#)

def

can_handle

(

self

,

device_info

,

debug

=

False

):

'''

Unix version of :meth:`can\_handle\_windows`.

```
:param device_info: Is a tuple of (vid, pid, bcd, manufacturer, product,  
serial number)
```

```
'''
```

```
return
```

```
True
```

```
can_handle
```

```
.
```

```
is_base_class_implementation
```

```
=
```

```
True
```

```
\[docs\]
```

```
def
```

```
open
```

```
(
```

```
self
```

```
,
```

```
connected_device
```

```
,
```

```
library_uuid
```

```
):
```

'''

Perform any device specific initialization. Called after the device is detected but before any other functions that communicate with the device.

For example: For devices that present themselves as USB Mass storage devices, this method would be responsible for mounting the device or if the device has been automounted, for finding out where it has been mounted. The method `:meth:`calibre.devices.usbms.device.Device.open`` has an implementation of this function that should serve as a good example for USB Mass storage devices.

This method can raise an `OpenFeedback` exception to display a message to the user.

:param `connected_device`: The device that we are trying to open. It is a tuple of (vendor id, product id, bcd, manufacturer name, product name, device serial number). However, some devices have no serial number and on Windows only the first three fields are present, the rest are `None`.

:param `library_uuid`: The UUID of the current calibre library. Can be `None` if there is no library (for example when used from the command line).

```
'''
```

```
raise
```

```
NotImplementedError  
(
```

```
\[docs\]
```

```
def
```

```
eject  
(  
    self  
):
```

```
'''
```

*Un-mount / eject the device from the OS. This does not check if there
are pending GUI jobs that need to communicate with the device.*

*NOTE: That this method may not be called on the same thread as the rest
of the device methods.*

```
'''
```

```
raise
```

```
NotImplementedError  
(
```

```
\[docs\]
```

```
def
```

```

post_yank_cleanup
(
    self
):

'''

    Called if the user yanks the device without ejecting it first.

'''

raise
NotImplementedError
()

\[docs\]

def
set_progress_reporter
(
    self
,
report_progress
):

'''

    Set a function to report progress information.

    :param report_progress: Function that is called with a % progress
                            (number between 0 and 100) for various tasks.

                            If it is called with -1 that means that the

```


task does not have any progress information

'''

raise

NotImplementedError

()

[\[docs\]](#)

def

get_device_information

(

self

,

end_session

=

True

):

''''

Ask device for device information. See L{DeviceInfoQuery}.

:return: (device name, device version, software version on device, MIME type)

The tuple can optionally have a fifth element, which is a

drive information dictionary. See usbms.driver for an example.

''''

raise

NotImplementedError

()

[\[docs\]](#)

def

get_driveinfo

(

self

):

'''

Return the driveinfo dictionary. Usually called from

get_device_information(), but if loading the driveinfo is slow for this

driver, then it should set SLOW_DRIVEINFO. In this case, this method

will be called by calibre after the book lists have been loaded. Note

that it is not called on the device thread, so the driver should cache

the drive info in the books() method and this function should return

the cached data.

'''

return

{}

[\[docs\]](#)

def

```
card_prefix
(
self
,

end_session
=
True
):

'''
    Return a 2 element list of the prefix to paths on the cards.

    If no card is present None is set for the card's prefix.

    E.G.

    ('/place', '/place2')

    (None, 'place2')

    ('place', None)

    (None, None)

'''

raise

NotImplementedError
()

\[docs\]

def

total_space
(
```

```
self
',

end_session
=
True
):

"""

    Get total space available on the mountpoints:

    1. Main memory

    2. Memory Card A

    3. Memory Card B

    :return: A 3 element list with total space in bytes of (1, 2, 3). If a
           particular device doesn't have any of these locations it should return 0.

    """

raise

NotImplementedError
()

\[docs\]

def

free_space
(
self
,
```

```
end_session
```

```
=
```

```
True
```

```
):
```

```
"""
```

Get free space available on the mountpoints:

1. Main memory

2. Card A

3. Card B

:return: A 3 element list with free space in bytes of (1, 2, 3). If a

particular device doesn't have any of these locations it should return -1.

```
"""
```

```
raise
```

```
NotImplementedError
```

```
()
```

```
\[docs\]
```

```
def
```

```
books
```

```
(
```

```
self
```

```
,
```

```
oncard
```

```
=
```

```
None
```

```

',

end_session
=
True
):

"""

    Return a list of e-books on the device.

    :param oncard: If 'carda' or 'cardb' return a list of e-books on the
                    specific storage card, otherwise return list of e-books
                    in main memory of device. If a card is specified and no
                    books are on the card return empty list.

    :return: A BookList.

    """

raise
NotImplementedError
()

\[docs\]

def
upload_books
(
self
,

```

```

files
,

names
,

on_card
=
None
,

end_session
=
True
,

metadata
=
None
):

'''

    Upload a list of books to the device. If a file already
    exists on the device, it should be replaced.

    This method should raise a :class:`FreeSpaceError` if there is not enough
    free space on the device. The text of the FreeSpaceError must contain the
    word "card" if ``on_card`` is not None otherwise it must contain the word "memory".

    :param files: A list of paths

    :param names: A list of file names that the books should have
                   once uploaded to the device. len(names) == len(files)

    :param metadata: If not None, it is a list of :class:`Metadata` objects.

```

The idea is to use the metadata to determine where on the device to put the book. `len(metadata) == len(files)`. Apart from the regular cover (path to cover), there may also be a thumbnail attribute, which should be used in preference. The thumbnail attribute is of the form (width, height, cover_data as jpeg).

:return: A list of 3-element tuples. The list is meant to be passed

to :meth:`add\_books\_to\_metadata`.

'''

raise

NotImplementedError

()

[\[docs\]](#)

@classmethod

def

add_books_to_metadata

(

cls

,

locations

,

metadata

,


```
booklists
```

```
):
```

```
'''
```

```
Add locations to the booklists. This function must not communicate with  
the device.
```

```
:param locations: Result of a call to L{upload_books}
```

```
:param metadata: List of :class:`Metadata` objects, same as for
```

```
    :meth:`upload_books`.
```

```
:param booklists: A tuple containing the result of calls to
```

```
    (:meth:`books(ocard=None)`,
```

```
    :meth:`books(ocard='carda')`,
```

```
    :meth:`books(ocard='cardb')`).
```

```
'''
```

```
raise
```

```
NotImplementedError
```

```
()
```

```
\[docs\]
```

```
def
```

```
delete_books
```

```
(
```

```
self
```

```
,
```

```
paths
,

end_session
=
True
):

'''

    Delete books at paths on device.

'''

raise

NotImplementedError
()

\[docs\]

@classmethod

def

remove_books_from_metadata
(
    cls
,

paths
,

booklists
):

'''
```

Remove books from the metadata list. This function must not communicate with the device.

:param paths: paths to books on the device.

:param booklists: A tuple containing the result of calls to

(:meth:`books(ocard=None)`,

:meth:`books(ocard='carda')`,

:meth:`books(ocard='cardb')`).

'''

raise

NotImplementedError

()

[\[docs\]](#)

def

sync_booklists

(

self

,

booklists

,

end_session

=

True

):

```
'''
```

```
    Update metadata on device.
```

```
    :param booklists: A tuple containing the result of calls to
```

```
        (:meth:`books(oncard=None)`,
```

```
        :meth:`books(oncard='carda')`,
```

```
        :meth:`books(oncard='cardb')`).
```

```
'''
```

```
raise
```

```
NotImplementedError
```

```
()
```

```
\[docs\]
```

```
def
```

```
    get_file
```

```
    (
```

```
    self
```

```
,
```

```
    path
```

```
,
```

```
    outfile
```

```
,
```

```
    end_session
```

```
=
```

```
    True
```

```
):
```

'''

Read the file at ``path`` on the device and write it to outfile.

:param outfile: file object like ``sys.stdout`` or the result of an

:func: ``open`` call.

'''

raise

NotImplementedError

()

[\[docs\]](#)

@classmethod

def

config_widget

(

cls

):

'''

Should return a QWidget. The QWidget contains the settings for the

device interface

'''

raise

NotImplementedError

()

[\[docs\]](#)

@classmethod

def

save_settings

(

cls

,

settings_widget

):

'''

Should save settings to disk. Takes the widget created in

:meth:`config\_widget` and saves all settings to disk.

'''

raise

NotImplementedError

()

[\[docs\]](#)

@classmethod

def

```
settings
```

```
(
```

```
cls
```

```
):
```

```
'''
```

Should return an opts object. The opts object should have at least one attribute `format\_map` which is an ordered list of formats for the device.

```
'''
```

```
raise
```

```
NotImplementedError
```

```
()
```

```
\[docs\]
```

```
def
```

```
set_plugboards
```

```
(
```

```
self
```

```
,
```

```
plugboards
```

```
,
```

```
pb_func
```

```
):
```

```
'''
```

provide the driver the current set of plugboards and a function to

*select a specific plugboard. This method is called immediately before
add_books and sync_booklists.*

pb_func is a callable with the following signature::

```
def pb_func(device_name, format, plugboards)
```

*You give it the current device name (either the class name or
DEVICE_PLUGBOARD_NAME), the format you are interested in (a 'real'
format or 'device_db'), and the plugboards (you were given those by
set_plugboards, the same place you got this method).*

:return: None or a single plugboard instance.

'''

pass

[\[docs\]](#)

def

set_driveinfo_name

(

self

,

location_code

,

name

):


```
'''
```

Set the device name in the driveinfo file to 'name'. This setting will persist until the file is re-created or the name is changed again.

Non-disk devices should implement this method based on the location codes returned by the get_device_information() method.

```
'''
```

```
pass
```

```
\[docs\]
```

```
def
```

```
    prepare_addable_books
```

```
    (
```

```
    self
```

```
,
```

```
    paths
```

```
):
```

```
'''
```

Given a list of paths, returns another list of paths. These paths point to addable versions of the books.

If there is an error preparing a book, then instead of a path, the position in the returned list for that book should be a three tuple:

(original_path, the exception instance, traceback)

'''

return

paths

[\[docs\]](#)

def

startup

(
self
):

'''

Called when calibre is starting the device. Do any initialization

required. Note that multiple instances of the class can be instantiated,

and thus `__init__` can be called multiple times, but only one instance

will have this method called. This method is called on the device

thread, not the GUI thread.

'''

pass

[\[docs\]](#)

def

shutdown

```
(
self
):

'''

    Called when calibre is shutting down, either for good or in preparation
    to restart. Do any cleanup required. This method is called on the
    device thread, not the GUI thread.

'''

pass

[docs]
def
get_device_uid
(
self
):

'''

    Must return a unique id for the currently connected device (this is
    called immediately after a successful call to open()). You must
    implement this method if you set ASK_TO_ALLOW_CONNECT = True

'''

raise
NotImplementedError
```

```
()
```

[\[docs\]](#)

```
def
```

```
ignore_connected_device
```

```
(  
self
```

```
,
```

```
uid
```

```
):
```

```
'''
```

*Should ignore the device identified by uid (the result of a call to
get_device_uid()) in the future. You must implement this method if you
set ASK_TO_ALLOW_CONNECT = True. Note that this function is called
immediately after open(), so if open() caches some state, the driver
should reset that state.*

```
'''
```

```
raise
```

```
NotImplementedError
```

```
()
```

[\[docs\]](#)

```
def
```

```
get_user_blacklisted_devices
```

```
(
```

```
self
):

'''
    Return map of device uid to friendly name for all devices that the user
    has asked to be ignored.
'''
```

```
return
```

```
{}
```

```
\[docs\]
```

```
def
```

```
set_user_blacklisted_devices
```

```
(
    self
```

```
,
```

```
devices
```

```
):
```

```
'''
```

```
    Set the list of device uids that should be ignored by this driver.
```

```
'''
```

```
pass
```

```
\[docs\]
```

```
def
```

```
specialize_global_preferences
```

```
(
```

```
self
```

```
,
```

```
device_prefs
```

```
):
```

```
'''
```

Implement this method if your device wants to override a particular preference. You must ensure that all call sites that want a preference that can be overridden use device_prefs['something'] instead of prefs['something']. Your method should call device_prefs.set_overrides(pref=val, pref=val, ...).

Currently used for:

metadata management (prefs['manage_device_metadata'])

```
'''
```

```
device_prefs
```

```
.
```

```
set_overrides
```

```
()
```

```
\[docs\]
```

```
def
```

```
set_library_info
```

```
(
```

```
self
```

```
,  
library_name  
,  
library_uuid  
,  
field_metadata  
):  
  
'''  
  
    Implement this method if you want information about the current calibre  
  
    library. This method is called at startup and when the calibre library  
  
    changes while connected.  
  
    '''  
  
pass  
  
# Dynamic control interface.  
  
# The following methods are probably called on the GUI thread. Any driver  
  
# that implements these methods must take pains to be thread safe, because  
  
# the device_manager might be using the driver at the same time that one of  
  
# these methods is called.
```

[\[docs\]](#)

def

is_dynamically_controllable

(
self
):

'''

*Called by the device manager when starting plugins. If this method returns
a string, then a) it supports the device manager's dynamic control
interface, and b) that name is to be used when talking to the plugin.*

*This method can be called on the GUI thread. A driver that implements
this method must be thread safe.*

'''

return

None

[\[docs\]](#)

def

start_plugin

(
self
):

'''

This method is called to start the plugin. The plugin should begin to accept device connections however it does that. If the plugin is already accepting connections, then do nothing.

This method can be called on the GUI thread. A driver that implements this method must be thread safe.

'''

pass

[\[docs\]](#)

def

stop_plugin
(
self
):

'''

This method is called to stop the plugin. The plugin should no longer accept connections, and should cleanup behind itself. It is likely that this method should call shutdown. If the plugin is already not accepting connections, then do nothing.

This method can be called on the GUI thread. A driver that implements this method must be thread safe.

'''

```
pass
```

```
\[docs\]
```

```
def
```

```
get_option
```

```
(  
    self  
    ,
```

```
    opt_string  
    ,
```

```
    default
```

```
    =
```

```
    None
```

```
):
```

```
    """
```

Return the value of the option indicated by opt_string. This method can be called when the plugin is not started. Return None if the option does not exist.

This method can be called on the GUI thread. A driver that implements this method must be thread safe.

```
    """
```

```
return
```

```
default
```

[\[docs\]](#)

def

set_option

(
self

,

opt_string

,

opt_value

):

"""

*Set the value of the option indicated by opt_string. This method can
be called when the plugin is not started.*

*This method can be called on the GUI thread. A driver that implements
this method must be thread safe.*

"""

pass

[\[docs\]](#)

def

is_running

(

self

):

```
"""
    Return True if the plugin is started, otherwise false

    This method can be called on the GUI thread. A driver that implements
    this method must be thread safe.
    """

return

False

\[docs\]

def
synchronize_with_db
(
    self
    ,
    db
    ,
    book_id
    ,
    book_metadata
    ,
    first_call
):
    """
        Called during book matching when a book on the device is matched with
```

a book in calibre's db. The method is responsible for synchronizing data from the device to calibre's db (if needed).

The method must return a two-value tuple. The first value is a set of calibre book ids changed if calibre's database was changed or None if the database was not changed. If the first value is an empty set then the metadata for the book on the device is updated with calibre's metadata and given back to the device, but no GUI refresh of that book is done. This is useful when the calibre data is correct but must be sent to the device.

The second value is itself a 2-value tuple. The first value in the tuple specifies whether a book format should be sent to the device. The intent is to permit verifying that the book on the device is the same as the book in calibre. This value must be None if no book is to be sent, otherwise return the base file name on the device (a string like foobar.epub). Be sure to include the extension in the name. The device subsystem will construct a send_books job for all books with not- None returned values. Note: other than to later retrieve the extension, the name is ignored in cases where the device uses a template to generate the file name, which most do. The second value in the returned tuple indicated whether the format is future-dated. Return True if it is, otherwise return False. calibre will display a dialog to the user

listing all future dated books.

*Extremely important: this method is called on the GUI thread. It must
be threadsafe with respect to the device manager's thread.*

book_id: the calibre id for the book in the database.

book_metadata: the Metadata object for the book coming from the device.

first_call: True if this is the first call during a sync, False otherwise

'''

return

(
None
,

(
None
,

False
)

[\[docs\]](#)

class

BookList

(
list
):

'''

A list of books. Each Book object must have the fields

#. title

#. authors

#. size (file size of the book)

#. datetime (a UTC time tuple)

#. path (path on the device to the book)

#. thumbnail (can be None) thumbnail is either a str/bytes object with the

image data or it should have an attribute image_path that stores an

absolute (platform native) path to the image

#. tags (a list of strings, can be empty).

'''

__getslice__

=

None

__setslice__

=

None

def

```
__init__
```

```
(  
self
```

```
,
```

```
oncard
```

```
,
```

```
prefix
```

```
,
```

```
settings
```

```
):
```

```
pass
```

```
\[docs\]
```

```
def
```

```
supports_collections
```

```
(
```

```
self
```

```
):
```

```
""" Return True if the device supports collections for this book list. """
```

```
raise
```

```
NotImplementedError
```

```
()
```

```
\[docs\]
```

```
def
```



```
add_book
(
self
,
book
,
replace_metadata
):

'''
    Add the book to the booklist. Intent is to maintain any device-internal
    metadata. Return True if booklists must be sync'ed
'''

raise
NotImplementedError
()

\[docs\]

def
remove_book
(
self
,
book
):

'''
    Remove a book from the booklist. Correct any device metadata at the
```

same time

'''

raise

NotImplementedError

()

[\[docs\]](#)

def

get_collections

(

self

,

collection_attributes

):

'''

Return a dictionary of collections created from collection_attributes.

Each entry in the dictionary is of the form collection name:[list of

books]

The list of books is sorted by book title, except for collections

created from series, in which case series_index is used.

:param collection_attributes: A list of attributes of the Book object

'''

raise

NotImplementedError
(

class

CurrentlyConnectedDevice
:

def

__init__
(
self
):

self
.
_device

=

None

@property

def

device
(
self
):

```
return
```

```
self
```

```
.
```

```
_device
```

```
# A device driver can check if a device is currently connected to calibre using
```

```
# the following code::
```

```
# from calibre.device.interface import currently_connected_device
```

```
# if currently_connected_device.device is None:
```

```
#     # no device connected
```

```
# The device attribute will be either None or the device driver object
```

```
# (DevicePlugin instance) for the currently connected device.
```

```
currently_connected_device
```

```
=
```

```
CurrentlyConnectedDevice
```

```
()
```

Source code for calibre.devices.usbms.cli

```
__license__
=
'GPL v3'

__copyright__
=
'2009, John Schember <john@nachtimwald.com>'

__docformat__
=
'restructuredtext en'

import
os
,
shutil
,
time

from
calibre
import
```

```
fsync
```

```
from
```

```
calibre.devices.errors
```

```
import
```

```
PathError
```

```
from
```

```
calibre.utils filenames
```

```
import
```

```
case_preserving_open_file
```

```
class
```

```
File
```

```
:
```

```
def
```

```
__init__
```

```
(
```

```
self
```

```
,
```

```
path
```

```
):
```

```
stats
```

```
=
```

```
os
.
stat
(
path
)

self
.
is_dir

=

os
.
path
.
isdir
(
path
)

self
.
is_readonly

=

not

os
.
access
(
path
,

os
.
W_OK
)
```

```
self
.  
ctime
```

```
=
```

```
stats  
.  
st_ctime
```

```
self  
.  
wtime
```

```
=
```

```
stats  
.  
st_mtime
```

```
self  
.  
size
```

```
=
```

```
stats  
.  
st_size
```

```
if
```

```
path  
.  
endswith  
(  
os
```

```
.
```



```
sep
):

path

=

path
[:
-
1
]

self
.
path

=

path

self
.
name

=

os
.
path
.
basename
(
path
)

def
```

```
check_transfer
```

```
(  
infile
```

```
,
```

```
dest  
):
```

```
infile
```

```
.
```

```
seek
```

```
(  
0  
)
```

```
dest
```

```
.
```

```
seek
```

```
(  
0  
)
```

```
return
```

```
infile
```

```
.
```

```
read
```

```
()
```

```
==
```

```
dest
```

```
.
```

```
read
```

```
()
```

[\[docs\]](#)

class

CLI

:

def

get_file

(

self

,

path

,

outfile

,

end_session

=

True

):

path

=

self

.

munge_path

(

path

)

with

lopen

(

```
path
,

'rb'
)

as

src
:

shutil
.
copyfileobj
(
src
,

outfile
)

def

put_file
(
self
,

infile
,

path
,

replace_file
=
False
,

end_session
```

```
=
True
):

path

=

self
.
munged_path
(
path
)

close

=

False

if

not

hasattr
(
infile
,
'read'
):

infile
,

close

=
```

```
lopen  
(  
infile  
,
```

```
'rb'  
),
```

```
True
```

```
infile  
.  
seek  
(  
0  
)
```

```
if
```

```
os  
.  
path  
.  
isdir  
(  
path  
):
```

```
path
```

```
=
```

```
os  
.  
path  
.  
join  
(  
path
```

```
,  
  
infile  
.  
name  
)  
  
if  
  
not  
  
replace_file  
  
and  
  
os  
.  
path  
.  
exists  
(  
path  
):  
  
raise  
  
PathError  
(  
'File already exists: '  
  
+  
  
path  
)  
  
dest  
,  
  
actual_path
```

```
=  
  
case_preserving_open_file  
(  
    path  
)  
  
    with  
  
    dest  
    :  
  
        try  
        :  
  
            shutil  
            .  
            copyfileobj  
            (  
                infile  
            ,  
  
                dest  
            )  
  
        except  
  
        OSError  
        :  
  
            print  
            (  
                'WARNING: First attempt to send file to device failed'  
            )  
  
            time  
            .
```



```
sleep  
(  
0.2  
)
```

```
infile  
.  
seek  
(  
0  
)
```

```
dest  
.  
seek  
(  
0  
)
```

```
dest  
.  
truncate  
(  
)
```

```
shutil  
.  
copyfileobj  
(  
infile  
,
```

```
dest  
)
```

```
fsync  
(  
dest  
)
```

```
# if not check_transfer(infile, dest): raise Exception('Transfer failed')
```

```
if
```

```
close
```

```
:
```

```
infile
```

```
.
```

```
close
```

```
()
```

```
return
```

```
actual_path
```

```
def
```

```
munge_path
```

```
(
```

```
self
```

```
,
```

```
path
```

```
):
```

```
if
```

```
path
```

```
.
```

```
startswith
```

```
(
```

```
'/'
```

```
)
```

and

not

```
(  
path  
.  
startswith  
(  
self  
.  
_main_prefix  
)  

```

or

```
(  
self  
.  
_card_a_prefix  

```

and

```
path  
.  
startswith  
(  
self  
.  
_card_a_prefix  
)  
)
```

or

```
(  
self  
.  
_card_b_prefix  

```

and

```
path
.
startswith
(
self
.
_card_b_prefix
)):
```

```
path
=
self
.
_main_prefix
```

```
+
path
[
1
:]
```

```
elif
```

```
path
.
startswith
(
'carda:'
):
```

```
path
=
```

```
path
.
replace
```

```
(  
    'carda:'  
    ,  
    self  
    .  
    _card_a_prefix  
    [:  
    -  
    1  
    ])  

```

elif

```
path  
.  
startswith  
(  
    'cardb:'  
    ):  

```

```
path
```

```
=
```

```
path  
.  
replace  
(  
    'cardb:'  
    ,  

```

```
self  
.  
_card_b_prefix  
[:  
-  
1  
])
```

```
return
```

```
path
```

```
def
```

```
list
```

```
(
```

```
self
```

```
,
```

```
path
```

```
,
```

```
recurse
```

```
=
```

```
False
```

```
,
```

```
end_session
```

```
=
```

```
True
```

```
,
```

```
munge
```

```
=
```

```
True
```

```
):
```

```
if
```

```
munge
```

```
:
```

```
path
```

```
=
```

```
self
```

```
.  
munged_path  
(  
    path  
)
```

```
if
```

```
    os
```

```
    .  
    path
```

```
    .  
    isfile  
    (  
        path  
    ):
```

```
return
```

```
[(  
    os  
    .  
    path  
    .  
    dirname  
    (  
        path  
    ),
```

```
[  
    File  
    (  
        path  
    )]]
```

```
entries
```

```
=
```

```
[
```

```
File
(
  os
  .
  path
  .
  join
  (
    path
    ,

    f
  )
)

for

f

in

os
.
listdir
(
  path
)

]

dirs

=

[(
  path
  ,

  entries
)]

for

_file
```


in

entries

:

if

recurse

and

_file

.

is_dir

:

dirs

[

len

(

dirs

);]

=

self

.

list

(

_file

.

path

,

recurse

=

True

,

munge

```
=
False
)

return

dirs

def

mkdir
(
self
,

path
,

end_session
=
True
):

if

self
.
SUPPORTS_SUB_DIRS
:

path

=

self
.
munged_path
(
```

```
path
)

os
.
mkdir
(
path
)

def

rm
(
self
,

path
,

end_session
=
True
):

path

=

self
.
munged_path
(
path
)

self
.
```

```
delete_books
```

```
([  
    path  
])
```

```
def
```

```
touch
```

```
(  
    self  
    ,
```

```
    path  
    ,
```

```
end_session  
=
```

```
True  
):
```

```
path
```

```
=
```

```
self
```

```
.  
munged_path  
(  
    path  
)
```

```
if
```

```
not
```

```
os
```

```
.  
path
```

```
.
```

```
exists  
(  
    path  
):
```

```
    lopen  
    (  
        path  
        ,  
        'wb'  
    )  
    .  
    close  
    ()
```

```
    if
```

```
        not
```

```
            os  
            .  
            path  
            .  
            isdir  
            (  
                path  
            ):
```

```
            os  
            .  
            utime  
            (  
                path  
                ,  
                None  
            )
```


Source code for calibre.devices.usbms.device

```
__license__
=
'GPL v3'

__copyright__
=
'2009, John Schember <john at nachtimwald.com> '
\
'2009, Kovid Goyal <kovid@kovidgoyal.net>'

__docformat__
=
'restructuredtext en'

'''
Generic device driver. This is not a complete stand alone driver. It is
intended to be subclassed with the relevant parts implemented for a particular
device. This class handles device detection.
'''
```

import

glob

import

os

import

re

import

subprocess

import

sys

import

time

from

collections

import

namedtuple

from

itertools

import

repeat

from

calibre

import

prints

from

calibre.constants

import

DEBUG

,

isfreebsd

,

islinux

,

ismacos

,

iswindows

from

calibre.devices.errors

import

DeviceError

from

calibre.devices.interface

import

DevicePlugin

from

```
calibre.devices.usbms.deviceconfig
```

```
import
```

```
DeviceConfig
```

```
from
```

```
calibre.utils filenames
```

```
import
```

```
ascii_filename
```

```
as
```

```
sanitize
```

```
from
```

```
polyglot.builtins
```

```
import
```

```
iteritems
```

```
,
```

```
string_or_bytes
```

```
if
```

```
ismacos
```

```
:
```

```
osx_sanitize_name_pat
```

```
=
```

```
re
```

```
.
```

```
compile
(
  r
  '[-.]'
)

if

iswindows
:

usb_info_cache

=

{}

def

eject_exe
():

base

=

sys
.
extensions_location

if

hasattr
(
  sys
  ,
  'new_app_layout'
```

```
)  
  
else  
  
os  
.  
path  
.  
dirname  
(  
sys  
.  
executable  
)  
  
return  
  
os  
.  
path  
.  
join  
(  
base  
,  
  
'calibre-eject.exe'  
)  
  
class  
  
USBDevice  
:  
  
def  
  
__init__  
(
```

```
self
,

dev
):

self
.
idVendor

=

dev
[
0
]

self
.
idProduct

=

dev
[
1
]

self
.
bcdDevice

=

dev
[
2
]
```

```
if
iswindows
:

# Getting this information requires communicating with the device

# we only do that in the can_handle_windows() method, if needed.

self
.
manufacturer

=

self
.
serial

=

self
.
product

=

None

else
:

self
.
manufacturer

=
```

```
dev
```

```
[
```

```
3
```

```
]
```

```
self
```

```
.
```

```
product
```

```
=
```

```
dev
```

```
[
```

```
4
```

```
]
```

```
self
```

```
.
```

```
serial
```

```
=
```

```
dev
```

```
[
```

```
5
```

```
]
```

```
def
```

```
match_serial
```

```
(
```

```
self
```

```
,
```

```
serial
```

```
):
```

```
return
```

```
self
.
serial

and

self
.
serial

==

serial


def

match_numbers
(
self
,
vid
,
pid
,
bcd
):

return

self
.
idVendor

==

vid
```


and

self

.

idProduct

==

pid

and

self

.

bcdDevice

==

bcd

def

match_strings

(

self

,

vid

,

pid

,

bcd

,

man

,

prod

```
):  
  
if  
  
not  
  
self  
.  
match_numbers  
(  
vid  
,  
  
pid  
,  
  
bcd  
):
```

```
return
```

```
False
```

```
if  
  
man  
  
==  
  
self  
.  
manufacturer  
  
and  
  
prod  
  
==  
  
self
```

```
.
product
:

return

True

# As of macOS 10.11.4 Apple started mangling the names returned via the

# IOKit registry. See

# https://www.mobileread.com/forums/showthread.php?t=273213

m

=

osx_sanitize_name_pat

.
sub
(
    _
,

(
    self
.
manufacturer

or

"
))

p
```

```
=  
  
osx_sanitize_name_pat
```

```
.  
sub
```

```
(  
    '  
    _  
,
```

```
(  
    self  
    .  
    product
```

```
or
```

```
"  
))
```

```
return
```

```
m
```

```
==
```

```
man
```

```
and
```

```
p
```

```
==
```

```
prod
```

```
\[docs\]
```

```
class
```

Device

```
(  
DeviceConfig  
,  
DevicePlugin  
):
```

```
'''
```

This class provides logic common to all drivers for devices that export themselves as USB Mass Storage devices. Provides implementations for mounting/ejecting of USBMS devices on all platforms.

```
'''
```

```
VENDOR_ID
```

```
=
```

```
0x0
```

```
PRODUCT_ID
```

```
=
```

```
0x0
```

```
BCD
```

```
=
```

```
None
```

VENDOR_NAME

=

None

#: String identifying the main memory of the device in the Windows PnP id

#: strings

#: This can be None, string, list of strings or compiled regex

WINDOWS_MAIN_MEM

=

None

#: String identifying the first card of the device in the Windows PnP id

#: strings

#: This can be None, string, list of strings or compiled regex

WINDOWS_CARD_A_MEM

=

None

#: String identifying the second card of the device in the Windows PnP id

#: strings

#: This can be None, string, list of strings or compiled regex

WINDOWS_CARD_B_MEM

=

None

#: Used by the new driver detection to disambiguate main memory from

#: storage cards. Should be a regular expression that matches the

#: main memory mount point assigned by macOS

OSX_MAIN_MEM_VOL_PAT

=

None

OSX_EJECT_COMMAND

=

[
'diskutil'

,

'eject'

]

MAIN_MEMORY_VOLUME_LABEL

=

"

STORAGE_CARD_VOLUME_LABEL

=

"

STORAGE_CARD2_VOLUME_LABEL

=

None

EBOOK_DIR_MAIN

=

"

EBOOK_DIR_CARD_A

=

"

EBOOK_DIR_CARD_B

=


```
"
```

```
DELETE_EXTS
```

```
=
```

```
[]
```

```
# USB disk-based devices can see the book files on the device, so can
```

```
# copy these back to the library
```

```
BACKLOADING_ERROR_MESSAGE
```

```
=
```

```
None
```

```
#: The maximum length of paths created on the device
```

```
MAX_PATH_LEN
```

```
=
```

```
250
```

```
#: Put news in its own folder
```

```
NEWS_IN_FOLDER
```

```
=
```

True

[\[docs\]](#)

def

reset

(

self

,

key

=

'-1'

,

log_packets

=

False

,

report_progress

=

None

,

detected_device

=

None

):

self

.

_main_prefix

=

self

```
.
_card_a_prefix

=

self
.
_card_b_prefix

=

None

self
.
detected_device

=

None

if

detected_device

is

None

else

USBDevice
(
detected_device
)

self
.
set_progress_reporter
(
report_progress
```

)

[\[docs\]](#)

def

set_progress_reporter

(

self

,

report_progress

):

self

.

report_progress

=

report_progress

self

.

report_progress

=

report_progress

if

self

.

report_progress

is

None

```
:  
  
self  
.  
report_progress  
  
=  
  
lambda  
  
x  
,  
  
y  
:  
  
x  
  
\[docs\]  
  
def  
  
card_prefix  
(  
self  
,  
  
end_session  
=  
True  
):  
  
return  
  
(  
self  
.  
_card_a_prefix  
,
```

```
self
.
_card_b_prefix
)

@classmethod

def

_windows_space
(
cls
,

prefix
):

if

not

prefix
:

return

0
,

0

prefix

=

prefix
[:
-

```

```
1
]

from

calibre_extensions

import

winutil

try
:

available_space
,

total_space
,

free_space

=

winutil
.
get_disk_free_space
(
prefix
)

except

OSError

as

err
:
```

```
if
```

```
err
```

```
.
```

```
winerror
```

```
==
```

```
winutil
```

```
.
```

```
ERROR_NOT_READY
```

```
:
```

```
# Disk not ready
```

```
time
```

```
.
```

```
sleep
```

```
(
```

```
3
```

```
)
```

```
available_space
```

```
,
```

```
total_space
```

```
,
```

```
free_space
```

```
=
```

```
winutil
```

```
.
```

```
get_disk_free_space
```

```
(
```

```
prefix
```

```
)
```


else

:

raise

return

total_space

,

available_space

[\[docs\]](#)

def

total_space

(

self

,

end_session

=

True

):

msz

=

casz

=

cbsz

```
=
```

```
0
```

```
if
```

```
not
```

```
iswindows
```

```
:
```

```
if
```

```
self
```

```
.
```

```
_main_prefix
```

```
is
```

```
not
```

```
None
```

```
:
```

```
stats
```

```
=
```

```
os
```

```
.
```

```
statvfs
```

```
(
```

```
self
```

```
.
```

```
_main_prefix
```

```
)
```

```
msz
```

=

stats

.

f_fsize

*

(

stats

.

f_blocks

+

stats

.

f_bavail

-

stats

.

f_bfree

)

if

self

.

_card_a_prefix

is

not

None

:

stats

=

os

.

statvfs

(

self

.

_card_a_prefix

)

casz

=

stats

.

f_fsize

*

(

stats

.

f_blocks

+

stats

.

f_bavail

-

stats

.

f_bfree

)

if

```
self
.  
_card_b_prefix
```

```
is
```

```
not
```

```
None
```

```
:
```

```
stats
```

```
=
```

```
os
```

```
.  
statvfs
```

```
(  
self
```

```
.  
_card_b_prefix  
)
```

```
cbsz
```

```
=
```

```
stats
```

```
.  
f_fsize
```

```
*
```

```
(  
stats
```

```
.  
f_blocks
```

```
+
```

```
stats
.  
f_bavail
```

```
-
```

```
stats  
.  
f_bfree  
)
```

```
else  
:
```

```
msz
```

```
=
```

```
self  
.  
_windows_space  
(  
self  
.  
_main_prefix  
)[  
0  
]
```

```
casz
```

```
=
```

```
self  
.  
_windows_space  
(  
self  
.  

```

```
_card_a_prefix
)[
0
]

cbsz

=

self
.
_windows_space
(
self
.
_card_b_prefix
)[
0
]

return

(
msz
,

casz
,

cbsz
)

\[docs\]

def

free_space
(
self
```

```
,  
  
end_session  
=  
True  
):  
  
msz  
  
=  
  
casz  
  
=  
  
cbsz  
  
=  
  
0  
  
if  
  
not  
  
iswindows  
:  
  
if  
  
self  
.  
_main_prefix  
  
is  
  
not  
  
None  
:
```


stats

=

os

.

statvfs

(

self

.

_main_prefix

)

msz

=

stats

.

f_fsize

*

stats

.

f_bavail

if

self

.

_card_a_prefix

is

not

None

:

stats

=

os

.

statvfs

(

self

.

_card_a_prefix

)

casz

=

stats

.

f_fsize

*

stats

.

f_bavail

if

self

.

_card_b_prefix

is

not

None

:

```
stats
```

```
=
```

```
os
```

```
.
```

```
statvfs
```

```
(
```

```
self
```

```
.
```

```
_card_b_prefix
```

```
)
```

```
cbsz
```

```
=
```

```
stats
```

```
.
```

```
f_fsize
```

```
*
```

```
stats
```

```
.
```

```
f_bavail
```

```
else
```

```
:
```

```
msz
```

```
=
```

```
self
```

```
.
```

```
_windows_space
```

```
(
```

```
self
.
_main_prefix
)[
1
]

casz

=

self
.
_windows_space
(
self
.
_card_a_prefix
)[
1
]

cbz

=

self
.
_windows_space
(
self
.
_card_b_prefix
)[
1
]

return
```

```
(  
msz  
,  
casz  
,  
cbsz  
)
```

def

```
windows_filter_pnp_id  
(  
self  
,  
  
pnp_id  
):
```

return

False

[\[docs\]](#)

def

```
windows_sort_drives  
(  
self  
,  
  
drives  
):
```

```
'''
```

Called to disambiguate main memory and storage card for devices that do not distinguish between them on the basis of `WINDOWS\_CARD\_NAME`.

For example: The EB600

'''

return

drives

[\[docs\]](#)

def

can_handle_windows

(

self

,

usbdevice

,

debug

=

False

):

if

hasattr

(

self

.

can_handle

,

'is_base_class_implementation'

```
):
```

```
# No custom can_handle implementation
```

```
return
```

```
True
```

```
# Delegate to the unix can_handle function, creating a unix like
```

```
# USBDevice object
```

```
from
```

```
calibre.devices.winusb
```

```
import
```

```
get_usb_info
```

```
dev
```

```
=
```

```
usb_info_cache
```

```
.
```

```
get
```

```
(
```

```
usbdevice
```

```
)
```

```
if
```

```
dev
```

```
is
```

None

:

try

:

data

=

get_usb_info

(

usbdevice

,

debug

=

debug

)

except

Exception

:

time

.

sleep

(

0.1

)

try

:

data


```
=

get_usb_info
(
usbdevice
,

debug
=
debug
)

except

Exception
:

data

=

{}

dev

=

usb_info_cache
[
usbdevice
]

=

namedtuple
(

'USBDevice'
```

```
,  
  
'vendor_id product_id bcd manufacturer product serial'  
)(  
  
usbdevice  
.  
vendor_id  
,  
  
usbdevice  
.  
product_id  
,  
  
usbdevice  
.  
bcd  
,  
  
data  
.  
get  
(  
'manufacturer'  
)  
  
or  
  
"  
,  
  
data  
.  
get  
(  
'product'  
)  
  
or
```

```
"
,

data
.
get
(
'serial_number'
)

or

"
)

if

debug
:

prints
(
f
'USB Info for device:
{
dev
}
'
)

return

self

.
can_handle
(
dev
,

debug
```

```
=
debug
)

def

open_windows
(
self
):

from

calibre.devices.scanner

import

drive_is_ok

from

calibre.devices.winusb

import

get_drive_letters_for_device

usbdev

=

self
.
device_being_opened

debug

=
```

```
DEBUG
```

```
or
```

```
getattr
```

```
(
```

```
self
```

```
,
```

```
'do_device_debug'
```

```
,
```

```
False
```

```
)
```

```
try
```

```
:
```

```
dlmap
```

```
=
```

```
get_drive_letters_for_device
```

```
(
```

```
usbdev
```

```
,
```

```
debug
```

```
=
```

```
debug
```

```
)
```

```
except
```

```
Exception
```

```
:
```

```
dlmap
```

```
=
```

```
{}
```

```
if
```

```
not
```

```
dlmap
```

```
.
```

```
get
```

```
(
```

```
'drive_letters'
```

```
):
```

```
time
```

```
.
```

```
sleep
```

```
(
```

```
7
```

```
)
```

```
dlmap
```

```
=
```

```
get_drive_letters_for_device
```

```
(
```

```
usbdev
```

```
,
```

```
debug
```

```
=
```

```
debug
```

```
)
```

```
if
debug
:

from

pprint

import

pformat

prints
(
f
'Drive letters for
{
usbdev
}
'
)

prints
(
pformat
(
dlmap
))

filtered

=

set
()
```

```
for
    dl
in
    dlmap
    [
        'drive_letters'
    ]:

    pnp_id

    =

    dlmap
    [
        'pnp_id_map'
    ][
        dl
    ]
    .
    upper
    ()

    if

    dl

    in

    dlmap
    [
        'readonly_drives'
    ]:

    filtered

    .
    add
    (
```



```
dl
)

if

debug
:

prints
(
'Ignoring the drive
%s
as it is readonly'

%

dl
)

elif

self
.
windows_filter_pnp_id
(
pnp_id
):

filtered
.
add
(
dl
)

if

debug
```

```
:  
  
prints  
(  
f  
'Ignoring the drive  
{  
dl  
}  
because of a PNP filter on  
{  
pnp_id  
}  
'  
)  
  
elif  
  
not  
  
drive_is_ok  
(  
dl  
,  
  
debug  
=  
debug  
):  
  
filtered  
.  
add  
(  
dl  
)  
  
if
```

```
debug
:

prints
(
'Ignoring the drive
%s
because failed to get free space for it'

%

dl
)

dlmap
[
'drive_letters'
]

=

[
dl

for

dl

in

dlmap
[
'drive_letters'
]

if

dl

not
```

in

filtered

]

if

not

dlmap

[

'drive_letters'

]:

raise

DeviceError

(

—
(

'Unable to detect any disk drives for the device:

%s

. Try rebooting'

)

%

self

.

get_gui_name

())

drives

=

{}

for

drive_letter

,

which

in

zip

(

dlmap

[

'drive_letters'

],

'main carda cardb'

.

split

()):

drives

[

which

]

=

drive_letter

+

':

\\

,

drives

```
=  
  
self  
.windows_sort_drives  
(  
drives  
)
```

```
self  
. _main_prefix
```

```
=  
  
drives  
.get  
(  
'main'  
)
```

```
self  
. _card_a_prefix
```

```
=  
  
drives  
.get  
(  
'carda'  
,  
None  
)
```

```
self  
.
```

```
_card_b_prefix
```

```
=
```

```
drives
```

```
.
```

```
get
```

```
(
```

```
'cardb'
```

```
,
```

```
None
```

```
)
```

```
@classmethod
```

```
def
```

```
run_ioreg
```

```
(
```

```
cls
```

```
,
```

```
raw
```

```
=
```

```
None
```

```
):
```

```
if
```

```
raw
```

```
is
```

```
not
```

```
None
```

```
:
```

return

raw

ioreg

=

`'/usr/sbin/ioreg'`

if

not

os

.

access

(

ioreg

,

os

.

X_OK

):

ioreg

=

`'ioreg'`

cmd

=

(

ioreg


```
+
'-w 0 -S -c IOMedia'
)
.
split
()

for

i

in

range
(
3
):

try
:

return

subprocess
.
Popen
(
cmd
,

stdout
=
subprocess
.
PIPE
)
.
communicate
()[
```

```
0
]

except

OSError
:

# Probably an interrupted system call

if

i

==

2
:

raise

time
.
sleep
(
2
)

def

osx_sort_names
(
self
,
names
):
```

```
return
```

```
names
```

```
@classmethod
```

```
def
```

```
osx_run_mount
```

```
(
```

```
cls
```

```
):
```

```
for
```

```
i
```

```
in
```

```
range
```

```
(
```

```
3
```

```
):
```

```
try
```

```
:
```

```
return
```

```
subprocess
```

```
.
```

```
Popen
```

```
(
```

```
'mount'
```

```
,
```

```
stdout
=
subprocess
.
PIPE
)
.
communicate
()[
0
]
```

except

```
OSError
:
```

```
# Probably an interrupted system call
```

if

```
i
==
```

```
2
:
```

raise

```
time
.
sleep
(
2
)
```

```
@classmethod
```

```
def
```

```
osx_get_usb_drives
```

```
(  
cls  
):
```

```
from
```

```
calibre_extensions.usbobserver
```

```
import
```

```
get_usb_drives
```

```
return
```

```
get_usb_drives  
()
```

```
def
```

```
_osx_bsd_names
```

```
(  
self  
):
```

```
drives
```

```
=
```

```
self
```

```
.
```

```
osx_get_usb_drives  
(
```

```
matches
```

```
=
```

```
[]
```

```
d
```

```
=
```

```
self
```

```
.  
detected_device
```

```
if
```

```
d
```

```
.  
serial  
:
```

```
for
```

```
path
```

```
,
```

```
vid
```

```
,
```

```
pid
```

```
,
```

```
bcd
```

```
,
```

```
ven
```

```
,  
  
prod  
,  
  
serial  
  
in  
  
drives  
:  
  
if  
  
d  
.  
match_serial  
(  
serial  
):  
  
matches  
.  
append  
(  
path  
)  
  
if  
  
not  
  
matches  
  
and  
  
d  
.  
manufacturer
```

and

d

.

product

:

for

path

,

vid

,

pid

,

bcd

,

man

,

prod

,

serial

in

drives

:

if

d

.

match_strings

(

vid


```
,
pid
',
bcd
',
man
',
prod
):

matches
.
append
(
path
)

if

not

matches
:

# Since Apple started mangling the names stored in the IOKit

# registry, we cannot trust match_strings() so fallback to matching

# on just numbers. See http://www.mobileread.com/forums/showthread.php?t=273213

for

path
```

```
,  
  
vid  
,  
  
pid  
,  
  
bcd  
,  
  
man  
,  
  
prod  
,  
  
serial  
  
in  
  
drives  
:  
  
if  
  
d  
.  
match_numbers  
(  
vid  
,  
  
pid  
,  
  
bcd  
):  
  
matches  
.
```

```
append
(
path
)

if

not

matches
:

from

pprint

import

pformat

raise

DeviceError
(

f
'Could not detect BSD names for
{
self
.
name
}
. Try rebooting.
\n
Output from osx_get_usb_drives():
\n
{
pformat
(
```

```
drives
```

```
)
```

```
}
```

```
,
```

```
)
```

```
pat
```

```
=
```

```
re
```

```
.
```

```
compile
```

```
(
```

```
r
```

```
'(?P<m>\d+)([a-z]+(?P<p>\d+)){0,1}'
```

```
)
```

```
def
```

```
nums
```

```
(
```

```
x
```

```
):
```

```
'Return (disk num, partition number)'
```

```
m
```

```
=
```

```
pat
```

```
.
```

```
search
```

```
(
```

```
x
```

```
)
```

```
if
```

```
m
```

```
is
```

```
None
```

```
:
```

```
return
```

```
(  
10000
```

```
,
```

```
-
```

```
1
```

```
)
```

```
g
```

```
=
```

```
m
```

```
.
```

```
groupdict
```

```
()
```

```
if
```

```
g
```

```
[
```

```
'p'
```

```
]
```

```
is
```

```
None
```

```
:  
  
g  
[  
  'p'  
]  
  
=  
  
0  
  
return  
  
list  
(  
  map  
  (  
    int  
    ,  
    (  
      g  
      .  
      get  
      (  
        'm'  
      ),  
      g  
      .  
      get  
      (  
        'p'  
      )))  
  
def  
cmp_key  
(
```

```
x
):
```

```
'''
```

Sorting based on the following scheme:

- disks without partitions are first*
- sub sorted based on disk number*
- disks with partitions are sorted first on
disk number, then on partition number*

```
'''
```

```
x
```

```
=
```

```
x
```

```
.
```

```
rpartition
```

```
(
```

```
'/'
```

```
)[
```

```
-
```

```
1
```

```
]
```

```
disk_num
```

```
,
```

```
part_num
```

```
=
```

```
nums
```

```
(
```

```
x
)

has_part

=

1

if

part_num

>

0

else

0

return

has_part
,
disk_num
,
part_num

matches
.
sort
(
key
=
cmp_key
)
```


drives

=

```
{  
'main'  
:  
matches  
[  
0  
]}
```

if

```
len  
(  
matches  
)
```

>

```
1  
:
```

```
drives  
[  
'carda'  
]
```

=

```
matches  
[  
1  
]
```

if

```
len  
(  
matches  
)
```

```
>
```

```
2  
:
```

```
drives  
[  
'cardb'  
]
```

```
=
```

```
matches  
[  
2  
]
```

```
return
```

```
drives
```

```
def
```

```
osx_bsd_names  
(  
self  
):
```

```
drives
```

```
=
```

```
{  
  
    for  
        i  
        in  
            range  
            (  
                3  
            ):  
  
        try  
            :  
  
            drives  
  
            =  
  
            self  
            .  
            _osx_bsd_names  
            ()  
  
            if  
  
            len  
            (  
                drives  
            )  
  
            >  
  
            1  
            :  
  
            # wait for device to settle and SD card (if any) to become available
```

```
return
```

```
drives
```

```
except
```

```
Exception
```

```
:
```

```
if
```

```
i
```

```
==
```

```
2
```

```
:
```

```
raise
```

```
time
```

```
.
```

```
sleep
```

```
(
```

```
3
```

```
)
```

```
return
```

```
drives
```

```
def
```

```
open_osx
```

```
(
```

```
self
):

from

calibre_extensions.usbobserver

import

get_mounted_filesystems

bsd_drives

=

self
.
osx_bsd_names
()

drives

=

self
.
osx_sort_names
(
    bsd_drives
.
copy
())

mount_map

=

get_mounted_filesystems
()
```

```
drives
```

```
=
```

```
{  
k  
:
```

```
mount_map
```

```
.
```

```
get
```

```
(  
v  
)
```

```
for
```

```
k
```

```
,
```

```
v
```

```
in
```

```
iteritems
```

```
(  
drives  
))
```

```
if
```

```
DEBUG
```

```
:
```

```
print
```

```
()
```

```
from
```

pprint

import

pprint

pprint

{

'bsd_drives'

:

bsd_drives

,

'mount_map'

:

mount_map

,

'drives'

:

drives

})

if

drives

.

get

(

'carda'

)

is

None

and

drives

.

get

(

'cardb'

)

is

not

None

:

drives

[

'carda'

]

=

drives

.

pop

(

'cardb'

)

if

drives

.

get

(

'main'

)

is

None

and

drives

.

get

(

'carda'

)

is

not

None

:

drives

[

'main'

]

=

drives

.

pop

(

'carda'

)

if

drives

.

get

(

'carda'

)

is

None

and

drives

.

get

(

'cardb'

)

is

not

None

:

drives

[

'carda'

]

=

drives

.

pop

(

'cardb'

)

if

drives

.

get

(

'main'

)

is

None

:

raise

DeviceError

(

(

'Unable to detect the

%s

mount point. Try rebooting.'

)

%

self

.

__class__

.

__name__

)

pat

=

self

.

OSX_MAIN_MEM_VOL_PAT

if

pat

is

not

None

and

len
(
drives
)

>

1

and

'main'

in

drives
:

if

pat
.
search
(
drives
[
'main'
)

is

None

:

main

=

```
drives  
[  
  'main'  
]
```

```
for
```

```
x
```

```
in
```

```
(  
  'carda'  
  ,  
  'cardb'  
):
```

```
if
```

```
x
```

```
in
```

```
drives
```

```
and
```

```
pat  
.  
search  
(  
  drives  
  [  
    x  
  ]):
```

```
drives  
[
```

```
'main'
]

=

drives
.
pop
(
x
)

drives
[
x
]

=

main

break

self
.
_main_prefix

=

drives
[
'main'
]
+
os
.
sep
```

```
def
get_card_prefix
(
c
):
```

```
ans
```

```
=
```

```
drives
```

```
.
get
(
c
,
```

```
None
)
```

```
if
```

```
ans
```

```
is
```

```
not
```

```
None
:
```

```
ans
```

```
+=
```

```
os
```

```
.
sep
```

```
return
```

```
ans
```

```
self
```

```
._card_a_prefix
```

```
=
```

```
get_card_prefix
```

```
(  
    'carda'  
)
```

```
self
```

```
._card_b_prefix
```

```
=
```

```
get_card_prefix
```

```
(  
    'cardb'  
)
```

```
def
```

```
find_device_nodes
```

```
(  
    self  
    ,
```

```
detected_device
```

```
=
```

```
None
```

```
);
```



```
def
```

```
    walk
```

```
    (
```

```
    base
```

```
    );
```

```
    base
```

```
    =
```

```
    os
```

```
    .
```

```
    path
```

```
    .
```

```
    abspath
```

```
    (
```

```
    os
```

```
    .
```

```
    path
```

```
    .
```

```
    realpath
```

```
    (
```

```
    base
```

```
    ))
```

```
for
```

```
    x
```

```
in
```

```
    os
```

```
    .
```

```
    listdir
```

```
    (
```

```
    base
```

```
    );
```

p

=

os

.

path

.

join

(

base

,

x

)

if

os

.

path

.

islink

(

p

)

or

not

os

.

access

(

p

,

os

.

```
R_OK
):
```

```
continue
```

```
isfile
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
isfile
```

```
(
```

```
p
```

```
)
```

```
yield
```

```
p
```

```
,
```

```
isfile
```

```
if
```

```
not
```

```
isfile
```

```
:
```

```
yield from
```

```
walk
```

```
(
```

```
p
```

```
)
```

def

raw2num

(
raw
):

raw

=

raw

.
lower
()

if

not

raw

.
startswith
(
'0x'
):

raw

=

'0x'

+

raw

```
return
```

```
int
```

```
(
```

```
raw
```

```
,
```

```
16
```

```
)
```

```
# Find device node based on vendor, product and bcd
```

```
d
```

```
,
```

```
j
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
dirname
```

```
,
```

```
os
```

```
.
```

```
path
```

```
.
```

```
join
```

```
usb_dir
```

```
=
```

```
None
```

```
if
    detected_device
    is
    None
:
```

```
    detected_device
    =
    self
    .
    detected_device
```

```
def
    test
    (
    val
    ,
    attr
    ):
    q
    =
    getattr
    (
    detected_device
    ,
    attr
```

```
)
```

```
return
```

```
q
```

```
==
```

```
val
```

```
def
```

```
getnum
```

```
(
```

```
usb_dir
```

```
):
```

```
def
```

```
rc
```

```
(
```

```
q
```

```
):
```

```
with
```

```
open
```

```
(
```

```
j
```

```
(
```

```
usb_dir
```

```
,
```

```
q
```

```
),
```

```
'rb'
```

```
)
```

```
as
```

```
f  
:
```

```
return
```

```
raw2num
```

```
(  
f
```

```
.  
read  
(
```

```
.  
decode  
(  
'utf-8'  
)
```

```
return
```

```
rc
```

```
for
```

```
x  
,
```

```
isfile
```

```
in
```

```
walk
```

```
(  
'/sys/devices'  
):
```



```
if
isfile

and

x
.
endswith
(
'idVendor'
):

usb_dir

=

d
(
x
)

for
y
in
(
'idProduct'
,
'idVendor'
,
'bcdDevice'
):

if
```

not

os

.

access

(

j

(

usb_dir

,

y

),

os

.

R_OK

):

usb_dir

=

None

break

if

usb_dir

is

None

:

continue

```
ven
,

prod
,

bcd

=

map
(
  getnum
  (
    usb_dir
  ),
  (
    'idVendor'
  ,
    'idProduct'
  ,
    'bcdDevice'
  ))

if

not

(
  test
  (
    ven
  ,
    'idVendor'
  )
and
```

```
test
(
prod
,
'idProduct'
)
```

and

```
test
(
bcd
,
'bcdDevice'
)):
```

usb_dir

=

None

continue

else

:

break

if

usb_dir

is

None

:

raise

DeviceError

(

—

(

'Unable to detect the

%s

disk drive.'

)

%

self

.

__class__

.

__name__

)

devnodes

,

ok

=

[],

{}

for

x

,

```
isfile
```

```
in
```

```
walk  
(  
usb_dir  
):
```

```
if
```

```
not
```

```
isfile
```

```
and
```

```
'/block/'
```

```
in
```

```
x  
:
```

```
parts
```

```
=
```

```
x  
.  
split  
(  
'/'  
)
```

```
idx
```

```
=
```

```
parts
.  
index  
(  
'block'  
)
```

```
if
```

```
idx  
  
==
```

```
len  
(  
parts  
)  
-  
2  
:
```

```
sz  
  
=
```

```
j  
(  
x  
,  
'size'  
)
```

```
node  
  
=
```

```
parts  
[  
idx
```

```
+
1
]

try
:

with

open
(
sz
,
'rb'
)

as

szf
:

exists

=

int
(
szf
.
read
()
.
decode
(
'utf-8'
))

>
```



```
0
```

```
if
```

```
exists
```

```
:
```

```
node
```

```
=
```

```
self
```

```
.
```

```
find_largest_partition
```

```
(
```

```
x
```

```
)
```

```
ok
```

```
[
```

```
node
```

```
]
```

```
=
```

```
True
```

```
else
```

```
:
```

```
ok
```

```
[
```

```
node
```

```
]
```

```
=
```

```
False
```

```
except
```

```
:
```

```
ok
```

```
[
```

```
node
```

```
]
```

```
=
```

```
False
```

```
if
```

```
DEBUG
```

```
and
```

```
not
```

```
ok
```

```
[
```

```
node
```

```
]:
```

```
print
```

```
(
```

```
f
```

```
,
```

```
\n
```

```
Ignoring the node:
```

```
{
```

```
node
```

```
}
```

```
as could not read size from:
```

```
{
```

```
sz
```

```
}
```

```
'  
)  
  
devnodes
```

```
.  
append  
(  
node  
)
```

```
devnodes
```

```
+=
```

```
list  
(  
repeat  
(  
None  
,  
  
3  
)
```

```
ans
```

```
=
```

```
[  
'/dev/'  
+  
x
```

```
if
```

```
ok
```

```
.  
get
```

```
(
x
,

False
)

else

None

for

x

in

devnodes
]

ans
.
sort
(
key
=
lambda

x
:

x
[
5
:]

if

x

else
```

```
'zzzzz'  
)  
  
return  
  
self  
.  
linux_swap_drives  
(  
ans  
[:  
3  
)
```

```
def  
  
linux_swap_drives  
(  
self  
,  
  
drives  
):
```

```
return  
  
drives
```

```
def  
  
node_mountpoint  
(  
self  
,  
  
node  
):
```

```
from
```

```
calibre.devices.udisks
```

```
import
```

```
node_mountpoint
```

```
return
```

```
node_mountpoint
```

```
(  
node  
)
```

```
def
```

```
find_largest_partition
```

```
(  
self  
,
```

```
path  
):
```

```
node
```

```
=
```

```
path
```

```
.  
split  
(  
'/'  
)[
```

```
-  
1
```

```
]
```

```
nodes
```

```
=
```

```
[]
```

```
for
```

```
x
```

```
in
```

```
glob
```

```
.
```

```
glob
```

```
(
```

```
path
```

```
+
```

```
'/'
```

```
+
```

```
node
```

```
+
```

```
'*'
```

```
):
```

```
sz
```

```
=
```

```
x
```

```
+
```

```
'/size'
```

```
if
```

not

os

.

access

(

sz

,

os

.

R_OK

):

continue

try

:

with

open

(

sz

,

'rb'

)

as

szf

:

sz

=


```
int
(  
szf  
.  
read  
(  
.  
decode  
(  
'utf-8'  
)
```

```
except  
:
```

```
continue
```

```
if
```

```
sz
```

```
>
```

```
0
```

```
:
```

```
nodes
```

```
.  
append
```

```
((
```

```
x
```

```
.)
```

```
split
```

```
(
```

```
'/'
```

```
)[
```

```
-
```

```
1
```

```
],
```

```
sz
))
```

```
nodes
.
sort
(
key
=
lambda
```

```
x
:
```

```
x
[
1
])
```

```
if
```

```
not
```

```
nodes
:
```

```
return
```

```
node
```

```
return
```

```
nodes
[
-
1
][
```

```
0  
]
```

```
def
```

```
open_linux  
(  
self  
):
```

```
def
```

```
mount  
(  
node  
,  
  
type  
):
```

```
mp
```

```
=
```

```
self  
.  
node_mountpoint  
(  
node  
)
```

```
if
```

```
mp
```

```
is
```

```
not
```

```
None
```

```
:
```

```
return
```

```
mp
```

```
,
```

```
0
```

```
def
```

```
do_mount
```

```
(
```

```
node
```

```
):
```

```
try
```

```
:
```

```
from
```

```
calibre.devices.udisks
```

```
import
```

```
mount
```

```
mount
```

```
(
```

```
node
```

```
)
```

```
return
```

0

except

:

print

(

'Udisks mount call failed:'

)

import

traceback

traceback

.

print_exc

()

return

1

ret

=

do_mount

(

node

)

if

```
ret

!=

0
:

return

None
,

ret

return

self
.
node_mountpoint
(
node
)
+
'/'
,

0

main
,

carda
,

cardb

=

self
```

```

    .
    find_device_nodes
    ()

    if

    main

    is

    None
    :

    raise

    DeviceError
    (
    -
    (
    'Unable to detect the
    %s
    disk drive. Either '

    'the device has already been ejected, or your '

    'kernel is exporting a deprecated version of SYSFS.'
    )

    %
    self

    .
    __class__
    .
    __name__
    )

    if

```

DEBUG

:

print

(

,

\n

Found device nodes:'

,

main

,

carda

,

cardb

)

self

.

_linux_mount_map

=

{}

mp

,

ret

=

mount

(

main

,


```
'main'
)

if
mp
is
None
:

raise
DeviceError
(

_
(
'Unable to mount main memory (Error code:
%d
)'
)
%
ret
)

if
not
mp
.
endswith
(
'/'
):
```

mp

+=

'/'

self

.
_linux_mount_map
[
main
]

=

mp

self

.
_main_prefix

=

mp

self

.
_linux_main_device_node

=

main

cards

=

[(

```
carda
,
'_card_a_prefix'
,
'carda'
),

(
cardb
,
'_card_b_prefix'
,
'cardb'
)]
```

for

```
card
,
prefix
,
typ
```

in

```
cards
:
```

if

```
card
```

is

None

:

continue

mp

,

ret

=

mount

(

card

,

typ

)

if

mp

is

None

:

print

(

'Unable to mount card (Error code:

%d

)'

%

ret

,

file

```
=
sys
.
stderr
)

else
:

if
not

mp
.
endswith
(
'/'
):

mp
+=
'/'

setattr
(
self
,
prefix
,
mp
)

self
```

```
.
_linux_mount_map
[
card
]

=

mp

self
.
filter_read_only_mount_points
()

def

filter_read_only_mount_points
(
self
):

def

is_readonly
(
mp
):

if

mp

is

None
```

```
:  
  
    return  
  
    True  
  
    path  
  
    =  
  
    os  
    .  
    path  
    .  
    join  
    (  
    mp  
    ,  
    'calibre_readonly_test'  
    )  
  
    ro  
  
    =  
  
    True  
  
    try  
    :  
  
    with  
  
    lopen  
    (  
    path  
    ,
```

```
'wb'
```

```
):
```

```
ro
```

```
=
```

```
False
```

```
except
```

```
:
```

```
pass
```

```
else
```

```
:
```

```
try
```

```
:
```

```
os
```

```
.
```

```
remove
```

```
(
```

```
path
```

```
)
```

```
except
```

```
:
```

```
pass
```

```
if
```


DEBUG

and

ro

:

print

(

,

\n

The mountpoint'

,

mp

,

'is readonly, ignoring it'

)

return

ro

for

mp

in

(

'_main_prefix'

,

'_card_a_prefix'

,

'_card_b_prefix'

);

if

is_readonly

(
getattr

(
self

,

mp

,

None

)):

setattr

(
self

,

mp

,

None

)

if

self

.

_main_prefix

is

None

:

```
for
p
in
(
'_card_a_prefix'
,
'_card_b_prefix'
):

nmp

=

getattr
(
self
,
p
,
None
)

if
nmp
is
not
None
:

self
```

```
.
_main_prefix

=

nmp

setattr
(
self
,
p
,
None
)

break

if
self
._main_prefix
is
None
:

raise
DeviceError
(
_
(
'The main memory of
```

```
%s
is read only. '

'This usually happens because of file system errors.'
)

%
self
.
__class__
.
__name__
)

if

self
.
_card_a_prefix

is

None

and

self
.
_card_b_prefix

is

not

None
:

self
```

```
._card_a_prefix
```

```
=
```

```
self
```

```
._card_b_prefix
```

```
self
```

```
._card_b_prefix
```

```
=
```

```
None
```

```
# -----
```

```
#
```

```
# open for FreeBSD
```

```
# find the device node or nodes that match the S/N we already have from the scanner
```

```
# and attempt to mount each one
```

```
# 1. get list of devices in /dev with matching s/n etc.
```

```
# 2. get list of volumes associated with each
```

```
# 3. attempt to mount each one using Hal
```

```
# 4. when finished, we have a list of mount points and associated dbus nodes
```

```
#
```

```
def
```

```
open_freebsd
```

```
(
self
):

# There should be some way to access the -v arg...

verbose

=

False

# this gives us access to the S/N, etc. of the reader that the scanner has found

# and the match routines for some of that data, like s/n, vendor ID, etc.

d
=
self
.
detected_device

if
not
d
.
serial
:

raise
DeviceError
```

```
(  
"Device has no S/N. Can't continue"  
)
```

```
from
```

```
.hal
```

```
import
```

```
get_hal
```

```
hal
```

```
=
```

```
get_hal  
(
```

```
vols
```

```
=
```

```
hal
```

```
.
```

```
get_volumes  
(  
d  
)
```

```
if
```

```
verbose  
:
```

```
print  
(  
"FBSD: "
```



```
,  
  
vols  
)  
  
ok  
,  
  
mv  
  
=  
  
hal  
.  
mount_volumes  
(  
vols  
)  
  
if  
  
not  
  
ok  
:  
  
raise  
  
DeviceError  
(  
_  
(  
'Unable to mount the device'  
)  
)  
  
for  
  
k
```

```
,  
  
v  
  
in  
  
mv  
.  
items  
():  
  
setattr  
(  
self  
,  
  
k  
,  
  
v  
)  
  
#  
  
# -----  
  
#  
  
# this one is pretty simple:  
  
# just umount each of the previously  
  
# mounted filesystems, using the stored volume object  
  
#  
  
def  
  
eject_freebsd  
(
```

```
self
):

from

.hal

import

get_hal

hal

=

get_hal
()

if

self
.
_main_prefix
:

hal
.
unmount
(
self
.
_main_vol
)

if

self
.
```

```
_card_a_prefix  
:
```

```
hal
```

```
.  
unmount
```

```
(  
self
```

```
.  
_card_a_vol  
)
```

```
if
```

```
self
```

```
.  
_card_b_prefix  
:
```

```
hal
```

```
.  
unmount
```

```
(  
self
```

```
.  
_card_b_vol  
)
```

```
self
```

```
.  
_main_prefix
```

```
=
```

```
self
```

```
.  
_main_vol
```

=

None

self

.

_card_a_prefix

=

self

.

_card_a_vol

=

None

self

.

_card_b_prefix

=

self

.

_card_b_vol

=

None

[\[docs\]](#)

def

```
open
(
self
,

connected_device
,

library_uuid
):
```

```
time
.
sleep
(
5
)
```

```
self
.
_main_prefix
=
```

```
self
.
_card_a_prefix
=
```

```
self
.
_card_b_prefix
=
```

None

```
self
.
```

```
device_being_opened
```

```
=
```

```
connected_device
```

```
try
```

```
:
```

```
if
```

```
islinux
```

```
:
```

```
try
```

```
:
```

```
self
```

```
.
```

```
open_linux
```

```
()
```

```
except
```

```
DeviceError
```

```
:
```

```
time
```

```
.
```

```
sleep
```

```
(
```

```
7
```

```
)
```

```
self
```

```
.
```

```
open_linux
()

if
isfreebsd
:

self
.
_main_vol

=

self
.
_card_a_vol

=

self
.
_card_b_vol

=

None

try
:

self
.
open_freebsd
()

except
```



```
DeviceError
```

```
:
```

```
time
```

```
.
```

```
sleep
```

```
(
```

```
2
```

```
)
```

```
self
```

```
.
```

```
open_freebsd
```

```
()
```

```
if
```

```
iswindows
```

```
:
```

```
self
```

```
.
```

```
open_windows
```

```
()
```

```
if
```

```
ismacos
```

```
:
```

```
try
```

```
:
```

```
self
```

```
.
```

```
open_osx
```

```
()

except
    DeviceError
    :

        time
        .
        sleep
        (
            7
        )

        self
        .
        open_osx
        ()

        self
        .
        current_library_uuid
        =
        library_uuid

        self
        .
        post_open_callback
        ()

    finally
        :
```

```
self
.  
device_being_opened
```

```
=
```

```
None
```

```
def
```

```
post_open_callback  
(  
self  
):
```

```
pass
```

```
def
```

```
eject_windows  
(  
self  
):
```

```
from
```

```
threading
```

```
import
```

```
Thread
```

```
drives
```

```
=
```

```
[]
```

for

x

in

```
(  
    '_main_prefix'  
,  
    '_card_a_prefix'  
,  
    '_card_b_prefix'  
):
```

x

=

```
getattr  
(  
    self  
,  
    x  
,  
    None  
)
```

if

x

is

not

None

:

drives

.

append

(

x

[

0

]

.

upper

())

def

do_it

(

drives

):

subprocess

.

Popen

([

eject_exe

[])

+

drives

,

creationflags

=

subprocess

.

CREATE_NO_WINDOW

```
)  
.  
wait  
(
```

```
t
```

```
=
```

```
Thread
```

```
(  
target  
=  
do_it  
,
```

```
args
```

```
=
```

```
[  
drives  
)
```

```
t
```

```
.  
daemon
```

```
=
```

```
True
```

```
t
```

```
.  
start  
(
```

```
self
```

```
.  
__save_win_eject_thread
```

```
=
```

```
t
```

```
def
```

```
eject_osx
```

```
(
```

```
self
```

```
):
```

```
for
```

```
x
```

```
in
```

```
(
```

```
'_main_prefix'
```

```
,
```

```
'_card_a_prefix'
```

```
,
```

```
'_card_b_prefix'
```

```
):
```

```
x
```

```
=
```

```
getattr
```

```
(
```

```
self
```

```
,
```

```
x
```

```
,
```

None

)

if

x

is

not

None

:

try

:

subprocess

.

Popen

(

self

.

OSX_EJECT_COMMAND

+

[

x

])

except

:

pass


```
def
eject_linux
(
self
):

from
calibre.devices.udisks

import

eject
,

umount

drives

=

[
d

for
d

in

self
.
find_device_nodes
()

if
d

]
```

for

d

in

drives

:

try

:

umount

(

d

)

except

:

pass

for

d

in

drives

:

try

:

```
eject
```

```
(
```

```
d
```

```
)
```

```
except
```

```
Exception
```

```
as
```

```
e
```

```
:
```

```
print
```

```
(
```

```
'Udisks eject call for:'
```

```
,
```

```
d
```

```
,
```

```
'failed:'
```

```
)
```

```
print
```

```
(
```

```
'
```

```
\t
```

```
'
```

```
,
```

```
e
```

```
)
```

```
\[docs\]
```

```
def
    eject
    (
        self
    ):

        if

            islinux
            :

                try
                :

                    self
                    .
                    eject_linux
                    ()

                except
                :

                    pass

            if

                isfreebsd
                :

                    try
                    :

                        self
                        .
```

```
eject_freebsd  
()  
  
except  
:  
  
pass  
  
if  
  
iswindows  
:  
  
try  
:  
  
self  
.  
eject_windows  
()
```

```
except  
:  
  
pass  
  
if
```

```
ismacos  
:  
  
try  
:
```

```
self
.
eject_osx
()

except
:

pass

self
.
_main_prefix

=

self
.
_card_a_prefix

=

self
.
_card_b_prefix

=

None

def

linux_post_yank
(
self
):
```

```
self
.  
_linux_mount_map  
  
=  
  
{}
```

[\[docs\]](#)

```
def  
  
post_yank_cleanup  
(  
    self  
):
```

```
if  
  
islinux  
:
```

```
try  
:  
  
    self  
    .  
    linux_post_yank  
    ()
```

```
except  
:
```

```
import
```

traceback

traceback

.
print_exc
()

self

.
_main_prefix

=

self

.
_card_a_prefix

=

self

.
_card_b_prefix

=

None

def

get_main_ebook_dir
(
self
,

for_upload
=

False

):


```
return

self
.
EBOOK_DIR_MAIN

def

get_carda_ebook_dir
(
self
,

for_upload
=
False
):

return

self
.
EBOOK_DIR_CARD_A

def

get_cardb_ebook_dir
(
self
,

for_upload
=
False
):

return
```

```
self
.  
EBOOK_DIR_CARD_B
```

```
def
```

```
_sanity_check  
(  
self  
,  
  
on_card  
,  
  
files  
):
```

```
from
```

```
calibre.devices.utils
```

```
import
```

```
sanity_check
```

```
sanity_check  
(  
on_card  
,
```

```
files  
,
```

```
self  
.  
card_prefix  
()),
```

```
self
.
free_space
())

def

get_dest_dir
(
prefix
,

candidates
):

if

isinstance
(
candidates
,

string_or_bytes
):

candidates

=

[
candidates
]

if

not

candidates
```

```
:
```

```
candidates
```

```
=
```

```
[  
"  
]
```

```
candidates
```

```
=
```

```
[
```

```
((  
os  
.  
path  
.  
join  
(  
prefix  
,
```

```
*  
(  
x  
.  
split  
(  
 '/'  
))))
```

```
if
```

```
x
```

```
else
```

```
prefix  
)
```

```
for
```

```
x
```

```
in
```

```
candidates  
]
```

```
existing
```

```
=
```

```
[  
x
```

```
for
```

```
x
```

```
in
```

```
candidates
```

```
if
```

```
os
```

```
.
```

```
path
```

```
.
```

```
exists
```

```
(
```

```
x
```

```
)]
```

```
if
```

not

existing

:

existing

=

candidates

return

existing

[

0

]

if

on_card

==

'carda'

:

candidates

=

self

.

get_carda_ebook_dir

(

for_upload

=

```
True
)

path

=

get_dest_dir
(
self
.
_card_a_prefix
,

candidates
)

elif

on_card

==

'cardb'
:

candidates

=

self
.
get_cardb_ebook_dir
(
for_upload
=
True
)
)
```

```
path

=

get_dest_dir
(
self
.
_card_b_prefix
,

candidates
)

else
:

candidates

=

self
.
get_main_ebook_dir
(
for_upload
=
True
)

path

=

get_dest_dir
(
self
.
_main_prefix
,
```



```
candidates
)
```

```
return
```

```
path
```

```
\[docs\]
```

```
def
```

```
sanitize_callback
```

```
(
    self
    ,
```

```
    path
):
```

```
'''
```

```
    Callback to allow individual device drivers to override the path sanitization
```

```
    used by :meth:`create_upload_path`.
```

```
'''
```

```
return
```

```
sanitize
```

```
(
    path
)
```

[\[docs\]](#)

def

filename_callback

(
self
,

default

,

mi

):

'''

Callback to allow drivers to change the default file name

set by :meth:`create\_upload\_path`.

'''

return

default

[\[docs\]](#)

def

sanitize_path_components

(
self
,

components

):

```
'''
    Perform any device specific sanitization on the path components
    for files to be uploaded to the device
'''
```

```
return
```

```
components
```

```
\[docs\]
```

```
def
```

```
get_annotations
```

```
(
    self
,
```

```
path_map
):
```

```
'''
    Resolve path_map to annotation_map of files found on the device
'''
```

```
return
```

```
{}
```

```
\[docs\]
```

```
def
```

```
add_annotation_to_library
(
    self
,
    db
,
    db_id
,
    annotation
):

'''
    Add an annotation to the calibre library
'''

pass

def
create_upload_path
(
    self
,
    path
,
    mdata
,
    fname
,
    create_dirs
=
```

```
True
):

from

calibre.devices.utils

import

create_upload_path

settings

=

self
.
settings
()

filepath

=

create_upload_path
(
mdata
,
fname
,

self
.
save_template
(),

self
.
sanitize_callback
```

```
,

prefix_path
=
os
.
path
.
abspath
(
path
),

maxlen
=
self
.
MAX_PATH_LEN
,

use_subdirs
=
self
.
SUPPORTS_SUB_DIRS

and

settings
.
use_subdirs
,

news_in_folder
=
self
.
NEWS_IN_FOLDER
,
```

```
filename_callback
```

```
=
```

```
self
```

```
.
```

```
filename_callback
```

```
,
```

```
sanitize_path_components
```

```
=
```

```
self
```

```
.
```

```
sanitize_path_components
```

```
)
```

```
filedir
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
dirname
```

```
(
```

```
filepath
```

```
)
```

```
if
```

```
create_dirs
```

```
and
```

```
not
```

```
os
.  
path  
.  
exists  
(  
filedir  
):
```

```
os  
.  
mkdirs  
(  
filedir  
)
```

```
return
```

```
filepath
```

```
def
```

```
create_annotations_path  
(  
self  
,  
  
mdata  
,  
  
device_path  
=  
None  
):
```

```
return
```



```
self
    .
    create_upload_path
    (
    os
    .
    path
    .
    abspath
    (
    '/<storage>'
    ),
    mdata
    ,
    'x.bookmark'
    ,
    create_dirs
    =
    False
    )
```

Source code for calibre.devices.usbms.driver

```
__license__
=
'GPL v3'

__copyright__
=
'2009, John Schember <john at nachtimwald.com>'

__docformat__
=
'restructuredtext en'

'''
Generic USB Mass storage device driver. This is not a complete stand alone
driver. It is intended to be subclassed with the relevant parts implemented
for a particular device.
'''

import
os
```

```
,  
  
time  
,  
  
json  
,  
  
shutil  
  
from  
  
itertools  
  
import  
  
cycle  
  
  
  
from  
  
calibre.constants  
  
import  
  
numeric_version  
,  
  
ismacos  
  
from  
  
calibre  
  
import  
  
prints  
,  
  
isbytestring  
,  
  
fsync
```

```
from
```

```
calibre.constants
```

```
import
```

```
filesystem_encoding
```

```
,
```

```
DEBUG
```

```
from
```

```
calibre.devices.usbms.cli
```

```
import
```

```
CLI
```

```
from
```

```
calibre.devices.usbms.device
```

```
import
```

```
Device
```

```
from
```

```
calibre.devices.usbms.books
```

```
import
```

```
BookList
```

```
,
```

```
Book
```

```
from
```

```
calibre.ebooks.metadata.book.json_codec
```

```
import
JsonCodec

from
polyglot.builtins

import

intervalues
,

string_or_bytes

def

debug_print
(
*
args
,
**
kw
):

base_time

=

getattr
(
debug_print
,

'base_time'
,

None
```

```
)  
  
if  
  
base_time  
  
is  
  
None  
:  
  
debug_print  
.  
base_time  
  
=  
  
base_time  
  
=  
  
time  
.  
monotonic  
()  
  
if  
  
DEBUG  
:  
  
prints  
(  
'DEBUG:  
%6.1f  
'  
%  
(  
time
```

```
.  
monotonic  
(  
-  
base_time  
)  
  
*  
args  
,  
  
**  
kw  
)
```

```
def
```

```
safe_walk  
(  
top  
,  
  
topdown  
=  
True  
,  
  
onerror  
=  
None  
,  
  
followlinks  
=  
False  
,  
  
maxdepth  
=  
128  
):
```

' A replacement for os.walk that does not die when it encounters undecodeable filenames in a linux filesystem'

if

maxdepth

<

0

:

return

islink

,

join

,

isdir

=

os

.

path

.

islink

,

os

.

path

.

join

,


```
os
.
path
.
isdir

# We may not have read permission for top, in which case we can't

# get a list of the files the directory contains. os.path.walk

# always suppressed the exception then, rather than blow up for a

# minor reason when (say) a thousand readable directories are still

# left to visit. That logic is copied here.

try
:

names

=

os
.
listdir
(
top
)

except

OSError
```

as

err

:

if

onerror

is

not

None

:

onerror

(

err

)

return

dirs

,

nondirs

=

[],

[]

for

name

```
in
```

```
names
```

```
:
```

```
if
```

```
isinstance
```

```
(
```

```
name
```

```
,
```

```
bytes
```

```
):
```

```
try
```

```
:
```

```
name
```

```
=
```

```
name
```

```
.
```

```
decode
```

```
(
```

```
filesystem_encoding
```

```
)
```

```
except
```

```
UnicodeDecodeError
```

```
:
```

```
debug_print
```

```
(
```

```
'Skipping undecodeable file:
```

```
%r  
,
```

```
%
```

```
name  
)
```

```
continue
```

```
if
```

```
isdir  
(  
join  
(  
top  
,
```

```
name  
)):
```

```
dirs  
.  
append  
(  
name  
)
```

```
else  
:
```

```
nondirs  
.  
append  
(  
name  
)
```

```
if
```

```
topdown  
:
```

```
yield
```

```
top  
,
```

```
dirs  
,
```

```
nondirs
```

```
for
```

```
name
```

```
in
```

```
dirs  
:
```

```
new_path
```

```
=
```

```
join  
(  
top  
,
```

```
name  
)
```

```
if  
  
followlinks
```

```
or
```

```
not
```

```
islink  
(  
new_path  
):
```

```
yield from
```

```
safe_walk  
(  
new_path  
,
```

```
topdown  
,
```

```
onerror  
,
```

```
followlinks  
,
```

```
maxdepth  
-  
1  
)
```

```
if
```

```
not
```

```
topdown  
:
```

yield

top

,

dirs

,

nondirs

CLI must come before Device as it implements the CLI functions that

are inherited from the device interface in Device.

[\[docs\]](#)

class

USBMS

(

CLI

,

Device

):

'''

The base class for all USBMS devices. Implements the logic for

sending/getting/updating metadata/caching metadata/etc.

'''

description

```
=
```

```
-  
(  
'Communicate with an e-book reader.'  
)
```

```
author
```

```
=
```

```
'John Schember'
```

```
supported_platforms
```

```
=
```

```
[  
'windows'  
,  
  
'osx'  
,  
  
'linux'  
]
```

```
# Store type instances of BookList and Book. We must do this because
```

```
# a) we need to override these classes in some device drivers, and
```

```
# b) the classmethods seem only to see real attributes declared in the
```

```
# class, not attributes stored in the class
```


booklist_class

=

BookList

book_class

=

Book

FORMATS

=

[]

CAN_SET_METADATA

=

[]

METADATA_CACHE

=

'metadata.calibre'

DRIVEINFO

=

'driveinfo.calibre'

```
SCAN_FROM_ROOT
```

```
=
```

```
False
```

```
def
```

```
    _update_driveinfo_record
```

```
(
```

```
    self
```

```
,
```

```
    dinfo
```

```
,
```

```
    prefix
```

```
,
```

```
    location_code
```

```
,
```

```
    name
```

```
=
```

```
None
```

```
):
```

```
from
```

```
    calibre.utils.date
```

```
import
```

```
    now
```

```
,
```

```
    isoformat
```

```
import
uuid

if
not
isinstance
(
dinfo
,
dict
):

dinfo
=
{}

if
dinfo
.
get
(
'device_store_uuid'
,
None
)
is
None
:
```

```
dinfo
[
    'device_store_uuid'
]
```

```
=
```

```
str
(
    uuid
    .
    uuid4
    ())
```

```
if
```

```
dinfo
.
get
(
    'device_name'
    ,
```

```
None
)
```

```
is
```

```
None
:
```

```
dinfo
[
    'device_name'
]
```

```
=
```

```
self
```

```
.
```

```
get_gui_name  
()
```

```
if
```

```
name
```

```
is
```

```
not
```

```
None
```

```
:
```

```
dinfo
```

```
[  
    'device_name'  
]
```

```
=
```

```
name
```

```
dinfo
```

```
[  
    'location_code'  
]
```

```
=
```

```
location_code
```

```
dinfo
```

```
[  
    'last_library_uuid'  
]
```

```
=
```

```
getattr
(
self
,
'current_library_uuid'
,
None
)
```

```
dinfo
[
'calibre_version'
]
```

```
=
```

```
[''
```

```
.'
```

```
join
```

```
([
```

```
str
```

```
(
```

```
i
```

```
)
```

```
for
```

```
i
```

```
in
```

```
numeric_version
])
```

```
dinfo
[
'date_last_connected'
]
```

```
=  
  
isoformat  
(  
    now  
)  
  
dinfo  
[  
    'prefix'  
]  
  
=  
  
prefix  
.  
replace  
(  
    '  
    \\  
    '  
    ,  
    '/'  
)  
  
return  
  
dinfo  
  
def  
_update_driveinfo_file  
(  
    self  
    ,  
    prefix  
    ,
```

```
location_code
```

```
,
```

```
name
```

```
=
```

```
None
```

```
):
```

```
from
```

```
calibre.utils.config
```

```
import
```

```
from_json
```

```
,
```

```
to_json
```

```
if
```

```
os
```

```
.
```

```
path
```

```
.
```

```
exists
```

```
(
```

```
os
```

```
.
```

```
path
```

```
.
```

```
join
```

```
(
```

```
prefix
```

```
,
```

```
self
```

```
.
```

```
DRIVEINFO
```

```
)):
```


with

lopen

(

os

.

path

.

join

(

prefix

,

self

.

DRIVEINFO

),

'rb'

)

as

f

:

try

:

driveinfo

=

json

.

loads

(

f

.

```
read
(),

object_hook
=
from_json
)

except
:

driveinfo

=

None

driveinfo

=

self
.
_update_driveinfo_record
(
driveinfo
,

prefix
,

location_code
,

name
)

data
```

```
=
```

```
json
```

```
.
```

```
 dumps
```

```
(
```

```
 driveinfo
```

```
,
```

```
 default
```

```
=
```

```
 to_json
```

```
)
```

```
if
```

```
not
```

```
 isinstance
```

```
(
```

```
 data
```

```
,
```

```
 bytes
```

```
):
```

```
 data
```

```
=
```

```
 data
```

```
.
```

```
 encode
```

```
(
```

```
 'utf-8'
```

```
)
```

```
with
```

```
    lopen
    (
    os
    .
    path
    .
    join
    (
    prefix
    ,

    self
    .
    DRIVEINFO
    ),

    'wb'
    )

    as

    f
    :

    f
    .
    write
    (
    data
    )

    fsync
    (
    f
    )

    else
    :
```

```
driveinfo

=

self
.
_update_driveinfo_record
({},

prefix
,

location_code
,

name
)

data

=

json
.
dumps
(
driveinfo
,

default
=
to_json
)

if

not

isinstance
(
data
```

```
,  
  
bytes  
):  
  
data  
  
=  
  
data  
.  
encode  
(  
    'utf-8'  
)  
  
with  
  
    lopen  
    (  
        os  
        .  
        path  
        .  
        join  
        (  
            prefix  
            ,  
  
            self  
            .  
            DRIVEINFO  
        ),  
  
        'wb'  
    )  
  
as  
  
f  
:
```

```
f
.
write
(
data
)
```

```
fsync
(
f
)
```

```
return
```

```
driveinfo
```

```
\[docs\]
```

```
def
```

```
get_device_information
(
self
,

```

```
end_session
=
```

```
True
):
```

```
self
```

```
.
report_progress
(
1.0
```


' If you device has a "Reset to factory defaults" type'

' of setting somewhere, use it. Underlying error:

%s

,

)

%

e

)

from

e

if

self

.

_main_prefix

is

not

None

:

try

:

self

.

driveinfo

[

```

'main'
]

=

self
.
_update_driveinfo_file
(
self
.
_main_prefix
,

'main'
)

except

PermissionError

as

e
:

if

ismacos
:

raise

PermissionError
(
-
(
'Permission was denied by macOS trying to access files in the main memory of'

```

' your device. You will need to grant permission explicitly by looking under'

' System Preferences > Security and Privacy > Privacy > Files and Folders.'

' Underlying error:

%s

'

)

%

e

)

from

e

raise_os_error

(

e

)

except

OSError

as

e

:

raise_os_error

(

```
e
)

try
:

if

self
.
_card_a_prefix

is

not

None
:

self
.
driveinfo
[
'A'
]

=

self
.
_update_driveinfo_file
(
self
.
_card_a_prefix
,
'A'
)
```

```
if
self
.
_card_b_prefix

is

not

None
:

self
.
driveinfo
[
'B'
]

=

self
.
_update_driveinfo_file
(
self
.
_card_b_prefix
,
'B'
)

except

OSError

as
```

```
e
:
```

```
raise
```

```
OSError
```

```
(
```

```
—  
(
```

```
'Failed to access files on the SD card in your'
```

```
' device. This can happen for many reasons. The SD card may be'
```

```
' corrupted, it may be too large for your device, it may be'
```

```
' write-protected, etc. Try a different SD card, or reformat'
```

```
' your SD card using the FAT32 filesystem. Also make sure'
```

```
' there are not too many files in the root of your SD card.'
```

```
' Underlying error:
```

```
%s
```

```
,
```

```
)
```

```
%
```

```
e
```

```
)
```

```
return
```

```
(
```

```
self
```

```
.  
get_gui_name  
(),
```

```
""
```

```
,
```

```
""
```

```
,
```

```
""
```

```
,
```

```
self
```

```
.  
driveinfo  
)
```

```
\[docs\]
```

```
def
```

```
set_driveinfo_name
```

```
(  
self
```

```
,
```

```
location_code
```

```
,
```

```
name
```

```
):
```

```
if
```

```
location_code
```

```
==
```

```
'main'
```

```
:
```

```
self
.  
_update_driveinfo_file  
(  
self  
.  
_main_prefix  
,  
  
location_code  
,  
  
name  
)
```

elif

```
location_code
```

```
==
```

```
'A'
```

```
:
```

```
self  
.  
_update_driveinfo_file  
(  
self  
.  
_card_a_prefix  
,  
  
location_code  
,  
  
name  
)
```


elif

location_code

==

'B'

:

self

.

_update_driveinfo_file

(

self

.

_card_b_prefix

,

location_code

,

name

)

def

formats_to_scan_for

(

self

):

return

set

(

self

.

settings

()

```
.  
format_map  
)
```

```
|
```

```
set  
(  
self  
. FORMATS  
)
```

```
def
```

```
is_allowed_book_file
```

```
(  
self
```

```
,
```

```
filename
```

```
,
```

```
path
```

```
,
```

```
prefix
```

```
):
```

```
return
```

```
True
```

```
\[docs\]
```

```
def
```

```
books
(
self
,

oncard
=
None
,

end_session
=
True
):

from

calibre.ebooks.metadata.meta

import

path_to_ext


debug_print
(
'USBMS: Fetching list of books from device. Device='
,

self
.
__class__
.
__name__
,

'oncard='
,
```

```
oncard
)

dummy_bl

=

self
.
booklist_class
(
None
,
None
,
None
)

if

oncard

==

'carda'

and

not

self
.
_card_a_prefix
:

self
```

```
.
report_progress
(
1.0
,
_
(
'Getting list of books on device...'
))
```

return

dummy_bl

elif

oncard

==

'cardb'

and

not

self

```
.
_card_b_prefix
:
```

self

```
.
report_progress
(
1.0
,
```

_

```
(  
'Getting list of books on device...'  
)
```

```
return
```

```
dummy_bl
```

```
elif
```

```
oncard
```

```
and
```

```
oncard
```

```
!=
```

```
'carda'
```

```
and
```

```
oncard
```

```
!=
```

```
'cardb'
```

```
:
```

```
self
```

```
.
```

```
report_progress
```

```
(
```

```
1.0
```

```
,
```

```
-
```

```
(  
'Getting list of books on device...'  
)
```

return

dummy_bl

prefix

=

self

._card_a_prefix

if

oncard

==

'carda'

else

\

self

._card_b_prefix

if

oncard

==

'cardb'

\

else

self

```
.
_main_prefix

ebook_dirs

=

self
.
get_carda_ebook_dir
()

if

oncard

==

'carda'

else
\

self
.
EBOOK_DIR_CARD_B

if

oncard

==

'cardb'

else
\

self
.
get_main_ebook_dir
```



```
()

debug_print
(
'USBMS: dirs are:'
,

prefix
,

ebook_dirs
)


# get the metadata cache

bl

=

self
.
booklist_class
(
oncard
,

prefix
,

self
.
settings
)

need_sync

=
```

```
self
.  
parse_metadata_cache  
(  
bl  
,  
  
prefix  
,  
  
self  
.  
METADATA_CACHE  
)
```

make a dict cache of paths so the lookup in the loop below is faster.

```
bl_cache
```

```
=
```

```
{}
```

```
for
```

```
idx
```

```
,
```

```
b
```

```
in
```

```
enumerate
```

```
(
```

```
bl
```

```
):
```

```
bl_cache
[
b
.
lpath
]

=

idx


all_formats

=

self
.
formats_to_scan_for
()


def

update_booklist
(
filename
,

path
,

prefix
):

changed

=

False
```

if

```
path_to_ext  
(  
    filename  
)
```

in

```
all_formats
```

and

self

```
.  
is_allowed_book_file  
(  
    filename  
,  
  
    path  
,  
  
    prefix  
):
```

try
:

```
lpath
```

```
=
```

os

```
.  
path
```

```
.  
join  
(
```

```
path
,

filename
)
.
partition
(
self
.
normalize_path
(
prefix
))[
2
]

if

lpath
.
startswith
(
os
.
sep
):

lpath

=

lpath
[
len
(
os
.
sep
):]
```

```
lpath
```

```
=
```

```
lpath
```

```
.
```

```
replace
```

```
(
```

```
'
```

```
\\
```

```
'
```

```
,
```

```
'/'
```

```
)
```

```
idx
```

```
=
```

```
bl_cache
```

```
.
```

```
get
```

```
(
```

```
lpath
```

```
,
```

```
None
```

```
)
```

```
if
```

```
idx
```

```
is
```

```
not
```

```
None
```

```
:
```

```
bl_cache
```

```
[
```

```
lpath
```

```
]
```

```
=
```

```
None
```

```
if
```

```
self
```

```
.
```

```
update_metadata_item
```

```
(
```

```
bl
```

```
[
```

```
idx
```

```
]):
```

```
# print 'update_metadata_item returned true'
```

```
changed
```

```
=
```

```
True
```

```
else
```

```
:
```

```
if
```

```
bl
```

```
.
```

```
add_book
```

```
(
self
.
book_from_path
(
prefix
,

lpath
),

replace_metadata
=
False
):

changed

=

True

except
:

# Probably a filename encoding error

import

traceback

traceback
.
print_exc
()

return
```


changed

if

isinstance

(
ebook_dirs
,

string_or_bytes
):

ebook_dirs

=

[
ebook_dirs
]

for

ebook_dir

in

ebook_dirs
:

ebook_dir

=

self

.
path_to_unicode
(
ebook_dir

```
)

if

self
.
SCAN_FROM_ROOT
:

ebook_dir

=

self
.
normalize_path
(
prefix
)

else
:

ebook_dir

=

self
.
normalize_path
(

os
.
path
.
join
(
```

```
prefix
',
*
(
ebook_dir
.
split
(
'/'
)))

if

ebook_dir

else

prefix
)

debug_print
(
'USBMS: scan from root'
,
self
.
SCAN_FROM_ROOT
,
ebook_dir
)

if

not

os
.
```

```
path
.
exists
(
ebook_dir
):

continue

# Get all books in the ebook_dir directory

if

self
.
SUPPORTS_SUB_DIRS

or

self
.
SUPPORTS_SUB_DIRS_FOR_SCAN
:

# build a list of files to check, so we can accurately report progress

flist
=
[]

for

path
,
```

```
dirs
',

files

in

safe_walk
(
ebook_dir
):

for

filename

in

files
:

if

filename

!=

self
.
METADATA_CACHE
:

flist
.
append
({
'filename'
:

self
```

```
.  
path_to_unicode  
(  
filename  
),
```

```
'path'
```

```
:  
self
```

```
.  
path_to_unicode  
(  
path  
))
```

```
for
```

```
i
```

```
,
```

```
f
```

```
in
```

```
enumerate  
(  
flist  
):
```

```
self
```

```
.  
report_progress
```

```
(
```

```
i
```

```
/
```

```
float
```

```
(
```

```
len
```

```
(
```

```
flist
```

```
)),  
  
_  
(  
'Getting list of books on device...'  
)
```

changed

=

update_booklist

```
(  
f  
[  
'filename'  
],
```

```
f  
[  
'path'  
],
```

```
prefix  
)
```

if

```
changed  
:
```

need_sync

=

True

```
else  
:
```

```
paths
```

```
=
```

```
os
```

```
.
```

```
listdir
```

```
(
```

```
ebook_dir
```

```
)
```

```
for
```

```
i
```

```
,
```

```
filename
```

```
in
```

```
enumerate
```

```
(
```

```
paths
```

```
):
```

```
self
```

```
.
```

```
report_progress
```

```
((
```

```
i
```

```
+
```

```
1
```

```
)
```

```
/
```

```
float
```

```
(
```

```
len
```



```
(
paths
)),

-
(
'Getting list of books on device...'
))

changed

=

update_booklist
(
self
.
path_to_unicode
(
filename
),

ebook_dir
,

prefix
)

if

changed
:

need_sync

=

True
```

Remove books that are no longer in the filesystem. Cache contains

indices into the booklist if book not in filesystem, None otherwise

Do the operation in reverse order so indices remain valid

for

idx

in

sorted

(

intervalues

(

bl_cache

),

reverse

=

True

,

key

=

lambda

x

:

-

1

if

x

is

None

else

x
):

if

idx

is

not

None

:

need_sync

=

True

del

bl
[
idx
]

debug_print

(
'USBMS: count found in cache:
%d
, count of files in metadata:
%d

```

, need_sync:
%s
'

%

(
len
(
bl_cache
),

len
(
bl
),

need_sync
))

if

need_sync
:

# self.count_found_in_bl != len(bl) or need_sync:

if

oncard

==

'cardb'
:

self
.
sync_booklists

```

```
((
None
,

None
,

bl
))

elif

oncard

==

'carda'
:

self
.
sync_booklists
((
None
,

bl
,

None
))

else
:

self
.
sync_booklists
((
```

```
bl
,

None
,

None
))

self
.
report_progress
(
1.0
,

_
(
'Getting list of books on device...'
))

debug_print
(
'USBMS: Finished fetching list of books from device. oncard='
,

oncard
)

return

bl

\[docs\]

def

upload_books
```

```
(
self
,

files
,

names
,

on_card
=
None
,

end_session
=
True
,

metadata
=
None
):

debug_print
(
'USBMS: uploading
%d
books'
%
(
len
(
files
)))

path
```

```
=  
  
self  
.  
_sanity_check  
(  
on_card  
,  
  
files  
)
```

```
paths
```

```
=
```

```
[]
```

```
names
```

```
=
```

```
iter  
(  
names  
)
```

```
metadata
```

```
=
```

```
iter  
(  
metadata  
)
```

```
for
```



```
i
',

infile

in

enumerate
(
files
):

mdata
',

fname

=

next
(
metadata
),

next
(
names
)

filepath

=

self
.
normalize_path
(
self
.
create_upload_path
```

```
(
    path
    ,
    mdata
    ,
    fname
))

if
not
hasattr
(
    infile
    ,
    'read'
):

    infile
    =
    self
    .
    normalize_path
    (
        infile
    )

    filepath
    =
    self
    .
    put_file
```

```
(  
infile  
,  
  
filepath  
,  
  
replace_file  
=  
True  
)
```

```
paths  
.  
append  
(  
filepath  
)
```

```
try  
:
```

```
self  
.  
upload_cover  
(  
os  
.  
path  
.  
dirname  
(  
filepath  
)  
,
```

```
os  
.  
path  
.  

```

```
splitext
(
os
.
path
.
basename
(
filepath
))[
0
],

mdata
,

filepath
)

except
:

# Failure to upload cover is not catastrophic

import

traceback

traceback
.
print_exc
()

self

report_progress
((
```

```
i
+
1
)

/

float
(
len
(
files
)),

-
(
'Transferring books to device...'
))

self
.
report_progress
(
1.0
,

-
(
'Transferring books to device...'
))

debug_print
(
'USBMS: finished uploading
%d
books'
%
(
len
(
```

```
files  
)))
```

```
return
```

```
list  
(  
zip  
(  
paths  
,  
  
cycle  
([  
on_card  
]))
```

```
\[docs\]
```

```
def
```

```
upload_cover  
(  
self  
,  
  
path  
,  
  
filename  
,  
  
metadata  
,  
  
filepath  
):
```

```
'''
```

Upload book cover to the device. Default implementation does nothing.

:param path: The full path to the folder where the associated book is located.

:param filename: The name of the book file without the extension.

:param metadata: metadata belonging to the book. Use metadata.thumbnail

for cover

:param filepath: The full path to the e-book file

'''

pass

[\[docs\]](#)

def

add_books_to_metadata

(

self

,

locations

,

metadata

,

booklists

):

debug_print

(

'USBMS: adding metadata for

%d

```
books'  
%  
(  
len  
(  
metadata  
)))
```

```
metadata
```

```
=
```

```
iter  
(  
metadata  
)
```

```
locations
```

```
=
```

```
tuple  
(  
locations  
)
```

```
for
```

```
i
```

```
,
```

```
location
```

```
in
```

```
enumerate  
(  
locations  
):
```



```
self
.
report_progress
((
i
+
1
)

/

float
(
len
(
locations
)),

-
(
'Adding books to device metadata listing...'
))

info

=

next
(
metadata
)

blist

=

2

if
```

```
location
[
1
]
```

```
==
```

```
'cardb'
```

```
else
```

```
1
```

```
if
```

```
location
[
1
]
```

```
==
```

```
'carda'
```

```
else
```

```
0
```

```
# Extract the correct prefix from the pathname. To do this correctly,
```

```
# we must ensure that both the prefix and the path are normalized
```

```
# so that the comparison will work. Book's __init__ will fix up
```

```
# lpath, so we don't need to worry about that here.
```

```
path
=
self
.
normalize_path
(
location
[
0
])
```

```
if
self
.
_main_prefix
:
```

```
prefix
=
self
.
_main_prefix
```

```
if
\
```

```
path
.
startswith
(
self
.
normalize_path
(
self
.

```

```
_main_prefix
))

else

None

if

not

prefix

and

self
.
_card_a_prefix
:

prefix

=

self
.
_card_a_prefix

if
\

path
.
startswith
(
self
.
normalize_path
(
self
.
```

```
_card_a_prefix  
)
```

```
else
```

```
None
```

```
if
```

```
not
```

```
prefix
```

```
and
```

```
self
```

```
.  
_card_b_prefix  
:
```

```
prefix
```

```
=
```

```
self
```

```
.  
_card_b_prefix
```

```
if
```

```
\
```

```
path
```

```
.  
startswith
```

```
(  
self
```

```
.  
normalize_path
```

```
(  
self
```

```
.
```

```
_card_b_prefix
))

else

None

if

prefix

is

None
:

prints
(
'in add_books_to_metadata. Prefix is None!'
,

path
,

self
.
_main_prefix
)

continue

lpath

=

path
.
partition
```

```
(  
prefix  
)[  
2  
]
```

if

```
lpath  
.  
startswith  
(  
    '/'  
)
```

or

```
lpath  
.  
startswith  
(  
    '  
    \\  
    '  
)  
:
```

```
lpath
```

```
=
```

```
lpath  
[  
1  
:]
```

```
book
```

```
=
```

```
self
```

```
.
book_class
(
prefix
,

lpath
,

other
=
info
)

if

book
.
size

is

None
:

book
.
size

=

os
.
stat
(
self
.
normalize_path
(
path
))
```



```
.
st_size

b

=

booklists
[
blist
]
.
add_book
(
book
,

replace_metadata
=
True
)

if

b
:

b
.
_new_book

=

True

self

.
report_progress
(
```

```
1.0
',
_
(
'Adding books to device metadata listing...'
))
```

```
debug_print
(
'USBMS: finished adding metadata'
)
```

```
def
```

```
delete_single_book
(
self
,
path
):
```

```
os
.
unlink
(
path
)
```

```
def
```

```
delete_extra_book_files
(
self
,
path
```

```
):  
  
filepath  
  
=  
  
os  
.  
path  
.  
splitext  
(  
path  
)[  
0  
]  
  
for  
  
ext  
  
in  
  
self  
.  
DELETE_EXTS  
:  
  
for  
  
x  
  
in  
  
(  
filepath  
,  
path  
):
```

```
x
```

```
+=
```

```
ext
```

```
if
```

```
os
```

```
.
```

```
path
```

```
.
```

```
exists
```

```
(
```

```
x
```

```
):
```

```
if
```

```
os
```

```
.
```

```
path
```

```
.
```

```
isdir
```

```
(
```

```
x
```

```
):
```

```
shutil
```

```
.
```

```
rmtree
```

```
(
```

```
x
```

```
,
```

```
ignore_errors
```

```
=
```

```
True
```

```
)

else
:

os
.
unlink
(
x
)

if

self
.
SUPPORTS_SUB_DIRS
:

try
:

os
.
removedirs
(
os
.
path
.
dirname
(
path
))

except
```

```
:  
  
pass  
  
\[docs\]  
  
def  
  
delete_books  
(  
    self  
,  
  
    paths  
,  
  
    end_session  
    =  
    True  
):  
  
    debug_print  
    (  
        'USBMS: deleting  
        %d  
        books'  
        %  
        (  
            len  
            (  
                paths  
            ))  
        )  
  
    for  
  
    i  
,
```

```
path
```

```
in
```

```
enumerate
```

```
(  
paths  
):
```

```
self
```

```
.  
report_progress
```

```
((  
i  
+  
1  
)
```

```
/
```

```
float
```

```
(  
len  
(  
paths  
)),
```

```
-  
(  
'Removing books from device...'  
)
```

```
path
```

```
=
```

```
self
```

```
.  
normalize_path  
(
```

```
path
)

if

os
.
path
.
exists
(
path
):

# Delete the ebook

self
.
delete_single_book
(
path
)

self
.
delete_extra_book_files
(
path
)

self
.
report_progress
(
1.0
,
```



```
    _  
(  
    'Removing books from device...'  
))
```

```
debug_print  
(  
    'USBMS: finished deleting  
    %d  
    books'  
    %  
    (  
    len  
    (  
    paths  
    )))
```

[\[docs\]](#)

def

```
remove_books_from_metadata  
(  
    self  
    ,  
    paths  
    ,  
    booklists  
):
```

```
debug_print  
(  
    'USBMS: removing metadata for  
    %d  
    books'  
    %  
    (  
    len
```

```
(
paths
)))

for

i
,

path

in

enumerate
(
paths
):

self
.
report_progress
((
i
+
1
)

/

float
(
len
(
paths
)),

-
(
'Removing books from device metadata listing...'
))
```

```
for
```

```
bl
```

```
in
```

```
booklists
```

```
:
```

```
for
```

```
book
```

```
in
```

```
bl
```

```
:
```

```
if
```

```
path
```

```
.
```

```
endswith
```

```
(
```

```
book
```

```
.
```

```
path
```

```
):
```

```
bl
```

```
.
```

```
remove_book
```

```
(
```

```
book
```

```
)
```

```
self
```

```

    .
    report_progress
    (
    1.0
    ,

    _
    (
    'Removing books from device metadata listing...'
    ))

    debug_print
    (
    'USBMS: finished removing metadata for
    %d
    books'
    %
    (
    len
    (
    paths
    )))

    # If you override this method and you use book._new_book, then you must

    # complete the processing before you call this method. The flag is cleared

    # at the end just before the return

    \[docs\]

    def

    sync_booklists
    (
    self
    ,

```

```
booklists
,

end_session
=
True
):

debug_print
(
'USBMS: starting sync_booklists'
)

json_codec

=

JsonCodec
()

if

not

os
.
path
.
exists
(
self
.
normalize_path
(
self
.
_main_prefix
)):

```

```
os
.  
mkdirs  
(  
self  
.  
normalize_path  
(  
self  
.  
_main_prefix  
)
```

```
def
```

```
write_prefix  
(  
prefix  
,  
  
listid  
):
```

```
if
```

```
(  
prefix
```

```
is
```

```
not
```

```
None
```

```
and
```

```
len  
(
```

```
booklists
)

>

listid

and

isinstance
(
booklists
[
listid
],
self
.
booklist_class
)):

if

not

os
.
path
.
exists
(
prefix
):

os
.
makedirs
(
self
.
```

```
normalize_path
(
prefix
))

with

lopen
(
self
.
normalize_path
(
os
.
path
.
join
(
prefix
,

self
.
METADATA_CACHE
)),

'wb'
)

as

f
:

json_codec
.
encode_to_file
(
f
,
```



```
booklists
```

```
[
```

```
listid
```

```
])
```

```
fsync
```

```
(
```

```
f
```

```
)
```

```
write_prefix
```

```
(
```

```
self
```

```
.
```

```
_main_prefix
```

```
,
```

```
0
```

```
)
```

```
write_prefix
```

```
(
```

```
self
```

```
.
```

```
_card_a_prefix
```

```
,
```

```
1
```

```
)
```

```
write_prefix
```

```
(
```

```
self
```

```
.
```

```
_card_b_prefix
```

```
,
```

```
2
```

```
)
```

```
# Clear the _new_book indication, as we are supposed to be done with
```

```
# adding books at this point
```

```
for
```

```
    blist
```

```
in
```

```
    booklists
```

```
    :
```

```
if
```

```
    blist
```

```
is
```

```
not
```

```
None
```

```
    :
```

```
for
```

```
    book
```

```
in
```

```
    blist
```

```
    :
```

```
        book
```

```
.  
_new_book
```

```
=
```

```
False
```

```
self
```

```
.  
report_progress
```

```
(  
1.0  
,
```

```
_  
(  
'Sending metadata to device...'  
)
```

```
debug_print
```

```
(  
'USBMS: finished sync_booklists'  
)
```

```
@classmethod
```

```
def
```

```
build_template_regexp  
(  
cls  
):
```

```
from
```

```
calibre.devices.utils
```

import

build_template_regexp

return

build_template_regexp

(

cls

.

save_template

())

@classmethod

def

path_to_unicode

(

cls

,

path

):

if

isbytestring

(

path

):

path

=

path

```
.  
decode  
(  
    filesystem_encoding  
)  
  
return  
  
path  
  
\[docs\]  
  
@classmethod  
  
def  
  
    normalize_path  
    (  
        cls  
        ,  
        path  
    ):  
  
    'Return path with platform native path separators'  
  
    if  
  
        path  
  
        is  
  
        None  
    :
```

```
return
```

```
None
```

```
if
```

```
os
```

```
.
```

```
sep
```

```
==
```

```
,
```

```
\\
```

```
,
```

```
:
```

```
path
```

```
=
```

```
path
```

```
.
```

```
replace
```

```
(
```

```
'/'
```

```
,
```

```
,
```

```
,
```

```
\\
```

```
,
```

```
)
```

```
else
```

```
:
```

```
path
```

```
=
```

```
path
```

```
.
```

```
replace
```

```
(
```

```
,
```

```
\\
```

```
,
```

```
,
```

```
('/')
```

```
)
```

```
return
```

```
cls
```

```
.
```

```
path_to_unicode
```

```
(
```

```
path
```

```
)
```

```
@classmethod
```

```
def
```

```
parse_metadata_cache
```

```
(
```

```
cls
```

```
,
```

```
bl
```

```
,
```

```
prefix
```

```
,
```

```
name
```

```
):
```

```
json_codec
```

```
=
```

```
JsonCodec  
()
```

```
need_sync
```

```
=
```

```
False
```

```
cache_file
```

```
=
```

```
cls
```

```
.
```

```
normalize_path
```

```
(
```

```
os
```

```
.
```

```
path
```

```
.
```

```
join
```

```
(
```

```
prefix
```

```
,
```

```
name
```

```
))
```

```
if
```

```
os
```

```
.
```

```
access
```

```
(
```



```
cache_file
,

os
.
R_OK
):

try
:

with

lopen
(
cache_file
,

'rb'
)

as

f
:

json_codec
.
decode_from_file
(
f
,

bl
,

cls
.
book_class
,
```

```
prefix  
)
```

```
except  
:
```

```
import
```

```
traceback
```

```
traceback  
.  
print_exc  
(
```

```
bl
```

```
=
```

```
[]
```

```
need_sync
```

```
=
```

```
True
```

```
else  
:
```

```
need_sync
```

```
=
```

```
True
```

```
return
```

```
need_sync
```

```
@classmethod
```

```
def
```

```
update_metadata_item
```

```
(
```

```
cls
```

```
,
```

```
book
```

```
):
```

```
changed
```

```
=
```

```
False
```

```
size
```

```
=
```

```
os
```

```
.
```

```
stat
```

```
(
```

```
cls
```

```
.
```

```
normalize_path
```

```
(
```

```
book
```

```
.
```

```
path
))
.
st_size

if

size

!=

book
.
size
:

changed

=

True

mi

=

cls
.
metadata_from_path
(
book
.
path
)

book
.
smart_update
(
```

```
mi
)

book
.
size

=

size

return

changed


@classmethod


def

metadata_from_path
(
cls
,
path
):

return

cls
.
metadata_from_formats
([
path
])
```

```
@classmethod
```

```
def
```

```
    metadata_from_formats
```

```
    (
```

```
        cls
```

```
    ,
```

```
        fmts
```

```
    ):
```

```
    from
```

```
        calibre.ebooks.metadata.meta
```

```
    import
```

```
        metadata_from_formats
```

```
    from
```

```
        calibre.customize.ui
```

```
    import
```

```
        quick_metadata
```

```
    with
```

```
        quick_metadata
```

```
    :
```

```
    return
```

```
        metadata_from_formats
```

```
    (
```

```
fmts
,

force_read_metadata
=
True
,

pattern
=
cls
.
build_template_regexp
())

@classmethod

def

book_from_path
(
cls
,

prefix
,

lpath
):

from

calibre.ebooks.metadata.book.base

import

Metadata
```

```
if
    cls
    .
    settings
    ()
    .
    read_metadata

or

cls
.
MUST_READ_METADATA
:

mi

=

cls
.
metadata_from_path
(
    cls
    .
    normalize_path
    (
        os
        .
        path
        .
        join
        (
            prefix
            ,
            lpath
        )))
```



```
else
:

from

calibre.ebooks.metadata.meta

import

metadata_from_filename

mi

=

metadata_from_filename
(
cls
.
normalize_path
(
os
.
path
.
basename
(
lpath
)),
cls
.
build_template_regexp
())

if

mi
```

is

None

:

mi

=

Metadata

(

os

.

path

.

splitext

(

os

.

path

.

basename

(

lpath

))[

0

],

[

-

(

'Unknown'

))

size

=

os

```
.
stat
(
cls
.
normalize_path
(
os
.
path
.
join
(
prefix
,
lpath
)))
.
st_size
```

book

=

cls

```
.
book_class
(
prefix
,
```

```
lpath
,
```

```
other
=
```

mi

,

size

=

size
)

return

book

Source code for calibre.ebooks.metadata.book.base

```
#!/usr/bin/env python
```

```
__license__
```

```
=
```

```
'GPL v3'
```

```
__copyright__
```

```
=
```

```
'2010, Kovid Goyal <kovid@kovidgoyal.net>'
```

```
__docformat__
```

```
=
```

```
'restructuredtext en'
```

```
import
```

```
copy
```

```
,
```

```
traceback
```

```
from
```

```
calibre

import

prints

from

calibre.constants

import

DEBUG

from

calibre.ebooks.metadata.book

import

(
    SC_COPYABLE_FIELDS
    ,

    SC_FIELDS_COPY_NOT_NULL
    ,

    STANDARD_METADATA_FIELDS
    ,

    TOP_LEVEL_IDENTIFIERS
    ,

    ALL_METADATA_FIELDS
)

from

calibre.library.field_metadata

import
```

```
FieldMetadata
```

```
from
```

```
calibre.utils.icu
```

```
import
```

```
sort_key
```

```
from
```

```
polyglot.builtins
```

```
import
```

```
iteritems
```

```
,
```

```
string_or_bytes
```

```
# Special sets used to optimize the performance of getting and setting
```

```
# attributes on Metadata objects
```

```
SIMPLE_GET
```

```
=
```

```
frozenset
```

```
(
```

```
STANDARD_METADATA_FIELDS
```

```
-
```

```
TOP_LEVEL_IDENTIFIERS
```

```
)
```

```
SIMPLE_SET
```

```
=
```

```
frozenset
(
SIMPLE_GET
```

```
-
```

```
{
'identifiers'
})
```

```
def
```

```
human_readable
```

```
(
size
,
```

```
precision
```

```
=
2
):
```

```
""" Convert a size in bytes into megabytes """
```

```
ans
```

```
=
```

```
size
```

```
/
```

```
(
1024
*
1024
)
```

```
if
```


ans

<

0.1

:

return

'<0.1MB'

return

(

'%.'

+

str

(

precision

)

+

'f'

+

'MB'

)

%

ans

NULL_VALUES

=

{

```
'user_metadata'
```

```
:
```

```
{},
```

```
'cover_data'
```

```
:
```

```
(
```

```
None
```

```
,
```

```
None
```

```
),
```

```
'tags'
```

```
:
```

```
[],
```

```
'identifiers'
```

```
:
```

```
{},
```

```
'languages'
```

```
:
```

```
[],
```

```
'device_collections'
```

```
:
```

```
[],
```

```
'author_sort_map'
```

```
:
```

```
{},
```

```
'authors'
```

```
:
```

```
[
```

```
-
```

```
(
```

```
'Unknown'
```

```
)],
```

```
'author_sort'
```

```
:
```

```
-
```

```
(
```

```
'Unknown'
```

```
),
```

```
'title'
```

```
:
```

```
-
```

```
(
```

```
'Unknown'
```

```
),
```

```
'user_categories'
```

```
:
```

```
{},

'author_link_map'

:

},

'language'

:

'und'

}

field_metadata

=

FieldMetadata

()

def

reset_field_metadata

():

global

field_metadata

field_metadata

=
```

```
FieldMetadata
```

```
()
```

```
ck
```

```
=
```

```
lambda
```

```
typ
```

```
:
```

```
icu_lower
```

```
(
```

```
typ
```

```
)
```

```
.
```

```
strip
```

```
()
```

```
.
```

```
replace
```

```
(
```

```
'.'
```

```
,
```

```
''
```

```
)
```

```
.
```

```
replace
```

```
(
```

```
'.'
```

```
,
```

```
,
```

```
''
```

```
)
```

```
cv
```

```
=
```

```
lambda
```

```
val
:
val
.
strip
()
.
replace
(
','
,
'|'
)
```

[\[docs\]](#)

class

Metadata

:

"""

A class representing all the metadata for a book. The various standard metadata fields are available as attributes of this object. You can also stick arbitrary attributes onto this object.

Metadata from custom columns should be accessed via the get() method, passing in the lookup name for the column, for example: "#mytags".

Use the :meth:`is\_null` method to test if a field is null.

This object also has functions to format fields into strings.

The list of standard metadata fields grows with time is in

:data: `STANDARD\_METADATA\_FIELDS`.

Please keep the method based API of this class to a minimum. Every method

becomes a reserved field name.

'''

`__calibre_serializable__`

`=`

True

def

`__init__`

`(`

`self`

`,`

`title`

`,`

`authors`

`=`

`(`

`-`

`(`

`'Unknown'`

`),),`

```
other
```

```
=
```

```
None
```

```
,
```

```
template_cache
```

```
=
```

```
None
```

```
,
```

```
formatter
```

```
=
```

```
None
```

```
):
```

```
'''
```

```
    @param title: title or ``_('Unknown')``
```

```
    @param authors: List of strings or []
```

```
    @param other: None or a metadata object
```

```
'''
```

```
_data
```

```
=
```

```
copy
```

```
.
```

```
deepcopy
```

```
(
```

```
NULL_VALUES
```

```
)
```

```
_data
```

```
.
```



```
pop
(
'language'
)

object
.
__setattr__
(
self
,
'_data'
,
_data
)

if
other
is
not
None
:

self
.
smart_update
(
other
)

else
:
```

```
if
```

```
title
```

```
:
```

```
self
```

```
.
```

```
title
```

```
=
```

```
title
```

```
if
```

```
authors
```

```
:
```

```
# List of strings or []
```

```
self
```

```
.
```

```
author
```

```
=
```

```
list
```

```
(
```

```
authors
```

```
)
```

```
if
```

```
authors
```

```
else
```

```
[]
```

```
# Needed for backward compatibility
```

```
self
```

```
.
```

```
authors
```

```
=
```

```
list
```

```
(
```

```
authors
```

```
)
```

```
if
```

```
authors
```

```
else
```

```
[]
```

```
from
```

```
calibre.ebooks.metadata.book.formatter
```

```
import
```

```
SafeFormat
```

```
self
```

```
.
```

```
formatter
```

```
=
```

```
SafeFormat
```

```
()
```

```
if
```

```
formatter
```

```
is
```

```
None
```

```
else
```

```
formatter
```

```
self
```

```
.
```

```
template_cache
```

```
=
```

```
template_cache
```

```
\[docs\]
```

```
def
```

```
is_null
```

```
(
```

```
self
```

```
,
```

```
field
```

```
):
```

```
'''
```

Return True if the value of field is null in this object.

'null' means it is unknown or evaluates to False. So a title of

_('Unknown') is null or a language of 'und' is null.

Be careful with numeric fields since this will return True for zero as well as None.

Also returns True if the field does not exist.

'''

```
try
:

null_val

=

NULL_VALUES
.
get
(
field
,

None
)

val

=

getattr
(
self
,

field
,
```

None

)

return

not

val

or

val

==

null_val

except

:

return

True

def

set_null

(

self

,

field

):

null_val

=

```
copy
.  
copy  
(  
    NULL_VALUES  
    .  
    get  
    (  
        field  
    ))  
  
setattr  
(  
    self  
    ,  
    field  
    ,  
    null_val  
)  
  
def  
__getattr__  
(  
    self  
    ,  
    field  
):  
  
    _data  
    =  
    object  
    .  
    __getattr__
```

```
(
self
,

'_data'
)

if

field

in

SIMPLE_GET
:

return

_data
.
get
(
field
,

None
)

if

field

in

TOP_LEVEL_IDENTIFIERS
:

return
```



```
_data
.
get
(
'identifiers'
)
.
get
(
field
,
None
)

if

field

==

'language'
:

try
:

return

_data
.
get
(
'languages'
,
[])[
0
]
```

```
except
```

```
:
```

```
return
```

```
NULL_VALUES
```

```
[
```

```
'language'
```

```
]
```

```
try
```

```
:
```

```
return
```

```
object
```

```
.
```

```
__getattr__
```

```
(
```

```
self
```

```
,
```

```
field
```

```
)
```

```
except
```

```
AttributeError
```

```
:
```

```
pass
```

```
if
```

```
field
```

in

```
_data  
[  
  'user_metadata'  
]:
```

d

=

```
_data  
[  
  'user_metadata'  
][  
  field  
]
```

val

=

```
d  
[  
  '#value#'  
]
```

if

```
d  
[  
  'datatype'  
]
```

!=

```
'composite'  
:
```

return

val

if

val

is

None

:

d

[

'#value#'

]

=

'RECURSIVE_COMPOSITE FIELD (Metadata) '

+

field

val

=

d

[

'#value#'

]

=

self

.

formatter

```
.
safe_format
(

d
[
'display'
][
'composite_template'
],

self
,

-
(
'TEMPLATE ERROR'
),

self
,

column_name
=
field
,

template_cache
=
self
.
template_cache
)
.
strip
()
```

```
return
```

```
val
```

```
if
```

```
field
```

```
.
```

```
startswith
```

```
(
```

```
'#'
```

```
)
```

```
and
```

```
field
```

```
.
```

```
endswith
```

```
(
```

```
'_index'
```

```
):
```

```
try
```

```
:
```

```
return
```

```
self
```

```
.
```

```
get_extra
```

```
(
```

```
field
```

```
[:
```

```
-
```

```
6
```

```
])
```

```
except
```

```
:
```

```
pass
```

```
raise
```

```
AttributeError
```

```
(
```

```
'Metadata object has no attribute named: '
```

```
+
```

```
repr
```

```
(
```

```
field
```

```
))
```

```
def
```

```
__setattr__
```

```
(
```

```
self
```

```
,
```

```
field
```

```
,
```

```
val
```

```
,
```

```
extra
```

```
=
```

```
None
```

```
):
```

```
_data
```

```
=
```

```
object
.  
__getattr__  
(  
self  
,  
'_data'  
)
```

if

field

in

SIMPLE_SET
:

if

val

is

None
:

val

=

copy

```
.  
copy  
(  
NULL_VALUES  
.  
get
```



```
(
field
,

None
))

_data
[
field
]

=

val

elif

field

in

TOP_LEVEL_IDENTIFIERS
:

field
,

val

=

self
.
_clean_identifier
(
field
,

val
```

```
)

identifiers

=

_data
[
'identifiers'
]

identifiers
.
pop
(
field
,
None
)

if

val
:

identifiers
[
field
]

=

val

elif

field
```

```
==
```

```
'identifiers'
```

```
:
```

```
if
```

```
not
```

```
val
```

```
:
```

```
val
```

```
=
```

```
copy
```

```
.
```

```
copy
```

```
(
```

```
NULL_VALUES
```

```
.
```

```
get
```

```
(
```

```
'identifiers'
```

```
,
```

```
None
```

```
))
```

```
self
```

```
.
```

```
set_identifiers
```

```
(
```

```
val
```

```
)
```

```
elif
```

field

==

'language'
:

langs

=

[]

if

val

and

val

.
lower
()

!=

'und'
:

langs

=

[
val
]

_data

```
[  
  'languages'  
]
```

```
=
```

```
langs
```

```
elif
```

```
field
```

```
in
```

```
_data
```

```
[  
  'user_metadata'  
]:
```

```
_data
```

```
[  
  'user_metadata'  
][  
  field  
][  
  '#value#'  
]
```

```
=
```

```
val
```

```
_data
```

```
[  
  'user_metadata'  
][  
  field  
][  
  '#extra#'  
]
```

```
=
```

```
extra
```

```
else
```

```
:
```

```
# You are allowed to stick arbitrary attributes onto this object as
```

```
# long as they don't conflict with global or user metadata names
```

```
# Don't abuse this privilege
```

```
self
```

```
.
```

```
__dict__
```

```
[
```

```
field
```

```
]
```

```
=
```

```
val
```

```
def
```

```
__iter__
```

```
(
```

```
self
```

```
):
```

```
return
```

```
iter
```

```
(  
object  
.  
__getattr__  
(  
self  
,  
  
'_data'  
)  
)
```

def

```
has_key  
(  
self  
,  
  
key  
)  
:
```

return

key

in

```
object  
.  
__getattr__  
(  
self  
,  
  
'_data'  
)  
)
```

[\[docs\]](#)

def

deepcopy

(

self

,

class_generator

=

lambda

:

Metadata

(

None

)):

''' Do not use this method unless you know what you are doing, if you

want to create a simple clone of this object, use :meth:`deepcopy\_metadata`

instead. Class_generator must be a function that returns an instance

of Metadata or a subclass of it.'''

m

=

class_generator

()

if

not

isinstance


```
(
m
,

Metadata
):

return

None

object
.
__setattr__
(
m
,
'__dict__'
,

copy
.
deepcopy
(
self
.
__dict__
))

return

m

def
deepcopy_metadata
(
self
```

```
):  
  
m  
  
=  
  
Metadata  
(  
  None  
)  
  
object  
.  
  __setattr__  
(  
  m  
,  
  '_data'  
,  
  copy  
.  
  deepcopy  
(  
  object  
.  
    __getattr__  
(  
  self  
,  
  '_data'  
)  
)  
)  
  
return  
  
m
```

```
def
get
(
self
,

field
,

default
=
None
):

try
:

return

self
.
__getattr__
(
field
)

except
AttributeError
:

return
default
```

```
def
get_extra
(
self
,

field
,

default
=
None
):

    _data

    =

    object
    .
    __getattr__
    (
    self
    ,

    '_data'
    )

    if

    field

    in

    _data
    [
    'user_metadata'
    ]:
```

```
try
:

return

_data
[
'user_metadata'
][
field
][
'#extra#'
]

except
:

return

default

raise

AttributeError
(

'Metadata object has no attribute named: '
+

repr
(
field
))

def
```

```
set
(
self
,

field
,

val
,

extra
=
None
):

self
.
__setattr__
(
field
,

val
,

extra
)
```

[\[docs\]](#)

```
def

get_identifiers
(
self
):
```

```
'''
    Return a copy of the identifiers dictionary.

    The dict is small, and the penalty for using a reference where a copy is
    needed is large. Also, we don't want any manipulations of the returned
    dict to show up in the book.
'''
```

```
ans
```

```
=
```

```
object
```

```
.
```

```
__getattr__
```

```
(
```

```
self
```

```
,
```

```
'_data'
```

```
)[
```

```
'identifiers'
```

```
]
```

```
if
```

```
not
```

```
ans
```

```
:
```

```
ans
```

```
=
```

```
{  
  
    return  
  
    copy  
    .  
    deepcopy  
    (  
    ans  
    )  
  
    def  
  
    _clean_identifier  
    (  
    self  
    ,  
  
    typ  
    ,  
  
    val  
    ):  
  
    if  
  
    typ  
    :  
  
        typ  
  
        =  
  
        ck  
        (  
        typ  
        )
```



```
if
```

```
val
```

```
:
```

```
val
```

```
=
```

```
cv
```

```
(
```

```
val
```

```
)
```

```
return
```

```
typ
```

```
,
```

```
val
```

```
\[docs\]
```

```
def
```

```
set_identifiers
```

```
(
```

```
self
```

```
,
```

```
identifiers
```

```
):
```

```
"""
```

Set all identifiers. Note that if you previously set ISBN, calling

this method will delete it.

'''

cleaned

=

```
{  
    ck  
    (  
    k  
    ):  
    cv  
    (  
    v  
    )
```

for

```
k  
,  
v
```

in

```
iteritems  
(  
    identifiers  
)
```

if

```
k
```

and

```
v  
}
```

```
object
{
  __getattribute__
  (
    self
  ,
    '_data'
  )
  [
    'identifiers'
  ]
  =
  cleaned

```

[\[docs\]](#)

def

```
set_identifier
(
  self
  ,
  typ
  ,
  val
):

```

'If val is empty, deletes identifier of type typ'

```
typ
,
val
=

```

```
self
.
_clean_identifier
(
typ
,
val
)
```

```
if
```

```
not
```

```
typ
:
```

```
return
```

```
identifiers
```

```
=
```

```
object
```

```
.
__getattr__
(
self
,

```

```
'_data'
)[
'identifiers'
]
```

```
identifiers
```

```
.
```

```
pop
(
typ
,
None
)

if

val
:

identifiers
[
typ
]

=

val

def

has_identifier
(
self
,

typ
):

identifiers

=

object
.
__getattr__
```

```
(
self
,

'_data'
)[
'identifiers'
]

return

typ

in

identifiers


# field-oriented interface. Intended to be the same as in LibraryDatabase


\[docs\]

def

standard_field_keys
(
self
):

'''

    return a list of all possible keys, even if this book doesn't have them

    '''
```

return

STANDARD_METADATA_FIELDS

[\[docs\]](#)

def

custom_field_keys

(
self
):

'''

return a list of the custom fields in this book

'''

return

iter

(
object

.
__getattr__

(
self

,

'_data'

)[
'user_metadata'
])

[\[docs\]](#)

def

```

all_field_keys
(
    self
):

'''

    All field keys known by this instance, even if their value is None

'''

_data

=

object
.
__getattr__
(
    self
,
    '_data'
)

return

frozenset
(
    ALL_METADATA_FIELDS
.
union
(
    frozenset
(
        _data
[
            'user_metadata'
        ])
    )
)

```


[\[docs\]](#)

def

metadata_for_field

(
self
,

key
):

'''

return metadata describing a standard or custom field.

'''

if

key

not

in

self

.
custom_field_keys
():

return

self

.
get_standard_metadata
(
key
,

```
make_copy
```

```
=
```

```
False
```

```
)
```

```
return
```

```
self
```

```
.
```

```
get_user_metadata
```

```
(
```

```
key
```

```
,
```

```
make_copy
```

```
=
```

```
False
```

```
)
```

```
\[docs\]
```

```
def
```

```
all_non_none_fields
```

```
(
```

```
self
```

```
):
```

```
'''
```

```
Return a dictionary containing all non-None metadata fields, including
```

```
the custom ones.
```

```
'''
```

```
result
```

```
=
```

```
{  
  
    _data  
  
    =  
  
    object  
    .  
    __getattr__  
    (  
    self  
    ,  
    '_data'  
    )  
  
    for  
    attr  
  
    in  
    STANDARD_METADATA_FIELDS  
    :  
  
        v  
  
        =  
  
        _data  
        .  
        get  
        (  
        attr  
        ,  
        None  
        )
```

```
if
```

```
v
```

```
is
```

```
not
```

```
None
```

```
:
```

```
result
```

```
[
```

```
attr
```

```
]
```

```
=
```

```
v
```

```
# separate these because it uses the self.get(), not _data.get()
```

```
for
```

```
attr
```

```
in
```

```
TOP_LEVEL_IDENTIFIERS
```

```
:
```

```
v
```

```
=
```

```
self
```

```
.
```

```
get
```

```
(
  attr
  ,
  None
)

if
  v
  is
  not
  None
:

result
[
  attr
]

=

v

for
  attr
  in
  _data
  [
    'user_metadata'
  ]:

  v
```

```
=  
  
self  
.  
get  
(  
attr  
,  
  
None  
)  
  
if  
  
v  
  
is  
  
not  
  
None  
:  
  
result  
[  
attr  
]  
  
=  
  
v  
  
if  
  
_data  
[  
'user_metadata'  
][  
attr
```

```

][
'datatype'
]

==

'series'
:

result
[
attr
+
'_index'
]

=

_data
[
'user_metadata'
][
attr
][
'#extra#'
]

return

result

# End of field-oriented interface

# Extended interfaces. These permit one to get copies of metadata dictionaries, and to

# get and set custom field metadata

```

[\[docs\]](#)

def

get_standard_metadata

(

self

,

field

,

make_copy

):

'''

return field metadata from the field if it is there. Otherwise return

None. field is the key name, not the label. Return a copy if requested,

just in case the user wants to change values in the dict.

'''

if

field

in

field_metadata

and

field_metadata

[

field


```
[[
'kind'
]

==

'field'
:

if

make_copy
:

return

copy
.
deepcopy
(
field_metadata
[
field
])

return

field_metadata
[
field
]

return

None

\[docs\]
```

```
def
get_all_standard_metadata
(
self
,
make_copy
):

'''
    return a dict containing all the standard field metadata associated with
    the book.
'''

if
not
make_copy
:

return
field_metadata

res
=
{}

for
k
```

in

field_metadata
:

if

field_metadata
[
k
][
'kind'
]

==

'field'
:

res
[
k
]

=

copy

.

deepcopy

(
field_metadata
[
k
])

return

res

[\[docs\]](#)

def

get_all_user_metadata

(
self

,

make_copy

):

'''

*return a dict containing all the custom field metadata associated with
the book.*

'''

_data

=

object

.

__getattr__

(
self

,

'_data'

)

user_metadata

=

_data

[

```
'user_metadata'  
]
```

```
if
```

```
not
```

```
make_copy  
:
```

```
return
```

```
user_metadata
```

```
res
```

```
=
```

```
{}
```

```
for
```

```
k
```

```
in
```

```
user_metadata  
:
```

```
res
```

```
[  
k  
]
```

```
=
```

```
copy
```

```
.
```

```
deepcopy
(
user_metadata
[
k
])
```

```
return
```

```
res
```

```
\[docs\]
```

```
def
```

```
get_user_metadata
```

```
(
self
```

```
,
```

```
field
```

```
,
```

```
make_copy
```

```
):
```

```
'''
```

return field metadata from the object if it is there. Otherwise return

None. field is the key name, not the label. Return a copy if requested,

just in case the user wants to change values in the dict.

```
'''
```

```
_data
```

```
=
```

```
object
.
__getattr__
(
self
,
'_data'
)

_data

=

_data
[
'user_metadata'
]

if

field

in

_data
:

if

make_copy
:

return

copy
.
deepcopy
```

```
(  
_data  
[  
field  
)
```

```
return
```

```
_data  
[  
field  
]
```

```
return
```

```
None
```

```
\[docs\]
```

```
def
```

```
set_all_user_metadata
```

```
(  
self  
,
```

```
metadata  
):
```

```
'''
```

```
    store custom field metadata into the object. Field is the key name
```

```
    not the label
```

```
'''
```

```
if
```


metadata

is

None

:

traceback

.

print_stack

()

return

um

=

{}

for

key

,

meta

in

iteritems

(

metadata

):

m

```
=  
  
meta  
.  
copy  
(  
  
    if  
  
        '#value#'  
  
        not  
  
        in  
  
        m  
        :  
  
        if  
  
        m  
        [  
        'datatype'  
        ]  
  
        ==  
  
        'text'  
  
        and  
  
        m  
        [  
        'is_multiple'  
        ]:  
  
        m  
        [  
        '#value#'  
        ]
```

=

[]

else

:

m

[

'#value#'

]

=

None

um

[

key

]

=

m

_data

=

object

.

__getattr__

(

self

,

'_data'

)

```
_data  
[  
    'user_metadata'  
]
```

```
=
```

```
um
```

```
\[docs\]
```

```
def
```

```
    set_user_metadata
```

```
    (  
        self
```

```
    ,
```

```
    field
```

```
    ,
```

```
    metadata
```

```
    ):
```

```
    """
```

```
        store custom field metadata for one column into the object. Field is
```

```
        the key name not the label
```

```
    """
```

```
if
```

```
    field
```

```
is
```

not

None

:

if

not

field

.

startswith

(

'#'

):

raise

AttributeError

(

'Custom field name

%s

must begin with

\

#

\

,

%

repr

(

field

))

if

metadata

is

None

:

traceback

.

print_stack

()

return

m

=

dict

(

metadata

)

Copying the elements should not be necessary. The objects referenced

in the dict should not change. Of course, they can be replaced.

for k,v in iteritems(metadata):

m[k] = copy.copy(v)

if

'#value#'

not

in

```
m
:

if

m
[
'datatype'
]

==

'text'

and

m
[
'is_multiple'
]:

m
[
'#value#'
]

=

[]

else
:

m
[
'#value#'
]
```

=

None

_data

=

object

.

__getattr__

(

self

,

'_data'

)

_data

[

'user_metadata'

][

field

]

=

m

[\[docs\]](#)

def

remove_stale_user_metadata

(

self

,

other_mi

):


```
"""
    Remove user metadata keys (custom column keys) if they
    don't exist in 'other_mi', which must be a metadata object
    """
```

```
me
```

```
=
```

```
self
```

```
.
```

```
get_all_user_metadata
```

```
(
```

```
make_copy
```

```
=
```

```
False
```

```
)
```

```
other
```

```
=
```

```
set
```

```
(
```

```
other_mi
```

```
.
```

```
custom_field_keys
```

```
())
```

```
new
```

```
=
```

```
{}
```

```
for
```

```
    k
```

```
    ,  
    v
```

```
in
```

```
me
```

```
.  
items  
():
```

```
if
```

```
    k
```

```
in
```

```
other  
:
```

```
new  
[  
    k  
]
```

```
=
```

```
v
```

```
self
```

```
.  
set_all_user_metadata  
(  
    new  
)
```

[\[docs\]](#)

def

template_to_attribute

(
self

,

other

,

ops

):

"""

*Takes a list [(src,dest), (src,dest)], evaluates the template in the
context of other, then copies the result to self[dest]. This is on a
best-efforts basis. Some assignments can make no sense.*

"""

if

not

ops

:

return

from

calibre.ebooks.metadata.book.formatter

import

SafeFormat

formatter

=

SafeFormat
(

for

op

in

ops
:

try
:

src

=

op
[
0
]

dest

=

op
[
1

```
]

val
=

formatter
.
safe_format
(
src
,
other
,
'PLUGBOARD TEMPLATE ERROR'
,
other
)

if

dest

==

'tags'
:

self
.
set
(
dest
,
[
f
```

```
.strip  
(  
  
for  
  
f  
  
in  
  
val  
.split  
(  
' '  
,  
)  
  
if  
  
f  
.strip  
(  
)  
  
elif  
  
dest  
  
==  
  
'authors'  
:  
  
self  
.set  
(  
dest  
,  
  
[
```

```
f
.
strip
()

for

f

in

val
.
split
(
'&'
)

if

f
.
strip
())

else
:

self

.
set
(
dest
,
val
)

except
:
```

```
if
```

```
DEBUG
```

```
:
```

```
traceback
```

```
.
```

```
print_exc
```

```
()
```

```
# Old Metadata API {{{
```

```
def
```

```
print_all_attributes
```

```
(
```

```
self
```

```
):
```

```
for
```

```
x
```

```
in
```

```
STANDARD_METADATA_FIELDS
```

```
:
```

```
prints
```

```
(
```

```
'
```

```
%s
```

```
:'
```

```
%
```

```
x
```

```
,
```



```
getattr
(
self
,
x
,
'None'
))

for
x
in
self
.
custom_field_keys
():

meta

=

self
.
get_user_metadata
(
x
,
make_copy
=
False
)

if
```

```
meta
```

```
is
```

```
not
```

```
None
```

```
:
```

```
prints
```

```
(
```

```
x
```

```
,
```

```
meta
```

```
)
```

```
prints
```

```
(
```

```
'-----'
```

```
)
```

```
\[docs\]
```

```
def
```

```
smart_update
```

```
(
```

```
self
```

```
,
```

```
other
```

```
,
```

```
replace_metadata
```

```
=
```

```
False
```

```
):  
  
"""  
    Merge the information in `other` into self. In case of conflicts, the information  
    in `other` takes precedence, unless the information in `other` is NULL.  
    """  
  
def  
copy_not_none  
(  
    dest  
,  
    src  
,  
    attr  
):  
    v  
    =  
    getattr  
(  
        src  
,  
        attr  
,  
        None  
    )  
  
    if
```

```
v
not
in
(
None
,
NULL_VALUES
.
get
(
attr
,
None
)):

setattr
(
dest
,
attr
,
copy
.
deepcopy
(
v
))

unknown

=
```

```
-  
(  
  'Unknown'  
)
```

```
if
```

```
  other
```

```
  .  
  title
```

```
and
```

```
  other
```

```
  .  
  title
```

```
!=
```

```
  unknown
```

```
  :
```

```
  self
```

```
  .  
  title
```

```
=
```

```
  other
```

```
  .  
  title
```

```
if
```

```
  hasattr
```

```
  (  
    other
```

```
  ,
```

```
  'title_sort'
```

```
):
```

```
self
```

```
.  
title_sort
```

```
=
```

```
other
```

```
.  
title_sort
```

```
if
```

```
other
```

```
.  
authors
```

```
and
```

```
(
```

```
other
```

```
.  
authors
```

```
[  
0  
]
```

```
!=
```

```
unknown
```

```
or
```

```
(
```

```
not
```

```
self
.
authors

or

(

len
(
self
.
authors
)
==

1

and

self
.
authors
[
0
]

==

unknown

and

getattr
(
self
,
'author_sort'
```

```
,  
  
None  
)  
  
==  
  
unknown  
  
)  
  
)  
  
):  
  
self  
.  
authors  
  
=  
  
list  
(  
other  
.  
authors  
)  
  
if  
  
hasattr  
(  
other  
,  
  
'author_sort_map'  
):
```



```
self
    .
    author_sort_map
    =
    dict
    (
    other
    .
    author_sort_map
    )
```

```
if
    hasattr
    (
    other
    ,
    'author_sort'
    ):
    self
    .
    author_sort
    =
    other
    .
    author_sort
```

```
if
    replace_metadata
    :
```

```
# SPECIAL_FIELDS = frozenset(['lpath', 'size', 'comments', 'thumbnail'])
```

```
for
```

```
attr
```

```
in
```

```
SC_COPYABLE_FIELDS
```

```
:
```

```
setattr
```

```
(
```

```
self
```

```
,
```

```
attr
```

```
,
```

```
getattr
```

```
(
```

```
other
```

```
,
```

```
attr
```

```
,
```

```
1.0
```

```
if
```

```
attr
```

```
==
```

```
'series_index'
```

```
else
```

None

))

self

.

tags

=

other

.

tags

self

.

cover_data

=

getattr

(

other

,

'cover_data'

,

NULL_VALUES

[

'cover_data'

])

self

.

set_all_user_metadata

(

other

.

get_all_user_metadata

```
(
    make_copy
    =
    True
))

for
x

in
SC_FIELDS_COPY_NOT_NULL
:

    copy_not_none
    (
        self
        ,
        other
        ,
        x
        )

    if
        callable
        (
            getattr
            (
                other
                ,
                'get_identifiers'
                ,
                None
            )
        ):
```

```
self
.  
set_identifiers  
(  
other  
.  
get_identifiers  
())  
  
# language is handled below  
  
else  
:  
  
for  
attr  
in  
SC_COPYABLE_FIELDS  
:  
  
copy_not_none  
(  
self  
,  
other  
,  
attr  
)  
  
for
```

```
x
```

```
in
```

```
SC_FIELDS_COPY_NOT_NULL
```

```
:
```

```
copy_not_none
```

```
(
```

```
self
```

```
,
```

```
other
```

```
,
```

```
x
```

```
)
```

```
if
```

```
other
```

```
.
```

```
tags
```

```
:
```

```
# Case-insensitive but case preserving merging
```

```
lotags
```

```
=
```

```
[
```

```
t
```

```
.
```

```
lower
```

```
()
```

```
for
```

```
t
```

```
in
```

```
other
```

```
.
```

```
tags
```

```
]
```

```
lstags
```

```
=
```

```
[
```

```
t
```

```
.
```

```
lower
```

```
()
```

```
for
```

```
t
```

```
in
```

```
self
```

```
.
```

```
tags
```

```
]
```

```
ot
```

```
,
```

```
st
```

```
=
```

```
map
```

```
(
```

```
frozenset
```

```
,  
  
(  
lotags  
,  
  
lstags  
)  
  
for  
  
t  
  
in  
  
st  
.  
intersection  
(  
ot  
):  
  
sidx  
  
=  
  
lstags  
.  
index  
(  
t  
)  
  
oidx  
  
=  
  
lotags  
.  
index
```



```
(  
t  
)
```

```
self
```

```
.  
tags  
[  
sidx  
]
```

```
=
```

```
other
```

```
.  
tags  
[  
oidx  
]
```

```
self
```

```
.  
tags
```

```
+=
```

```
[  
t
```

```
for
```

```
t
```

```
in
```

```
other
```

```
.  
tags
```

```
if
```

```
t
.
lower
()

in

ot
-
st
]

if

getattr
(
other
,
'cover_data'
,
False
):

other_cover

=

other
.
cover_data
[
-
1
]

self_cover
```

```
=  
  
self  
.cover_data  
[  
-  
1  
]  
  
if  
  
self  
.cover_data  
  
else  
  
b  
"  
  
if  
  
not  
  
self_cover  
:  
  
self_cover  
  
=  
  
b  
"  
  
if  
  
not  
  
other_cover
```

```
:
```



```
other_cover
```

```
=
```

```
b  
"
```

```
if
```

```
len  
(  
other_cover  
)
```

```
>
```

```
len  
(  
self_cover  
):
```

```
self  
.  
cover_data
```

```
=
```

```
other  
.  
cover_data
```

```
if
```

```
callable  
(  
getattr
```

```
(
other
,

'custom_field_keys'
,

None
)):
```

for

x

in

```
other
.
custom_field_keys
():
```

meta

=

```
other
.
get_user_metadata
(
x
,

make_copy
=
True
)
```

if

meta

is

not

None

:

self_tags

=

self

.

get

(

x

,

[])

if

isinstance

(

self_tags

,

string_or_bytes

):

self_tags

=

[]

self

.

```
set_user_metadata  
(  
x  
,  
  
meta  
)  
  
# get... did the deepcopy
```

```
other_tags
```

```
=
```

```
other
```

```
.
```

```
get
```

```
(
```

```
x
```

```
,
```

```
[])
```

```
if
```

```
meta
```

```
[
```

```
'datatype'
```

```
]
```

```
==
```

```
'text'
```

```
and
```

```
meta
```

```
[
```

```
'is_multiple'
```

```
]:
```

Case-insensitive but case preserving merging

lotags

=

[
t

.

lower
()

for

t

in

other_tags
]

try
:

lstags

=

[
t

.

lower
()

for

t

in


```
self_tags  
]
```

```
except
```

```
TypeError  
:
```

```
# Happens if x is not a text, is_multiple field
```

```
# on self
```

```
lstags
```

```
=
```

```
[]
```

```
self_tags
```

```
=
```

```
[]
```

```
ot
```

```
,
```

```
st
```

```
=
```

```
map
```

```
(
```

```
frozenset
```

```
,
```

```
(
lotags
,
lstags
))

for

t

in

st
.
intersection
(
ot
):

sidx

=

lstags
.
index
(
t
)

oidx

=

lotags
.
index
(
t
```

```
)

self_tags
[
    idx
]

=

other_tags
[
    oidx
]

self_tags

+=

[
    t

for

t

in

other_tags

if

t

.
lower
()

in

ot
-
st
```

```
]

setattr
(
self
,

x
,

self_tags
)


my_comments

=

getattr
(
self
,

'comments'
,

''
)


other_comments

=

getattr
(
other
,

'comments'
,
```

```
"
)

if
not
my_comments
:

my_comments

=

"

if
not
other_comments
:

other_comments

=

"

if
len
(
other_comments
.
strip
())
```

```
>

len
(
my_comments
.
strip
()):

self
.
comments

=

other_comments

# Copy all the non-none identifiers

if
callable
(
getattr
(
other
,
'get_identifiers'
,
None
)):

d

=
```

```
self
.  
get_identifiers  
(  

```

```
s
```

```
=
```

```
other  
.  
get_identifiers  
(  

```

```
d  
.  
update  
([  
v
```

```
for
```

```
v
```

```
in
```

```
iteritems  
(  
s  
)
```

```
if
```

```
v  
[  
1  
]
```

```
is
```

not

None

])

self

.

set_identifiers

(

d

)

else

:

other structure not Metadata. Copy the top-level identifiers

for

attr

in

TOP_LEVEL_IDENTIFIERS

:

copy_not_none

(

self

,

other

,

attr

)


```
other_lang
```

```
=
```

```
getattr
```

```
(
```

```
other
```

```
,
```

```
'languages'
```

```
,
```

```
[])
```

```
if
```

```
other_lang
```

```
and
```

```
other_lang
```

```
!=
```

```
[
```

```
'und'
```

```
]:
```

```
self
```

```
.
```

```
languages
```

```
=
```

```
list
```

```
(
```

```
other_lang
```

```
)
```

```
if
    not
        getattr
        (
            self
            ,
            'series'
            ,
            None
        ):

        self
        .
        series_index
        =
        None

    def
        format_series_index
        (
            self
            ,
            val
            =
            None
        ):

        from
            calibre.ebooks.metadata

        import
```

```
fmt_sidx
```

```
v
```

```
=
```

```
self
```

```
.
```

```
series_index
```

```
if
```

```
val
```

```
is
```

```
None
```

```
else
```

```
val
```

```
try
```

```
:
```

```
x
```

```
=
```

```
float
```

```
(
```

```
v
```

```
)
```

```
except
```

```
Exception
```

```
:
```

```
x
```

```
=
```

```
1
```

```
return
```

```
fmt_sidx
```

```
(
```

```
x
```

```
)
```

```
def
```

```
authors_from_string
```

```
(
```

```
self
```

```
,
```

```
raw
```

```
):
```

```
from
```

```
calibre.ebooks.metadata
```

```
import
```

```
string_to_authors
```

```
self
```

```
.
```

```
authors
```

```
=
```

```
string_to_authors  
(  
raw  
)
```

```
def
```

```
format_authors  
(  
self  
):
```

```
from
```

```
calibre.ebooks.metadata
```

```
import
```

```
authors_to_string
```

```
return
```

```
authors_to_string  
(  
self  
.  
authors  
)
```

```
def
```

```
format_tags  
(  
self  
):
```

return

','

.

join

([

str

(

t

)

for

t

in

sorted

(

self

.

tags

,

key

=

sort_key

))

def

format_rating

(

self

,

v

=

None

,

```
divide_by
```

```
=
```

```
1
```

```
):
```

```
if
```

```
v
```

```
is
```

```
None
```

```
:
```

```
if
```

```
self
```

```
.
```

```
rating
```

```
is
```

```
not
```

```
None
```

```
:
```

```
return
```

```
str
```

```
(
```

```
self
```

```
.
```

```
rating
```

```
/
```

```
divide_by
```

```
)
```

```
return
```

```
'None'
```

```
return
```

```
str
```

```
(
```

```
v
```

```
/
```

```
divide_by
```

```
)
```

```
\[docs\]
```

```
def
```

```
format_field
```

```
(
```

```
self
```

```
,
```

```
key
```

```
,
```

```
series_with_index
```

```
=
```

```
True
```

```
):
```

```
'''
```

```
    Returns the tuple (display_name, formatted_value)
```

```
'''
```

```
name
```



```
,  
  
val  
,  
  
ign  
,  
  
ign  
  
=  
  
self  
.  
format_field_extended  
(  
key  
,  
  
series_with_index  
)  
  
return  
  
(  
name  
,  
  
val  
)  
  
def  
format_field_extended  
(  
self  
,  
  
key  
,
```

```
series_with_index
=
True
):

from
calibre.ebooks.metadata

import

authors_to_string

'''
    returns the tuple (display_name, formatted_value, original_value,
    field_metadata)
'''

from
calibre.utils.date

import

format_date

# Handle custom series index

if

key
.
startswith
(
```

```
'#'  
)  
  
and  
  
key  
.  
endswith  
(  
    '_index'  
)  
:  
  
tkey  
  
=  
  
key  
[:  
-  
6  
]  
  
# strip the _index  
  
cmeta  
  
=  
  
self  
.  
get_user_metadata  
(  
    tkey  
,  
  
    make_copy  
    =  
    False  
)
```

```
if
    cmeta
    and
    cmeta
    [
        'datatype'
    ]
    ==
    'series'
    :

    if
        self
        .
        get
        (
            tkey
        ):

        res
        =
        self
        .
        get_extra
        (
            tkey
        )

    return
    (
        str
```

```
(
cmeta
[
'name'
]
+
'_index'
),

self
.
format_series_index
(
res
),

res
,

cmeta
)

else
:

return

(
str
(
cmeta
[
'name'
]
+
'_index'
),
"
,
```

```
"
,
cmeta
)

if
key
in
self
.
custom_field_keys
():

res

=

self
.
get
(
key
,
None
)

# get evaluates all necessary composites

cmeta

=

self
```

```
.
get_user_metadata
(
key
,

make_copy
=
False
)

name

=

str
(
cmeta
[
'name'
])

if

res

is

None

or

res

==

"
:

# can't check "not res" because of numeric fields
```

```
return
```

```
(  
name
```

```
,
```

```
res
```

```
,
```

```
None
```

```
,
```

```
None
```

```
)
```

```
orig_res
```

```
=
```

```
res
```

```
datatype
```

```
=
```

```
cmeta
```

```
[
```

```
'datatype'
```

```
]
```

```
if
```

```
datatype
```

```
==
```

```
'text'
```

```
and
```



```
cmeta
[
    'is_multiple'
]:
```

```
res
```

```
=
```

```
cmeta
[
    'is_multiple'
][
    'list_to_ui'
]
.
join
(
    res
)
```

```
elif
```

```
datatype
```

```
==
```

```
'series'
```

```
and
```

```
series_with_index
:
```

```
if
```

```
self
```

```
.
```

```
get_extra
```

```
(
key
)

is

not

None
:

res

=

res

+
\

' [
%s
]'
%
self
.
format_series_index
(
val
=
self
.
get_extra
(
key
))

elif

datatype
```

```
==

'datetime'
:

res

=

format_date
(
res
,

cmeta
[
'display'
]
.
get
(
'date_format'
,
'dd MMM yyyy'
))
```

elif

datatype

```
==
```

```
'bool'
:
```

res

```
=
```

```
-
```

```
(  
'Yes'  
)  
  
if  
  
res  
  
else  
  
_  
(  
'No'  
)  
  
elif  
  
datatype  
  
==  
  
'rating'  
:  
  
res  
  
=  
  
,  
%.2g  
,  
%  
(  
res  
/  
2  
)  
  
elif
```

```
datatype

in

[
'int'
,

'float'
]:

try
:

fmt

=

cmeta
[
'display'
]
.
get
(
'number_format'
,

None
)

res

=

fmt
.
format
(
res
```

```
)

except
:

pass

return

(
name
,
str
(
res
),
orig_res
,
cmeta
)

# convert top-level ids into their value

if

key

in

TOP_LEVEL_IDENTIFIERS
:

fmeta
```

```
=
```

```
field_metadata
```

```
[
```

```
'identifiers'
```

```
]
```

```
name
```

```
=
```

```
key
```

```
res
```

```
=
```

```
self
```

```
.
```

```
get
```

```
(
```

```
key
```

```
,
```

```
None
```

```
)
```

```
return
```

```
(
```

```
name
```

```
,
```

```
res
```

```
,
```

```
res
```

```
,
```

```
fmeta
)
```

```
# Translate aliases into the standard field name
```

```
fmkey
```

```
=
```

```
field_metadata
```

```
.
```

```
search_term_to_field_key
```

```
(
```

```
key
```

```
)
```

```
if
```

```
fmkey
```

```
in
```

```
field_metadata
```

```
and
```

```
field_metadata
```

```
[
```

```
fmkey
```

```
][
```

```
'kind'
```

```
]
```

```
==
```

```
'field'
```

```
:
```



```
res
```

```
=
```

```
self
```

```
.
```

```
get
```

```
(
```

```
key
```

```
,
```

```
None
```

```
)
```

```
fmeta
```

```
=
```

```
field_metadata
```

```
[
```

```
fmkey
```

```
]
```

```
name
```

```
=
```

```
str
```

```
(
```

```
fmeta
```

```
[
```

```
'name'
```

```
])
```

```
if
```

```
res
```

```
is
```

None

or

res

==

"

:

return

(
name

,

res

,

None

,

None

)

orig_res

=

res

name

=

str

(

fmeta

[

```
'name'
])

datatype

=

fmeta
[
'datatype'
]

if

key

==

'authors'
:

res

=

authors_to_string
(
res
)

elif

key

==

'series_index'
:
```

```
res

=

self
.
format_series_index
(
res
)

elif

datatype

==

'text'

and

fmeta
[
'is_multiple'
]:

if

isinstance
(
res
,
dict
):

res

=
```

```
[
k

+

','.

+

v

for

k
,
v

in

res
.
items
()]

res

=

fmeta
[
'is_multiple'
][
'list_to_ui'
]
.
join
(
sorted
(
filter
(
```

None

,

res

),

key

=

sort_key

))

elif

datatype

==

'series'

and

series_with_index

:

res

=

res

+

' ['

%s

]'

%

self

.

format_series_index

()

elif

datatype

==

'datetime'

:

res

=

format_date

(

res

,

fmeta

[

'display'

]

.

get

(

'date_format'

,

'dd MMM yyyy'

))

elif

datatype

==

'rating'

:

```
res

=

',
%.2g
',
%
(
res
/
2
)

elif

key

==

'size'
:

res

=

human_readable
(
res
)

return

(
name
,
str
(
```



```
res
),

orig_res
,

fmeta
)


return

(
None
,

None
,

None
,

None
,

None
)


def

__unicode__representation__
(
self
):

'''

    A string representation of this object, suitable for printing to

    console

'''
```

```
from
```

```
calibre.utils.date
```

```
import
```

```
isoformat
```

```
from
```

```
calibre.ebooks.metadata
```

```
import
```

```
authors_to_string
```

```
ans
```

```
=
```

```
[]
```

```
def
```

```
fmt
```

```
(
```

```
x
```

```
,
```

```
y
```

```
):
```

```
ans
```

```
.
```

```
append
```

```
(
```

```
'
%-20s
:
%s
'

%
(
str
(
x
),

str
(
y
)))

fmt
(
'Title'
,

self
.
title
)

if

self
.
title_sort
:

fmt
(
'Title sort'
,
```

```
self
.
title_sort
)

if

self
.
authors
:

fmt
(
'Author(s)'
,

authors_to_string
(
self
.
authors
)

+

((
' ['

+

self
.
author_sort

+

']'
)
```

```
if
    self
    .
    author_sort
```

```
and
```

```
self
    .
    author_sort
```

```
!=
```

```
—
(
    'Unknown'
)
```

```
else
```

```
"
))
```

```
if
```

```
self
    .
    publisher
    :
```

```
fmt
(
    'Publisher'
    ,
```

```
self
    .
    publisher
)
```

```
if
getattr
(
self
,
'book_producer'
,
False
):
```

```
fmt
(
'Book Producer'
,
self
.
book_producer
)
```

```
if
self
.
tags
:
```

```
fmt
(
'Tags'
,
' '
.
join
```

```
([
str
(
t
)

for

t

in

self
.
tags
]))

if

self
.
series
:

fmt
(
'Series'
,

self
.
series

+

' #'
's
'
%
self
.

```

```
format_series_index  
())
```

```
if
```

```
not
```

```
self
```

```
.
```

```
is_null
```

```
(
```

```
'languages'
```

```
):
```

```
fmt
```

```
(
```

```
'Languages'
```

```
,
```

```
','
```

```
.
```

```
join
```

```
(
```

```
self
```

```
.
```

```
languages
```

```
))
```

```
if
```

```
self
```

```
.
```

```
rating
```

```
is
```

```
not
```

```
None
```

```
:
```



```
fmt
(
  'Rating'
,
  (
    '
    %.2g
    '
    %
    (
      float
      (
        self
        .
        rating
      )
      /
      2
    ))

    if

    self
    .
    rating

    else

    ''
  )

    if

    self
    .
    timestamp

    is
```

not

None

:

fmt

(

'Timestamp'

,

isoformat

(

self

.

timestamp

))

if

self

.

pubdate

is

not

None

:

fmt

(

'Published'

,

isoformat

(

self

.

```
pubdate
))

if

self
.
rights

is

not

None
:

fmt
(
'Rights'
,

str
(
self
.
rights
))

if

self
.
identifiers
:

fmt
(
'Identifiers'
,
```

```
' '
.
join
([
'
%S
:
%S
'
%
(
k
,
v
)
for
k
,
v
in

iteritems
(
self
.
identifiers
)))

if

self
.
comments
:
```

```
fmt
(  
  'Comments'  
,  
  self  
  .  
  comments  
)
```

```
for  
key  
in  
self  
  .  
  custom_field_keys  
  ():
```

```
val  
  
=  
  
self  
  .  
  get  
  (  
    key  
  ,  
  None  
)
```

```
if  
  
val
```

```
:  
  
(  
name  
,  
val  
)  
  
=  
  
self  
.  
format_field  
(  
key  
)  
  
fmt  
(  
name  
,  
str  
(  
val  
))  
  
return  
  
,  
\\n  
,  
.  
join  
(  
ans  
)
```

[\[docs\]](#)

def

to_html

(

self

):

'''

A HTML representation of this object.

'''

from

calibre.ebooks.metadata

import

authors_to_string

from

calibre.utils.date

import

isoformat

ans

=

[(

```
-
(
'Title'
),

str
(
self
.
title
))]

ans

+=

[(
-
(
'Author(s)'
),

(
authors_to_string
(
self
.
authors
)

if

self
.
authors

else

-
(
'Unknown'
)))]
```


ans

+=

```
[(  
    -  
    (  
        'Publisher'  
    ),
```

```
    str  
    (  
        self  
        .  
        publisher  
    )  
    )  
    ]
```

ans

+=

```
[(  
    -  
    (  
        'Producer'  
    ),
```

```
    str  
    (  
        self  
        .  
        book_producer  
    )  
    )  
    ]
```

ans

+=

```
[(
```

```
-  
(  
  'Comments'  
)
```

```
str  
(  
  self  
  .  
  comments  
)])
```

```
ans
```

```
+=
```

```
[(  
  'ISBN'  
,
```

```
str  
(  
  self  
  .  
  isbn  
)])
```

```
ans
```

```
+=
```

```
[(  
  -  
(  
  'Tags'  
)
```

```
','
```

```
  .  
  join  
  ([
```

```
str
(
t
)

for

t

in

self
.
tags
]]])

if

self
.
series
:

ans

+=

[(
ngettext
(
'Series'
,
'Series'
,
1
),
str
(
```

```

self
.
series
)

+

' #
%s
'

%
self
.
format_series_index
())

ans

+=

[(
-
(
'Languages'
),

','
.
join
(
self
.
languages
))]

if

self
.
timestamp

```

```
is
not
None
:

ans

+=

[(
-
(
'Timestamp'
),

str
(
isoformat
(
self
.
timestamp
,

as_utc
=
False
,

sep
=
', '
)))]

if
self
.
pubdate
```

is

not

None

:

ans

+=

[(

—
(

'Published'

),

str

(

isoformat

(

self

.

pubdate

,

as_utc

=

False

,

sep

=

' '

]]))]

if

self

.

rights

is

not

None

:

ans

+=

[(

-

(

'Rights'

),

str

(

self

.

rights

))]

for

key

in

self

.

custom_field_keys

():

val

=

```
self
.
get
(
key
,

None
)

if

val
:

(
name
,

val
)

=

self
.
format_field
(
key
)

ans

+=

[(
name
,
```



```
val
)]

for

i
,

x

in

enumerate
(
ans
):

ans
[
i
]

=

'<tr><td><b>
%s
</b></td><td>
%s
</td></tr>'
%
x

return

'<table>
%s
</table>'
%
'
\n
```

```
'  
.  
join  
(  
ans  
)  
  
__str__  
  
=  
  
__unicode__representation__  
  
  
def  
  
__nonzero__  
(  
self  
):  
  
return  
  
bool  
(  
self  
.  
title  
  
or  
  
self  
.  
author  
  
or  
  
self  
.  
comments
```

```
or
```

```
self
```

```
.
```

```
tags
```

```
)
```

```
__bool__
```

```
=
```

```
__nonzero__
```

```
# }}}
```

```
def
```

```
field_from_string
```

```
(
```

```
field
```

```
,
```

```
raw
```

```
,
```

```
field_metadata
```

```
):
```

```
""" Parse the string raw to return an object that is suitable for calling
```

```
set() on a Metadata object. """
```

```
dt
```

```
=
```

```
field_metadata
```

```
[
```

```
'datatype'
```

```
]
```

```
val
```

```
=
```

```
object
```

```
if
```

```
dt
```

```
in
```

```
{
```

```
'int'
```

```
,
```

```
'float'
```

```
};
```

```
val
```

```
=
```

```
int
```

```
(
```

```
raw
```

```
)
```

```
if
```

```
dt
```

```
==
```

```
'int'
```

```
else
```

```
float
```

```
(  
raw  
)
```

```
elif
```

```
dt
```

```
==
```

```
'rating'  
:
```

```
val
```

```
=
```

```
float
```

```
(  
raw  
)
```

```
*
```

```
2
```

```
elif
```

```
dt
```

```
==
```

```
'datetime'  
:
```

```
from  
  
calibre.utils.iso8601
```

```
import
```

```
parse_iso8601
```

```
try  
:
```

```
val
```

```
=
```

```
parse_iso8601
```

```
(  
raw  
,
```

```
require_aware  
=
```

```
True  
)
```

```
except
```

```
Exception  
:
```

```
from
```

```
calibre.utils.date
```

```
import
```

```
parse_only_date
```

```
val
=
parse_only_date
(
raw
)
```

elif

dt

==

```
'bool'
:
```

if

```
raw
.
lower
()
```

in

```
{
'true'
,
'yes'
,
'y'
}:

```

```
val
```

=

True

elif

raw

.

lower

()

in

{

'false'

,

'no'

,

'n'

}:

val

=

False

else

:

raise

ValueError

(

'Unknown value for

%s

:

%s


```
'  
%  
(  
field  
,  
  
raw  
)  
  
elif  
  
dt  
  
==  
  
'text'  
:  
  
ism  
  
=  
  
field_metadata  
[  
'is_multiple'  
]  
  
if  
  
ism  
:  
  
val  
  
=  
  
[  
x  
.
```

```
strip
()

for

x

in

raw
.
split
(
ism
[
'ui_to_list'
]])

if

field

==

'identifiers'
:

val

=

{
x
.
partition
(
':'
)[
0
]:
x
```

```
.
partition
(
    ':'
)[
    -
    1
]

for
x
in
val
}

elif

field

==

'languages'
:

from
calibre.utils.localization

import

canonicalize_lang

val

=

[
```

```
canonicalize_lang
```

```
(
```

```
x
```

```
)
```

```
for
```

```
x
```

```
in
```

```
val
```

```
]
```

```
val
```

```
=
```

```
[
```

```
x
```

```
for
```

```
x
```

```
in
```

```
val
```

```
if
```

```
x
```

```
]
```

```
if
```

```
val
```

```
is
```

```
object
```

:

val

=

raw

return

val

Source code for calibre.ebooks.metadata.sources.base

```
#!/usr/bin/env python

# vim:fileencoding=UTF-8:ts=4:sw=4:sta:et:sts=4:ai

from
__future__

import
absolute_import
,
division
,
print_function
,
unicode_literals

__license__

=

'GPL v3'

__copyright__

=

'2011, Kovid Goyal <kovid@kovidgoyal.net>'
```

```
__docformat__  
  
=  
  
'restructuredtext en'
```

```
import
```

```
re  
,
```

```
threading
```

```
from
```

```
functools
```

```
import
```

```
total_ordering
```

```
from
```

```
calibre
```

```
import
```

```
browser  
,
```

```
random_user_agent
```

```
from
```

```
calibre.customize
```

```
import
```

```
Plugin
```

```
from
```

```
calibre.ebooks.metadata
```

```
import
```

```
check_isbn
```

```
from
```

```
calibre.ebooks.metadata.author_mapper
```

```
import
```

```
cap_author_token
```

```
from
```

```
calibre.utils.localization
```

```
import
```

```
canonicalize_lang
```

```
,
```

```
get_lang
```

```
from
```

```
polyglot.builtins
```

```
import
```

```
iteritems
```

```
,
```

```
cmp
```

```
def
```

```
create_log
```



```
(
ostream
=
None
):

from

calibre.utils.logging

import

ThreadSafeLog
,

FileStream

log

=

ThreadSafeLog
(
level
=
ThreadSafeLog
.
DEBUG
)

log
.
outputs

=

[
FileStream
(
ostream
```

```

    ])

    return

    log

    # Comparing Metadata objects for relevance {{{

    words

    =

    (
    "the"
    ,
    "a"
    ,
    "an"
    ,
    "of"
    ,
    "and"
    )

    prefix_pat

    =

    re
    .
    compile
    (
    r
    '^('
    %s
    )\s+'

```

```
%  
(  
    "|"  
    .  
    join  
    (  
        words  
    ))  
))
```

```
trailing_paren_pat
```

```
=
```

```
re  
.  
compile  
(  
    r  
    '\(.*\)$'  
)
```

```
whitespace_pat
```

```
=
```

```
re  
.  
compile  
(  
    r  
    '\s+'  
)
```

```
def
```

```
cleanup_title  
(  
    s  
):
```

```
if
not
s
:
s
=
-
(
'Unknown'
)
s
=
s
.
strip
()
.
lower
()
s
=
prefix_pat
.
sub
(
' ',
,
s
```

```
)

s

=

trailing_paren_pat
.
sub
(
"
,

s
)

s

=

whitespace_pat
.
sub
(
" "
,

s
)

return

s
.
strip
()
```

[\[docs\]](#)

@total_ordering

class

InternalMetadataCompareKeyGen

:

"""

Generate a sort key for comparison of the relevance of Metadata objects, given a search query. This is used only to compare results from the same metadata source, not across different sources.

The sort key ensures that an ascending order sort is a sort by order of decreasing relevance.

The algorithm is:

- * Prefer results that have at least one identifier the same as for the query*
- * Prefer results with a cached cover URL*
- * Prefer results with all available fields filled in*
- * Prefer results with the same language as the current user interface language*
- * Prefer results that are an exact title match to the query*
- * Prefer results with longer comments (greater than 10% longer)*
- * Use the relevance of the result as reported by the metadata source's search engine*

```
'''

def
__init__
(
self
,
mi
,
source_plugin
,
title
,
authors
,
identifiers
):

same_identifier

=

2

idents

=

mi
.
get_identifiers
()
```

for

k

,

v

in

iteritems
(
identifiers
):

if

idents

.

get

(

k

)

==

v

:

same_identifier

=

1

break

all_fields


```
=  
  
1  
  
if  
  
source_plugin  
.  
test_fields  
(  
mi  
)  
  
is  
  
None  
  
else  
  
2  
  
  
  
exact_title  
  
=  
  
1  
  
if  
  
title  
  
and  
\  
  
cleanup_title  
(  
title  
)  
  
==
```

```
cleanup_title  
(  
  mi  
  .  
  title  
)
```

```
else
```

```
2
```

```
language
```

```
=
```

```
1
```

```
if
```

```
mi
```

```
.  
language  
:
```

```
mil
```

```
=
```

```
canonicalize_lang  
(  
  mi  
  .  
  language  
)
```

```
if
```

mil

!=

'und'

and

mil

!=

canonicalize_lang
(
get_lang
()):

language

=

2

has_cover

=

2

if

(
not

source_plugin

.

cached_cover_url_is_reliable

or

```
source_plugin
```

```
.
```

```
get_cached_cover_url
```

```
(
```

```
mi
```

```
.
```

```
identifiers
```

```
)
```

```
is
```

```
None
```

```
)
```

```
else
```

```
1
```

```
self
```

```
.
```

```
base
```

```
=
```

```
(
```

```
same_identifier
```

```
,
```

```
has_cover
```

```
,
```

```
all_fields
```

```
,
```

```
language
```

```
,
```

```
exact_title
```

```
)
```

```
self
.
comments_len

=

len
((
mi
.
comments

or

"
)
.
strip
())

self
.
extra

=

getattr
(
mi
,
'source_relevance'
,
0
)

def
```

```
compare_to_other
```

```
(  
    self
```

```
,
```

```
    other  
):
```

```
    a
```

```
    =
```

```
    cmp  
    (  
        self
```

```
        .
```

```
        base
```

```
,
```

```
        other
```

```
        .
```

```
        base
```

```
)
```

```
    if
```

```
        a
```

```
        !=
```

```
        0
```

```
        :
```

```
        return
```

```
        a
```

```
    cx
```

```
,
```

```
cy
```

```
=
```

```
self
```

```
.
```

```
comments_len
```

```
,
```

```
other
```

```
.
```

```
comments_len
```

```
if
```

```
cx
```

```
and
```

```
cy
```

```
:
```

```
t
```

```
=
```

```
(
```

```
cx
```

```
+
```

```
cy
```

```
)
```

```
/
```

```
20
```

```
delta
```

=

cy

-

cx

if

abs
(
delta
)

>

t
:

return

-

1

if

delta

<

0

else

1

return


```
cmp
(
self
.
extra
,

other
.
extra
)


def
__eq__
(
self
,
other
):

return
self
.
compare_to_other
(
other
)

==

0


def
__ne__
```

```
(
self
,
other
):

return

self
.
compare_to_other
(
other
)

!=

0


def
__lt__
(
self
,
other
):

return

self
.
compare_to_other
(
other
)
```

<

0

def

__le__

(

self

,

other

):

return

self

.

compare_to_other

(

other

)

<=

0

def

__gt__

(

self

,

other

):

```
return

self
.
compare_to_other
(
other
)

>

0
```

```
def

__ge__
(
self
,

other
):
```

```
return

self
.
compare_to_other
(
other
)

>=

0

# }}}}
```

```
def
```

```
get_cached_cover_urls
```

```
(  
mi  
):
```

```
from
```

```
calibre.customize.ui
```

```
import
```

```
metadata_plugins
```

```
plugins
```

```
=
```

```
list
```

```
(  
metadata_plugins  
([  
'identify'  
]))
```

```
for
```

```
p
```

```
in
```

```
plugins
```

```
:
```

```
url
```

```
=
```

```
p
.  
get_cached_cover_url  
(  
mi  
.  
identifiers  
)
```

```
if
```

```
url  
:
```

```
yield
```

```
(  
p  
,  
url  
)
```

```
def
```

```
dump_caches  
():
```

```
from
```

```
calibre.customize.ui
```

```
import
```

```
metadata_plugins
```

```
return

{
p
.
name
:
p
.
dump_caches
()

for

p

in

metadata_plugins
([
'identify'
])}

def

load_caches
(
dump
):

from

calibre.customize.ui

import

metadata_plugins

plugins
```

```
=
```

```
list
```

```
(
```

```
metadata_plugins
```

```
([
```

```
'identify'
```

```
]))
```

```
for
```

```
p
```

```
in
```

```
plugins
```

```
:
```

```
cache
```

```
=
```

```
dump
```

```
.
```

```
get
```

```
(
```

```
p
```

```
.
```

```
name
```

```
,
```

```
None
```

```
)
```

```
if
```

```
cache
```

```
:
```



```
p
.  
load_caches  
(  
cache  
)
```

```
def
```

```
fixauthors  
(  
authors  
):
```

```
if
```

```
not
```

```
authors  
:
```

```
return
```

```
authors
```

```
ans
```

```
=
```

```
[]
```

```
for
```

```
x
```

```
in
```

```
authors
:

ans
.
append
(
' '
.
join
(
map
(
cap_author_token
,
x
.
split
()))
)

return

ans

def
fixcase
(
x
):

if
x
:
```

```
from
```

```
calibre.utils.titlecase
```

```
import
```

```
titlecase
```

```
x
```

```
=
```

```
titlecase
```

```
(
```

```
x
```

```
)
```

```
return
```

```
x
```

```
class
```

```
Option
```

```
:
```

```
__slots__
```

```
=
```

```
[
```

```
'type'
```

```
,
```

```
'default'
```

```
,
```

```
'label'
',
'desc'
',
'name'
',
'choices'
]
```

```
def

__init__
(
self
,
name
,
type_
,
default
,
label
,
desc
,
choices
=
None
):

'''
```

:param name: The name of this option. Must be a valid python identifier

:param type_: The type of this option, one of ('number', 'string',

'bool', 'choices')

:param default: The default value for this option

:param label: A short (few words) description of this option

:param desc: A longer description of this option

:param choices: A dict of possible values, used only if type='choices'.

dict is of the form {key:human readable label, ...}

'''

self

.

name

,

self

.

type

,

self

.

default

,

self

.

label

,

self

.

desc

```
=
```

```
(  
name  
,
```

```
type_  
,
```

```
default  
,
```

```
label  
,
```

```
desc  
)
```

```
if
```

```
choices
```

```
and
```

```
not
```

```
isinstance
```

```
(  
choices  
,
```

```
dict  
):
```

```
choices
```

```
=
```

```
dict
```

```
((  
x  
,  
x  
)  
  
for  
x  
  
in  
choices  
)
```

```
self  
.  
choices  
  
=  
  
choices
```

```
\[docs\]  
class
```

Source

```
(  
Plugin  
):
```

```
type
```

```
=
```

```
-  
(  
  'Metadata source'  
)
```

author

=

'Kovid Goyal'

supported_platforms

=

```
[  
  'windows'  
,  
  'osx'  
,  
  'linux'  
]
```

#: Set of capabilities supported by this plugin.

#: Useful capabilities are: 'identify', 'cover'

capabilities

=

```
frozenset  
(
```


#: List of metadata fields that can potentially be download by this plugin

#: during the identify phase

touched_fields

=

frozenset

()

#: Set this to True if your plugin returns HTML formatted comments

has_html_comments

=

False

#: Setting this to True means that the browser object will indicate

#: that it supports gzip transfer encoding. This can speedup downloads

#: but make sure that the source actually supports gzip transfer encoding

#: correctly first

supports_gzip_transfer_encoding

=

False

#: Set this to True to ignore HTTPS certificate errors when connecting

#: to this source.

ignore_ssl_errors

=

False

#: Cached cover URLs can sometimes be unreliable (i.e. the download could

#: fail or the returned image could be bogus). If that is often the case

#: with this source, set to False

cached_cover_url_is_reliable

=

True

#: A list of :class:`Option` objects. They will be used to automatically

#: construct the configuration widget for this plugin

options

=

()

#: A string that is displayed at the top of the config widget for this

#: plugin

config_help_message

=

None

#: If True this source can return multiple covers for a given query

can_get_multiple_covers

=

False

#: If set to True covers downloaded by this plugin are automatically trimmed.

auto_trim_covers

=

False

#: If set to True, and this source returns multiple results for a query,

#: some of which have ISBNs and some of which do not, the results without

#: ISBNs will be ignored

prefer_results_with_isbn

=

True

def

`__init__`

(

self

,

*

args

,

**

kwargs

):

Plugin

.

`__init__`

(

self

,

*

args

,

```

**
kwargs
)

self
.
running_a_test

=

False

# Set to True when using identify_test()

self
.
_isbn_to_identifier_cache

=

{}

self
.
_identifier_to_cover_url_cache

=

{}

self
.
cache_lock

=

threading
.
```

```
RLock
()

self
.
_config_obj

=

None

self
.
_browser

=

None

self
.
prefs
.
defaults
[
'ignore_fields'
]

=

[]

for

opt

in

self
```

```
.
options
:

self

.
prefs

.
defaults
[
opt
.
name
]

=

opt
.
default


# Configuration {{{


\[docs\]


def

is_configured
(
self
):

'''

Return False if your plugin needs to be configured before it can be
```

used. For example, it might need a username/password/API key.

'''

return

True

def

is_customizable

(
self
):

return

True

[\[docs\]](#)

def

customization_help

(
self
):

return

'This plugin can only be customized using the GUI'

[\[docs\]](#)

def


```
config_widget
(
    self
):

from

calibre.gui2.metadata.config

import

ConfigWidget

return

ConfigWidget
(
    self
)

\[docs\]

def

save_settings
(
    self
,

config_widget
):

config_widget
.
commit
()
```

```
@property
```

```
def
```

```
prefs
```

```
(
```

```
self
```

```
):
```

```
if
```

```
self
```

```
.
```

```
_config_obj
```

```
is
```

```
None
```

```
:
```

```
from
```

```
calibre.utils.config
```

```
import
```

```
JSONConfig
```

```
self
```

```
.
```

```
_config_obj
```

```
=
```

```
JSONConfig
```

```
(
```

```
'metadata_sources/'
```

```
%s
```

```
.json'
```

```
%
self
.
name
)

return

self
.
_config_obj

# }}}

# Browser {{{

@property

def

user_agent
(
self
):

# Pass in an index to random_user_agent() to test with a particular

# user agent

return

random_user_agent
()
```

```
@property
```

```
def
```

```
browser
```

```
(  
    self  
):
```

```
if
```

```
self
```

```
.  
_browser
```

```
is
```

```
None
```

```
:
```

```
self
```

```
.  
_browser
```

```
=
```

```
browser
```

```
(  
    user_agent
```

```
=
```

```
self
```

```
.  
user_agent
```

```
,
```

```
verify_ssl_certificates
```

```
=
```

not

self

```
.  
ignore_ssl_errors  
)
```

if

self

```
.  
supports_gzip_transfer_encoding  
:
```

self

```
.  
_browser  
.  
set_handle_gzip  
(  
True  
)
```

return

self

```
.  
_browser  
.  
clone_browser  
(
```

}}}}

Caching {{{

```
def
get_related_isbns
(
self
,
id_
):

with
self
.
cache_lock
:

for
isbn
,
q
in
iteritems
(
self
.
_isbn_to_identifier_cache
):

if
q
==
```

```
id_  
:
```

```
yield
```

```
isbn
```

```
def
```

```
cache_isbn_to_identifier
```

```
(  
self  
,
```

```
isbn  
,
```

```
identifier  
):
```

```
with
```

```
self
```

```
.  
cache_lock  
:
```

```
self
```

```
.  
_isbn_to_identifier_cache  
[  
isbn  
]
```

```
=
```

```
identifier
```

def

cached_isbn_to_identifier

(
self

,

isbn

):

with

self

.

cache_lock

:

return

self

.

_isbn_to_identifier_cache

.

get

(
isbn

,

None

)

def

cache_identifier_to_cover_url

(
self


```
,  
  
id_  
,  
  
url  
):  
  
with  
  
self  
.  
cache_lock  
:  
  
self  
.  
_identifier_to_cover_url_cache  
[  
id_  
]  
  
=  
  
url  
  
def  
  
cached_identifier_to_cover_url  
(  
self  
,  
  
id_  
):  
  
with
```

```
self
    .
cache_lock
:

return

self
    .
    _identifier_to_cover_url_cache
    .
get
(
id_
,
None
)

def

dump_caches
(
self
):

with

self
    .
cache_lock
:

return

{
'isbn_to_identifier'
:

```

```
self
    .
    _isbn_to_identifier_cache
    .
    copy
    (),

    'identifier_to_cover'
    :
    self
    .
    _identifier_to_cover_url_cache
    .
    copy
    ()}
```

def

```
load_caches
(
    self
    ,

    dump
    ):


```

with

```
self
    .
    cache_lock
    :

    self
    .
    _isbn_to_identifier_cache
    .
    update
```

```
(
dump
[
'isbn_to_identifier'
])

self
.
_identifier_to_cover_url_cache
.
update
(
dump
[
'identifier_to_cover'
])
```

```
# }}
```

```
# Utility functions {{{
```

```
\[docs\]
```

```
def
```

```
get_author_tokens
```

```
(
self
```

```
,
```

```
authors
```

```
,
```

```
only_first_author
```

```
=
```

True

):

'''

*Take a list of authors and return a list of tokens useful for an
AND search query. This function tries to return tokens in
first name middle names last name order, by assuming that if a comma is
in the author name, the name is in lastname, other names form.*

'''

if

authors

:

Leave ' in there for Irish names

remove_pat

=

re

.

compile

(

r

'[!@#\$\$%^&*() () 「」

{}

`~"\'s\[\\/]'

)

replace_pat

```
=  
  
re  
.  
compile  
(  
r  
['-+.:;,, 〇 ; :']  
)
```

if

```
only_first_author  
:
```

```
authors
```

```
=
```

```
authors  
[:  
1  
]
```

for

```
au
```

in

```
authors  
:
```

```
has_comma
```

```
=
```

```
''  
,
```

in

au

au

=

replace_pat

.

sub

(

' '

,

au

)

parts

=

au

.

split

()

if

has_comma

:

au probably in ln, fn form

parts

=

```
parts  
[  
1  
:]
```

```
+
```

```
parts  
[:  
1  
]
```

```
for
```

```
tok
```

```
in
```

```
parts  
:
```

```
tok
```

```
=
```

```
remove_pat
```

```
.
```

```
sub
```

```
(
```

```
"
```

```
,
```

```
tok
```

```
)
```

```
.
```

```
strip
```

```
()
```

```
if
```



```
len
(
tok
)

>

2

and

tok
.
lower
()

not

in

(
'von'
,
'van'
,
_
(
'Unknown'
)
.
lower
()):

yield

tok
```

[\[docs\]](#)

def

get_title_tokens

(
self

,

title

,

strip_joiners

=

True

,

strip_subtitle

=

False

):

'''

Take a title and return a list of tokens useful for an AND search query.

Excludes connectives(optionally) and punctuation.

'''

if

title

:

strip sub-titles

if

```
strip_subtitle
```

```
:
```

```
subtitle
```

```
=
```

```
re
```

```
.
```

```
compile
```

```
(
```

```
r
```

```
'([\[\]\{\}.*?[\]\}\[\]/:
```

```
\\
```

```
].*$)'
```

```
)
```

```
if
```

```
len
```

```
(
```

```
subtitle
```

```
.
```

```
sub
```

```
(
```

```
"
```

```
,
```

```
title
```

```
))
```

```
>
```

```
1
```

```
:
```

```
title
```

```
=
```

subtitle

```
.  
sub  
(  
"  
,  
title  
)
```

title_patterns

```
=  
  
[(  
re  
.  
compile  
(  
pat  
,  
  
re  
.  
IGNORECASE  
),  
  
repl  
)
```

for

```
pat  
,
```

```
repl
```

in

```
[
```

Remove things like: (2010) (Omnibus) etc.

```
(  
r  
'(?i)[(]{\[\](\d  
{4}  
|omnibus|anthology|hardcover|audiobook|audio\scd|paperback|turtleback|mass\s*market|  
edition|ed\.)(\)]}'  
,  
"  
)
```

Remove any strings that contain the substring edition inside

parentheses

```
(  
r  
'(?i)[(]{\[\].*?(edition|ed\.).*?[\)]}'  
,  
"  
)
```

Remove commas used as separators in numbers

```
(  
r  
'(\d+),(\d+)'  
,  
  
r  
'\1\2'  
)
```

```
# Remove hyphens only if they have whitespace before them
```

```
(  
r  
'(\\s-'  
,  
''  
)
```

```
# Replace other special chars with a space
```

```
(  
r  
"'[:;,!@$%^&*()  
{  
.'~"\\s\\[\\]/ <> 「」 ""'''  
,  
''  
)  
  
]]
```

```
for
```

```
pat  
,
```

```
repl
```

```
in
```

```
title_patterns  
:
```

title

=

pat

.

sub

(

repl

,

title

)

tokens

=

title

.

split

()

for

token

in

tokens

:

token

=

token

```
.  
strip  
(
```

```
.  
strip  
(  
    """  
)
```

```
.  
strip  
(  
    """  
)
```

if

token

and

(
not

strip_joiners

or

token

```
.  
lower  
(
```

not

in

```
(  
'a'  
,
```

```
'and'
```



```
,  
  
'the'  
,  
  
'&'  
)):  
  
yield  
  
token  
  
\[docs\]  
  
def  
  
split_jobs  
(  
    self  
,  
  
    jobs  
,  
  
    num  
):  
  
    'Split a list of jobs into at most num groups, as evenly as possible'  
  
    groups  
  
    =  
  
    []  
  
    for  
  
    i
```

in

range

(

num

)]

jobs

=

list

(

jobs

)

while

jobs

:

for

gr

in

groups

:

try

:

job

=

jobs

```
.  
pop  
(  
  
except  
  
IndexError  
:
```

break

```
gr  
.  
append  
(  
job  
)
```

return

```
[  
g
```

for

```
g
```

in

```
groups
```

if

```
g  
]
```

[\[docs\]](#)

```
def
test_fields
(
self
,
mi
):

'''
    Return the first field from self.touched_fields that is null on the
    mi object
'''

for
key
in
self
.touched_fields
:

if
key
.startswith
(
'identifier:'
):

key
```

=

key

.

partition

(

'.'

)

[

-

1

]

if

not

mi

.

has_identifier

(

key

):

return

'identifier: '

+

+

key

elif

mi

.

is_null

(

key

):

```
return
```

```
key
```

```
\[docs\]
```

```
def
```

```
clean_downloaded_metadata
```

```
(
```

```
self
```

```
,
```

```
mi
```

```
):
```

```
'''
```

Call this method in your plugin's identify method to normalize metadata

before putting the Metadata object into result_queue. You can of

course, use a custom algorithm suited to your metadata source.

```
'''
```

```
docase
```

```
=
```

```
mi
```

```
.
```

```
language
```

```
==
```

```
'eng'
```

or

mi

.

is_null

(

'language'

)

if

docase

and

mi

.

title

:

mi

.

title

=

fixcase

(

mi

.

title

)

mi

.

authors

=

fixauthors

```
(  
  mi  
  .  
  authors  
)
```

if

```
mi  
.  
tags
```

and

```
docase  
:
```

```
mi  
.  
tags
```

```
=
```

```
list  
(  
  map  
(  
    fixcase  
  ,
```

```
mi  
.  
tags  
)
```

```
mi  
.  
isbn
```

```
=
```



```
check_isbn
```

```
(
```

```
mi
```

```
.
```

```
isbn
```

```
)
```

```
def
```

```
download_multiple_covers
```

```
(
```

```
self
```

```
,
```

```
title
```

```
,
```

```
authors
```

```
,
```

```
urls
```

```
,
```

```
get_best_cover
```

```
,
```

```
timeout
```

```
,
```

```
result_queue
```

```
,
```

```
abort
```

```
,
```

```
log
```

```
,
```

```
prefs_name
```

```
=
```

```
'max_covers'
```

```
):  
  
if  
not  
urls  
:  
  
log  
(  
    'No images found for, title:  
    %r  
    and authors:  
    %r  
,  
    %  
(  
title  
,  
authors  
)  
  
return  
  
from  
threading  
import  
Thread  
  
import  
time
```

```
if

prefs_name
:

urls

=

urls
[:
self
.
prefs
[
prefs_name
]]

if

get_best_cover
:

urls

=

urls
[:
1
]

log
(
'Downloading
%d
covers'
%
```

```
len
(
urls
))

workers

=

[
Thread
(
target
=
self
.
download_image
,
args
=
(
u
,
timeout
,
log
,
result_queue
))
for
u
in
urls
]
```

```
for
```

```
w
```

```
in
```

```
workers
```

```
:
```

```
w
```

```
.
```

```
daemon
```

```
=
```

```
True
```

```
w
```

```
.
```

```
start
```

```
()
```

```
alive
```

```
=
```

```
True
```

```
start_time
```

```
=
```

```
time
```

```
.
```

```
time
```

```
()
```

while

alive

and

not

abort

.

is_set

()

and

time

.

time

()

-

start_time

<

timeout

:

alive

=

False

for

w

in

```
workers
:
```

```
if
```

```
w
```

```
.
```

```
is_alive  
():
```

```
alive
```

```
=
```

```
True
```

```
break
```

```
abort
```

```
.
```

```
wait  
(  
0.1  
)
```

```
def
```

```
download_image
```

```
(
```

```
self
```

```
,
```

```
url
```

```
,
```

```
timeout
```

```
,  
  
log  
,  
  
result_queue  
):
```

```
try  
:
```

```
ans
```

```
=
```

```
self
```

```
.  
browser
```

```
.  
open_novisit  
(  
url  
,
```

```
timeout  
=  
timeout  
)
```

```
.  
read  
()
```

```
result_queue
```

```
.  
put  
((  
self  
,
```

```
ans
```



```
))

log
(
'Downloaded cover from:
%s
'
%
url
)

except

Exception
:

self
.
log
.
exception
(
'Failed to download cover from:
%r
'
%
url
)

# }}}

# Metadata API {{{
```

[\[docs\]](#)

def

get_book_url

(

self

,

identifiers

):

'''

Return a 3-tuple or None. The 3-tuple is of the form:

(identifier_type, identifier_value, URL).

The URL is the URL for the book identified by identifiers at this

source. identifier_type, identifier_value specify the identifier

corresponding to the URL.

This URL must be browsable to by a human using a browser. It is meant

to provide a clickable link for the user to easily visit the books page

at this source.

If no URL is found, return None. This method must be quick, and

consistent, so only implement it if it is possible to construct the URL

from a known scheme given identifiers.

'''

return

None

[\[docs\]](#)

def

get_book_url_name

(

self

,

idtype

,

idval

,

url

):

'''

Return a human readable name from the return value of get_book_url().

'''

return

self

.

name

[\[docs\]](#)

def

get_book_urls

(

self

```
,
identifiers
):

'''
    Override this method if you would like to return multiple urls for this book.

    Return a list of 3-tuples. By default this method simply calls :func:`get_book_url`.
'''

data

=

self
.
get_book_url
(
    identifiers
)

if
data
is
None
:

return

()

return
```

```
(  
data  
,)
```

[\[docs\]](#)

```
def
```

```
get_cached_cover_url
```

```
(  
self  
,
```

```
identifiers  
):
```

```
'''
```

Return cached cover URL for the book identified by

the identifiers dict or None if no such URL exists.

Note that this method must only return validated URLs, i.e. not URLs

that could result in a generic cover image or a not found error.

```
'''
```

```
return
```

```
None
```

[\[docs\]](#)

```
def
```

```
id_from_url
```

```
(
    self
    ,
    url
):

    """
        Parse a URL and return a tuple of the form:

        (identifier_type, identifier_value).

        If the URL does not match the pattern for the metadata source,

        return None.
    """
```

return

None

[\[docs\]](#)

def

identify_results_keygen

```
(
    self
    ,
```

title

=

None

,

authors

=

None

```

',

identifiers
=
{}):

'''

    Return a function that is used to generate a key that can sort Metadata
    objects by their relevance given a search query (title, authors,
    identifiers).

    These keys are used to sort the results of a call to :meth:`identify`.

    For details on the default algorithm see
    :class:`InternalMetadataCompareKeyGen`. Re-implement this function in
    your plugin if the default algorithm is not suitable.

'''

def
keygen
(
mi
):

return

InternalMetadataCompareKeyGen
(
mi
,

```

```
self
,
title
,
authors
,
identifiers
)
```

```
return
```

```
keygen
```

```
\[docs\]
```

```
def
```

```
identify
(
self
,
log
,
result_queue
,
abort
,
title
=
None
,
```



```
authors
```

```
=
```

```
None
```

```
,
```

```
identifiers
```

```
=
```

```
{},
```

```
timeout
```

```
=
```

```
30
```

```
):
```

```
'''
```

Identify a book by its Title/Author/ISBN/etc.

If identifiers(s) are specified and no match is found and this metadata source does not store all related identifiers (for example, all ISBNs of a book), this method should retry with just the title and author (assuming they were specified).

If this metadata source also provides covers, the URL to the cover should be cached so that a subsequent call to the get covers API with the same ISBN/special identifier does not need to get the cover URL again. Use the caching API for this.

Every Metadata object put into result_queue by this method must have a `source\_relevance` attribute that is an integer indicating the order in

which the results were returned by the metadata source for this query.

This integer will be used by :meth:`compare\_identify\_results`. If the order is unimportant, set it to zero for every result.

Make sure that any cover/ISBN mapping information is cached before the Metadata object is put into result_queue.

:param log: A log object, use it to output debugging information/errors

:param result_queue: A result Queue, results should be put into it.

Each result is a Metadata object

:param abort: If abort.is_set() returns True, abort further processing and return as soon as possible

:param title: The title of the book, can be None

:param authors: A list of authors of the book, can be None

:param identifiers: A dictionary of other identifiers, most commonly
*{'isbn':'1234...'}
...*

:param timeout: Timeout in seconds, no network request should hang for longer than timeout.

:return: None if no errors occurred, otherwise a unicode representation of the error suitable for showing to the user

'''

return

None

[\[docs\]](#)

def

download_cover

(

self

,

log

,

result_queue

,

abort

,

title
=

None

,

authors
=

None

,

identifiers
=

{},

timeout
=

30

,

get_best_cover

```

=
False
):

'''
    Download a cover and put it into result_queue. The parameters all have
    the same meaning as for :meth:`identify`. Put (self, cover_data) into
    result_queue.

    This method should use cached cover URLs for efficiency whenever
    possible. When cached data is not present, most plugins simply call
    identify and use its results.

    If the parameter get_best_cover is True and this plugin can get
    multiple covers, it should only get the "best" one.
'''

pass

# }}}

```

Source code for calibre.ebooks.oeb.polish.container

```
#!/usr/bin/env python
```

```
# License: GPLv3 Copyright: 2013, Kovid Goyal <kovid at kovidgoyal.net>
```

```
import
```

```
errno
```

```
import
```

```
hashlib
```

```
import
```

```
logging
```

```
import
```

```
os
```

```
import
```

```
re
```

```
import
```

```
shutil
```

```
import
```

```
sys
```

import

unicodedata

import

uuid

from

collections

import

defaultdict

from

css_parser

import

getUrls

,

replaceUrls

from

io

import

BytesIO

from

itertools

import

count

```
from
```

```
calibre
```

```
import
```

```
CurrentDir
```

```
,
```

```
walk
```

```
from
```

```
calibre.constants
```

```
import
```

```
iswindows
```

```
from
```

```
calibre.customize.ui
```

```
import
```

```
plugin_for_input_format
```

```
,
```

```
plugin_for_output_format
```

```
from
```

```
calibre.ebooks
```

```
import
```

```
escape_xpath_attr
```

```
from
```

```
calibre.ebooks.chardet
```

```
import
xml_to_unicode

from
calibre.ebooks.conversion.plugins.epub_input

import

(

ADOBE_OBFUSCATION
,

IDPF_OBFUSCATION
,

decrypt_font_data

)

from
calibre.ebooks.conversion.preprocess

import

(

CSSPreProcessor

as

cssp
,

HTMLPreProcessor

)
```


from

calibre.ebooks.metadata.opf3

import

(

CALIBRE_PREFIX

,

ensure_prefix

,

items_with_property

,

read_prefixes

)

from

calibre.ebooks.metadata.utils

import

parse_opf_version

from

calibre.ebooks.mobi

import

MobiError

from

calibre.ebooks.mobi.reader.headers

```
import
```

```
MetadataHeader
```

```
from
```

```
calibre.ebooks.mobi.tweak
```

```
import
```

```
set_cover
```

```
from
```

```
calibre.ebooks.oeb.base
```

```
import
```

```
(
```

```
DC11_NS
```

```
,
```

```
OEB_DOCS
```

```
,
```

```
OEB_STYLES
```

```
,
```

```
OPF
```

```
,
```

```
OPF2_NS
```

```
,
```

```
Manifest
```

```
,
```

```
itercsslinks
```

```
,
```

```
iterlinks
```

```
,  
  
rewrite_links  
,  
  
serialize  
,  
  
urlquote  
,  
  
urlunquote  
)  
  
from  
  
calibre.ebooks.oeb.parse_utils  
  
import  
  
NotHTML  
,  
  
parse_html  
  
from  
  
calibre.ebooks.oeb.polish.errors  
  
import  
  
DRMError  
,  
  
InvalidBook  
  
from  
  
calibre.ebooks.oeb.polish.parsing  
  
import
```

```
parse
```

```
as
```

```
parse_html_tweak
```

```
from
```

```
calibre.ebooks.oeb.polish.utils
```

```
import
```

```
(
```

```
CommentFinder
```

```
,
```

```
PositionFinder
```

```
,
```

```
guess_type
```

```
,
```

```
parse_css
```

```
)
```

```
from
```

```
calibre.ptempfile
```

```
import
```

```
PersistentTemporaryDirectory
```

```
,
```

```
PersistentTemporaryFile
```

```
from
```

```
calibre.utils\_filenames
```

import

hardlink_file

,

nlinks_file

,

retry_on_fail

from

[calibre.utils.ipc.simple_worker](#)

import

WorkerError

,

fork_job

from

[calibre.utils.logging](#)

import

default_log

from

[calibre.utils.xml_parse](#)

import

safe_xml_fromstring

from

[calibre.utils.zipfile](#)

import

```
ZipFile
```

```
from
```

```
polyglot.builtins
```

```
import
```

```
iteritems
```

```
from
```

```
polyglot.urllib
```

```
import
```

```
urlparse
```

```
exists
```

```
,
```

```
join
```

```
,
```

```
relpath
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
exists
```

```
,
```

```
os
```

```
.
```

```
path
```

```
.
```

```
join
```

```
,
```

```
os
```

```
.
```

```
path
```

```
.
```

```
relpath
```

```
OEB_FONTS
```

```
=
```

```
{
```

```
guess_type
```

```
(
```

```
'a.ttf'
```

```
),
```

```
guess_type
```

```
(
```

```
'b.otf'
```

```
),
```

```
guess_type
```

```
(
```

```
'a.woff'
```

```
),
```

```
'application/x-font-ttf'
```

```
,
```

```
'application/x-font-otf'
```

```
,
```

```
'application/font-sfnt'
```

```
}
```

```
OPF_NAMESPACES
```

```
=
```

```
{
```

```
'opf'
```

```
:
OPF2_NS
,

'dc'
:
DC11_NS
}

null

=

object
()

class

CSSPreProcessor
(
cssp
):

def

__call__
(
self
,

data
):

return

self
.
MS_PAT
```



```
.
sub
(
self
.
ms_sub
,
data
)

def
clone_dir
(
src
,
dest
):

' Clone a folder using hard links for the files, dest must already exist '

for
x
in
os
.listdir
(
src
):
dpath
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
join
```

```
(
```

```
dest
```

```
,
```

```
x
```

```
)
```

```
spath
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
join
```

```
(
```

```
src
```

```
,
```

```
x
```

```
)
```

```
if
```

```
os
```

```
.
```

```
path
```

```
.
```

```
isdir
```

```
(
```

```
spath
```

```
):
```

```
os
.
mkdir
(
dpath
)

clone_dir
(
spath
,
dpath
)

else
:

try
:

hardlink_file
(
spath
,
dpath
)

except
:

shutil
.
copy2
(
```

```
spath
',

dpath
)

def

clone_container
(
container
,

dest_dir
):

' Efficiently clone a container using hard links '

dest_dir

=

os
.
path
.
abspath
(
os
.
path
.
realpath
(
dest_dir
))

clone_data
```

```
=  
  
container  
.  
clone_data  
(  
    dest_dir  
)
```

```
cls
```

```
=  
  
type  
(  
    container  
)
```

```
if
```

```
cls
```

```
is
```

```
Container  
:
```

```
return
```

```
cls  
(  
    None  
,
```

```
None  
,
```

```
container  
.
```

```
log
,

clone_data
=
clone_data
)

return

cls
(
None
,

container
.
log
,

clone_data
=
clone_data
)

def

name_to_abspath
(
name
,

root
):

return

os
.
```

```
path
.  
abspath  
(  
join  
(  
root  
,  
  
*  
name  
.  
split  
(  
'/'  
)))
```

def

```
abspath_to_name  
(  
path  
,  
  
root  
):
```

return

```
relpath  
(  
os  
.  
path  
.  
abspath  
(  
path  
),
```

```
root
)
```

```
.
replace
```

```
(
os
```

```
.
sep
```

```
,
```

```
'/'
```

```
)
```

```
def
```

```
name_to_href
```

```
(
```

```
name
```

```
,
```

```
root
```

```
,
```

```
base
```

```
=
```

```
None
```

```
,
```

```
quote
```

```
=
```

```
urlquote
```

```
):
```

```
fullpath
```

```
=
```

```
name_to_abspath
```

```
(
```

```
name
```



```
,  
  
root  
)  
  
basepath  
  
=  
  
root  
  
if  
  
base  
  
is  
  
None  
  
else  
  
os  
.  
path  
.  
dirname  
(  
name_to_abspath  
(  
base  
,  
root  
))  
  
path  
  
=  
  
relpath  
(
```

```
fullpath
,

basepath
)
.
replace
(
os
.
sep
,
'/'
)

return

quote
(
path
)

def
href_to_name
(
href
,
root
,
base
=
None
):

base
```

```
=  
  
root  
  
if  
  
base  
  
is  
  
None  
  
else  
  
os  
.  
path  
.  
dirname  
(  
name_to_abspath  
(  
base  
,  
root  
))  
  
try  
:  
  
purl  
  
=  
  
urlparse  
(  
href  
)
```

```
except
```

```
ValueError
```

```
:
```

```
return
```

```
None
```

```
if
```

```
purl
```

```
.
```

```
scheme
```

```
or
```

```
not
```

```
purl
```

```
.
```

```
path
```

```
:
```

```
return
```

```
None
```

```
href
```

```
=
```

```
urlunquote
```

```
(
```

```
purl
```

```
.
```

```
path
```

```
)
```

```
if
```

```
iswindows
```

```
and
```

```
':'
```

```
in
```

```
href
```

```
:
```

```
# path manipulations on windows fail for paths with : in them, so we
```

```
# assume all such paths are invalid/absolute paths.
```

```
return
```

```
None
```

```
fullpath
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
join
```

```
(
```

```
base
```

```
,
```

```
*
```

```
href
```

```
.
```

```
split  
(  
    '/'  
))
```

```
return
```

```
unicodedata
```

```
.
```

```
normalize
```

```
(
```

```
'NFC'
```

```
,
```

```
abspath_to_name
```

```
(
```

```
fullpath
```

```
,
```

```
root
```

```
))
```

```
class
```

```
ContainerBase
```

```
:
```

```
# {{{
```

```
'''
```

A base class that implements just the parsing methods. Useful to create

virtual containers for testing.

```
'''
```

#: The mode used to parse HTML and CSS (polishing uses tweak_mode=False and the editor uses tweak_mode=True)

tweak_mode

=

False

def

__init__

(

self

,

log

):

self

.

log

=

log

self

.

parsed_cache

=

}

self

.

```
mime_map

=

{}

self
.
encoding_map

=

{}

self
.
html_preprocessor

=

HTMLPreProcessor
()

self
.
css_preprocessor

=

CSSPreProcessor
()

def
guess_type
(
self
,
```



```
name
):
```

```
' Return the expected mimetype for the specified file name based on its extension. '
```

```
# epubcheck complains if the mimetype for text documents is set to
```

```
# text/html in EPUB 2 books. Sigh.
```

```
ans
```

```
=
```

```
guess_type
(
name
)
```

```
if
```

```
ans
```

```
==
```

```
'text/html'
:
```

```
ans
```

```
=
```

```
'application/xhtml+xml'
```

```
if
```

```
ans
```

```
in
```

```
{  
'application/x-font-truetype'  
,  
  
'application/vnd.ms-opentype'  
};
```

```
opfversion
```

```
=
```

```
self  
.  
opf_version_parsed  
[:  
2  
]
```

```
if
```

```
opfversion
```

```
>
```

```
(  
3  
,  
  
0  
):
```

```
return
```

```
'application/font-sfnt'
```

```
if
    opfversion
    >=
    (
    3
    ,
    0
    ):

    # bloody epubcheck has recently decided it likes this mimetype

    # for ttf files

    return

    'application/vnd.ms-opentype'

    return

    ans

    def
    decode
    (
    self
    ,
    data
    ,
    normalize_to_nfc
    =
```

True

):

"""

Automatically decode ``data`` into a ``unicode`` object.

:param normalize_to_nfc: Normalize returned unicode to the NFC normal form as is required by both the EPUB and AZW3 formats.

"""

def

fix_data

(

d

):

return

d

.

replace

(

'

`\r\n`

'

,

'

`\n`

'

)

.

replace

(

'

`\r`

```
'  
,  
'  
\n'  
)  
  
if  
  
isinstance  
(  
data  
,  
  
str  
):  
  
return  
  
fix_data  
(  
data  
)  
  
bom_enc  
  
=  
  
None  
  
if  
  
data  
[:  
4  
]  
  
in
```

```
{
  b
  '
  \0\0\xfe\xff
  '
  ,
  b
  '
  \xff\xfe\0\0
  '
}:

```

bom_enc

=

```
{
  b
  '
  \0\0\xfe\xff
  '
  :
  'utf-32-be'
  ,
  b
  '
  \xff\xfe\0\0
  '
  :
  'utf-32-le'
}[
data
[:
4
]]

```

data

```
=
```

```
data
```

```
[
```

```
4
```

```
:]
```

```
elif
```

```
data
```

```
[:
```

```
2
```

```
]
```

```
in
```

```
{
```

```
b
```

```
,
```

```
\xff\xfe
```

```
,
```

```
,
```

```
b
```

```
,
```

```
\xfe\xff
```

```
,
```

```
};
```

```
bom_enc
```

```
=
```

```
{
```

```
b
```

```
,
```

```
\xff\xfe
```

```
,
```

```
:
```

```
'utf-16-le'
```

```
,  
  
b  
,  
\xfe\xff  
,  
  
:  
'utf-16-be'  
}  
data  
[:  
2  
]]  
  
data  
  
=  
  
data  
[  
2  
:]  
  
elif  
  
data  
[:  
3  
]  
  
==  
  
b  
,  
\xef\xbb\xbf  
,  
  
:  
  
bom_enc
```



```
=  
  
'utf-8'  
  
data  
  
=  
  
data  
[  
3  
:]  
  
if  
  
bom_enc  
  
is  
  
not  
  
None  
:  
  
try  
:  
  
self  
.  
used_encoding  
  
=  
  
bom_enc  
  
return  
  
fix_data
```

```
(
data
.
decode
(
bom_enc
))

except

UnicodeDecodeError
:

pass

try
:

self
.
used_encoding
=
'utf-8'

return

fix_data
(
data
.
decode
(
'utf-8'
))
```

```
except
UnicodeDecodeError
:
```

```
pass
```

```
data
,
self
.
used_encoding
```

```
=
xml_to_unicode
(
data
)
```

```
if
normalize_to_nfc
:
```

```
data
=
unicodedata
```

```
.
normalize
(
'NFC'
,
```

```
data
)
```

```
return
```

```
fix_data  
(  
    data  
)
```

```
def
```

```
parse_xml  
(  
    self  
,  
  
    data  
):
```

```
    data  
    ,  
  
    self  
    .  
    used_encoding  
  
    =
```

```
    xml_to_unicode  
(
```

```
        data  
        ,
```

```
        strip_encoding_pats  
        =
```

```
True
```

```
,
```

```
assume_utf8
=
True
,

resolve_entities
=
True
)

data

=

unicodedata
.
normalize
(
'NFC'
,

data
)

return

safe_xml_fromstring
(
data
)

def

parse_xhtml
(
self
,

data
```

```
,  
  
fname  
=  
'<string>'  
,  
  
force_html5_parse  
=  
False  
):  
  
if  
  
self  
.  
tweak_mode  
:  
  
return  
  
parse_html_tweak  
(  
data  
,  
  
log  
=  
self  
.  
log  
,  
  
decoder  
=  
self  
.  
decode  
,  
  
force_html5_parse
```

```
=  
force_html5_parse  
)
```

```
else  
:
```

```
try  
:
```

```
return
```

```
parse_html  
(
```

```
data
```

```
,
```

```
log
```

```
=
```

```
self
```

```
.
```

```
log
```

```
,
```

```
decoder
```

```
=
```

```
self
```

```
.
```

```
decode
```

```
,
```

```
preprocessor
```

```
=
```

```
self
```

```
.
```

```
html_preprocessor
```

```
,
```

```
filename
=
fname
,

non_html_file_tags
=
{
'ncx'
})
```

except

```
NotHTML
:
```

return

```
self
.
parse_xml
(
data
)
```

def

```
parse_css
(
self
,

data
,

fname
=
```



```
'<string>'
,

is_declaration
=
False
):

return

parse_css
(
data
,

fname
=
fname
,

is_declaration
=
is_declaration
,

decode
=
self
.
decode
,

log_level
=
logging
.
WARNING
,

css_preprocessor
=
```

```
(  
None  
  
if  
  
self  
.  
tweak_mode  
  
else  
  
self  
.  
css_preprocessor  
)  
  
# }}
```

[\[docs\]](#)

class

Container

```
(  
ContainerBase  
):
```

```
# {{{
```

```
'''
```

A container represents an open e-book as a folder full of files and an

OPF file. There are two important concepts:

** The root folder. This is the base of the e-book. All the e-books*

files are inside this folder or in its sub-folders.

** Names: These are paths to the books' files relative to the root*

folder. They always contain POSIX separators and are unquoted. They

can be thought of as canonical identifiers for files in the book.

Most methods on the container object work with names. Names are always

in the NFC Unicode normal form.

** Clones: the container object supports efficient on-disk cloning, which is used to*

implement checkpoints in the e-book editor. In order to make this work, you should

never access files on the filesystem directly. Instead, use :meth:`raw\_data` or

:meth:`open` to read/write to component files in the book.

When converting between hrefs and names use the methods provided by this

class, they assume all hrefs are quoted.

'''

#: The type of book (epub for EPUB files and azw3 for AZW3 files)

book_type

=

'oeb'

#: If this container represents an unzipped book (a directory)

```
is_dir
```

```
=
```

```
False
```

```
SUPPORTS_TITLEPAGES
```

```
=
```

```
True
```

```
SUPPORTS_FILENAMES
```

```
=
```

```
True
```

```
@property
```

```
def
```

```
book_type_for_display
```

```
(
```

```
self
```

```
):
```

```
return
```

```
self
```

```
.
```

```
book_type
```

```
.
```

```
upper
```

```
()
```

```
def
```

```
    __init__
```

```
    (
```

```
    self
```

```
    ,
```

```
    rootpath
```

```
    ,
```

```
    opfpath
```

```
    ,
```

```
    log
```

```
    ,
```

```
    clone_data
```

```
    =
```

```
    None
```

```
):
```

```
ContainerBase
```

```
·
```

```
    __init__
```

```
    (
```

```
    self
```

```
    ,
```

```
    log
```

```
)
```

```
self
```

```
·
```

```
root
```

```
=
```

```
clone_data
```

```
[  
    'root'  
]  
  
if  
  
    clone_data  
  
is  
  
not  
  
None  
  
else  
  
    os  
    .  
    path  
    .  
    abspath  
    (  
        rootpath  
    )  
  
self  
    .  
    name_path_map  
    =  
  
    {}  
  
self  
    .  
    dirtied  
    =  
  
    set
```

```
()

self
.
pretty_print

=

set
()

self
.
cloned

=

False

self
.
cache_names

=

(
'parsed_cache'
,

'mime_map'
,

'name_path_map'
,

'encoding_map'
,

'dirtied'
,
```

```
'pretty_print'  
)
```

```
self  
.  
href_to_name_cache
```

```
=
```

```
{}
```

```
if
```

```
clone_data
```

```
is
```

```
not
```

```
None
```

```
:
```

```
self
```

```
.  
cloned
```

```
=
```

```
True
```

```
for
```

```
x
```

```
in
```

```
(
```



```
'name_path_map'  
,  
  
'opf_name'  
,  
  
'mime_map'  
,  
  
'pretty_print'  
,  
  
'encoding_map'  
,  
  
'tweak_mode'  
):
```

```
setattr
```

```
(  
self  
,
```

```
x  
,
```

```
clone_data
```

```
[  
x  
])
```

```
self
```

```
.  
opf_dir
```

```
=
```

```
os
```

```
.  
path
```

```
.
```

```
dirname
(
self
.
name_path_map
[
self
.
opf_name
])

return

# Map of relative paths with '/' separators from root of unzipped ePub

# to absolute paths on filesystem with os-specific separators

opfpath

=

os
.
path
.
abspath
(
os
.
path
.
realpath
(
opfpath
))

all_opf_files
```

```
=  
  
[]  
  
for  
  
    dirpath  
    ,  
    _dirnames  
    ,  
    filenames  
  
    in  
  
    os  
    .  
    walk  
    (  
    self  
    .  
    root  
    ):  
  
    for  
  
        f  
  
        in  
  
            filenames  
            :  
  
                path  
  
                =  
  
                join
```

```
(
dirpath
,
f
)

name

=

self
.
abspath_to_name
(
path
)

self
.
name_path_map
[
name
]

=

path

self
.
mime_map
[
name
]

=

guess_type
(
```

```
path
)

# Special case if we have stumbled onto the opf

if
path

==

opfpath
:

self
.
opf_name

=

name

self
.
opf_dir

=

os
.
path
.
dirname
(
path
)

self
```

```
.  
mime_map  
[  
name  
]
```

```
=
```

```
guess_type  
(  
'a.opf'  
)
```

```
if
```

```
path
```

```
.  
lower  
()
```

```
.  
endswith  
(  
'opf'  
):
```

```
all_opf_files
```

```
.  
append  
((  
name  
,
```

```
os
```

```
.  
path
```

```
.  
dirname  
(  
path  
)))
```

```
if
not
hasattr
(
self
,
'opf_name'
)
and
all_opf_files
:
self
.
opf_name
,
self
.
opf_dir
=
all_opf_files
[
0
]
self
.
mime_map
[
self
.
```

```
opf_name
]

=

guess_type
(
'a.opf'
)

if

not

hasattr
(
self
,
'opf_name'
):

raise

InvalidBook
(
'Could not locate opf file:
%r
'
%
opfpath
)

# Update mime map with data from the OPF

self
.
```



```
refresh_mime_map
()

def

refresh_mime_map
(
self
):

for

item

in

self
.
opf_xpath
(
'//opf:manifest/opf:item[@href and @media-type]'
):

href

=

item
.
get
(
'href'
)

name

=
```

```
self
.
href_to_name
(
href
,

self
.
opf_name
)

mt

=

item
.
get
(
'media-type'
)

if

name

in

self
.
mime_map

and

name

!=

self
.
```

```
opf_name
```

```
and
```

```
mt
```

```
:
```

```
# some epub3s include the opf in the manifest with an incorrect mime type
```

```
self
```

```
.
```

```
mime_map
```

```
[
```

```
name
```

```
]
```

```
=
```

```
mt
```

```
def
```

```
data_for_clone
```

```
(
```

```
self
```

```
,
```

```
dest_dir
```

```
=
```

```
None
```

```
):
```

```
dest_dir
```

```
=
```

```
dest_dir
```

```
or

self
.
root

return

{

'root'
:

dest_dir
,

'opf_name'
:

self
.
opf_name
,

'mime_map'
:

self
.
mime_map
.
copy
(),

'pretty_print'
:

set
```

```
(
self
.
pretty_print
),

'encoding_map'
:

self
.
encoding_map
.
copy
()),

'tweak_mode'
:

self
.
tweak_mode
,

'name_path_map'
:

{

name
:
os
.
path
.
join
(
dest_dir
,
```

```
os
.
path
.
relpath
(
path
,

self
.
root
))

for

name
,

path

in

iteritems
(
self
.
name_path_map
)}

}

def

clone_data
(
self
,
```

```
dest_dir
):

Container
.
commit
(
self
,

keep_parsed
=
True
)

self
.
cloned
=
True

clone_dir
(
self
.
root
,

dest_dir
)

return

self
.
data_for_clone
```

```
(
dest_dir
)
```

[\[docs\]](#)

def

add_name_to_manifest

```
(
self
```

```
,
```

name

```
,
```

process_manifest_item

```
=
```

None

```
):
```

' Add an entry to the manifest for a file with the specified name. Returns the manifest id. '

all_ids

```
=
```

```
{
```

x

```
.
```

get

```
(
```

'id'

```
)
```

for

x

in

self

.

opf_xpath

(

'//*[@id]'

)}

c

=

0

item_id

=

'id'

while

item_id

in

all_ids

:

c

+=

1

item_id

```
=
```

```
'id'
```

```
+
```

```
,
```

```
%d
```

```
,
```

```
%
```

```
c
```

```
manifest
```

```
=
```

```
self
```

```
.
```

```
opf_xpath
```

```
(
```

```
'//opf:manifest'
```

```
)[
```

```
0
```

```
]
```

```
href
```

```
=
```

```
self
```

```
.
```

```
name_to_href
```

```
(
```

```
name
```

```
,
```

```
self
```

```
.
```

```
opf_name
```

```
)
```

```
item
```

```
=
```

```
manifest
```

```
.
```

```
makeelement
```

```
(
```

```
OPF
```

```
(
```

```
'item'
```

```
),
```

```
id
```

```
=
```

```
item_id
```

```
,
```

```
href
```

```
=
```

```
href
```

```
)
```

```
item
```

```
.
```

```
set
```

```
(
```

```
'media-type'
```

```
,
```

```
self
```

```
.
```

```
mime_map
```

```
[
```

```
name
```

```
])
```

```
self
```

```
.
insert_into_xml
(
manifest
,

item
)

if

process_manifest_item

is

not

None
:

process_manifest_item
(
item
)

self
.
dirty
(
self
.
opf_name
)

return

item_id
```

[\[docs\]](#)

def

manifest_has_name

(
self
,

name

):

''' Return True if the manifest has an entry corresponding to name '''

all_names

=

{
self

.

href_to_name

(

x

.

get

(

'href'

),

self

.

opf_name

)

for

x

in

```
self
.  
opf_xpath  
(  
    '//opf:manifest/opf:item[@href]'  
)}
```

```
return
```

```
name
```

```
in
```

```
all_names
```

```
\[docs\]
```

```
def
```

```
make_name_unique
```

```
(  
    self  
    ,
```

```
    name  
):
```

```
""" Ensure that `name` does not already exist in this book. If it does, return a modified version  
that does not exist. """
```

```
counter
```

```
=
```

```
count  
()
```

```
while
```

```
self
.  
has_name_case_insensitive  
(  
name  
)
```

or

```
self
.  
manifest_has_name  
(  
name  
):
```

c

=

```
next  
(  
counter  
)
```

+

1

base

,

ext

=

name

```
.  
rpartition  
(
```

```
'  
)[::  
2  
]  
  
if  
  
c  
  
>  
  
1  
:  
  
base  
  
=  
  
base  
.  
rpartition  
(  
'_  
)[  
0  
]  
  
name  
  
=  
  
'  
%s  
-  
%d  
.  
%s  
'  
  
%
```



```
(
    base
    ,

    c
    ,

    ext
)

return

name

\[docs\]

def

add_file
(
    self
    ,

    name
    ,

    data
    ,

    media_type
    =
    None
    ,

    spine_index
    =
    None
    ,

    modify_name_if_needed
```

```
=
False
,

process_manifest_item
=
None
):

    """Add a file to this container. Entries for the file are

        automatically created in the OPF manifest and spine

        (if the file is a text document) """

    if
        '..'

    in
        name
        :

    raise
        ValueError
        (
            'Names are not allowed to have .. in them'
        )

    href

    =

    self
    .
    name_to_href
    (
```

```
name
,

self
.
opf_name
)

if

self
.
has_name_case_insensitive
(
name
)

or

self
.
manifest_has_name
(
name
):

if

not

modify_name_if_needed
:

raise

ValueError
((
'A file with the name
%s
already exists'
```

```
%  
  
name  
)  
  
if  
  
self  
.  
has_name_case_insensitive  
(  
name  
)  
  
else  
  
(  
'An item with the href  
%s  
already exists in the manifest'  
  
%  
  
href  
)  
  
name  
  
=  
  
self  
.  
make_name_unique  
(  
name  
)  
  
href
```

```
=  
  
self  
.  
name_to_href  
(  
    name  
,  
    self  
.  
opf_name  
)  
  
path  
  
=  
  
self  
.  
name_to_abspath  
(  
    name  
)  
  
base  
  
=  
  
os  
.  
path  
.  
dirname  
(  
    path  
)  
  
if
```

not

os

.

path

.

exists

(

base

):

os

.

makedirs

(

base

)

with

lopen

(

path

,

'wb'

)

as

f

:

if

hasattr

(

data

,

```
'read'
):

shutil
.
copyfileobj
(
data
,

f
)

else
:

f
.
write
(
data
)

mt

=

media_type

or

self
.
guess_type
(
name
)
```

```
self
.  
name_path_map  
[  
name  
]
```

```
=
```

```
path
```

```
self
.  
mime_map  
[  
name  
]
```

```
=
```

```
mt
```

```
if
```

```
self
.  
ok_to_be_unmanifested  
(  
name  
):
```

```
return
```

```
name
```

```
item_id
```

```
=
```



```
self
.  
add_name_to_manifest  
(  
name  
,  
  
process_manifest_item  
=  
process_manifest_item  
)
```

```
if
```

```
mt
```

```
in
```

```
OEB_DOCS  
:
```

```
manifest
```

```
=
```

```
self  
.  
opf_xpath  
(  
'//opf:manifest'  
)[  
0  
]
```

```
spine
```

```
=
```

```
self  
.
```

```
opf_xpath  
(  
    '//opf:spine'  
)[  
    0  
]
```

```
si
```

```
=
```

```
manifest
```

```
.  
makeelement  
(  
    OPF  
(  
    'itemref'  
),
```

```
idref
```

```
=
```

```
item_id  
)
```

```
self
```

```
.  
insert_into_xml  
(  
    spine  
,
```

```
si
```

```
,
```

```
index
```

```
=
```

```
spine_index  
)
```

```
return
```

```
name
```

```
\[docs\]
```

```
def
```

```
rename
```

```
(  
self  
,
```

```
current_name
```

```
,
```

```
new_name
```

```
):
```

```
''' Renames a file from current_name to new_name. It automatically
```

```
rebases all links inside the file if the folder the file is in
```

```
changes. Note however, that links are not updated in the other files
```

```
that could reference this file. This is for performance, such updates
```

```
should be done once, in bulk. '''
```

```
if
```

```
current_name
```

```
in
```

```
self
```

```
.
```

```
names_that_must_not_be_changed
```

```
:
```

raise

ValueError

(
'Renaming of
%s
is not allowed'

%

current_name
)

if

self

.
exists

(
new_name
)

and

(
new_name

==

current_name

or

new_name

.
lower
()

!=

current_name

```
.
lower
()):

# The destination exists and does not differ from the current name only by case

raise

ValueError
(
f
'Cannot rename
{
current_name
}
to
{
new_name
}
as
{
new_name
}
already exists'
)

new_path

=

self
.
name_to_abspath
(
new_name
)

base
```

```
=  
  
os  
.path  
dirname  
(  
new_path  
)  
  
if  
  
os  
.path  
.isfile  
(  
base  
):  
  
raise  
  
ValueError  
(  
f  
'Cannot rename  
{  
current_name  
}  
to  
{  
new_name  
}  
as  
{  
base  
}  
is a file'  
)
```

```
if
```

```
not
```

```
os
```

```
.
```

```
path
```

```
.
```

```
exists
```

```
(
```

```
base
```

```
):
```

```
os
```

```
.
```

```
makedirs
```

```
(
```

```
base
```

```
)
```

```
old_path
```

```
=
```

```
parent_dir
```

```
=
```

```
self
```

```
.
```

```
name_to_abspath
```

```
(
```

```
current_name
```

```
)
```

```
self
```

```
.
```

```
commit_item
```

```
(  
current_name  
)
```

```
os
```

```
.
```

```
rename
```

```
(
```

```
old_path
```

```
,
```

```
new_path
```

```
)
```

```
# Remove empty directories
```

```
while
```

```
parent_dir
```

```
:
```

```
parent_dir
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
dirname
```

```
(
```

```
parent_dir
```

```
)
```

```
try
```

```
:
```



```
os
.  
rmdir  
(  
parent_dir  
)  
  
except  
  
OSError  
:  
  
break  
  
for  
  
x  
  
in  
  
(  
'mime_map'  
,  
  
'encoding_map'  
):  
  
x  
  
=  
  
getattr  
(  
self  
,  
  
x  
)
```

if

current_name

in

x

:

x

[

new_name

]

=

x

[

current_name

]

self

.

name_path_map

[

new_name

]

=

new_path

for

x

in

```
self
.
cache_names
:

x

=

getattr
(
self
,
x
)

try
:

x
.
pop
(
current_name
,
None
)

except
TypeError
:

x
.
discard
```

```
(  
current_name  
)
```

```
if
```

```
current_name
```

```
==
```

```
self
```

```
.  
opf_name  
:
```

```
self
```

```
.  
opf_name
```

```
=
```

```
new_name
```

```
if
```

```
os
```

```
.  
path
```

```
.  
dirname
```

```
(  
old_path  
)
```

```
!=
```

```
os
```

```
.  
path
```

```
.
```

```
dirname
(
new_path
):

from

calibre.ebooks.oeb.polish.replace

import

LinkRebaser

repl

=

LinkRebaser
(
self
,

current_name
,

new_name
)

self
.
replace_links
(
new_name
,

repl
)

self
```

```
.
dirty
(
new_name
)
```

[\[docs\]](#)

def

```
replace_links
(
self
,
name
,
replace_func
):
```

""" Replace all links in name using replace_func, which must be a callable that accepts a URL and returns the replaced URL. It must also have a 'replaced' attribute that is set to True if any actual replacement is done. Convenient ways of creating such callables are using the :class:`LinkReplacer` and :class:`LinkRebaser` classes. """

```
media_type
```

```
=
```

```
self
```

```
.
```

```
mime_map
```

```
.
```

```
get
```

```
(
name
,
guess_type
(
name
))

if

name

==

self
.
opf_name
:

replace_func
.
file_type

=

'opf'

for

elem

in

self
.
opf_xpath
(
'//*[@href]'
):
```

```
elem
.
set
(
'href'
,
replace_func
(
elem
.
get
(
'href'
)))
```

elif

```
media_type
.
lower
()
```

in

```
OEB_DOCS
:
```

```
replace_func
.
file_type
=
```

```
'text'
```

```
rewrite_links
(
```



```
self
.  
parsed  
(  
name  
)  
  
replace_func  
)
```

elif

```
media_type  
.  
lower  
( )
```

in

```
OEB_STYLES  
:
```

```
replace_func  
.  
file_type  
  
=
```

```
'style'
```

```
replaceUrls  
(  
self  
.  
parsed  
(  
name  
)  
,
```

```
replace_func
```

```
)

elif

media_type

.
lower
()

==

guess_type
(
'toc.ncx'
):

replace_func
.
file_type

=

'ncx'

for

elem

in

self

.
parsed
(
name
)
.
xpath
(
'//*[@src]'
```

```
):  
  
    elem  
    .  
    set  
    (  
    'src'  
    ,  
    replace_func  
    (  
    elem  
    .  
    get  
    (  
    'src'  
    )))  
  
    if  
  
    replace_func  
    .  
    replaced  
    :  
  
    self  
    .  
    dirty  
    (  
    name  
    )  
  
    return  
  
    replace_func  
    .  
    replaced
```

[\[docs\]](#)

def

iterlinks

(

self

,

name

,

get_line_numbers

=

True

):

""" Iterate over all links in name. If get_line_numbers is True the

yields results of the form (link, line_number, offset). Where

line_number is the line_number at which the link occurs and offset is

the number of characters from the start of the line. Note that offset

could actually encompass several lines if not zero. """

media_type

=

self

.

mime_map

.

get

(

name

,

guess_type

```
(
    name
))

if

name

==

self
.
opf_name
:

for

elem

in

self
.
opf_xpath
(
    '//*[@href]'
):

yield

(
    elem
    .
    get
    (
        'href'
    ),
    elem
    .
```

```
sourceline
,
0
)

if

get_line_numbers

else

elem
.
get
(
'href'
)

elif

media_type
.
lower
()

in

OEB_DOCS
:

for

el
,

attr
,

link
,
```

```
pos
```

```
in
```

```
iterlinks
```

```
(
```

```
self
```

```
.
```

```
parsed
```

```
(
```

```
name
```

```
)):
```

```
yield
```

```
(
```

```
link
```

```
,
```

```
el
```

```
.
```

```
sourceline
```

```
,
```

```
pos
```

```
)
```

```
if
```

```
get_line_numbers
```

```
else
```

```
link
```

```
elif
```

```
media_type
```

```
.
```

```
lower
```

```
()

in

OEB_STYLES
:

if

get_line_numbers
:

with

self
.
open
(
name
,
'rb'
)

as

f
:

raw

=

self
.
decode
(
f
.
read
```



```
()
```

```
.
```

```
replace
```

```
(
```

```
'
```

```
\r\n
```

```
'
```

```
,
```

```
'
```

```
\n
```

```
'
```

```
)
```

```
.
```

```
replace
```

```
(
```

```
'
```

```
\r
```

```
'
```

```
,
```

```
'
```

```
\n
```

```
'
```

```
)
```

```
.
```

```
position
```

```
=
```

```
PositionFinder
```

```
(
```

```
raw
```

```
)
```

```
is_in_comment
```

```
=
```

```
CommentFinder
```

```
(
```

```
raw
)

for

link
,

offset

in

itercsslinks
(
raw
):

if

not

is_in_comment
(
offset
):

Inum
,

col

=

position
(
offset
)

yield
```

```
link
,

Inum
,

col

else
:

for

link

in

getUrls
(
    self
    .
    parsed
    (
        name
    )):

yield

link

elif

media_type
.
lower
()

==
```

```
guess_type  
(  
    'toc.ncx'  
):
```

```
for
```

```
    elem
```

```
in
```

```
    self
```

```
    .  
    parsed  
    (  
        name  
    )
```

```
    .  
    xpath  
    (  
        '//*[@src]'  
    ):
```

```
yield
```

```
(  
    elem
```

```
    .  
    get  
    (  
        'src'  
    ),
```

```
    elem
```

```
    .  
    sourceline  
,
```

```
    0  
)
```

```
if
    get_line_numbers

else
    elem
    .
    get
    (
        'src'
    )

    [docs]

def
    abspath_to_name
    (
        self
        ,
        fullpath
        ,
        root
    )
    =
    None
):

'''
    Convert an absolute path to a canonical name relative to :attr:`root`

    :param root: The base folder. By default the root for this container object is used.

    '''
```

```
# OS X silently changes all file names to NFD form. The EPUB  
  
# spec requires all text including filenames to be in NFC form.  
  
# The proper fix is to implement a VFS that maps between  
  
# canonical names and their file system representation, however,  
  
# I dont have the time for that now. Note that the container  
  
# ensures that all text files are normalized to NFC when  
  
# decoding them anyway, so there should be no mismatch between  
  
# names in the text and NFC canonical file names.
```

```
return
```

```
unicodedata
```

```
.
```

```
normalize
```

```
(
```

```
'NFC'
```

```
,
```

```
abspath_to_name
```

```
(
```

```
fullpath
```

```
,
```

```
root
```

```
or
```

```
self
```

```
.  
root  
)
```

[\[docs\]](#)

def

```
name_to_abspath  
(  
    self  
,  
    name  
):
```

' Convert a canonical name to an absolute OS dependent path '

return

```
name_to_abspath  
(  
    name  
,  
    self  
.  
    root  
)
```

[\[docs\]](#)

def

```
exists  
(  
    self  
,
```

```
name
):
```

```
""" True iff a file/folder corresponding to the canonical name exists. Note
    that this function suffers from the limitations of the underlying OS
    filesystem, in particular case (in)sensitivity. So on a case
    insensitive filesystem this will return True even if the case of name
    is different from the case of the underlying filesystem file. See also :meth:`has_name`"""
```

```
return
```

```
os
.
path
.
exists
(
self
.
name_to_abspath
(
name
))
```

```
\[docs\]
```

```
def
```

```
href_to_name
(
self
,
href
,
```



```
base
=
None
):

'''
    Convert an href (relative to base) to a name. base must be a name or
    None, in which case self.root is used.
'''

key

=

href
,

base

ans

=

self
.
href_to_name_cache
.
get
(
key
,

null
)

if
```

```
ans
```

```
is
```

```
null
```

```
:
```

```
ans
```

```
=
```

```
self
```

```
.
```

```
href_to_name_cache
```

```
[
```

```
key
```

```
]
```

```
=
```

```
href_to_name
```

```
(
```

```
href
```

```
,
```

```
self
```

```
.
```

```
root
```

```
,
```

```
base
```

```
=
```

```
base
```

```
)
```

```
return
```

```
ans
```

[\[docs\]](#)

def

name_to_href

(
self

,

name

,

base

=

None

):

'''Convert a name to a href relative to base, which must be a name or

None in which case self.root is used as the base'''

return

name_to_href

(
name
,

self

.

root

,

base

=

base

)

[\[docs\]](#)

```
def
```

```
    opf_xpath
```

```
    (
```

```
    self
```

```
    ,
```

```
    expr
```

```
    ):
```

```
    ' Convenience method to evaluate an XPath expression on the OPF file, has the opf: and dc:  
    namespace prefixes pre-defined. '
```

```
return
```

```
    self
```

```
    .
```

```
    opf
```

```
    .
```

```
    xpath
```

```
    (
```

```
    expr
```

```
    ,
```

```
    namespaces
```

```
    =
```

```
    OPF_NAMESPACES
```

```
    )
```

```
\[docs\]
```

```
def
```

```
    has_name
```

```
    (
```

```
    self
```

```
    ,
```

```
    name
```

```
    ):
```

```
""" Return True iff a file with the same canonical name as that specified exists. Unlike  
:meth:`exists` this method is always case-sensitive. """
```

```
return
```

```
name
```

```
and
```

```
name
```

```
in
```

```
self
```

```
.
```

```
name_path_map
```

```
def
```

```
has_name_and_is_not_empty
```

```
(
```

```
self
```

```
,
```

```
name
```

```
):
```

```
if
```

```
not
```

```
self
```

```
.
```

```
has_name
```

```
(
```

```
name
```

```
):
```

```
return
```

```
False
```

```
try
```

```
:
```

```
return
```

```
os
```

```
.
```

```
path
```

```
.
```

```
getsize
```

```
(
```

```
self
```

```
.
```

```
name_path_map
```

```
[
```

```
name
```

```
])
```

```
>
```

```
0
```

```
except
```

```
OSError
```

```
:
```

```
return
```

```
False
```

```
def
```

```
has_name_case_insensitive
```

```
(
```

```
self
```

```
,
```

```
name
```

```
):
```

```
if
```

```
not
```

```
name
```

```
:
```

```
return
```

```
False
```

```
name
```

```
=
```

```
name
```

```
.
```

```
lower
```

```
()
```

```
for
```

```
q
```

```
in
```

```
self
```

```
.
```

```
name_path_map
```

```
:
```

if

q

.

lower

()

==

name

:

return

True

return

False

[\[docs\]](#)

def

relpath

(

self

,

path

,

base

=

None

):


```
"""Convert an absolute path (with os separators) to a path relative to  
base (defaults to self.root). The relative path is *not* a name. Use  
:meth:`abspath_to_name` for that."""
```

```
return
```

```
relpath
```

```
(
```

```
path
```

```
,
```

```
base
```

```
or
```

```
self
```

```
.
```

```
root
```

```
)
```

```
def
```

```
ok_to_be_unmanifested
```

```
(
```

```
self
```

```
,
```

```
name
```

```
):
```

```
return
```

```
name
```

```
in
```

```
self
.  
names_that_need_not_be_manifested
```

@property

def

```
names_that_need_not_be_manifested  
(  
self  
):
```

' Set of names that are allowed to be missing from the manifest. Depends on the e-book file format. '

return

```
{  
self  
.  
opf_name  
}
```

@property

def

```
names_that_must_not_be_removed  
(  
self  
):
```

' Set of names that must never be deleted from the container. Depends on the e-book file

```
format. '

return

{
self
.
opf_name
}

@property

def

names_that_must_not_be_changed
(
self
):

' Set of names that must never be renamed. Depends on the e-book file format. '

return

set
()

def

parse
(
self
,
path
,
```

```
mime  
):
```

```
with
```

```
lopen  
(  
path  
,  
'rb'  
)
```

```
as
```

```
src  
:
```

```
data
```

```
=
```

```
src  
.  
read  
()
```

```
if
```

```
mime
```

```
in
```

```
OEB_DOCS  
:
```

```
data
```

```
=  
  
self  
.  
parse_xhtml  
(  
data  
,  
  
self  
.  
relpath  
(  
path  
)  
)  
  
elif  
  
mime  
[  
-  
4  
:]  
  
in  
  
{  
'+xml'  
,  
  
'/xml'  
}:  
  
data  
  
=  
  
self  
.  
parse_xml  
(
```

```
data
)

elif

mime

in

OEB_STYLES
:
```

```
data

=

self
.
parse_css
(
data
,
self
.
relpath
(
path
))
```

```
return
```

```
data
```

```
\[docs\]
```

```
def
```

```

raw_data
(
    self
    ,

    name
    ,

    decode
    =
    True
    ,

    normalize_to_nfc
    =
    True
):

'''
    Return the raw data corresponding to the file specified by name

    :param decode: If True and the file has a text based MIME type, decode it and return a
    unicode object instead of raw bytes.

    :param normalize_to_nfc: If True the returned unicode object is normalized to the NFC
    normal form as is required for the EPUB and AZW3 file formats.

    '''

with
self
.
open
(
    name
)

```

```
as

nf
:

ans

=

nf
.
read
()

mime

=

self
.
mime_map
.
get
(
name
,

guess_type
(
name
))

if

decode

and

(
mime
```


in

OEB_STYLES

or

mime

in

OEB_DOCS

or

mime

==

'text/plain'

or

mime

[

-

4

:]

in

{

'+xml'

,

'/xml'

});

ans

=

```
self
.  
decode  
(  
ans  
,  
  
normalize_to_nfc  
=  
normalize_to_nfc  
)
```

return

ans

[\[docs\]](#)

def

```
parsed  
(  
self  
,  
  
name  
):
```

""" Return a parsed representation of the file specified by name. For

HTML and XML files an lxml tree is returned. For CSS files a css_parser

stylesheet is returned. Note that parsed objects are cached for

performance. If you make any changes to the parsed object, you must

*call :meth:`dirty` so that the container knows to update the cache. See also
:meth:`replace`.*"""

```
ans

=

self
.
parsed_cache
.
get
(
name
,
None
)

if

ans

is

None
:

self
.
used_encoding

=

None

mime

=

self
.
mime_map
```

```
.  
get  
(  
    name  
,  
  
    guess_type  
(  
        name  
    ))  
  
    ans  
  
    =  
  
    self  
    .  
    parse  
    (  
        self  
        .  
        name_path_map  
        [  
            name  
        ],  
  
        mime  
    )  
  
    self  
    .  
    parsed_cache  
    [  
        name  
    ]  
  
    =  
  
    ans
```

```
self
.  
encoding_map  
[  
name  
]
```

```
=
```

```
self  
.  
used_encoding
```

```
return
```

```
ans
```

```
\[docs\]
```

```
def
```

```
replace
```

```
(  
self
```

```
,
```

```
name
```

```
,
```

```
obj
```

```
):
```

```
'''
```

Replace the parsed object corresponding to name with obj, which must be

a similar object, i.e. an lxml tree for HTML/XML or a css_parser

stylesheet for a CSS file.

```
'''

self
.
parsed_cache
[
name
]

=

obj

self
.
dirty
(
name
)

@property

def

opf
(
self
):

' The parsed OPF file '

return

self
.
parsed
(
```

```
self
    .
    opf_name
)

@property

def
    mi
    (
    self
    ):

    """ The metadata of this book as a Metadata object. Note that this
        object is constructed on the fly every time this property is requested,
        so use it sparingly. """

from
    calibre.ebooks.metadata.opf2

import
    OPF

    as

    O

    mi

    =

    self
```

```
.
serialize_item
(
    self
.
opf_name
)

return

O
(
    BytesIO
(
        mi
    ),

    basedir
    =
    self
.
opf_dir
,

    unquote_urls
    =
    False
,

    populate_spine
    =
    False
)
.
to_book_metadata
()

@property
```



```
def
opf_version
(
self
):

' The version set on the OPF
\'
s <package> element '
```

```
try
:
```

```
return

self
.
opf_xpath
(
'//opf:package/@version'
)[
0
]
```

```
except
IndexError
:
```

```
return

''
```

```
@property
```

def

opf_version_parsed

(
self
):

' The version set on the OPF

\'
s <package> element as a tuple of integers '

return

parse_opf_version

(
self
.
opf_version
)

@property

def

manifest_id_map

(
self
):

' Mapping of manifest id to canonical names '

return

{

```
item
.
get
(
'id'
):
self
.
href_to_name
(
item
.
get
(
'href'
),
self
.
opf_name
)

for
item
in
self
.
opf_xpath
(
'//opf:manifest/opf:item[@href and @id]'
)}

@property

def
```

```

manifest_type_map
(
    self
):

    ' Mapping of manifest media-type to list of canonical names of that media-type '

    ans

    =

    defaultdict
    (
        list
    )

    for
    item
    in
    self
    .
    opf_xpath
    (
        '//opf:manifest/opf:item[@href and @media-type]'
    ):

    ans
    [
    item
    .
    get
    (
        'media-type'
    )
    .
    lower

```

```
()]
.
append
(
self
.
href_to_name
(
item
.
get
(
'href'
),
self
.
opf_name
))

return

{
mt
:
tuple
(
v
)
for
mt
,
v
in
iteritems
```

```
(  
ans  
)}
```

[\[docs\]](#)

def

manifest_items_with_property

```
(  
self  
,
```

```
property_name  
):
```

' All manifest items that have the specified property '

prefixes

=

read_prefixes

```
(  
self  
.  
opf  
)
```

for

item

in

items_with_property

```
(
```

```
self
.  
opf  
,  
  
property_name  
,  
  
prefixes  
):
```

```
href
```

```
=
```

```
item  
.  
get  
(  
'href'  
)
```

```
if
```

```
href  
:
```

```
yield
```

```
self  
.  
href_to_name  
(  
item  
.  
get  
(  
'href'  
),
```

```
self
.  
opf_name  
)
```

[\[docs\]](#)

def

```
manifest_items_of_type  
(  
    self  
,  
    predicate  
):
```

""" The names of all manifest items whose media-type matches predicate.

*`predicate` can be a set, a list, a string or a function taking a single
 argument, which will be called with the media-type. """*

if

```
isinstance  
(  
    predicate  
,  
    str  
):
```

```
    predicate
```

```
    =
```

```
    predicate
```

```
    .
```


`__eq__`

elif

`hasattr`

`(`

`predicate`

`,`

`'__contains__'`

`):`

`predicate`

`=`

`predicate`

`.`

`__contains__`

for

`mt`

`,`

`names`

in

`iteritems`

`(`

`self`

`.`

`manifest_type_map`

`):`

if

`predicate`

```
(  
mt  
):
```

yield from

names

[\[docs\]](#)

def

apply_unique_properties

```
(  
self
```

```
,
```

name

```
,
```

```
*
```

properties

```
):
```

```
""" Ensure that the specified properties are set on only the manifest item  
    identified by name. You can pass None as the name to remove the  
    property from all items. """
```

properties

```
=
```

frozenset

```
(  
properties  
)
```

```
removed_names
```

```
,
```

```
added_names
```

```
=
```

```
[],
```

```
[]
```

```
for
```

```
p
```

```
in
```

```
properties
```

```
:
```

```
if
```

```
p
```

```
.
```

```
startswith
```

```
(
```

```
'calibre:'
```

```
):
```

```
ensure_prefix
```

```
(
```

```
self
```

```
.
```

```
opf
```

```
,
```

```
None
```

```
,
```

```
'calibre'
,
CALIBRE_PREFIX
)

break

for
item

in

self
.
opf_xpath
(
'//opf:manifest/opf:item'
):

iname

=

self
.
href_to_name
(
item
.
get
(
'href'
),

self
.
opf_name
```

```
)

props

=

(
  item
  .
  get
  (
    'properties'
  )
  or
  ""
)
.
split
()

lprops

=

{
  p
  .
  lower
  ()
  for
  p
  in
  props
}
```

for

prop

in

properties

:

if

prop

.

lower

()

in

lprops

:

if

name

!=

iname

:

removed_names

.

append

(

iname

)

props

```
=  
  
[  
  p  
  for  
  p  
  in  
  props  
  if  
  p  
  .  
  lower  
  ()  
  !=  
  prop  
  ]  
  
if  
  props  
  :  
  
  item  
  .  
  set  
  (  
    'properties'  
  ,  
  ''  
  .  
  join
```

```
(
  props
))

else
:

del

item
.
attrib
[
  'properties'
]

else
:

if

name

==

iname
:

added_names
.
append
(
  iname
)

props
.
```



```
append
(
prop
)

item
.
set
(
'properties'
,
''
.
join
(
props
))

self
.
dirty
(
self
.
opf_name
)

return

removed_names
,
added_names

\[docs\]

def
```

```
add_properties
(
    self
,
    name
,
    *
    properties
):

""" Add the specified properties to the manifest item identified by name. """

    properties

    =

    frozenset
    (
        properties
    )

    if

    not

    properties
    :

    return

    True

    for

    p
```

```
in

properties
:

if

p
.
startswith
(
'calibre:'
):

ensure_prefix
(
self
.
opf
,

None
,

'calibre'
,

CALIBRE_PREFIX
)

break

for

item

in

self
```

```
.  
opf_xpath  
(  
    '//opf:manifest/opf:item'  
):
```

```
iname
```

```
=
```

```
self
```

```
.  
href_to_name  
(  
    item
```

```
.  
get  
(  
    'href'  
),
```

```
self
```

```
.  
opf_name  
)
```

```
if
```

```
name
```

```
==
```

```
iname  
:
```

```
props
```

```
=
```

```
frozenset
```

```
((  
item  
.  
get  
(  
'properties'  
)
```

or

```
"  
)  
.  
split  
(())
```

|

properties

```
item  
.  
set  
(  
'properties'  
,  
''
```

```
..  
join  
(  
props  
)
```

return

True

return

False

@property

def

guide_type_map

(

self

):

' Mapping of guide type to canonical name '

return

{

item

.

get

(

'type'

,

"

):

self

.

href_to_name

(

item

.

get

(

'href'

),

self

.

opf_name

```

)

for
item
in
self
.
opf_xpath
(
'//opf:guide/opf:reference[@href and @type]'
)}

@property

def
spine_iter
(
self
):

    """ An iterator that yields item, name is_linear for every item in the
        books' spine. item is the lxml element, name is the canonical file name
        and is_linear is True if the item is linear. See also: :attr:`spine_names` and
        :attr:`spine_items`. """

    manifest_id_map
    =
    self
    .

```

```
manifest_id_map

non_linear

=

[]

for

item

in

self
.
opf_xpath
(
'//opf:spine/opf:itemref[@idref]'
):

idref

=

item
.
get
(
'idref'
)

name

=

manifest_id_map
.
get
```



```
(
idref
,
None
)

path

=

self
.
name_path_map
.
get
(
name
,
None
)

if
path
:

if
item
.
get
(
'linear'
,
'yes'
)
```

```
==  
  
'yes'  
:  
  
yield  
  
item  
,  
  
name  
,  
  
True  
  
else  
:  
  
non_linear  
.  
append  
((  
item  
,  
  
name  
))  
  
for  
  
item  
,  
  
name  
  
in  
  
non_linear  
:
```

yield

item

,

name

,

False

def

index_in_spine

(

self

,

name

):

manifest_id_map

=

self

.

manifest_id_map

for

i

,

item

in

```
enumerate
(
self
.
opf_xpath
(
'//opf:spine/opf:itemref[@idref]'
)):
```

idref

=

item

```
.
get
(
'idref'
)
```

q

=

manifest_id_map

```
.
get
(
idref
,
```

```
None
)
```

if

q

==

```
name
:

return

i

@property

def

spine_names
(
self
):

""" An iterator yielding name and is_linear for every item in the
books' spine. See also: :attr:`spine_iter` and :attr:`spine_items`. """

for
item
,
name
,
linear

in

self
.
spine_iter
:
```

yield

name

,

linear

@property

def

spine_items

(

self

):

""" An iterator yielding the path for every item in the

books' spine. See also: :attr:`spine\_iter` and :attr:`spine\_items`. """

for

name

,

linear

in

self

.

spine_names

:

yield

```
self
.  
name_path_map  
[  
name  
]
```

[\[docs\]](#)

```
def
```

```
remove_from_spine
```

```
(  
self
```

```
,
```

```
spine_items
```

```
,
```

```
remove_if_no_longer_in_spine
```

```
=
```

```
True
```

```
):
```

```
'''
```

```
    Remove the specified items (by canonical name) from the spine. If  
    ``remove_if_no_longer_in_spine``
```

```
    is True, the items are also deleted from the book, not just from the spine.
```

```
'''
```

```
nixed
```

```
=
```

```
set
```

```
()  
  
for  
  
(  
    name  
,  
    remove  
,  
  
    (  
        item  
,  
  
        xname  
,  
  
        linear  
    )  
  
    in  
  
    zip  
    (  
        spine_items  
,  
  
        self  
        .  
        spine_iter  
    ):  
  
    if  
  
        remove  
  
        and  
  
        name
```



```
==
```

```
xname
```

```
:
```

```
self
```

```
.
```

```
remove_from_xml
```

```
(
```

```
item
```

```
)
```

```
nixed
```

```
.
```

```
add
```

```
(
```

```
name
```

```
)
```

```
if
```

```
remove_if_no_longer_in_spine
```

```
:
```

```
# Remove from the book if no longer in spine
```

```
nixed
```

```
--
```

```
{
```

```
name
```

```
for
```

```
name
```

```
,
```

```
linear
```

```
in
```

```
self
```

```
.
```

```
spine_names
```

```
}
```

```
for
```

```
name
```

```
in
```

```
nixed
```

```
:
```

```
self
```

```
.
```

```
remove_item
```

```
(
```

```
name
```

```
)
```

```
\[docs\]
```

```
def
```

```
set_spine
```

```
(
```

```
self
```

```
,
```

```
spine_items
```

```
):
```

```
''' Set the spine to be spine_items where spine_items is an iterable of
```

the form (name, linear). Will raise an error if one of the names is not present in the manifest. '''

```
imap
```

```
=
```

```
self
```

```
.
```

```
manifest_id_map
```

```
imap
```

```
=
```

```
{
```

```
name
```

```
:
```

```
item_id
```

```
for
```

```
item_id
```

```
,
```

```
name
```

```
in
```

```
iteritems
```

```
(
```

```
imap
```

```
)}
```

```
items
```

```
=
```

```
[
```

```
item
for
item
,
name
,
linear
in
self
.
spine_iter
]

tail
,

last_tail

=

(
items
[
0
]
.
tail
,
items
[
-
1
]
.
tail
```

```
)  
  
if  
items  
else  
  
(  
,  
    \n  
,  
,  
    \n  
)  
  
for  
i  
in  
items  
:  
  
    self  
    .  
    remove_from_xml  
    (  
    i  
    )  
  
spine  
=  
self
```

```
.
opf_xpath
(
'//opf:spine'
)[
0
]

spine
.
text
=

tail

for

name
,

linear

in

spine_items
:

i

=

spine
.
makeelement
(
'{
%s
}itemref'
```

```
%  
  
OPF_NAMESPACES  
[  
  'opf'  
],  
  
nsmmap  
=  
{  
  'opf'  
:  
  OPF_NAMESPACES  
[  
  'opf'  
]})  
  
i  
.  
tail  
  
=  
  
tail  
  
i  
.  
set  
(  
  'idref'  
,  
  
imap  
[  
  name  
)  
  
spine  
.  
append
```

```
(  
i  
)  
  
if  
not  
linear  
:  
  
i  
.  
set  
(  
    'linear'  
,  
    'no'  
)  
  
if  
len  
(  
    spine  
)  
  
>  
  
0  
:  
  
spine  
[  
    -  
    1  
]  
.
```



```
tail
```

```
=
```

```
last_tail
```

```
self
```

```
.
```

```
dirty
```

```
(
```

```
self
```

```
.
```

```
opf_name
```

```
)
```

```
\[docs\]
```

```
def
```

```
remove_item
```

```
(
```

```
self
```

```
,
```

```
name
```

```
,
```

```
remove_from_guide
```

```
=
```

```
True
```

```
):
```

```
'''
```

Remove the item identified by name from this container. This removes all references to the item in the OPF manifest, guide and spine as well as from any internal caches.

```
'''

removed

=

set
()

for
elem

in
self
.
opf_xpath
(
'//opf:manifest/opf:item[@href]'
):

if
self
.
href_to_name
(
elem
.
get
(
'href'
),
self
.
opf_name
)
```

```
==

name
:

id_

=

elem
.
get
(
'id'
,
None
)

if
id_
is
not
None
:

removed
.
add
(
id_
)

self
.
```

```
remove_from_xml
```

```
(  
    elem  
)
```

```
self
```

```
.
```

```
dirty
```

```
(  
    self
```

```
.
```

```
opf_name
```

```
)
```

```
if
```

```
removed
```

```
:
```

```
for
```

```
spine
```

```
in
```

```
self
```

```
.
```

```
opf_xpath
```

```
(  
    '//opf:spine'  
)
```

```
tocref
```

```
=
```

```
spine
```

```
.
```

```
attrib
```

```
.  
get  
(  
    'toc'  
,  
    None  
)  
  
if  
  
tocref  
  
and  
  
tocref  
  
in  
  
removed  
:  
  
spine  
.  
attrib  
.  
pop  
(  
    'toc'  
,  
    None  
)  
  
self  
.  
dirty  
(  
    self  
.  

```

```
opf_name
)

for
item
in
self
.
opf_xpath
(
'//opf:spine/opf:itemref[@idref]'
):

idref
=
item
.
get
(
'idref'
)

if
idref
in
removed
:

self
.
```

```
remove_from_xml
(
item
)

self
.
dirty
(
self
.
opf_name
)

for
meta
in
self
.
opf_xpath
(
'//opf:meta[@name="cover" and @content]'
):

if
meta
.
get
(
'content'
)
in
removed
```

```
:  
  
self  
.  
remove_from_xml  
(  
    meta  
)  
  
self  
.  
dirty  
(  
    self  
.  
    opf_name  
)  
  
if  
  
    remove_from_guide  
    :  
  
    for  
  
        item  
  
    in  
  
        self  
        .  
        opf_xpath  
        (  
            '//opf:guide/opf:reference[@href]'  
        ):  
  
    if
```



```
self
.  
href_to_name  
(  
item
```

```
.  
get  
(  
'href'  
),
```

```
self  
.  
opf_name  
)
```

```
==
```

```
name  
:
```

```
self  
.  
remove_from_xml  
(  
item  
)
```

```
self  
.  
dirty  
(  
self  
.  
opf_name  
)
```

```
path
```

```
=  
  
self  
.  
name_path_map  
.  
pop  
(  
name  
,  
  
None  
)  
  
if  
  
path  
  
and  
  
os  
.  
path  
.  
exists  
(  
path  
):  
  
os  
.  
remove  
(  
path  
)  
  
self  
.  
mime_map
```

```
.  
pop  
(  
name  
,  
None  
)
```

```
self  
.  
parsed_cache  
.  
pop  
(  
name  
,  
None  
)
```

```
self  
.  
dirtied  
.  
discard  
(  
name  
)
```

[\[docs\]](#)

```
def  
  
dirty  
(  
self  
,  
name
```

```
):
```

```
""" Mark the parsed object corresponding to name as dirty. See also: :meth:`parsed`. """
```

```
self
```

```
.
```

```
dirtied
```

```
.
```

```
add
```

```
(
```

```
name
```

```
)
```

```
\[docs\]
```

```
def
```

```
remove_from_xml
```

```
(
```

```
self
```

```
,
```

```
item
```

```
):
```

```
'Removes item from parent, fixing indentation (works only with self closing items)'
```

```
parent
```

```
=
```

```
item
```

```
.
```

```
getparent
```

```
()
```

```
idx
```

```
=
```

```
parent
```

```
.
```

```
index
```

```
(
```

```
item
```

```
)
```

```
if
```

```
idx
```

```
==
```

```
0
```

```
:
```

```
# We are removing the first item - only care about adjusting
```

```
# the tail if this was the only child
```

```
if
```

```
len
```

```
(
```

```
parent
```

```
)
```

```
==
```

```
1
```

```
:
```

```
parent
```

```
.
```

```
text
```

```
=
```

```
item
```

```
.
```

```
tail
```

```
else
```

```
:
```

```
# Make sure the preceding item has this tail
```

```
parent
```

```
[
```

```
idx
```

```
-
```

```
1
```

```
]
```

```
.
```

```
tail
```

```
=
```

```
item
```

```
.
```

```
tail
```

```
parent
```

```
.
```

```
remove
```

```
(
```

```
item
```

```
)
```

```
return
```

```
item
```

[\[docs\]](#)

def

insert_into_xml

(
self

,

parent

,

item

,

index

=

None

):

"""Insert item into parent (or append if index is None), fixing

indentation. Only works with self closing items."""

if

index

is

None

:

parent

.

append

(

item

)

else

:

parent

.

insert

(

index

,

item

)

idx

=

parent

.

index

(

item

)

if

idx

==

0

:

item

.

tail

=


```
parent
```

```
.
```

```
text
```

```
# If this is the only child of this parent element, we need a
```

```
# little extra work as we have gone from a self-closing <foo />
```

```
# element to <foo><item /></foo>
```

```
if
```

```
len
```

```
(
```

```
parent
```

```
)
```

```
==
```

```
1
```

```
:
```

```
sibling
```

```
=
```

```
parent
```

```
.
```

```
getprevious
```

```
()
```

```
if
```

```
sibling
```

```
is
```

None

:

Give up!

return

parent

.

text

=

sibling

.

text

item

.

tail

=

sibling

.

tail

else

:

item

.

tail

=

parent

[

idx

-

1

]

.

tail

if

idx

==

len

(

parent

)

-

1

:

parent

[

idx

-

1

]

.

tail

=

parent

.

text

[\[docs\]](#)

```

def
    opf_get_or_create
    (
        self
        ,
        name
    ):

    """ Convenience method to either return the first XML element with the
        specified name or create it under the opf:package element and then
        return it, if it does not already exist. """

    ans
    =
    self
    .
    opf_xpath
    (
        '//opf:'
        +
        name
    )

    if

    ans
    :

    return

    ans
    [
    0

```

```
]

self
.
dirty
(
self
.
opf_name
)

package

=

self
.
opf_xpath
(
'//opf:package'
)[
0
]

item

=

package
.
makeelement
(
OPF
(
name
))

item
.
```

```
tail
```

```
=
```

```
,
```

```
\n
```

```
,
```

```
package
```

```
.
```

```
append
```

```
(
```

```
item
```

```
)
```

```
return
```

```
item
```

```
\[docs\]
```

```
def
```

```
generate_item
```

```
(
```

```
self
```

```
,
```

```
name
```

```
,
```

```
id_prefix
```

```
=
```

```
None
```

```
,
```

```
media_type
```

```
=
```

```
None
```

```
,
```

```
unique_href
=
True
):
```

```
"""Add an item to the manifest with href derived from the given
    name. Ensures uniqueness of href and id automatically. Returns
    generated item."""
```

```
id_prefix
=
```

```
id_prefix
or
```

```
'id'
```

```
media_type
=
```

```
media_type
or
```

```
guess_type
(
name
)
```

```
if
```

```
unique_href
:
```

name

=

self

.

make_name_unique

(

name

)

href

=

self

.

name_to_href

(

name

,

self

.

opf_name

)

base

,

ext

=

href

.

rpartition

(

' '

.


```
)[
0
::
2
]

all_ids

=

{
x
.
get
(
'id'
)

for
x

in

self
.
opf_xpath
(
'//*[@id]'
)}

c

=

0

item_id

=
```

id_prefix

while

item_id

in

all_ids

:

c

+=

1

item_id

=

id_prefix

+

,

%d

,

%

c

manifest

=

self

.

```
opf_xpath  
(  
    '//opf:manifest'  
)[  
    0  
]
```

item

=

```
manifest  
.  
makeelement  
(  
    OPF  
(  
        'item'  
    ),
```

```
id  
=  
item_id  
,
```

```
href  
=  
href  
)
```

```
item  
.  
set  
(  
    'media-type'  
,  
    media_type  
)
```

```
self
.  
insert_into_xml  
(  
manifest  
,  
  
item  
)
```

```
self  
.  
dirty  
(  
self  
.  
opf_name  
)
```

```
name
```

```
=
```

```
self  
.  
href_to_name  
(  
href  
,  

```

```
self  
.  
opf_name  
)
```

```
self  
.  
name_path_map  
[
```

```
name
]

=

path

=

self
.
name_to_abspath
(
name
)

self
.
mime_map
[
name
]

=

media_type


# Ensure that the file corresponding to the newly created item exists


# otherwise cloned containers will fail when they try to get the number


# of links to the file


base

=

os
```

```
.  
path  
.  
dirname  
(  
path  
)
```

```
if
```

```
not
```

```
os  
.  
path  
.  
exists  
(  
base  
):
```

```
os  
.  
mkdirs  
(  
base  
)
```

```
lopen  
(  
path  
,  
'wb'  
)  
.  
close  
(
```

```
return

item

def

format_opf
(
self
):

try
:

mdata

=

self
.
opf_xpath
(
'//opf:metadata'
)[
0
]

except

IndexError
:

pass

else
:
```

```
mdata
```

```
.
```

```
text
```

```
=
```

```
,
```

```
\n
```

```
,
```

```
remove
```

```
=
```

```
set
```

```
()
```

```
for
```

```
child
```

```
in
```

```
mdata
```

```
:
```

```
child
```

```
.
```

```
tail
```

```
=
```

```
,
```

```
\n
```

```
,
```

```
try
```

```
:
```



```
if
(
  child
  .
  get
  (
    'name'
    ,
    ""
  )
  .
  startswith
  (
    'calibre:'
  )
)
```

and

```
child
.
get
(
  'content'
  ,
  ""
)
.
strip
()
```

in

```
{
'
}
```

```
,  
"  
}):  
  
remove  
.  
add  
(  
child  
)  
  
except  
  
AttributeError  
:  
  
continue  
  
# Happens for XML comments  
  
for  
  
child  
  
in  
  
remove  
:  
  
mdata  
.  
remove  
(  
child  
)
```

```
if
len
(
mdata
)

>

0
:

mdata
[
-
1
]
.
tail
=
'
\n
'

# Ensure name comes before content, needed for Nooks

for
meta
in
self
.
opf_xpath
(
'//opf:meta[@name="cover"]'
):
```

```
if
```

```
'content'
```

```
in
```

```
meta
```

```
.
```

```
attrib
```

```
:
```

```
meta
```

```
.
```

```
set
```

```
(
```

```
'content'
```

```
,
```

```
meta
```

```
.
```

```
attrib
```

```
.
```

```
pop
```

```
(
```

```
'content'
```

```
))
```

```
\[docs\]
```

```
def
```

```
serialize_item
```

```
(
```

```
self
```

```
,
```

```
name
```

```
):
```

```
""" Convert a parsed object (identified by canonical name) into a bytestring. See :meth:`parsed`.  
"""
```

```
data
```

```
=
```

```
root
```

```
=
```

```
self
```

```
.
```

```
parsed
```

```
(
```

```
name
```

```
)
```

```
if
```

```
name
```

```
==
```

```
self
```

```
.
```

```
opf_name
```

```
:
```

```
self
```

```
.
```

```
format_opf
```

```
()
```

```
data
```

```
=  
  
serialize  
(  
    data  
,  
  
    self  
    .  
    mime_map  
    [  
        name  
    ],  
  
    pretty_print  
    =  
    name  
  
    in  
  
    self  
    .  
    pretty_print  
    )  
  
    if  
  
    name  
  
    ==  
  
    self  
    .  
    opf_name  
  
    and  
  
    root  
    .  
    nsmap  
    .
```

```

get
(
None
)

==

OPF2_NS
:

# Needed as I can't get lxml to output opf:role and

# not output <opf:metadata> as well

data

=

re
.
sub
(
br
'(<[/]{0,1})opf:'
,

r
'\1'
,

data
)

return

data

```

[\[docs\]](#)

def

commit_item

(

self

,

name

,

keep_parsed

=

False

):

*""" Commit a parsed object to disk (it is serialized and written to the
underlying file). If ``keep\_parsed`` is True the parsed representation
is retained in the cache. See also: :meth:`parsed` """*

if

name

not

in

self

.

parsed_cache

:

return

data


```
=  
  
self  
.  
serialize_item  
(  
    name  
)  
  
self  
.  
dirtied  
.  
discard  
(  
    name  
)  
  
if  
  
not  
  
keep_parsed  
:  
  
self  
.  
parsed_cache  
.  
pop  
(  
    name  
)  
  
dest  
  
=
```

```
self
.
name_path_map
[
name
]

if

self
.
cloned

and

nlinks_file
(
dest
)

>

1
:

# Decouple this file from its links

os
.
unlink
(
dest
)

with

lopen
(
dest
```

```
,
```

```
'wb'
```

```
)
```

```
as
```

```
f
```

```
:
```

```
f
```

```
.
```

```
write
```

```
(
```

```
data
```

```
)
```

```
\[docs\]
```

```
def
```

```
filesize
```

```
(
```

```
self
```

```
,
```

```
name
```

```
):
```

```
""" Return the size in bytes of the file represented by the specified
```

```
canonical name. Automatically handles dirtied parsed objects. See also:
```

```
:meth:`parsed` """
```

```
if
```

```
name
```

```
in

self
.
dirtied
:

self
.
commit_item
(
name
,

keep_parsed
=
True
)

path

=

self
.
name_to_abspath
(
name
)

return

os
.
path
.
getsize
(
path
```

```
)
```

[\[docs\]](#)

```
def
```

```
    get_file_path_for_processing
```

```
    (
```

```
    self
```

```
    ,
```

```
    name
```

```
    ,
```

```
    allow_modification
```

```
    =
```

```
    True
```

```
):
```

```
""" Similar to open() except that it returns a file path, instead of an open file object. """
```

```
if
```

```
    name
```

```
in
```

```
    self
```

```
    .
```

```
    dirtied
```

```
    :
```

```
    self
```

```
    .
```

```
    commit_item
```

```
    (
```

```
    name
```

```
)
```

```
self
.
parsed_cache
.
pop
(
name
,
False
)

path

=

self
.
name_to_abspath
(
name
)

base

=

os
.
path
.
dirname
(
path
)

if

not
```

```
os
·
path
·
exists
(
base
):
```

```
os
·
mkdirs
(
base
)
```

```
else
:
```

```
if
self
·
cloned
```

```
and
allow_modification
and
```

```
os
·
path
·
exists
(
path
)
```

and

nlinks_file

(

path

)

>

1

:

Decouple this file from its links

temp

=

path

+

'xxx'

shutil

.

copyfile

(

path

,

temp

)

if

iswindows

:


```
retry_on_fail
```

```
(
```

```
os
```

```
.
```

```
unlink
```

```
,
```

```
path
```

```
)
```

```
else
```

```
:
```

```
os
```

```
.
```

```
unlink
```

```
(
```

```
path
```

```
)
```

```
os
```

```
.
```

```
rename
```

```
(
```

```
temp
```

```
,
```

```
path
```

```
)
```

```
return
```

```
path
```

```
\[docs\]
```

```
def
open
(
self
,
name
,
mode
=
'rb'
):

""" Open the file pointed to by name for direct read/write. Note that

    this will commit the file if it is dirtied and remove it from the parse

    cache. You must finish with this file before accessing the parsed

    version of it again, or bad things will happen. """

return

lopen
(
self
.
get_file_path_for_processing
(
name
,
mode

not

in

{
```

```
'r'
',
'rb'
}},
mode
)
```

[\[docs\]](#)

```
def
```

```
commit
(
self
,
```

```
outpath
=
None
,
```

```
keep_parsed
=
False
):
```

```
'''
```

Commit all dirtied parsed objects to the filesystem and write out the e-book file at outpath.

:param output: The path to write the saved e-book file to. If None, the path of the original book file is used.

:param keep_parsed: If True the parsed representations of committed items are kept in the cache.

```
'''
```

for

name

in

tuple

(

self

.

dirtied

):

self

.

commit_item

(

name

,

keep_parsed

=

keep_parsed

)

def

compare_to

(

self

,

other

):

if

set

(

```
self
.
name_path_map
)

!=

set
(
other
.
name_path_map
):

return

'Set of files is not the same'

mismatches

=

[]

for

name
,

path

in

iteritems
(
self
.
name_path_map
):
```

```
opath
```

```
=
```

```
other
```

```
.
```

```
name_path_map
```

```
[
```

```
name
```

```
]
```

```
with
```

```
lopen
```

```
(
```

```
path
```

```
,
```

```
'rb'
```

```
)
```

```
as
```

```
f1
```

```
,
```

```
lopen
```

```
(
```

```
opath
```

```
,
```

```
'rb'
```

```
)
```

```
as
```

```
f2
```

```
:
```

```
if
```

```
f1
.
read
()

!=

f2
.
read
():

mismatches
.
append
(
'The file
%s
is not the same'
%
name
)

return

'
\n
'
.
join
(
mismatches
)

# }}}

# EPUB {{{
```

```
class
```

```
InvalidEpub
```

```
(  
InvalidBook  
):
```

```
pass
```

```
class
```

```
ObfuscationKeyMissing
```

```
(  
InvalidEpub  
):
```

```
pass
```

```
OCF_NS
```

```
=
```

```
'urn:oasis:names:tc:opendocument:xmlns:container'
```

```
VCS_IGNORE_FILES
```

```
=
```

```
frozenset
```

```
(  
'gitignore .hgignore .agignore .bzrignore'  
.  
split  
())
```



```
VCS_DIRS
```

```
=
```

```
frozenset
```

```
((
```

```
'git'
```

```
,
```

```
'hg'
```

```
,
```

```
'svn'
```

```
,
```

```
'bzt'
```

```
))
```

```
def
```

```
walk_dir
```

```
(
```

```
basedir
```

```
):
```

```
for
```

```
dirpath
```

```
,
```

```
dirname
```

```
,
```

```
filenames
```

```
in
```

```
os
```

```
.
```

```
walk
```

```
(
basedir
):

for
vcsdir

in
VCS_DIRS
:

try
:

dirname
.
remove
(
vcsdir
)

except
Exception
:

pass

is_root

=

os
.
path
```

```
.
abspath
(
os
.
path
.
normcase
(
dirpath
))
```

```
==
```

```
os
.
path
.
abspath
(
os
.
path
.
normcase
(
basedir
))
```

yield

```
is_root
,
```

```
dirpath
,
```

None

for

fname

in

filenames

:

if

fname

not

in

VCS_IGNORE_FILES

:

yield

is_root

,

dirpath

,

fname

class

EpubContainer

(

Container

):

book_type

```
=  
  
'epub'  
  
  
@property  
  
def  
  
book_type_for_display  
(  
    self  
):  
  
    ans  
  
    =  
  
    self  
    .  
    book_type  
    .  
    upper  
    ()  
  
    try  
    :  
  
        v  
  
        =  
  
        self  
        .  
        opf_version_parsed  
  
    except
```

Exception

:

pass

else

:

try

:

if

v

.

major

==

2

:

ans

+=

'2'

else

:

if

not

```
v
.  
minor  
:
```

```
ans
```

```
+=
```

```
f  
,  
{  
v  
.  
major  
}  
,
```

```
else  
:
```

```
ans
```

```
+=
```

```
f  
,  
{  
v  
.  
major  
}  
.  
{  
v  
.  
minor  
}  
,
```

```
except
```

```
Exception
```

```
:
```

```
pass
```

```
return
```

```
ans
```

```
META_INF
```

```
=
```

```
{
```

```
'container.xml'
```

```
:
```

```
True
```

```
,
```

```
'manifest.xml'
```

```
:
```

```
False
```

```
,
```

```
'encryption.xml'
```

```
:
```

```
False
```

```
,
```



```
'metadata.xml'
```

```
:
```

```
False
```

```
,
```

```
'signatures.xml'
```

```
:
```

```
False
```

```
,
```

```
'rights.xml'
```

```
:
```

```
False
```

```
,
```

```
}
```

```
def
```

```
    __init__
```

```
(
```

```
    self
```

```
,
```

```
    pathtoepub
```

```
,
```

```
    log
```

```
,
```

```
    clone_data
```

```
=
```

```
None
```

```
,
```

```
tdir
=
None
):

if

clone_data

is

not

None
:

super
()
.
__init__
(
None
,

None
,

log
,

clone_data
=
clone_data
)

for

x
```

```
in
(
'pathtoepub'
,
'obfuscated_fonts'
,
'is_dir'
):
```

```
setattr
(
self
,
x
,
clone_data
[
x
])
```

```
return
```

```
self
.
pathtoepub
=
pathtoepub
```

```
if
```

```
tdir
```

is

None

:

tdir

=

PersistentTemporaryDirectory

(

'_epub_container'

)

tdir

=

os

.

path

.

abspath

(

os

.

path

.

realpath

(

tdir

))

self

.

root

=

```
tdir

self
.
is_dir
=
os
.
path
.
isdir
(
    pathtoepub
)

if
self
.
is_dir
:

for
    is_root
    ,
    dirpath
    ,
    fname
in
    walk_dir
    (
        self
        .
```

```
pathtoepub
):
```

```
if
```

```
is_root
:
```

```
base
```

```
=
```

```
tdir
```

```
else
```

```
:
```

```
base
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
join
```

```
(
```

```
tdir
```

```
,
```

```
os
```

```
.
```

```
path
```

```
.
```

```
relpath
```

```
(
```

```
dirpath
```

```
,
```

```
self
.  
pathtoepub  
)
```

```
if
```

```
fname
```

```
is
```

```
None
```

```
:
```

```
os
```

```
.  
mkdir  
(  
base  
)
```

```
if
```

```
fname
```

```
is
```

```
not
```

```
None
```

```
:
```

```
shutil
```

```
.  
copy  
(  
os  
.  
path
```

```
.
join
(
dirpath
,
fname
),
os
.
path
.
join
(
base
,
fname
))

else
:

with

lopen
(
self
.
pathtoepub
,
'rb'
)

as

stream
:
```



```
try
:

zf

=

ZipFile
(
stream
)

zf
.
extractall
(
tdir
)

except
:

log
.
exception
(
'EPUB appears to be invalid ZIP file, trying a'

'more forgiving ZIP parser'
)

from

calibre.utils.localunzip

import
```

```
extractall
```

```
stream
```

```
.
```

```
seek
```

```
(
```

```
0
```

```
)
```

```
extractall
```

```
(
```

```
stream
```

```
,
```

```
path
```

```
=
```

```
tdir
```

```
)
```

```
try
```

```
:
```

```
os
```

```
.
```

```
remove
```

```
(
```

```
join
```

```
(
```

```
tdir
```

```
,
```

```
'mimetype'
```

```
))
```

```
except
```

```
OSError
```

```
:
```

```
pass
```

```
# Ensure all filenames are in NFC normalized form
```

```
# has no effect on HFS+ filesystems as they always store filenames
```

```
# in NFD form
```

```
for
```

```
filename
```

```
in
```

```
walk
```

```
(
```

```
self
```

```
.
```

```
root
```

```
):
```

```
n
```

```
=
```

```
unicodedata
```

```
.
```

```
normalize
```

```
(
```

```
'NFC'
```

```
,
```

```
filename
```

```
)
```

```
if
```

```
n
```

```
!=
```

```
filename
```

```
:
```

```
s
```

```
=
```

```
filename
```

```
+
```

```
'suff1x'
```

```
os
```

```
.
```

```
rename
```

```
(
```

```
filename
```

```
,
```

```
s
```

```
)
```

```
os
```

```
.
```

```
rename
```

```
(
```

```
s
```

```
,
```

```
n
```

```
)
```

```
container_path
```

```
=
```

```
join
```

```
(
```

```
self
```

```
.
```

```
root
```

```
,
```

```
'META-INF'
```

```
,
```

```
'container.xml'
```

```
)
```

```
if
```

```
not
```

```
exists
```

```
(
```

```
container_path
```

```
):
```

```
raise
```

```
InvalidEpub
```

```
(
```

```
'No META-INF/container.xml in epub'
```

```
)
```

```
with
```

```
open
```

```
(
```

```
container_path
```

```
,
'rb'
)

as

cf
:

container

=

safe_xml_fromstring
(
cf
.
read
())

opf_files

=

container
.
xpath
((

r
'child::ocf:rootfiles/ocf:rootfile'

'[@media-type="
%s
" and @full-path]'
%
guess_type
(
```

```
'a.opf'
)

),

namespaces
=
{
'ocf'
:
OCF_NS
}

)

if

not

opf_files
:

raise

InvalidEpub
(
'META-INF/container.xml contains no link to OPF file'
)

opf_path

=

os
.
path
.
join
```

```
(
self
.
root
,

*
(
urlunquote
(
opf_files
[
0
]
.
get
(
'full-path'
))
.
split
(
'/'
)))

if
not
exists
(
opf_path
):

raise
InvalidEpub
(
'OPF file does not exist at location pointed to'
```



```
' by META-INF/container.xml'  
)
```

```
super  
(
```

```
..  
__init__
```

```
(  
tdir  
,
```

```
opf_path  
,
```

```
log  
)
```

```
self
```

```
..  
obfuscated_fonts
```

```
=
```

```
{}
```

```
if
```

```
'META-INF/encryption.xml'
```

```
in
```

```
self
```

```
..  
name_path_map  
:
```

```
self
```

```
.
process_encryption
()

self
.
parsed_cache
[
'META-INF/container.xml'
]

=

container

def

clone_data
(
self
,

dest_dir
):

ans

=

super
()
.
clone_data
(
dest_dir
)

ans
```

```
[
'pathtoepub'
]

=

self
.
pathtoepub

ans
[
'obfuscated_fonts'
]

=

self
.
obfuscated_fonts
.
copy
()

ans
[
'is_dir'
]

=

self
.
is_dir

return

ans
```

```
def
rename
(
self
,
old_name
,
new_name
):

is_opf
=
old_name
==
self
.
opf_name

super
()
.
rename
(
old_name
,
new_name
)

if
is_opf
```

```

:

for
elem
in
self
.
parsed
(
'META-INF/container.xml'
)
.
xpath
((

r
'child::ocf:rootfiles/ocf:rootfile'

['@media-type="
%s
" and @full-path]'
%
guess_type
(
'a.opf'
)

),

namespaces
=
{
'ocf'
:
OCF_NS
}

```

```
);
```

```
# The asinine epubcheck cannot handle quoted filenames in
```

```
# container.xml
```

```
elem
```

```
.
```

```
set
```

```
(
```

```
'full-path'
```

```
,
```

```
self
```

```
.
```

```
opf_name
```

```
)
```

```
self
```

```
.
```

```
dirty
```

```
(
```

```
'META-INF/container.xml'
```

```
)
```

```
if
```

```
old_name
```

```
in
```

```
self
```

```
.
```

```
obfuscated_fonts
```

```
:
```

```
self
.
obfuscated_fonts
[
new_name
]

=

self
.
obfuscated_fonts
.
pop
(
old_name
)

enc

=

self
.
parsed
(
'META-INF/encryption.xml'
)

for

cr

in

enc
.
xpath
(
'//*[local-name()="CipherReference" and @URI]'
```

```
):  
  
if  
  
self  
.  
href_to_name  
(  
cr  
.  
get  
(  
'URI'  
)  
  
==  
  
old_name  
:  
  
cr  
.  
set  
(  
'URI'  
,  
self  
.  
name_to_href  
(  
new_name  
)  
  
self  
.  
dirty  
(  
'META-INF/encryption.xml'  
)
```


@property

def

names_that_need_not_be_manifested
(
self
):

return

super
(
.
names_that_need_not_be_manifested

|

{
'META-INF/'

+

x

for

x

in

self
.
META_INF
}

```
def

ok_to_be_unmanifested
(
self
,

name
):

return

name

in

self
.
names_that_need_not_be_manifested

or

name
.
startswith
(
'META-INF/'
)

@property

def

names_that_must_not_be_removed
(
self
):
```

```
return

super
()

.
names_that_must_not_be_removed

|

{
'META-INF/container.xml'
}
```

@property

```
def

names_that_must_not_be_changed
(
self
):
```

```
return

super
()

.
names_that_must_not_be_changed

|

{
'META-INF/'

+

x

for
```

```
x
```

```
in
```

```
self
```

```
.
```

```
META_INF
```

```
}
```

```
def
```

```
remove_item
```

```
(
```

```
self
```

```
,
```

```
name
```

```
,
```

```
remove_from_guide
```

```
=
```

```
True
```

```
):
```

```
# Handle removal of obfuscated fonts
```

```
if
```

```
name
```

```
==
```

```
'META-INF/encryption.xml'
```

```
:
```

```
self
```

```
.
```

```
obfuscated_fonts
.
clear
()

if

name

in

self
.
obfuscated_fonts
:

self
.
obfuscated_fonts
.
pop
(
name
,
None
)

enc

=

self
.
parsed
(
'META-INF/encryption.xml'
)
```

```
for
em
in
enc
.
xpath
(
'//*[local-name()="EncryptionMethod" and @Algorithm]'
):

alg
=
em
.
get
(
'Algorithm'
)

if
alg
not
in
{
ADOBE_OBFUSCATION
,
IDPF_OBFUSCATION
}:

continue
```

```
try
:

cr

=

em
.
getparent
()
.
xpath
(
'descendant::*[local-name()="CipherReference" and @URI]'
)[
0
]
```

```
except
(
IndexError
,
ValueError
,
KeyError
):
```

```
continue
```

```
if
name
```

```
==

self
.
href_to_name
(
cr
.
get
(
'URI'
)):

self
.
remove_from_xml
(
em
.
getparent
())

self
.
dirty
(
'META-INF/encryption.xml'
)

super
()
.
remove_item
(
name
,

remove_from_guide
=
remove_from_guide
```



```
)

def
process_encryption
(
self
):

fonts

=

{}

enc

=

self
.
parsed
(
'META-INF/encryption.xml'
)

for
em

in
enc
.
xpath
(
'//*[local-name()="EncryptionMethod" and @Algorithm]'
):
```

```
alg
=
em
.
get
(
'Algorithm'
)

if
alg
not
in
{
ADOBE_OBFUSCATION
,
IDPF_OBFUSCATION
}:

raise
DRMError
()

try
:

cr

=
```

```
em
.
getparent
()
.
xpath
(
'descendant::*[local-name()="CipherReference" and @URI]'
)[
0
]
```

except

```
(
IndexError
,
ValueError
,
KeyError
):
```

continue

```
name
=
self
.
href_to_name
(
cr
.
get
(
'URI'
```

```
))

path

=

self
.
name_path_map
.
get
(
name
,
None
)

if

path

is

not

None
:

    fonts
    [
name
    ]

    =

    alg

    if
```

not

fonts

:

return

package_id

=

raw_unique_identifier

=

idpf_key

=

None

for

attrib

,

val

in

iteritems

(

self

.

opf

.

attrib

);

if

attrib

.

endswith

(

'unique-identifier'

):

package_id

=

val

break

if

package_id

is

not

None

:

for

elem

in

self

.

opf_xpath

```
(  
'//*[@id=  
%s  
'  
%  
escape_xpath_attr  
(  
package_id  
)):
```

if

elem

.
text
:

raw_unique_identifier

=

elem

.
text

break

if

raw_unique_identifier

is

not

None

:

```
idpf_key

=

raw_unique_identifer

idpf_key

=

re
.
sub
(
['[
\u0020\u0009\u000d\u000a
]']
,
"
,
idpf_key
)

idpf_key

=

hashlib
.
sha1
(
idpf_key
.
encode
(
'utf-8'
))
.
digest
```



```
()

key

=

None

for

item

in

self
.
opf_xpath
(
'//*[local-name()="metadata"]/*'

'[local-name()="identifier"]'
):

scheme

=

None

for

xkey

in

item
.
attrib
```

```
.  
keys  
():  
  
if  
  
xkey  
.  
endswith  
(  
  'scheme'  
):
```

```
scheme  
  
=
```

```
item  
.  
get  
(  
  xkey  
)
```

```
if  
  
(  
  scheme
```

```
and
```

```
scheme  
.  
lower  
())
```

```
==
```

```
'uuid'  
)
```

```
or
\

(
item
.
text

and

item
.
text
.
startswith
(
'urn:uuid:'
)):

try
:

key

=

item
.
text
.
rpartition
(
':')
[
-
1
]

key
```

=

uuid

.

UUID

(

key

)

.

bytes

except

Exception

:

self

.

log

.

exception

(

'Failed to parse obfuscation key'

)

key

=

None

for

font

,

alg

in

iteritems

(

fonts

):

tkey

=

key

if

alg

==

ADOBE_OBFUSCATION

else

idpf_key

if

not

tkey

:

raise

ObfuscationKeyMissing

(

'Failed to find obfuscation key'

)

```
raw
```

```
=
```

```
self
```

```
.
```

```
raw_data
```

```
(
```

```
font
```

```
,
```

```
decode
```

```
=
```

```
False
```

```
)
```

```
raw
```

```
=
```

```
decrypt_font_data
```

```
(
```

```
tkey
```

```
,
```

```
raw
```

```
,
```

```
alg
```

```
)
```

```
with
```

```
self
```

```
.
```

```
open
```

```
(
```

```
font
```

```
,
```

```
'wb'  
)  
  
as  
  
f  
:  
  
f  
.  
write  
(  
raw  
)  
  
self  
.  
obfuscated_fonts  
[  
font  
]  
  
=  
  
(  
alg  
,  
  
tkey  
)  
  
def  
  
update_modified_timestamp  
(  
self  
):
```

from

calibre.ebooks.metadata.opf3

import

set_last_modified_in_opf

set_last_modified_in_opf

(
self

.
opf
)

self

.
dirty
(
self
.
opf_name
)

def

commit

(
self
,

outpath
=

None

,

keep_parsed
=

False


```
):  
  
if  
  
self  
.  
opf_version_parsed  
.  
major  
  
==  
  
3  
:  
  
self  
.  
update_modified_timestamp  
(  
  
super  
(  
.  
commit  
(  
keep_parsed  
=  
keep_parsed  
)  
  
container_path  
  
=  
  
join  
(  
self  
.  
root
```

```
,  
  
'META-INF'  
,  
  
'container.xml'  
)  
  
if  
  
not  
  
exists  
(  
    container_path  
):  
  
    raise  
  
InvalidEpub  
(  
    'No META-INF/container.xml in EPUB, this typically happens if the temporary files calibre'  
  
    'is using are deleted by some other program while calibre is running'  
)  
  
restore_fonts  
  
=  
  
{}  
  
for  
  
name  
  
in
```

```
self
.
obfuscated_fonts
:

if

name

not

in

self
.
name_path_map
:

continue

alg
,

key

=

self
.
obfuscated_fonts
[
name
]

# Decrypting and encrypting are the same operation (XOR with key)

restore_fonts
[
```

```
name
```

```
]
```

```
=
```

```
data
```

```
=
```

```
self
```

```
.
```

```
raw_data
```

```
(
```

```
name
```

```
,
```

```
decode
```

```
=
```

```
False
```

```
)
```

```
with
```

```
self
```

```
.
```

```
open
```

```
(
```

```
name
```

```
,
```

```
'wb'
```

```
)
```

```
as
```

```
f
```

```
:
```

```
f
```

```
.
```

```
write
```

```
(  
decrypt_font_data  
(  
key  
,  
  
data  
,  
  
alg  
))
```

```
if
```

```
outpath
```

```
is
```

```
None
```

```
:
```

```
outpath
```

```
=
```

```
self
```

```
.
```

```
pathtoepub
```

```
if
```

```
self
```

```
.
```

```
is_dir
```

```
:
```

```
# First remove items from the source dir that do not exist any more
```

```
for
    is_root
    ,
    dirpath
    ,
    fname

in
    walk_dir
    (
    self
    .
    pathtoepub
    ):

    if
        fname

    is

    not

    None
    :

    if
        is_root

    and

        fname

    ==

    'mimetype'
```

```
:
    continue

    base
    =
    self
    .
    root
    if
    is_root
    else
    os
    .
    path
    .
    join
    (
    self
    .
    root
    ,
    os
    .
    path
    .
    relpath
    (
    dirpath
    ,
    self
    .
    pathtoepub
```

```
))
```

```
fpath
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
join
```

```
(
```

```
base
```

```
,
```

```
fname
```

```
)
```

```
if
```

```
not
```

```
os
```

```
.
```

```
path
```

```
.
```

```
exists
```

```
(
```

```
fpath
```

```
):
```

```
os
```

```
.
```

```
remove
```

```
(
```

```
os
```

```
.
```

```
path
```

```
.
```

```
join
```



```
(  
    dirpath  
    ,  
    fname  
))  
  
try  
:  
  
    os  
    .  
    rmdir  
    (  
    dirpath  
    )  
  
except  
    OSError  
  
as  
  
    err  
    :  
  
    if  
  
        err  
        .  
        errno  
  
        !=  
  
        errno  
        .  
        ENOTEMPTY  
    :
```

```
raise
```

```
# Now copy over everything from root to source dir
```

```
for
```

```
    dirpath
```

```
,
```

```
    dirnames
```

```
,
```

```
    filenames
```

```
in
```

```
    os
```

```
.
```

```
    walk
```

```
(
```

```
        self
```

```
.
```

```
        root
```

```
    );
```

```
is_root
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
abspath
```

```
(
```

```
os
```

```
.
```

```
path
```

```
.
```

```
normcase
(
  dirpath
))

==

os
.
path
.
abspath
(
  os
.
  path
.
  normcase
  (
    self
.
    root
  ))

base

=

self
.
pathtoepub

if

is_root

else

os
.
path
.
```

```
join
(
self
.
pathtoepub
,

os
.
path
.
relpath
(
dirpath
,

self
.
root
))
```

```
try
:
```

```
os
.
mkdir
(
base
)
```

```
except
```

```
OSError
```

```
as
```

```
err
:
```

if

err

.

errno

!=

errno

.

EEXIST

:

raise

for

fname

in

filenames

:

with

lopen

(

os

.

path

.

join

(

dirpath

,

fname

),

```
'rb'  
)  
  
as  
  
src  
,  
  
lopen  
(  
os  
.  
path  
.  
join  
(  
base  
,  
  
fname  
),  
  
'wb'  
)  
  
as  
  
dest  
:  
  
shutil  
.  
copyfileobj  
(  
src  
,  
  
dest  
)
```

```
else
:

from

calibre.ebooks.tweak

import

zip_rebuilder

with

lopen
(
join
(
self
.
root
,
'mimetype'
),
'wb'
)

as

f
:

et

=

guess_type
(
```

```
'a.epub'  
)  
  
if  
  
not  
  
isinstance  
(  
et  
,  
  
bytes  
):  
  
et  
  
=  
  
et  
.  
encode  
(  
'ascii'  
)  
  
f  
.  
write  
(  
et  
)  
  
zip_rebuilder  
(  
self  
.  
root  
,
```



```
    outpath
)

for

name
,

data

in

iteritems
(
    restore_fonts
):

with

self
.
open
(
    name
,
    'wb'
)

as

f
:

f
.
write
(
    data
```

```
)
```

```
@property
```

```
def
```

```
path_to_ebook
```

```
(
```

```
self
```

```
):
```

```
return
```

```
self
```

```
.
```

```
pathtoepub
```

```
@path_to_ebook
```

```
.
```

```
setter
```

```
def
```

```
path_to_ebook
```

```
(
```

```
self
```

```
,
```

```
val
```

```
):
```

```
self
```

```
.
```

```
pathtoepub
```

```
=  
  
val  
  
# }}}  
  
# AZW3 {{{  
  
class  
  
InvalidMobi  
(  
InvalidBook  
):  
  
pass  
  
def  
  
do_explode  
(  
path  
,  
  
dest  
):  
  
from  
  
calibre.ebooks.mobi.reader.mobi6  
  
import  
  
MobiReader
```

```
from
calibre.ebooks.mobi.reader.mobi8

import

Mobi8Reader

with

    lopen
    (
    path
    ,
    'rb'
    )

    as

    stream
    :

    mr

    =

    MobiReader
    (
    stream
    ,

    default_log
    ,

    None

    ,

    None
    )
```

with

```
CurrentDir  
(  
dest  
):
```

```
mr
```

```
=
```

```
Mobi8Reader  
(  
mr  
,  
  
default_log  
,  
  
for_tweak  
=  
True  
)
```

```
opf
```

```
=
```

```
os  
.  
path  
.  
abspath  
(  
mr  
(  
))
```

```
obfuscated_fonts
```

```
=
```

```
mr
```

```
.
```

```
encrypted_fonts
```

```
return
```

```
opf
```

```
,
```

```
obfuscated_fonts
```

```
def
```

```
opf_to_azw3
```

```
(
```

```
opf
```

```
,
```

```
outpath
```

```
,
```

```
container
```

```
):
```

```
from
```

```
calibre.ebooks.conversion.plumber
```

```
import
```

```
Plumber
```

```
,
```

```
create_oebbook
```

```
class
```

```
Item
```

```
(  
Manifest
```

```
.
```

```
Item
```

```
):
```

```
def
```

```
_parse_css
```

```
(  
self
```

```
,
```

```
data
```

```
):
```

```
# The default CSS parser used by oeb.base inserts the h namespace
```

```
# and resolves all @import rules. We dont want that.
```

```
return
```

```
container
```

```
.
```

```
parse_css
```

```
(  
data  
)
```

```
def
```

```
specialize
(
  oeb
):

oeb
.
manifest
.
Item
=
Item

plumber
=
Plumber
(
  opf
,
  outpath
,
  container
.
  log
)

plumber
.
setup_options
()
```



```
inp
=
plugin_for_input_format
(
'azw3'
)
```

```
outp
=
plugin_for_output_format
(
'azw3'
)
```

```
plumber
.
opts
.
mobi_passthrough
```

```
=
True
```

```
oeb
=
create_oebbook
(
container
.
log
,
```

```
opf
```

```
,  
  
plumber  
.  
opts  
,  
  
specialize  
=  
specialize  
)  
  
set_cover  
(  
oeb  
)  
  
outp  
.  
convert  
(  
oeb  
,  
  
outpath  
,  
  
inp  
,  
  
plumber  
.  
opts  
,  
  
container  
.  
log  
)
```

```
def
epub_to_azw3
(
    epub
    ,
    outpath
    =
    None
):

    container

    =

    get_container
    (
        epub
        ,

        tweak_mode
        =
        True
    )

    outpath

    =

    outpath

    or

    (
        epub
        .
        rpartition
        (
            '!'
            .
```

```
)[
0
]

+

'.azw3'
)

opf_to_azw3
(
container
.
name_to_abspath
(
container
.
opf_name
),
outpath
,
container
)

class

AZW3Container
(
Container
):

book_type

=

'.azw3'
```

SUPPORTS_TITLEPAGES

=

False

SUPPORTS_FILENAMES

=

False

def

__init__

(

self

,

pathtoazw3

,

log

,

clone_data

=

None

,

tdir

=

None

):

if

```
clone_data
```

```
is
```

```
not
```

```
None
```

```
:
```

```
super
```

```
()
```

```
.
```

```
__init__
```

```
(
```

```
None
```

```
,
```

```
None
```

```
,
```

```
log
```

```
,
```

```
clone_data
```

```
=
```

```
clone_data
```

```
)
```

```
for
```

```
x
```

```
in
```

```
(
```

```
'pathtoazw3'
```

```
,
```

```
'obfuscated_fonts'
```

```
):
```

```
setattr
(
self
,

x
,

clone_data
[
x
])
```

```
return
```

```
self
.
pathtoazw3
=
pathtoazw3
```

```
if
```

```
tdir
```

```
is
```

```
None
```

```
:
```

```
tdir
```

```
=
```

```
PersistentTemporaryDirectory
```

```
(  
    '_azw3_container'  
)
```

```
tdir
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
abspath
```

```
(
```

```
os
```

```
.
```

```
path
```

```
.
```

```
realpath
```

```
(
```

```
tdir
```

```
))
```

```
self
```

```
.
```

```
root
```

```
=
```

```
tdir
```

```
with
```

```
lopen
```

```
(
```

```
    pathtoazw3
```

```
,
```

```
    'rb'
```

```
)
```



```
as
```

```
stream
```

```
:
```

```
raw
```

```
=
```

```
stream
```

```
.
```

```
read
```

```
(
```

```
3
```

```
)
```

```
if
```

```
raw
```

```
==
```

```
b
```

```
'TPZ'
```

```
:
```

```
raise
```

```
InvalidMobi
```

```
(
```

```
—
```

```
(
```

```
'This is not a MOBI file. It is a Topaz file.'
```

```
))
```

```
try
```

```
:
```

```
header
```

```
=
```

```
MetadataHeader
```

```
(
```

```
stream
```

```
,
```

```
default_log
```

```
)
```

```
except
```

```
MobiError
```

```
:
```

```
raise
```

```
InvalidMobi
```

```
(
```

```
—
```

```
(
```

```
'This is not a MOBI file.'
```

```
))
```

```
if
```

```
header
```

```
.
```

```
encryption_type
```

```
!=
```

```
0
```

```
:
```

raise

DRMError
(

kf8_type

=

header

.

kf8_type

if

kf8_type

is

None

:

raise

InvalidMobi

(

—
(

'This MOBI file does not contain a KF8 format '

'book. KF8 is the new format from Amazon. calibre can '

'only edit MOBI files that contain KF8 books. Older '

```
'MOBI files without KF8 are not editable.'
))

if
    kf8_type
    ==
    'joint'
    :

    raise
        InvalidMobi
        (
            -
            (
                'This MOBI file contains both KF8 and '

                'older Mobi6 data. calibre can only edit MOBI files '

                'that contain only KF8 data.'
            ))

        try
            :

            opf_path
            ,

            obfuscated_fonts

            =
```

```
fork_job
(

'calibre.ebooks.oeb.polish.container'
,

'do_explode'
,

args
=
(
pathtoazw3
,

tdir
),

no_output
=
True
)[
'result'
]

except

WorkerError

as

e
:

log
(
e
.
orig_tb
```

```
)
```

```
raise
```

```
InvalidMobi
```

```
(
```

```
'Failed to explode MOBI'
```

```
)
```

```
super
```

```
()
```

```
.
```

```
__init__
```

```
(
```

```
tdir
```

```
,
```

```
opf_path
```

```
,
```

```
log
```

```
)
```

```
self
```

```
.
```

```
obfuscated_fonts
```

```
=
```

```
{
```

```
x
```

```
.
```

```
replace
```

```
(
```

```
os
```

```
.
```

```
sep
```

```
,
```

```
'/'
```

```
)  
  
for  
  
x  
  
in  
  
obfuscated_fonts  
}
```

```
def  
  
clone_data  
(  
self  
,  
  
dest_dir  
):
```

```
ans  
  
=  
  
super  
(  
.  
clone_data  
(  
dest_dir  
)
```

```
ans  
[  
'pathtoazw3'  
]  
  
=
```

```
self
.
pathtoazw3

ans
[
'obfuscated_fonts'
]

=

self
.
obfuscated_fonts
.
copy
()

return

ans

def

commit
(
self
,

outpath
=
None
,

keep_parsed
=
False
):
```



```
super
()  
.  
commit  
(  
keep_parsed  
=  
keep_parsed  
)
```

```
if
```

```
outpath
```

```
is
```

```
None
```

```
:
```

```
outpath
```

```
=
```

```
self
```

```
.
```

```
pathtoazw3
```

```
opf_to_azw3
```

```
(
```

```
self
```

```
.
```

```
name_path_map
```

```
[
```

```
self
```

```
.
```

```
opf_name
```

```
],
```

```
outpath
,

self
)

@property

def

path_to_ebook
(
self
):

return

self
.
pathtoazw3

@property

.
setter

def

path_to_ebook
(
self
,

val
):
```

```
self
.
pathtoazw3

=

val

@property

def

names_that_must_not_be_changed
(
self
):

return

set
(
self
.
name_path_map
)

# }}}

def

get_container
(
path
,

log
=
```

```
None
,

tdir
=
None
,

tweak_mode
=
False
):
```

```
if
log
```

```
is
None
:
```

```
log
=
default_log
```

```
try
:
```

```
isdir
=
```

```
os
.
path
.
```

```
isdir
```

```
(  
    path  
)
```

```
except
```

```
Exception
```

```
:
```

```
isdir
```

```
=
```

```
False
```

```
ebook
```

```
=
```

```
(  
    AZW3Container
```

```
if
```

```
path
```

```
.
```

```
rpartition
```

```
(
```

```
    ''
```

```
)
```

```
-
```

```
1
```

```
]
```

```
.
```

```
lower
```

```
()
```

```
in
```

```
{  
  'azw3'  
,  
  
  'mobi'  
,  
  
  'original_azw3'  
,  
  
  'original_mobi'  
}
```

and

not

isdir

else

```
EpubContainer  
(  
  path  
,  
  
  log  
,  
  
  tdir  
  =  
  tdir  
)
```

ebook

```
.  
tweak_mode
```

=

```
tweak_mode
```

```
return
```

```
ebook
```

```
def
```

```
test_roundtrip  
():
```

```
ebook
```

```
=
```

```
get_container  
(  
sys  
.  
argv  
[  
-  
1  
)
```

```
p
```

```
=
```

```
PersistentTemporaryFile  
(  
suffix  
=  
'.'  
+  
sys  
.  
argv  
[
```

```
-  
1  
]  
.  
rpartition  
(  
    '  
    .'  
)[  
-  
1  
])  
  
p  
.  
close  
()  
  
ebook  
.  
commit  
(  
    outpath  
=  
    p  
    .  
    name  
)  
  
ebook2  
=  
  
get_container  
(  
    p  
    .  
    name  
)
```



```
ebook3

=

get_container
(
p
.
name
)
```

```
diff

=

ebook3
.
compare_to
(
ebook2
)
```

```
if
```

```
diff
```

```
is
```

```
not
```

```
None
```

```
:
```

```
print
(
diff
)
```

```
if
__name__
==
'__main__'
:

test_roundtrip
()
```

Source code for calibre.ebooks.oeb.polish.cover

```
__license__
=
'GPL v3'

__copyright__
=
'2013, Kovid Goyal <kovid at kovidgoyal.net>'

__docformat__
=
'restructuredtext en'

import
shutil
,
re
,
os

from
calibre.ebooks.oeb.base
```

import

OPF

,

OEB_DOCS

,

XPath

,

XLINK

,

xml2text

from

[calibre.ebooks.oeb.polish.replace](#)

import

replace_links

,

get_recommended_folders

from

[calibre.utils.imghdr](#)

import

identify

from

[polyglot.builtins](#)

import

iteritems

```
def
set_azw3_cover
(
container
,
cover_path
,
report
,
options
=
None
):

existing_image
=
options
is
not
None
and
options
.
get
(
'existing_image'
,
```

False

)

name

=

None

found

=

True

for

gi

in

container

.

opf_xpath

(

'//opf:guide/opf:reference[@href and contains(@type, "cover")]'

):

href

=

gi

.

get

(

'href'

)

name

=

container

.

href_to_name

(

href

,

container

.

opf_name

)

container

.

remove_from_xml

(

gi

)

if

existing_image

:

name

=

cover_path

found

=

False

else

:

if

name

is

None

or

not

container

.

has_name

(

name

):

item

=

container

.

generate_item

(

name

=

'cover.jpeg'

,

id_prefix

=

'cover'


```
)  
  
name  
  
=  
  
container  
.  
href_to_name  
(  
item  
.  
get  
(  
'href'  
),  
  
container  
.  
opf_name  
)
```

found

=

False

href

=

```
container  
.  
name_to_href  
(  
name  
,
```

container

```
.  
opf_name  
)  
  
guide  
  
=  
  
container  
.  
opf_xpath  
(  
  '//opf:guide'  
) [  
  0  
]  
  
container  
.  
insert_into_xml  
(  
  guide  
  ,  
  
  guide  
  .  
  makeelement  
  (  
  
    OPF  
    (  
      'reference'  
    ),  
  
    href  
    =  
    href  
    ,  
  
    type
```

```
=  
'cover'  
)  
  
if  
  
not  
  
existing_image  
:  
  
with  
  
lopen  
(  
cover_path  
,  
  
'rb'  
)  
  
as  
  
src  
,  
  
container  
.  
open  
(  
name  
,  
  
'wb'  
)  
  
as  
  
dest  
:
```

```
shutil
.
copyfileobj
(
src
,
dest
)

container
.
dirty
(
container
.
opf_name
)

report
(
    _
    (
        'Cover updated'
    )
    if
found
else
    _
    (
        'Cover inserted'
    )
))

def
```

```
get_azw3_raster_cover_name
(
  container
):

items

=

container
.
opf_xpath
(
  '//opf:guide/opf:reference[@href and contains(@type, "cover")]'
)

if
items
:

return

container
.
href_to_name
(
  items
  [
    0
  ]
  .
  get
  (
    'href'
  ))
```

def

mark_as_cover_azw3

(

container

,

name

):

href

=

container

.

name_to_href

(

name

,

container

.

opf_name

)

found

=

False

for

item

in

container

.

```
opf_xpath
(  
  '//opf:guide/opf:reference[@href and contains(@type, "cover")]'  
):
```

```
item
```

```
.
```

```
set
```

```
(
```

```
'href'
```

```
,
```

```
href
```

```
)
```

```
found
```

```
=
```

```
True
```

```
if
```

```
not
```

```
found
```

```
:
```

```
for
```

```
guide
```

```
in
```

```
container
```

```
.
```

```
opf_xpath
```

```
(
```

```
'//opf:guide'
```

```
):  
  
container  
.  
insert_into_xml  
(  
  guide  
  ,  
  
  guide  
  .  
  makeelement  
  (  
  
    OPF  
    (  
      'reference'  
    ),  
  
    href  
    =  
    href  
    ,  
  
    type  
    =  
    'cover'  
  ))  
  
container  
.  
dirty  
(  
  container  
  .  
  opf_name  
)
```



```
def

get_raster_cover_name
(
container
):

if

container
.
book_type

==

'azw3'
:

return

get_azw3_raster_cover_name
(
container
)

return

find_cover_image
(
container
,

strict
=
True
)

def
```

```
get_cover_page_name  
(  
    container  
):
```

```
if
```

```
    container  
    .  
    book_type  
  
    ==
```

```
    'azw3'  
    :
```

```
return
```

```
return
```

```
find_cover_page  
(  
    container  
)
```

```
\[docs\]
```

```
def
```

```
set_cover  
(  
    container  
    ,  
  
    cover_path  
    ,
```

```
report
```

```
=
```

```
None
```

```
,
```

```
options
```

```
=
```

```
None
```

```
):
```

```
'''
```

Set the cover of the book to the image pointed to by cover_path.

:param cover_path: Either the absolute path to an image file or the

canonical name of an image in the book. When using an image in the book,

you must also set options, see below.

:param report: An optional callable that takes a single argument. It will

be called with information about the tasks being processed.

:param options: None or a dictionary that controls how the cover is set. The dictionary can have entries:

*****keep_aspect****: True or False (Preserve aspect ratio of covers in EPUB)*

*****no_svg****: True or False (Use an SVG cover wrapper in the EPUB titlepage)*

*****existing****: True or False (``cover\_path`` refers to an existing image in the book)*

```
'''
```

```
report
```

```
=
```

```
report

or

(
  lambda

x
:
x
)

if

container
.
book_type

==

'azw3'
:

set_azw3_cover
(
  container
,

  cover_path
,

  report
,

  options
=
options
)

else
```

```
:  
  
set_epub_cover  
(  
    container  
,  
  
    cover_path  
,  
  
    report  
,  
  
    options  
    =  
    options  
)
```

[\[docs\]](#)

def

```
mark_as_cover  
(  
    container  
,  
  
    name  
):
```

```
'''
```

Mark the specified image as the cover image.

```
'''
```

if

```
name
```

not

in

container

.

mime_map

:

raise

ValueError

(

'Cannot mark

%s

as cover as it does not exist'

%

name

)

mt

=

container

.

mime_map

[

name

]

if

not

is_raster_image

(

mt

```
):  
  
raise  
  
ValueError  
(  
    'Cannot mark  
    %s  
    as the cover image as it is not a raster image'  
  
    %  
  
    name  
)  
  
if  
  
    container  
    .  
    book_type  
  
    ==  
  
    'azw3'  
    :  
  
        mark_as_cover_azw3  
        (  
            container  
            ,  
  
            name  
        )  
  
    else  
    :  
  
        mark_as_cover_epub
```

```
(
container
,

name
)

#####
#

# The delightful EPUB cover processing


def

is_raster_image
(
media_type
):

return

media_type

and

media_type
.
lower
()

in

{

'image/png'
,

'image/jpeg'
```



```
,  
  
'image/jpg'  
,  
  
'image/gif'  
}
```

COVER_TYPES

=

{

'coverimagestandard'

,

'other.ms-coverimage-standard'

,

'other.ms-titleimage-standard'

,

'other.ms-titleimage'

,

'other.ms-coverimage'

,

'other.ms-thumbimage-standard'

,

'other.ms-thumbimage'

,

'thumbimagestandard'

,

```
'cover'  
}
```

```
def
```

```
find_cover_image2
```

```
(  
    container  
    ,
```

```
    strict
```

```
    =
```

```
    False
```

```
):
```

```
    manifest_id_map
```

```
    =
```

```
    container
```

```
    .
```

```
    manifest_id_map
```

```
    mm
```

```
    =
```

```
    container
```

```
    .
```

```
    mime_map
```

```
for
```

```
    meta
```

```
in
```

```
container
.  
opf_xpath  
(  
  '//opf:meta[@name="cover" and @content]'  
):
```

```
item_id
```

```
=
```

```
meta  
.  
get  
(  
  'content'  
)
```

```
name
```

```
=
```

```
manifest_id_map
```

```
..  
get  
(  
  item_id  
  ,
```

```
None  
)
```

```
media_type
```

```
=
```

```
mm
```

```
..  
get  
(
```

```
name
,
None
)

if
is_raster_image
(
media_type
):

return
name

# First look for a guide item with type == 'cover'

guide_type_map
=
container
.
guide_type_map

for
ref_type
,
name
in
iteritems
```

```
(  
guide_type_map  
):
```

```
if
```

```
ref_type
```

```
.
```

```
lower
```

```
()
```

```
==
```

```
'cover'
```

```
and
```

```
is_raster_image
```

```
(
```

```
mm
```

```
.
```

```
get
```

```
(
```

```
name
```

```
,
```

```
None
```

```
)):
```

```
return
```

```
name
```

```
if
```

```
strict
```

```
:
```

```
return

# Find the largest image from all possible guide cover items

largest_cover
=

(
None
,
0
)

for
ref_type
,
name
in
iteritems
(
guide_type_map
):

if
ref_type
.
lower
()
in
```

COVER_TYPES

and

is_raster_image

(
mm

.

get

(
name
,

None

)):

path

=

container

.

name_path_map

.

get

(
name
,

None

)

if

path

:

sz

=

```
os
.  
path  
.  
getsize  
(  
path  
)
```

```
if
```

```
sz
```

```
>
```

```
largest_cover  
[  
1  
]:
```

```
largest_cover
```

```
=
```

```
(  
name  
,  
sz  
)
```

```
if
```

```
largest_cover  
[  
0  
]:
```



```
return
```

```
largest_cover
```

```
[
```

```
0
```

```
]
```

```
def
```

```
find_cover_image3
```

```
(
```

```
container
```

```
):
```

```
for
```

```
name
```

```
in
```

```
container
```

```
.
```

```
manifest_items_with_property
```

```
(
```

```
'cover-image'
```

```
):
```

```
return
```

```
name
```

```
manifest_id_map
```

```
=
```

```
container
```

```
.
```

```
manifest_id_map
```

```
mm
```

```
=
```

```
container
```

```
.
```

```
mime_map
```

```
for
```

```
meta
```

```
in
```

```
container
```

```
.
```

```
opf_xpath
```

```
(
```

```
'//opf:meta[@name="cover" and @content]'
```

```
):
```

```
item_id
```

```
=
```

```
meta
```

```
.
```

```
get
```

```
(
```

```
'content'
```

```
)
```

```
name
```

```
=
```

```
manifest_id_map
```

```
.  
get  
(  
    item_id  
,  
    None  
)  
  
media_type  
  
=  
  
mm  
.  
get  
(  
    name  
,  
    None  
)  
  
if  
  
is_raster_image  
(  
    media_type  
):  
  
return  
  
name  
  
def  
  
find_cover_image  
(
```

```
container
,

strict
=
False
):

'Find a raster image marked as a cover in the OPF'

ver

=

container
.
opf_version_parsed

if

ver
.
major

<

3
:

return

find_cover_image2
(
container
,

strict
=
strict
```

```
)  
  
else  
:  
  
return  
  
find_cover_image3  
(  
    container  
)
```

```
def  
  
get_guides  
(  
    container  
):  
  
    guides  
  
    =  
  
    container  
    .  
    opf_xpath  
    (  
        '//opf:guide'  
    )
```

```
if  
  
not  
  
guides  
:
```

```
container
.
insert_into_xml
(
container
.
opf
,

container
.
opf
.
makeelement
(

OPF
(
'guide'
)))
```

guides

=

```
container
.
opf_xpath
(
'//opf:guide'
)
```

return

guides

def

```
mark_as_cover_epub
```

```
(
```

```
container
```

```
,
```

```
name
```

```
):
```

```
mmap
```

```
=
```

```
{
```

```
v
```

```
:
```

```
k
```

```
for
```

```
k
```

```
,
```

```
v
```

```
in
```

```
iteritems
```

```
(
```

```
container
```

```
.
```

```
manifest_id_map
```

```
)}  
  
if  
  
name  
  
not  
  
in
```

```
mmap
:
```

```
raise
```

```
ValueError
```

```
(
    'Cannot mark
    %s
    as cover as it is not in manifest'
```

```
%
```

```
name
)
```

```
mid
```

```
=
```

```
mmap
[
    name
]
```

```
ver
```

```
=
```

```
container
```

```
.
opf_version_parsed
```

```
# Remove all entries from the opf that identify a raster image as cover
```

```
for
```


meta

in

container

.

opf_xpath

(

'//opf:meta[@name="cover" and @content]'

):

container

.

remove_from_xml

(

meta

)

for

ref

in

container

.

opf_xpath

(

'//opf:guide/opf:reference[@href and @type]'

):

if

ref

.

get

(

'type'

)

```
.
lower
()

not

in

COVER_TYPES
:

continue

rname

=

container
.
href_to_name
(
ref
.
get
(
'href'
),
container
.
opf_name
)

mt

=

container
.
mime_map
```

```
.
get
(
  rname
,
  None
)

if

is_raster_image
(
  mt
):

  container
  .
  remove_from_xml
  (
    ref
  )

  if

  ver
  .
  major

  <

  3
  :

  # Add reference to image in <metadata>

  for
```

metadata

in

container

.

opf_xpath

(

'//opf:metadata'

):

m

=

metadata

.

makeelement

(

OPF

(

'meta'

),

name

=

'cover'

,

content

=

mid

)

container

.

insert_into_xml

(

metadata

,

```
m
)
```

```
# If no entry for cover exists in guide, insert one that points to this
```

```
# image
```

```
if
```

```
not
```

```
container
```

```
.
```

```
opf_xpath
```

```
(
```

```
'//opf:guide/opf:reference[@type="cover"]'
```

```
):
```

```
for
```

```
guide
```

```
in
```

```
get_guides
```

```
(
```

```
container
```

```
):
```

```
container
```

```
.
```

```
insert_into_xml
```

```
(
```

```
guide
```

```
,
```

```
guide
·
makeelement
(

OPF
(
'reference'
),

type
=
'cover'
,

href
=
container
·
name_to_href
(
name
,

container
·
opf_name
)))

else
:

container
·
apply_unique_properties
(
name
,

'cover-image'
```

```
)
```

```
container
```

```
.
```

```
dirty
```

```
(
```

```
container
```

```
.
```

```
opf_name
```

```
)
```

```
\[docs\]
```

```
def
```

```
mark_as_titlepage
```

```
(
```

```
container
```

```
,
```

```
name
```

```
,
```

```
move_to_start
```

```
=
```

```
True
```

```
):
```

```
'''
```

```
Mark the specified HTML file as the titlepage of the EPUB.
```

```
:param move_to_start: If True the HTML file is moved to the start of the spine
```

```
'''
```

```
ver
=
container
.
opf_version_parsed
```

```
if
```

```
move_to_start
:
```

```
for
```

```
item
,
```

```
q
,
```

```
linear
```

```
in
```

```
container
.
spine_iter
:
```

```
if
```

```
name
```

```
==
```

```
q
:
```


break

if

not

linear

:

item

.

set

(

'linear'

,

'yes'

)

if

item

.

getparent

()

.

index

(

item

)

>

0

:

container

.

```
insert_into_xml
(
    item
    .
    getparent
    (),

    item
    ,

    0
)

if
ver
.
major

<

3
:

for
ref

in
container
.
opf_xpath
(
    '//opf:guide/opf:reference[@type="cover"]'
):

ref
.
getparent
```

```
()  
.  
remove  
(  
ref  
)  
  
for  
  
guide  
  
in  
  
get_guides  
(  
container  
):  
  
container  
.  
insert_into_xml  
(  
guide  
,  
  
guide  
.  
makeelement  
(  
  
OPF  
(  
'reference'  
),  
  
type  
=  
'cover'  
,  

```

```
href
=
container
.
name_to_href
(
name
,

container
.
opf_name
)))

else
:

container
.
apply_unique_properties
(
name
,

'calibre:title-page'
)

container
.
dirty
(
container
.
opf_name
)
```

```
def
```

```
find_cover_page
```

```
(
```

```
container
```

```
):
```

```
'Find a document marked as a cover in the OPF'
```

```
ver
```

```
=
```

```
container
```

```
.
```

```
opf_version_parsed
```

```
mm
```

```
=
```

```
container
```

```
.
```

```
mime_map
```

```
if
```

```
ver
```

```
.
```

```
major
```

```
<
```

```
3
```

```
:
```

```
guide_type_map
```

```
=  
  
container  
.  
guide_type_map
```

```
for
```

```
ref_type  
,
```

```
name
```

```
in
```

```
iteritems  
(  
guide_type_map  
):
```

```
if
```

```
ref_type  
.  
lower  
()
```

```
==
```

```
'cover'
```

```
and
```

```
mm  
.  
get  
(  
name  
,
```

```
''
```

```
)
.
lower
()

in

OEB_DOCS
:

return

name

else
:

for

name

in

container
.
manifest_items_with_property
(
'calibre:title-page'
):

return

name

from

calibre.ebooks.oeb.polish.toc
```

```
import
```

```
get_landmarks
```

```
for
```

```
landmark
```

```
in
```

```
get_landmarks
```

```
(
```

```
container
```

```
):
```

```
if
```

```
landmark
```

```
[
```

```
'type'
```

```
]
```

```
==
```

```
'cover'
```

```
and
```

```
mm
```

```
.
```

```
get
```

```
(
```

```
landmark
```

```
[
```

```
'dest'
```

```
],
```

```
"
```

```
)
```

```
.
```

```
lower
```



```
()
```

```
in
```

```
OEB_DOCS
```

```
:
```

```
return
```

```
landmark
```

```
[
```

```
'dest'
```

```
]
```

```
def
```

```
fix_conversion_titlepage_links_in_nav
```

```
(
```

```
container
```

```
):
```

```
from
```

```
calibre.ebooks.oeb.polish.toc
```

```
import
```

```
find_existing_nav_toc
```

```
cover_page_name
```

```
=
```

```
find_cover_page
```

```
(
```

```
container
```

```
)
```

```
if
not
cover_page_name
:

return

nav_page_name

=

find_existing_nav_toc
(
container
)

if
not
nav_page_name
:

return

for
elem
in
container
.
parsed
(
```

```
nav_page_name
)
.
xpath
(
'//*[@data-calibre-removed-titlepage]'
):
```

```
elem
.
attrib
.
pop
(
'data-calibre-removed-titlepage'
)
```

```
elem
```

```
.
set
(
'href'
,
```

```
container
.
name_to_href
(
cover_page_name
,
nav_page_name
))
```

```
container
.
dirty
(
nav_page_name
)
```

```
def
```

```
find_cover_image_in_page
```

```
(
```

```
    container
```

```
,
```

```
    cover_page
```

```
):
```

```
    root
```

```
    =
```

```
    container
```

```
    .
```

```
    parsed
```

```
    (
```

```
        cover_page
```

```
    )
```

```
    body
```

```
    =
```

```
    XPath
```

```
    (
```

```
        '//h:body'
```

```
    )(
```

```
        root
```

```
    )
```

```
if
```

```
    len
```

```
    (
```

```
        body
```

```
)  
  
!=  
  
1  
:  
  
return  
  
body  
  
=  
  
body  
[  
0  
]  
  
images  
  
=  
  
[]  
  
for  
  
img  
  
in  
  
XPath  
(  
'descendant::h:img[@src]|descendant::svg:svg/descendant::svg:image'  
)(  
body  
):  
  
href
```

```
=
```

```
img
```

```
.
```

```
get
```

```
(
```

```
'src'
```

```
)
```

```
or
```

```
img
```

```
.
```

```
get
```

```
(
```

```
XLINK
```

```
(
```

```
'href'
```

```
))
```

```
if
```

```
href
```

```
:
```

```
name
```

```
=
```

```
container
```

```
.
```

```
href_to_name
```

```
(
```

```
href
```

```
,
```

```
base
```

```
=
```

```
cover_page
```

```
)
```

```
images
```

```
.
```

```
append
```

```
(
```

```
name
```

```
)
```

```
text
```

```
=
```

```
re
```

```
.
```

```
sub
```

```
(
```

```
r
```

```
'\s+'
```

```
,
```

```
"
```

```
,
```

```
xml2text
```

```
(
```

```
body
```

```
))
```

```
if
```

```
text
```

```
or
```

```
len
```

```
(
```

```
images
```

```
)
```

```
>
```

```

1
:

# Document has more content than a single image

return

if
images
:

return

images
[
0
]

def
clean_opf
(
container
):

'Remove all references to covers from the OPF'

manifest_id_map
=
container
.
```



```
manifest_id_map

for
meta

in
container
.
opf_xpath
(
'//opf:meta[@name="cover" and @content]'
):

name

=

manifest_id_map
.
get
(
meta
.
get
(
'content'
,
None
),
None
)

container
.
remove_from_xml
(
```

```
meta
)

if

name

and

name

in

container
.
name_path_map
:

yield

name

gtm

=

container
.
guide_type_map

for

ref

in

container
.
```

```
opf_xpath
(  
  '//opf:guide/opf:reference[@type]'  
):
```

```
typ
```

```
=
```

```
ref
```

```
.
```

```
get
```

```
(
```

```
  'type'
```

```
,
```

```
"
```

```
)
```

```
if
```

```
typ
```

```
.
```

```
lower
```

```
()
```

```
in
```

```
COVER_TYPES
```

```
:
```

```
container
```

```
.
```

```
remove_from_xml
```

```
(
```

```
  ref
```

```
)
```

```
name
```

```
=
```

```
gtm
```

```
.
```

```
get
```

```
(
```

```
typ
```

```
,
```

```
None
```

```
)
```

```
if
```

```
name
```

```
and
```

```
name
```

```
in
```

```
container
```

```
.
```

```
name_path_map
```

```
:
```

```
yield
```

```
name
```

```
ver
```

```
=
```

```
container
```

```
.
```

```
opf_version_parsed
```

```
if
ver
.
major

>

2
:

removed_names

=

container
.
apply_unique_properties
(
None
,
'cover-image'
,
'calibre:title-page'
)[
0
]

for
name
in
removed_names
:
```

yield

name

container

.

dirty

(

container

.

opf_name

)

def

create_epub_cover

(

container

,

cover_path

,

existing_image

,

options

=

None

):

from

[calibre.ebooks.conversion.config](#)

import

load_defaults

```
from
```

```
calibre.ebooks.oeb.transforms.cover
```

```
import
```

```
CoverManager
```

```
try
```

```
:
```

```
    ext
```

```
    =
```

```
    cover_path
```

```
    .
```

```
    rpartition
```

```
    (
```

```
    '..',
```

```
    )
```

```
    -
```

```
    1
```

```
    ]
```

```
    .
```

```
    lower
```

```
    ()
```

```
except
```

```
    Exception
```

```
    :
```

```
        ext
```

```
        =
```

```
'jpeg'

cname
,
tname
=
'cover.'
+
ext
,
'titlepage.xhtml'

recommended_folders
=

get_recommended_folders
(
container
,
(
cname
,
tname
))

if
existing_image
:
```



```
raster_cover
=
existing_image

manifest_id
=
{
v
:
k
for
k
,
v
in
iteritems
(
container
.
manifest_id_map
)][
existing_image
]

raster_cover_item
=
container
.
opf_xpath
```

```
(  
  '//opf:manifest/*[@id="  
  %s  
  "']  
  
  %  
  
  manifest_id  
)[  
  0  
]  
  
else  
:  
  
  folder  
  
  =  
  
  recommended_folders  
  [  
  cname  
  ]  
  
  if  
  
  folder  
  :  
  
  cname  
  
  =  
  
  folder  
  
  +  
  
  '/'
```

```
+  
  
cname  
  
raster_cover_item  
  
=  
  
container  
.  
generate_item  
(  
  cname  
,  
  
  id_prefix  
  =  
  'cover'  
)  
  
raster_cover  
  
=  
  
container  
.  
href_to_name  
(  
  raster_cover_item  
  .  
  get  
  (  
    'href'  
  ),  
  
  container  
  .  
  opf_name  
)
```

with

container

.

open

(

raster_cover

,

'wb'

)

as

dest

:

if

callable

(

cover_path

):

cover_path

(

'write_image'

,

dest

)

else

:

with

lopen

```
(
cover_path
,
'rb'
)

as

src
:

shutil
.
copyfileobj
(
src
,
dest
)

if

options

is

None
:

opts

=

load_defaults
(
'epub_output'
)
```

```
keep_aspect
```

```
=
```

```
opts
```

```
.
```

```
get
```

```
(
```

```
'preserve_cover_aspect_ratio'
```

```
,
```

```
False
```

```
)
```

```
no_svg
```

```
=
```

```
opts
```

```
.
```

```
get
```

```
(
```

```
'no_svg_cover'
```

```
,
```

```
False
```

```
)
```

```
else
```

```
:
```

```
keep_aspect
```

```
=
```

```
options
```

```
.
```

```
get
```

```
(
```

```
'keep_aspect'
```

```
,
```

```
False
```

```
)
```

```
no_svg
```

```
=
```

```
options
```

```
.
```

```
get
```

```
(
```

```
'no_svg'
```

```
,
```

```
False
```

```
)
```

```
if
```

```
no_svg
```

```
:
```

```
style
```

```
=
```

```
'style="height: 100
```

```
%%
```

```
'''
```

```
templ
```

```
=
```

```
CoverManager
```

```
.
```

```
NONSVG_TEMPLATE
```

```
.  
replace  
(  
    '__style__'  
    ,  
  
    style  
)
```

```
has_svg
```

```
=
```

```
False
```

```
else  
:
```

```
if
```

```
callable  
(  
    cover_path  
):
```

```
templ
```

```
=
```

```
(  
    options
```

```
or
```

```
{})
```

```
.  
get  
(
```



```
'template'
```

```
,
```

```
CoverManager
```

```
.
```

```
SVG_TEMPLATE
```

```
)
```

```
has_svg
```

```
=
```

```
'xlink:href'
```

```
in
```

```
templ
```

```
else
```

```
:
```

```
width
```

```
,
```

```
height
```

```
=
```

```
600
```

```
,
```

```
800
```

```
has_svg
```

```
=
```

```
True
```

```
try
:

if
existing_image
:

width
,
height
=
identify
(
container
.
raw_data
(
existing_image
,

decode
=
False
))[
1
:]

else
:

with
lopen
(
```

```
cover_path
,

'rb'
)

as

csrc
:

width
,

height

=

identify
(
csrc
)[
1
:]

except
:

container
.
log
.
exception
(
"Failed to get width and height of cover"
)

ar
```

```
=  
  
'xMidYMid meet'
```

```
if
```

```
keep_aspect
```

```
else
```

```
'none'
```

```
templ
```

```
=
```

```
CoverManager
```

```
.
```

```
SVG_TEMPLATE
```

```
.
```

```
replace
```

```
(
```

```
'__ar__'
```

```
,
```

```
ar
```

```
)
```

```
templ
```

```
=
```

```
templ
```

```
.
```

```
replace
```

```
(
```

```
'__viewbox__'
```

```
,
```

```
'0 0
```

```
%d
```

```
%d
',
%
(
width
,
height
))

templ

=

templ
.
replace
(
'__width__'
,

str
(
width
))

templ

=

templ
.
replace
(
'__height__'
,

str
(
height
```

```
))

folder

=

recommended_folders
[
tname
]

if

folder
:

tname

=

folder

+

'/'

+

tname

titlepage_item

=

container
.
generate_item
(
tname
```

```
,  
  
id_prefix  
=  
'titlepage'  
)  
  
titlepage  
  
=  
  
container  
.  
href_to_name  
(  
titlepage_item  
.  
get  
(  
'href'  
),  
  
container  
.  
opf_name  
)  
  
raw  
  
=  
  
templ  
  
%  
  
container  
.  
name_to_href  
(  
raster_cover
```

```
,  
  
titlepage  
)
```

with

```
container
```

```
.  
open  
(  
titlepage  
,  
  
'wb'  
)
```

as

```
f  
:
```

if

not

```
isinstance
```

```
(  
raw  
,
```

```
bytes  
):
```

```
raw
```

```
=
```

```
raw
```

```
.
```



```
encode
(
'utf-8'
)

f
.
write
(
raw
)

# We have to make sure the raster cover item has id="cover" for the moron

# that wrote the Nook firmware

if

raster_cover_item
.
get
(
'id'
)

!=

'cover'
:

from

calibre.ebooks.oeb.base

import

uuid_id
```

```
newid
```

```
=
```

```
uuid_id  
(
```

```
for
```

```
item
```

```
in
```

```
container
```

```
.
```

```
opf_xpath
```

```
(
```

```
'//*[@id="cover"]'
```

```
):
```

```
item
```

```
.
```

```
set
```

```
(
```

```
'id'
```

```
,
```

```
newid
```

```
)
```

```
for
```

```
item
```

```
in
```

```
container
```

```
.
```

```
opf_xpath  
(  
  '//*[@idref="cover"]'  
):
```

```
item
```

```
.  
set  
(  
  'idref'  
,
```

```
newid  
)
```

```
raster_cover_item
```

```
.  
set  
(  
  'id'  
,  
  
  'cover'  
)
```

```
spine
```

```
=
```

```
container  
.  
opf_xpath  
(  
  '//opf:spine'  
)[  
  0  
]
```

```
ref
=

spine
.
makeelement
(
OPF
(
'itemref'
),

idref
=
titlepage_item
.
get
(
'id'
))

container
.
insert_into_xml
(
spine
,

ref
,

index
=
0
)

ver
=
```

```
container
.  
opf_version_parsed
```

```
if
```

```
ver
```

```
.  
major
```

```
<
```

```
3
```

```
:
```

```
guide
```

```
=
```

```
container  
.  
opf_get_or_create  
(  
'guide'  
)
```

```
container  
.  
insert_into_xml  
(  
guide  
,
```

```
guide  
.  
makeelement  
(
```

```
OPF
```

```
(
'reference'
),

type
=
'cover'
,

title
=
-
(
'Cover'
),

href
=
container
.
name_to_href
(
titlepage
,

base
=
container
.
opf_name
)))

metadata
=
container
.
opf_get_or_create
(
'metadata'
```

```
)

meta
=

metadata
.
makeelement
(
OPF
(
'meta'
),
name
=
'cover'
)

meta
.
set
(
'content'
,
raster_cover_item
.
get
(
'id'
))

container
.
insert_into_xml
(
metadata
,
```

```
meta
)
```

```
else
:
```

```
container
.
apply_unique_properties
(
raster_cover
,
'cover-image'
)
```

```
container
.
apply_unique_properties
(
titlepage
,
'calibre:title-page'
)
```

```
if
```

```
has_svg
:
```

```
container
.
add_properties
(
titlepage
,
```



```
'svg'  
)
```

```
return
```

```
raster_cover
```

```
,
```

```
titlepage
```

```
def
```

```
remove_cover_image_in_page
```

```
(
```

```
container
```

```
,
```

```
page
```

```
,
```

```
cover_images
```

```
):
```

```
for
```

```
img
```

```
in
```

```
container
```

```
.
```

```
parsed
```

```
(
```

```
page
```

```
)
```

```
.
```

```
xpath
```

```
(  
'//*[local-name()="img" and @src]'  
):
```

href

=

img

.

get

(

'src'

)

name

=

container

.

href_to_name

(

href

,

page

)

if

name

in

cover_images

:

img

```
.
getparent
()
.
remove
(
img
)

break

def

has_epub_cover
(
container
):

if

find_cover_image
(
container
):

return

True

if

find_cover_page
(
container
):
```

```
return
```

```
True
```

```
spine_items
```

```
=
```

```
tuple
```

```
(
```

```
container
```

```
.
```

```
spine_items
```

```
)
```

```
if
```

```
spine_items
```

```
:
```

```
candidate
```

```
=
```

```
container
```

```
.
```

```
abspath_to_name
```

```
(
```

```
spine_items
```

```
[
```

```
0
```

```
])
```

```
if
```

```
find_cover_image_in_page
```

```
(
```

```
container
```

```
,
```

```
candidate
)
```

```
is
```

```
not
```

```
None
```

```
:
```

```
return
```

```
True
```

```
return
```

```
False
```

```
def
```

```
set_epub_cover
```

```
(
```

```
container
```

```
,
```

```
cover_path
```

```
,
```

```
report
```

```
,
```

```
options
```

```
=
```

```
None
```

```
,
```

```
image_callback
```

```
=
```

```
None
):

existing_image

=

options

is

not

None

and

options
.
get
(
'existing_image'
,

False
)

if

existing_image
:

existing_image

=

cover_path

cover_image
```

=

```
find_cover_image  
(  
    container  
)
```

cover_page

=

```
find_cover_page  
(  
    container  
)
```

wrapped_image

=

extra_cover_page

=

None

updated

=

False

log

=

container

.

```
log
```

```
possible_removals
```

```
=
```

```
set
```

```
(  
    clean_opf  
    (  
        container  
    ))
```

```
possible_removals
```

```
# TODO: Handle possible_removals and also iterate over links in the removed
```

```
# pages and handle possibly removing stylesheets referred to by them.
```

```
image_callback_called
```

```
=
```

```
False
```

```
spine_items
```

```
=
```

```
tuple
```

```
(  
    container  
    .  
    spine_items  
)
```



```
if
cover_page

is
None

and

spine_items
:

# Check if the first item in the spine is a simple cover wrapper

candidate

=

container
.
abspath_to_name
(
spine_items
[
0
])

if

find_cover_image_in_page
(
container
,
candidate
)
```

```
is
not
None
:

cover_page

=

candidate

if
cover_page
is
not
None
:

log
(
'Found existing cover page'
)

wrapped_image

=

find_cover_image_in_page
(
container
,
```

```
cover_page
)

if
len
(
spine_items
)

>

1
:

# Look for an extra cover page

c

=

container
.
abspath_to_name
(
spine_items
[
1
])

if

c

!=

cover_page
:
```

```
candidate

=

find_cover_image_in_page
(
container
,
c
)

if
candidate

and
candidate

in
{
wrapped_image
,
cover_image
}:

log
(
'Found an extra cover page that is a simple wrapper, removing it'
)

# This page has only a single image and that image is the

# cover image, remove it.
```

```
container
.
remove_item
(
c
)

extra_cover_page

=

c

spine_items

=

spine_items
[:
1
]

+

spine_items
[
2
:]

elif

candidate

is

None
:
```

Remove the cover image if it is the first image in this

page

remove_cover_image_in_page

(

container

,

c

,

{

wrapped_image

,

cover_image

})

if

wrapped_image

is

not

None

:

The cover page is a simple wrapper around a single cover image,

we can remove it safely.

```
log
(
f
'Existing cover page
{
cover_page
}
is a simple wrapper, removing it'
)
```

```
container
.
remove_item
(
cover_page
)
```

if

wrapped_image

!=

existing_image
:

if

image_callback

is

not

None

and

not

```
image_callback_called  
:
```

```
image_callback  
(  
    cover_image  
,  
    wrapped_image  
)
```

```
image_callback_called  
  
=
```

True

```
container  
.  
remove_item  
(  
    wrapped_image  
)
```

```
updated  
  
=
```

True

if

```
image_callback  
  
is
```

not

None

and

not

image_callback_called
:

image_callback_called

=

True

image_callback
(
cover_image
,
wrapped_image
)

if

cover_image

and

cover_image

!=

wrapped_image
:

Remove the old cover image

```
if
```

```
cover_image
```

```
!=
```

```
existing_image
```

```
:
```

```
container
```

```
.
```

```
remove_item
```

```
(
```

```
cover_image
```

```
)
```

```
# Insert the new cover
```

```
raster_cover
```

```
,
```

```
titlepage
```

```
=
```

```
create_epub_cover
```

```
(
```

```
container
```

```
,
```

```
cover_path
```

```
,
```

```
existing_image
```

```
,
```

```
options
```

```
=
```

```
options
)
```

```
report
(
-
(
'Cover updated'
)
```

```
if
```

```
updated
```

```
else
```

```
-
(
'Cover inserted'
))
```

```
# Replace links to the old cover image/cover page
```

```
link_sub
```

```
=
```

```
{
s
:
d
```

```
for
```

```
s
```

```
,
```

```
d
```

in

iteritems

{

cover_page

:

titlepage

,

wrapped_image

:

raster_cover

,

cover_image

:

raster_cover

,

extra_cover_page

:

titlepage

})

if

s

is

not

None

and

s

```
!=  
  
d  
}  
  
if  
  
link_sub  
:  
  
replace_links  
(  
container  
,  
  
link_sub  
,  
  
frag_map  
=  
lambda  
  
x  
,  
  
y  
:  
None  
)  
  
return  
  
raster_cover  
,  
  
titlepage
```


Source code for calibre.ebooks.oeb.polish.css

```
#!/usr/bin/env python
```

```
__license__
```

```
=
```

```
'GPL v3'
```

```
__copyright__
```

```
=
```

```
'2014, Kovid Goyal <kovid at kovidgoyal.net>'
```

```
import
```

```
re
```

```
from
```

```
collections
```

```
import
```

```
defaultdict
```

```
from
```

```
functools
```

import

partial

from

operator

import

itemgetter

from

css_parser.css

import

CSSRule

,

CSSStyleDeclaration

from

css_selectors

import

parse

,

SelectorSyntaxError

from

calibre

import

force_unicode


```
from
```

```
calibre.ebooks.oeb.base
```

```
import
```

```
OEB_STYLES
```

```
,
```

```
OEB_DOCS
```

```
,
```

```
XHTML
```

```
,
```

```
css_text
```

```
from
```

```
calibre.ebooks.oeb.normalize_css
```

```
import
```

```
normalize_filter_css
```

```
,
```

```
normalizers
```

```
from
```

```
calibre.ebooks.oeb.polish.pretty
```

```
import
```

```
pretty_script_or_style
```

```
,
```

```
pretty_xml_tree
```

```
,
```

```
serialize
```

from

calibre.utils.icu

import

numeric_sort_key

from

css_selectors

import

Select

,

SelectorError

from

polyglot.builtins

import

iteritems

,

itervalues

from

polyglot.functools

import

lru_cache

class

SelectorStatus

```
:  
  
any_used  
:  
  
bool  
  
=  
  
False  
  
any_unused  
:  
  
bool  
  
=  
  
False  
  
def  
  
mark_used_selectors  
(  
rules  
,  
  
log  
,  
  
select  
):  
  
ans  
  
=  
  
SelectorStatus
```

```
()

for
rule

in
rules
:

for
selector

in
rule
.
selectorList
:

if

getattr
(
selector
,
'calibre_used'
,

False
):

ans
.
any_used
```

```
=
```

```
True
```

```
continue
```

```
try
```

```
:
```

```
if
```

```
select
```

```
.
```

```
has_matches
```

```
(
```

```
selector
```

```
.
```

```
selectorText
```

```
):
```

```
selector
```

```
.
```

```
calibre_used
```

```
=
```

```
True
```

```
ans
```

```
.
```

```
any_used
```

```
=
```

```
True
```

```
else
```

```
:  
  
ans  
.  
any_unused  
  
=  
  
True  
  
selector  
.  
calibre_used  
  
=  
  
False  
  
except  
  
SelectorError  
:  
  
# Cannot parse/execute this selector, be safe and assume it  
  
# matches something  
  
selector  
.  
calibre_used  
  
=  
  
True  
  
ans
```

```
.
any_used

=

True

return

ans

def

get_imported_sheets
(
name
,
container
,
sheets
,
recursion_level
=
10
,
sheet
=
None
):

ans

=

set
```

```
()

sheet

=

sheet

or

sheets
[
  name
]

for

rule

in

sheet
.
cssRules
.
rulesOfType
(
  CSSRule
.
  IMPORT_RULE
):

if

rule
.
href
:
```



```
iname
=
container
.
href_to_name
(
rule
.
href
,
name
)

if

iname

in

sheets
:

ans
.
add
(
iname
)

if

recursion_level

>

0
:
```

```
for
imported_sheet

in

tuple
(
ans
):

ans

|=

get_imported_sheets
(
imported_sheet
,

container
,

sheets
,

recursion_level
=
recursion_level
-
1
)

ans
.
discard
(
name
)
```

return

ans

def

merge_declarations

(

first

,

second

):

for

prop

in

second

.

getProperties

():

first

.

setProperty

(

prop

)

def

merge_identical_selectors

```
(
sheet
):

' Merge rules that have identical selectors '
```

```
selector_map
```

```
=
```

```
defaultdict
(
list
)
```

```
for
```

```
rule
```

```
in
```

```
sheet
.
cssRules
.
rulesOfType
(
CSSRule
.
STYLE_RULE
):
```

```
selector_map
[
rule
.
selectorText
]
```

```
.
```

```
append
(
rule
)

remove

=

[]

for

rule_group

in

intervalvalues
(
selector_map
):

if

len
(
rule_group
)

>

1
:

for

i

in
```

```
range
(
1
,

len
(
rule_group
)):

merge_declarations
(
rule_group
[
0
]
.
style
,

rule_group
[
i
]
.
style
)

remove
.
append
(
rule_group
[
i
])

for
```

```
rule
```

```
in
```

```
remove
```

```
:
```

```
sheet
```

```
.
```

```
cssRules
```

```
.
```

```
remove
```

```
(
```

```
rule
```

```
)
```

```
return
```

```
len
```

```
(
```

```
remove
```

```
)
```

```
def
```

```
merge_identical_properties
```

```
(
```

```
sheet
```

```
):
```

```
' Merge rules having identical properties '
```

```
properties_map
```

```
=
```

```
defaultdict
```

```
(  
list  
)  
  
def  
  
declaration_key  
(  
declaration  
):  
  
return  
  
tuple  
(  
sorted  
(  
  
((  
prop  
.  
name  
,  
  
prop  
.  
propertyValue  
.  
value  
)  
  
for  
  
prop  
  
in  
  
declaration  
.  

```



```
getProperties  
()),
```

```
key  
=  
itemgetter  
(  
0  
)  
  
))
```

```
for
```

```
idx  
,
```

```
rule
```

```
in
```

```
enumerate  
(  
sheet  
.  
cssRules  
):
```

```
if
```

```
rule
```

```
.  
type
```

```
==
```

```
CSSRule
```

```
.
```

```
STYLE_RULE
```

```
:
```

```
properties_map
```

```
[
```

```
declaration_key
```

```
(
```

```
rule
```

```
.
```

```
style
```

```
)]
```

```
.
```

```
append
```

```
((
```

```
idx
```

```
,
```

```
rule
```

```
))
```

```
removals
```

```
=
```

```
[]
```

```
num_merged
```

```
=
```

```
0
```

```
for
```

```
rule_group
```

```
in
```

```
properties_map
```

```
.
```

```
values
```

```
():
```

```
if
```

```
len
```

```
(
```

```
rule_group
```

```
)
```

```
<
```

```
2
```

```
:
```

```
continue
```

```
num_merged
```

```
+=
```

```
len
```

```
(
```

```
rule_group
```

```
)
```

```
selectors
```

```
=
```

```
rule_group
```

```
[
```

```
0
```

```
][
```

```
1
```

```
]
```

```
.
```

```
selectorList
```

```
seen
```

```
=
```

```
{
```

```
s
```

```
.
```

```
selectorText
```

```
for
```

```
s
```

```
in
```

```
selectors
```

```
}
```

```
rules
```

```
=
```

```
iter
```

```
(
```

```
rule_group
```

```
)
```

```
next
```

```
(
```

```
rules
```

```
)
```

```
for
```

```
idx
```

```
,
```

rule

in

rules

:

removals

.

append

(

idx

)

for

s

in

rule

.

selectorList

:

q

=

s

.

selectorText

if

q

not

in

seen

:

seen

.

add

(

q

)

selectors

.

append

(

s

)

for

idx

in

sorted

(

removals

,

reverse

=

True

):

sheet

.

cssRules

.

```
pop
(
idx
)

return

num_merged

def

remove_unused_selectors_and_rules
(
rules_container
,

rules
,

removal_stats
):

ans

=

SelectorStatus
()

for

r

in

rules
:
```

```
removals

=

[]

for

i
,

sel

in

enumerate
(
r
.
selectorList
):

if

getattr
(
sel
,

'calibre_used'
,

True
):

ans
.
any_used
```



```
=  
  
True  
  
else  
:  
  
removals  
.  
append  
(  
i  
)  
  
if  
  
removals  
:  
  
ans  
.  
any_unused  
  
=  
  
True  
  
if  
  
len  
(  
removals  
)  
  
==  
  
len  
(
```

```
r
.
selectorList
):

rules_container
.
remove
(
r
)

removal_stats
[
'rules'
]

+=

1

else
:

removal_stats
[
'selectors'
]

+=

len
(
removals
)

for
```

```
i
in

reversed
(
removals
):
```

```
del
```

```
r
.
selectorList
[
i
]
```

```
return
```

```
ans
```

```
\[docs\]
```

```
def
```

```
remove_unused_css
(
```

```
container
,
```

```
report
=
```

```
None
```

```
,
```

```
remove_unused_classes
=
False
,

merge_rules
=
False
,

merge_rules_with_identical_properties
=
False
,

remove_unreferenced_sheets
=
False
,

):

'''
```

Remove all unused CSS rules from the book. An unused CSS rule is one that does not match any actual content.

:param report: An optional callable that takes a single argument. It is called with information about the operations being performed.

:param remove_unused_classes: If True, class attributes in the HTML that do not match any CSS rules are also removed.

:param merge_rules: If True, rules with identical selectors are merged.

:param merge_rules_with_identical_properties: If True, rules with identical properties are merged.

:param remove_unreferenced_sheets: If True, stylesheets that are not referenced by any content are removed

'''

report

=

report

or

(

lambda

x

:

x

)

def

safe_parse

(

name

):

try

:

return

container

.

parsed

(

```
name
)

except
TypeError
:

pass

sheets

=

{
name
:
safe_parse
(
name
)

for
name
,
mt

in

iteritems
(
container
.
mime_map
)

if
```

```
mt
```

```
in
```

```
OEB_STYLES
```

```
}
```

```
sheets
```

```
=
```

```
{
```

```
k
```

```
:
```

```
v
```

```
for
```

```
k
```

```
,
```

```
v
```

```
in
```

```
iteritems
```

```
(
```

```
sheets
```

```
)
```

```
if
```

```
v
```

```
is
```

```
not
```

```
None
```

```
}
```

```
num_merged

=

num_rules_merged

=

0

if

merge_rules
:

for

name
,

sheet

in

iteritems
(
sheets
):

num

=

merge_identical_selectors
(
sheet
)

if
```


num

:

container

.

dirty

(

name

)

num_merged

+=

num

if

merge_rules_with_identical_properties

:

for

name

,

sheet

in

iteritems

(

sheets

):

num

```
=  
  
merge_identical_properties  
(  
sheet  
)
```

```
if
```

```
num  
:
```

```
container  
.  
dirty  
(  
name  
)
```

```
num_rules_merged
```

```
+=
```

```
num
```

```
import_map
```

```
=  
  
{  
name  
:  
get_imported_sheets  
(  
name  
,  
container  
,
```

```
sheets  
)
```

```
for
```

```
name
```

```
in
```

```
sheets  
}
```

```
unreferenced_sheets
```

```
=
```

```
set  
(  
sheets  
)
```

```
if
```

```
remove_unused_classes  
:
```

```
class_map
```

```
=
```

```
{  
name  
: {  
icu_lower  
(  
x  
)
```

```
for
```

```
x
```

```
in
```

```
classes_in_rule_list
```

```
(
```

```
sheet
```

```
.
```

```
cssRules
```

```
)}
```

```
for
```

```
name
```

```
,
```

```
sheet
```

```
in
```

```
iteritems
```

```
(
```

```
sheets
```

```
)}
```

```
style_rules
```

```
=
```

```
{
```

```
name
```

```
:
```

```
tuple
```

```
(
```

```
sheet
```

```
.
```

```
cssRules
```

```
.
```

```
rulesOfType
```

```
(
```

```
CSSRule
```

```
.
STYLE_RULE
))

for

name
,
sheet

in

iteritems
(
sheets
)}

removal_stats

=

{
'rules'
:
0
,
'selectors'
:
0
}

num_of_removed_classes

=

0
```

for

name

,

mt

in

iteritems

(

container

.

mime_map

):

if

mt

not

in

OEB_DOCS

:

continue

root

=

container

.

parsed

(

```
name
)

select

=

Select
(
root
,

ignore_inappropriate_pseudo_classes
=
True
)

used_classes

=

set
()

for

style

in

root
.
xpath
(
'//*[local-name()="style"]'
):

if
```

```
style
```

```
.
```

```
get
```

```
(
```

```
'type'
```

```
,
```

```
'text/css'
```

```
)
```

```
==
```

```
'text/css'
```

```
and
```

```
style
```

```
.
```

```
text
```

```
:
```

```
sheet
```

```
=
```

```
container
```

```
.
```

```
parse_css
```

```
(
```

```
style
```

```
.
```

```
text
```

```
)
```

```
if
```

```
merge_rules
```

```
:
```

```
num
```


=

```
merge_identical_selectors  
(  
  sheet  
)
```

if

```
num  
:
```

```
num_merged
```

+=

```
num
```

```
container
```

```
.  
dirty  
(  
  name  
)
```

if

```
merge_rules_with_identical_properties  
:
```

```
num
```

=

```
merge_identical_properties  
(  
  sheet
```

```
)  
  
if  
  
num  
:  
  
num_rules_merged  
  
+=  
  
num  
  
container  
.  
dirty  
(  
name  
)  
  
if  
  
remove_unused_classes  
:  
  
used_classes  
  
|=  
  
{  
icu_lower  
(  
x  
)  
  
for  
  
x
```

in

```
classes_in_rule_list  
(  
    sheet  
    .  
    cssRules  
)}
```

imports

=

```
get_imported_sheets  
(  
    name  
    ,  
    container  
    ,  
    sheets  
    ,  
    sheet  
    =  
    sheet  
)
```

for

imported_sheet

in

```
imports  
:
```

unreferenced_sheets

```
.
discard
(
imported_sheet
)

mark_used_selectors
(
style_rules
[
imported_sheet
],

container
.
log
,

select
)

if

remove_unused_classes
:

used_classes

|=

class_map
[
imported_sheet
]

rules

=
```

```
tuple
(
sheet
.
cssRules
.
rulesOfType
(
CSSRule
.
STYLE_RULE
))
```

if

```
mark_used_selectors
(
rules
,
container
.
log
,
select
)
.
any_unused
:
```

```
remove_unused_selectors_and_rules
(
sheet
.
cssRules
,
rules
,
```

```
removal_stats  
)
```

```
style
```

```
.
```

```
text
```

```
=
```

```
force_unicode
```

```
(
```

```
sheet
```

```
.
```

```
cssText
```

```
,
```

```
'utf-8'
```

```
)
```

```
pretty_script_or_style
```

```
(
```

```
container
```

```
,
```

```
style
```

```
)
```

```
container
```

```
.
```

```
dirty
```

```
(
```

```
name
```

```
)
```

```
for
```

```
link
```

in

root

.

xpath

(

'//*[local-name()="link" and @href]'

):

sname

=

container

.

href_to_name

(

link

.

get

(

'href'

),

name

)

if

sname

not

in

sheets

:

continue

```
mark_used_selectors
```

```
(
```

```
style_rules
```

```
[
```

```
sname
```

```
],
```

```
container
```

```
.
```

```
log
```

```
,
```

```
select
```

```
)
```

```
if
```

```
remove_unused_classes
```

```
:
```

```
used_classes
```

```
|=
```

```
class_map
```

```
[
```

```
sname
```

```
]
```

```
unreferenced_sheets
```

```
.
```

```
discard
```

```
(
```

```
sname
```

```
)
```

```
for
```


iname

in

import_map

[
sname
]:

unreferenced_sheets

.

discard

(
iname
)

mark_used_selectors

(
style_rules
[
iname
],

container

.

log

,

select

)

if

remove_unused_classes

:

used_classes

```
|=  
  
class_map  
[  
  iname  
]  
  
if  
  
remove_unused_classes  
:  
  
for  
  
  elem  
  
in  
  
    root  
    .  
    xpath  
    (  
      '//*[@class]'  
    ):  
  
      original_classes  
      ,  
  
      classes  
  
      =  
  
      elem  
      .  
      get  
      (  
        'class'  
      ,
```

```
"
)
.
split
(),

[]

for

x

in

original_classes
:

if

icu_lower
(
x
)

in

used_classes
:

classes
.
append
(
x
)

if

len
```

```
(
classes
)

!=

len
(
original_classes
):

if

classes
:

elem
.
set
(
'class'
,
''
.
join
(
classes
))

else
:

del

elem
.
attrib
[
```

```
'class'
```

```
]
```

```
num_of_removed_classes
```

```
+=
```

```
len
```

```
(
```

```
original_classes
```

```
)
```

```
-
```

```
len
```

```
(
```

```
classes
```

```
)
```

```
container
```

```
.
```

```
dirty
```

```
(
```

```
name
```

```
)
```

```
for
```

```
name
```

```
,
```

```
sheet
```

```
in
```

```
iteritems
```

```
(
```

```
sheets
```

```
):
```

if

name

in

unreferenced_sheets

:

continue

q

=

remove_unused_selectors_and_rules

(

sheet

.

cssRules

,

style_rules

[

name

],

removal_stats

)

if

q

.

any_unused

:

```
container
.
dirty
(
name
)

num_sheets_removed

=

0

if

remove_unreferenced_sheets

and

len
(
unreferenced_sheets
):

num_sheets_removed

+=

len
(
unreferenced_sheets
)

for

uname

in
```

unreferenced_sheets

:

container

.

remove_item

(

uname

)

num_changes

=

num_merged

+

num_of_removed_classes

+

num_rules_merged

+

removal_stats

[

'rules'

]

+

removal_stats

[

'selectors'

]

+


```
num_sheets_removed

if
num_changes
>
0
:

if
removal_stats
[
'rules'
]:

report
(
ngettext
(
'Removed one unused CSS style rule'
,
'Removed
{}
unused CSS style rules'
,
removal_stats
[
'rules'
])
.
format
(
removal_stats
[
```

```
'rules'
]))

if

removal_stats
[
'selectors'
]:

report
(
ngettext
(
'Removed one unused CSS selector'
,

'Removed
{}
unused CSS selectors'
,

removal_stats
[
'selectors'
])
.
format
(
removal_stats
[
'selectors'
]))

if

num_of_removed_classes

>
```

```
0
:

report
(
  ngettext
  (
    'Removed one unused class from the HTML'
    ,
    'Removed
    {}
    unused classes from the HTML'
    ,
    num_of_removed_classes
  )
  .
  format
  (
    num_of_removed_classes
  ))

if
num_merged
>
0
:

report
(
  ngettext
  (
    'Merged one CSS style rule with identical selectors'
    ,
```

```
'Merged
{}
CSS style rules with identical selectors'
,

num_merged
)
.
format
(
num_merged
))

if

num_rules_merged

>

0
:

report
(
ngettext
(
'Merged one CSS style rule with identical properties'
,

'Merged
{}
CSS style rules with identical properties'
,

num_rules_merged
)
.
format
```

```
(
num_rules_merged
))

if

num_sheets_removed
:

report
(
ngettext
(
'Removed one unreferenced stylesheet'
,
'Removed
{}
unreferenced stylesheets'
,
num_sheets_removed
)
.
format
(
num_sheets_removed
))

if

not

removal_stats
[
'rules'
]:
```

```
report
(
-
(
'No unused CSS style rules found'
))
```

if

not

```
removal_stats
[
'selectors'
]:
```

```
report
(
-
(
'No unused CSS selectors found'
))
```

if

remove_unused_classes

and

num_of_removed_classes

==

0

:

```
report
(
-
```

```
(
'No unused class attributes found'
))

if

merge_rules

and

num_merged

==

0
:

report
(
-
(
'No style rules that could be merged found'
))

if

remove_unreferenced_sheets

and

num_sheets_removed

==

0
:

report
(
```

```
-  
(  
'No unused stylesheets found'  
)
```

```
return
```

```
num_changes
```

```
>
```

```
0
```

```
def
```

```
filter_declaration
```

```
(  
style  
,
```

```
properties  
=  
()):
```

```
changed
```

```
=
```

```
False
```

```
for
```

```
prop
```

```
in
```

```
properties  
:
```



```
if
  style
  .
  removeProperty
  (
  prop
  )
```

```
!=
```

```
"
```

```
:
```

```
changed
```

```
=
```

```
True
```

```
all_props
```

```
=
```

```
set
(
  style
  .
  keys
  ())
```

```
for
```

```
prop
```

```
in
```

```
style
```

```
.
```

```
getProperties  
():
```

```
n
```

```
=
```

```
normalizers
```

```
.
```

```
get
```

```
(
```

```
prop
```

```
.
```

```
name
```

```
,
```

```
None
```

```
)
```

```
if
```

```
n
```

```
is
```

```
not
```

```
None
```

```
:
```

```
normalized
```

```
=
```

```
n
```

```
(
```

```
prop
```

```
.
```

```
name
```

```
,
```

```
prop
.  
propertyValue  
)
```

removed

=

```
properties  
.  
intersection  
(  
  set  
(  
  normalized  
)
```

if

```
removed  
:
```

changed

=

True

```
style  
.  
removeProperty  
(  
  prop  
  .  
  name  
)
```

for

prop

in

set

(

normalized

)

-

removed

-

all_props

:

style

.

setProperty

(

prop

,

normalized

[

prop

])

return

changed

def

```
filter_sheet
```

```
(
```

```
sheet
```

```
,
```

```
properties
```

```
=
```

```
()):
```

```
from
```

```
css_parser.css
```

```
import
```

```
CSSRule
```

```
changed
```

```
=
```

```
False
```

```
remove
```

```
=
```

```
[]
```

```
for
```

```
rule
```

```
in
```

```
sheet
```

```
.
```

```
cssRules
```

```
.
```

```
rulesOfType  
(  
  CSSRule  
  .  
  STYLE_RULE  
):
```

if

```
filter_declaration  
(  
  rule  
  .  
  style  
  ,  
  properties  
):
```

changed

=

True

if

```
rule  
.  
style  
.  
length
```

==

0

:

remove

```
.  
append  
(  
rule  
)
```

```
for
```

```
rule
```

```
in
```

```
remove  
:
```

```
sheet
```

```
.  
cssRules
```

```
.  
remove  
(  
rule  
)
```

```
return
```

```
changed
```

```
def
```

```
transform_inline_styles  
(  
container  
,  
name  
,
```

```
transform_sheet
,

transform_style
):

root

=

container
.
parsed
(
name
)

changed

=

False

for

style

in

root
.
xpath
(
'//*[local-name()="style"]'
):

if

style
```



```
.
text

and

(
style
.
get
(
'type'
)

or

'text/css'
)

.
lower
()

==

'text/css'
:

sheet

=

container
.
parse_css
(
style
.
text
)

if
```

```
transform_sheet  
(  
sheet  
):
```

```
changed
```

```
=
```

```
True
```

```
style
```

```
.  
text
```

```
=
```

```
force_unicode
```

```
(  
sheet
```

```
.  
cssText  
,
```

```
'utf-8'  
)
```

```
pretty_script_or_style
```

```
(  
container  
,
```

```
style  
)
```

```
for
```

```
elem
```

```
in

root
.
xpath
(
'//*[@style]'
):

text

=

elem
.
get
(
'style'
,
None
)

if

text
:

style

=

container
.
parse_css
(
text
,
is_declaration
```

```
=  
True  
)
```

```
if
```

```
transform_style  
(  
style  
):
```

```
changed
```

```
=  
True
```

```
if
```

```
style  
.  
length
```

```
==
```

```
0  
:
```

```
del
```

```
elem  
.  
attrib  
[  
'style'  
]
```

```
else
```

```
:  
  
elem  
.  
set  
(  
    'style'  
,  
    force_unicode  
(  
    style  
    .  
    getCssText  
(  
    separator  
    =  
    ' '  
    ),  
    'utf-8'  
))  
  
return  
  
changed  
  
def  
  
transform_css  
(  
    container  
,  
    transform_sheet  
    =  
    None  
,
```

```
transform_style
```

```
=
```

```
None
```

```
,
```

```
names
```

```
=
```

```
()):
```

```
if
```

```
not
```

```
names
```

```
:
```

```
types
```

```
=
```

```
OEB_STYLES
```

```
|
```

```
OEB_DOCS
```

```
names
```

```
=
```

```
[]
```

```
for
```

```
name
```

```
,
```

```
mt
```

in

```
iteritems  
(  
    container  
    .  
    mime_map  
):
```

if

mt

in

```
types  
:
```

```
names  
.  
append  
(  
    name  
)
```

doc_changed

=

False

for

name

in

```
names
:

mt

=

container
.
mime_map
[
name
]

if

mt

in

OEB_STYLES
:

sheet

=

container
.
parsed
(
name
)

if

transform_sheet
(
sheet
```



```
);

container
.
dirty
(
name
)

doc_changed

=

True

elif

mt

in

OEB_DOCS
:

if

transform_inline_styles
(
container
,

name
,

transform_sheet
,

transform_style
);
```

```
container
.  
dirty  
(  
name  
)
```

```
doc_changed  
  
=
```

```
True
```

```
return
```

```
doc_changed
```

```
\[docs\]  
def
```

```
filter_css  
(  
container  
,
```

```
properties  
,
```

```
names  
=  
()):
```

```
'''
```

Remove the specified CSS properties from all CSS rules in the book.

:param properties: Set of properties to remove. For example: ``{'font-family', 'color'}``.

:param names: The files from which to remove the properties. Defaults to all HTML and CSS files in the book.

'''

properties

=

normalize_filter_css

(

properties

)

return

transform_css

(

container

,

transform_sheet

=

partial

(

filter_sheet

,

properties

=

properties

),

transform_style

```
=  
partial  
(  
  filter_declaration  
,  
  
  properties  
  =  
  properties  
)  
  
names  
=  
names  
)
```

def

```
_classes_in_selector  
(  
  selector  
,  
  
  classes  
):
```

for

attr

in

```
(  
  'selector'  
,  
  
  'subselector'  
,  
  
  'parsed_tree'  
):
```

```
s
```

```
=
```

```
getattr
```

```
(
```

```
selector
```

```
,
```

```
attr
```

```
,
```

```
None
```

```
)
```

```
if
```

```
s
```

```
is
```

```
not
```

```
None
```

```
:
```

```
_classes_in_selector
```

```
(
```

```
s
```

```
,
```

```
classes
```

```
)
```

```
cn
```

```
=
```

```
getattr
(
selector
,
'class_name'
,
None
)

if
cn
is
not
None
:

classes
.
add
(
cn
)

@lru_cache
(
maxsize
=
4096
)

def
classes_in_selector
```

```
(
text
):

classes

=

set
()

try
:

for
selector

in

parse
(
text
):

_classes_in_selector
(
selector
,
classes
)

except

SelectorSyntaxError
:
```

```
pass
```

```
return
```

```
classes
```

```
def
```

```
classes_in_rule_list  
(  
    css_rules  
):
```

```
    classes
```

```
    =
```

```
    set  
    ()
```

```
for
```

```
    rule
```

```
in
```

```
    css_rules  
    :
```

```
    if
```

```
        rule
```

```
        .
```

```
        type
```

```
    ==
```



```
rule
.
STYLE_RULE
:

classes

|=

classes_in_selector
(
rule
.
selectorText
)

elif

hasattr
(
rule
,
'cssRules'
):

classes

|=

classes_in_rule_list
(
rule
.
cssRules
)

return
```

classes

def

iter_declarations

(
sheet_or_rule
):

if

hasattr

(
sheet_or_rule
,

'cssRules'

):

for

rule

in

sheet_or_rule

.
cssRules
:

yield from

iter_declarations

(
rule
)

elif

```
hasattr  
(  
    sheet_or_rule  
,  
    'style'  
):
```

yield

```
sheet_or_rule  
.  
style
```

elif

```
isinstance  
(  
    sheet_or_rule  
,  
    CSSStyleDeclaration  
):
```

yield

```
sheet_or_rule
```

def

```
remove_property_value  
(  
    prop  
,
```

```
predicate
):
```

```
""" Remove the Values that match the predicate from this property. If all
values of the property would be removed, the property is removed from its
parent instead. Note that this means the property must have a parent (a
CSSStyleDeclaration). """
```

```
removed_vals
```

```
=
```

```
list
(
filter
(
predicate
,
prop
.
propertyValue
))
```

```
if
```

```
len
(
removed_vals
)
```

```
==
```

```
len
(
prop
```

```
.
```

```
propertyValue  
):
```

```
prop
```

```
.
```

```
parent
```

```
.
```

```
removeProperty
```

```
(
```

```
prop
```

```
.
```

```
name
```

```
)
```

```
else
```

```
:
```

```
x
```

```
=
```

```
css_text
```

```
(
```

```
prop
```

```
.
```

```
propertyValue
```

```
)
```

```
for
```

```
v
```

```
in
```

```
removed_vals
```

```
:
```

```
x
```

```
=  
  
x  
.  
replace  
(  
css_text  
(  
v  
),  
"  
)  
.  
strip  
(  
  
prop  
.  
propertyValue  
.  
cssText  
  
=  
  
x  
  
return  
  
bool  
(  
removed_vals  
)  
  
RULE_PRIORITIES  
  
=
```

```
{
  t
  :
  i

  for

  i
  ,

  t

  in

  enumerate
  ((
  CSSRule
  .
  COMMENT
  ,

  CSSRule
  .
  CHARSET_RULE
  ,

  CSSRule
  .
  IMPORT_RULE
  ,

  CSSRule
  .
  NAMESPACE_RULE
  ))}
```

def

```
sort_sheet
(
  container
```

```
,
sheet_or_text
):

''' Sort the rules in a stylesheet. Note that in the general case this can
    change the effective styles, but for most common sheets, it should be safe.
'''

sheet
=
container
.
parse_css
(
sheet_or_text
)

if
isinstance
(
sheet_or_text
,
str
)
else
sheet_or_text

def
text_sort_key
```



```
(
x
):

return

numeric_sort_key
(
str
(
x

or

"
))

def

selector_sort_key
(
x
):

return

(
x
.
specificity
,

text_sort_key
(
x
.
selectorText
))
```

```
def
rule_sort_key
(
rule
):

primary

=

RULE_PRIORITIES
.
get
(
rule
.
type
,

len
(
RULE_PRIORITIES
))

secondary

=

text_sort_key
(
getattr
(
rule
,

'atkeyword'
,
```

```
"
)

or

"
)

tertiary

=

None

if

rule
.
type

==

CSSRule
.
STYLE_RULE
:

primary

+=

1

selectors

=

sorted
(
```

```
rule
.  
selectorList  
,  
  
key  
=  
selector_sort_key  
)
```

```
tertiary  
  
=  
  
selector_sort_key  
(  
selectors  
[  
0  
)
```

```
rule  
.  
selectorText  
  
=  
  
, '  
, '  
.  
join  
(  
s  
.  
selectorText  
  
for  
  
s  
  
in
```

```
selectors
)

elif

rule
.
type

==

CSSRule
.
FONT_FACE_RULE
:

try
:

tertiary

=

text_sort_key
(
rule
.
style
.
getPropertyValue
(
'font-family'
))

except

Exception
:
```

pass

return

primary

,

secondary

,

tertiary

sheet

.

cssRules

.

sort

(

key

=

rule_sort_key

)

return

sheet

def

add_stylesheet_links

(

container

,

name

,

```
text
):

root

=

container
.
parse_html
(
text
,
name
)

head

=

root
.
xpath
(
'//*[local-name() = "head"]'
)

if

not

head
:

return
```

```
head
=
head
[
0
]

sheets
=
tuple
(
container
.
manifest_items_of_type
(
lambda

mt
:

mt

in

OEB_STYLES
))

if

not

sheets
:

return
```


for

sname

in

sheets

:

link

=

head

.

makeelement

(

XHTML

(

'link'

),

type

=

'text/css'

,

rel

=

'stylesheet'

,

href

=

container

.

name_to_href

(

sname

,

```
name
))

head
.
append
(
link
)

pretty_xml_tree
(
head
)

return

serialize
(
root
,
'text/html'
)

def

rename_class_in_rule_list
(
css_rules
,
old_name
,
new_name
):
```

this regex will not match class names inside attribute value selectors

and it will match id selectors that contain .old_name but its the best

that can be done without implementing a full parser for CSS selectors

pat

=

re

.

compile

(

rf

'(?<=\..)

{

re

.

escape

(

old_name

)

}

(?:\W|\$)'

)

def

repl

(

m

):

return

```
m
.
group
()
.
replace
(
old_name
,
new_name
)
```

```
changed
```

```
=
```

```
False
```

```
for
```

```
rule
```

```
in
```

```
css_rules
:
```

```
if
```

```
rule
```

```
.
type
```

```
==
```

```
rule
```

```
.
STYLE_RULE
```

```
:  
  
old  
  
=  
  
rule  
.  
selectorText
```

```
q  
  
=  
  
pat  
.  
sub  
(  
repl  
,  
old  
)
```

```
if
```

```
q  
  
!=
```

```
old  
:
```

```
changed
```

```
=
```

```
True
```

```
rule
    .
    selectorText
```

```
=
```

```
q
```

```
elif
```

```
hasattr
(
    rule
    ,
    'cssRules'
):
```

```
if
```

```
rename_class_in_rule_list
(
    rule
    .
    cssRules
    ,
    old_name
    ,
    new_name
):
```

```
changed
```

```
=
```

```
True
```

return

changed

def

rename_class_in_doc

(

container

,

root

,

old_name

,

new_name

):

changed

=

False

pat

=

re

.

compile

(

rf

'(?:^|\W)

{

re

```
.
escape
(
  old_name
)
}
(?:\W|$)'
)
```

def

```
repl
(
  m
):
```

return

```
m
.
group
()
.
replace
(
  old_name
,
  new_name
)
```

for

elem

in

root


```
.  
xpath  
(  
  '//*[@class]'  
)  
):
```

old

=

elem

```
.  
get  
(  
  'class'  
)
```

if

old
:

new

=

pat

```
.  
sub  
(  
  repl  
,
```

old
)

if

new

```
!=
```

```
old
```

```
:
```

```
changed
```

```
=
```

```
True
```

```
elem
```

```
.
```

```
set
```

```
(
```

```
'class'
```

```
,
```

```
new
```

```
)
```

```
for
```

```
style
```

```
in
```

```
root
```

```
.
```

```
xpath
```

```
(
```

```
'//*[local-name()="style"]'
```

```
):
```

```
if
```

```
style
```

```
.
```

```
get
(
  'type'
,
  'text/css'
)

==

'text/css'

and

style
.
text
:

sheet

=

container
.
parse_css
(
  style
.
  text
)

if

rename_class_in_rule_list
(
  sheet
.
  cssRules
,
```

```
old_name
',

new_name
):

changed

=

True

style
.
text

=

force_unicode
(
sheet
.
cssText
,

'utf-8'
)

return

changed

def

rename_class
(
container
,
```

```
old_name  
,  
new_name  
):
```

```
changed
```

```
=
```

```
False
```

```
if
```

```
not
```

```
old_name
```

```
or
```

```
old_name
```

```
==
```

```
new_name  
:
```

```
return
```

```
changed
```

```
for
```

```
sheet_name
```

```
in
```

```
container
```

```
.
manifest_items_of_type
(
  lambda

  mt
  :

  mt

  in

  OEB_STYLES
):

sheet

=

container
.
parsed
(
  sheet_name
)

if

rename_class_in_rule_list
(
  sheet
.
cssRules
,

  old_name
,

  new_name
):
```

```
container
.  
dirty  
(  
sheet_name  
)
```

```
changed
```

```
=
```

```
True
```

```
for
```

```
doc_name
```

```
in
```

```
container  
.  
manifest_items_of_type  
(  
lambda
```

```
mt  
:
```

```
mt
```

```
in
```

```
OEB_DOCS  
):
```

```
doc
```

```
=
```

```
container
.  
parsed  
(  
doc_name  
)
```

if

```
rename_class_in_doc  
(  
container  
,  
  
doc  
,  
  
old_name  
,  
  
new_name  
):
```

```
container  
.  
dirty  
(  
doc_name  
)
```

```
changed
```

```
=
```

True

return

```
changed
```


Source code for calibre.ebooks.oeb.polish.fonts

```
#!/usr/bin/env python

__license__
=
'GPL v3'

__copyright__
=
'2014, Kovid Goyal <kovid at kovidgoyal.net>'

from
calibre.ebooks.oeb.base

import
css_text

from
calibre.ebooks.oeb.polish.container

import
OEB_STYLES
,
```

```
OEB_DOCS
```

```
from
```

```
calibre.ebooks.oeb.normalize_css
```

```
import
```

```
normalize_font
```

```
from
```

```
tinycss.fonts3
```

```
import
```

```
parse_font_family
```

```
,
```

```
parse_font
```

```
,
```

```
serialize_font_family
```

```
,
```

```
serialize_font
```

```
from
```

```
polyglot.builtins
```

```
import
```

```
iteritems
```

```
def
```

```
unquote
```

```
(
```

```
x
```

```
):
```

```
if
```

```
x
```

```
and
```

```
len
```

```
(
```

```
x
```

```
)
```

```
>
```

```
1
```

```
and
```

```
x
```

```
[
```

```
0
```

```
]
```

```
==
```

```
x
```

```
[
```

```
-
```

```
1
```

```
]
```

```
and
```

```
x
```

```
[
```

```
0
```

```
]
```

```
in
```

```
(
```

```
"""
,
"""
):

x

=

x
[
1
:
-
1
]

return

x

def
font_family_data_from_declaration
(
style
,
families
):

font_families

=

[]
```

```
f
=
style
.
getProperty
(
'font'
)

if
f
is
not
None
:
f
=
normalize_font
(
f
.
propertyValue
,
font_family_as_list
=
True
)
.
get
(
```

```
'font-family'
```

```
,
```

```
None
```

```
)
```

```
if
```

```
f
```

```
is
```

```
not
```

```
None
```

```
:
```

```
font_families
```

```
=
```

```
[
```

```
unquote
```

```
(
```

```
x
```

```
)
```

```
for
```

```
x
```

```
in
```

```
f
```

```
]
```

```
f
```

```
=
```

```
style
    .
    getProperty
    (
        'font-family'
    )

    if

    f

    is

    not

    None
    :

    font_families

    =

    parse_font_family
    (
        css_text
        (
            f
            .
            propertyValue
        ))

    for

    f

    in

    font_families
    :
```



```
families
```

```
[  
f  
]
```

```
=
```

```
families
```

```
.
```

```
get
```

```
(
```

```
f
```

```
,
```

```
False
```

```
)
```

```
def
```

```
font_family_data_from_sheet
```

```
(
```

```
sheet
```

```
,
```

```
families
```

```
):
```

```
for
```

```
rule
```

```
in
```

```
sheet
```

```
.
```

```
cssRules
```

```
:
```

if

rule

.

type

==

rule

.

STYLE_RULE

:

font_family_data_from_declaration

(

rule

.

style

,

families

)

elif

rule

.

type

==

rule

.

FONT_FACE_RULE

:

ff

=

```
rule
.
style
.
getProperty
(
'font-family'
)

if

ff

is

not

None
:

for

f

in

parse_font_family
(
css_text
(
ff
.
propertyValue
)):

families
[
f
]
```

```
=
```

```
True
```

```
def
```

```
font_family_data
```

```
(
```

```
container
```

```
):
```

```
families
```

```
=
```

```
{}
```

```
for
```

```
name
```

```
,
```

```
mt
```

```
in
```

```
iteritems
```

```
(
```

```
container
```

```
.
```

```
mime_map
```

```
):
```

```
if
```

```
mt
```

in

OEB_STYLES

:

sheet

=

container

.

parsed

(

name

)

font_family_data_from_sheet

(

sheet

,

families

)

elif

mt

in

OEB_DOCS

:

root

=

container

.

```
parsed
(
  name
)

for

style

in

root
.
xpath
(
  '//*[local-name() = "style"]'
):

if

style
.
text

and

style
.
get
(
  'type'
,
  'text/css'
)
.
lower
()

==
```

```
'text/css'
:

sheet

=

container
.
parse_css
(
style
.
text
)

font_family_data_from_sheet
(
sheet
,

families
)

for

style

in

root
.
xpath
(
'//*[@style'
):

if
```

```
style
:

style

=

container
.
parse_css
(
style
,

is_declaration
=
True
)

font_family_data_from_declaration
(
style
,

families
)

return

families


def

change_font_in_declaration
(
style
,
```



```
old_name
',
new_name
=
None
):
```

```
changed
=
False
```

```
ff
=
style
.
getProperty
(
'font-family'
)
```

```
if
```

```
ff
```

```
is
```

```
not
```

```
None
:
```

```
fams
```

```
=
```

```
parse_font_family  
(  
  css_text  
(  
    ff  
    .  
    propertyValue  
  ))
```

```
nfams
```

```
=
```

```
list  
(  
  filter  
(  
    None  
    ,  
    [  
      new_name
```

```
if
```

```
x
```

```
==
```

```
old_name
```

```
else
```

```
x
```

```
for
```

```
x
```

```
in
```

```
fams
]))

if

fams

!=

nfams
:

if

nfams
:

ff
.
propertyValue
.
cssText

=

serialize_font_family
(
nfams
)

else
:

style
.
removeProperty
(
ff
```

```
.
name
)

changed

=

True

ff

=

style
.
getProperty
(
'font'
)

if

ff

is

not

None
:

props

=

parse_font
(
css_text
```

```
(  
ff  
.  
propertyValue  
))
```

fams

=

props

```
.  
get  
(  
'font-family'  
)
```

or

[]

nfams

=

```
list  
(  
filter  
(  
None
```

,

```
[  
new_name
```

if

x

==

```
old_name
```

```
else
```

```
x
```

```
for
```

```
x
```

```
in
```

```
fams
```

```
]))
```

```
if
```

```
fams
```

```
!=
```

```
nfams
```

```
:
```

```
props
```

```
[
```

```
'font-family'
```

```
]
```

```
=
```

```
nfams
```

```
if
```

```
nfams
```

```
:
```

```
ff
```

```
.  
propertyValue  
.  
cssText  
  
=  
  
serialize_font  
(  
  props  
)  
  
else  
:  
  
  style  
  .  
  removeProperty  
  (  
    ff  
    .  
    name  
  )  
  
  changed  
  
  =  
  
  True  
  
  return  
  changed  
  
  def  
  remove_embedded_font
```

```
(  
    container  
,  
    sheet  
,  
    rule  
,  
    sheet_name  
):
```

```
    src
```

```
    =
```

```
    getattr
```

```
    (  
        rule
```

```
        .  
        style
```

```
        .  
        getProperty
```

```
        (  
            'src'  
        ),
```

```
        'value'
```

```
    ,
```

```
    None
```

```
)
```

```
if
```

```
    src
```

```
is
```

```
not
```


None

:

if

src

.

startswith

(

'url('

):

src

=

src

[

4

:

-

1

]

sheet

.

cssRules

.

remove

(

rule

)

if

src

:

```
src
```

```
=
```

```
unquote
```

```
(
```

```
src
```

```
)
```

```
name
```

```
=
```

```
container
```

```
.
```

```
href_to_name
```

```
(
```

```
src
```

```
,
```

```
sheet_name
```

```
)
```

```
if
```

```
container
```

```
.
```

```
has_name
```

```
(
```

```
name
```

```
):
```

```
container
```

```
.
```

```
remove_item
```

```
(
```

```
name
```

```
)
```

def

change_font_in_sheet

(

container

,

sheet

,

old_name

,

new_name

,

sheet_name

):

changed

=

False

removals

=

[]

for

rule

in

sheet

```
.
cssRules
:

if

rule
.
type

==

rule
.
STYLE_RULE
:

changed

|=

change_font_in_declaration
(
rule
.
style
,

old_name
,

new_name
)

elif

rule
.
type
```

```
==  
  
rule  
.  
FONT_FACE_RULE  
:
```

```
ff
```

```
=
```

```
rule  
.  
style  
.  
getProperty  
(  
  'font-family'  
)
```

```
if
```

```
ff
```

```
is
```

```
not
```

```
None
```

```
:
```

```
families
```

```
=
```

```
{  
x
```

```
for
```

x

in

```
parse_font_family  
(  
  css_text  
(  
    ff  
    .  
    propertyValue  
  ))}
```

if

old_name

in

```
families  
:
```

changed

=

True

removals

```
.  
append  
(  
  rule  
)
```

for

rule

in

reversed

(
removals
):

remove_embedded_font

(
container
,

sheet

,

rule

,

sheet_name

)

return

changed

[\[docs\]](#)

def

change_font

(
container
,

old_name

,

new_name

```
=
None
):

'''
    Change a font family from old_name to new_name. Changes all occurrences of
    the font family in stylesheets, style tags and style attributes.

    If the old_name refers to an embedded font, it is removed. You can set
    new_name to None to remove the font family instead of changing it.
'''

changed

=

False

for
name
,
mt
in
tuple
(
iteritems
(
container
.
mime_map
)):

```



```
if
```

```
mt
```

```
in
```

```
OEB_STYLES
```

```
:
```

```
sheet
```

```
=
```

```
container
```

```
.
```

```
parsed
```

```
(
```

```
name
```

```
)
```

```
if
```

```
change_font_in_sheet
```

```
(
```

```
container
```

```
,
```

```
sheet
```

```
,
```

```
old_name
```

```
,
```

```
new_name
```

```
,
```

```
name
```

```
):
```

container

.

dirty

(

name

)

changed

=

True

elif

mt

in

OEB_DOCS

:

root

=

container

.

parsed

(

name

)

for

style

in

```
root
.  
xpath  
(  
  '//*[local-name() = "style"]'  
):
```

if

```
style  
.  
text
```

and

```
style  
.  
get  
(  
  'type'  
,  
  
  'text/css'  
)  
.  
lower  
(
```

```
==
```

```
'text/css'  
:
```

```
sheet
```

```
=
```

```
container
```

```
.  
parse_css  
(
```

```
style
```

```
.
```

```
text
```

```
)
```

```
if
```

```
change_font_in_sheet
```

```
(
```

```
container
```

```
,
```

```
sheet
```

```
,
```

```
old_name
```

```
,
```

```
new_name
```

```
,
```

```
name
```

```
):
```

```
container
```

```
.
```

```
dirty
```

```
(
```

```
name
```

```
)
```

```
changed
```

```
=
```

```
True
```

```
for
```

```
elem
```

```
in
```

```
root
```

```
.
```

```
xpath
```

```
(
```

```
'//*[@style]'
```

```
):
```

```
style
```

```
=
```

```
elem
```

```
.
```

```
get
```

```
(
```

```
'style'
```

```
,
```

```
"
```

```
)
```

```
if
```

```
style
```

```
:
```

```
style
```

```
=
```

```
container
```

```
.
```

```
parse_css
```

```
(
```

```
style
```

```
,
```

```
is_declaration
=
True
)

if

change_font_in_declaration
(
style
,

old_name
,

new_name
):

style

=

css_text
(
style
)
.
strip
()
.
rstrip
(
','
)
.
strip
()

if
```

```
style
:
```

```
elem
```

```
.
set
(
'style'
,
style
)
```

```
else
:
```

```
del
```

```
elem
.
attrib
[
'style'
]
```

```
container
```

```
.
dirty
(
name
)
```

```
changed
```

```
=
```

```
True
```

return

changed

Source code for calibre.ebooks.oeb.polish.jacket

```
#!/usr/bin/env python
```

```
__license__
```

```
=
```

```
'GPL v3'
```

```
__copyright__
```

```
=
```

```
'2013, Kovid Goyal <kovid at kovidgoyal.net>'
```

```
__docformat__
```

```
=
```

```
'restructuredtext en'
```

```
from
```

```
calibre.customize.ui
```

```
import
```

```
output_profiles
```

```
from
```

```
calibre.ebooks.conversion.config
```

```
import
```

```
load_defaults
```

```
from
```

```
calibre.ebooks.oeb.base
```

```
import
```

```
XPath
```

```
,
```

```
OPF
```

```
from
```

```
calibre.ebooks.oeb.polish.cover
```

```
import
```

```
find_cover_page
```

```
from
```

```
calibre.ebooks.oeb.transforms.jacket
```

```
import
```

```
render_jacket
```

```
as
```

```
render
```

```
,
```

```
referenced_images
```

```
def
```

```
render_jacket
(
container
,

jacket
):

mi

=

container
.
mi

ps

=

load_defaults
(
'page_setup'
)

op

=

ps
.
get
(
'output_profile'
,

'default'
)
```

opmap

=

{

x

.

short_name

:

x

for

x

in

output_profiles

()}

output_profile

=

opmap

.

get

(

op

,

opmap

[

'default'

])

root

=

```
render
(
  mi
  ,
  output_profile
)

for

img
,
path

in

referenced_images
(
  root
):

  container
  .
  log
  (
    'Embedding referenced image:
    %s
    into jacket'

    %

    path
  )

  ext

  =

  path
```

```
.
rpartition
(
    ':'
)[
    -
    1
]

jacket_item

=

container
.
generate_item
(
    'jacket_image.'
+
    ext
,

    id_prefix
    =
    'jacket_img'
)

name

=

container
.
href_to_name
(
    jacket_item
.
get
(
    'href'
),
```

```
container
.  
opf_name  
)
```

with

```
open  
(  
path  
,  
  
'rb'  
)
```

as

```
f  
:
```

```
container  
.  
parsed_cache  
[  
name  
]
```

```
=
```

```
f  
.  
read  
()
```

```
container  
.  
commit_item  
(  
name
```

```
)  
  
href  
  
=  
  
container  
.  
name_to_href  
(  
name  
,  
  
jacket  
)  
  
img  
.  
set  
(  
'src'  
,  
  
href  
)  
  
return  
  
root  
  
def  
  
is_legacy_jacket  
(  
root  
):
```


return

len

(
root

.
xpath
(

'//*[starts-with(@class,"calibrerescale") and (local-name()="h1" or local-name()="h2")]'
))

>

0

def

is_current_jacket

(
root
):

return

len

(
XPath
(

'//h:meta[@name="calibre-content" and @content="jacket"]'
)(
root
))

>

0

def

find_existing_jacket

(

container

):

for

item

in

container

.

spine_items

:

name

=

container

.

abspath_to_name

(

item

)

if

container

.

book_type

==

```
'azw3'  
:  
  
root  
  
=  
  
container  
.  
parsed  
(  
name  
)  
  
if  
  
is_current_jacket  
(  
root  
):  
  
return  
  
name  
  
else  
:  
  
if  
  
name  
.  
rpartition  
(  
'/'  
)[  
-  
1
```

```
]
.
startswith
(
'jacket'
)

and

name
.
endswith
(
'.xhtml'
):

root

=

container
.
parsed
(
name
)

if

is_current_jacket
(
root
)

or

is_legacy_jacket
(
root
):
```

```
return
```

```
name
```

```
def
```

```
replace_jacket
```

```
(
```

```
container
```

```
,
```

```
name
```

```
):
```

```
root
```

```
=
```

```
render_jacket
```

```
(
```

```
container
```

```
,
```

```
name
```

```
)
```

```
container
```

```
.
```

```
parsed_cache
```

```
[
```

```
name
```

```
]
```

```
=
```

```
root
```

```
container
.
dirty
(
name
)

\[docs\]
def

remove_jacket
(
container
):

' Remove an existing jacket, if any. Returns False if no existing jacket was found. '

name

=

find_existing_jacket
(
container
)

if

name

is

not

None
:
```

```
remove_jacket_images  
(  
    container  
,  
  
    name  
)
```

```
container  
.  
remove_item  
(  
    name  
)
```

return

True

return

False

def

```
remove_jacket_images  
(  
    container  
,  
  
    name  
):
```

root

=

container

.

parsed_cache

[

name

]

for

img

in

root

.

xpath

(

'//*[local-name() = "img" and @src]'

):

iname

=

container

.

href_to_name

(

img

.

get

(

'src'

),

name

)

if


```
container
```

```
.
```

```
has_name
```

```
(
```

```
iname
```

```
):
```

```
container
```

```
.
```

```
remove_item
```

```
(
```

```
iname
```

```
)
```

```
\[docs\]
```

```
def
```

```
add_or_replace_jacket
```

```
(
```

```
container
```

```
):
```

```
""" Either create a new jacket from the book's metadata or replace an
```

```
existing jacket. Returns True if an existing jacket was replaced. """
```

```
name
```

```
=
```

```
find_existing_jacket
```

```
(
```

```
container
```

```
)
```

```
found
```

```
=
```

```
True
```

```
if
```

```
name
```

```
is
```

```
None
```

```
:
```

```
jacket_item
```

```
=
```

```
container
```

```
.
```

```
generate_item
```

```
(
```

```
'jacket.xhtml'
```

```
,
```

```
id_prefix
```

```
=
```

```
'jacket'
```

```
)
```

```
name
```

```
=
```

```
container
```

```
.
```

```
href_to_name
```

```
(
```

```
jacket_item
```

```
.
```

```
get
```

```
(
```

```
'href'
```

```
),
```

```
container
```

```
.
```

```
opf_name
```

```
)
```

```
found
```

```
=
```

```
False
```

```
if
```

```
found
```

```
:
```

```
remove_jacket_images
```

```
(
```

```
container
```

```
,
```

```
name
```

```
)
```

```
replace_jacket
```

```
(
```

```
container
```

```
,
```

```
name
```

```
)
```

```
if
not
found
:

# Insert new jacket into spine

index
=
0

sp
=
container
.
abspath_to_name
(
next
(
container
.
spine_items
))

if
sp
==
find_cover_page
```

```
(
  container
):

index

=

1

itemref

=

container
.
opf
.
makeelement
(
  OPF
  (
    'itemref'
  ),

  idref
  =
  jacket_item
  .
  get
  (
    'id'
  ))

container
.
insert_into_xml
(
  container
```

```
.  
opf_xpath  
(  
    '//opf:spine'  
)[  
    0  
],  
  
itemref  
,  
  
index  
=  
index  
)  
  
return  
  
found
```

Source code for calibre.ebooks.oeb.polish.pretty

```
#!/usr/bin/env python
```

```
__license__
```

```
=
```

```
'GPL v3'
```

```
__copyright__
```

```
=
```

```
'2013, Kovid Goyal <kovid at kovidgoyal.net>'
```

```
import
```

```
textwrap
```

```
from
```

```
polyglot.builtins
```

```
import
```

```
iteritems
```

```
# from lxml.etree import Element
```

```
from
```

```
calibre
```

```
import
```

```
force_unicode
```

```
from
```

```
calibre.ebooks.oeb.base
```

```
import
```

```
(
```

```
serialize
```

```
,
```

```
OEB_DOCS
```

```
,
```

```
barename
```

```
,
```

```
OEB_STYLES
```

```
,
```

```
XPNSMAP
```

```
,
```

```
XHTML
```

```
,
```

```
SVG
```

```
)
```

```
from
```

```
calibre.ebooks.oeb.polish.container
```

```
import
```



```
OPF_NAMESPACES
```

```
from
```

```
calibre.ebooks.oeb.polish.utils
```

```
import
```

```
guess_type
```

```
from
```

```
calibre.utils.icu
```

```
import
```

```
sort_key
```

```
def
```

```
isspace
```

```
(  
x  
):
```

```
return
```

```
not
```

```
x
```

```
.
```

```
strip
```

```
(  
'
```

```
\u0009\u000a\u000c\u000d\u0020
```

```
'
```

```
)
```

def

pretty_xml_tree

(

elem

,

level

=

0

,

indent

=

' '

):

""" XML beautifier, assumes that elements that have children do not have textual content. Also assumes that there is no text immediately after closing tags. These are true for opf/ncx and container.xml files. If either of the assumptions are violated, there should be no data loss, but pretty printing won't produce optimal results."""

if

(

not

elem

.

text

and

len

(

```
elem  
)
```

```
>
```

```
0  
)
```

```
or
```

```
(  
elem
```

```
.  
text
```

```
and
```

```
isspace
```

```
(  
elem
```

```
.  
text  
)):
```

```
elem
```

```
.  
text
```

```
=
```

```
'  
\n  
'
```

```
+
```

```
(  
indent
```

```
*
```

```
(
```

```
level
+
1
))

for

i
,

child

in

enumerate
(
elem
):

pretty_xml_tree
(
child
,

level
=
level
+
1
,

indent
=
indent
)

if

not
```

child

.

tail

or

isspace

(

child

.

tail

):

|

=

level

+

1

if

i

==

len

(

elem

)

-

1

:

|

```
-=
```

```
1
```

```
child
```

```
.
```

```
tail
```

```
=
```

```
,
```

```
\n
```

```
,
```

```
+
```

```
(
```

```
indent
```

```
*
```

```
|
```

```
)
```

```
def
```

```
pretty_opf
```

```
(
```

```
root
```

```
):
```

```
# Put all dc: tags first starting with title and author. Preserve order for
```

```
# the rest.
```

```
def
```

```
dckey
(
x
):

return

{
'title'
:
0
,

'creator'
:
1
}
.
get
(
barename
(
x
.
tag
),
2
)

for

metadata

in

root
.
xpath
(
```

```
'//opf:metadata'
',
namespaces
=
OPF_NAMESPACES
):

dc_tags

=

metadata
.
xpath
(
'./*[namespace-uri()='
%s
"]'

%

OPF_NAMESPACES
[
'dc'
])

dc_tags
.
sort
(
key
=
dckey
)

for

x
```


in

reversed

```
(  
dc_tags  
):
```

metadata

```
.  
insert  
(  
0  
,  
  
x  
)
```

Group items in the manifest

spine_ids

=

root

```
.  
xpath  
(  
'//opf:spine/opf:itemref/@idref'  
,
```

namespaces

```
=  
OPF_NAMESPACES  
)
```

spine_ids

=

```
{
x
:
i

for

i
,

x

in

enumerate
(
spine_ids
)}


def

manifest_key
(
x
):

mt

=

x
.
get
(
'media-type'
,
"
)
)
```

href

=

x

.

get

(

'href'

,

"

)

ext

=

href

.

rpartition

(

'

['

-

1

]

.

lower

()

cat

=

1000

if

mt

in

OEB_DOCS

:

cat

=

0

elif

mt

==

guess_type

(

'a.ncx'

):

cat

=

1

elif

mt

in

OEB_STYLES

:

cat

=

2

elif

mt

.

startswith

(

'image/'

):

cat

=

3

elif

ext

in

{

'otf'

,

'ttf'

,

'woff'

};

cat

=

4

elif

mt

.

startswith

(

'audio/'

):

cat

=

5

elif

mt

.

startswith

(

'video/'

):

cat

=

6

if

```
cat
```

```
==
```

```
0
```

```
:
```

```
i
```

```
=
```

```
spine_ids
```

```
.
```

```
get
```

```
(
```

```
x
```

```
.
```

```
get
```

```
(
```

```
'id'
```

```
,
```

```
None
```

```
),
```

```
1000000000
```

```
)
```

```
else
```

```
:
```

```
i
```

```
=
```

```
sort_key
```

```
(
```

```
href
```

```
)
```

```
return
```

```
(  
cat  
,  
i  
)
```

```
for
```

```
manifest
```

```
in
```

```
root  
.  
xpath  
(  
'//opf:manifest'  
,  
namespaces  
=  
OPF_NAMESPACES  
):
```

```
try  
:
```

```
children
```

```
=
```

```
sorted  
(  
manifest
```



```
,  
  
key  
=  
manifest_key  
)  
  
except  
  
AttributeError  
:  
  
continue  
  
# There are comments so dont sort since that would mess up the comments  
  
for  
  
x  
  
in  
  
reversed  
(  
children  
):  
  
manifest  
.  
insert  
(  
0  
,  
  
x  
)
```

```
SVG_TAG
```

```
=
```

```
SVG
```

```
(
```

```
'svg'
```

```
)
```

```
BLOCK_TAGS
```

```
=
```

```
frozenset
```

```
(
```

```
map
```

```
(
```

```
XHTML
```

```
,
```

```
(
```

```
'address'
```

```
,
```

```
'article'
```

```
,
```

```
'aside'
```

```
,
```

```
'audio'
```

```
,
```

```
'blockquote'
```

```
,
```

```
'body'
```

```
,
```

```
'canvas'
```

```
,
```

'col'

,

'colgroup'

,

'dd'

,

'div'

,

'dl'

,

'dt'

,

'fieldset'

,

'figcaption'

,

'figure'

,

'footer'

,

'form'

,

'h1'

,

'h2'

,

'h3'

```
,  
  
'h4'  
,  
  
'h5'  
,  
  
'h6'  
,  
  
'header'  
,  
  
'hgroup'  
,  
  
'hr'  
,  
  
'li'  
,  
  
'noscript'  
,  
  
'ol'  
,  
  
'output'  
,  
  
'p'  
,  
  
'pre'  
,  
  
'script'  
,  
  
'section'
```

```
,  
  
'style'  
,  
  
'table'  
,  
  
'tbody'  
,  
  
'td'  
,  
  
'tfoot'  
,  
  
'th'  
,  
  
'thead'  
,  
  
'tr'  
,  
  
'ul'  
,  
  
'video'  
,  
  
'img'  
)))  
  
|  
  
{  
SVG_TAG  
}
```

def

isblock

(
x
):

if

callable

(
x
.
tag
)

or

not

x
.
tag
:

return

True

if

x
.
tag

in

BLOCK_TAGS

:

```
return
```

```
True
```

```
return
```

```
False
```

```
def
```

```
has_only_blocks
```

```
(  
x  
):
```

```
if
```

```
hasattr
```

```
(  
x  
.  
tag  
,
```

```
'split'  
)
```

```
and
```

```
len
```

```
(  
x  
)
```

```
==
```

```
0
```

```
:
```

```
# Tag with no children,
```

```
return
```

```
False
```

```
if
```

```
x
```

```
.
```

```
text
```

```
and
```

```
not
```

```
isspace
```

```
(
```

```
x
```

```
.
```

```
text
```

```
):
```

```
return
```

```
False
```

```
for
```

```
child
```

```
in
```

```
x
```

```
:
```


if

not

isblock

(

child

)

or

(

child

.

tail

and

not

isspace

(

child

.

tail

)):

return

False

return

True

def

indent_for_tag

```
(
x
):

prev

=

x
.
getprevious
()

x

=

x
.
getparent
()
.
text

if

prev

is

None

else

prev
.
tail

if
```

not

x

:

return

"

s

=

x

.

rpartition

(

'

\n

'

)[

-

1

]

return

s

if

isspace

(

s

)

else

"

def

set_indent

(
elem
,

attr
,

indent
):

x

=

getattr

(
elem
,

attr
)

if

not

x
:

x

=

indent

```
else
```

```
:
```

```
lines
```

```
=
```

```
x
```

```
.
```

```
splitlines
```

```
()
```

```
if
```

```
isspace
```

```
(
```

```
lines
```

```
[
```

```
-
```

```
1
```

```
]):
```

```
lines
```

```
[
```

```
-
```

```
1
```

```
]
```

```
=
```

```
indent
```

```
else
```

```
:
```

```
lines
```

```
.
```

```
append
(
indent
)

x

=

'
\n
'

.
join
(
lines
)

setattr
(
elem
,
attr
,
x
)

def
pretty_block
(
parent
,
level
=
1
```

```

',

indent
=
    ,
):

""" Surround block tags with blank lines and recurse into child block tags
    that contain only other block tags """

if
not
parent
    .
text
or
isspace
(
    parent
    .
text
):

parent
    .
text

=

"

nn

=

```

```
'  
  \n  
'  
  
if  
  
  hasattr  
  (  
    parent  
    .  
    tag  
    ,  
  
    'strip'  
  )  
  
and  
  
  barename  
  (  
    parent  
    .  
    tag  
  )  
  
in  
  
  {  
    'tr'  
    ,  
  
    'td'  
    ,  
  
    'th'  
  }  
  
else  
  
  '  
  \n\n'  
'
```


parent

.

text

=

parent

.

text

+

nn

+

(

indent

*

level

)

for

i

,

child

in

enumerate

(

parent

):

if

```
isblock
(  
child  
)
```

and

```
has_only_blocks  
(  
child  
):
```

```
pretty_block  
(  
child  
,
```

```
level  
=  
level  
+  
1  
,
```

```
indent  
=  
indent  
)
```

elif

```
child  
.  
tag
```

```
==
```

```
SVG_TAG  
:
```

```
pretty_xml_tree
```

```
(
```

```
child
```

```
,
```

```
level
```

```
=
```

```
level
```

```
,
```

```
indent
```

```
=
```

```
indent
```

```
)
```

```
|
```

```
=
```

```
level
```

```
if
```

```
i
```

```
==
```

```
len
```

```
(
```

```
parent
```

```
)
```

```
-
```

```
1
```

```
:
```

```
|
```

-=

1

if

not

child

.

tail

or

isspace

(

child

.

tail

):

child

.

tail

=

"

child

.

tail

=

child

.

tail

+

nn

+

(
indent

*

|
)

def

pretty_script_or_style

(
container
,

child
):

if

child
.
text
:

indent

=

indent_for_tag

(
child
)

```
if
    child
    .
    tag
    .
    endswith
    (
    'style'
    ):

    child
    .
    text

    =

    force_unicode
    (
    pretty_css
    (
    container
    ,
    "
    ,
    child
    .
    text
    ),
    'utf-8'
    )

    child
    .
    text

    =
```

```
textwrap
```

```
.
```

```
dedent
```

```
(
```

```
child
```

```
.
```

```
text
```

```
)
```

```
child
```

```
.
```

```
text
```

```
=
```

```
,
```

```
\n
```

```
,
```

```
+
```

```
,
```

```
\n
```

```
,
```

```
.
```

```
join
```

```
((
```

```
indent
```

```
+
```

```
x
```

```
)
```

```
if
```

```
x
```

```
else
```

```
''
```

```
for
```

```
x
```

```
in
```

```
child
```

```
.
```

```
text
```

```
.
```

```
splitlines
```

```
())
```

```
set_indent
```

```
(
```

```
child
```

```
,
```

```
'text'
```

```
,
```

```
indent
```

```
)
```

```
def
```

```
pretty_html_tree
```

```
(
```

```
container
```

```
,
```

```
root
```

```
):
```

```
root
```

```
.
```

```
text
```



```
=  
  
'  
  \n\n'  
'  
  
for  
  
  child  
  
in  
  
    root  
    :  
  
      child  
      .  
      tail  
  
      =  
  
      '  
      \n\n'  
      '  
  
if  
  
      hasattr  
      (  
      child  
      .  
      tag  
      ,  
  
      'endswith'  
      )  
  
and  
  
      child
```

```
.  
tag  
.  
endswith  
(  
    '}head'  
):
```

```
pretty_xml_tree  
(  
    child  
)
```

for

body

in

root

```
.  
findall  
(  
    'h:body'  
    ,
```

```
namespaces  
=  
XPNSMAP  
):
```

```
pretty_block  
(  
    body  
)
```

Special case the handling of a body that contains a single block tag

with all content. In this case we prettify the containing block tag

even if it has non block children.

if

```
(  
len  
(  
body  
)
```

```
==
```

```
1
```

and

not

```
callable  
(  
body  
[  
0  
]  
.  
tag  
)
```

and

```
isblock  
(  
body  
[  
0  
])
```

and

not

has_only_blocks
(

body
[
0
)

and

barename
(
body
[
0
]
.
tag
)

not

in

(

'pre'

,

'p'

,

'h1'

,

'h2'

,

'h3'

```
,  
  
'h4'  
,  
  
'h5'  
,  
  
'h6'  
)  
  
and  
  
len  
(  
body  
[  
0  
)  
  
>  
  
0  
)  
  
pretty_block  
(  
body  
[  
0  
],  
  
level  
=  
2  
)  
  
if  
  
container
```

is

not

None

:

Handle <script> and <style> tags

for

child

in

root

.

xpath

(

'//*[local-name()="script" or local-name()="style"]'

):

pretty_script_or_style

(

container

,

child

)

[\[docs\]](#)

def

fix_html

(

```
container
,

raw
):

' Fix any parsing errors in the HTML represented as a string in raw. Fixing is done using the
HTML5 parsing algorithm. '

root

=

container
.
parse_xhtml
(
raw
)

return

serialize
(
root
,

'text/html'
)

\[docs\]
def

pretty_html
(
container
,

name
```

```
,  
  
raw  
):  
  
    ' Pretty print the HTML represented as a string in raw '
```

```
    root  
  
    =  
  
    container  
    .  
    parse_xhtml  
    (  
    raw  
    )  
  
    pretty_html_tree  
    (  
    container  
    ,  
  
    root  
    )
```

```
return
```

```
    serialize  
    (  
    root  
    ,  
  
    'text/html'  
    )
```

```
\[docs\]
```

```
def
```



```
pretty_css
```

```
(
```

```
container
```

```
,
```

```
name
```

```
,
```

```
raw
```

```
):
```

```
' Pretty print the CSS represented as a string in raw '
```

```
sheet
```

```
=
```

```
container
```

```
.
```

```
parse_css
```

```
(
```

```
raw
```

```
)
```

```
return
```

```
serialize
```

```
(
```

```
sheet
```

```
,
```

```
'text/css'
```

```
)
```

```
\[docs\]
```

```
def
```

```
pretty_xml
```

```
(
  container
  ,
  name
  ,
  raw
):
```

' Pretty print the XML represented as a string in raw. If ``name`` is the name of the OPF, extra OPF-specific prettying is performed. '

```
root
=
container
.
parse_xml
(
  raw
)
```

if

```
name
==
container
.
opf_name
:
```

```
pretty_opf
(
  root
)
```

```
pretty_xml_tree
```

```
(  
root  
)
```

```
return
```

```
serialize
```

```
(  
root  
,
```

```
'text/xml'  
)
```

```
\[docs\]
```

```
def
```

```
fix_all_html
```

```
(  
container  
):
```

```
' Fix any parsing errors in all HTML files in the container. Fixing is done using the HTML5 parsing  
algorithm. '
```

```
for
```

```
name
```

```
,
```

```
mt
```

```
in
```

```
iteritems
```

```
(
```

```
container
    .
    mime_map
):

if

    mt

in

    OEB_DOCS
    :

    container
    .
    parsed
    (
    name
    )

    container
    .
    dirty
    (
    name
    )

\[docs\]
def

    pretty_all
    (
    container
    ):

    ' Pretty print all HTML/CSS/XML files in the container '
```

```
xml_types
```

```
=
```

```
{  
  guess_type  
  (  
    'a.ncx'  
  ),  

```

```
  guess_type  
  (  
    'a.xml'  
  ),  

```

```
  guess_type  
  (  
    'a.svg'  
  )}  

```

```
for
```

```
  name  
  ,
```

```
  mt
```

```
in
```

```
  iteritems  
  (  
    container  
    .  
    mime_map  
  ):
```

```
  prettied
```

```
=
```

False

if

mt

in

OEB_DOCS

:

pretty_html_tree

(

container

,

container

.

parsed

(

name

))

prettied

=

True

elif

mt

in

OEB_STYLES

:

container

.

parsed

(

name

)

prettied

=

True

elif

name

==

container

.

opf_name

:

root

=

container

.

parsed

(

name

)

pretty_opf

(

root

)

```
pretty_xml_tree  
(  
    root  
)
```

```
prettied
```

```
=
```

```
True
```

```
elif
```

```
mt
```

```
in
```

```
xml_types  
:
```

```
pretty_xml_tree  
(  
    container  
    .  
    parsed  
(  
    name  
    ))
```

```
prettied
```

```
=
```

```
True
```

```
if
```



```
prettied  
:
```

```
container
```

```
.  
dirty  
(  
name  
)
```

Source code for calibre.ebooks.oeb.polish.replace

```
#!/usr/bin/env python
```

```
__license__
```

```
=
```

```
'GPL v3'
```

```
__copyright__
```

```
=
```

```
'2013, Kovid Goyal <kovid at kovidgoyal.net>'
```

```
__docformat__
```

```
=
```

```
'restructuredtext en'
```

```
import
```

```
codecs
```

```
,
```

```
shutil
```

```
,
```

```
os
```

```
,
```

posixpath

from

polyglot.builtins

import

iteritems

,

itervalues

from

functools

import

partial

from

collections

import

Counter

,

defaultdict

from

calibre

import

sanitize_file_name

from

```
calibre.ebooks.chardet
```

```
import
```

```
strip_encoding_declarations
```

```
from
```

```
calibre.ebooks.oeb.base
```

```
import
```

```
css_text
```

```
from
```

```
calibre.ebooks.oeb.polish.css
```

```
import
```

```
iter_declarations
```

```
,
```

```
remove_property_value
```

```
from
```

```
calibre.ebooks.oeb.polish.utils
```

```
import
```

```
extract
```

```
from
```

```
polyglot.urlib
```

```
import
```

```
urlparse
```

```
,
```

```
urlunparse
```

```
class
```

```
LinkReplacer
```

```
:
```

```
def
```

```
__init__
```

```
(
```

```
self
```

```
,
```

```
base
```

```
,
```

```
container
```

```
,
```

```
link_map
```

```
,
```

```
frag_map
```

```
):
```

```
self
```

```
.
```

```
base
```

```
=
```

```
base
```

```
self
```

```
.
```

```
frag_map
```

```
=
```

```
frag_map
```

```
self
```

```
.
```

```
link_map
```

```
=
```

```
link_map
```

```
self
```

```
.
```

```
container
```

```
=
```

```
container
```

```
self
```

```
.
```

```
replaced
```

```
=
```

```
False
```

```
def
```

```
__call__
```

```
(
```

```
self
```

```
,
```

```
url
```

```
):
```

if

url

and

url

.

startswith

(

'#'

):

repl

=

self

.

frag_map

(

self

.

base

,

url

[

1

:])

if

not

repl

or

```
repl
```

```
==
```

```
url
```

```
[  
1  
:]:
```

```
return
```

```
url
```

```
self
```

```
.  
replaced
```

```
=
```

```
True
```

```
return
```

```
'#'
```

```
+
```

```
repl
```

```
name
```

```
=
```

```
self
```

```
.  
container
```

```
.  
href_to_name  
(
```



```
url
,

self
.
base
)

if

not

name
:

return

url

nname

=

self
.
link_map
.
get
(
name
,

None
)

if

not
```

```
nname
:

return

url

purl

=

urlparse
(
url
)

href

=

self
.
container
.
name_to_href
(
nname
,
self
.
base
)

if

purl
.
fragment
```

```
:  
  
nfrag  
  
=  
  
self  
.  
frag_map  
(  
    name  
    ,  
  
    purl  
    .  
    fragment  
    )  
  
if  
  
nfrag  
:  
  
    href  
  
    +=  
  
    '#  
    %s'  
    ,  
  
    %  
    nfrag  
  
if  
  
href  
  
!=
```

```
url
:

self
.
replaced

=

True

return

href

class

IdReplacer
:

def

__init__
(
self
,

base
,

container
,

id_map
):
```

```
self
```

```
.
```

```
base
```

```
,
```

```
self
```

```
.
```

```
container
```

```
,
```

```
self
```

```
.
```

```
replaced
```

```
=
```

```
base
```

```
,
```

```
container
```

```
,
```

```
False
```

```
self
```

```
.
```

```
id_map
```

```
=
```

```
id_map
```

```
def
```

```
__call__
```

```
(
```

```
self
```

```
,
```

```
url
```

```
);  
  
if  
  
url  
  
and  
  
url  
.  
startswith  
(  
  '#'  
):
```

```
repl  
  
=  
  
self  
.  
id_map  
.  
get  
(  
  self  
.  
base  
,  
  
{})  
.  
get  
(  
  url  
  [  
    1  
  :])
```

```
if
```

repl

is

None

or

repl

==

url

[

1

:]:

return

url

self

.

replaced

=

True

return

'#'

+

repl

name

```
=  
  
self  
.container  
.href_to_name  
(  
url  
,  
  
self  
.base  
)  
  
if  
  
not  
  
name  
:  
  
return  
  
url  
  
id_map  
  
=  
  
self  
.id_map  
.get  
(  
name  
)
```


if

id_map

is

None

:

return

url

purl

=

urlparse

(

url

)

nfrag

=

id_map

.

get

(

purl

.

fragment

)

if

nfrag

is

None

:

return

url

purl

=

purl

.

_replace

(

fragment

=

nfrag

)

href

=

urlunparse

(

purl

)

if

href

!=

```
url
:

self
.
replaced

=

True

return

href


class

LinkRebaser
:

def

__init__
(
self
,
container
,
old_name
,
new_name
):
```

```
self
.  
old_name  
,  
  
self  
.  
new_name  
  
=  
  
old_name  
,  
  
new_name
```

```
self  
.  
container  
  
=  
  
container
```

```
self  
.  
replaced  
  
=
```

False

```
def  
  
__call__  
(  
self  
,
```

```
url
):

if

url

and

url
.
startswith
(
'#'
):

return

url

purl

=

urlparse
(
url
)

frag

=

purl
.
fragment

name
```

```
=  
  
self  
.container  
.href_to_name  
(  
url  
,  
  
self  
.old_name  
)  
  
if  
  
not  
  
name  
:  
  
return  
  
url  
  
if  
  
name  
  
==  
  
self  
.old_name  
:
```

```
name

=

self
.
new_name

href

=

self
.
container
.
name_to_href
(
name
,

self
.
new_name
)

if

frag
:

href

+=

'#'

+

frag
```

if

href

!=

url

:

self

.

replaced

=

True

return

href

[\[docs\]](#)

def

replace_links

(

container

,

link_map

,

frag_map

=

lambda


```
name
```

```
,
```

```
frag
```

```
:
```

```
frag
```

```
,
```

```
replace_in_opf
```

```
=
```

```
False
```

```
):
```

```
'''
```

Replace links to files in the container. Will iterate over all files in the container and change the specified links in them.

:param link_map: A mapping of old canonical name to new canonical name. For example:
:code:`{'images/old.png': 'images/new.png'}`

:param frag_map: A callable that takes two arguments ``(name, anchor)`` and

returns a new anchor. This is useful if you need to change the anchors in

HTML files. By default, it does nothing.

:param replace_in_opf: If False, links are not replaced in the OPF file.

```
'''
```

```
for
```

```
name
```

```
,
```

```
media_type
```

in

```
iteritems  
(  
    container  
    .  
    mime_map  
):
```

if

name

==

```
container  
.  
opf_name
```

and

not

```
replace_in_opf  
:
```

continue

repl

=

```
LinkReplacer  
(  
    name  
    ,  
    container  
    ,
```

```

link_map
,

frag_map
)

container
.
replace_links
(
name
,

repl
)

def

replace_ids
(
container
,

id_map
):

"""

    Replace all links in the container that pointed to the changed ids.

    :param id_map: A mapping of {name:id_map} where each id_map is a mapping of
    {old_id:new_id}

    :return: True iff at least one link was changed

    """

```

```
changed

=

False

for

name
,

media_type

in

iteritems
(
container
.
mime_map
):

repl

=

IdReplacer
(
name
,

container
,

id_map
)

container
.
replace_links
```

```
(
  name
  ,
  repl
)

if

name

==

container
.
opf_name
:

imap

=

id_map
.
get
(
  name
  ,
  {})

for

item

in

container
.
opf_xpath
```

```
(  
'//*[@idref]'  
):
```

```
old_id
```

```
=
```

```
item
```

```
.
```

```
get
```

```
(
```

```
'idref'
```

```
)
```

```
if
```

```
old_id
```

```
is
```

```
not
```

```
None
```

```
:
```

```
new_id
```

```
=
```

```
imap
```

```
.
```

```
get
```

```
(
```

```
old_id
```

```
)
```

```
if
```

```
new_id
```

```
is
```

```
not
```

```
None
```

```
:
```

```
item
```

```
.
```

```
set
```

```
(
```

```
'idref'
```

```
,
```

```
new_id
```

```
)
```

```
if
```

```
repl
```

```
.
```

```
replaced
```

```
:
```

```
changed
```

```
=
```

```
True
```

```
return
```

```
changed
```

```
def
```

```
smarten_punctuation
```

```
(
```

```
container
```

```
,
```

```
report
```

```
):
```

```
from
```

```
calibre.ebooks.conversion.preprocess
```

```
import
```

```
smarten_punctuation
```

```
smartened
```

```
=
```

```
False
```

```
for
```

```
path
```

```
in
```

```
container
```

```
.
```

```
spine_items
```

```
:
```

```
name
```

```
=
```

```
container
```



```
.  
abspath_to_name  
(  
    path  
)
```

```
changed
```

```
=
```

```
False
```

```
with
```

```
container
```

```
.  
open  
(  
    name  
,  
    'r+b'  
)
```

```
as
```

```
f  
:
```

```
html
```

```
=
```

```
container
```

```
.  
decode  
(  
    f  
.  
read
```

```
()
```

```
newhtml
```

```
=
```

```
smarten_punctuation
```

```
(
```

```
html
```

```
,
```

```
container
```

```
.
```

```
log
```

```
)
```

```
if
```

```
newhtml
```

```
!=
```

```
html
```

```
:
```

```
changed
```

```
=
```

```
True
```

```
report
```

```
(
```

```
-
```

```
(
```

```
'Smartened punctuation in:
```

```
%s
```

```
,
```

```
)
```

```
%
name
)

newhtml

=

strip_encoding_declarations
(
newhtml
)

f
.
seek
(
0
)

f
.
truncate
()

f
.
write
(
codecs
.
BOM_UTF8
+
newhtml
.
encode
(
```

```
'utf-8'
))

if

changed
:

# Add an encoding declaration (it will be added automatically when

# serialized)

root

=

container
.
parsed
(
  name
)

for

m

in

root
.
xpath
(
  'descendant::*[local-name()="meta" and @http-equiv]'
):

m
```

```
.
getparent
()
.
remove
(
m
)

container
.
dirty
(
name
)

smartened

=

True

if

not

smartened
:

report
(
-
(
'No punctuation that could be smartened found'
))

return
```

smartened

[\[docs\]](#)

def

rename_files

(

container

,

file_map

):

'''

Rename files in the container, automatically updating all links to them.

:param file_map: A mapping of old canonical name to new canonical name, for

example: :code:`{'text/chapter1.html': 'chapter1.html'}`.

'''

overlap

=

set

(

file_map

)

.

intersection

(

set

(

```
itervalues
(
file_map
)))

if

overlap
:

raise

ValueError
(
'Circular rename detected. The files
%s
are both rename targets and destinations'

%

', '
.
join
(
overlap
))

for

name
,

dest

in

iteritems
(
file_map
):
```

if

container

.

exists

(

dest

):

if

name

!=

dest

and

name

.

lower

()

==

dest

.

lower

():

A case change on an OS with a case insensitive file-system.

continue

raise


```
ValueError
(
'Cannot rename
{0}
to
{1}
as
{1}
already exists'
.
format
(
name
,
dest
))

if
len
(
tuple
(
intervalues
(
file_map
)))
!=
len
(
set
(
intervalues
(
file_map
))):

raise
```

```
ValueError
(
'Cannot rename, the set of destination files contains duplicates'
)
```

```
link_map
```

```
=
```

```
{}
```

```
for
```

```
current_name
```

```
,
```

```
new_name
```

```
in
```

```
iteritems
```

```
(
```

```
file_map
```

```
):
```

```
container
```

```
.
```

```
rename
```

```
(
```

```
current_name
```

```
,
```

```
new_name
```

```
)
```

```
if
```

```
new_name
```

```
!=

container
.
opf_name
:

# OPF is handled by the container

link_map
[
current_name
]

=

new_name

replace_links
(
container
,

link_map
,

replace_in_opf
=
True
)

def

replace_file
(
container
,

name
```

```
,  
  
path  
,  
  
basename  
,  
  
force_mt  
=  
None  
):  
  
dirname  
,  
  
base  
  
=  
  
name  
.  
rpartition  
(  
    '/'  
)[  
    0  
::  
    2  
]  
  
nname  
  
=  
  
sanitize_file_name  
(  
    basename  
)
```

```
if
```

```
    dirname
```

```
    :
```

```
    nname
```

```
    =
```

```
    dirname
```

```
    +
```

```
    '/'
```

```
    +
```

```
    nname
```

```
with
```

```
    open
```

```
    (
```

```
    path
```

```
    ,
```

```
    'rb'
```

```
    )
```

```
as
```

```
    src
```

```
    :
```

```
if
```

```
    name
```

```
    !=
```

```
nname
```

```
:
```

```
count
```

```
=
```

```
0
```

```
b
```

```
,
```

```
e
```

```
=
```

```
nname
```

```
.
```

```
rpartition
```

```
(
```

```
''
```

```
:
```

```
)[
```

```
0
```

```
::
```

```
2
```

```
]
```

```
while
```

```
container
```

```
.
```

```
exists
```

```
(
```

```
nname
```

```
):
```

```
count
```

```
+=
```

1

nname

=

b

+

(

'

—

%d

.

%s

'

%

(

count

,

e

))

rename_files

(

container

,

{

name

:

nname

})

mt

```
=  
  
force_mt  
  
or  
  
container  
.  
guess_type  
(  
    nname  
)  
  
container  
.  
mime_map  
[  
    nname  
]  
  
=  
  
mt  
  
for  
  
itemid  
,  
  
q  
  
in  
  
iteritems  
(  
    container  
.  
    manifest_id_map  
):
```



```

if
q
==

nname
:

for

item

in

container
.
opf_xpath
(
'//opf:manifest/opf:item[@href and @id="
%s
"]'

%

itemid
):

item
.
set
(
'media-type'
,
mt
)

container
.

```

```
dirty
(  
container  
.  
opf_name  
)
```

with

```
container  
.  
open  
(  
nname  
,  
'wb'  
)
```

as

```
dest  
:
```

```
shutil  
.  
copyfileobj  
(  
src  
,  
dest  
)
```

def

```
mt_to_category  
(  
container
```

```
,  
  
mt  
):  
  
from  
  
calibre.ebooks.oeb.polish.utils  
  
import  
  
guess_type  
  
from  
  
calibre.ebooks.oeb.polish.container  
  
import  
  
OEB_FONTS  
  
from  
  
calibre.ebooks.oeb.base  
  
import  
  
OEB_DOCS  
,  
  
OEB_STYLES  
  
if  
  
mt  
  
in  
  
OEB_DOCS
```

```
:  
  
category  
  
=  
  
'text'  
  
elif  
  
mt  
  
in  
  
OEB_STYLES  
:  
  
category  
  
=  
  
'style'  
  
elif  
  
mt  
  
in  
  
OEB_FONTS  
:  
  
category  
  
=  
  
'font'
```

elif

mt

==

guess_type

(

'a.opf'

):

category

=

'opf'

elif

mt

==

guess_type

(

'a.ncx'

):

category

=

'toc'

else

:

```
category
```

```
=
```

```
mt
```

```
.
```

```
partition
```

```
(
```

```
'/'
```

```
)[
```

```
0
```

```
]
```

```
return
```

```
category
```

```
\[docs\]
```

```
def
```

```
get_recommended_folders
```

```
(
```

```
container
```

```
,
```

```
names
```

```
):
```

```
""" Return the folders that are recommended for the given filenames. The
```

```
recommendation is based on where the majority of files of the same type are
```

```
located in the container. If no files of a particular type are present, the
```

```
recommended folder is assumed to be the folder containing the OPF file. """
```

```
from
```

```
calibre.ebooks.oeb.polish.utils
```

```
import
```

```
guess_type
```

```
counts
```

```
=
```

```
defaultdict
```

```
(
```

```
Counter
```

```
)
```

```
for
```

```
name
```

```
,
```

```
mt
```

```
in
```

```
iteritems
```

```
(
```

```
container
```

```
.
```

```
mime_map
```

```
):
```

```
folder
```

```
=
```

```
name
```

```
.
```

```
rpartition
```

```
(  
'/'  
)[  
0  
]  
  
if  
  
'/'  
  
in  
  
name  
  
else  
  
"  
  
counts  
[  
mt_to_category  
(  
container  
,  
  
mt  
)][  
folder  
]  
  
+=  
  
1  
  
try  
:  
  
opf_folder
```



```
=  
  
counts  
[  
    'opf'  
]  
.  
most_common  
(  
    1  
)[  
    0  
][  
    0  
]
```

except

```
KeyError  
:
```

```
opf_folder
```

```
=
```

```
"
```

```
recommendations
```

```
=
```

```
{  
    category  
    :  
    counter  
    .  
    most_common  
    (  
        1
```

```
)[
0
][
0
]

for

category
,

counter

in

iteritems
(
counts
)}

return

{
n
:
recommendations
.
get
(
mt_to_category
(
container
,

guess_type
(
os
.
path
.
basename
(
```

```
n
))),

opf_folder
)

for

n

in

names
}

def

normalize_case
(
container
,
val
):

def

safe_listdir
(
x
):

try
:

return

os
```

```
.  
listdir  
(  
x  
)
```

```
except
```

```
OSError  
:
```

```
return
```

```
()
```

```
parts
```

```
=
```

```
val
```

```
.  
split  
(  
'/'  
)
```

```
ans
```

```
=
```

```
[]
```

```
for
```

```
i
```

```
in
```

```
range
(
len
(
parts
)):

q

=

'/'
.
join
(
parts
[:
i
+
1
])

x

=

container
.
name_to_abspath
(
q
)

xl

=

parts
[
```

```
i
]
.
lower
()

candidates

=

[
c

for

c

in

safe_listdir
(
os
.
path
.
dirname
(
x
))

if

c

!=

parts
[
i
]

and
```

```
c
.
lower
()

==

xl
]

ans
.
append
(
candidates
[
0
]

if

candidates

else

parts
[
i
])

return

'/'
.
join
(
ans
)
```

```
def
rationalize_folders
(
container
,
folder_type_map
):

all_names

=

set
(
container
.
mime_map
)

new_names

=

set
()

name_map

=

{}

for
key
```



```
in  
  
tuple  
(  
  folder_type_map  
):
```

```
val  
  
=  
  
folder_type_map  
[  
  key  
]
```

```
folder_type_map  
[  
  key  
]  
  
=
```

```
normalize_case  
(  
  container  
  ,
```

```
val  
)
```

```
for
```

```
name
```

```
in
```

```
all_names  
:
```

if

name

.

startswith

(

'META-INF/'

):

continue

category

=

mt_to_category

(

container

,

container

.

mime_map

[

name

])

folder

=

folder_type_map

.

get

(

category

,

None

```
)  
  
if  
  
folder  
  
is  
  
not  
  
None  
:  
  
bn  
  
=  
  
posixpath  
.  
basename  
(  
name  
)  
  
new_name  
  
=  
  
posixpath  
.  
join  
(  
folder  
,  
  
bn  
)  
  
if
```

```
new_name
```

```
!=
```

```
name
```

```
:
```

```
c
```

```
=
```

```
0
```

```
while
```

```
new_name
```

```
in
```

```
all_names
```

```
or
```

```
new_name
```

```
in
```

```
new_names
```

```
:
```

```
c
```

```
+=
```

```
1
```

```
n
```

```
,
```

```
ext
```

```
=
```

```
bn
```

```
.
```

```
rpartition
```

```
(
```

```
'.'
```

```
)[
```

```
0
```

```
::
```

```
2
```

```
]
```

```
new_name
```

```
=
```

```
posixpath
```

```
.
```

```
join
```

```
(
```

```
folder
```

```
,
```

```
'
```

```
'
```

```
%s
```

```
—
```

```
%d
```

```
.
```

```
%s
```

```
'
```

```
'
```

```
%
```

```
%
```

```
(
```

```
n
```

```
,
```

```
c
```

```
,  
  
ext  
)  
  
name_map  
[  
  name  
]  
  
=  
  
new_name  
  
new_names  
.  
add  
(  
  new_name  
)  
  
return  
  
name_map  
  
def  
  
remove_links_in_sheet  
(  
  href_to_name  
,  
  
  sheet  
,  
  
  predicate  
):
```

```
import_rules_to_remove
```

```
=
```

```
[]
```

```
changed
```

```
=
```

```
False
```

```
for
```

```
i
```

```
,
```

```
r
```

```
in
```

```
enumerate
```

```
(
```

```
sheet
```

```
):
```

```
if
```

```
r
```

```
.
```

```
type
```

```
==
```

```
r
```

```
.
```

```
IMPORT_RULE
```

```
:
```

```
name
```

```
=
```

```
href_to_name
```

```
(
```

```
  r
```

```
  .
```

```
  href
```

```
)
```

```
if
```

```
  predicate
```

```
  (
```

```
    name
```

```
    ,
```

```
    r
```

```
    .
```

```
    href
```

```
    ,
```

```
None
```

```
);
```

```
import_rules_to_remove
```

```
  .
```

```
  append
```

```
  (
```

```
    i
```

```
  )
```

```
for
```

```
  i
```

```
in
```



```
sorted
(
import_rules_to_remove
,
reverse
=
True
):
```

```
sheet
.
deleteRule
(
i
)
```

```
changed
=
```

```
True
```

```
for
```

```
dec
```

```
in
```

```
iter_declarations
(
sheet
):
```

```
changed
=
```

```
remove_links_in_declaration  
(  
  href_to_name  
,  
  
  dec  
,  
  
  predicate  
)
```

or

changed

return

changed

def

```
remove_links_in_declaration  
(  
  href_to_name  
,  
  
  style  
,  
  
  predicate  
):
```

def

```
check_pval  
(  
  v  
):
```

```
if
v
.
type

==

v
.
URI
:

name

=

href_to_name
(
v
.
uri
)

return

predicate
(
name
,

v
.
uri
,

None
)
```

```
return
```

```
False
```

```
changed
```

```
=
```

```
False
```

```
for
```

```
p
```

```
in
```

```
tuple
```

```
(
```

```
style
```

```
.
```

```
getProperties
```

```
(
```

```
all
```

```
=
```

```
True
```

```
)):
```

```
changed
```

```
=
```

```
remove_property_value
```

```
(
```

```
p
```

```
,
```

```
check_pval
```

```
)
```

or

changed

return

changed

def

remove_links_to

(
container
,

predicate
):

*""" predicate must be a function that takes the arguments (name, href,
fragment=None) and returns True iff the link should be removed """*

from

calibre.ebooks.oeb.base

import

iterlinks

,

OEB_DOCS

,

OEB_STYLES

,

XPath

,

XHTML

stylepath

=

XPath

(

'//h:style'

)

styleattrpath

=

XPath

(

'//*[@style]'

)

changed

=

set

()

for

name

,

mt

in

```
iteritems
(
container
.
mime_map
):
```

```
removed
```

```
=
```

```
False
```

```
if
```

```
mt
```

```
in
```

```
OEB_DOCS
:
```

```
root
```

```
=
```

```
container
```

```
.
parsed
(
name
)
```

```
for
```

```
el
```

```
,
```

```
attr
```

```
,  
  
href  
,  
  
pos  
  
in  
  
iterlinks  
(  
root  
,  
  
find_links_in_css  
=  
False  
):  
  
hname  
  
=  
  
container  
.  
href_to_name  
(  
href  
,  
  
name  
)  
  
frag  
  
=  
  
href  
.  
partition  
(
```



```
'#'
```

```
)[
```

```
-
```

```
1
```

```
]
```

```
if
```

```
predicate
```

```
(
```

```
hname
```

```
,
```

```
href
```

```
,
```

```
frag
```

```
):
```

```
if
```

```
attr
```

```
is
```

```
None
```

```
:
```

```
el
```

```
.
```

```
text
```

```
=
```

```
None
```

```
else
```

```
:
```

if

el

.

tag

==

XHTML

(

'link'

)

or

el

.

tag

==

XHTML

(

'img'

):

extract

(

el

)

else

:

del

el

.

attrib

```
[
  attr
]

removed

=

True

for

tag

in

stylepath
(
  root
):

if

tag
.
text

and

(
  tag
  .
  get
  (
    'type'
  )
  or

  'text/css'
```

```
)  
.  
lower  
(  
  
==  
  
'text/css'  
:  
  
sheet  
  
=  
  
container  
.  
parse_css  
(  
tag  
.  
text  
)  
  
if  
  
remove_links_in_sheet  
(  
partial  
(  
container  
.  
href_to_name  
,  
  
base  
=  
name  
),  
  
sheet  
,
```

```
predicate
):
```

```
tag
.
text
```

```
=
```

```
css_text
(
sheet
)
```

```
removed
```

```
=
```

```
True
```

```
for
```

```
tag
```

```
in
```

```
styleattrpath
(
root
):
```

```
style
```

```
=
```

```
tag
.
get
```

```
(  
'style'  
)  
  
if  
  
style  
:  
  
style  
  
=  
  
container  
.  
parse_css  
(  
style  
,  
  
is_declaration  
=  
True  
)  
  
if  
  
remove_links_in_declaration  
(  
partial  
(  
container  
.  
href_to_name  
,  
  
base  
=  
name  
),
```

style

,

predicate

):

removed

=

True

tag

.

set

(

'style'

,

css_text

(

style

))

elif

mt

in

OEB_STYLES

:

removed

=

remove_links_in_sheet

```
(
  partial
  (
    container
    .
    href_to_name
    ,

    base
    =
    name
  ),

  container
  .
  parsed
  (
    name
  ),

  predicate
)
```

if

```
removed
:
```

```
changed
```

```
.
add
(
  name
)
```

for

```
i
```

in


```
changed
:
```

```
container
```

```
.
dirty
(
i
)
```

```
return
```

```
changed
```

```
def
```

```
get_spine_order_for_all_files
(
container
):
```

```
linear_names
```

```
,
```

```
non_linear_names
```

```
=
```

```
[],
```

```
[]
```

```
for
```

```
name
```

```
,
```

```
is_linear
```

```
in
```

```
container
```

```
.
```

```
spine_names
```

```
:
```

```
(
```

```
linear_names
```

```
if
```

```
is_linear
```

```
else
```

```
non_linear_names
```

```
)
```

```
.
```

```
append
```

```
(
```

```
name
```

```
)
```

```
all_names
```

```
=
```

```
linear_names
```

```
+
```

```
non_linear_names
```

```
spine_names
```

```
=
```

```
frozenset  
(  
all_names  
)
```

```
ans
```

```
=
```

```
{}
```

```
for
```

```
spine_pos
```

```
,
```

```
name
```

```
in
```

```
enumerate
```

```
(  
all_names  
):
```

```
ans
```

```
.
```

```
setdefault
```

```
(  
name
```

```
,
```

```
(  
spine_pos
```

```
,
```

```
-
```

```
1
```

```
))
```

for

i

,

href

in

enumerate

(

container

.

iterlinks

(

name

,

get_line_numbers

=

False

)):

lname

=

container

.

href_to_name

(

href

,

name

)

if

```
lname

not

in

spine_names
:

ans
.
setdefault
(
lname
,

(
spine_pos
,

i
))

return

ans
```

Source code for calibre.ebooks.oeb.polish.split

```
#!/usr/bin/env python
```

```
__license__
```

```
=
```

```
'GPL v3'
```

```
__copyright__
```

```
=
```

```
'2013, Kovid Goyal <kovid at kovidgoyal.net>'
```

```
import
```

```
copy
```

```
,
```

```
os
```

```
,
```

```
re
```

```
from
```

```
polyglot.builtins
```

```
import
```

```
string_or_bytes
```

```
from
```

```
calibre.ebooks.oeb.base
```

```
import
```

```
barename
```

```
,
```

```
XPNSMAP
```

```
,
```

```
XPath
```

```
,
```

```
OPF
```

```
,
```

```
XHTML
```

```
,
```

```
OEB_DOCS
```

```
from
```

```
calibre.ebooks.oeb.polish.errors
```

```
import
```

```
MalformedMarkup
```

```
from
```

```
calibre.ebooks.oeb.polish.toc
```

```
import
```

```
node_from_loc
```

```
from
```

```
calibre.ebooks.oeb.polish.replace
```

```
import
```

```
LinkRebaser
```

```
from
```

```
polyglot.builtins
```

```
import
```

```
iteritems
```

```
from
```

```
polyglot.urllib
```

```
import
```

```
urlparse
```

```
class
```

```
AbortError
```

```
(  
    ValueError  
):
```

```
pass
```

```
def
```

```
in_table
```

```
(  
    node  
):
```



```
while
```

```
node
```

```
is
```

```
not
```

```
None
```

```
:
```

```
if
```

```
node
```

```
.
```

```
tag
```

```
.
```

```
endswith
```

```
(
```

```
'}table'
```

```
):
```

```
return
```

```
True
```

```
node
```

```
=
```

```
node
```

```
.
```

```
getparent
```

```
()
```

```
return
```

False

def

adjust_split_point

(

split_point

,

log

):

'''

Move the split point up its ancestor chain if it has no content

before it. This handles the common case:

<div id="chapter1"><h2>Chapter 1</h2>...</div> with a page break on the

h2.

'''

sp

=

split_point

while

True

:

parent

```
=  
  
sp  
.  
getparent  
(  
  
if  
  
(  
  
parent  
  
is  
  
None  
  
or  
  
barename  
(  
parent  
.  
tag  
)  
  
in  
  
{  
'body'  
,  
  
'html'  
}  
  
or  
  
(  
parent
```

```
.  
text
```

```
and
```

```
parent
```

```
.  
text
```

```
.  
strip  
(())
```

```
or
```

```
parent
```

```
.  
index  
(  
sp  
)
```

```
>
```

```
0
```

```
):
```

```
break
```

```
sp
```

```
=
```

```
parent
```

```
if
```

sp

is

not

split_point
:

log

.

debug

(
'Adjusted split point to ancestor'
)

return

sp

def

get_body

(
root
):

return

root

.

find

(
'h:body'
,

namespaces

```

=
XPNSMAP
)

def
do_split
(
split_point
,
log
,
before
=
True
):
'''
Split tree into a *before* and an *after* tree at ``split_point``.

:param split_point: The Element at which to split
:param before: If True tree is split before split_point, otherwise after split_point
:return: before_tree, after_tree
'''

if
before
:

# We cannot adjust for after since moving an after split point to a

```

parent will cause breakage if the parent contains any content

after the original split point

split_point

=

adjust_split_point

(

split_point

,

log

)

tree

=

split_point

.

getroottree

()

path

=

tree

.

getpath

(

split_point

)

```
tree
,

tree2

=

copy
.
deepcopy
(
tree
),

copy
.
deepcopy
(
tree
)

root
,

root2

=

tree
.
getroot
(),

tree2
.
getroot
()

body
,
```



```
body2
```

```
=
```

```
map
```

```
(
```

```
get_body
```

```
,
```

```
(
```

```
root
```

```
,
```

```
root2
```

```
))
```

```
split_point
```

```
=
```

```
root
```

```
.
```

```
xpath
```

```
(
```

```
path
```

```
)[
```

```
0
```

```
]
```

```
split_point2
```

```
=
```

```
root2
```

```
.
```

```
xpath
```

```
(
```

```
path
```

```
)[
```

```
0
```

```
]
```

```
def
```

```
    nix_element
```

```
    (
```

```
    elem
```

```
    ,
```

```
    top
```

```
    =
```

```
    True
```

```
    ): 
```

```
# Remove elem unless top is False in which case replace elem by its
```

```
# children
```

```
    parent
```

```
    =
```

```
    elem
```

```
    .
```

```
    getparent
```

```
    ()
```

```
if
```

```
    top
```

```
    :
```

```
    parent
```

```
    .
```

```
    remove
```

```
    (
```

```
elem
)

else
:

index

=

parent
.
index
(
elem
)

parent
[
index
:
index
+
1
]

=

list
(
elem
.
iterchildren
())
```

Tree 1

```
hit_split_point
```

```
=
```

```
False
```

```
keep_descendants
```

```
=
```

```
False
```

```
split_point_descendants
```

```
=
```

```
frozenset
```

```
(
```

```
split_point
```

```
.
```

```
iterdescendants
```

```
())
```

```
for
```

```
elem
```

```
in
```

```
tuple
```

```
(
```

```
body
```

```
.
```

```
iterdescendants
```

```
)):
```

```
if
```

```
elem
```

is

split_point
:

hit_split_point

=

True

if

before
:

nix_element
(
elem
)

else

:

We want to keep the descendants of the split point in

Tree 1

keep_descendants

=

True

```
# We want the split point element, but not its tail
```

```
elem
```

```
.
```

```
tail
```

```
=
```

```
,
```

```
\n
```

```
,
```

```
continue
```

```
if
```

```
hit_split_point
```

```
:
```

```
if
```

```
keep_descendants
```

```
:
```

```
if
```

```
elem
```

```
in
```

```
split_point_descendants
```

```
:
```

```
# elem is a descendant keep it
```

```
continue
```

```
else
```

```
:
```

```
# We are out of split_point, so prevent further set
```

```
# lookups of split_point_descendants
```

```
keep_descendants
```

```
=
```

```
False
```

```
nix_element
```

```
(
```

```
elem
```

```
)
```

```
# Tree 2
```

```
ancestors
```

```
=
```

```
frozenset
```

```
(
```

```
XPath
```

```
(
```

```
'ancestor::*'
```

```
)(
```

```
split_point2
```

```
))
```

for

elem

in

tuple

(

body2

.

iterdescendants

()):

if

elem

is

split_point2

:

if

not

before

:

Keep the split point element's tail, if it contains non-whitespace

text

tail

=


```
elem
.
tail

if

tail

and

not

tail
.
isspace
():

parent

=

elem
.
getparent
()

idx

=

parent
.
index
(
elem
)

if
```

```
idx
```

```
==
```

```
0
```

```
:
```

```
parent
```

```
.
```

```
text
```

```
=
```

```
(
```

```
parent
```

```
.
```

```
text
```

```
or
```

```
"
```

```
)
```

```
+
```

```
tail
```

```
else
```

```
:
```

```
sib
```

```
=
```

```
parent
```

```
[
```

```
idx
```

```
-
```

```
1
```

```
]
```

```
sib
```

```
.
```

```
tail
```

```
=
```

```
(
```

```
sib
```

```
.
```

```
tail
```

```
or
```

```
"
```

```
)
```

```
+
```

```
tail
```

```
# Remove the element itself
```

```
nix_element
```

```
(
```

```
elem
```

```
)
```

```
break
```

```
if
```

```
elem
```

```
in
```

```
ancestors
```

```
:
```

We have to preserve the ancestors as they could have CSS

styles that are inherited/applicable, like font or

width. So we only remove the text, if any.

elem

.

text

=

,

\n

,

else

:

nix_element

(

elem

,

top

=

False

)

body2

.

text

=

```
'  
    \n  
'  
  
    return  
  
    tree  
,  
  
    tree2  
  
    class  
  
    SplitLinkReplacer  
    :  
  
    def  
  
    __init__  
    (  
    self  
    ,  
  
    base  
    ,  
  
    bottom_anchors  
    ,  
  
    top_name  
    ,  
  
    bottom_name  
    ,  
  
    container
```

```
);  
  
self  
.bottom_anchors  
,  
  
self  
.bottom_name  
  
=  
  
bottom_anchors  
,  
  
bottom_name  
  
  
self  
.container  
,  
  
self  
.top_name  
  
=  
  
container  
,  
  
top_name  
  
  
self  
.base  
  
=
```

```
base

self
.
replaced

=

False


def
__call__
(
self
,

url
):

if

url

and

url
.
startswith
(
'#'
):

return

url
```

```
name

=

self
.
container
.
href_to_name
(
url
,
self
.
base
)

if
name
!=
self
.
top_name
:

return

url

purl

=

urlparse
(
url
```



```
)  
  
if  
  
purl  
.  
fragment  
  
and  
  
purl  
.  
fragment  
  
in  
  
self  
.  
bottom_anchors  
:  
  
url  
  
=  
  
self  
.  
container  
.  
name_to_href  
(  
self  
.  
bottom_name  
,  
  
self  
.  
base  
)
```

+

'#'

+

purl

.

fragment

self

.

replaced

=

True

return

url

[\[docs\]](#)

def

split

(

container

,

name

,

loc_or_xpath

,

before

```
=
True
,
totals
=
None
):

'''

Split the file specified by name at the position specified by loc_or_xpath.

Splitting automatically migrates all links and references to the affected
files.

:param loc_or_xpath: Should be an XPath expression such as
    //h:div[@id="split_here"]. Can also be a *loc* which is used internally to
    implement splitting in the preview panel.

:param before: If True the split occurs before the identified element otherwise after it.

:param totals: Used internally

'''

root

=

container
.
parsed
(
name
)
```

```
if
    isinstance
    (
        loc_or_xpath
        ,
        str
    ):

    split_point

    =

    root
    .
    xpath
    (
        loc_or_xpath
    )[
    0
    ]

else
:

try
:

    split_point

    =

    node_from_loc
    (
        root
        ,
```

```
loc_or_xpath
```

```
,
```

```
totals
```

```
=
```

```
totals
```

```
)
```

```
except
```

```
MalformedMarkup
```

```
:
```

```
# The webkit HTML parser and the container parser have yielded
```

```
# different node counts, this can happen if the file is valid XML
```

```
# but contains constructs like nested <p> tags. So force parse it
```

```
# with the HTML 5 parser and try again.
```

```
raw
```

```
=
```

```
container
```

```
.
```

```
raw_data
```

```
(
```

```
name
```

```
)
```

```
root
```

```
=
```

```
container
.  
parse_xhtml  
(  
raw  
,  
  
fname  
=  
name  
,  
  
force_html5_parse  
=  
True  
)
```

```
try  
:
```

```
split_point  
  
=
```

```
node_from_loc  
(  
root  
,
```

```
loc_or_xpath  
,
```

```
totals  
=  
totals  
)
```

```
except
```

MalformedMarkup

:

raise

MalformedMarkup

(

—
(

'The file

%s

has malformed markup. Try running the Fix HTML tool'

' before splitting'

)

%

name

)

container

.

replace

(

name

,

root

)

if

in_table

(

split_point

):

raise

```
AbortError  
(  
    'Cannot split inside tables'  
)
```

if

```
split_point  
.  
tag  
.  
endswith  
(  
    '}body'  
):
```

raise

```
AbortError  
(  
    'Cannot split on the <body> tag'  
)
```

tree1

,

tree2

=

```
do_split  
(  
    split_point  
    ,
```

container

.

log


```
,  
  
before  
=  
before  
)  
  
root1  
,  
  
root2  
  
=  
  
tree1  
.  
getroot  
(),  
  
tree2  
.  
getroot  
()  
  
anchors_in_top  
  
=  
  
frozenset  
(  
root1  
.  
xpath  
(  
'//*[@id'  
)  
)  
  
|  
  
frozenset  
(
```

```
root1
.
xpath
(
'//*[@name'
))

|

{
"
}

anchors_in_bottom

=

frozenset
(
root2
.
xpath
(
'//*[@id'
))

|

frozenset
(
root2
.
xpath
(
'//*[@name'
))

base
,

ext
```

```
=
```

```
name
```

```
.
```

```
rpartition
```

```
(
```

```
'.'
```

```
)[
```

```
0
```

```
::
```

```
2
```

```
]
```

```
base
```

```
=
```

```
re
```

```
.
```

```
sub
```

```
(
```

```
r
```

```
'_split\d+$'
```

```
,
```

```
"
```

```
,
```

```
base
```

```
)
```

```
nname
```

```
,
```

```
s
```

```
=
```

```
None
```

```
,
```

0

while

not

nname

or

container

.

exists

(

nname

):

s

+=

1

nname

=

,

%s

_split

%d

.

%s

,

%

(

base

```
,  
  
s  
,  
  
ext  
)  
  
manifest_item  
  
=  
  
container  
.  
generate_item  
(  
  nname  
,  
  
  media_type  
  =  
  container  
  .  
  mime_map  
  [  
    name  
  ])  
  
bottom_name  
  
=  
  
container  
.  
href_to_name  
(  
  manifest_item  
  .  
  get  
(  
    'href'
```

```
),  
  
container  
.  
opf_name  
)
```

```
# Fix links in the split trees
```

```
for
```

```
r
```

```
in
```

```
(  
root1  
,
```

```
root2  
):
```

```
for
```

```
a
```

```
in
```

```
r  
.  
xpath  
(  
'//*[@href]'  
):
```

```
url
```

```
=
```

```
a
.
get
(
  'href'
)

if

url
.
startswith
(
  '#'
):

fname
=
name

else
:

fname
=
container
.
href_to_name
(
  url
  ,
  name
)
```

if

fname

==

name

:

purl

=

urlparse

(

url

)

if

purl

.

fragment

in

anchors_in_top

:

if

r

is

root2

:


```
a
.
set
(
  'href'
,
  f
,
  {
    container
    .
    name_to_href
    (
      name
    ,
    bottom_name
    )
  }
  #
  {
    purl
    .
    fragment
  }
  '
)

else
:

a
.
set
(
  'href'
,
  '#'
)
```

+

purl

.

fragment

)

elif

purl

.

fragment

in

anchors_in_bottom

:

if

r

is

root1

:

a

.

set

(

'href'

,

f

,

{

container

.

```
name_to_href  
(  
  bottom_name  
  ,
```

```
  name  
)  
}
```

```
#
```

```
{  
  purl
```

```
  .  
  fragment  
}
```

```
'
```

```
)
```

```
else
```

```
:
```

```
a
```

```
  .  
  set
```

```
(  
  'href'
```

```
  ,
```

```
  '#'
```

```
  +
```

```
  purl
```

```
  .  
  fragment  
)
```

```
# Fix all links in the container that point to anchors in the bottom tree
```

```
for

fname
,

media_type

in

iteritems
(
container
.
mime_map
):

if

fname

not

in

{
name
,

bottom_name
}:

repl

=

SplitLinkReplacer
(
fname
,

anchors_in_bottom
```

```
,  
  
name  
,  
  
bottom_name  
,  
  
container  
)
```

```
container  
.  
replace_links  
(  
fname  
,  
  
repl  
)
```

```
container  
.  
replace  
(  
name  
,  
  
root1  
)
```

```
container  
.  
replace  
(  
bottom_name  
,  
  
root2
```

```
)

spine
=
container
.
opf_xpath
(
'//opf:spine'
)[
0
]

for
spine_item
,
spine_name
,
linear
in
container
.
spine_iter
:

if
spine_name
==
name
```

```
:
```

```
break
```

```
index
```

```
=
```

```
spine
```

```
.
```

```
index
```

```
(
```

```
spine_item
```

```
)
```

```
+
```

```
1
```

```
si
```

```
=
```

```
spine
```

```
.
```

```
makeelement
```

```
(
```

```
OPF
```

```
(
```

```
'itemref'
```

```
),
```

```
idref
```

```
=
```

```
manifest_item
```

```
.
```

```
get
```

```
(
```

```
'id'
```

```
))
```

```
if
```

```
not
```

```
linear
```

```
:
```

```
si
```

```
.
```

```
set
```

```
(
```

```
'linear'
```

```
,
```

```
'no'
```

```
)
```

```
.
```

```
container
```

```
.
```

```
insert_into_xml
```

```
(
```

```
spine
```

```
,
```

```
si
```

```
,
```

```
index
```

```
=
```

```
index
```

```
)
```

```
.
```

```
container
```

```
.
```

```
dirty
```

```
(
```

```
container
```



```
.  
opf_name  
)
```

```
return
```

```
bottom_name
```

```
\[docs\]
```

```
def
```

```
multisplit
```

```
(  
container
```

```
,
```

```
name
```

```
,
```

```
xpath
```

```
,
```

```
before
```

```
=
```

```
True
```

```
):
```

```
'''
```

Split the specified file at multiple locations (all tags that match the specified XPath expression). See also: :func:`split`.

Splitting automatically migrates all links and references to the affected

files.

:param before: If True the splits occur before the identified element otherwise after it.

```
'''
```

```
root
=
container
.
parsed
(
name
)

nodes
=
root
.
xpath
(
xpath
,
namespaces
=
XPNSMAP
)

if
not
nodes
:

raise
AbortError
(
```

```
-  
(  
    'The expression  
    %s  
    did not match any nodes'  
)
```

```
%
```

```
xpath  
)
```

```
for
```

```
split_point
```

```
in
```

```
nodes  
:
```

```
if
```

```
in_table  
(  
    split_point  
):
```

```
raise
```

```
AbortError  
(  
    'Cannot split inside tables'  
)
```

```
if
```

```
split_point
```

```
.
```

```
tag
.
endswith
(
    '}body'
):

raise

AbortError
(
    'Cannot split on the <body> tag'
)

for

i
,

tag

in

enumerate
(
    nodes
):

tag
.
set
(
    'calibre-split-point'
,

str
(
    i
))
```

```
current
```

```
=
```

```
name
```

```
all_names
```

```
=
```

```
[  
    name  
]
```

```
for
```

```
i
```

```
in
```

```
range  
(  
    len  
(  
        nodes  
    )):  

```

```
current
```

```
=
```

```
split  
(  
    container  
    ,  

```

```
current
```

```
,  
  
'/*[@calibre-split-point="  
%d  
"]'
```

```
%
```

```
i
```

```
,
```

```
before
```

```
=
```

```
before
```

```
)
```

```
all_names
```

```
.
```

```
append
```

```
(
```

```
current
```

```
)
```

```
for
```

```
x
```

```
in
```

```
all_names
```

```
:
```

```
for
```

```
tag
```

```
in
```

```
container
```

```
.
parsed
(
x
)
.
xpath
(
'//*[@calibre-split-point]'
):
```

```
tag
```

```
.
attrib
```

```
.
pop
(
'calibre-split-point'
)
```

```
container
```

```
.
dirty
(
x
)
```

```
return
```

```
all_names
[
1
:]
```

```
class
```

```
MergeLinkReplacer
```

```
:
```

```
def
```

```
    __init__
```

```
    (
```

```
    self
```

```
    ,
```

```
    base
```

```
    ,
```

```
    anchor_map
```

```
    ,
```

```
    master
```

```
    ,
```

```
    container
```

```
):
```

```
    self
```

```
    .
```

```
    container
```

```
    ,
```

```
    self
```

```
    .
```

```
    anchor_map
```

```
    =
```

```
    container
```

```
    ,
```

```
    anchor_map
```

```
    self
```

```
    .
```

```
    master
```



```
=
```

```
master
```

```
self
```

```
.
```

```
base
```

```
=
```

```
base
```

```
self
```

```
.
```

```
replaced
```

```
=
```

```
False
```

```
def
```

```
__call__
```

```
(
```

```
self
```

```
,
```

```
url
```

```
):
```

```
if
```

```
url
```

```
and
```

```
url
```

```
.  
startswith  
(  
    '#'  
):  
  
return  
  
url  
  
name  
  
=  
  
self  
.  
container  
.  
href_to_name  
(  
    url  
    ,  
    self  
    .  
    base  
    )  
  
amap  
  
=  
  
self  
.  
anchor_map  
.  
get  
(  
    name  
    ,
```

None

)

if

amap

is

None

:

return

url

purl

=

urlparse

(

url

)

frag

=

purl

.

fragment

or

"

```
frag

=

amap
.
get
(
frag
,

frag
)

url

=

self
.
container
.
name_to_href
(
self
.
master
,

self
.
base
)

+

'#'

+

frag
```

```
self
.  
replaced
```

```
=
```

```
True
```

```
return
```

```
url
```

```
def
```

```
add_text
```

```
(  
body  
,
```

```
text  
):
```

```
if
```

```
len  
(  
body  
)
```

```
>
```

```
0  
:
```

```
body  
[  
-
```

```
1
]
.
tail

=

(
body
[
-
1
]
.
tail

or

"
)

+

text

else
:

body
.
text

=

(
body
.
text

or
```

```
"
)

+

text


def

all_anchors
(
root
):

return

set
(
root
.
xpath
(
'//*[@id'
))

|

set
(
root
.
xpath
(
'//*[@name'
))

def
```

```
all_stylesheets
(
  container
  ,
  name
):

for

link

in

XPath
(
  '//h:head/h:link[@href]'
){
  container
  .
  parsed
  (
    name
  )):

  name

  =

  container
  .
  href_to_name
  (
    link
    .
    get
    (
      'href'
    ),
    name
```



```
)
```

```
typ
```

```
=
```

```
link
```

```
.
```

```
get
```

```
(
```

```
'type'
```

```
,
```

```
'text/css'
```

```
)
```

```
if
```

```
typ
```

```
==
```

```
'text/css'
```

```
:
```

```
yield
```

```
name
```

```
def
```

```
unique_anchor
```

```
(
```

```
seen_anchors
```

```
,
```

```
current
```

```
):
```

```
c
```

```
=
```

```
0
```

```
ans
```

```
=
```

```
current
```

```
while
```

```
ans
```

```
in
```

```
seen_anchors
```

```
:
```

```
c
```

```
+=
```

```
1
```

```
ans
```

```
=
```

```
,
```

```
%s
```

```
-
```

```
%d
```

```
,
```

```
%
```

```
(  
current  
,
```

```
c  
)
```

```
return
```

```
ans
```

```
def
```

```
remove_name_attributes  
(  
root  
):
```

```
# Remove all name attributes, replacing them with id attributes
```

```
for
```

```
elem
```

```
in
```

```
root
```

```
.
```

```
xpath
```

```
(  
'//*[@id and @name]'  
):
```

```
del
```

```
elem
.
attrib
[
'name'
]

for

elem

in

root
.
xpath
(
'//*[@name]'
):

elem
.
set
(
'id'
,
elem
.
attrib
.
pop
(
'name'
))

def

merge_html
```

```
(
  container
  ,
  names
  ,
  master
  ,
  insert_page_breaks
  =
  False
):
```

```
p
=
container
.
parsed
```

```
root
=
p
(
  master
)
```

Ensure master has a <head>

```
head
=
```

```
root
.
find
(
'h:head'
,

namespaces
=
XPNSMAP
)

if

head

is

None
:

head

=

root
.
makeelement
(
XHTML
(
'head'
))

container
.
insert_into_xml
(
root
,
```

```
head
```

```
,
```

```
0
```

```
)
```

```
seen_anchors
```

```
=
```

```
all_anchors
```

```
(
```

```
root
```

```
)
```

```
seen_stylesheets
```

```
=
```

```
set
```

```
(
```

```
all_stylesheets
```

```
(
```

```
container
```

```
,
```

```
master
```

```
))
```

```
master_body
```

```
=
```

```
p
```

```
(
```

```
master
```

```
)
```

```
.
```

```
findall
(
'h:body'
,

namespaces
=
XPNSMAP
)[
-
1
]

master_base

=

os
.
path
.
dirname
(
master
)

anchor_map

=

{
n
: {}}

for

n

in

names
```


if

n

!=

master

}

first_anchor_map

=

{}

for

name

in

names

:

if

name

==

master

:

continue

Insert new stylesheets into master

for

sheet

in

all_stylesheets

(

container

,

name

):

if

sheet

not

in

seen_stylesheets

:

seen_stylesheets

.

add

(

sheet

)

link

=

head

.

```
makeelement
```

```
(
```

```
XHTML
```

```
(
```

```
'link'
```

```
),
```

```
rel
```

```
=
```

```
'stylesheet'
```

```
,
```

```
type
```

```
=
```

```
'text/css'
```

```
,
```

```
href
```

```
=
```

```
container
```

```
.
```

```
name_to_href
```

```
(
```

```
sheet
```

```
,
```

```
master
```

```
))
```

```
container
```

```
.
```

```
insert_into_xml
```

```
(
```

```
head
```

```
,
```

```
link
```

```
)
```

```
# Rebase links if master is in a different directory
```

```
if
os
.
path
.
dirname
(
name
)

!=

master_base
:

container
.
replace_links
(
name
,

LinkRebaser
(
container
,

name
,

master
))

root

=
```

```
p
(
  name
)

children

=

[]

for

body

in

p
(
  name
)
.
findall
(
  'h:body'
,
  namespaces
  =
  XPNSMAP
):

children
.
append
(
  body
.
  text
```

if

body

.

text

and

body

.

text

.

strip

()

else

,

`\n\n`

,

)

children

.

extend

(

body

)

first_child

=

`"`

for

first_child

in

children

:

if

not

isinstance

(

first_child

,

string_or_bytes

):

break

if

isinstance

(

first_child

,

string_or_bytes

):

body contained only text, no tags

first_child

=

body

.

makeelement

```
(
XHTML
(
'p'
))
```

first_child

```
.
text
,
```

```
children
[
0
]
```

```
=
```

```
children
[
0
],
```

first_child

amap

```
=
```

```
anchor_map
[
name
]
```

```
remove_name_attributes
(
root
)
```


for

elem

in

root

.

xpath

(

'//*[@id]'

):

val

=

elem

.

get

(

'id'

)

if

not

val

:

continue

if

val

in

seen_anchors

:

nval

=

unique_anchor

(

seen_anchors

,

val

)

elem

.

set

(

'id'

,

nval

)

amap

[

val

]

=

nval

else

:

```
seen_anchors
```

```
.
```

```
add
```

```
(
```

```
val
```

```
)
```

```
if
```

```
'id'
```

```
not
```

```
in
```

```
first_child
```

```
.
```

```
attrib
```

```
:
```

```
first_child
```

```
.
```

```
set
```

```
(
```

```
'id'
```

```
,
```

```
unique_anchor
```

```
(
```

```
seen_anchors
```

```
,
```

```
'top'
```

```
))
```

```
seen_anchors
```

```
.
```

```
add
```

```
(
first_child
.
get
(
'id'
))

first_anchor_map
[
name
]

=

first_child
.
get
(
'id'
)

if

insert_page_breaks
:

first_child
.
set
(
'style'
,

first_child
.
get
(
'style'
```

```
,  
"  
)  
+  
'; page-break-before: always'  
)  
  
amap  
[  
"  
]  
=  
  
first_child  
.  
get  
(  
'id'  
)  
  
# Fix links that point to local changed anchors  
  
for  
  
a  
  
in  
  
XPath  
(  
'//h:a[starts-with(@href, "#")]'  
)(  
root  
):
```

```
q
=
a
.
get
(
'href'
)[
1
:]
```

```
if
```

```
q
```

```
in
```

```
amap
:
```

```
a
.
set
(
'href'
,
'#'
```

```
+
```

```
amap
[
q
])
```

for

child

in

children

:

if

isinstance

(

child

,

string_or_bytes

):

add_text

(

master_body

,

child

)

else

:

master_body

.

append

(

copy

.

deepcopy

(

```
child
))

container
.
remove_item
(
name
,

remove_from_guide
=
False
)

# Fix all links in the container that point to merged files

for

fname
,

media_type

in

iteritems
(
container
.
mime_map
):

repl

=
```



```
MergeLinkReplacer
```

```
(
```

```
  fname
```

```
,
```

```
  anchor_map
```

```
,
```

```
  master
```

```
,
```

```
  container
```

```
)
```

```
  container
```

```
.
```

```
  replace_links
```

```
(
```

```
    fname
```

```
,
```

```
    repl
```

```
)
```

```
return
```

```
  first_anchor_map
```

```
def
```

```
  merge_css
```

```
(
```

```
  container
```

```
,
```

```
  names
```

```
,
```

```
master
):

p

=

container
.
parsed

msheet

=

p
(
master
)

master_base

=

os
.
path
.
dirname
(
master
)

merged

=

set
()
```

for

name

in

names

:

if

name

==

master

:

continue

Rebase links if master is in a different directory

if

os

.

path

.

dirname

(

name

)

!=

master_base

:

container

.

replace_links

(

name

,

LinkRebaser

(

container

,

name

,

master

))

sheet

=

p

(

name

)

Remove charset rules

cr

=

[

r

for

r

in

sheet

.

cssRules

if

r

.

type

==

r

.

CHARSET_RULE

]

[

sheet

.

deleteRule

(

sheet

.

cssRules

.

index

(

r

))

for

r

in

cr
]

for

rule

in

sheet

.
cssRules
:

msheet

.
add
(
rule
)

container

.
remove_item
(
name
)

merged

.
add
(
name
)

Remove links to merged stylesheets in the html files, replacing with a

link to the master sheet

for

name

,

mt

in

iteritems

(

container

.

mime_map

):

if

mt

in

OEB_DOCS

:

removed

=

False

root

```
=  
  
p  
(  
  name  
)  
  
for  
  
link  
  
in  
  
XPath  
(  
  '//h:link[@href]'  
)(  
  root  
):  
  
  q  
  
  =  
  
  container  
  .  
  href_to_name  
  (  
    link  
    .  
    get  
    (  
      'href'  
    ),  
    name  
  )  
  
if
```


q

in

merged

:

container

.

remove_from_xml

(

link

)

removed

=

True

if

removed

:

container

.

dirty

(

name

)

if

removed

and

master

not

in

set

```
(  
all_stylesheets  
(  
container  
,
```

```
name  
)):
```

head

=

root

```
.  
find  
(  
'h:head'  
,
```

```
namespaces  
=  
XPNSMAP  
)
```

if

head

is

not

None

```
:  
  
link  
  
=  
  
head  
.  
makeelement  
(  
XHTML  
(  
'link'  
),  
  
type  
=  
'text/css'  
,  
  
rel  
=  
'stylesheet'  
,  
  
href  
=  
container  
.  
name_to_href  
(  
master  
,  
  
name  
)  
)  
  
container  
.  
insert_into_xml  
(
```

```
head
```

```
,
```

```
link
```

```
)
```

```
\[docs\]
```

```
def
```

```
merge
```

```
(
```

```
container
```

```
,
```

```
category
```

```
,
```

```
names
```

```
,
```

```
master
```

```
):
```

```
'''
```

Merge the specified files into a single file, automatically migrating all

links and references to the affected files. The file must all either be HTML or CSS files.

:param category: Must be either ``text`` for HTML files or ``styles`` for CSS files

:param names: The list of files to be merged

*:param master: Which of the merged files is the *master* file, that is, the file that will remain after merging.*

```
'''
```

if

category

not

in

{
'text'

,

'styles'
}:

raise

AbortError

(
'Cannot merge files of type:
%s'
,

%

category
)

if

len
(
names
)

<

2
:

raise

AbortError

```
(  
'Must specify at least two files to be merged'  
)
```

if

master

not

in

names
:

raise

AbortError

```
(  
'The master file (  
%s  
) must be one of the files being merged'
```

%

master
)

if

category

==

```
'text'  
:  
  
merge_html  
(  
  container  
,  
  names  
,  
  master  
)
```

elif

```
category  
  
==
```

```
'styles'  
:  
  
merge_css  
(  
  container  
,  
  names  
,  
  master  
)
```

```
container  
.  
dirty  
(
```

master
)

Source code for calibre.ebooks.oeb.polish.toc

```
#!/usr/bin/env python
```

```
__license__
```

```
=
```

```
'GPL v3'
```

```
__copyright__
```

```
=
```

```
'2013, Kovid Goyal <kovid at kovidgoyal.net>'
```

```
__docformat__
```

```
=
```

```
'restructuredtext en'
```

```
import
```

```
re
```

```
from
```

```
collections
```

```
import
```

```
Counter
,
OrderedDict

from
functools
import
partial

from
operator
import
itemgetter

from
lxml
import
etree

from
lxml.builder
import
ElementMaker

from
calibre
```

```
import

__version__

from

calibre.ebooks.oeb.base

import

(

XPath
,

uuid_id
,

xml2text
,

NCX
,

NCX_NS
,

XML
,

XHTML
,

XHTML_NS
,

serialize
,

EPUB_NS
,
```

```
XML_NS
,
OEB_DOCS
)

from
calibre.ebooks.oeb.polish.errors

import

MalformedMarkup

from
calibre.ebooks.oeb.polish.utils

import

guess_type
,

extract

from
calibre.ebooks.oeb.polish.opf

import

set_guide_item
,

get_book_language

from
calibre.ebooks.oeb.polish.pretty

import

pretty_html_tree
```

```
,  
  
pretty_xml_tree  
  
from  
  
calibre.translations.dynamic  
  
import  
  
translate  
  
from  
  
calibre.utils.localization  
  
import  
  
get_lang  
,  
  
canonicalize_lang  
,  
  
lang_as_iso639_1  
  
from  
  
polyglot.builtins  
  
import  
  
iteritems  
  
from  
  
polyglot.urlib  
  
import  
  
urlparse
```

```
ns
=
etree
.
FunctionNamespace
(
'calibre_xpath_extensions'
)
```

```
ns
.
prefix
=
'calibre'
```

```
ns
[
'lower-case'
]
```

```
=
lambda
```

```
c
,
```

```
x
:
```

```
x
.
lower
()
```

```
if
```

```
hasattr
(
```

```
x
,

'lower'
)

else

x

class

TOC
:

toc_title

=

None

def

__init__
(
self
,

title
=

None
,

dest
=

None
,
```

```
frag
=
None
):

self
.
title
,

self
.
dest
,

self
.
frag
=

title
,

dest
,

frag

self
.
dest_exists
=

self
.
dest_error
=
```


None

if

self

.

title

:

self

.

title

=

self

.

title

.

strip

()

self

.

parent

=

None

self

.

children

=

[]

```
self
.  
page_list
```

```
=
```

```
[]
```

```
def
```

```
add
```

```
(  
self
```

```
,
```

```
title
```

```
,
```

```
dest
```

```
,
```

```
frag
```

```
=
```

```
None
```

```
):
```

```
  
c
```

```
=
```

```
TOC
```

```
(
```

```
title
```

```
,
```

```
dest
```

```
,
```

```
frag
```

```
)
```

```
self
```

```
.  
children
```

```
.  
append
```

```
(  
c  
)
```

```
c  
.parent
```

```
=
```

```
self
```

```
return
```

```
c
```

```
def
```

```
remove
```

```
(  
self  
,
```

```
child  
):
```

```
self
```

```
.  
children
```

```
.
```

```
remove  
(  
    child  
)
```

```
child  
.  
parent
```

```
=
```

```
None
```

```
def
```

```
remove_from_parent  
(  
    self  
):
```

```
if
```

```
self  
.  
parent
```

```
is
```

```
None  
:
```

```
return
```

```
idx
```

```
=
```

```
self
```

```
.
```

```
parent
```

```
.
```

```
children
```

```
.
```

```
index
```

```
(
```

```
self
```

```
)
```

```
for
```

```
child
```

```
in
```

```
reversed
```

```
(
```

```
self
```

```
.
```

```
children
```

```
):
```

```
child
```

```
.
```

```
parent
```

```
=
```

```
self
```

```
.
```

```
parent
```

```
self
```

```
.
```

```
parent
```

```
.
```

```
children
```

```
.
```

```
insert
```

```
(
```

```
idx
```

```
,
```

```
child
```

```
)
```

```
self
```

```
.
```

```
parent
```

```
.
```

```
children
```

```
.
```

```
remove
```

```
(
```

```
self
```

```
)
```

```
self
```

```
.
```

```
parent
```

```
=
```

```
None
```

```
def
```

```
__iter__
```

```
(
```

```
self
```

```
):
```

```
yield from
```

```
self
```

```
.
```

```
children
```

```
def
```

```
    __len__
```

```
    (
```

```
    self
```

```
):
```

```
    return
```

```
    len
```

```
    (
```

```
    self
```

```
    .
```

```
    children
```

```
)
```

```
def
```

```
    iterdescendants
```

```
    (
```

```
    self
```

```
,
```

```
    level
```

```
    =
```

```
    None
```

```
):
```

```
    gc_level
```

```
    =
```

```
    None
```

```
    if
```

level

is

None

else

level

+

1

for

child

in

self

:

if

level

is

None

:

yield

child

else

:

yield

level

,

child

yield from

child

.

iterdescendants

(

level

=

gc_level

)

def

remove_duplicates

(

self

,

only_text

=

True

):

seen

=

set

()

```
remove
```

```
=
```

```
[]
```

```
for
```

```
child
```

```
in
```

```
self
```

```
:
```

```
key
```

```
=
```

```
child
```

```
.
```

```
title
```

```
if
```

```
only_text
```

```
else
```

```
(
```

```
child
```

```
.
```

```
title
```

```
,
```

```
child
```

```
.
```

```
dest
```

```
,
```

```
(
  child
  .
  frag

  or

  None
))

if

  key

  in

  seen
  :

  remove
  .
  append
  (
    child
  )

  else
  :

  seen
  .
  add
  (
    key
  )

  child
  .
```

```
remove_duplicates  
(
```

```
for
```

```
child
```

```
in
```

```
remove  
:
```

```
self
```

```
.
```

```
remove  
(  
child  
)
```

```
@property
```

```
def
```

```
depth  
(  
self  
):
```

```
"""The maximum depth of the navigation tree rooted at this node."""
```

```
try  
:
```

```
return
```

```
max
(
node
.
depth

for

node

in

self
)

+

1

except

ValueError
:

return

1


@property

def

last_child
(
self
):
```

return

self

.

children

[

-

1

]

if

self

.

children

else

None

def

get_lines

(

self

,

lvl

=

0

):

frag

=

(

'#'

+

self

```
.  
frag  
)
```

```
if
```

```
self
```

```
.  
frag
```

```
else
```

```
"
```

```
ans
```

```
=
```

```
[(  
,
```

```
\t  
,
```

```
*
```

```
lvl  
)
```

```
+
```

```
'TOC:
```

```
%s
```

```
-->
```

```
%s%s
```

```
,
```

```
%
```

```
(
```

```
self
```

```
.
```

```
title
```

```
,
```

```
self
```

```
.
```

```
dest
',

frag
)]

for

child

in

self
:

ans
.
extend
(
child
.
get_lines
(
lvl
+
1
))

return

ans

def

__str__
(
self
):
```



```
return
```

```
,
```

```
\n
```

```
,
```

```
.
```

```
join
```

```
(
```

```
self
```

```
.
```

```
get_lines
```

```
())
```

```
def
```

```
to_dict
```

```
(
```

```
self
```

```
,
```

```
node_counter
```

```
=
```

```
None
```

```
):
```

```
ans
```

```
=
```

```
{
```

```
'title'
```

```
:
```

```
self
```

```
.
```

```
title
```

```
,
```

```
'dest'  
:  
self  
.  
dest  
,  
  
'frag'  
:  
self  
.  
frag  
,  
  
'children'  
:[  
c  
.  
to_dict  
(  
node_counter  
)  
  
for  
  
c  
  
in  
  
self  
.  
children  
]  
  
}  
  
if  
  
self
```

```
.  
dest_exists
```

```
is
```

```
not
```

```
None
```

```
:
```

```
ans
```

```
[  
  'dest_exists'  
]
```

```
=
```

```
self
```

```
.  
dest_exists
```

```
if
```

```
self
```

```
.  
dest_error
```

```
is
```

```
not
```

```
None
```

```
:
```

```
ans
```

```
[  
  'dest_error'  
]
```

```
=
```

```
self
.
dest_error

if

node_counter

is

not

None
:

ans
[
'id'
]

=

next
(
node_counter
)

return

ans

@property

def

as_dict
```

```
(
self
):

return

self
.
to_dict
()

def

child_xpath
(
tag
,
name
):

return

tag
.
xpath
(
'./*[calibre:lower-case(local-name()) = "
%S
"]'
%
name
)

def

add_from_navpoint
```

```
(
  container
  ,
  navpoint
  ,
  parent
  ,
  ncx_name
):
```

```
dest
```

```
=
```

```
frag
```

```
=
```

```
text
```

```
=
```

```
None
```

```
nl
```

```
=
```

```
child_xpath
```

```
(
  navpoint
  ,
```

```
'navlabel'
```

```
)
```

```
if
```

```
nl
:

nl
=

nl
[
0
]

text

=

"

for

txt

in

child_xpath
(
nl
,
'text'
):

text

+=

etree
.
```

```
tostring
(
txt
,

method
=
'text'
,

encoding
=
'unicode'
,

with_tail
=
False
)

content

=

child_xpath
(
navpoint
,

'content'
)

if

content
:

content
```



```
=  
  
content  
[  
0  
]  
  
href  
  
=  
  
content  
.  
get  
(  
'src'  
,  
None  
)  
  
if  
  
href  
:  
  
dest  
  
=  
  
container  
.  
href_to_name  
(  
href  
,  
  
base  
=  
ncx_name
```

```
)  
  
frag  
  
=  
  
urlparse  
(  
href  
)  
.  
fragment
```

or

None

return

```
parent  
.  
add  
(  
text
```

or

None

,

```
dest
```

or

None

,

```
frag
```

or

None

)

def

process_ncx_node

(

container

,

node

,

toc_parent

,

ncx_name

):

for

navpoint

in

node

.

xpath

(

'./*[calibre:lower-case(local-name()) = "navpoint"]'

):

child

=

add_from_navpoint

(

container

```
,  
  
navpoint  
,  
  
toc_parent  
,  
  
ncx_name  
)
```

if

child

is

not

None

:

```
process_ncx_node  
(  
    container  
,  
  
    navpoint  
,  
  
    child  
,  
  
    ncx_name  
)
```

def

parse_ncx

```
(
  container
  ,
  ncx_name
):

root

=

container
.
parsed
(
  ncx_name
)

toc_root

=

TOC
()

navmaps

=

root
.
xpath
(
  '//*[calibre:lower-case(local-name()) = "navmap"]'
)

if

navmaps
```

```
:
```

```
process_ncx_node
```

```
(
```

```
container
```

```
,
```

```
navmaps
```

```
[
```

```
0
```

```
],
```

```
toc_root
```

```
,
```

```
ncx_name
```

```
)
```

```
toc_root
```

```
.
```

```
lang
```

```
=
```

```
toc_root
```

```
.
```

```
uid
```

```
=
```

```
None
```

```
for
```

```
attr
```

```
,
```

```
val
```

```
in
```

```
iteritems
```

```
(  
root
```

```
.  
attrib  
):
```

```
if
```

```
attr
```

```
.  
endswith  
(  
'lang'  
):
```

```
toc_root
```

```
.  
lang
```

```
=
```

```
str  
(  
val  
)
```

```
break
```

```
for
```

```
uid
```

```
in
```

```
root
```

```
.  
xpath
```

```
(  
  '//*[calibre:lower-case(local-name()) = "meta" and @name="dtb:uid"]/@content'  
):
```

```
if
```

```
uid  
:
```

```
toc_root
```

```
.  
uid
```

```
=
```

```
str  
(  
uid  
)
```

```
break
```

```
for
```

```
pl
```

```
in
```

```
root
```

```
.  
xpath  
(  
  '//*[calibre:lower-case(local-name()) = "pagelist"]'  
):
```

```
for
```

```
pt
```


in

pl

.

xpath

(

'descendant::*[calibre:lower-case(local-name()) = "pagetarget"]'

):

pagenum

=

pt

.

get

(

'value'

)

if

pagenum

:

href

=

pt

.

xpath

(

'descendant::*[calibre:lower-case(local-name()) = "content"]/@src'

)

if

```
href
:

dest

=

container
.
href_to_name
(
href
[
0
],
base
=
ncx_name
)

frag

=

urlparse
(
href
[
0
])
.
fragment

or

None

toc_root
.
```

```
page_list
```

```
.
```

```
append
```

```
{
```

```
'dest'
```

```
:
```

```
dest
```

```
,
```

```
'pagenum'
```

```
:
```

```
pagenum
```

```
,
```

```
'frag'
```

```
:
```

```
frag
```

```
})
```

```
return
```

```
toc_root
```

```
def
```

```
add_from_li
```

```
(
```

```
container
```

```
,
```

```
li
```

```
,
```

```
parent
```

```
,
```

```
nav_name
```

```
):  
  
dest  
  
=  
  
frag  
  
=  
  
text  
  
=  
  
None  
  
for  
  
x  
  
in  
  
li  
.  
iterchildren  
(  
XHTML  
(  
'a'  
),  
  
XHTML  
(  
'span'  
)):  
  
text  
  
=
```

```
etree
.
tostring
(
x
,

method
=
'text'
,

encoding
=
'unicode'
,

with_tail
=
False
)
.
strip
()

or

''
.
join
(
x
.
xpath
(
'descendant-or-self::*/@title'
))
.
strip
()

href
```

```
=  
  
x  
.  
get  
(  
    'href'  
)  
  
if  
  
    href  
    :  
  
    dest  
  
    =  
  
    nav_name  
  
    if  
  
        href  
        .  
        startswith  
        (  
            '#'  
        )  
  
    else  
  
        container  
        .  
        href_to_name  
        (  
            href  
        ,  
  
        base  
        =
```

```
nav_name  
)
```

```
frag
```

```
=
```

```
urlparse  
(  
href  
)
```

```
.  
fragment
```

```
or
```

```
None
```

```
break
```

```
return
```

```
parent
```

```
.  
add  
(  
text
```

```
or
```

```
None
```

```
,
```

```
dest
```

```
or
```

```
None
```

```
,
```

```
frag

or

None
)

def

first_child
(
parent
,

tagname
):

try
:

return

next
(
parent
.
iterchildren
(
tagname
))

except

StopIteration
:

return
```


None

def

process_nav_node

(

container

,

node

,

toc_parent

,

nav_name

):

for

li

in

node

.

iterchildren

(

XHTML

(

'li'

)):

child

=

add_from_li

```
(
  container
  ,
  li
  ,
  toc_parent
  ,
  nav_name
)
```

```
ol
```

```
=
```

```
first_child
(
  li
  ,
```

```
XHTML
(
  'ol'
))
```

```
if
```

```
child
```

```
is
```

```
not
```

```
None
```

```
and
```

```
ol
```

is

not

None

:

process_nav_node

(

container

,

ol

,

child

,

nav_name

)

def

parse_nav

(

container

,

nav_name

):

root

=

container

.

parsed

(

```
nav_name  
)
```

```
toc_root
```

```
=
```

```
TOC  
(
```

```
toc_root
```

```
.
```

```
lang
```

```
=
```

```
toc_root
```

```
.
```

```
uid
```

```
=
```

```
None
```

```
et
```

```
=
```

```
'{  
%s  
}type'
```

```
%
```

```
EPUB_NS
```

```
for
```

```
nav
```

in

root

.

iterdescendants

(

XHTML

(

'nav'

)):

if

nav

.

get

(

et

)

==

'toc'

:

ol

=

first_child

(

nav

,

XHTML

(

'ol'

))

```
if
ol
is
not
None
:

process_nav_node
(
container
,
ol
,
toc_root
,
nav_name
)

for
h
in
nav
.iterchildren
(
*
map
(
XHTML
,
```

```
'h1 h2 h3 h4 h5 h6'
```

```
.
```

```
split
```

```
(())):
```

```
text
```

```
=
```

```
etree
```

```
.
```

```
tostring
```

```
(
```

```
h
```

```
,
```

```
method
```

```
=
```

```
'text'
```

```
,
```

```
encoding
```

```
=
```

```
'unicode'
```

```
,
```

```
with_tail
```

```
=
```

```
False
```

```
)
```

```
or
```

```
h
```

```
.
```

```
get
```

```
(
```

```
'title'
```

```
)
```

```
if
```

```
text
:
```

```
toc_root
.
toc_title
```

```
=
```

```
text
```

```
break
```

```
break
```

```
return
```

```
toc_root
```

```
def
```

```
verify_toc_destinations
```

```
(
container
,
```

```
toc
):
```

```
anchor_map
```

```
=
```

```
{}
```


anchor_xpath

=

XPath

(

'//*[@id|//h:a/@name'

)

for

item

in

toc

.

iterdescendants

():

name

=

item

.

dest

if

not

name

:

item

.

dest_exists

```
=
```

```
False
```

```
item
```

```
.
```

```
dest_error
```

```
=
```

```
—  
(
```

```
'No file named
```

```
%s
```

```
exists'
```

```
)
```

```
%
```

```
name
```

```
continue
```

```
try
```

```
:
```

```
root
```

```
=
```

```
container
```

```
.
```

```
parsed
```

```
(
```

```
name
```

```
)
```

```
except
```

```
KeyError
:

item
.
dest_exists

=

False

item
.
dest_error

=

-
(
'No file named
%s
exists'
)
%
name

continue

if

not

hasattr
(
root
,
'xpath'
):
```

item

.

dest_exists

=

False

item

.

dest_error

=

—

(

'No HTML file named

%s

exists'

)

%

name

continue

if

not

item

.

frag

:

item

.

dest_exists

=

True

continue

if

name

not

in

anchor_map

:

anchor_map

[

name

]

=

frozenset

(

anchor_xpath

(

root

))

item

.

dest_exists

=

item

```
.
frag

in

anchor_map
[
name
]

if

not

item

.
dest_exists
:

item

.
dest_error

=

-
(

'The anchor
%(a)s
does not exist in file
%(f)s
'
)
%
dict
(

a
```

```
=
item
.
frag
,

f
=
name
)
```

def

```
find_existing_ncx_toc
(
container
):
```

toc

=

```
container
.
opf_xpath
(
'//opf:spine/@toc'
)
```

if

```
toc
:
```

toc

=

```
container
.
manifest_id_map
.
get
(
toc
[
0
],
None
)

if
not
toc
:

ncx

=

guess_type
(
'a.ncx'
)

toc

=

container
.
manifest_type_map
.
get
(
```



```
ncx
,

[
None
)][
0
]

return

toc

or

None

def

find_existing_nav_toc
(
container
):

for
name
in
container
.manifest_items_with_property
(
'nav'
):

return
```

name

def

get_x_toc

(

container

,

find_toc

,

parse_toc

,

verify_destinations

=

True

):

def

empty_toc

():

ans

=

TOC

()

ans

.

lang

=

ans

.

uid

=

None

return

ans

toc

=

find_toc

(

container

)

ans

=

empty_toc

()

if

toc

is

None

or

not

```
container
.
has_name
(
toc
)

else

parse_toc
(
container
,

toc
)

ans
.
toc_file_name

=

toc

if

toc

and

container
.
has_name
(
toc
)

else

None
```

if

verify_destinations

:

verify_toc_destinations

(

container

,

ans

)

return

ans

def

get_toc

(

container

,

verify_destinations

=

True

):

ver

=

container

.

opf_version_parsed

```
if
```

```
ver
```

```
.
```

```
major
```

```
<
```

```
3
```

```
:
```

```
return
```

```
get_x_toc
```

```
(
```

```
container
```

```
,
```

```
find_existing_ncx_toc
```

```
,
```

```
parse_ncx
```

```
,
```

```
verify_destinations
```

```
=
```

```
verify_destinations
```

```
)
```

```
else
```

```
:
```

```
ans
```

```
=
```

```
get_x_toc
```

```
(
```

```
container
,
find_existing_nav_toc
,
parse_nav
,
verify_destinations
=
verify_destinations
)
```

```
if
```

```
len
(
ans
)
```

```
==
```

```
0
:
```

```
ans
```

```
=
```

```
get_x_toc
(
container
,
```

```
find_existing_ncx_toc
,
```

```
parse_ncx
,
```

```
verify_destinations
=
verify_destinations
)

return

ans

def

get_guide_landmarks
(
container
):

for

ref

in

container
.
opf_xpath
(
'./opf:guide/opf:reference'
):

href
,

title
,

rtype

=
```



```
ref
.  
get  
(  
  'href'  
)  
,
```

```
ref  
.  
get  
(  
  'title'  
)  
,
```

```
ref  
.  
get  
(  
  'type'  
)  
)
```

```
href  
,
```

```
frag
```

```
=
```

```
href  
.  
partition  
(  
  '#'  
)[::  
  2  
]
```

```
name
```

```
=
```

```
container
.  
href_to_name  
(  
href  
,
```

```
container  
.  
opf_name  
)
```

if

```
container  
.  
has_name  
(  
name  
):
```

yield

```
{  
'dest'  
:  
name  
,
```

```
'frag'  
:  
frag  
,
```

```
'title'  
:  
title
```

or

```
"
,

'type'
:
rtype

or

"
}

def

get_nav_landmarks
(
container
):

nav

=

find_existing_nav_toc
(
container
)

if

nav

and

container
.
has_name
(
nav
```

```
):  
  
root  
  
=  
  
container  
.  
parsed  
(  
nav  
)  
  
et  
  
=  
  
'{  
%s  
}type'  
  
%  
  
EPUB_NS  
  
for  
  
elem  
  
in  
  
root  
.  
iterdescendants  
(  
XHTML  
(  
'nav'  
)):
```

```
if
```

```
elem
```

```
.
```

```
get
```

```
(
```

```
et
```

```
)
```

```
==
```

```
'landmarks'
```

```
:
```

```
for
```

```
li
```

```
in
```

```
elem
```

```
.
```

```
iterdescendants
```

```
(
```

```
XHTML
```

```
(
```

```
'li'
```

```
)):
```

```
for
```

```
a
```

```
in
```

```
li
```

```
.
```

```
iterdescendants
```

```
(
```

```
XHTML
```

```
(
    'a'
)):

    href
    ,
    rtype

    =

    a
    .
    get
    (
        'href'
    ),
    a
    .
    get
    (
        et
    )

    if

    href
    :

    title

    =

    etree
    .
    tostring
    (
        a
    ,
```

```
method
```

```
=
```

```
'text'
```

```
,
```

```
encoding
```

```
=
```

```
'unicode'
```

```
,
```

```
with_tail
```

```
=
```

```
False
```

```
)
```

```
.
```

```
strip
```

```
()
```

```
href
```

```
,
```

```
frag
```

```
=
```

```
href
```

```
.
```

```
partition
```

```
(
```

```
'#'
```

```
)[::
```

```
2
```

```
]
```

```
name
```

```
=
```

```
container
```

```
.
```

```
href_to_name
```

```
(
```

```
href
```

```
,
```

```
nav
```

```
)
```

```
if
```

```
container
```

```
.
```

```
has_name
```

```
(
```

```
name
```

```
):
```

```
yield
```

```
{
```

```
'dest'
```

```
:
```

```
name
```

```
,
```

```
'frag'
```

```
:
```

```
frag
```

```
,
```

```
'title'
```

```
:
```

```
title
```

```
or
```

```
"
```

```
,
```

```
'type'
```

```
:
```



```
rtype
```

```
or
```

```
"
```

```
}
```

```
break
```

```
def
```

```
get_landmarks
```

```
(
```

```
container
```

```
):
```

```
ver
```

```
=
```

```
container
```

```
.
```

```
opf_version_parsed
```

```
if
```

```
ver
```

```
.
```

```
major
```

```
<
```

```
3
```

```
:
```

```
return
```

```
list
(
  get_guide_landmarks
  (
    container
  )
)
```

ans

=

```
list
(
  get_nav_landmarks
  (
    container
  )
)
```

if

```
len
(
  ans
)
```

==

```
0
:
```

ans

=

```
list
(
  get_guide_landmarks
  (
    container
  )
)
```

```
return
```

```
ans
```

```
def
```

```
ensure_id
```

```
(  
    elem  
    ,
```

```
    all_ids  
    ):
```

```
    elem_id
```

```
    =
```

```
    elem
```

```
    .  
    get  
    (  
        'id'  
    )
```

```
if
```

```
    elem_id  
    :
```

```
return
```

```
False
```

```
,
```

```
    elem_id
```

```
if
```

```
elem
```

```
.
```

```
tag
```

```
==
```

```
XHTML
```

```
(
```

```
'a'
```

```
):
```

```
anchor
```

```
=
```

```
elem
```

```
.
```

```
get
```

```
(
```

```
'name'
```

```
,
```

```
None
```

```
)
```

```
if
```

```
anchor
```

```
:
```

```
elem
```

```
.
```

```
set
```

```
(
```

```
'id'
```

```
,
```

```
anchor  
)
```

```
return
```

```
False
```

```
,
```

```
anchor
```

```
c
```

```
=
```

```
0
```

```
while
```

```
True
```

```
:
```

```
c
```

```
+=
```

```
1
```

```
q
```

```
=
```

```
f
```

```
'toc_
```

```
{
```

```
c
```

```
}
```

```
,
```

```
if
```

```
q
```

```
not
```

```
in
```

```
all_ids  
:
```

```
elem
```

```
.
```

```
set
```

```
(  
'id'
```

```
,
```

```
q
```

```
)
```

```
all_ids
```

```
.
```

```
add
```

```
(  
q  
)
```

```
break
```

```
return
```

```
True
```

```
,
```

```
elem
```

```
.
```

```
get
(
'id'
)

def

elem_to_toc_text
(
elem
):

text

=

xml2text
(
elem
)
.
strip
()

if

not

text
:

text

=

elem
.
get
```

```
(  
  'title'  
,  
  ''  
)
```

```
if
```

```
not
```

```
text  
:
```

```
text
```

```
=
```

```
elem
```

```
.  
get  
(  
  'alt'  
,  
  ''  
)
```

```
text
```

```
=
```

```
re
```

```
.  
sub  
(  
  r'  
  '\s+'  
,
```



```

    ,

    text

    .
    strip
    ()

    text

    =

    text
    [:
    1000
    ]
    .
    strip
    ()

    if

    not

    text
    :

    text

    =

    -
    (
    '(Untitled)'
    )

    return

    text
```

```
def
```

```
item_at_top
```

```
(  
    elem  
):
```

```
try
```

```
:
```

```
    body
```

```
=
```

```
    XPath
```

```
(  
    '//h:body'  
)(  
    elem
```

```
    .
```

```
    getroottree
```

```
()
```

```
    .
```

```
    getroot
```

```
())[
```

```
    0
```

```
]
```

```
except
```

```
(
```

```
    TypeError
```

```
,
```

```
    IndexError
```

```
,
```

```
KeyError
',

AttributeError
):

return

False

tree

=

body
.
getroottree
()

path

=

tree
.
getpath
(
elem
)

for

el

in

body
.
iterdescendants
```

```
(  
  etree  
  .  
  Element  
):
```

```
  epath
```

```
  =
```

```
  tree  
  .  
  getpath  
  (  
  el  
  )
```

```
  if
```

```
    epath
```

```
    ==
```

```
    path  
    :
```

```
      break
```

```
  try  
  :
```

```
  if
```

```
    el
```

```
    .  
    tag
```

```
    .  
    endswith
```

```
(  
'}img'  
)  
  
or  
  
(  
el  
.  
text  
  
and  
  
el  
.  
text  
.  
strip  
()):  
  
return  
  
False  
  
except  
:  
  
return  
  
False  
  
if  
  
not  
  
path  
.  
startswith  
(
```

```
epath
):

# Only check tail of non-parent elements
```

```
if
```

```
el
.
tail
```

```
and
```

```
el
.
tail
.
strip
():
```

```
return
```

```
False
```

```
return
```

```
True
```

```
\[docs\]
```

```
def
```

```
from_xpaths
(
container
,
```

```
xpaths
):
```

```
'''
```

```
Generate a Table of Contents from a list of XPath expressions. Each  
expression in the list corresponds to a level of the generate ToC. For  
example: :code:`['//h:h1', '//h:h2', '//h:h3']` will generate a three level  
Table of Contents from the ``<h1>``, ``<h2>`` and ``<h3>`` tags.
```

```
'''
```

```
tocroot
```

```
=
```

```
TOC  
()
```

```
xpaths
```

```
=
```

```
[  
XPath  
(  
xp  
)
```

```
for
```

```
xp
```

```
in
```

```
xpaths
```

```
]

# Find those levels that have no elements in all spine items

maps

=

OrderedDict
()

empty_levels

=

{
i
+
1
for

i
,

xp

in

enumerate
(
xpath
s
)}

for

spinepath
```


in

```
container
.  
spine_items  
:
```

name

=

```
container  
.  
abspath_to_name  
(  
spinepath  
)
```

root

=

```
container  
.  
parsed  
(  
name  
)
```

level_item_map

=

```
maps  
[  
name  
]
```

=

```
{
i
+
1
:
frozenset
(
xp
(
root
))

for

i
,

xp

in

enumerate
(
xpath
s
)}

for

lvl
,

elems

in

iteritems
(
level_item_map
):

if
```

```
elems
```

```
:
```

```
empty_levels
```

```
.
```

```
discard
```

```
(
```

```
lvl
```

```
)
```

```
# Remove empty levels from all level_maps
```

```
if
```

```
empty_levels
```

```
:
```

```
for
```

```
name
```

```
,
```

```
lmap
```

```
in
```

```
tuple
```

```
(
```

```
iteritems
```

```
(
```

```
maps
```

```
)):
```

```
lmap
```

```
=
```

```
{
  lvl
:
items

for

  lvl
  ,

  items

in

  iteritems
  (
  lmap
  )

if

  lvl

not

in

  empty_levels
}

lmap

=

sorted
(
  iteritems
  (
  lmap
  ),
  key
```

```
=  
itemgetter  
(  
0  
)
```

lmap

```
=  
  
{  
i  
+  
1  
:  
items
```

for

```
i  
,  
  
(  
l  
,  
  
items  
)
```

in

```
enumerate  
(  
lmap  
)}
```

```
maps  
[  
name  
]
```

```
=
```

```
lmap
```

```
node_level_map
```

```
=
```

```
{  
tocroot  
:
```

```
0  
}
```

```
def
```

```
parent_for_level  
(  
child_level  
):
```

```
limit
```

```
=
```

```
child_level
```

```
-
```

```
1
```

```
def
```

```
process_node  
(
```

```
node
):

child

=

node
.
last_child

if

child

is

None
:

return

node

lvl

=

node_level_map
[
child
]

return

node

if
```

```
lvl
```

```
>
```

```
limit
```

```
else
```

```
child
```

```
if
```

```
lvl
```

```
==
```

```
limit
```

```
else
```

```
process_node
```

```
(
```

```
child
```

```
)
```

```
return
```

```
process_node
```

```
(
```

```
tocroot
```

```
)
```

```
for
```

```
name
```

```
,
```

```
level_item_map
```


in

```
iteritems  
(  
    maps  
):
```

root

=

```
container  
.  
parsed  
(  
    name  
)
```

item_level_map

=

```
{  
e  
:  
i
```

for

```
i  
,
```

elems

in

```
iteritems  
(  
    level_item_map  
)
```

```
for
```

```
e
```

```
in
```

```
elems
```

```
}
```

```
item_dirtied
```

```
=
```

```
False
```

```
all_ids
```

```
=
```

```
set
```

```
(
```

```
root
```

```
.
```

```
xpath
```

```
(
```

```
'//*[@id'
```

```
))
```

```
for
```

```
item
```

```
in
```

```
root
```

```
.
```

```
iterdescendants
```

```
(
```

```
etree
.
Element
):

lvl

=

item_level_map
.
get
(
item
,
None
)

if
lvl
is
None
:

continue

text

=

elem_to_toc_text
(
item
)
```

parent

=

parent_for_level

(

lvl

)

if

item_at_top

(

item

):

dirtied

,

elem_id

=

False

,

None

else

:

dirtied

,

elem_id

=

```
ensure_id
(
item
,
all_ids
)

item_dirtied

=

dirtied

or

item_dirtied

toc

=

parent
.
add
(
text
,
name
,
elem_id
)

node_level_map
[
toc
]
```

```
=  
  
lvl  
  
toc  
.  
dest_exists
```

```
=  
  
True
```

```
if  
  
item_dirtied  
:
```

```
container  
.  
commit_item  
(  
    name  
,  
  
    keep_parsed  
=  
    True  
)
```

```
return
```

```
tocroot
```

```
\[docs\]  
def
```

```
from_links
(
container
):

'''

    Generate a Table of Contents from links in the book.

'''

toc

=

TOC
()

link_path

=

XPath
(
'//h:a[@href]'
)

seen_titles
,

seen_dests

=

set
(),

set
()
```

for

name

,

is_linear

in

container

.

spine_names

:

root

=

container

.

parsed

(

name

)

for

a

in

link_path

(

root

):

href


```
=  
  
a  
.  
get  
(  
    'href'  
)  
  
if  
  
    not  
  
    href  
  
    or  
  
    not  
  
    href  
    .  
    strip  
    ():  
  
    continue  
  
frag  
  
=  
  
None  
  
if  
  
    href  
    .  
    startswith  
    (  
        '#'
```

```
):
```

```
dest
```

```
=
```

```
name
```

```
frag
```

```
=
```

```
href
```

```
[
```

```
1
```

```
:]
```

```
else
```

```
:
```

```
href
```

```
,
```

```
—
```

```
,
```

```
frag
```

```
=
```

```
href
```

```
.
```

```
partition
```

```
(
```

```
'#'
```

```
)
```

```
dest
```

=

container

.

href_to_name

(

href

,

base

=

name

)

frag

=

frag

or

None

if

(

dest

,

frag

)

in

seen_dests

:

continue

```
seen_dests
```

```
.
```

```
add
```

```
((
```

```
dest
```

```
,
```

```
frag
```

```
))
```

```
text
```

```
=
```

```
elem_to_toc_text
```

```
(
```

```
a
```

```
)
```

```
if
```

```
text
```

```
in
```

```
seen_titles
```

```
:
```

```
continue
```

```
seen_titles
```

```
.
```

```
add
```

```
(
```

```
text
```

```
)
```

```
toc
.
add
(
text
,

dest
,

frag
=
frag
)


verify_toc_destinations
(
container
,

toc
)


for

child

in

toc
:

if

not

child
.
dest_exists
```

```
:
```

```
toc
```

```
.  
remove  
(  
child  
)
```

```
return
```

```
toc
```

```
def
```

```
find_text  
(  
node  
):
```

```
LIMIT
```

```
=
```

```
200
```

```
pat
```

```
=
```

```
re  
.compile  
(  
r  
'\s+'  
)
```

```
for
```

```
child
```

```
in
```

```
node
```

```
:
```

```
if
```

```
isinstance
```

```
(
```

```
child
```

```
,
```

```
etree
```

```
.
```

```
_Element
```

```
):
```

```
text
```

```
=
```

```
xml2text
```

```
(
```

```
child
```

```
)
```

```
.
```

```
strip
```

```
()
```

```
text
```

```
=
```

```
pat
```

```
.
```

```
sub
(
    ,
text
)

if

len
(
text
)

<

1
:

continue

if

len
(
text
)

>

LIMIT
:

# Look for less text in a child of this node, recursively

ntext
```



```
return
n\text
```

```
[docs]  
def  
  
from_files
```

```
container
):

'''
    Generate a Table of Contents from files in the book.
'''

toc

=

TOC
()

for
i
,
spinepath

in

enumerate
(
container
.
spine_items
):

name

=

container
.
abspath_to_name
```

```
(  
  spinepath  
)
```

```
root
```

```
=
```

```
container
```

```
.  
parsed  
(  
  name  
)
```

```
body
```

```
=
```

```
XPath  
(  
  '//h:body'  
)(  
  root  
)
```

```
if
```

```
not
```

```
body  
:
```

```
continue
```

```
text
```

```
=
```

```
find_text
```

```
(
```

```
body
```

```
[
```

```
0
```

```
])
```

```
if
```

```
not
```

```
text
```

```
:
```

```
text
```

```
=
```

```
name
```

```
.
```

```
rpartition
```

```
(
```

```
'/'
```

```
)[
```

```
-
```

```
1
```

```
]
```

```
if
```

```
i
```

```
==
```

```
0
```

```
and
```

```
text
```

```
.
rpartition
(
    ' '
)[
    0
]
.
lower
()

in

{
    'titlepage'
,
    'cover'
}:

text

=

-
(
    'Cover'
)

toc

.
add
(
    text
,
    name
)

return
```

toc

def

node_from_loc

(
root
,

locs
,

totals
=

None
):

node

=

root

.

xpath

(
'//*[local-name()="body"]'
)[
0
]

for

i

,

loc

in

```
enumerate
```

```
(  
locs  
):
```

```
children
```

```
=
```

```
tuple
```

```
(  
node  
.  
iterchildren  
(  
etree  
.  
Element  
)  
)
```

```
if
```

```
totals
```

```
is
```

```
not
```

```
None
```

```
and
```

```
totals
```

```
[  
i  
]
```

```
!=
```

```
len
```

```
(
```

```
children
):
```

```
raise
```

```
MalformedMarkup
()
```

```
node
```

```
=
```

```
children
[
loc
]
```

```
return
```

```
node
```

```
def
```

```
add_id
(
container
,
```

```
name
,
```

```
loc
,
```

```
totals
=
```

```
None
):
```



```
root
```

```
=
```

```
container
```

```
.
```

```
parsed
```

```
(
```

```
name
```

```
)
```

```
try
```

```
:
```

```
node
```

```
=
```

```
node_from_loc
```

```
(
```

```
root
```

```
,
```

```
loc
```

```
,
```

```
totals
```

```
=
```

```
totals
```

```
)
```

```
except
```

```
MalformedMarkup
```

```
:
```

```
# The webkit HTML parser and the container parser have yielded
```

different node counts, this can happen if the file is valid XML

but contains constructs like nested <p> tags. So force parse it

with the HTML 5 parser and try again.

raw

=

container

.

raw_data

(

name

)

root

=

container

.

parse_xhtml

(

raw

,

fname

=

name

,

force_html5_parse

=

True

)

```
try
:

node

=

node_from_loc
(
root
,

loc
,

totals
=
totals
)

except

MalformedMarkup
:

raise

MalformedMarkup
(
-
(
'The file
%s
has malformed markup. Try running the Fix HTML tool'

' before editing.'
)
```

```
%
```

```
name  
)
```

```
container
```

```
.  
replace  
(  
name  
,
```

```
root  
)
```

```
if
```

```
not
```

```
node
```

```
.  
get  
(  
'id'  
):
```

```
ensure_id  
(  
node  
,
```

```
set  
(  
root
```

```
.  
xpath  
(  
'//*[@id'
```

```
)))

container
.
commit_item
(
name
,

keep_parsed
=
True
)

return

node
.
get
(
'id'
)

def

create_ncx
(
toc
,

to_href
,

btitle
,

lang
,
```

```
uid
):
```

```
lang
```

```
=
```

```
lang
```

```
.  
replace  
(  
  ' _'  
  ,  
  ' _'  
)
```

```
ncx
```

```
=
```

```
etree
```

```
.  
Element  
(  
  NCX  
(  
    'ncx'  
  ),
```

```
attrib
```

```
=
```

```
{  
  'version'  
  :  
    '2005-1'  
  ,
```

```
XML
```

```
(  
  'lang'  
):  
  
  lang  
},  
  
  nsmap  
=  
  {  
    None  
  :  
  
    NCX_NS  
  })  
  
  head  
  
  =  
  
  etree  
  .  
  SubElement  
  (  
    ncx  
  ,  
  
    NCX  
    (  
      'head'  
    ))  
  
  etree  
  .  
  SubElement  
  (  
    head  
  ,  
  
    NCX
```

```
(
  'meta'
),

name
=
'dtb:uid'
,

content
=
str
(
  uid
))

etree
.
SubElement
(
  head
,

  NCX
  (
    'meta'
  ),

  name
  =
  'dtb:depth'
,

  content
  =
  str
  (
    toc
    .
    depth
```



```
))
```

```
generator
```

```
=
```

```
"
```

```
.
```

```
join
```

```
([
```

```
'calibre ('
```

```
,
```

```
__version__
```

```
,
```

```
')
```

```
])
```

```
etree
```

```
.
```

```
SubElement
```

```
(
```

```
head
```

```
,
```

```
NCX
```

```
(
```

```
'meta'
```

```
),
```

```
name
```

```
=
```

```
'dtb:generator'
```

```
,
```

```
content
```

```
=
```

```
generator
```

```
)
```

```
etree
.
SubElement
(
head
,

NCX
(
'meta'
),

name
=
'dtb:totalPageCount'
,

content
=
'0'
)
```

```
etree
.
SubElement
(
head
,

NCX
(
'meta'
),

name
=
'dtb:maxPageNumber'
,

content
```

```
=  
'0'  
)
```

title

```
=
```

etree

```
.  
SubElement  
(  
ncx  
,
```

```
NCX  
(  
'docTitle'  
)
```

text

```
=
```

etree

```
.  
SubElement  
(  
title  
,
```

```
NCX  
(  
'text'  
)
```

text

```
.  
text
```

```
=  
  
btitle  
  
navmap  
  
=  
  
etree  
.  
SubElement  
(  
ncx  
,  
  
NCX  
(  
'navMap'  
)  
  
spat  
  
=  
  
re  
.  
compile  
(  
r  
'\s+'  
)  
  
play_order  
  
=  
  
Counter  
(
```

def

process_node

(

xml_parent

,

toc_parent

):

for

child

in

toc_parent

:

play_order

[

'c'

]

+=

1

point

=

etree

.

SubElement

(

xml_parent

,

```
NCX
(
'navPoint'
),

id
=
'num_
%d
'

%

play_order
[
'c'
],

playOrder
=
str
(
play_order
[
'c'
]))

label

=

etree
.
SubElement
(
point
,

NCX
(
```

```
'navLabel'
```

```
))
```

```
title
```

```
=
```

```
child
```

```
.
```

```
title
```

```
if
```

```
title
```

```
:
```

```
title
```

```
=
```

```
spat
```

```
.
```

```
sub
```

```
(
```

```
' '
```

```
,
```

```
title
```

```
)
```

```
etree
```

```
.
```

```
SubElement
```

```
(
```

```
label
```

```
,
```

```
NCX
```

```
(
```

```
'text'
```

```
))
```

```
.
```

```
text
```

```
=
```

```
title
```

```
if
```

```
child
```

```
.
```

```
dest
```

```
:
```

```
href
```

```
=
```

```
to_href
```

```
(
```

```
child
```

```
.
```

```
dest
```

```
)
```

```
if
```

```
child
```

```
.
```

```
frag
```

```
:
```

```
href
```

```
+=
```

```
'#'
```



```
+
child
.
frag

etree
.
SubElement
(
point
,

NCX
(
'content'
),

src
=
href
)

process_node
(
point
,

child
)

process_node
(
navmap
,

toc
)
```

return

ncx

def

commit_ncx_toc

(

container

,

toc

,

lang

=

None

,

uid

=

None

):

tocname

=

find_existing_ncx_toc

(

container

)

if

tocname

is

None

:

item

=

container

.

generate_item

(

'toc.ncx'

,

id_prefix

=

'toc'

)

tocname

=

container

.

href_to_name

(

item

.

get

(

'href'

),

base

=

container

.

opf_name

)

```
ncx_id
=
item
.
get
(
'id'
)

[
s
.
set
(
'toc'
,
ncx_id
)

for

s

in

container
.
opf_xpath
(
'//opf:spine'
)]

if

not

lang
```

```
:  
  
lang  
  
=  
  
get_lang  
(  
  
for  
  
|  
  
in  
  
container  
.  
opf_xpath  
(  
    '//dc:language'  
):  
  
|  
  
=  
  
canonicalize_lang  
(  
    xml2text  
(  
        |  
    )  
    .  
    strip  
())  
  
if  
  
|
```

:

lang

=

|

lang

=

lang_as_iso639_1

(

|

)

or

|

break

lang

=

lang_as_iso639_1

(

lang

)

or

lang

if

not

uid
:

uid

=

uuid_id
(

eid

=

container

.

opf

.

get

(

'unique-identifier'

,

None

)

if

eid

:

m

=

container

.

```
opf_xpath
```

```
(  
  '//*[@id="  
  %s  
  "]'  
  %  
  eid  
)
```

```
if
```

```
m  
:
```

```
uid
```

```
=
```

```
xml2text
```

```
(  
  m  
  [  
  0  
  ])
```

```
title
```

```
=
```

```
-  
(  
  'Table of Contents'  
)
```

```
m
```

```
=
```



```
container
```

```
.
```

```
opf_xpath
```

```
(
```

```
'//dc:title'
```

```
)
```

```
if
```

```
m
```

```
:
```

```
x
```

```
=
```

```
xml2text
```

```
(
```

```
m
```

```
[
```

```
0
```

```
])
```

```
.
```

```
strip
```

```
()
```

```
title
```

```
=
```

```
x
```

```
or
```

```
title
```

```
to_href
```

```
=  
  
partial  
(  
  container  
  .  
  name_to_href  
  ,  
  
  base  
  =  
  tocname  
)
```

```
root
```

```
=  
  
create_ncx  
(  
  toc  
  ,  
  
  to_href  
  ,  
  
  title  
  ,  
  
  lang  
  ,  
  
  uid  
)
```

```
container  
.  
replace  
(  
  tocname  
  ,
```

```
root
)
```

```
container
.
pretty_print
```

```
.
add
(
tocname
)
```

```
def
```

```
ensure_single_nav_of_type
(
root
,
ntype
=
'toc'
):
```

```
et
```

```
=
```

```
'{
%s
}type'
```

```
%
```

```
EPUB_NS
```

```
navs
```

```
=  
  
[  
n  
  
for  
  
n  
  
in  
  
root  
.  
iterdescendants  
(  
XHTML  
(  
'nav'  
))  
  
if  
  
n  
.  
get  
(  
et  
)  
  
==  
  
ntype  
]  
  
for  
  
x  
  
in  
  
navs
```

```
[
  1
]:

extract
(
  x
)

if

navs
:

nav

=

navs
[
  0
]

tail

=

nav
.
tail

attrib

=

dict
(
  nav
```

```
.
attrib
)

nav

.
clear
()

nav

.
attrib
.
update
(
  attrib
)

nav

.
tail

=

tail

else
:

nav

=

root

.
makeelement
(
  XHTML
```

```
(  
'nav'  
)
```

```
first_child  
(  
root  
,
```

```
XHTML  
(  
'body'  
)
```

```
.  
append  
(  
nav  
)
```

```
nav  
.  
set  
(  
{  
%s  
}type'  
  
%
```

```
EPUB_NS  
,  
  
ntype  
)
```

```
return
```

```
nav
```

```
def
ensure_container_has_nav
(
container
,
lang
=
None
,
previous_nav
=
None
):

tocname
=

find_existing_nav_toc
(
container
)

if
previous_nav
is
not
None
:

nav_name
```



```
=  
  
container  
.  
href_to_name  
(  
previous_nav  
[  
0  
)  
  
if  
  
nav_name  
  
and  
  
container  
.  
exists  
(  
nav_name  
):  
  
tocname  
  
=  
  
nav_name  
  
container  
.  
apply_unique_properties  
(  
tocname  
,  
'nav'  
)
```

if

tocname

is

None

:

item

=

container

.

generate_item

(

'nav.xhtml'

,

id_prefix

=

'nav'

)

item

.

set

(

'properties'

,

'nav'

)

tocname

=

```
container
.
href_to_name
(
item
.
get
(
'href'
),

base
=
container
.
opf_name
)
```

```
if
```

```
previous_nav
```

```
is
```

```
not
```

```
None
```

```
:
```

```
root
```

```
=
```

```
previous_nav
```

```
[
```

```
1
```

```
]
```

```
else
```

```
:
```

```
root
=
container
.
parse_xhtml
(
P
(
'templates/new_nav.html'
,
data
=
True
)
.
decode
(
'utf-8'
))

container
.
replace
(
tocname
,
root
)

else
:

root
```

```
=  
  
container  
.  
parsed  
(  
  tocname  
)  
  
if  
  
  lang  
  :  
  
    lang  
  
    =  
  
    lang_as_iso639_1  
    (  
      lang  
    )  
  
    or  
  
    lang  
  
    root  
    .  
    set  
    (  
      'lang'  
    ,  
    lang  
    )  
  
    root  
    .
```

```
set
(
  '{
    %s
  }lang'

  %

  XML_NS
  ,

  lang
)

return

tocname
,

root

def

collapse_li
(
  parent
):

for

li

in

parent
.
iterdescendants
(
  XHTML
```

```
(  
'li'  
)):
```

```
if
```

```
len
```

```
(  
li  
)
```

```
==
```

```
1  
:
```

```
li
```

```
.  
text
```

```
=
```

```
None
```

```
li
```

```
[  
0  
]
```

```
.  
tail
```

```
=
```

```
None
```

```
def
```

```
create_nav_li
```

```
(
  container
  ,
  ol
  ,
  entry
  ,
  tocname
):

  li

  =

  ol
  .
  makeelement
  (
    XHTML
    (
      'li'
    )
  )

  ol
  .
  append
  (
    li
  )

  a

  =

  li
  .
  makeelement
```



```
(
XHTML
(
'a'
))

li
.
append
(
a
)

href

=

container
.
name_to_href
(
entry
[
'dest'
],
tocname
)

if

entry
[
'frag'
]:

href

+=
```

```
'#'
```

```
+
```

```
entry
```

```
[
```

```
'frag'
```

```
]
```

```
a
```

```
.
```

```
set
```

```
(
```

```
'href'
```

```
,
```

```
href
```

```
)
```

```
return
```

```
a
```

```
def
```

```
set_landmarks
```

```
(
```

```
container
```

```
,
```

```
root
```

```
,
```

```
tocname
```

```
,
```

```
landmarks
```

```
):
```

```
nav
```

```
=
```

```
ensure_single_nav_of_type
```

```
(
```

```
root
```

```
,
```

```
'landmarks'
```

```
)
```

```
nav
```

```
.
```

```
set
```

```
(
```

```
'hidden'
```

```
,
```

```
"
```

```
)
```

```
ol
```

```
=
```

```
nav
```

```
.
```

```
makeelement
```

```
(
```

```
XHTML
```

```
(
```

```
'ol'
```

```
))
```

```
nav
```

```
.
```

```
append
```

```
(  
ol  
)  
  
for  
  
entry  
  
in  
  
landmarks  
:  
  
if  
  
entry  
[  
  'type'  
)  
  
and  
  
container  
.  
has_name  
(  
entry  
[  
  'dest'  
)  
  
and  
  
container  
.  
mime_map  
[  
entry  
[  
  'dest'  
]]
```

in

OEB_DOCS

:

a

=

create_nav_li

(

container

,

ol

,

entry

,

tocname

)

a

.

set

(

{

%s

}type'

%

EPUB_NS

,

entry

[

'type'

])

```
a
.
text
```

```
=
```

```
entry
[
'title'
]
```

```
or
```

```
None
```

```
pretty_xml_tree
(
nav
)
```

```
collapse_li
(
nav
)
```

```
def
```

```
commit_nav_toc
(
container
,
```

```
toc
,
```

```
lang
=
```

None

,

landmarks

=

None

,

previous_nav

=

None

):

tocname

,

root

=

ensure_container_has_nav

(

container

,

lang

=

lang

,

previous_nav

=

previous_nav

)

nav

=

ensure_single_nav_of_type

(

```
root
,

'toc'
)

if

toc

.
toc_title
:

nav

.
append
(
nav

.
makeelement
(
XHTML
(
'h1'
)))

nav
[
-
1
]

.
text

=

toc

.
toc_title
```



```
rnode
```

```
=
```

```
nav
```

```
.
```

```
makeelement
```

```
(
```

```
XHTML
```

```
(
```

```
'ol'
```

```
))
```

```
nav
```

```
.
```

```
append
```

```
(
```

```
rnode
```

```
)
```

```
to_href
```

```
=
```

```
partial
```

```
(
```

```
container
```

```
.
```

```
name_to_href
```

```
,
```

```
base
```

```
=
```

```
tocname
```

```
)
```

```
spat
```

```
=  
  
re  
.  
compile  
(  
r  
'\s+'  
)  
  
def  
  
process_node  
(  
xml_parent  
,  
  
toc_parent  
):  
  
for  
  
child  
  
in  
  
toc_parent  
:  
  
li  
  
=  
  
xml_parent  
.  
makeelement  
(  
XHTML  
(
```

```
'li'  
))  
  
xml_parent  
.  
append  
(  
li  
)
```

title

=

child
.
title

or

"

title

=

spat
.
sub
(
" "
,

title
)

strip
(

```
a
=
li
.
makeelement
(
XHTML
(
'a'
if
child
.
dest
else
'span'
))
```

```
a
.
text
=
title
li
.
append
(
a
)
if
```

```
child
.
dest
:

href

=

to_href
(
child
.
dest
)
```

if

```
child
.
frag
:

href

+=

'#'
+
child
.
frag
```

```
a
.
set
(
'href'
,
```

```
href  
)
```

```
if
```

```
len  
(  
child  
):
```

```
ol
```

```
=
```

```
li  
.  
makeelement  
(  
XHTML  
(  
'ol'  
))
```

```
li  
.  
append  
(  
ol  
)
```

```
process_node  
(  
ol  
,  
child  
)
```

```
process_node
```

```
(
```

```
  rnode
```

```
,
```

```
  toc
```

```
)
```

```
pretty_xml_tree
```

```
(
```

```
  nav
```

```
)
```

```
collapse_li
```

```
(
```

```
  nav
```

```
)
```

```
nav
```

```
.
```

```
tail
```

```
=
```

```
,
```

```
  \n
```

```
,
```

```
if
```

```
toc
```

```
.
```

```
page_list
```

```
:
```

```
nav
=
ensure_single_nav_of_type
(
root
,
'page-list'
)
```

```
nav
.
set
(
'hidden'
,
"
)
)
```

```
ol
=
nav
.
makeelement
(
XHTML
(
'ol'
))
```

```
nav
.
append
(
ol
```



```
)  
  
for  
entry  
in  
toc  
.page_list  
:  
  
if  
container  
.has_name  
(  
entry  
[  
'dest'  
])  
  
and  
container  
.mime_map  
[  
entry  
[  
'dest'  
]]  
  
in  
OEB_DOCS  
:
```

```
a
=
create_nav_li
(
container
,
ol
,
entry
,
tocname
)

a
.
text
=
str
(
entry
[
'pagenum'
])

pretty_xml_tree
(
nav
)

collapse_li
(
nav
)
```

```
container
```

```
.
```

```
replace
```

```
(
```

```
tocname
```

```
,
```

```
root
```

```
)
```

```
def
```

```
commit_toc
```

```
(
```

```
container
```

```
,
```

```
toc
```

```
,
```

```
lang
```

```
=
```

```
None
```

```
,
```

```
uid
```

```
=
```

```
None
```

```
):
```

```
commit_ncx_toc
```

```
(
```

```
container
```

```
,
```

```
toc
```

```
,
```

```
lang
=
lang
,

uid
=
uid
)

if

container
.
opf_version_parsed
.
major

>

2
:

commit_nav_toc
(
container
,

toc
,

lang
=
lang
)

def

remove_names_from_toc
```

```
(
container
,
names
):

changed

=

[]

names

=

frozenset
(
names
)

for

find_toc
,

parse_toc
,

commit_toc

in

(

(
find_existing_ncx_toc
,
```

```
parse_ncx
,

commit_ncx_toc
),

(
find_existing_nav_toc
,

parse_nav
,

commit_nav_toc
),

):
```

```
toc
```

```
=
```

```
get_x_toc
(
container
,
```

```
find_toc
,
```

```
parse_toc
,
```

```
verify_destinations
=
```

```
False
```

```
)
```

```
if
len
(
toc
)

>

0
:

remove

=

[]

for
node
in
toc
.
iterdescendants
():

if
node
.
dest
in
names
:
```

```
remove
```

```
.
```

```
append
```

```
(
```

```
node
```

```
)
```

```
if
```

```
remove
```

```
:
```

```
for
```

```
node
```

```
in
```

```
reversed
```

```
(
```

```
remove
```

```
):
```

```
node
```

```
.
```

```
remove_from_parent
```

```
()
```

```
commit_toc
```

```
(
```

```
container
```

```
,
```

```
toc
```

```
)
```

```
changed
```



```
.
append
(
  find_toc
  (
    container
  )
)

return

changed

def

find_inline_toc
(
  container
):

for

name
,

linear

in

container
.
spine_names
:

if

container
.
parsed
```

```
(
name
)
.
xpath
(
'//*[local-name()="body" and @id="calibre_generated_inline_toc"]'
):
```

return

name

def

toc_to_html

```
(
toc
```

,

container

,

toc_name

,

title

,

lang

=

None

):

def

process_node

```
(
```

```
html_parent
```

```
,
```

```
toc
```

```
,
```

```
level
```

```
=
```

```
1
```

```
,
```

```
indent
```

```
=
```

```
' '
```

```
,
```

```
style_level
```

```
=
```

```
2
```

```
):
```

```
li
```

```
=
```

```
html_parent
```

```
.
```

```
makeelement
```

```
(
```

```
XHTML
```

```
(
```

```
'li'
```

```
))
```

```
li
```

```
.
```

```
tail
```

```
=
```

```
'
```

```
\n
',
+

(
indent
*
level
)

html_parent
.
append
(
li
)

name
,

frag

=

toc
.
dest
,

toc
.
frag

href

=

'#'
```

```
if
name
:

href

=

container
.
name_to_href
(
name
,
toc_name
)
```

```
if
frag
:

href
```

```
+=
```

```
'#'
```

```
+
```

```
frag
```

```
a
```

```
=
```

```
li
```

```
.  
makeelement
```

```
(  
XHTML  
(  
'a'  
),
```

```
href  
=  
href  
)
```

```
  
a  
.  
text
```

```
=  
  
toc
```

```
.  
title
```

```
  
li  
.  
append  
(  
a  
)
```

```
if
```

```
len  
(  
toc  
)
```

```
>
```

```
0
```

```
:  
  
parent  
  
=  
  
li  
.  
makeelement  
(  
XHTML  
(  
'ul'  
)  
)  
  
parent  
.  
set  
(  
'class'  
,  
  
'level  
%d'  
,  
  
%d  
  
(  
style_level  
)  
)  
  
li  
.  
append  
(  
parent  
)
```

```
a
.
tail

=

'
\n\n
'

+

(
indent
*
(
level
+
2
))

parent
.
text

=

'
\n
'

+
(
indent
*
(
level
+
3
))

parent
```



```
.
tail

=

'
\n\n
'

+

(
indent
*
(
level
+
1
))

for
child
in
toc
:

process_node
(
parent
,
child
,
level
+
3
,
```

```
style_level
```

```
=
```

```
style_level
```

```
+
```

```
1
```

```
)
```

```
parent
```

```
[
```

```
-
```

```
1
```

```
]
```

```
.
```

```
tail
```

```
=
```

```
,
```

```
\n
```

```
,
```

```
+
```

```
(
```

```
indent
```

```
*
```

```
(
```

```
level
```

```
+
```

```
2
```

```
))
```

```
E
```

```
=
```

```
ElementMaker
```

```
(
  namespace
  =
  XHTML_NS
,

  nsmap
  =
  {
    None
    :
    XHTML_NS
  })
```

```
html
```

```
=
```

```
E
```

```
.
```

```
html
```

```
(
```

```
E
```

```
.
```

```
head
```

```
(
```

```
E
```

```
.
```

```
title
```

```
(
```

```
title
```

```
),
```

```
E
```

```
.
```

```
style
```

```
(
```

```
P
(
'templates/inline_toc_styles.css'
,
```

```
data
=
True
)
.
decode
(
'utf-8'
),

type
=
'text/css'
),
```

```
),
```

```
E
.
body
(
```

```
E
.
h2
(
title
),
```

```
E
.
ul
(),
```

```
id
=
"calibre_generated_inline_toc"
,

)

)
```

```
ul

=

html
[
1
][
1
]

ul
.
set
(
'class'
,

'level1'
)
```

```
for

child

in
```

```
toc
:

process_node
(
ul
,
child
)

if

lang
:

html
.
set
(
'lang'
,
lang
)

pretty_html_tree
(
container
,
html
)

return

html
```

[\[docs\]](#)

def

create_inline_toc

(

container

,

title

=

None

):

'''

Create an inline (HTML) Table of Contents from an existing NCX Table of Contents.

:param title: The title for this table of contents.

'''

lang

=

get_book_language

(

container

)

default_title

=

'Table of Contents'

if

lang
:

lang

=

lang_as_iso639_1
(
lang
)

or

lang

default_title

=

translate
(
lang
,

default_title
)

title

=

title

or


```
default_title
```

```
toc
```

```
=
```

```
get_toc
```

```
(  
    container  
)
```

```
if
```

```
len
```

```
(  
    toc  
)
```

```
==
```

```
0
```

```
:
```

```
return
```

```
None
```

```
toc_name
```

```
=
```

```
find_inline_toc
```

```
(  
    container  
)
```

```
name
```

```
=
```

```
toc_name
```

```
html
```

```
=
```

```
toc_to_html
```

```
(
```

```
toc
```

```
,
```

```
container
```

```
,
```

```
name
```

```
,
```

```
title
```

```
,
```

```
lang
```

```
)
```

```
raw
```

```
=
```

```
serialize
```

```
(
```

```
html
```

```
,
```

```
'text/html'
```

```
)
```

```
if
```

```
name
is
None
:

name
,
c
=
'toc.xhtml'
,
0

while

container
.
has_name
(
name
):

c
+=
1

name
=
'toc
```

```
%d
.html'

%

c

container
.
add_file
(
name
,

raw
,

spine_index
=
0
)

else
:

with

container
.
open
(
name
,

'wb'
)

as

f
```

```
:  
  
f  
.  
write  
(  
raw  
)  
  
set_guide_item  
(  
container  
,  
  
'toc'  
,  
  
title  
,  
  
name  
,  
  
frag  
=  
'calibre_generated_inline_toc'  
)  
  
return  
  
name
```

Source code for calibre.gui2.actions

```
#!/usr/bin/env python
```

```
__license__
```

```
=
```

```
'GPL v3'
```

```
__copyright__
```

```
=
```

```
'2010, Kovid Goyal <kovid@kovidgoyal.net>'
```

```
__docformat__
```

```
=
```

```
'restructuredtext en'
```

```
from
```

```
functools
```

```
import
```

```
partial
```

```
from
```

```
zipfile
```

```
import
```

```
ZipFile
```

```
from
```

```
qt.core
```

```
import
```

```
(  
QToolButton  
,
```

```
QAction  
,
```

```
QIcon  
,
```

```
QObject  
,
```

```
QMenu  
,
```

```
QKeySequence  
)
```

```
from
```

```
calibre
```

```
import
```

```
prints
```

```
from
```

```
calibre.constants
```

```
import
```

```
ismacos
```

```
from
```

```
calibre.gui2
```

```
import
```

```
Dispatcher
```

```
from
```

```
calibre.gui2.keyboard
```

```
import
```

```
NameConflict
```

```
from
```

```
polyglot.builtins
```

```
import
```

```
string_or_bytes
```

```
def
```

```
menu_action_unique_name
```

```
(
```

```
plugin
```

```
,
```

```
unique_name
```

```
):
```

```
return
```



```
,
%s
: menu action :
%s
'
%
(
plugin
.
unique_name
,
unique_name
)
```

[\[docs\]](#)

class

InterfaceAction

```
(
QObject
):
```

```
'''
```

A plugin representing an "action" that can be taken in the graphical user interface. All the items in the toolbar and context menus are implemented by these plugins.

Note that this class is the base class for these plugins, however, to integrate the plugin with calibre's plugin system, you have to make a

wrapper class that references the actual plugin. See the

`:mod: `calibre.customize.builtins`` module for examples.

If two `:class: `InterfaceAction`` objects have the same name, the one with higher priority takes precedence.

Sub-classes should implement the `:meth: `genesis``, `:meth: `library_changed``,
`:meth: `location_selected``, `:meth: `shutting_down``,
`:meth: `initialization_complete`` and `:meth: `tag_browser_context_action`` methods.

Once initialized, this plugin has access to the main calibre GUI via the
`:attr: `gui`` member. You can access other plugins by name, for example::

```
self.gui.iactions['Save To Disk']
```

To access the actual plugin, use the `:attr: `interface_action_base_plugin`` attribute, this attribute only becomes available after the plugin has been initialized. Useful if you want to use methods from the plugin class like `do_user_config()`.

The `QAction` specified by `:attr: `action_spec`` is automatically create and made available as `self.qaction``.

'''

#: The plugin name. If two plugins with the same name are present, the one

#: with higher priority takes precedence.

name

=

'Implement me'

#: The plugin priority. If two plugins with the same name are present, the one

#: with higher priority takes precedence.

priority

=

1

#: The menu popup type for when this plugin is added to a toolbar

popup_type

=

QToolButton

.

ToolButtonPopupMode

.

MenuButtonPopup

#: Whether this action should be auto repeated when its shortcut

#: key is held down.

auto_repeat

=

False

#: Of the form: (text, icon_path, tooltip, keyboard shortcut)

#: icon, tooltip and keyboard shortcut can be None

#: shortcut must be a string, None or tuple of shortcuts.

#: If None, a keyboard shortcut corresponding to the action is not

#: registered. If you pass an empty tuple, then the shortcut is registered

#: with no default key binding.

action_spec

=

(
'text'
,

'icon'
,

None

,

None

)

#: If True, a menu is automatically created and added to self.qaction

action_add_menu

=

False

#: If True, a clone of self.qaction is added to the menu of self.qaction

#: If you want the text of this action to be different from that of

#: self.qaction, set this variable to the new text

action_menu_clone_qaction

=

False

#: Set of locations to which this action must not be added.

#: See :attr:`all\_locations` for a list of possible locations

dont_add_to

```
=
```

```
frozenset  
()
```

```
#: Set of locations from which this action must not be removed.
```

```
#: See :attr:`all_locations` for a list of possible locations
```

```
dont_remove_from
```

```
=
```

```
frozenset  
()
```

```
all_locations
```

```
=
```

```
frozenset  
([  
    'toolbar'  
,  
    'toolbar-device'  
,  
    'context-menu'  
,  
    'context-menu-device'  
,  
    'toolbar-child'
```

```
,  
  
'menubar'  
,  
  
'menubar-device'  
,  
  
'context-menu-cover-browser'  
,  
  
'context-menu-split'  
)
```

#: Type of action

#: 'current' means acts on the current view

#: 'global' means an action that does not act on the current view, but rather

#: on calibre as a whole

action_type

=

'global'

#: If True, then this InterfaceAction will have the opportunity to interact

#: with drag and drop events. See the methods, :meth:`accept\_enter\_event`,

#: :meth:`accept\_drag\_move\_event`, :meth:`drop\_event` for details.

accepts_drops

=

False

def

__init__

(

self

,

parent

,

site_customization

):

QObject

.

__init__

(

self

,

parent

)

self

.

setObjectName

(

self

.

name


```
)

self
.
gui
=
parent

self
.
site_customization
=
site_customization

self
.
interface_action_base_plugin
=
None

[docs]

def
accept_enter_event
(
self
,
event
,
```

```
mime_data
):
```

```
''' This method should return True iff this interface action is capable
      of handling the drag event. Do not call accept/ignore on the event,
      that will be taken care of by the calibre UI.'''
```

```
return
```

```
False
```

```
\[docs\]
```

```
def
```

```
accept_drag_move_event
(
    self
    ,
    event
    ,
    mime_data
):
```

```
''' This method should return True iff this interface action is capable
      of handling the drag event. Do not call accept/ignore on the event,
      that will be taken care of by the calibre UI.'''
```

```
return
```

False

[\[docs\]](#)

def

drop_event

(
self

,

event

,

mime_data

):

''' This method should perform some useful action and return True

iff this interface action is capable of handling the drop event. Do not

call accept/ignore on the event, that will be taken care of by the

calibre UI. You should not perform blocking/long operations in this

function. Instead emit a signal or use QTimer.singleShot and return

quickly. See the builtin actions for examples.'''

return

False

def

do_genesis

(

self

):

```
self
.
Dispatcher

=

partial
(
Dispatcher
,

parent
=
self
)

self
.
create_action
()

self
.
gui
.
addAction
(
self
.
qaction
)

self
.
gui
.
addAction
(
```

```
self
.  
menuless_qaction  
)
```

```
self  
.  
genesis  
(
```

```
self  
.  
location_selected  
(  
'library'  
)
```

```
@property
```

```
def  
  
unique_name  
(  
self  
):
```

```
bn
```

```
=
```

```
self  
.  
__class__  
.  
__name__
```

```
if

getattr
(
self
.
interface_action_base_plugin
,

'name'
):

bn

=

self
.
interface_action_base_plugin
.
name

return

'Interface Action:
%s
(
%s
)'
%
(
bn
,

self
.
name
)
```

```
def
create_action
(
self
,

spec
=
None
,

attr
=
'qaction'
,

shortcut_name
=
None
,

persist_shortcut
=
False
):

if
spec
is
None
:

spec
=
self
```

```
.  
action_spec
```

```
text
```

```
,
```

```
icon
```

```
,
```

```
tooltip
```

```
,
```

```
shortcut
```

```
=
```

```
spec
```

```
if
```

```
icon
```

```
is
```

```
not
```

```
None
```

```
:
```

```
action
```

```
=
```

```
QAction
```

```
(
```

```
QIcon
```

```
(
```

```
|
```

```
(
```

```
icon
```



```
)),  
  
text  
,  
  
self  
.  
gui  
)  
  
else  
:  
  
action  
  
=  
  
QAction  
(  
text  
,  
  
self  
.  
gui  
)  
  
if  
  
attr  
  
==  
  
'qaction'  
:  
  
if  
  
hasattr
```

```
(
self
.
action_menu_clone_qaction
,
'rstrip'
):
```

```
mt
```

```
=
```

```
str
(
self
.
action_menu_clone_qaction
)
```

```
else
:
```

```
mt
```

```
=
```

```
action
.
text
()
```

```
self
.
menuless_qaction
=
```

```
ma
```

```
=
```

```
QAction
```

```
(
```

```
action
```

```
.
```

```
icon
```

```
()),
```

```
mt
```

```
,
```

```
self
```

```
.
```

```
gui
```

```
)
```

```
ma
```

```
.
```

```
triggered
```

```
.
```

```
connect
```

```
(
```

```
action
```

```
.
```

```
trigger
```

```
)
```

```
for
```

```
a
```

```
in
```

```
((
```

```
action
```

```
,
```

```
ma
```

```
)
```

```
if
    attr
    ==
    'qaction'
else
    (
    action
    ,)):

a
.
setAutoRepeat
(
self
.
auto_repeat
)

text
=
tooltip

if
    tooltip
else
    text

a
.
```

```
setToolTip  
(  
text  
)
```

```
a  
.  
setStatusTip  
(  
text  
)
```

```
a  
.  
setWhatsThis  
(  
text  
)
```

```
shortcut_action
```

```
=
```

```
action
```

```
desc
```

```
=
```

```
tooltip
```

```
if
```

```
tooltip
```

```
else
```

```
None
```

```
if
attr
==
'qaction'
:

shortcut_action
=
ma

if
shortcut
is
not
None
:

keys
=
((
shortcut
,))

if
isinstance
(
shortcut
```

```
,  
string_or_bytes  
)
```

else

```
tuple  
(  
    shortcut  
)
```

if

shortcut_name

is

None

and

```
spec  
[  
    0  
]:
```

shortcut_name

=

```
str  
(  
    spec  
[  
    0  
])
```

```
if
    shortcut_name
    and
    self
    .
    action_spec
    [
    0
    ]
    and
    not
    (
        attr
        ==
        'qaction'
        and
        self
        .
        popup_type
        ==
        QPushButton
        .
        ToolButtonPopupMode
        .
        InstantPopup
    ):
    try
```



```
:  
  
self  
.  
gui  
.  
keyboard  
.  
register_shortcut  
(  
self  
.  
unique_name  
  
+  
  
'_'  
  
+  
  
attr  
,  
  
shortcut_name  
,  
  
default_keys  
=  
keys  
,  
  
action  
=  
shortcut_action  
,  
  
description  
=  
desc  
,
```

```
group
=
self
.
action_spec
[
0
],

persist_shortcut
=
persist_shortcut
)

except

NameConflict

as

e
:

try
:

prints
(
str
(
e
))

except
:
```

pass

shortcut_action

.

setShortcuts

([

QKeySequence

(

key

,

QKeySequence

.

SequenceFormat

.

PortableText

)

for

key

in

keys

])

else

:

self

.

shortcut_action_for_context_menu

=

shortcut_action

```
if
ismacos
:

# In Qt 5 keyboard shortcuts dont work unless the

# action is explicitly added to the main window

self
.
gui
.
addAction
(
shortcut_action
)

if
attr
is
not
None
:

setattr
(
self
,
attr
,
```

```
action  
)
```

```
if
```

```
attr
```

```
==
```

```
'qaction'
```

```
and
```

```
self
```

```
.  
action_add_menu  
:
```

```
menu
```

```
=
```

```
QMenu  
()
```

```
action
```

```
.  
setMenu  
(  
menu  
)
```

```
if
```

```
self
```

```
.  
action_menu_clone_qaction  
:
```

```
menu
.  
addAction  
(  
self  
.  
menuless_qaction  
)
```

```
return
```

```
action
```

```
\[docs\]
```

```
def
```

```
create_menu_action
```

```
(  
self
```

```
,
```

```
menu
```

```
,
```

```
unique_name
```

```
,
```

```
text
```

```
,
```

```
icon
```

```
=
```

```
None
```

```
,
```

```
shortcut
```

```
=
None
,

description
=
None
,

triggered
=
None
,

shortcut_name
=
None
,

persist_shortcut
=
False
):
```

```
'''
```

Convenience method to easily add actions to a QMenu.

Returns the created QAction. This action has one extra attribute

calibre_shortcut_unique_name which if not None refers to the unique

name under which this action is registered with the keyboard manager.

:param menu: The QMenu the newly created action will be added to

:param unique_name: A unique name for this action, this must be

globally unique, so make it as descriptive as possible. If in doubt, add

an UUID to it.

:param text: The text of the action.

:param icon: Either a QIcon or a file name. The file name is passed to the I() builtin, so you do not need to pass the full path to the images folder.

:param shortcut: A string, a list of strings, None or False. If False, no keyboard shortcut is registered for this action. If None, a keyboard shortcut with no default keybinding is registered. String and list of strings register a shortcut with default keybinding as specified.

:param description: A description for this action. Used to set tooltips.

:param triggered: A callable which is connected to the triggered signal of the created action.

:param shortcut_name: The text displayed to the user when customizing the keyboard shortcuts for this action. By default it is set to the value of ``text``.

:param persist_shortcut: Shortcuts for actions that don't always appear, or are library dependent, may disappear when other keyboard shortcuts are edited unless ``persist\_shortcut`` is set True.

'''

if

shortcut_name

is

None

:

shortcut_name

=

str

(

text

)

ac

=

menu

.

addAction

(

text

)

if

icon

is

not

None

:

```
if
```

```
not
```

```
isinstance
```

```
(
```

```
icon
```

```
,
```

```
QIcon
```

```
):
```

```
icon
```

```
=
```

```
QIcon
```

```
(
```

```
 |
```

```
(
```

```
icon
```

```
))
```

```
ac
```

```
.
```

```
setIcon
```

```
(
```

```
icon
```

```
)
```

```
keys
```

```
=
```

```
()
```

```
if
```

shortcut

is

not

None

and

shortcut

is

not

False

:

keys

=

((
shortcut
,)

if

isinstance

(
shortcut
,

string_or_bytes
)

else

tuple

```
(
shortcut
))

unique_name

=

menu_action_unique_name
(
self
,
unique_name
)

if

description

is

not

None
:

ac
.
setToolTip
(
description
)

ac
.
setStatusTip
(
description
```

```
)

ac
.
setWhatsThis
(
description
)

ac
.
calibre_shortcut_unique_name

=

unique_name

if

shortcut

is

not

False
:

self
.
gui
.
keyboard
.
register_shortcut
(
unique_name
,
```

```
shortcut_name
```

```
,
```

```
default_keys
```

```
=
```

```
keys
```

```
,
```

```
action
```

```
=
```

```
ac
```

```
,
```

```
description
```

```
=
```

```
description
```

```
,
```

```
group
```

```
=
```

```
self
```

```
.
```

```
action_spec
```

```
[
```

```
0
```

```
],
```

```
persist_shortcut
```

```
=
```

```
persist_shortcut
```

```
)
```

```
# In Qt 5 keyboard shortcuts dont work unless the
```

```
# action is explicitly added to the main window and on OSX and
```

Unity since the menu might be exported, the shortcuts won't work

self

.

gui

.

addAction

(

ac

)

if

triggered

is

not

None

:

ac

.

triggered

.

connect

(

triggered

)

return

ac

[\[docs\]](#)

def

```
load_resources
```

```
(  
self
```

```
,
```

```
names
```

```
):
```

```
'''
```

*If this plugin comes in a ZIP file (user added plugin), this method
will allow you to load resources from the ZIP file.*

For example to load an image::

```
    pixmap = QPixmap()
```

```
    pixmap.loadFromData(tuple(self.load_resources(['images/icon.png']).values())[0])
```

```
    icon = QIcon(pixmap)
```

:param names: List of paths to resources in the ZIP file using / as separator

:return: A dictionary of the form ``{name : file\_contents}``. Any names

that were not found in the ZIP file will not be present in the

dictionary.

```
'''
```

```
if
```

```
self
```



```
.
plugin_path

is

None
:

raise

ValueError
(
'This plugin was not loaded from a ZIP file'
)

ans

=

{}

with

ZipFile
(
self
.
plugin_path
,
'r'
)
as

zf
:

for
```

candidate

in

zf

.

namelist

():

if

candidate

in

names

:

ans

[

candidate

]

=

zf

.

read

(

candidate

)

return

ans

[\[docs\]](#)

```
def
```

```
genesis
```

```
(  
self  
):
```

```
'''
```

Setup this plugin. Only called once during initialization. self.gui is available. The action specified by :attr:`action\_spec` is available as ``self.qaction``.

```
'''
```

```
pass
```

```
\[docs\]
```

```
def
```

```
location_selected
```

```
(  
self  
,
```

```
loc  
):
```

```
'''
```

Called whenever the book list being displayed in calibre changes.

Currently values for loc are: ``library, main, card and cardb``.

This method should enable/disable this action and its sub actions as

appropriate for the location.

'''

pass

[\[docs\]](#)

def

library_changed

(
self
,

db
):

'''

Called whenever the current library is changed.

:param db: The LibraryDatabase corresponding to the current library.

'''

pass

[\[docs\]](#)

def

gui_layout_complete

(
self

```
):
```

```
'''
```

```
Called once per action when the layout of the main GUI is  
completed. If your action needs to make changes to the layout, they  
should be done here, rather than in :meth:`initialization_complete`.
```

```
'''
```

```
pass
```

```
\[docs\]
```

```
def
```

```
initialization_complete
```

```
(  
    self  
):
```

```
'''
```

```
Called once per action when the initialization of the main GUI is  
completed.
```

```
'''
```

```
pass
```

```
\[docs\]
```

```
def
```

```
tag_browser_context_action
```

```
(  
self
```

```
,
```

```
index
```

```
):
```

```
'''
```

Called when displaying the context menu in the Tag browser. ``index`` is

the QModelIndex that points to the Tag browser item that was right clicked.

Test it for validity with index.valid() and get the underlying TagTreeItem

object with index.data(Qt.ItemDataRole.UserRole). Any action objects

yielded by this method will be added to the context menu.

```
'''
```

```
if
```

```
False
```

```
:
```

```
yield
```

```
QAction
```

```
()
```

```
\[docs\]
```

```
def
```

```
shutting_down
```

```
(
```

```
self
```

);

'''

Called once per plugin when the main GUI is in the process of shutting down. Release any used resources, but try not to block the shutdown for long periods of time.

'''

pass

Source code for calibre.gui2.preferences

```
#!/usr/bin/env python
```

```
__license__
```

```
=
```

```
'GPL v3'
```

```
__copyright__
```

```
=
```

```
'2010, Kovid Goyal <kovid@kovidgoyal.net>'
```

```
__docformat__
```

```
=
```

```
'restructuredtext en'
```

```
import
```

```
textwrap
```

```
from
```

```
qt.core
```

```
import
```

```
(
```



```
QWidget
,

pyqtSignal
,

QCheckBox
,

QAbstractSpinBox
,

QApplication
,

QLineEdit
,

QComboBox
,

Qt
,

QIcon
,

QDialog
,

QVBoxLayout
,

QDialogButtonBox
)

from

calibre.customize.ui
```

import

preferences_plugins

from

calibre.utils.config

import

ConfigProxy

from

calibre.gui2.complete2

import

EditWithComplete

from

calibre.gui2.widgets

import

HistoryLineEdit

from

pyglot.builtins

import

string_or_bytes

class

AbortCommit

(

Exception

```
):
```

```
pass
```

```
class
```

```
AbortInitialize
```

```
(
```

```
Exception
```

```
):
```

```
pass
```

```
\[docs\]
```

```
class
```

```
ConfigWidgetInterface
```

```
:
```

```
'''
```

This class defines the interface that all widgets displayed in the Preferences dialog must implement. See :class:`ConfigWidgetBase` for a base class that implements this interface and defines various convenience methods as well.

```
'''
```

#: This signal must be emitted whenever the user changes a value in this

#: widget

changed_signal

=

None

#: Set to True iff the :meth:`restore\_to\_defaults` method is implemented.

supports_restoring_to_defaults

=

True

#: The tooltip for the "Restore defaults" button

restore_defaults_desc

=

(

'Restore settings to default values. '

'You have to click Apply to actually save the default settings.'

)

#: If True the Preferences dialog will not allow the user to set any more

#: preferences. Only has effect if :meth:`commit` returns True.

restart_critical

=

False

[\[docs\]](#)

def

genesis

(
self
,

gui
):

'''

*Called once before the widget is displayed, should perform any
necessary setup.*

:param gui: The main calibre graphical user interface

'''

raise

NotImplementedError

```
()
```

[\[docs\]](#)

```
def
```

```
initialize
```

```
(  
self  
):
```

```
'''
```

*Should set all config values to their initial values (the values
stored in the config files). A "return" statement is optional. Return
False if the dialog is not to be shown.*

```
'''
```

```
raise
```

```
NotImplementedError
```

```
()
```

[\[docs\]](#)

```
def
```

```
restore_defaults
```

```
(  
self  
):
```

```
'''
```

Should set all config values to their defaults.

```
'''

pass

[docs]

def

commit
(
self
):

'''

Save any changed settings. Return True if the changes require a
restart, False otherwise. Raise an :class:`AbortCommit` exception
to indicate that an error occurred. You are responsible for giving the
user feedback about what the error is and how to correct it.

'''

return

False

[docs]

def

refresh_gui
(
self
,

```

```
gui
):
```

```
'''
```

Called once after this widget is committed. Responsible for causing the gui to reread any changed settings. Note that by default the GUI re-initializes various elements anyway, so most widgets won't need to use this method.

```
'''
```

```
pass
```

```
class
```

```
Setting
```

```
:
```

```
CHOICES_SEARCH_FLAGS
```

```
=
```

```
Qt
```

```
.
```

```
MatchFlag
```

```
.
```

```
MatchExactly
```

```
|
```

```
Qt
```

```
.
```

```
MatchFlag
```



```
.  
MatchCaseSensitive
```

```
def
```

```
    __init__
```

```
(
```

```
    self
```

```
,
```

```
    name
```

```
,
```

```
    config_obj
```

```
,
```

```
    widget
```

```
,
```

```
    gui_name
```

```
=
```

```
    None
```

```
,
```

```
    empty_string_is_None
```

```
=
```

```
    True
```

```
,
```

```
    choices
```

```
=
```

```
    None
```

```
,
```

```
    restart_required
```

```
=
```

```
    False
```

```
):
```

self

.

name

,

self

.

gui_name

=

name

,

gui_name

self

.

empty_string_is_None

=

empty_string_is_None

self

.

restart_required

=

restart_required

self

.

choices

=

choices

```
if
gui_name

is

None
:

self
.
gui_name

=

'opt_'
+
name

self
.
config_obj

=

config_obj

self
.
gui_obj

=

getattr
(
widget
,

self
```

```
.
gui_name
)

self
.
widget
=
widget

if
isinstance
(
self
.
gui_obj
,
QCheckBox
):

self
.
datatype
=
'bool'

self
.
gui_obj
.
stateChanged
.
```

```
connect
(
self
.
changed
)

elif

isinstance
(
self
.
gui_obj
,
QAbstractSpinBox
):

self
.
datatype
=
'number'

self
.
gui_obj
.
valueChanged
.
connect
(
self
.
changed
)
```

elif

isinstance

(

self

.

gui_obj

,

(

QLineEdit

,

HistoryLineEdit

)):

self

.

datatype

=

'string'

self

.

gui_obj

.

textChanged

.

connect

(

self

.

changed

)

if

```
isinstance
(
self
.
gui_obj
,

HistoryLineEdit
):

self
.
gui_obj
.
initialize
(
'preferences_setting_'

+

self
.
name
)

elif

isinstance
(
self
.
gui_obj
,

QComboBox
):

self
.
datatype
```

```
=
```

```
'choice'
```

```
self
```

```
.
```

```
gui_obj
```

```
.
```

```
editTextChanged
```

```
.
```

```
connect
```

```
(
```

```
self
```

```
.
```

```
changed
```

```
)
```

```
self
```

```
.
```

```
gui_obj
```

```
.
```

```
currentIndexChanged
```

```
.
```

```
connect
```

```
(
```

```
self
```

```
.
```

```
changed
```

```
)
```

```
else
```

```
:
```

```
raise
```

```
ValueError
```

```
(
```

```
'Unknown data type
```



```
%s
'

%

self
.
gui_obj
.
__class__
)

if

isinstance
(
self
.
config_obj
,
ConfigProxy
)

and
\

not

str
(
self
.
gui_obj
.
toolTip
()):

h
```

```
=  
  
self  
.  
config_obj  
.  
help  
(  
self  
.  
name  
)  
  
if  
  
h  
:  
  
self  
.  
gui_obj  
.  
setToolTip  
(  
h  
)  
  
tt  
  
=  
  
str  
(  
self  
.  
gui_obj  
.  
toolTip  
(  
))
```

```
if
tt
:

if
not

str
(
self
.
gui_obj
.
whatsThis
()):

self
.
gui_obj
.
setWhatsThis
(
tt
)

if
not

str
(
self
.
gui_obj
.
statusTip
()):
```

```
self
.
gui_obj
.
setStatusTip
(
tt
)

tt

=

'
\n
'
.
join
(
textwrap
.
wrap
(
tt
,
70
))

self
.
gui_obj
.
setToolTip
(
tt
)
```

```
def
```

```
changed
```

```
(
```

```
self
```

```
,
```

```
*
```

```
args
```

```
):
```

```
self
```

```
.
```

```
widget
```

```
.
```

```
changed_signal
```

```
.
```

```
emit
```

```
()
```

```
def
```

```
initialize
```

```
(
```

```
self
```

```
):
```

```
self
```

```
.
```

```
gui_obj
```

```
.
```

```
blockSignals
```

```
(
```

```
True
```

```
)
```

```
if
```

```
self
.
datatype

==

'choice'
:

choices

=

self
.
choices

or

[]

if

isinstance
(
self
.
gui_obj
,
EditWithComplete
):

self
.
gui_obj
.
all_items
```

```
=  
  
choices  
  
else  
:  
  
self  
.  
gui_obj  
.  
clear  
()  
  
for  
  
x  
  
in  
  
choices  
:  
  
if  
  
isinstance  
(  
x  
,  
string_or_bytes  
):  
  
x  
  
=  
  
(
```

```
x
,

x
)

self
.
gui_obj
.
addItem
(
x
[
0
],

(
x
[
1
]))

self
.
set_gui_val
(
self
.
get_config_val
(
default
=
False
))

self
.
gui_obj
.
```



```
blockSignals
```

```
(
```

```
False
```

```
)
```

```
self
```

```
.
```

```
initial_value
```

```
=
```

```
self
```

```
.
```

```
get_gui_val
```

```
()
```

```
def
```

```
commit
```

```
(
```

```
self
```

```
):
```

```
val
```

```
=
```

```
self
```

```
.
```

```
get_gui_val
```

```
()
```

```
oldval
```

```
=
```

```
self
```

```
.
```

```
get_config_val  
(
```

```
changed
```

```
=
```

```
val
```

```
!=
```

```
oldval
```

```
if
```

```
changed
```

```
:
```

```
self
```

```
.
```

```
set_config_val
```

```
(
```

```
self
```

```
.
```

```
get_gui_val
```

```
())
```

```
return
```

```
changed
```

```
and
```

```
self
```

```
.
```

```
restart_required
```

```
def
restore_defaults
(
self
):
```

```
self
.
set_gui_val
(
self
.
get_config_val
(
default
=
True
))
```

```
def
get_config_val
(
self
,
default
=
False
):
```

```
if
default
:
```

```
val
```

```
=  
  
self  
.  
config_obj  
.  
defaults  
[  
self  
.  
name  
]  
  
else  
:  
  
val  
  
=  
  
self  
.  
config_obj  
[  
self  
.  
name  
]  
  
return  
  
val  
  
def  
  
set_config_val  
(
```

```
self
,

val
):

self
.
config_obj
[
self
.
name
]
=

val

def

set_gui_val
(
self
,

val
):

if

self
.
datatype
==

'bool'
:
```

```
self
.  
gui_obj  
.  
setChecked  
(  
    bool  
(  
    val  
))
```

elif

```
self  
.  
datatype
```

```
==
```

```
'number'  
:
```

```
self  
.  
gui_obj  
.  
setValue  
(  
    val  
)
```

elif

```
self  
.  
datatype
```

```
==
```

```
'string'
```

```
:
```

```
self
```

```
.
```

```
gui_obj
```

```
.
```

```
setText
```

```
(
```

```
val
```

```
if
```

```
val
```

```
else
```

```
"
```

```
)
```

```
elif
```

```
self
```

```
.
```

```
datatype
```

```
==
```

```
'choice'
```

```
:
```

```
if
```

```
isinstance
```

```
(
```

```
self
```

```
.
```

```
gui_obj
```

```
,
```

```
EditWithComplete  
):
```

```
self
```

```
.
```

```
gui_obj
```

```
.
```

```
setText
```

```
(
```

```
val
```

```
)
```

```
else
```

```
:
```

```
idx
```

```
=
```

```
self
```

```
.
```

```
gui_obj
```

```
.
```

```
findData
```

```
((
```

```
val
```

```
),
```

```
role
```

```
=
```

```
Qt
```

```
.
```

```
ItemDataRole
```

```
.
```

```
UserRole
```

```
,
```

```
flags
```



```
=  
self  
.  
CHOICES_SEARCH_FLAGS  
)
```

```
if
```

```
idx
```

```
==
```

```
-
```

```
1
```

```
:
```

```
idx
```

```
=
```

```
0
```

```
self
```

```
.
```

```
gui_obj
```

```
.
```

```
setCurrentIndex
```

```
(
```

```
idx
```

```
)
```

```
def
```

```
get_gui_val
```

```
(
```

```
self
```

```
):
```

```
if

self
.
datatype

==

'bool'
:

val

=

bool
(
self
.
gui_obj
.
isChecked
())

elif

self
.
datatype

==

'number'
:

val

=
```

```
self
    .
    gui_obj
    .
    value
    ()

elif

self
    .
    datatype

==

'string'
:

val

=

str
(
    self
    .
    gui_obj
    .
    text
    ())
    .
    strip
    ()

if

self
    .
    empty_string_is_None
```

and

not

val

:

val

=

None

elif

self

.

datatype

==

'choice'

:

if

isinstance

(

self

.

gui_obj

,

EditWithComplete

):

val

=

```
str
(
self
.
gui_obj
.
text
())

else
:

idx

=

self
.
gui_obj
.
currentIndex
()

if
idx

<

0
:

idx

=

0
```

```
val
=

str
(
self
.
gui_obj
.
itemData
(
idx
)

or

"
)

return

val

class

CommaSeparatedList
(
Setting
):

def

set_gui_val
(
self
,
```

```
val  
):
```

```
x
```

```
=
```

```
"
```

```
if
```

```
val  
:
```

```
x
```

```
=
```

```
' , '
```

```
.
```

```
join
```

```
(
```

```
val
```

```
)
```

```
self
```

```
.
```

```
gui_obj
```

```
.
```

```
setText
```

```
(
```

```
x
```

```
)
```

```
def
```

```
get_gui_val
```

```
(  
    self  
):
```

```
    val
```

```
    =
```

```
    str
```

```
    (  
        self
```

```
        .  
        gui_obj
```

```
        .  
        text  
        ())
```

```
        .  
        strip  
        ())
```

```
    ans
```

```
    =
```

```
    []
```

```
    if
```

```
    val  
    :
```

```
    ans
```

```
    =
```

```
    [  
        x
```

```
        .
```



```
strip  
()
```

```
for
```

```
x
```

```
in
```

```
val
```

```
.  
split  
(  
' '  
,  
)]
```

```
ans
```

```
=
```

```
[  
x
```

```
for
```

```
x
```

```
in
```

```
ans
```

```
if
```

```
x  
]
```

```
return
```

```
ans
```

[\[docs\]](#)

class

ConfigWidgetBase

(
QWidget
,

ConfigWidgetInterface
):

'''

Base class that contains code to easily add standard config widgets like checkboxes, combo boxes, text fields and so on. See the :meth:`register` method.

This class automatically handles change notification, resetting to default, translation between gui objects and config objects, etc. for registered settings.

If your config widget inherits from this class but includes setting that are not registered, you should override the :class:`ConfigWidgetInterface` methods and call the base class methods inside the overrides.

'''

changed_signal

```
=
```

```
pyqtSignal  
(
```

```
restart_now
```

```
=
```

```
pyqtSignal  
(
```

```
supports_restoring_to_defaults
```

```
=
```

```
True
```

```
restart_critical
```

```
=
```

```
False
```

```
def
```

```
__init__
```

```
(
```

```
self
```

```
,
```

```
parent
```

```
=
```

```
None
```

```
):
```

QWidget

```
·  
__init__
```

```
(  
self
```

```
,
```

```
parent  
)
```

```
if
```

```
hasattr
```

```
(  
self
```

```
,
```

```
'setupUi'  
):
```

```
self
```

```
·
```

```
setupUi
```

```
(  
self  
)
```

```
self
```

```
·
```

```
settings
```

```
=
```

```
{}
```

[\[docs\]](#)

```
def
register
(
self
,
name
,
config_obj
,
gui_name
=
None
,
choices
=
None
,
restart_required
=
False
,
empty_string_is_None
=
True
,
setting
=
Setting
):
'''
```

Register a setting.

:param name: The setting name

:param config: The config object that reads/writes the setting

:param gui_name: The name of the GUI object that presents an interface

to change the setting. By default it is assumed to be

``opt\_ ' + name``.

:param choices: If this setting is a multiple choice (combobox) based

setting, the list of choices. The list is a list of two

element tuples of the form: ``[(gui name, value), ...]``

:param setting: The class responsible for managing this setting. The

default class handles almost all cases, so this param

is rarely used.

'''

setting

=

setting

(

name

,

config_obj

,

self

,

gui_name

```
=
gui_name
,

choices
=
choices
,

restart_required
=
restart_required
,

empty_string_is_None
=
empty_string_is_None
)
```

return

```
self
.
register_setting
(
setting
)
```

def

```
register_setting
(
self
,

setting
):
```

```
self
.  
settings  
[  
setting  
.  
name  
]
```

```
=
```

```
setting
```

```
return
```

```
setting
```

```
\[docs\]
```

```
def
```

```
initialize  
(  
self  
):
```

```
for
```

```
setting
```

```
in
```

```
self  
.  
settings  
.  
values  
():
```


setting

.

initialize

()

[\[docs\]](#)

def

commit

(

self

,

*

args

):

restart_required

=

False

for

setting

in

self

.

settings

.

values

():

```
rr
=
setting
.
commit
()

if

rr
:

restart_required

=

True

return

restart_required

\[docs\]

def

restore_defaults
(
self
,

*
args
):

for
```

setting

in

self

.

settings

.

values

():

setting

.

restore_defaults

()

def

get_plugin

(

category

,

name

):

for

plugin

in

preferences_plugins

():

if

plugin

```
.
category

==

category

and

plugin
.
name

==

name
:

return

plugin

raise

ValueError
(

'No Preferences Plugin with category:
%s
and name:
%s
found'

%

(
category
,
```

```
name
))
```

```
class
```

```
ConfigDialog
```

```
(
QDialog
):
```

```
def
```

```
set_widget
```

```
(
self
,
```

```
w
):
```

```
self
```

```
.
w
```

```
=
```

```
w
```

```
def
```

```
accept
```

```
(
self
):
```

```
try
:

self
.
restart_required

=

self
.
w
.
commit
()

except

AbortCommit
:

return

QDialog
.
accept
(
self
)

def

init_gui
():

from
```

```
calibre.gui2.ui
```

```
import
```

```
Main
```

```
from
```

```
calibre.gui2.main
```

```
import
```

```
option_parser
```

```
from
```

```
calibre.library
```

```
import
```

```
db
```

```
parser
```

```
=
```

```
option_parser  
()
```

```
opts
```

```
,
```

```
args
```

```
=
```

```
parser
```

```
.
```

```
parse_args
([])

actions

=

tuple
(
Main
.
create_application_menubar
())

db

=

db
()

gui

=

Main
(
opts
)

gui
.
initialize
(
db
.
library_path
,
```



```
db
,

actions
,

show_gui
=
False
)

return

gui

def

show_config_widget
(
category
,

name
,

gui
=
None
,

show_restart_msg
=
False
,

parent
=
None
,
```

```
never_shutdown
```

```
=
```

```
False
```

```
):
```

```
'''
```

Show the preferences plugin identified by category and name

:param gui: gui instance, if None a hidden gui is created

:param show_restart_msg: If True and the preferences plugin indicates a restart is required, show a message box telling the user to restart

:param parent: The parent of the displayed dialog

:return: True iff a restart is required for the changes made by the user to take effect

```
'''
```

```
from
```

```
calibre.gui2
```

```
import
```

```
gprefs
```

```
pl
```

```
=
```

```
get_plugin
```

```
(
```

```
category
,
name
)

d
=

ConfigDialog
(
parent
)

d
.
resize
(
750
,
550
)

conf_name
=

'config_widget_dialog_geometry_
%s
—
%s
'
%
(
category
,
name
```

```
)

geom

=

gprefs
.
get
(
  conf_name
,
  None
)

d
.
setWindowTitle
(
  -
  (
    'Configure '
  )
)

+

pl
.
gui_name
)

d
.
setWindowIcon
(
  QIcon
  (
    |
  (
```

```
'config.png'
```

```
)))
```

```
bb
```

```
=
```

```
QDialogButtonBox
```

```
(
```

```
d
```

```
)
```

```
bb
```

```
.
```

```
setStandardButtons
```

```
(
```

```
QDialogButtonBox
```

```
.
```

```
StandardButton
```

```
.
```

```
Apply
```

```
|
```

```
QDialogButtonBox
```

```
.
```

```
StandardButton
```

```
.
```

```
Cancel
```

```
|
```

```
QDialogButtonBox
```

```
.
```

```
StandardButton
```

```
.
```

```
RestoreDefaults
```

```
)
```

```
bb
```

```
.
```

```
accepted
```

```
.
```

```
connect
```

```
(  
d  
.accept  
)  
  
bb  
.rejected  
.connect  
(  
d  
.reject  
)  
  
w  
  
=  
  
pl  
.create_widget  
(  
d  
)  
  
d  
.set_widget  
(  
w  
)  
  
bb  
.button  
(
```

```
QDialogButtonBox
.
StandardButton
.
RestoreDefaults
)
.
clicked
.
connect
(
w
.
restore_defaults
)

bb
.
button
(
QDialogButtonBox
.
StandardButton
.
RestoreDefaults
)
.
setEnabled
(
w
.
supports_restoring_to_defaults
)

bb
.
button
(
QDialogButtonBox
.
StandardButton
```

```
.
Apply
)

.setEnabled
(
False
)

bb

button
(
QDialogButtonBox
.
StandardButton
.
Apply
)

clicked

connect
(
d
.
accept
)

def

onchange
():

b

=

bb
```



```
.  
button  
(  
    QDialogButtonBox  
.  
    StandardButton  
.  
    Apply  
)
```

```
b  
.  
setEnabled  
(  
    True  
)
```

```
b  
.  
setDefault  
(  
    True  
)
```

```
b  
.  
setAutoDefault  
(  
    True  
)
```

```
w  
.  
changed_signal  
.  
connect  
(  
    onchange  
)
```

```
bb
.  
button  
(  
QDialogButtonBox  
.  
StandardButton  
.  
Cancel  
)  
.  
setFocus  
(  
Qt  
.  
FocusReason  
.  
OtherFocusReason  
)  
  
l  
  
=  
  
QVBoxLayout  
(  
  
d  
.  
setLayout  
(  
l  
)  
  
l  
.  
addWidget  
(
```

```
w
)

l
.
addWidget
(
bb
)
```

```
mygui
```

```
=
```

```
gui
```

```
is
```

```
None
```

```
if
```

```
gui
```

```
is
```

```
None
```

```
:
```

```
gui
```

```
=
```

```
init_gui
()
```

```
mygui
```

```
=
```

```
True
```

```
w
```

```
.
```

```
genesis
```

```
(
```

```
gui
```

```
)
```

```
w
```

```
.
```

```
initialize
```

```
()
```

```
if
```

```
geom
```

```
is
```

```
not
```

```
None
```

```
:
```

```
QApplication
```

```
.
```

```
instance
```

```
()
```

```
.
```

```
safe_restore_geometry
```

```
(
```

```
d
```

```
,
```

```
geom
```

```
)
```

```
d
.
exec
()

geom

=

bytearray
(
d
.
saveGeometry
())

gprefs
[
conf_name
]

=

geom

rr

=

getattr
(
d
,
'restart_required'
,
False
```

```
)

if

show_restart_msg

and

rr
:

from

calibre.gui2

import

warning_dialog

warning_dialog
(
gui
,

'Restart required'
,

'Restart required'
,

show
=
True
)

if

mygui
```

and

not

never_shutdown

:

gui

.

shutdown

()

return

rr

Testing {{{

def

test_widget

(

category

,

name

,

gui

=

None

):

show_config_widget

(

category

,

```
name
',

gui
=
gui
',

show_restart_msg
=
True
)
```

```
def
```

```
test_all
():
```

```
from
```

```
qt.core
```

```
import
```

```
QApplication
```

```
app
```

```
=
```

```
QApplication
([])
```

```
app
```

```
gui
```



```
=
```

```
init_gui  
(
```

```
for
```

```
plugin
```

```
in
```

```
preferences_plugins  
():
```

```
test_widget
```

```
(  
plugin
```

```
.  
category  
,
```

```
plugin
```

```
.  
name  
,
```

```
gui
```

```
=  
gui  
)
```

```
gui
```

```
.  
shutdown  
()
```

```
if
```

```
__name__
```

```
==
```

```
'__main__'
```

```
:
```

```
test_all
```

```
()
```

```
# }}
```

Source code for calibre.gui2.tweak_book.boss

```
#!/usr/bin/env python
```

```
# License: GPLv3 Copyright: 2013, Kovid Goyal <kovid at kovidgoyal.net>
```

```
import
```

```
errno
```

```
import
```

```
os
```

```
import
```

```
shutil
```

```
import
```

```
sys
```

```
import
```

```
tempfile
```

```
from
```

```
functools
```

```
import
```

```
partial
```

```
,  
  
wraps  
  
from  
  
qt.core  
  
import  
  
(  
  
    QApplication  
,  
  
    QCheckBox  
,  
  
    QDialog  
,  
  
    QDialogButtonBox  
,  
  
    QGridLayout  
,  
  
    QIcon  
,  
  
    QInputDialog  
,  
  
    QLabel  
,  
  
    QMimeData  
,  
  
    QObject
```

```
,  
  
QSize  
,  
  
Qt  
,  
  
QTimer  
,  
  
QUrl  
,  
  
QVBoxLayout  
,  
  
pyqtSignal  
)
```

```
from
```

```
calibre
```

```
import
```

```
isbytestring
```

```
,
```

```
prints
```

```
from
```

```
calibre.constants
```

```
import
```

```
cache_dir
```

```
,
```

```
iswindows

from
calibre.ebooks.oeb.base

import
urlnormalize

from
calibre.ebooks.oeb.polish.container

import
(

OEB_DOCS
,

OEB_STYLES
,

clone_container
,

get_container

as

_gc
,

guess_type

)

from
calibre.ebooks.oeb.polish.cover
```

```
import

(

mark_as_cover
,

mark_as_titlepage
,

set_cover

)

from

calibre.ebooks.oeb.polish.css

import

filter_css
,

rename_class

from

calibre.ebooks.oeb.polish.main

import

SUPPORTED
,

tweak_polish

from

calibre.ebooks.oeb.polish.pretty

import
```

```
fix_all_html
,

pretty_all

from

calibre.ebooks.oeb.polish.replace

import

(

get_recommended_folders
,

rationalize_folders
,

rename_files
,

replace_file

)

from

calibre.ebooks.oeb.polish.split

import

AbortError
,

merge
,

multisplit
,

split
```



```
from
```

```
calibre.ebooks.oeb.polish.toc
```

```
import
```

```
create_inline_toc
```

```
,
```

```
remove_names_from_toc
```

```
from
```

```
calibre.ebooks.oeb.polish.utils
```

```
import
```

```
(
```

```
link_stylesheets
```

```
,
```

```
setup_css_parser_serialization
```

```
as
```

```
scs
```

```
)
```

```
from
```

```
calibre.gui2
```

```
import
```

```
(
```

```
add_to_recent_docs
```

```
,
```

```
choose_dir
,
choose_files
,
choose_save_file
,
error_dialog
,

info_dialog
,
open_url
,
question_dialog
)

from

calibre.gui2.dialogs.confirm_delete

import

confirm

from

calibre.gui2.tweak_book

import

(

actions
,
```

```
current_container
,
dictionaries
,
editor_name
,
editors
,
set_book_locale
,

set_current_container
,

tprefs
)

from

calibre.gui2.tweak_book.completion.worker

import

completion_worker

from

calibre.gui2.tweak_book.editor

import

editor_from_syntax
,

syntax_from_mime
```

```
from

calibre.gui2.tweak_book.editor.insert_resource

import

NewBook
,

get_resource_data

from

calibre.gui2.tweak_book.file_list

import

FILE_COPY_MIME
,

NewFileDialog

from

calibre.gui2.tweak_book.preferences

import

Preferences

from

calibre.gui2.tweak_book.preview

import

parse_worker

from

calibre.gui2.tweak_book.save

import
```

```
(  
  
    SaveManager  
    ,  
  
    find_first_existing_ancestor  
    ,  
  
    save_container  
  
)  
  
from  
  
calibre.gui2.tweak_book.search  
  
import  
  
    run_search  
    ,  
  
    validate_search_request  
  
from  
  
calibre.gui2.tweak_book.spell  
  
import  
  
(  
  
    find_next  
  
    as  
  
    find_next_word  
    ,  
  
    find_next_error
```

```
)  
  
from  
calibre.gui2.tweak_book.toc  
  
import  
TOCEditor  
  
from  
calibre.gui2.tweak_book.undo  
  
import  
GlobalUndoHistory  
  
from  
calibre.gui2.tweak_book.widgets  
  
import  
(  
  
AddCover  
,  
  
BusyCursor  
,  
  
FilterCSS  
,  
  
ImportForeign  
,  
  
InsertLink  
,  
  
InsertSemantics
```

```
,  
  
InsertTag  
,  
  
MultiSplit  
,  
  
QuickOpen  
,  
  
RationalizeFolders  
)  
  
from  
  
calibre.ptempfile  
  
import  
  
PersistentTemporaryDirectory  
,  
  
TemporaryDirectory  
  
from  
  
calibre.utils.config  
  
import  
  
JSONConfig  
  
from  
  
calibre.utils.icu  
  
import  
  
numeric_sort_key
```

from

calibre.utils.imghdr

import

identify

from

calibre.utils.tdir_in_cache

import

tdir_in_cache

from

polyglot.builtins

import

as_bytes

,

iteritems

,

itervalues

,

string_or_bytes

from

polyglot.urllib

import

urlparse

_diff_dialogs

=

[]

last_used_transform_rules

=

[]

last_used_html_transform_rules

=

[]

def

get_container

(

*

args

,

**

kwargs

):

kwargs

[

'tweak_mode'

]

=

True

container

```
=
```

```
_gc
```

```
(
```

```
*
```

```
args
```

```
,
```

```
**
```

```
kwargs
```

```
)
```

```
return
```

```
container
```

```
def
```

```
setup_css_parser_serialization
```

```
():
```

```
scs
```

```
(
```

```
tprefs
```

```
[
```

```
'editor_tab_stop_width'
```

```
])
```

```
def
```

```
in_thread_job
```

```
(
```

```
func
```

```
):
```

@wraps

```
(  
func  
)
```

def

ans

```
(  
*  
args  
,  
  
**  
kwargs  
):
```

with

BusyCursor
():

return

```
func  
(  
*  
args  
,  
  
**  
kwargs  
)
```

return

ans

```
def
```

```
get_boss  
():
```

```
return
```

```
get_boss  
.  
boss
```

```
\[docs\]  
class
```

```
Boss
```

```
(  
    QObject  
):
```

```
handle_completion_result_signal
```

```
=
```

```
pyqtSignal  
(  
    object  
)
```

```
def
```

```
__init__  
(  
    self
```

```
,  
  
parent  
,  
  
notify  
=  
None  
):  
  
QObject  
.  
__init__  
(  
self  
,  
  
parent  
)  
  
self  
.  
global_undo  
  
=  
  
GlobalUndoHistory  
(  
  
self  
.  
container_count  
  
=  
  
0  
  
self  
.  

```

```
tdir
```

```
=
```

```
None
```

```
self
```

```
.
```

```
save_manager
```

```
=
```

```
SaveManager
```

```
(
```

```
parent
```

```
,
```

```
notify
```

```
)
```

```
self
```

```
.
```

```
save_manager
```

```
.
```

```
report_error
```

```
.
```

```
connect
```

```
(
```

```
self
```

```
.
```

```
report_save_error
```

```
)
```

```
self
```

```
.
```

```
save_manager
```

```
.
```

```
check_for_completion
```

```
.
```

```
connect
```

```
(
self
.
check_terminal_save
)

self
.
doing_terminal_save

=

False

self
.
ignore_preview_to_editor_sync

=

False

setup_css_parser_serialization
()

get_boss
.
boss

=

self

self
.
gui

=
```

```
parent
```

```
completion_worker
```

```
()
```

```
.
```

```
result_callback
```

```
=
```

```
self
```

```
.
```

```
handle_completion_result_signal
```

```
.
```

```
emit
```

```
self
```

```
.
```

```
handle_completion_result_signal
```

```
.
```

```
connect
```

```
(
```

```
self
```

```
.
```

```
handle_completion_result
```

```
,
```

```
Qt
```

```
.
```

```
ConnectionType
```

```
.
```

```
QueuedConnection
```

```
)
```

```
self
```

```
.
```

```
completion_request_count
```

```
=
```



```
0
```

```
self
```

```
.  
editor_cache
```

```
=
```

```
JSONConfig  
(  
    'editor-cache'  
    ,
```

```
base_path  
=  
cache_dir  
())
```

```
d
```

```
=
```

```
self
```

```
.  
editor_cache
```

```
.  
defaults
```

```
d  
[  
    'edit_book_state'  
]
```

```
=
```

```
{}
```

```
d  
[
```

```
'edit_book_state_order'  
]
```

```
=
```

```
[]
```

```
def
```

```
__call__
```

```
(  
self  
,
```

```
gui  
):
```

```
self
```

```
.  
gui
```

```
=
```

```
gui
```

```
gui
```

```
.  
message_popup
```

```
.  
undo_requested
```

```
.  
connect
```

```
(  
self
```

```
.  
do_global_undo  
)
```

```
fl
=
gui
.
file_list

fl
.
delete_requested
.
connect
(
self
.
delete_requested
)

fl
.
reorder_spine
.
connect
(
self
.
reorder_spine
)

fl
.
rename_requested
.
connect
(
self
.
rename_requested
)
```

```
fl
.  
bulk_rename_requested  
.  
connect  
(  
self  
.  
bulk_rename_requested  
)
```

```
fl  
.  
edit_file  
.  
connect  
(  
self  
.  
edit_file_requested  
)
```

```
fl  
.  
merge_requested  
.  
connect  
(  
self  
.  
merge_requested  
)
```

```
fl  
.  
mark_requested  
.  
connect
```

```
(
self
.
mark_requested
)

fl
.
export_requested
.
connect
(
self
.
export_requested
)

fl
.
replace_requested
.
connect
(
self
.
replace_requested
)

fl
.
link_stylesheets_requested
.
connect
(
self
.
link_stylesheets_requested
)
```

```
fl
.  
initiate_file_copy  
.  
connect  
(  
self  
.  
copy_files_to_clipboard  
)
```

```
fl  
.  
initiate_file_paste  
.  
connect  
(  
self  
.  
paste_files_from_clipboard  
)
```

```
fl  
.  
open_file_with  
.  
connect  
(  
self  
.  
open_file_with  
)
```

```
self  
.  
gui  
.  
central  
.  
current_editor_changed
```

```
.  
connect  
(  
self  
.  
apply_current_editor_state  
)
```

```
self
```

```
.  
gui
```

```
.  
central
```

```
.  
close_requested
```

```
.  
connect
```

```
(  
self
```

```
.  
editor_close_requested  
)
```

```
self
```

```
.  
gui
```

```
.  
central
```

```
.  
search_panel
```

```
.  
search_triggered
```

```
.  
connect
```

```
(  
self
```

```
.  
search  
)
```

```
self
.  
gui  
.  
text_search
```

```
.  
find_text
```

```
.  
connect
```

```
(  
self
```

```
.  
find_text  
)
```

```
self
```

```
.  
gui
```

```
.  
preview
```

```
.  
sync_requested
```

```
.  
connect
```

```
(  
self
```

```
.  
sync_editor_to_preview  
)
```

```
self
```

```
.  
gui
```

```
.  
preview
```

```
.  
split_start_requested
```

```
.  
connect
```

```
(  
self
```



```
.
split_start_requested
)
```

```
self
```

```
.
gui
```

```
.
preview
```

```
.
split_requested
```

```
.
connect
```

```
(
self
```

```
.
split_requested
)
```

```
self
```

```
.
gui
```

```
.
preview
```

```
.
link_clicked
```

```
.
connect
```

```
(
self
```

```
.
link_clicked
)
```

```
self
```

```
.
gui
```

```
.
preview
```

```
.
```

```
render_process_restarted
.  
connect  
(  
self  
.  
report_render_process_restart  
)
```

```
self  
.  
gui  
.  
preview  
.  
open_file_with  
.  
connect  
(  
self  
.  
open_file_with  
)
```

```
self  
.  
gui  
.  
preview  
.  
edit_file  
.  
connect  
(  
self  
.  
edit_file_requested  
)
```

```
self
```

```
.  
gui  
.  
check_book  
.  
item_activated  
.  
connect  
(  
self  
.  
check_item_activated  
)
```

```
self  
.  
gui  
.  
check_book  
.  
check_requested  
.  
connect  
(  
self  
.  
check_requested  
)
```

```
self  
.  
gui  
.  
check_book  
.  
fix_requested  
.  
connect  
(  
self  
.  

```

```
fix_requested
)
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
toc_view
```

```
.
```

```
navigate_requested
```

```
.
```

```
connect
```

```
(
```

```
self
```

```
.
```

```
link_clicked
```

```
)
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
toc_view
```

```
.
```

```
refresh_requested
```

```
.
```

```
connect
```

```
(
```

```
self
```

```
.
```

```
commit_all_editors_to_container
```

```
)
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
image_browser
```

```
.
```

```
image_activated
```

```
.
connect
(
self
.
image_activated
)

self
.
gui
.
checkpoints
.
revert_requested
.
connect
(
self
.
revert_requested
)

self
.
gui
.
checkpoints
.
compare_requested
.
connect
(
self
.
compare_requested
)

self
.
```

```
gui
·
saved_searches
·
run_saved_searches
·
connect
(
self
·
run_saved_searches
)

self
·
gui
·
saved_searches
·
copy_search_to_search_panel
·
connect
(
self
·
gui
·
central
·
search_panel
·
paste_saved_search
)

self
·
gui
·
central
·
search_panel
```

```
.
save_search

.
connect
(
self

.
save_search
)

self

.
gui

.
central

.
search_panel

.
show_saved_searches

.
connect
(
self

.
show_saved_searches
)

self

.
gui

.
spell_check

.
find_word

.
connect
(
self

.
find_word
)
)
```

```
self
.  
gui  
.  
spell_check  
.  
refresh_requested  
.  
connect  
(  
self  
.  
commit_all_editors_to_container  
)
```

```
self  
.  
gui  
.  
spell_check  
.  
word_replaced  
.  
connect  
(  
self  
.  
word_replaced  
)
```

```
self  
.  
gui  
.  
spell_check  
.  
word_ignored  
.  
connect
```



```
(
self
.
word_ignored
)

self
.
gui
.
spell_check
.
change_requested
.
connect
(
self
.
word_change_requested
)

self
.
gui
.
live_css
.
goto_declaration
.
connect
(
self
.
goto_style_declaration
)

self
.
gui
.
```

```
manage_fonts
.  
container_changed  
.  
connect  
(  
self  
.  
apply_container_update_to_gui  
)
```

```
self  
.  
gui  
.  
manage_fonts  
.  
embed_all_fonts  
.  
connect  
(  
self  
.  
manage_fonts_embed  
)
```

```
self  
.  
gui  
.  
manage_fonts  
.  
subset_all_fonts  
.  
connect  
(  
self  
.  
manage_fonts_subset  
)
```

```
self
.  
gui  
.  
reports  
.  
edit_requested  
.  
connect  
(  
self  
.  
reports_edit_requested  
)
```

```
self  
.  
gui  
.  
reports  
.  
refresh_starting  
.  
connect  
(  
self  
.  
commit_all_editors_to_container  
)
```

```
self  
.  
gui  
.  
reports  
.  
delete_requested  
.  
connect  
(
```

```
self
.
delete_requested
)

def

report_render_process_restart
(
self
):

self
.
gui
.
show_status_message
(
_
(
'The Qt WebEngine Render process crashed and has been restarted'
))

@property

def

currently_editing
(
self
):

' Return the name of the file being edited currently or None if no file is being edited '

return
```

```
editor_name
(  
    self  
    .  
    gui  
    .  
    central  
    .  
    current_editor  
)
```

```
def
```

```
preferences  
(  
    self  
):
```

```
orig_spell  
  
=  
  
tprefs  
[  
    'inline_spell_check'  
]
```

```
orig_size  
  
=  
  
tprefs  
[  
    'toolbar_icon_size'  
]
```

```
p
```

```
=
```

```
Preferences
```

```
(  
self  
.  
gui  
)
```

```
ret
```

```
=
```

```
p  
.  
exec  
()
```

```
if
```

```
p  
.  
dictionaries_changed  
:
```

```
dictionaries  
.  
clear_caches  
()
```

```
dictionaries  
.  
initialize  
(  
force  
=  
True  
)
```

```
# Reread user dictionaries
```

```
if
```

```
p
```

```
.  
toolbars_changed  
:
```

```
self
```

```
.  
gui
```

```
.  
populate_toolbars  
(
```

```
for
```

```
ed
```

```
in
```

```
itervalues  
(  
editors  
):
```

```
if
```

```
hasattr  
(  
ed  
,
```

```
'populate_toolbars'  
):
```

```
ed
.  
populate_toolbars  
()
```

```
if
```

```
orig_size
```

```
!=
```

```
tprefs  
[  
  'toolbar_icon_size'  
]:
```

```
for
```

```
ed
```

```
in
```

```
itervalues  
(  
  editors  
):
```

```
if
```

```
hasattr  
(  
  ed  
,  
  'bars'  
):
```

```
for
```



```
bar
```

```
in
```

```
ed
```

```
.
```

```
bars
```

```
:
```

```
bar
```

```
.
```

```
setIconSize
```

```
(
```

```
QSize
```

```
(
```

```
tprefs
```

```
[
```

```
'toolbar_icon_size'
```

```
],
```

```
tprefs
```

```
[
```

```
'toolbar_icon_size'
```

```
]))
```

```
if
```

```
ret
```

```
==
```

```
QDialog
```

```
.
```

```
DialogCode
```

```
.
```

```
Accepted
```

```
:
```

```
setup_css_parser_serialization
```

```
()

self
.
gui
.
apply_settings
()

self
.
refresh_file_list
()

self
.
gui
.
preview
.
start_refresh_timer
()

if

ret

==

QDialog
.
DialogCode
.
Accepted

or

p
.
```

```
dictionaries_changed  
:
```

```
for
```

```
ed
```

```
in
```

```
itervalues  
(  
editors  
):
```

```
ed  
.  
apply_settings  
(  
dictionaries_changed  
=  
p  
.  
dictionaries_changed  
)
```

```
if
```

```
orig_spell
```

```
!=
```

```
tprefs  
[  
'inline_spell_check'  
]:
```

```
from
```

```
calibre.gui2.tweak\_book.editor.syntax.html
```

```
import
```

```
refresh_spell_check_status
```

```
refresh_spell_check_status  
(
```

```
for
```

```
ed
```

```
in
```

```
itervalues  
(  
    editors  
):
```

```
try
```

```
:
```

```
ed
```

```
.
```

```
editor
```

```
.
```

```
highlighter
```

```
.
```

```
rehighlight
```

```
()
```

```
except
```

```
AttributeError
```

```
:
```

```
pass
```

```
def
```

```
mark_requested
```

```
(
```

```
self
```

```
,
```

```
name
```

```
,
```

```
action
```

```
):
```

```
self
```

```
.
```

```
commit_dirty_opf
```

```
()
```

```
c
```

```
=
```

```
current_container
```

```
()
```

```
if
```

```
action
```

```
==
```

```
'cover'
```

```
:
```

```
mark_as_cover
```

```
(
```

```
current_container  
(),
```

```
name  
)
```

```
elif
```

```
action
```

```
.  
startswith  
(  
    'titlepage:'  
):
```

```
action
```

```
,
```

```
move_to_start
```

```
=
```

```
action
```

```
.  
partition  
(  
    ':'  
)[  
    0  
    ::  
    2  
]
```

```
move_to_start
```

```
=
```

```
move_to_start
```

```
==
```

'True'

```
mark_as_titlepage  
(  
current_container  
(),
```

```
name  
,
```

```
move_to_start  
=  
move_to_start  
)
```

if

```
c  
.  
opf_name
```

in

```
editors  
:
```

```
editors  
[  
c  
.  
opf_name  
]  
.  
replace_data  
(  
c  
.  
raw_data
```

```
(
c
.
opf_name
))

self
.
gui
.
file_list
.
build
(
c
)

self
.
set_modified
()

def

mkdtemp
(
self
,

prefix
=
"
):

self
.
container_count
```



```
+=
```

```
1
```

```
return
```

```
tempfile
```

```
.
```

```
mkdtemp
```

```
(
```

```
prefix
```

```
=
```

```
,
```

```
%s%05d
```

```
_'
```

```
%
```

```
(
```

```
prefix
```

```
,
```

```
self
```

```
.
```

```
container_count
```

```
),
```

```
dir
```

```
=
```

```
self
```

```
.
```

```
tdir
```

```
)
```

```
def
```

```
_check_before_open
```

```
(
```

```
self
```

```
):
```

```
if
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
action_save
```

```
.
```

```
isEnabled
```

```
():
```

```
if
```

```
not
```

```
question_dialog
```

```
(
```

```
self
```

```
.
```

```
gui
```

```
,
```

```
—
```

```
(
```

```
'Unsaved changes'
```

```
),
```

```
—
```

```
(
```

```
'The current book has unsaved changes. If you open a new book, they will be lost.'
```

```
' Are you sure you want to proceed?'
```

```
)):
```

```
return
```

```
if
self
.
save_manager
.
has_tasks
:

return

info_dialog
(
self
.
gui
,

—
(
'Cannot open'
),

—
(
'The current book is being saved, you cannot open a new book until'

'the saving is completed'
),

show
=
True
)

return

True
```

```
def
```

```
new_book
```

```
(
```

```
self
```

```
):
```

```
if
```

```
not
```

```
self
```

```
.
```

```
_check_before_open
```

```
():
```

```
return
```

```
d
```

```
=
```

```
NewBook
```

```
(
```

```
self
```

```
.
```

```
gui
```

```
)
```

```
if
```

```
d
```

```
.
```

```
exec
```

```
()
```

```
==

QDialog
.
DialogCode
.
Accepted
:

fmt

=

d
.
fmt
.
lower
()

path

=

choose_save_file
(
self
.
gui
,

'edit-book-new-book'
,

—
(
'Choose file location'
),

filters
```

```
=
[(
fmt
.
upper
()),

(
fmt
,))],

all_files
=
False
)

if

path

is

not

None
:

if

not

path

.
lower
()

.
endswith
(
'.'
)

+

```

```
fmt
):
```

```
path
```

```
=
```

```
path
```

```
+
```

```
'.'
```

```
+
```

```
fmt
```

```
from
```

```
calibre.ebooks.oeb.polish.create
```

```
import
```

```
create_book
```

```
create_book
```

```
(
d
```

```
·
mi
```

```
,
```

```
path
```

```
,
```

```
fmt
```

```
=
```

```
fmt
```

```
)
```

```
self
.  
open_book  
(  
path  
=  
path  
)
```

```
def
```

```
import_book  
(  
self  
,  
  
path  
=  
None  
):
```

```
if
```

```
not
```

```
self  
.  
_check_before_open  
():
```

```
return
```

```
d
```

```
=
```



```
ImportForeign
```

```
(  
self
```

```
.
```

```
gui
```

```
)
```

```
if
```

```
hasattr
```

```
(
```

```
path
```

```
,
```

```
'rstrip'
```

```
):
```

```
d
```

```
.
```

```
set_src
```

```
(
```

```
os
```

```
.
```

```
path
```

```
.
```

```
abspath
```

```
(
```

```
path
```

```
))
```

```
if
```

```
d
```

```
.
```

```
exec
```

```
()
```

```
==
```

```
QDialog
```

```
.
DialogCode

.
Accepted
:

for

name

in

tuple
(
editors
):

self

.
close_editor
(
name
)

from

calibre.ebooks.oeb.polish.import\_book

import

import_book_as_epub

src
,

dest

=
```

```
d
.
data

self
.
_clear_notify_data

=

True


def

func
(
src
,

dest
,

tdir
):

import_book_as_epub
(
src
,

dest
)

return

get_container
(
dest
```

```
,  
  
tdir  
=  
tdir  
)  
  
self  
.gui  
.blocking_job  
(  
    'import_book'  
,  
  
    _  
(  
    'Importing book, please wait...'  
),  
  
self  
.book_opened  
,  
  
func  
,  
  
src  
,  
  
dest  
,  
  
tdir  
=  
self  
.mkdtemp  
())
```

[\[docs\]](#)

def

open_book

(

self

,

path

=

None

,

edit_file

=

None

,

clear_notify_data

=

True

,

open_folder

=

False

,

search_text

=

None

):

'''

Open the e-book at ``path`` for editing. Will show an error if the e-book is not in a supported format or the current book has unsaved changes.

:param edit_file: The name of a file inside the newly opened book to start editing. Can also be a list of names.

```
'''

if

isinstance
(
path
,
(
list
,
tuple
))

and

path
:

# Can happen from an file_event_hook on OS X when drag and dropping

# onto the icon in the dock or using open -a

path

=

path
[
-
1
]
```

```
if
not
self
.
_check_before_open
():

return

if
not
hasattr
(
path
,
'partition'
):

if
open_folder
:

path

=

choose_dir
(
self
.
gui
,
```

```
'open-book-folder-for-tweaking'
```

```
,
```

```
-  
(
```

```
'Choose book folder'
```

```
))
```

```
if
```

```
path
```

```
:
```

```
path
```

```
=
```

```
[
```

```
path
```

```
]
```

```
else
```

```
:
```

```
path
```

```
=
```

```
choose_files
```

```
(
```

```
self
```

```
.
```

```
gui
```

```
,
```

```
'open-book-for-tweaking'
```

```
,
```



```
-  
(  
    'Choose book'  
)  
  
[(  
    -  
    (  
        'Books'  
    ),  
  
    [  
        x  
        .  
        lower  
        ()  
  
        for  
  
        x  
  
        in  
  
        SUPPORTED  
    ]]),  
  
    all_files  
    =  
    False  
    ,  
  
    select_only_single_file  
    =  
    True  
    )  
  
if  
  
not
```

```
path
:

return

path

=

path
[
0
]

if

not

os
.
path
.
exists
(
path
):

return

error_dialog
(
self
.
gui
,
_
(
```

```
'File not found'
```

```
),
```

```
-  
(
```

```
'The file
```

```
%s
```

```
does not exist.'
```

```
)
```

```
%
```

```
path
```

```
,
```

```
show
```

```
=
```

```
True
```

```
)
```

```
isdir
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
isdir
```

```
(
```

```
path
```

```
)
```

```
ext
```

```
=
```

```
path
```

```
.
```

```
rpartition
(
    '
'
)[
-
1
]
.
upper
()

if

ext

not

in

SUPPORTED

and

not

isdir
:

from

calibre.ebooks.oeb.polish.import_book

import

IMPORTABLE

if

ext

.
```

```
lower
()

in

IMPORTABLE
:

return

self
.
import_book
(
path
)

return

error_dialog
(
self
.
gui
,

—
(
'Unsupported format'
),

—
(
'Tweaking is only supported for books in the
%s
formats.'

' Convert your book to one of these formats first.'
)
```

```
%  
  
_  
(  
'and '  
)  
.  
join  
(  
sorted  
(  
SUPPORTED  
)),  
  
show  
=  
True  
)  
  
for  
  
name  
  
in  
  
tuple  
(  
editors  
):  
  
self  
.  
close_editor  
(  
name  
)
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
preview
```

```
.
```

```
clear
```

```
()
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
live_css
```

```
.
```

```
clear
```

```
()
```

```
self
```

```
.
```

```
container_count
```

```
=
```

```
-
```

```
1
```

```
if
```

```
self
```

```
.
```

```
tdir
```

```
:
```

```
shutil
```

```
.
```

```
rmtree
```

```
(
```

```
self
```

```
.
tdir
,

ignore_errors
=
True
)

# We use the cache dir rather than the temporary dir to try and prevent

# temp file cleaners from nuking ebooks. See https://bugs.launchpad.net/bugs/1740460

self
.
tdir

=

tdir_in_cache
(
'ee'
)

self
.
_edit_file_on_open

=

edit_file

self
.
_search_text_on_open

=
```



```
search_text

self
.
_clear_notify_data

=

clear_notify_data

self
.
gui
.
blocking_job
(
'open_book'
,

_
(
'Opening book, please wait...'
),

self
.
book_opened
,

get_container
,

path
,

tdir
=
self
.
mkdtemp
```

```
((  
  
def  
  
book_opened  
(  
self  
,  
  
job  
):  
  
    ef  
  
    =  
  
    getattr  
    (  
    self  
    ,  
  
    '_edit_file_on_open'  
    ,  
  
    None  
    )  
  
    cn  
  
    =  
  
    getattr  
    (  
    self  
    ,  
  
    '_clear_notify_data'  
    ,  
  
    True
```

```
)

st

=

getattr
(
self
,

'_search_text_on_open'
,

None
)

self

.

_edit_file_on_open

=

self

.

_search_text_on_open

=

None


if

job

.

traceback

is
```

not

None

:

if

'DRMError:'

in

job

.

traceback

:

from

calibre.gui2.dialogs.drm_error

import

DRMErrorMessage

return

DRMErrorMessage

(

self

.

gui

)

.

exec

()

if

'ObfuscationKeyMissing:'

```
in
```

```
job
```

```
.
```

```
traceback
```

```
:
```

```
return
```

```
error_dialog
```

```
(
```

```
self
```

```
.
```

```
gui
```

```
,
```

```
—  
(
```

```
'Failed to open book'
```

```
),
```

```
—  
(
```

```
'Failed to open book, it has obfuscated fonts, but the obfuscation key is missing from the OPF.'
```

```
' Do an EPUB to EPUB conversion before trying to edit this book.'
```

```
),
```

```
show
```

```
=
```

```
True
```

```
)
```

```
return
```

```
error_dialog
```

```
(
self
.
gui
,

_
(
'Failed to open book'
),

_
(
'Failed to open book, click "Show details" for more information.'
),

det_msg
=
job
.
traceback
,

show
=
True
)

if

cn
:

self
.
save_manager
.
clear_notify_data
()
```

```
self
.  
gui  
.  
check_book  
.  
clear_at_startup  
()
```

```
self  
.  
gui  
.  
spell_check  
.  
clear_caches  
()
```

```
dictionaries  
.  
clear_ignored  
(),
```

```
dictionaries  
.  
clear_caches  
()
```

```
parse_worker  
.  
clear  
()
```

```
container
```

```
=
```

```
job
```

```
.
```

```
result
```

```
set_current_container
```

```
(
```

```
container
```

```
)
```

```
completion_worker
```

```
()
```

```
.
```

```
clear_caches
```

```
()
```

```
with
```

```
BusyCursor
```

```
():
```

```
self
```

```
.
```

```
current_metadata
```

```
=
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
current_metadata
```

```
=
```

```
container
```

```
.
```

```
mi
```



```
lang
=
container
.
opf_xpath
(
'//dc:language/text()'
)
```

or

```
[
self
.
current_metadata
.
language
]
```

```
set_book_locale
(
lang
[
0
])
```

```
self
.
global_undo
.
open_book
(
container
)
```

```
self
.
gui
```

```
.  
update_window_title  
(
```

```
self
```

```
.  
gui
```

```
.  
file_list
```

```
.  
current_edited_name
```

```
=
```

```
None
```

```
self
```

```
.  
gui
```

```
.  
file_list
```

```
.  
build
```

```
(  
container
```

```
,
```

```
preserve_state
```

```
=
```

```
False
```

```
)
```

```
self
```

```
.  
gui
```

```
.  
action_save
```

```
.  
setEnabled
```

```
(
```

False

)

self

.

update_global_history_actions

()

recent_books

=

list

(

tprefs

.

get

(

'recent-books'

,

[]))

path

=

os

.

path

.

abspath

(

container

.

path_to_ebook

)

if

```
path
```

```
in
```

```
recent_books  
:
```

```
recent_books
```

```
.
```

```
remove
```

```
(
```

```
path
```

```
)
```

```
recent_books
```

```
.
```

```
insert
```

```
(
```

```
0
```

```
,
```

```
path
```

```
)
```

```
tprefs
```

```
[
```

```
'recent-books'
```

```
]
```

```
=
```

```
recent_books
```

```
[:
```

```
10
```

```
]
```

```
self
```

```
.
```

```
gui
.  
update_recent_books  
(
```

```
if
```

```
iswindows  
:
```

```
try  
:
```

```
add_to_recent_docs  
(  
path  
)
```

```
except
```

```
Exception  
:
```

```
import
```

```
traceback
```

```
traceback  
.  
print_exc  
(
```

```
if
```

```
ef  
:
```

```
if
    isinstance
    (
        ef
        ,
        str
    ):

    ef

    =

    [
        ef
    ]

for
    i
    in
        ef
        :

    self
    .
    gui
    .
    file_list
    .
    request_edit
    (
        i
    )
```

```
else
```

```
:
```

```
if
```

```
tprefs
```

```
[
```

```
'restore_book_state'
```

```
]:
```

```
self
```

```
.
```

```
restore_book_edit_state
```

```
()
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
toc_view
```

```
.
```

```
update_if_visible
```

```
()
```

```
self
```

```
.
```

```
add_savepoint
```

```
(
```

```
—
```

```
(
```

```
'Start of editing session'
```

```
))
```

```
if
```

```
st
```

```
:
```

```
self
.
find_initial_text
(
st
)

def

update_editors_from_container
(
self
,

container
=
None
,

names
=
None
):

c

=

container

or

current_container
()

for
```



```
name
',
ed

in

tuple
(
iteritems
(
editors
)):

if

c
.
has_name
(
name
):

if

names

is

None

or

name

in

names
:
```

```
ed
.
replace_data
(
c
.
raw_data
(
name
))

ed
.
is_synced_to_container

=

True

else
:

self
.
close_editor
(
name
)

def

refresh_file_list
(
self
):

container
```

```
=
```

```
current_container  
(
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
file_list
```

```
.
```

```
build
```

```
(
```

```
container
```

```
)
```

```
completion_worker
```

```
(
```

```
.
```

```
clear_caches
```

```
(
```

```
'names'
```

```
)
```

```
\[docs\]
```

```
def
```

```
apply_container_update_to_gui
```

```
(
```

```
self
```

```
,
```

```
mark_as_modified
```

```
=
```

```
True
```

```
):
```

```
'''
```

```
    Update all the components of the user interface to reflect the latest data in the current book container.
```

```
    :param mark_as_modified: If True, the book will be marked as modified, so the user will be prompted to save it
```

```
    when quitting.
```

```
'''
```

```
self
```

```
.
```

```
refresh_file_list
```

```
()
```

```
self
```

```
.
```

```
update_global_history_actions
```

```
()
```

```
self
```

```
.
```

```
update_editors_from_container
```

```
()
```

```
if
```

```
mark_as_modified
```

```
:
```

```
self
```

```
.
```

```
set_modified
```

```
()

self
.
gui
.
toc_view
.
update_if_visible
()

completion_worker
()
.
clear_caches
()

self
.
gui
.
preview
.
start_refresh_timer
()

@in_thread_job

def

delete_requested
(
self
,

spine_items
,
```

```
other_items
):

self
.
add_savepoint
(
_
(
'Before: Delete files'
))

self
.
commit_dirty_opf
()

c
=

current_container
()

c
.
remove_from_spine
(
spine_items
)

for
name
in
other_items
```

```
:  
  
c  
.  
remove_item  
(  
name  
)
```

```
self  
.  
set_modified  
()
```

```
self  
.  
gui  
.  
file_list  
.  
delete_done  
(  
spine_items  
,  
other_items  
)
```

```
spine_names
```

```
=
```

```
[  
x
```

```
for
```

```
x
```

```
,
```

```
remove
```

```
in
```

```
spine_items
```

```
if
```

```
remove
```

```
]
```

```
completion_worker
```

```
()
```

```
.
```

```
clear_caches
```

```
(
```

```
'names'
```

```
)
```

```
items
```

```
=
```

```
spine_names
```

```
+
```

```
list
```

```
(
```

```
other_items
```

```
)
```

```
for
```

```
name
```

```
in
```

```
items
```



```
:  
  
if  
    name  
  
in  
    editors  
    :  
  
    self  
    .  
    close_editor  
    (  
    name  
    )  
  
if  
  
not  
    editors  
    :  
  
    self  
    .  
    gui  
    .  
    preview  
    .  
    clear  
    ()  
  
    self  
    .  
    gui  
    .
```

```
live_css
.  
clear  
()  
  
changed  
  
=  
  
remove_names_from_toc  
(  
current_container  
()),  
  
spine_names  
  
+  
  
list  
(  
other_items  
))  
  
if  
  
changed  
:  
  
self  
.  
gui  
.  
toc_view  
.  
update_if_visible  
()  
  
for
```

toc

in

changed

:

if

toc

and

toc

in

editors

:

editors

[

toc

]

.

replace_data

(

c

.

raw_data

(

toc

))

if

c

.

opf_name

in

editors

:

editors

[

c

.

opf_name

]

.

replace_data

(

c

.

raw_data

(

c

.

opf_name

))

self

.

gui

.

message_popup

(

ngettext

(

'One file deleted'

,

,

}

files deleted'

,

```
len
(  
items  
)  
.  
format  
(  
len  
(  
items  
)))
```

```
def
```

```
commit_dirty_opf  
(  
self  
):
```

```
c
```

```
=
```

```
current_container  
()
```

```
if
```

```
c
```

```
.  
opf_name
```

```
in
```

```
editors
```

```
and
```

```
not
```

```
editors
[
c
.
opf_name
]
.
is_synced_to_container
:

self
.
commit_editor_to_container
(
c
.
opf_name
)

self
.
gui
.
update_window_title
()

def

reorder_spine
(
self
,

items
):

if
```

not

self

.

ensure_book

():

return

self

.

add_savepoint

(

—
(

'Before: Re-order text'

))

c

=

current_container

()

c

.

set_spine

(

items

)

self

.

set_modified

()

```
self
.
gui
.
file_list
.
build
(
current_container
())

# needed as the linear flag may have changed on some items

if

c
.
opf_name

in

editors
:

editors
[
c
.
opf_name
]
.
replace_data
(
c
.
raw_data
(
c
.
opf_name
```



```
))

completion_worker
()
.
clear_caches
(
'names'
)

def

add_file
(
self
):

if

not

self
.
ensure_book
(
-
(
'You must first open a book to edit, before trying to create new files in it.'
)):

return

self
.
commit_dirty_opf
()
```

```
d
=
NewFileDialog
(
self
.
gui
)
```

```
if
```

```
d
.
exec
()
```

```
!=
```

```
QDialog
.
DialogCode
.
Accepted
:
```

```
return
```

```
added_name
```

```
=
```

```
self
.
do_add_file
(
d
.

```

```
file_name
,
d
.
file_data
,
using_template
=
d
.
using_template
,
edit_file
=
True
)
```

```
if
```

```
d
.
file_name
.
rpartition
(
'.'
)[
2
]
.
lower
()
```

```
in
```

```
(
'ttf'
,
```

```
'otf'
',

'woff'
):

from

calibre.gui2.tweak_book.manage_fonts

import

show_font_face_rule_for_font_file

show_font_face_rule_for_font_file
(
d
.
file_data
,

added_name
,

self
.
gui
)

def

do_add_file
(
self
,

file_name
,
```

```
data
,

using_template
=
False
,

edit_file
=
False
):

self
.
add_savepoint
(
_
(
'Before: Add file
%s
'
)

%

self
.
gui
.
elided_text
(
file_name
))

c

=

current_container
()
```

```
adata
=
data
.
replace
(
b
'%CURSOR%'
,
b
"
)
if
using_template
else
data
spine_index
=
c
.
index_in_spine
(
self
.
currently_editing
or
"
)
```

if

spine_index

is

not

None

:

spine_index

+=

1

try

:

added_name

=

c

.

add_file

(

file_name

,

adata

,

spine_index

=

spine_index

)

```
except
```

```
:
```

```
self
```

```
.
```

```
rewind_savepoint
```

```
()
```

```
raise
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
file_list
```

```
.
```

```
build
```

```
(
```

```
c
```

```
)
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
file_list
```

```
.
```

```
select_name
```

```
(
```

```
file_name
```

```
)
```

```
if
```

```
c
```

```
.
```


opf_name

in

editors

:

editors

[

c

.

opf_name

]

.

replace_data

(

c

.

raw_data

(

c

.

opf_name

))

mt

=

c

.

mime_map

[

file_name

]

syntax

=

```
syntax_from_mime
```

```
(
```

```
file_name
```

```
,
```

```
mt
```

```
)
```

```
if
```

```
syntax
```

```
and
```

```
edit_file
```

```
:
```

```
if
```

```
using_template
```

```
:
```

```
self
```

```
.
```

```
edit_file
```

```
(
```

```
file_name
```

```
,
```

```
syntax
```

```
,
```

```
use_template
```

```
=
```

```
data
```

```
.
```

```
decode
```

```
(
```

```
'utf-8'
```

```
))
```

else

:

self

.

edit_file

(

file_name

,

syntax

)

self

.

set_modified

()

completion_worker

()

.

clear_caches

(

'names'

)

return

added_name

def

add_files

(

self

```
):  
  
if  
not  
self  
.ensure_book  
(  
    _  
(  
    'You must first open a book to edit, before trying to create new files in it.'  
)):  
  
return  
  
files  
  
=  
  
choose_files  
(  
    self  
.gui  
,  
  
    'tweak-book-bulk-import-files'  
,  
  
    _  
(  
    'Choose files'  
))  
  
if
```

```
files
:

folder_map

=

get_recommended_folders
(
current_container
()),

files
)
```

```
files

=

{
x
: (
'/'
.
join
((
folder
,

os
.
path
.
basename
(
x
)))
```

```
if
```

```
folder
```

else

os

.

path

.

basename

(

x

))

for

x

,

folder

in

iteritems

(

folder_map

)}

self

.

add_savepoint

(

—

(

'Before Add files'

))

c

=

current_container

```
()

added_fonts

=

set
()

for

path

in

sorted
(
files
,
key
=
numeric_sort_key
):

name

=

files
[
path
]

i

=

0
```

while

c

.

exists

(

name

)

or

c

.

manifest_has_name

(

name

)

or

c

.

has_name_case_insensitive

(

name

):

i

+=

1

name

,

ext

=

name

.

rpartition

(

'

)

0

::

2

]

name

=

'

%s

—

%d

.

%s

'

%

(

name

,

i

,

ext

)

try

:

with

```
open
(
path
,

'rb'
)

as

f
:

c
.
add_file
(
name
,

f
.
read
())

except
:

self
.
rewind_savepoint
()

raise

if
```

```
name
```

```
.
```

```
rpartition
```

```
(
```

```
[''
```

```
)[
```

```
2
```

```
]
```

```
.
```

```
lower
```

```
()
```

```
in
```

```
(
```

```
'ttf'
```

```
,
```

```
'otf'
```

```
,
```

```
'woff'
```

```
):
```

```
added_fonts
```

```
.
```

```
add
```

```
(
```

```
name
```

```
)
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
file_list
```

```
.
```

```
build
```

```
(
```

```
c
```

```
)
```

```
if
```

```
c
```

```
.
```

```
opf_name
```

```
in
```

```
editors
```

```
:
```

```
editors
```

```
[
```

```
c
```

```
.
```

```
opf_name
```

```
]
```

```
.
```

```
replace_data
```

```
(
```

```
c
```

```
.
```

```
raw_data
```

```
(
```

```
c
```

```
.
```

```
opf_name
```

```
))
```

```
self
```

```
.
```

```
set_modified
```

```
()
```

```
completion_worker
```

```
()
```

```
.
```

```
clear_caches
```

```
(
'names'
)

if

added_fonts
:

from

calibre.gui2.tweak_book.manage_fonts

import

show_font_face_rule_for_font_files

show_font_face_rule_for_font_files
(
c
,
added_fonts
,
self
.
gui
)

def

add_cover
(
self
):
```

```
if
    not
        self
            .
            ensure_book
            ():

    return

    d
    =
    AddCover
    (
        current_container
        ()),

    self
        .
        gui
        )

    d
        .
        import_requested
        .
        connect
        (
            self
            .
            do_add_file
            )

try
:
```

```
if
d
·
exec
()

==

QDialog
·
DialogCode
·
Accepted

and

d
·
file_name

is

not

None
:

report

=

[]

with

BusyCursor
():
```

```
self
.
add_savepoint
(
_
(
'Before: Add cover'
))

set_cover
(
current_container
()),

d
.
file_name
,

report
.
append
,

options
=
{

'existing_image'
:
True
,

'keep_aspect'
:
tprefs
[
'add_cover_preserve_aspect_ratio'
])})
```



```
self
    .
    apply_container_update_to_gui
    ()

finally
    :

    d
    .
    import_requested
    .
    disconnect
    ()

def
ensure_book
(
    self
    ,

    msg
    =
    None
):

    msg

    =

    msg

    or

    _
    (
    'No book is currently open. You must first open a book.'
```

```
)

if

current_container
()

is

None
:

error_dialog
(
  self
  .
  gui
  ,

  _
  (
    'No book open'
  ),

  msg
  ,

  show
  =
  True
)

return

False

return

True
```

```
def
edit_toc
(
self
):

if
not
self
.
ensure_book
(
_
(
'You must open a book before trying to edit the Table of Contents.'
)):

return

self
.
add_savepoint
(
_
(
'Before: Edit Table of Contents'
))

d
=
TOCEditor
```

```
(
    title
    =
    self
    .
    current_metadata
    .
    title
    ,

    parent
    =
    self
    .
    gui
)

if

d
.
exec
()

!=

QDialog
.
DialogCode
.
Accepted
:

self
.
rewind_savepoint
()

return
```

with

BusyCursor
():

self

.
set_modified
()

self

.
update_editors_from_container
()

self

.
gui
.
toc_view
.
update_if_visible
()

self

.
gui
.
file_list
.
build
(
current_container
())

def

```
insert_inline_toc
(
  self
):

if
not
self
.
ensure_book
():

return

self
.
commit_all_editors_to_container
()

self
.
add_savepoint
(
  _
(
    'Before: Insert inline Table of Contents'
  ))

name

=

create_inline_toc
(
  current_container
```

```
    ())

    if
    name

    is
    None
    :

    self
    .
    rewind_savepoint
    ()

    return

    error_dialog
    (
    self
    .
    gui
    ,

    —
    (
    'No Table of Contents'
    ),

    —
    (

    'Cannot create an inline Table of Contents as this book has no existing'

    ' Table of Contents. You must first create a Table of Contents using the'
```

```
' Edit Table of Contents tool.'
),

show
=
True
)

self
.
apply_container_update_to_gui
()

self
.
edit_file
(
name
,

'html'
)

def

polish
(
self
,

action
,

name
,

parent
=
None
```



```
):  
  
if  
not  
self  
.ensure_book  
():  
  
return  
  
from  
calibre.gui2.tweak_book.polish  
import  
get_customization  
,  
show_report  
  
customization  
=  
get_customization  
(  
action  
,  
name  
,  
parent  
or
```

```
self
.
gui
)

if
customization

is
None
:

return

with
BusyCursor
():

self
.
add_savepoint
(
_
(
'Before:
%s
'
)
%

name
)
```

```
try
:

report
,

changed

=

tweak_polish
(
current_container
()),

{
action
:
True
},

customization
=
customization
)

except
:

self
.
rewind_savepoint
()

raise

if
```

```
changed
:

self
.
apply_container_update_to_gui
()

self
.
gui
.
update_window_title
()

if

not

changed
:

self
.
rewind_savepoint
()

show_report
(
changed
,

self
.
current_metadata
.
title
,
```

```
report
,
parent

or

self
.
gui
,

self
.
show_current_diff
)


def

transform_html
(
self
):


global

last_used_html_transform_rules


if

not

self
.
ensure_book
(
-
(
```

'You must first open a book in order to transform styles.'

)):

return

from

calibre.gui2.html_transform_rules

import

RulesDialog

from

calibre.ebooks.html_transform_rules

import

transform_container

d

=

RulesDialog

(

self

.

gui

)

d

.

rules

=

```
last_used_html_transform_rules
```

```
d
```

```
.  
transform_scope
```

```
=
```

```
tprefs
```

```
[  
  'html_transform_scope'  
]
```

```
ret
```

```
=
```

```
d
```

```
.  
exec  
()
```

```
last_used_html_transform_rules
```

```
=
```

```
d
```

```
.  
rules
```

```
scope
```

```
=
```

```
d
```

```
.  
transform_scope
```

```
tprefs
.  
set  
(  
  'html_transform_scope'  
,  
  
scope  
)
```

```
if
```

```
ret
```

```
!=
```

```
QDialog
```

```
.  
DialogCode
```

```
.  
Accepted  
:
```

```
return
```

```
mime_map
```

```
=
```

```
current_container  
()
```

```
.  
mime_map
```

```
names
```

```
=
```



```
()

if

scope

==

'current'
:

if

not

self
.
currently_editing

or

mime_map
.
get
(
self
.
currently_editing
)

not

in

OEB_DOCS
:

return

error_dialog
```

```
(
self
.
gui
,

-
(
'No HTML file'
),

-
(
'Not currently editing an HTML file'
),

show
=
True
)

names

=

(
self
.
currently_editing
,)

elif

scope

==

'open'
:
```

```
names

=

tuple
(
name

for

name

in

editors

if

mime_map
.
get
(
name
)

in

OEB_DOCS
)

if

not

names
:

return

error_dialog
(
```

```
self
.
gui
,

-
(
'No HTML files'
),

-
(
'Not currently editing any HTML files'
),

show
=
True
)

elif

scope

==

'selected'
:

names

=

tuple
(
name

for

name
```

```
in

self
.
gui
.
file_list
.
file_list
.
selected_names
```

```
if

mime_map
.
get
(
name
)

in

OEB_DOCS
)

if

not

names
:
```

```
return

error_dialog
(
self
.
gui
,
```

```
-
(
'No HTML files'
),

-
(
'No HTML files are currently selected in the File browser'
),

show
=
True
)

with

BusyCursor
():

self
.
add_savepoint
(
-
(
'Before HTML transformation'
))

try
:

changed

=

transform_container
(
```

```
current_container  
(  
  
last_used_html_transform_rules  
,  
  
names  
)
```

```
except  
:
```

```
self  
.  
rewind_savepoint  
()
```

```
raise
```

```
if
```

```
changed  
:
```

```
self  
.  
apply_container_update_to_gui  
()
```

```
if
```

```
not
```

```
changed  
:
```

```
self
.
rewind_savepoint
()

return

info_dialog
(
self
.
gui
,

—
(
'No changes'
),

—
(
'No HTML was changed.'
),

show
=
True
)

self
.
show_current_diff
()

def

transform_styles
(
self
```



```
):  
  
global  
last_used_transform_rules  
  
if  
not  
self  
.ensure_book  
(  
_(  
'You must first open a book in order to transform styles.'  
)):  
  
return  
  
from  
calibre.gui2.css_transform_rules  
  
import  
RulesDialog  
  
from  
calibre.ebooks.css_transform_rules  
  
import  
transform_container
```

```
d
=
RulesDialog
(
self
.
gui
)

d
.
rules
=
last_used_transform_rules

ret
=
d
.
exec
()

last_used_transform_rules
=
d
.
rules

if
ret
```

```
!=

QDialog
·
DialogCode
·
Accepted
:

return

with

BusyCursor
():

self
·
add_savepoint
(
—
(
'Before style transformation'
))

try
:

changed

=

transform_container
(
current_container
()),
```

```
last_used_transform_rules
)

except
:

self
.
rewind_savepoint
()

raise

if

changed
:

self
.
apply_container_update_to_gui
()

if

not

changed
:

self
.
rewind_savepoint
()
```

```
info_dialog
(
self
.
gui
,

_
(
'No changes'
),

_
(

'No styles were changed.'
),

show
=
True
)

return

self
.
show_current_diff
()

def

get_external_resources
(
self
):
```

```
if
not
self
.
ensure_book
(
_
(
'You must first open a book in order to transform styles.'
)):

return

from
calibre.gui2.tweak_book.download

import
DownloadResources

with
BusyCursor
():

self
.
add_savepoint
(
_
(
'Before: Get external resources'
))

try
```

```
:  
  
d  
  
=  
  
DownloadResources  
(  
    self  
    .  
    gui  
)  
  
d  
    .  
    exec  
    ()  
  
except  
  
Exception  
:  
  
    self  
    .  
    rewind_savepoint  
    ()  
  
raise  
  
if  
  
d  
    .  
    resources_replaced  
    :
```

```
self
.  
apply_container_update_to_gui  
(
```

```
if
```

```
d
```

```
.  
show_diff  
:
```

```
self
```

```
.  
show_current_diff  
(
```

```
else
```

```
:
```

```
self
```

```
.  
rewind_savepoint  
(
```

```
def
```

```
manage_fonts  
(  
self  
):
```

```
if
```

```
not
```



```
self
.  
ensure_book  
(  
_  
(  
'No book is currently open. You must first open a book to manage fonts.'  
)):
```

```
return
```

```
self  
.  
commit_all_editors_to_container  
()
```

```
self  
.  
gui  
.  
manage_fonts  
.  
display  
()
```

```
def
```

```
manage_fonts_embed  
(  
self  
):
```

```
self  
.  
polish  
(
```

```
'embed'
,

-
(
'Embed all fonts'
),

parent
=
self
.
gui
.
manage_fonts
)

self
.
gui
.
manage_fonts
.
refresh
()

def

manage_fonts_subset
(
self
):

self
.
polish
(
'subset'
,
```

```
–  
(  
    'Subset all fonts'  
),
```

```
parent  
=  
self  
.  
gui  
.  
manage_fonts  
)
```

```
# Renaming {{{
```

```
def
```

```
rationalize_folders  
(  
    self  
):
```

```
if
```

```
not
```

```
self  
.  
ensure_book  
():
```

```
return
```

```
c
```

```
=

current_container
()

if
not

c
.
SUPPORTS_FILENAMES
:

return

error_dialog
(
self
.
gui
,

—
(
'Not supported'
),

—
(
'The
%s
format does not support file and folder names internally, therefore'

'arranging files into folders is not allowed.'
)

%
```

```
c
.
book_type
.
upper
(),

show
=
True
)

d

=

RationalizeFolders
(
self
.
gui
)

if

d
.
exec
()

!=

QDialog
.
DialogCode
.
Accepted
:
```

```
return

self
.
commit_all_editors_to_container
()

name_map

=

rationalize_folders
(
c
,

d
.
folder_map
)

if

not

name_map
:

confirm
(
-
(

'The files in this book are already arranged into folders'
),

'already-arranged-into-folders'
,
```

```
self
.  
gui  
,  
  
pixmap  
=  
'dialog_information.png'  
,  
  
title  
=  
—  
(  
'Nothing to do'  
)  
,  
  
show_cancel_button  
=  
False  
,  
  
config_set  
=  
tprefs  
,  
  
confirm_msg  
=  
—  
(  
'Show this message &again'  
)  
  
return  
  
self  
.
```

```
add_savepoint
(
    _
(
    'Before: Arrange into folders'
))

self
.
gui
.
blocking_job
(

    'rationalize_folders'
,

    _
(
    'Renaming and updating links...'
),

    partial
(
        self
        .
        rename_done
    ,

        name_map
    ),

    rename_files
,

    current_container
    (),

    name_map
)
```



```
def
```

```
    rename_requested
```

```
    (
```

```
        self
```

```
    ,
```

```
        oldname
```

```
    ,
```

```
        newname
```

```
    ):
```

```
        self
```

```
        .
```

```
        commit_all_editors_to_container
```

```
        ()
```

```
if
```

```
    guess_type
```

```
    (
```

```
        oldname
```

```
    )
```

```
    !=
```

```
    guess_type
```

```
    (
```

```
        newname
```

```
    ):
```

```
    args
```

```
    =
```

```
os
```

```
.
path
.
splitext
(
oldname
)

+

os
.
path
.
splitext
(
newname
)

if

not

confirm
(

-
(
'You are changing the file type of
{0}
<b>
{1}
</b> to
{2}
<b>
{3}
</b>.'

' Doing so can cause problems, are you sure?'
)
```

```
.
format
(
*
args
),

'confirm-filetype-change'
,

parent
=
self
.
gui
,

title
=
_
(
'Are you sure?'
),

config_set
=
tprefs
):

return

if

urlnormalize
(
newname
)

!=
```

```
newname
:

if

not

confirm
(

_
(
'The name you have chosen
{0}
contains special characters, internally'

' it will look like:
{1}
Try to use only the English alphabet [a-z], numbers [0-9], '

' hyphens and underscores for file names. Other characters can cause problems for '

' different e-book viewers. Are you sure you want to proceed?'
)
.
format
(

'<pre>
%s
</pre>'
%
newname
,

'<pre>
```

```
%s
</pre>'

%

urlnormalize
(
newname
)),

'confirm-urlunsafe-change'
,

parent
=
self
.
gui
,

title
=
-
(
'Are you sure?'
),

config_set
=
tprefs
):

return

self
.
add_savepoint
(
-
(
```

```
'Before: Rename
%s
'
)

%

oldname
)

name_map

=

{
oldname
:
newname
}

self
.
gui
.
blocking_job
(

'rename_file'
,

_
(
'Renaming and updating links...'
),

partial
(
self
.
rename_done
```

```
,  
  
name_map  
,  
  
from_filelist  
=  
self  
.  
gui  
.  
file_list  
.  
current_name  
,  
  
rename_files  
,  
  
current_container  
(  
  
name_map  
)  
  
  
def  
  
bulk_rename_requested  
(  
self  
,  
  
name_map  
)  
:  
  
self  
.  
add_savepoint  
(
```

```
-  
(  
    'Before: Bulk rename'  
))  
  
self  
.  
gui  
.  
blocking_job  
(  
  
    'bulk_rename_files'  
    ,  
  
    -  
    (  
        'Renaming and updating links...'  
    ),  
  
    partial  
    (  
        self  
        .  
        rename_done  
    ,  
  
    name_map  
    ,  
  
    from_filelist  
    =  
    self  
    .  
    gui  
    .  
    file_list  
    .  
    current_name  
    ),
```



```
rename_files
',

current_container
()),

name_map
)
```

```
def
```

```
rename_done
(
self
',

name_map
',

job
',

from_filelist
=
None
):
```

```
if
```

```
job
.
traceback
```

```
is
```

```
not
```

```
None
:
```

```
return

error_dialog
(
    self
    .
    gui
    ,

    _
    (
        'Failed to rename files'
    ),

    _
    (
        'Failed to rename files, click "Show details" for more information.'
    ),

    det_msg
    =
    job
    .
    traceback
    ,

    show
    =
    True
)

self
.
gui
.
file_list
.
build
```

```
(  
current_container  
)
```

```
self  
.  
set_modified  
(
```

```
for
```

```
oldname  
,  
newname
```

```
in
```

```
iteritems  
(  
name_map  
):
```

```
if
```

```
oldname
```

```
in
```

```
editors  
:
```

```
editors  
[  
newname  
]
```

```
=
```

```
ed

=

editors
.
pop
(
oldname
)

ed
.
change_document_name
(
newname
)

self
.
gui
.
central
.
rename_editor
(
editors
[
newname
],
newname
)

if

self
.
gui
.
```

```
preview
·
current_name

==

oldname
:

self
·
gui
·
preview
·
current_name

=

newname

self
·
apply_container_update_to_gui
()

if

from_filelist
:

self
·
gui
·
file_list
·
select_names
(
```

```
frozenset
(
intervalvalues
(
name_map
)),

current_name
=
name_map
.
get
(
from_filelist
))

self
.
gui
.
file_list
.
file_list
.
setFocus
(
Qt
.
FocusReason
.
PopupFocusReason
)

# }}}

# Global history {{{
```

```
def

do_global_undo
(
self
):

container

=

self
.
global_undo
.
undo
()

if

container

is

not

None
:

set_current_container
(
container
)

self
.
apply_container_update_to_gui
()
```

```
def

do_global_redo
(
self
):

container

=

self
.
global_undo
.
redo
()

if

container

is

not

None
:

set_current_container
(
container
)

self
.
apply_container_update_to_gui
```



```
()

def
update_global_history_actions
(
self
):

gu

=

self
.
global_undo

for

x
,

text

in

((
'undo'
,

_
(
'&Revert to'
)),

(
'redo'
,

```

```
-
(
'&Revert to'
)))

ac

=

getattr
(
self
.
gui
,
'action_global_'
's
'

%

x
)

ac
.
setEnabled
(
getattr
(
gu
,
'can_'

+

x
))
```

```
ac
.
setText
(
text
```

```
+
```

```
' "
```

```
%s
```

```
""
```

```
%
```

```
(
getattr
```

```
(
gu
```

```
,
```

```
x
```

```
+
```

```
'_msg'
```

```
)
```

```
or
```

```
'...'
```

```
))
```

```
\[docs\]
```

```
def
```

```
add_savepoint
```

```
(
```

```
self
```

```
,
```

```
msg  
):
```

```
' Create a restore checkpoint with the name specified as ``msg`` '
```

```
self
```

```
.  
commit_all_editors_to_container  
(
```

```
nc
```

```
=
```

```
clone_container  
(  
current_container  
()),
```

```
self
```

```
.  
mkdtemp  
())
```

```
self
```

```
.  
global_undo  
.  
add_savepoint  
(  
nc  
,  
msg  
)
```

```
set_current_container  
(
```

```
nc
)

self
.
update_global_history_actions
()
```

[\[docs\]](#)

def

```
rewind_savepoint
(
self
):
```

' Undo the previous creation of a restore checkpoint, useful if you create a checkpoint, then abort the operation with no changes '

container

=

```
self
.
global_undo
.
rewind_savepoint
()
```

if

container

is

not

None

:

```
set_current_container  
(  
    container  
)
```

self

.

```
update_global_history_actions  
()
```

def

create_diff_dialog

(

self

,

revert_msg

=

—

(

'&Revert changes'

),

show_open_in_editor

=

True

):

global

_diff_dialogs

from

```
calibre.gui2.tweak_book.diff.main
```

```
import
```

```
Diff
```

```
def
```

```
line_activated
```

```
(  
    name  
,
```

```
    lnum  
,
```

```
    right  
):
```

```
if
```

```
right  
:
```

```
self
```

```
.  
edit_file_requested  
(  
    name  
,
```

```
None
```

```
,  
  
guess_type  
(  
    name  
))
```

if

name

in

editors

:

editor

=

editors

[

name

]

editor

.

go_to_line

(

lnum

)

editor

.

setFocus

(

Qt

.

FocusReason

.

OtherFocusReason

)

self


```
.
gui
.
raise_
()

d

=

Diff
(
revert_button_msg
=
revert_msg
,

show_open_in_editor
=
show_open_in_editor
)

[
x
.
break_cycles
()

for

x

in

_diff_dialogs

if

not

x
```

```
.
isVisible
()]

_diff_dialogs

=

[
x

for

x

in

_diff_dialogs

if

x
.
isVisible
()]

+

[
d
]

d
.
show
(),

d
.
raise_
(),
```

```
d
.
setFocus
(
Qt
.
FocusReason
.
OtherFocusReason
),

d
.
setWindowModality
(
Qt
.
WindowModality
.
NonModal
)

if

show_open_in_editor
:

d
.
line_activated
.
connect
(
line_activated
)

return

d
```

[\[docs\]](#)

def

show_current_diff

(

self

,

allow_revert

=

True

,

to_container

=

None

):

'''

Show the changes to the book from its last checkpointed state

:param allow_revert: If True the diff dialog will have a button to allow the user to revert all changes

:param to_container: A container object to compare the current container to. If None, the previously checkpointed container is used

'''

self

.

commit_all_editors_to_container

()

```
k
=
{}
if
allow_revert
else
{
'revert_msg'
:
None
}

d
=
self
.
create_diff_dialog
(
**
k
)

previous_container
=
self
.
global_undo
.
previous_container
```

```
connect_lambda
(
d
.
revert_requested
,
self
,
lambda

self
:

self
.
revert_requested
(
previous_container
))

other

=

to_container

or

self
.
global_undo
.
previous_container

d
.
container_diff
```

```
(
other
,

self
.
global_undo
.
current_container
,

names
=
(
self
.
global_undo
.
label_for_container
(
other
),

self
.
global_undo
.
label_for_container
(
self
.
global_undo
.
current_container
)))

def

ask_to_show_current_diff
(
self
```

```
,
name
',
title
',
msg
',
allow_revert
=
True
',
to_container
=
None
):

if

tprefs
.
get
(
'skip_ask_to_show_current_diff_for_'
+
name
):

return

d

=
```



```
QDialog
(
self
.
gui
)

k

=

QVBoxLayout
(
d
)

d
.
setWindowTitle
(
title
)

k
.
addWidget
(
QLabel
(
msg
))

k
.
confirm

=

cb
```

```
=  
  
QCheckBox  
(  
    -  
(  
    'Show this popup again'  
))  
  
k  
.  
addWidget  
(  
    cb  
)  
  
cb  
.  
setChecked  
(  
    True  
)  
  
connect_lambda  
(  
    cb  
    .  
    toggled  
    ,  
  
    d  
    ,  
  
    lambda  
  
    d  
    ,  
  
    checked
```

```
:

tprefs
.
set
(
'skip_ask_to_show_current_diff_for_'

+

name
,

not

checked
))

d
.
bb

=

bb

=

QDialogButtonBox
(
QDialogButtonBox
.
StandardButton
.
Close
,

d
)

k
```

```
.  
addWidget  
(  
bb  
)
```

```
bb  
.  
accepted
```

```
.  
connect  
(  
d  
.  
accept  
)
```

```
bb  
.  
rejected
```

```
.  
connect  
(  
d  
.  
reject  
)
```

```
d  
.  
b
```

```
=
```

```
b
```

```
=
```

```
bb
```

```
.
```

```
addButton
(  
-  
(  
'See what &changed'  
)  
,
```

```
QDialogButtonBox  
.  
ButtonRole  
.  
AcceptRole  
)
```

```
b  
.  
setIcon  
(  
QIcon  
(  
I  
(  
'diff.png'  
))),
```

```
b  
.  
setAutoDefault  
(  
False  
)
```

```
bb  
.  
button  
(  
QDialogButtonBox  
.  
StandardButton  
.  
Close
```

```
)  
.  
setDefault  
(  
    True  
)  
  
if  
  
d  
.  
exec  
(  
  
==  
  
QDialog  
.  
DialogCode  
.  
Accepted  
:  
  
self  
.  
show_current_diff  
(  
    allow_revert  
    =  
    allow_revert  
,  
  
    to_container  
    =  
    to_container  
)  
  
def
```

```
compare_book
(
self
):

if

not

self
.
ensure_book
():

return

self
.
commit_all_editors_to_container
()

c

=

current_container
()

path

=

choose_files
(
self
.
gui
,
```

```
'select-book-for-comparison'
```

```
,
```

```
-  
(
```

```
'Choose book'
```

```
),
```

```
filters
```

```
=
```

```
[
```

```
(
```

```
-  
(
```

```
,
```

```
%s
```

```
books'
```

```
)
```

```
%
```

```
c
```

```
.
```

```
book_type
```

```
.
```

```
upper
```

```
()),
```

```
(
```

```
c
```

```
.
```

```
book_type
```

```
,)),
```

```
select_only_single_file
```

```
=
```

```
True
```

```
,
```

```
all_files
```



```
=
False
)

if
path
and
path
[
0
]:

with
TemporaryDirectory
(
'__compare'
)

as
tdir
:

other

=

_gc
(
path
[
0
],
tdir
=
```

```
tdir
,

tweak_mode
=
True
)

d

=

self
.
create_diff_dialog
(
revert_msg
=
None
)

d
.
container_diff
(
other
,

c
,

names
=
(
_
(
'Other book'
),
_
```

```
(  
'Current book'  
)))
```

```
def
```

```
revert_requested
```

```
(  
self  
,
```

```
container  
):
```

```
self
```

```
.  
commit_all_editors_to_container  
()
```

```
nc
```

```
=
```

```
self
```

```
.  
global_undo
```

```
.  
revert_to  
(  
container  
)
```

```
set_current_container
```

```
(  
nc  
)
```

```
self
.  
apply_container_update_to_gui  
(
```

```
def
```

```
compare_requested  
(  
self  
,  
  
container  
):
```

```
self  
.  
show_current_diff  
(  
to_container  
=  
container  
)
```

```
# }}}  
  

```

```
\[docs\]
```

```
def
```

```
set_modified  
(  
self  
):
```

```
' Mark the book as having been modified '
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
action_save
```

```
.
```

```
setEnabled
```

```
(
```

```
True
```

```
)
```

```
def
```

```
request_completion
```

```
(
```

```
self
```

```
,
```

```
name
```

```
,
```

```
completion_type
```

```
,
```

```
completion_data
```

```
,
```

```
query
```

```
=
```

```
None
```

```
):
```

```
if
```

```
completion_type
```

```
is
```

None

:

completion_worker

()

.

clear_caches

(

completion_data

)

return

request_id

=

(

self

.

completion_request_count

,

name

)

self

.

completion_request_count

+=

1

completion_worker

()

.

```
queue_completion
```

```
(
```

```
request_id
```

```
,
```

```
completion_type
```

```
,
```

```
completion_data
```

```
,
```

```
query
```

```
)
```

```
return
```

```
request_id
```

```
[
```

```
0
```

```
]
```

```
def
```

```
handle_completion_result
```

```
(
```

```
self
```

```
,
```

```
result
```

```
):
```

```
name
```

```
=
```

```
result
```

```
.
```

```
request_id
```

```
[
```

```
1
]

editor

=

editors
.
get
(
name
)

if

editor

is

not

None
:

editor
.
handle_completion_result
(
result
)

def

fix_html
(
self
,
```



```
current
):

if

current
:

ed

=

self
.
gui
.
central
.
current_editor

if

hasattr
(
ed
,

'fix_html'
):

ed
.
fix_html
()

else
:
```

with

BusyCursor

():

self

.

add_savepoint

(

—

(

'Before: Fix HTML'

))

fix_all_html

(

current_container

()))

self

.

update_editors_from_container

()

self

.

set_modified

()

self

.

ask_to_show_current_diff

(

'html-fix'

,

```
-  
(  
    'Fixing done'  
),  
  
-  
(  
    'All HTML files fixed'  
))
```

def

```
pretty_print  
(  
    self  
    ,  
  
    current  
    ):
```

if

```
current  
:
```

```
ed
```

```
=
```

```
self
```

```
.  
gui
```

```
.  
central
```

```
.  
current_editor
```

```
ed
```

```
.
pretty_print
(
editor_name
(
ed
))

else
:

with

BusyCursor
():

self
.
add_savepoint
(
_
(
'Before: Beautify files'
))

pretty_all
(
current_container
())

self
.
update_editors_from_container
()

self
.
```

```
set_modified
()

QApplication
.
alert
(
self
.
gui
)

self
.
ask_to_show_current_diff
(
'beautify'
,

_
(
'Beautified'
),

_
(
'All files beautified'
))

def

mark_selected_text
(
self
):

ed
```

```
=  
  
self  
.  
gui  
.  
central  
.  
current_editor  
  
if  
  
ed  
  
is  
  
not  
  
None  
:  
  
ed  
.  
mark_selected_text  
()  
  
if  
  
ed  
.  
has_marked_text  
:  
  
self  
.  
gui  
.  
central  
.
```

```
search_panel
.  
set_where  
(  
'selected-text'  
)
```

```
else  
:
```

```
self  
.  
gui  
.  
central  
.  
search_panel  
.  
unset_marked  
(
```

```
def
```

```
editor_action  
(  
self  
,  
  
action  
):
```

```
ed
```

```
=
```

```
self  
.  
gui
```

```
.
central
.
current_editor

edname

=

editor_name
(
ed
)

if

hasattr
(
ed
,

'action_triggered'
):

if

action

and

action
[
0
]

==

'insert_resource'
:
```



```
rtype
```

```
=
```

```
action
```

```
[
```

```
1
```

```
]
```

```
if
```

```
rtype
```

```
==
```

```
'image'
```

```
and
```

```
ed
```

```
.
```

```
syntax
```

```
not
```

```
in
```

```
{
```

```
'css'
```

```
,
```

```
'html'
```

```
};
```

```
return
```

```
error_dialog
```

```
(
```

```
self
```

```
.
```

```
gui
,

-
(
'Not supported'
),

-
(

'Inserting images is only supported for HTML and CSS files.'
),

show
=
True
)

rdata

=

get_resource_data
(
rtype
,

self
.
gui
)

if

rdata

is

None
```

```
:  
  
return  
  
if  
rtype  
  
==  
  
'image'  
:  
  
chosen_name  
,  
  
chosen_image_is_external  
,  
  
fullpage  
,  
  
preserve_ar  
  
=  
  
rdata  
  
if  
  
chosen_image_is_external  
:  
  
with  
  
open  
(  
chosen_image_is_external
```

```
[
1
],

'rb'
)

as

f
:

current_container
()
.
add_file
(
chosen_image_is_external
[
0
],

f
.
read
())

self
.
refresh_file_list
()

chosen_name

=

chosen_image_is_external
[
0
]
```

```
href
=
current_container
()
.
name_to_href
(
chosen_name
,
edname
)

fmt
,
width
,
height
=
identify
(
current_container
()
.
raw_data
(
chosen_name
,
decode
=
False
))
```

```
ed
.
insert_image
(
href
,

fullpage
=
fullpage
,

preserve_aspect_ratio
=
preserve_ar
,

width
=
width
,

height
=
height
)
```

elif

```
action
[
0
]

==

'insert_hyperlink'
:

self
```

```
.  
commit_all_editors_to_container  
(  

```

```
d
```

```
=
```

```
InsertLink  
(  
current_container  
(  

```

```
edname  
,
```

```
initial_text  
=  
ed
```

```
.  
get_smart_selection  
(  

```

```
parent  
=  

```

```
self
```

```
.  
gui  
)
```

```
if
```

```
d
```

```
.  
exec  
(  

```

```
==
```

```
QDialog
```

```
.
```

DialogCode

.

Accepted

:

ed

.

insert_hyperlink

(

d

.

href

,

d

.

text

,

template

=

d

.

rendered_template

)

elif

action

[

0

]

==

'insert_tag'

:

d


```
=  
  
InsertTag  
(  
parent  
=  
self  
.  
gui  
)  
  
if  
  
d  
.  
exec  
(  
  
==  
  
QDialog  
.  
DialogCode  
.  
Accepted  
:  
  
ed  
.  
insert_tag  
(  
d  
.  
tag  
)  
  
else  
:
```

```
ed
.
action_triggered
(
action
)

def

rename_class
(
self
,

class_name
):

self
.
commit_all_editors_to_container
()

text
,

ok

=

QInputDialog
.
getText
(
self
.
gui
,
_
```

```
(  
    'New class name'  
)
```

```
—  
(
```

```
    'Rename the class
```

```
    {}  
    to?'  
)
```

```
·  
format  
(  
    class_name  
)
```

```
if
```

```
    ok  
    :
```

```
    self  
    ·  
    add_savepoint  
    (  
    —
```

```
    (  
    'Before: Rename  
    {}  
    '
```

```
    )  
    ·  
    format  
    (  
    class_name  
    ))
```

```
with
```

```
BusyCursor
():

changed

=

rename_class
(
current_container
()),

class_name
,

text
.
strip
())

if

changed
:

self
.
apply_container_update_to_gui
()

self
.
show_current_diff
()

else
:
```

```
self
.
rewind_savepoint
()

return

info_dialog
(
self
.
gui
,

_
(
'No matches'
),

_
(

'No class
{}
found to change'
)
.
format
(
class_name
),

show
=
True
)
```

```
def
set_semantics
(
self
):

if

not

self
.
ensure_book
():

return

self
.
commit_all_editors_to_container
()

c
=
current_container
()

if

c
.
book_type

==
```

```
'azw3'
:

return

error_dialog
(
self
.
gui
,

_
(
'Not supported'
),

_
(

'Semantics are not supported for the AZW3 format.'
),

show
=
True
)

d

=

InsertSemantics
(
c
,

parent
=
self
```

```
.
gui
)

if

d
.
exec
()

==

QDialog
.
DialogCode
.
Accepted

and

d
.
changes
:

self
.
add_savepoint
(
-
(
'Before: Set Semantics'
))

d
.
apply_changes
(
current_container
```



```
()

self
.
apply_container_update_to_gui
()

def

filter_css
(
self
):

self
.
commit_all_editors_to_container
()

c

=

current_container
()

ed

=

self
.
gui
.
central
.
current_editor
```

current_name

=

editor_name

(

ed

)

if

current_name

and

c

.

mime_map

[

current_name

]

not

in

OEB_DOCS

|

OEB_STYLES

:

current_name

=

None

```
d
=
FilterCSS
(
current_name
=
current_name
,
parent
=
self
.
gui
)

if

d
.
exec
()

==

QDialog
.
DialogCode
.
Accepted

and

d
.
filtered_properties
:
```

```
self
.
add_savepoint
(
_
(
'Before: Filter style information'
))
```

with

```
BusyCursor
():
```

```
changed
```

```
=
```

```
filter_css
(
current_container
()),
```

```
d
.
filtered_properties
,
```

```
names
```

```
=
```

```
d
.
filter_names
)
```

if

```
changed
:
```

```
self
.
apply_container_update_to_gui
()

self
.
show_current_diff
()

else
:

self
.
rewind_savepoint
()

return

info_dialog
(
self
.
gui
,

—
(
'No matches'
),

—
(
'No matching style rules were found'
),
```

```
show
```

```
=
```

```
True
```

```
)
```

```
def
```

```
show_find
```

```
(
```

```
self
```

```
):
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
central
```

```
.
```

```
show_find
```

```
()
```

```
ed
```

```
=
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
central
```

```
.
```

```
current_editor
```

```
if
```

```
ed
```

```
is
not
None
and
hasattr
(
ed
,
'selected_text'
):

text

=

ed
.
selected_text

if
text

and
text
.
strip
():

self
.
gui
.
central
```

```
.
pre_fill_search
(
text
)

def

show_text_search
(
self
):

self
.
gui
.
text_search_dock
.
show
()

self
.
gui
.
text_search
.
find
.
setFocus
(
Qt
.
FocusReason
.
OtherFocusReason
)
```



```
def
search_action_triggered
(
self
,
action
,
overrides
=
None
):

ss

=

self
.
gui
.
saved_searches
.
isVisible
()

trigger_saved_search

=

ss

and

(
not
```

```
self
    .
gui
    .
central
    .
search_panel
    .
isVisible
()

or

self
    .
gui
    .
saved_searches
    .
has_focus
())

if

trigger_saved_search
:

return

self
    .
gui
    .
saved_searches
    .
trigger_action
(
action
,
overrides
```

```
=
overrides
)

self
.
search
(
action
,
overrides
)

def
run_saved_searches
(
self
,
searches
,
action
):

ed

=

self
.
gui
.
central
.
current_editor
```

```
name
```

```
=
```

```
editor_name
```

```
(
```

```
ed
```

```
)
```

```
searchable_names
```

```
=
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
file_list
```

```
.
```

```
searchable_names
```

```
if
```

```
not
```

```
searches
```

```
or
```

```
not
```

```
validate_search_request
```

```
(
```

```
name
```

```
,
```

```
searchable_names
```

```
,
```

```
getattr
```

```
(  
    ed  
    ,  
    'has_marked_text'  
    ,  
    False  
    ),
```

```
searches  
[  
    0  
],
```

```
self  
.  
gui  
):
```

```
return
```

```
ret
```

```
=
```

```
run_search  
(  
    searches
```

```
,
```

```
    action
```

```
,
```

```
    ed
```

```
,
```

```
    name
```

```
,
```

```
    searchable_names
```

```
,

self
.
gui
,

self
.
show_editor
,

self
.
edit_file
,

self
.
show_current_diff
,

self
.
add_savepoint
,

self
.
rewind_savepoint
,

self
.
set_modified
)

ed

=
```

```
ret

is

True

and

self
.
gui
.
central
.
current_editor

if

getattr
(
ed
,
'has_line_numbers'
,

False
):

ed
.
editor
.
setFocus
(
Qt
.
FocusReason
.
OtherFocusReason
)
```

```
else
```

```
:
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
saved_searches
```

```
.
```

```
setFocus
```

```
(
```

```
Qt
```

```
.
```

```
FocusReason
```

```
.
```

```
OtherFocusReason
```

```
)
```

```
def
```

```
search
```

```
(
```

```
self
```

```
,
```

```
action
```

```
,
```

```
overrides
```

```
=
```

```
None
```

```
):
```

```
# Run a search/replace
```

```
sp
```



```
=
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
central
```

```
.
```

```
search_panel
```

```
# Ensure the search panel is visible
```

```
sp
```

```
.
```

```
setVisible
```

```
(
```

```
True
```

```
)
```

```
ed
```

```
=
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
central
```

```
.
```

```
current_editor
```

```
name
```

```
=
```

```
editor_name
```

```
(
```

```
ed
```

```
)

state
=
sp
.
state

if
overrides
:

state
.
update
(
overrides
)

searchable_names
=

self
.
gui
.
file_list
.
searchable_names

if
not
```

```
validate_search_request
```

```
(
```

```
name
```

```
,
```

```
searchable_names
```

```
,
```

```
getattr
```

```
(
```

```
ed
```

```
,
```

```
'has_marked_text'
```

```
,
```

```
False
```

```
),
```

```
state
```

```
,
```

```
self
```

```
.
```

```
gui
```

```
):
```

```
return
```

```
ret
```

```
=
```

```
run_search
```

```
(
```

```
state
```

```
,
```

```
action
```

```
,
```

```
ed
,

name
,

searchable_names
,

self
.
gui
,

self
.
show_editor
,

self
.
edit_file
,

self
.
show_current_diff
,

self
.
add_savepoint
,

self
.
rewind_savepoint
,

self
.
```

```
set_modified
)

ed

=

ret

is

True

and

self
.
gui
.
central
.
current_editor

if

getattr
(
ed
,
'has_line_numbers'
,
False
):

ed
.
editor
.
```

```
setFocus
(  
    Qt  
    .  
    FocusReason  
    .  
    OtherFocusReason  
)
```

else

:

self

gui

saved_searches

setFocus

```
(  
    Qt
```

FocusReason

OtherFocusReason

)

def

find_text

```
(  
    self
```

,

state

):

from

`calibre.gui2.tweak_book.text_search`

`import`

`run_text_search`

`searchable_names`

`=`

`self`

`.`

`gui`

`.`

`file_list`

`.`

`searchable_names`

`ed`

`=`

`self`

`.`

`gui`

`.`

`central`

`.`

`current_editor`

`name`

`=`

`editor_name`

`(`

`ed`

`)`

```
if
not
validate_search_request
(
name
,
searchable_names
,
getattr
(
ed
,
'has_marked_text'
,
False
),
state
,
self
.
gui
):

return

ret
=
run_text_search
(
state
```



```
,  
  
ed  
,  
  
name  
,  
  
searchable_names  
,  
  
self  
.  
gui  
,  
  
self  
.  
show_editor  
,  
  
self  
.  
edit_file  
)  
  
ed  
  
=  
  
ret  
  
is  
  
True  
  
and  
  
self  
.  
gui  
.
```

```
central
.
current_editor

if

getattr
(
ed
,
'has_line_numbers'
,
False
):

ed
.
editor
.
setFocus
(
Qt
.
FocusReason
.
OtherFocusReason
)

def
find_initial_text
(
self
,
text
):
```

```
from
```

```
calibre.gui2.tweak_book.search
```

```
import
```

```
get_search_regex
```

```
from
```

```
calibre.gui2.tweak_book.text_search
```

```
import
```

```
file_matches_pattern
```

```
search
```

```
=
```

```
{
```

```
'find'
```

```
:
```

```
text
```

```
,
```

```
'mode'
```

```
:
```

```
'normal'
```

```
,
```

```
'case_sensitive'
```

```
:
```

```
True
```

```
,
```

```
'direction'
:

'down'
}

pat

=

get_search_regex
(
search
)

searchable_names

=

set
(
self
.
gui
.
file_list
.
searchable_names
[
'text'
])

for
name
,
ed
in
```

```
iteritems
(
editors
):

searchable_names
.
discard
(
name
)

if

ed
.
find_text
(
pat
,

complete
=
True
):

self
.
show_editor
(
name
)

return

for
```

```
name

in

searchable_names
:

if

file_matches_pattern
(
name
,

pat
):

self
.
edit_file
(
name
)

if

editors
[
name
]
.
find_text
(
pat
,

complete
=
True
);
```

```
return
```

```
def
```

```
find_word
```

```
(  
self
```

```
,
```

```
word
```

```
,
```

```
locations
```

```
):
```

```
# Go to a word from the spell check dialog
```

```
ed
```

```
=
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
central
```

```
.
```

```
current_editor
```

```
name
```

```
=
```

```
editor_name
```

```
(
```

```
ed
```

```
)
```

```
find_next_word
```

```
(  
word  
,
```

```
locations  
,
```

```
ed  
,
```

```
name  
,
```

```
self
```

```
.  
gui
```

```
,
```

```
self
```

```
.  
show_editor
```

```
,
```

```
self
```

```
.  
edit_file
```

```
)
```

```
def
```

```
next_spell_error
```

```
(  
self  
):
```

```
# Go to the next spelling error
```



```
ed
```

```
=
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
central
```

```
.
```

```
current_editor
```

```
name
```

```
=
```

```
editor_name
```

```
(
```

```
ed
```

```
)
```

```
find_next_error
```

```
(
```

```
ed
```

```
,
```

```
name
```

```
,
```

```
self
```

```
.
```

```
gui
```

```
,
```

```
self
```

```
.
```

```
show_editor
```

```
,
```

```
self
    .
    edit_file
    ,

self
    .
    close_editor
    )

def

word_change_requested
(
    self
    ,

    w
    ,

    new_word
):

if

self
    .
    commit_all_editors_to_container
    ():

self
    .
    gui
    .
    spell_check
    .
    change_word_after_update
    (
    w
```

```
,
new_word
)

else
:

self
.
gui
.
spell_check
.
do_change_word
(
w
,
new_word
)

def

word_replaced
(
self
,
changed_names
):

self
.
set_modified
()
```

```
self
.  
update_editors_from_container  
(  
names  
=  
set  
(  
changed_names  
))
```

```
def
```

```
word_ignored  
(  
self  
,  
  
word  
,  
  
locale  
):
```

```
if
```

```
tprefs  
[  
'inline_spell_check'  
]:
```

```
for
```

```
ed
```

```
in
```

```
itervalues  
(
```

```
editors
):

try
:

ed
.
editor
.
recheck_word
(
word
,
locale
)

except

AttributeError
:

pass

def
editor_link_clicked
(
self
,
url
):

ed
```

```
=  
  
self  
.  
gui  
.  
central  
.  
current_editor
```

```
name
```

```
=  
  
editor_name  
(  
ed  
)
```

```
if
```

```
url  
.  
startswith  
(  
'#'  
):
```

```
target
```

```
=  
  
name
```

```
else  
:
```

```
target

=

current_container
()
.
href_to_name
(
url
,
name
)

frag

=

url
.
partition
(
'#'
)[
-
1
]

if

current_container
()
.
has_name
(
target
):

self
```

```
.  
link_clicked  
(  
    target  
,  
  
    frag  
,  
  
    show_anchor_not_found  
    =  
    True  
)  
  
else  
:  
  
try  
:  
  
    purl  
  
    =  
  
    urlparse  
    (  
        url  
    )  
  
except  
  
    ValueError  
:  
  
return  
  
if
```



```
purl
```

```
.
```

```
scheme
```

```
not
```

```
in
```

```
{
```

```
"
```

```
,
```

```
'file'
```

```
};
```

```
open_url
```

```
(
```

```
QUrl
```

```
(
```

```
url
```

```
))
```

```
else
```

```
:
```

```
error_dialog
```

```
(
```

```
self
```

```
.
```

```
gui
```

```
,
```

```
—
```

```
(
```

```
'Not found'
```

```
),
```

```
—
```

```
(
```

```
'No file with the name  
%s  
was found in the book'  
)
```

```
%
```

```
target
```

```
,
```

```
show
```

```
=
```

```
True
```

```
)
```

```
def
```

```
editor_class_clicked
```

```
(
```

```
self
```

```
,
```

```
class_data
```

```
):
```

```
from
```

```
calibre.gui2.tweak_book.jump_to_class
```

```
import
```

```
find_first_matching_rule
```

```
,
```

```
NoMatchingTagFound
```

```
,
```

```
NoMatchingRuleFound
```

```
ed
```

```
=
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
central
```

```
.
```

```
current_editor
```

```
name
```

```
=
```

```
editor_name
```

```
(
```

```
ed
```

```
)
```

```
try
```

```
:
```

```
res
```

```
=
```

```
find_first_matching_rule
```

```
(
```

```
current_container
```

```
()),
```

```
name
```

```
,
```

```
ed
```

```
.
```

```
get_raw_data
(),

class_data
)

except

(
NoMatchingTagFound
,

NoMatchingRuleFound
):

res

=

None

if

res

is

not

None

and

res

.
file_name

and

res
```

```
.
rule_address
:

editor

=

self

.
open_editor_for_name
(
res
.
file_name
)

if

editor
:

editor

.
goto_css_rule
(
res
.
rule_address
,

sourceline_address
=
res
.
style_tag_address
)

else
```

```
:  
  
error_dialog  
(  
    self  
    .  
    gui  
    ,  
  
    _  
    (  
        'No matches found'  
    ),  
  
    _  
    (  
        'No style rules that match the class  
        {}  
        were found'  
    )  
    .  
    format  
    (  
  
        class_data  
        [  
            'class'  
        ]),  
  
        show  
        =  
        True  
    )  
  
    def  
  
    save_search  
    (  
        self
```

```
):  
  
state  
  
=  
  
self  
.  
gui  
.  
central  
.  
search_panel  
.  
state  
  
self  
.  
show_saved_searches  
(  
  
self  
.  
gui  
.  
saved_searches  
.  
add_predefined_search  
(  
state  
)  
  
def  
  
show_saved_searches  
(  
self  
):
```

```
self
.  
gui  
.  
saved_searches_dock  
.  
show  
(
```

```
saved_searches
```

```
=
```

```
show_saved_searches
```

```
def
```

```
create_checkpoint  
(  
self  
):
```

```
text
```

```
,
```

```
ok
```

```
=
```

```
QInputDialog
```

```
.
```

```
getText
```

```
(
```

```
self
```

```
.
```

```
gui
```

```
,
```



```
-
(
    'Choose name'
),

-
(

    'Choose a name for the checkpoint.
    \n
    You can later restore the book'

    ' to this checkpoint via the
    \n
    "Revert to..." entries in the Edit menu.'
))

if

    ok
    :

    self
    .
    add_savepoint
    (
        text
    )

def

    commit_editor_to_container
    (
        self
    ,
```

```
name
,

container
=
None
):

container

=

container

or

current_container
()

ed

=

editors
[
name
]

with

container
.
open
(
name
,
'wb'
)
```

```
as
```

```
f
```

```
:
```

```
f
```

```
.
```

```
write
```

```
(
```

```
ed
```

```
.
```

```
data
```

```
)
```

```
if
```

```
name
```

```
==
```

```
container
```

```
.
```

```
opf_name
```

```
:
```

```
container
```

```
.
```

```
refresh_mime_map
```

```
()
```

```
lang
```

```
=
```

```
container
```

```
.
```

```
opf_xpath
```

```
(
```

```
'//dc:language/text()'
```

```
)  
  
or  
  
[  
self  
.  
current_metadata  
.  
language  
]  
  
set_book_locale  
(  
lang  
[  
0  
])  
  
if  
  
container  
  
is  
  
current_container  
():  
  
ed  
.  
is_synced_to_container  
=  
True  
  
if  
  
name
```

```
==
```

```
container
```

```
.
```

```
opf_name
```

```
:
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
file_list
```

```
.
```

```
build
```

```
(
```

```
container
```

```
)
```

```
\[docs\]
```

```
def
```

```
commit_all_editors_to_container
```

```
(
```

```
self
```

```
):
```

```
""" Commit any changes that the user has made to files open in editors to
```

```
the container. You should call this method before performing any
```

```
actions on the current container """
```

```
changed
```

```
=
```

False

with

BusyCursor
():

for

name
,

ed

in

iteritems
(
editors
):

if

not

ed

.
is_synced_to_container
:

self

.
commit_editor_to_container
(
name
)

```
ed
.  
is_synced_to_container
```

```
=
```

```
True
```

```
changed
```

```
=
```

```
True
```

```
return
```

```
changed
```

```
\[docs\]
```

```
def
```

```
save_book
```

```
(  
self  
):
```

```
' Save the book. Saving is performed in the background '
```

```
self
```

```
.  
gui
```

```
.  
update_window_title  
()
```

```
c
```

```
=  
  
current_container  
(  
  
    for  
  
    name  
    ,  
  
    ed  
  
    in  
  
    iteritems  
    (  
        editors  
    ):  
  
    if  
  
    ed  
    .  
    is_modified  
  
    or  
  
    not  
  
    ed  
    .  
    is_synced_to_container  
    :  
  
    self  
    .  
    commit_editor_to_container  
    (  
        name
```



```
,
```

```
c  
)
```

```
ed
```

```
.  
is_modified
```

```
=
```

```
False
```

```
path_to_ebook
```

```
=
```

```
os
```

```
.  
path
```

```
.  
abspath
```

```
(  
c
```

```
.  
path_to_ebook  
)
```

```
destdir
```

```
=
```

```
os
```

```
.  
path
```

```
.  
dirname
```

```
(  
path_to_ebook  
)
```

```
if
not
c
.
is_dir

and
not

os
.
path
.
exists
(
destdir
):

info_dialog
(
self
.
gui
,

_
(
'Path does not exist'
),

_
(

'The file you are editing (
%s
) no longer exists. You have to choose a new save location.'
```

```
)

%

path_to_ebook
,

show_copy_button
=
False
,

show
=
True
)

fmt

=

path_to_ebook
.
rpartition
(
'\.
')[
-
1
]
.
lower
()

start_dir

=

find_first_existing_ancestor
(
```

```
path_to_ebook
)

path

=

choose_save_file
(

self
.
gui
,

'choose-new-save-location'
,

_
(
'Choose file location'
),

initial_path
=
os
.
path
.
join
(
start_dir
,

os
.
path
.
basename
(
path_to_ebook
```

```
)),  
  
filters  
=  
[(  
    fmt  
    .  
    upper  
    (),  
  
    (  
        fmt  
        ,))],  
  
all_files  
=  
False  
)  
  
if  
  
path  
  
is  
  
not  
  
None  
:  
  
if  
  
not  
  
path  
.  
lower  
()  
.  
endswith
```

```
(
    '..'

+

fmt
):

path

=

path

+

'..'

+

fmt

path

=

os
.
path
.
abspath
(
    path
)

c
.
path_to_ebook

=
```

```
path
```

```
self
```

```
.
```

```
global_undo
```

```
.
```

```
update_path_to_ebook
```

```
(
```

```
path
```

```
)
```

```
else
```

```
:
```

```
return
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
action_save
```

```
.
```

```
setEnabled
```

```
(
```

```
False
```

```
)
```

```
tdir
```

```
=
```

```
self
```

```
.
```

```
mkdtemp
```

```
(
```

```
prefix
```

```
=
```

```
'save-'
)

container

=

clone_container
(
c
,

tdir
)

self
.
save_manager
.
schedule
(
tdir
,

container
)

def

save_copy
(
self
):

self
.
gui
.
update_window_title
```



```
()

c

=

current_container
()

if

c
.
is_dir
:

return

error_dialog
(
self
.
gui
,
_
(
'Cannot save a copy'
),
_
(
'Saving a copy of a folder based book is not supported'
),
show
=
True
```

```
)

ext

=

c
.
path_to_ebook
.
rpartition
(
'.'
)[
-
1
]

path

=

choose_save_file
(

self
.
gui
,

'tweak_book_save_copy'
,

—
(
'Choose path'
),

initial_filename
```

```
=
self
.
current_metadata
.
title

+

'.'

+

ext
,

filters
=
[(
-
(
'Book (
%s
)'
)
)

%

ext
.
upper
()),

[
ext
.
lower
()]],

all_files
=
False
```

```
)  
  
if  
not  
path  
:  
  
return  
  
if  
..  
..  
not  
in  
os  
.  
path  
.  
basename  
(  
path  
):  
  
path  
  
+=  
  
..  
..  
+  
  
ext  
.  
lower
```

```
()

tdir

=

self
.
mkdtemp
(
prefix
=
'save-copy-'
)

container

=

clone_container
(
c
,

tdir
)

for
name
,

ed

in
iteritems
(
editors
):
```

```
if
    ed
    .
    is_modified

or

not

ed
.
is_synced_to_container
:

self
.
commit_editor_to_container
(
    name
    ,
    container
)

def
do_save
(
    c
    ,
    path
    ,
    tdir
):
```

```
save_container
```

```
(
```

```
c
```

```
,
```

```
path
```

```
)
```

```
shutil
```

```
.
```

```
rmtree
```

```
(
```

```
tdir
```

```
,
```

```
ignore_errors
```

```
=
```

```
True
```

```
)
```

```
return
```

```
path
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
blocking_job
```

```
(
```

```
'save_copy'
```

```
,
```

```
—
```

```
(
```

```
'Saving copy, please wait...'
```

```
),
```

```
self
.  
copy_saved  
,  
  
do_save  
,  
  
container  
,  
  
path  
,  
  
tdir  
)
```

```
def
```

```
copy_saved  
(  
self  
,  
  
job  
):
```

```
if
```

```
job  
.  
traceback
```

```
is
```

```
not
```

```
None
```

```
:
```



```
return

error_dialog
(
    self
    .
    gui
    ,

    _
    (
        'Failed to save copy'
    ),

    _
    (
        'Failed to save copy, click "Show details" for more information.'
    ),

    det_msg
    =
    job
    .
    traceback
    ,

    show
    =
    True
)

msg

=

_
(
    'Copy saved to
    %s
    '
```

```
)

%

job
.
result

info_dialog
(
self
.
gui
,

_
(
'Copy saved'
),

msg
,

show
=
True
)

self
.
gui
.
show_status_message
(
msg
,

5
)
```

```
def
report_save_error
(
self
,
tb
):

if
self
.
doing_terminal_save
:

prints
(
tb
,
file
=
sys
.
stderr
)

self
.
abort_terminal_save
()

self
.
set_modified
()
```

```
error_dialog
(
self
.
gui
,

-
(
'Could not save'
),

-
(
'Saving of the book failed. Click "Show details"'

' for more information. You can try to save a copy'

' to a different location, via File->Save a copy'
),

det_msg
=
tb
,

show
=
True
)

def

go_to_line_number
(
self
```

```
):  
  
ed  
  
=  
  
self  
.  
gui  
.  
central  
.  
current_editor  
  
if  
  
ed  
  
is  
  
None  
  
or  
  
not  
  
ed  
.  
has_line_numbers  
:  
  
return  
  
num  
,  
  
ok  
  
=
```

```
QInputDialog
.
getInt
(
self
.
gui
,

_
(
'Enter line number'
),

(
'Line number:'
),

ed
.
current_line
,

1
,

max
(
100000
,

ed
.
number_of_lines
))

if
ok
:
```

```
ed
```

```
.  
current_line
```

```
=
```

```
num
```

```
def
```

```
split_start_requested
```

```
(  
    self  
):
```

```
    self
```

```
.  
commit_all_editors_to_container  
()
```

```
    self
```

```
.  
gui
```

```
.  
preview
```

```
.  
do_start_split  
()
```

```
@in_thread_job
```

```
def
```

```
split_requested  
(
```

```
self
,
name
,
loc
,
totals
):

self
.
add_savepoint
(
_
(
'Before: Split
%s
'
)

%

self
.
gui
.
elided_text
(
name
))

try
:

bottom_name

=
```



```
split  
(  
current_container  
()),
```

```
name  
,
```

```
loc  
,
```

```
totals  
=  
totals  
)
```

```
except
```

```
AbortError  
:
```

```
self  
.  
rewind_savepoint  
()
```

```
raise
```

```
self  
.  
apply_container_update_to_gui  
()
```

```
self  
.  
edit_file  
(
```

```
bottom_name
```

```
,
```

```
'html'
```

```
)
```

```
def
```

```
multisplit
```

```
(
```

```
self
```

```
):
```

```
ed
```

```
=
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
central
```

```
.
```

```
current_editor
```

```
if
```

```
ed
```

```
.
```

```
syntax
```

```
!=
```

```
'html'
```

```
:
```

```
return
```

```
name
```

```
=
```

```
editor_name
```

```
(
```

```
ed
```

```
)
```

```
if
```

```
name
```

```
is
```

```
None
```

```
:
```

```
return
```

```
d
```

```
=
```

```
MultiSplit
```

```
(
```

```
self
```

```
.
```

```
gui
```

```
)
```

```
if
```

```
d
```

```
.
```

```
exec
```

```
()
```

```
==

QDialog
{
    DialogCode
    Accepted
    :

with

BusyCursor
():

self
{
    add_savepoint
    (
        _
    (
        'Before: Split
        %s
        '
    )
    )

    %

    self
    {
        gui
        {
            elided_text
            (
                name
            )
        )

        try
        :
    }
}
```

```
multisplit
(
current_container
()),

name
,

d
.
xpath
)

except

AbortError
:

self
.
rewind_savepoint
()

raise

self
.
apply_container_update_to_gui
()

def

open_editor_for_name
(
self
,

```

```
name
):

if

name

in

editors
:

editor

=

editors
[
name
]

self
.
gui
.
central
.
show_editor
(
editor
)

else
:

try
:
```

```
mt

=

current_container
()
.
mime_map
[
name
]

except

KeyError
:

error_dialog
(
self
.
gui
,

-
(
'Does not exist'
),

-
(

'The file
%s
does not exist. If you were trying to click an item in'

'the Table of Contents, you may'
```

```
' need to refresh it by right-clicking and choosing "Refresh".'  
)
```

```
%
```

```
name
```

```
,
```

```
show
```

```
=
```

```
True
```

```
)
```

```
return
```

```
None
```

```
syntax
```

```
=
```

```
syntax_from_mime
```

```
(
```

```
name
```

```
,
```

```
mt
```

```
)
```

```
if
```

```
not
```

```
syntax
```

```
:
```

```
error_dialog
```

```
(
```



```
self
.
gui
,

-
(
'Unsupported file format'
),

-
(
'Editing files of type
%s
is not supported'
)

%

mt
,

show
=
True
)

return

None

editor

=

self
.
edit_file
```

```
(
    name
    ,
    syntax
)

return

editor

def
link_clicked
(
    self
    ,
    name
    ,
    anchor
    ,
    show_anchor_not_found
    =
    False
):

    if
    not
    name
    :

    return
```

```
editor

=

self
.
open_editor_for_name
(
name
)

if

anchor

and

hasattr
(
editor
,
'go_to_anchor'
)

:

if

editor
.
go_to_anchor
(
anchor
):

self
.
gui
```

```
.
preview
.
pending_go_to_anchor

=

anchor

elif

show_anchor_not_found
:

error_dialog
(
self
.
gui
,

_
(
'Not found'
),

_
(

'The anchor
%s
was not found in this file'
)

%

anchor
,

show
```

```
=
True
)

@in_thread_job

def
check_item_activated
(
self
,
item
):

is_mult

=

item
.
has_multiple_locations

and

getattr
(
item
,
'current_location_index'
,
None
)

is
```

not

None

name

=

item

.

all_locations

[

item

.

current_location_index

][

0

]

if

is_mult

else

item

.

name

editor

=

None

if

name

in

```
editors
```

```
:
```

```
editor
```

```
=
```

```
editors
```

```
[
```

```
name
```

```
]
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
central
```

```
.
```

```
show_editor
```

```
(
```

```
editor
```

```
)
```

```
else
```

```
:
```

```
try
```

```
:
```

```
editor
```

```
=
```

```
self
```

```
.
```

```
edit_file_requested
```

```
(
```

```
name
,

None
,

current_container
()
.
mime_map
[
name
])

except

KeyError
:

error_dialog
(
self
.
gui
,

—
(
'File deleted'
),

—
(

'The file
{}
has already been deleted, re-run Check Book to update the results.'
)
.
format
```



```
(
    name
),

show
=
True
)

if

getattr
(
    editor
,
    'has_line_numbers'
,
    False
):

if

is_mult
:

    editor
    .
    go_to_line
    (
        *
        (
            item
            .
            all_locations
            [
                item
                .
                current_location_index
```

```
][  
1  
:  
3  
]))
```

```
else
```

```
:
```

```
editor
```

```
.
```

```
go_to_line
```

```
(
```

```
item
```

```
.
```

```
line
```

```
or
```

```
0
```

```
,
```

```
item
```

```
.
```

```
col
```

```
or
```

```
0
```

```
)
```

```
editor
```

```
.
```

```
set_focus
```

```
()
```

```
@in_thread_job
```

```
def
```

```
check_requested
```

```
(
```

```
self
```

```
,
```

```
*
```

```
args
```

```
):
```

```
if
```

```
current_container
```

```
()
```

```
is
```

```
None
```

```
:
```

```
return
```

```
self
```

```
.
```

```
commit_all_editors_to_container
```

```
()
```

```
c
```

```
=
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
check_book
```

```
c
.  
parent  
()  
.  
show  
()
```

```
c  
.  
parent  
()  
.  
raise_  
()
```

```
c  
.  
run_checks  
(  
current_container  
())
```

```
def
```

```
spell_check_requested  
(  
self  
):
```

```
if
```

```
current_container  
()
```

```
is
```

None

:

return

self

.
commit_all_editors_to_container
(

self

.
add_savepoint
(
_
(
'Before: Spell Check'
)
)

self

.
gui
.
spell_check
.
show
(

@in_thread_job

def

fix_requested
(
self
,

```
errors
):

self
.
add_savepoint
(
-
(
'Before: Auto-fix errors'
))

c

=

self
.
gui
.
check_book

c
.
parent
()
.
show
()

c
.
parent
()
.
raise_
()
```

changed

=

c

.

fix_errors

(

current_container

()),

errors

)

if

changed

:

self

.

apply_container_update_to_gui

()

self

.

set_modified

()

else

:

self

.

rewind_savepoint

()

```
@in_thread_job
```

```
def
```

```
merge_requested
```

```
(  
    self
```

```
,
```

```
category
```

```
,
```

```
names
```

```
,
```

```
master
```

```
):
```

```
    self
```

```
    .
```

```
    add_savepoint
```

```
    (
```

```
        _
```

```
    (
```

```
        'Before: Merge files into
```

```
        %s
```

```
        '
```

```
    )
```

```
    %
```

```
    self
```

```
    .
```

```
    gui
```

```
    .
```

```
    elided_text
```

```
    (
```

```
        master
```

```
    ))
```



```
try
```

```
:
```

```
merge
```

```
(
```

```
current_container
```

```
()),
```

```
category
```

```
,
```

```
names
```

```
,
```

```
master
```

```
)
```

```
except
```

```
AbortError
```

```
:
```

```
self
```

```
.
```

```
rewind_savepoint
```

```
()
```

```
raise
```

```
self
```

```
.
```

```
apply_container_update_to_gui
```

```
()
```

```
if
```

```
master
```

```
in
```

```
editors
```

```
:
```

```
self
```

```
.
```

```
show_editor
```

```
(
```

```
master
```

```
)
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
message_popup
```

```
(
```

```
—
```

```
(
```

```
,
```

```
{}
```

```
files merged'
```

```
)
```

```
.
```

```
format
```

```
(
```

```
len
```

```
(
```

```
names
```

```
)))
```

```
@in_thread_job
```

```
def
```

```
link_stylesheets_requested
```

```
(  
self
```

```
,
```

```
names
```

```
,
```

```
sheets
```

```
,
```

```
remove
```

```
):
```

```
self
```

```
.
```

```
add_savepoint
```

```
(
```

```
—
```

```
(
```

```
'Before: Link stylesheets'
```

```
))
```

```
changed_names
```

```
=
```

```
link_stylesheets
```

```
(
```

```
current_container
```

```
()),
```

```
names
```

```
,
```

```
sheets
```

```
,
```

```
remove
```

```
)
```

```
if
changed_names
:

self
.
update_editors_from_container
(
names
=
changed_names
)

self
.
set_modified
()

@in_thread_job

def
export_requested
(
self
,
name_or_names
,
path
):

if
```

```
isinstance
(
name_or_names
,
string_or_bytes
):
```

```
return
```

```
self
.
export_file
(
name_or_names
,
path
)
```

```
for
```

```
name
```

```
in
```

```
name_or_names
:
```

```
dest
```

```
=
```

```
os
```

```
.
path
```

```
.
abspath
(
```

```
os
.  
path  
.  
join  
(  
path  
,  
name  
)
```

```
if
```

```
 '/'
```

```
in
```

```
name
```

```
or
```

```
os
```

```
 .  
sep
```

```
in
```

```
name  
:
```

```
try  
:
```

```
os
```

```
 .  
mkdirs  
(  
os
```

```
 .
```

```
path
.
dirname
(
dest
))

except

OSError

as

err
:

if

err
.
errno

!=

errno
.
EEXIST
:

raise

self

.
export_file
(
name
,
dest
```

```
)

def
open_file_with
(
self
,
file_name
,
fmt
,
entry
):

if
file_name

in
editors

and
not
editors
[
file_name
]
.
is_synced_to_container
:

self
```



```
.  
commit_editor_to_container  
(  
file_name  
)
```

with

```
current_container  
(
```

```
.  
open  
(  
file_name  
)
```

as

```
src  
:
```

```
tdir
```

```
=
```

```
PersistentTemporaryDirectory  
(  
suffix  
=  
'-ee-ow'  
)
```

with

```
open  
(  
os
```

```
.  
path
```

```
.
```

```
join
(
tdir
,

os
.
path
.
basename
(
file_name
)),

'wb'
)

as

dest
:


shutil
.
copyfileobj
(
src
,

dest
)

from

calibre.gui2.open_with

import

run_program
```

```
run_program
(
entry
,

dest
.
name
,

self
)

if

question_dialog
(
self
.
gui
,

_
(
'File opened'
),

_
(

'When you are done editing
{0}
click "Import" to update'

' the file in the book or "Discard" to lose any changes.'
)
.
format
(
file_name
```

```
),

yes_text
=
-
(
'Import'
),

no_text
=
-
(
'Discard'
)

):

self
.
add_savepoint
(
-
(
'Before: Replace
%s
'
)

%

file_name
)

with

open
(
dest
```

```
.
name
,

'rb'
)

as

src
,

current_container
()

.
open
(
file_name
,

'wb'
)

as

cdest
:

shutil

.
copyfileobj
(
src
,

cdest
)

self

.
apply_container_update_to_gui
```

```
()

try
:

shutil
.
rmtree
(
tdir
)

except

Exception
:

pass

@in_thread_job

def

copy_files_to_clipboard
(
self
,

names
):

names

=
```

```
tuple
(
names
)

for

name

in

names
:

if

name

in

editors

and

not

editors
[
name
]
.
is_synced_to_container
:

self
.
commit_editor_to_container
(
name
)
```

```
container
=
current_container
()

md
=
QMimeData
()

url_map
=
{

name
:
container
.
get_file_path_for_processing
(
name
,

allow_modification
=
False
)

for
name
```


in

names

}

md

.

setUrls

(

list

(

map

(

QUrl

.

fromLocalFile

,

list

(

url_map

.

values

()))))

import

json

md

.

setData

(

FILE_COPY_MIME

,

as_bytes

```
(
json
.
dumps
({

name
:

(
url_map
[
name
],

container
.
mime_map
.
get
(
name
))

for

name

in

names

)))

QApplication
.
instance
()
.
clipboard
```

```
()  
.  
setMimeData  
(  
md  
)  
  
@in_thread_job  
  
def  
  
paste_files_from_clipboard  
(  
self  
):  
  
md  
  
=  
  
QApplication  
.  
instance  
(  
.  
clipboard  
(  
.  
mimeData  
(  
  
if  
  
md  
.  
hasUrls  
(
```

and

md

.

hasFormat

(

FILE_COPY_MIME

):

import

json

self

.

commit_all_editors_to_container

()

name_map

=

json

.

loads

(

bytes

(

md

.

data

(

FILE_COPY_MIME

)))

container

=

```
current_container  
(
```

```
for
```

```
name  
,
```

```
(  
path  
,
```

```
mt  
)
```

```
in
```

```
iteritems  
(  
name_map  
):
```

```
with
```

```
lopen  
(  
path  
,
```

```
'rb'  
)
```

```
as
```

```
f  
:
```

```
container  
.  
add_file
```

```
(
    name
    ,

    f
    .
    read
    (
    ),

    media_type
    =
    mt
    ,

    modify_name_if_needed
    =
    True
)

self
.
apply_container_update_to_gui
()

def

export_file
(
    self
    ,

    name
    ,

    path
    ):

if
```

name

in

editors

and

not

editors

[

name

]

.

is_synced_to_container

:

self

.

commit_editor_to_container

(

name

)

with

current_container

()

.

open

(

name

,

'rb'

)

as

src

```
,  
  
open  
(  
    path  
,  
  
    'wb'  
)  
  
as  
  
dest  
:  
  
    shutil  
    .  
    copyfileobj  
    (  
        src  
    ,  
  
        dest  
    )  
  
@in_thread_job  
  
def  
  
replace_requested  
(  
    self  
,  
  
    name  
,  
  
    path  
,
```



```
    basename
    ,
    force_mt
):

    self
    .
    add_savepoint
    (
    _
    (
    'Before: Replace
    %s
    '
    )
    %

    name
    )

    replace_file
    (
    current_container
    ( ),
    name
    ,
    path
    ,
    basename
    ,
    force_mt
    )
```

```
self
.  
apply_container_update_to_gui  
()
```

```
def
```

```
browse_images  
(  
self  
):
```

```
self  
.  
gui  
.  
image_browser  
.  
refresh  
()
```

```
self  
.  
gui  
.  
image_browser  
.  
show  
()
```

```
self  
.  
gui  
.  
image_browser  
.  
raise_  
()
```

def

show_reports

(

self

):

if

not

self

.

ensure_book

(

—

(

'You must first open a book in order to see the report.'

)):

return

self

.

gui

.

reports

.

refresh

()

self

.

gui

.

reports

```
.
show
()

self

.
gui

.
reports

.
raise_
()

def

reports_edit_requested
(
self
,

name
):

mt

=

current_container
()

.
mime_map

.
get
(
name
,

guess_type
(
```

```
name
))

self
.
edit_file_requested
(
name
,
None
,
mt
)

def
image_activated
(
self
,
name
):

mt

=

current_container
()
.
mime_map
.
get
(
name
,
```

```
guess_type
(  
    name  
)
```

```
self
```

```
.  
edit_file_requested
```

```
(  
    name  
    ,
```

```
None
```

```
,
```

```
mt  
)
```

```
def
```

```
check_external_links
```

```
(  
    self  
):
```

```
if
```

```
self
```

```
.
```

```
ensure_book
```

```
(
```

```
—
```

```
(
```

```
'You must first open a book in order to check links.'
```

```
)):
```

```
self
```

```
.
commit_all_editors_to_container
()

self
.
gui
.
check_external_links
.
show
()

def

compress_images
(
self
):

if

not

self
.
ensure_book
(
_
(
'You must first open a book in order to compress images.'
)):

return

from
```

calibre.gui2.tweak_book.polish

import

show_report

,

CompressImages

,

CompressImagesProgress

d

=

CompressImages

(

self

.

gui

)

if

d

.

exec

()

==

QDialog

.

DialogCode

.

Accepted

:

with


```
BusyCursor
():

self
.
add_savepoint
(
-
(
'Before: compress images'
))

d

=

CompressImagesProgress
(
names
=
d
.
names
,
jpeg_quality
=
d
.
jpeg_quality
,
parent
=
self
.
gui
)
```

```
if

d
.
exec
()

!=

QDialog
.
DialogCode
.
Accepted
:

self
.
rewind_savepoint
()

return

changed
,
report
=

d
.
result

if

changed

is
```

None

and

report
:

self

.
rewind_savepoint
()

return

error_dialog

(
self

.
gui
,

—
(
'Unexpected error'
),

—
(

'Failed to compress images, click "Show details" for more information'
),

det_msg

=
report
,

show
=

```
True
)

if
changed
:

self
.
apply_container_update_to_gui
()

else
:

self
.
rewind_savepoint
()

show_report
(
changed
,

self
.
current_metadata
.
title
,

report
,

self
.
```

```
gui
',

self
.
show_current_diff
)

def

sync_editor_to_preview
(
self
,

name
,

sourceline_address
):

editor

=

self
.
edit_file
(
name
,

'html'
)

self
.
ignore_preview_to_editor_sync
```

```
=  
  
True  
  
try  
:  
  
    editor  
    .  
    goto_sourceline  
    (  
    *  
    sourceline_address  
    )  
  
finally  
:  
  
    self  
    .  
    ignore_preview_to_editor_sync  
  
=  
  
False  
  
  
def  
  
do_sync_preview_to_editor  
(  
    self  
    ,  
  
    wait_for_highlight_to_finish  
    =  
    False  
):
```

```
if  
  
self  
.  
ignore_preview_to_editor_sync  
:
```

```
return
```

```
ed  
  
=  
  
self  
.  
gui  
.  
central  
.  
current_editor
```

```
if  
  
ed  
  
is  
  
not  
  
None  
:
```

```
name  
  
=  
  
editor_name
```

```
(
ed
)

if
name
is
not
None
and
getattr
(
ed
,
'syntax'
,
None
)
==
'html'
:

hl
=
getattr
(
ed
,
```



```
'highlighter'
,

None
)

if

wait_for_highlight_to_finish
:

if

getattr
(
hl
,
'is_working'
,

False
):

QTimer
.
singleShot
(
75
,

self
.
sync_preview_to_editor_on_highlight_finish
)

return
```

```
ct

=

ed
.
current_tag
()

self
.
gui
.
preview
.
sync_to_editor
(
name
,

ct
)

if

hl

is

not

None

and

hl
.
is_working
:
```

```
QTimer
.  
singleShot  
(  
75  
,  
  
self  
.  
sync_preview_to_editor_on_highlight_finish  
)
```

[\[docs\]](#)

def

```
sync_preview_to_editor  
(  
self  
):
```

' Sync the position of the preview panel to the current cursor position in the current editor '

```
self  
.  
do_sync_preview_to_editor  
(  

```

def

```
sync_preview_to_editor_on_highlight_finish  
(  
self  
):
```

```
self
```

```
.  
do_sync_preview_to_editor  
(  
    wait_for_highlight_to_finish  
    =  
    True  
)
```

```
def
```

```
show_partial_cfi_in_editor  
(  
    self  
,  
    name  
,  
    cfi  
):
```

```
    editor
```

```
    =
```

```
    self  
    .  
    edit_file  
    (  
        name  
        ,  
        'html'  
    )
```

```
if
```

```
not
```

```
editor
```

```
or
```

```
not
```

```
editor
```

```
.
```

```
has_line_numbers
```

```
:
```

```
return
```

```
False
```

```
from
```

```
calibre.ebooks.oeb.polish.parsing
```

```
import
```

```
parse
```

```
from
```

```
calibre.ebooks.epub.cfi.parse
```

```
import
```

```
decode_cfi
```

```
root
```

```
=
```

```
parse
```

```
(
```

```
editor
.  
get_raw_data  
(  
  
decoder  
=  
lambda  
  
x  
:  
  
x  
.  
decode  
(  
'utf-8'  
)  
  
line_numbers  
=  
True  
,  
  
linenumber_attribute  
=  
'data-lnum'  
)  
  
node  
  
=  
  
decode_cfi  
(  
root  
,  
  
cfi  
)
```

def

barename

(
x
):

return

x

.

tag

.

partition

(

'

)

[

-

1

]

if

node

is

not

None

:

Inum

=

node

```
.
get
(
'data-Inum'
)

if

Inum
:

tags_before

=

[]

for

tag

in

root
.
xpath
(
'//*[@data-Inum="
%s
"]'

%

Inum
):

tags_before
.
append
```



```
(  
  barename  
  (  
    tag  
  ))  
)
```

if

tag

is

node
:

break

else

:

tags_before

.

append

```
(  
  barename  
  (  
    node  
  ))  
)
```

Inum

=

int

```
(  
  Inum  
)
```

return

editor

.

goto_sourceline

(

lnum

,

tags_before

,

attribute

=

'id'

if

node

.

get

(

'id'

)

else

None

)

return

False

def

goto_style_declaration

(

self

```
,  
  
data  
):  
  
name  
  
=  
  
data  
[  
    'name'  
]  
  
editor  
  
=  
  
self  
.  
edit_file  
(  
    name  
    ,  
  
    syntax  
    =  
    data  
    [  
        'syntax'  
    ]  
)  
  
self  
.  
gui  
.  
live_css  
.  
navigate_to_declaration  
(
```

```
data
,

editor
)

def

init_editor
(
self
,

name
,

editor
,

data
=
None
,

use_template
=
False
):

editor
.
undo_redo_state_changed
.
connect
(
self
.
editor_undo_redo_state_changed
)
```

editor

.

data_changed

.

connect

(

self

.

editor_data_changed

)

editor

.

copy_available_state_changed

.

connect

(

self

.

editor_copy_available_state_changed

)

editor

.

cursor_position_changed

.

connect

(

self

.

sync_preview_to_editor

)

editor

.

cursor_position_changed

.

connect

(

```
self
.  
update_cursor_position  
)
```

```
if
```

```
hasattr  
(  
editor  
,  
  
'word_ignored'  
):
```

```
editor  
.  
word_ignored  
.  
connect  
(  
self  
.  
word_ignored  
)
```

```
if
```

```
hasattr  
(  
editor  
,  
  
'link_clicked'  
):
```

```
editor  
.  
link_clicked
```

```
.
connect
(
self
.
editor_link_clicked
)

if

hasattr
(
editor
,

'class_clicked'
):

editor
.
class_clicked
.
connect
(
self
.
editor_class_clicked
)

if

hasattr
(
editor
,

'rename_class'
):
```

```
editor
.  
rename_class  
.  
connect  
(  
self  
.  
rename_class  
)  
  
if  
  
getattr  
(  
editor  
,  
  
'syntax'  
,  
  
None  
)  
  
==  
  
'html'  
:  
  
editor  
.  
smart_highlighting_updated  
.  
connect  
(  
self  
.  
gui  
.  
live_css  
.  

```



```
sync_to_editor
)

if

hasattr
(
editor
,

'set_request_completion'
):

editor
.
set_request_completion
(
partial
(
self
.
request_completion
,

name
),
name
)

if

data

is

not

None
:
```

if

use_template

:

editor

.

init_from_template

(

data

)

else

:

editor

.

data

=

data

editor

.

is_synced_to_container

=

True

editor

.

modification_state_changed

.

connect

```
(  
self  
.  
editor_modification_state_changed  
)
```

```
self  
.  
gui  
.  
central  
.  
add_editor  
(  
name  
,  
editor  
)
```

[\[docs\]](#)

```
def
```

```
edit_file  
(  
self  
,  
name  
,  
syntax  
=  
None  
,  
use_template  
=
```

None

):

""" Open the file specified by name in an editor

:param syntax: The media type of the file, for example, ``text/html``. If not specified it is guessed from the file extension.

:param use_template: A template to initialize the opened editor with

"""

editor

=

editors

.

get

(

name

,

None

)

if

editor

is

None

:

syntax

=

syntax

or

syntax_from_mime

(

name

,

guess_type

(

name

))

if

use_template

is

None

:

data

=

current_container

()

.

raw_data

(

name

)

if

isbytestring

(

```
data  
)
```

and

syntax

in

```
{  
'html'  
,  
  
'css'  
,  
  
'text'  
,  
  
'xml'  
,  
  
'javascript'  
};
```

try

:

data

=

```
data  
.  
decode  
(  
'utf-8'  
)
```

except

```
UnicodeDecodeError
:

return

error_dialog
(
self
.
gui
,

_
(
'Cannot decode'
),

_
(

'Cannot edit
%s
as it appears to be in an unknown character encoding'
)

%

name
,

show
=
True
)

else
:
```

```
data

=

use_template

editor

=

editors
[
name
]

=

editor_from_syntax
(
syntax
,

self
.
gui
.
editor_tabs
)

self
.
init_editor
(
name
,

editor
,

data
,
```



```
use_template
=
bool
(
use_template
))

if

tprefs
[
'pretty_print_on_open'
]:

editor
.
pretty_print
(
name
)

self
.
show_editor
(
name
)

return

editor

\[docs\]

def

show_editor
```

```
(
self
,

name
):

' Show the editor that is editing the file specified by ``name`` '

self
.
gui
.
central
.
show_editor
(
editors
[
name
])

editors
[
name
]
.
set_focus
()

def
edit_file_requested
(
self
,

name
,
```

```
syntax
=
None
,

mime
=
None
):

if

name

in

editors
:

self
.
gui
.
central
.
show_editor
(
editors
[
name
])

return

editors
[
name
]
```

mime

=

mime

or

current_container

()

.

mime_map

.

get

(

name

,

guess_type

(

name

))

syntax

=

syntax

or

syntax_from_mime

(

name

,

mime

)

if

not

syntax

:

return

error_dialog

(

self

.

gui

,

—
(

'Unsupported file format'

),

—
(

'Editing files of type

%s

is not supported'

)

%

mime

,

show

=

True

)

return

```
self
.  
edit_file  
(  
name  
,  
  
syntax  
)  
  
  
def  
  
edit_next_file  
(  
self  
,  
  
backwards  
=  
False  
):  
  
self  
.  
gui  
.  
file_list  
.  
edit_next_file  
(  
self  
.  
currently_editing  
,  
  
backwards  
)
```

```
def
quick_open
(
self
):

if
not
self
.ensure_book
(
-
(
'No book is currently open. You must first open a book to edit.'
)):

return

c
=
current_container
()

files
=
[
name
for
```

```
name
,
mime
in

iteritems
(
c
.
mime_map
)

if

c
.
exists
(
name
)

and

syntax_from_mime
(
name
,
mime
)

is

not

None
]

d
```



```
=

QuickOpen
(
files
,

parent
=
self
.
gui
)

if

d
.
exec
()

==

QDialog
.
DialogCode
.
Accepted

and

d
.
selected_result

is

not

None
:
```

```
self
.  
edit_file_requested  
(  
d  
.  
selected_result  
,
```

```
None
```

```
,  
  
c  
.  
mime_map  
[  
d  
.  
selected_result  
)
```

```
# Editor basic controls {{{
```

```
def
```

```
do_editor_undo  
(  
self  
):
```

```
ed
```

```
=
```

```
self
```

```
.  
gui
```

```
.
```

```
central
.  
current_editor
```

```
if
```

```
ed
```

```
is
```

```
not
```

```
None
```

```
:
```

```
ed
```

```
.
```

```
undo
```

```
()
```

```
def
```

```
do_editor_redo
```

```
(
```

```
self
```

```
):
```

```
ed
```

```
=
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
central
```

```
.
```

```
current_editor
```

```
if
```

```
ed
```

```
is
```

```
not
```

```
None
```

```
:
```

```
ed
```

```
.
```

```
redo
```

```
()
```

```
def
```

```
do_editor_copy
```

```
(
```

```
self
```

```
):
```

```
ed
```

```
=
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
central
```

```
.
```

```
current_editor
```

```
if
```

ed

is

not

None

:

ed

.

copy

()

def

do_editor_cut

(

self

):

ed

=

self

.

gui

.

central

.

current_editor

if

ed

is

not

None

:

ed

.

cut

()

def

do_editor_paste

(

self

):

ed

=

self

.

gui

.

central

.

current_editor

if

ed

is

not

None

:

ed

.

paste

()

def

editor_data_changed

(

self

,

editor

):

self

.

gui

.

preview

.

start_refresh_timer

()

for

name

,

ed

in

iteritems

```
(
    editors
):

    if
        ed
        is
        editor
        :

        self
        .
        gui
        .
        toc_view
        .
        start_refresh_timer
        (
            name
        )

        break

    def
        editor_undo_redo_state_changed
        (
            self
            ,

            *
            args
        ):
```



```
self
.
apply_current_editor_state
()

def

editor_copy_available_state_changed
(
self
,

*
args
):

self
.
apply_current_editor_state
()

def

editor_modification_state_changed
(
self
,

is_modified
):

self
.
apply_current_editor_state
()
```

```
if
    is_modified
    :

    self
    .
    set_modified
    ()

    # }}}

def
    apply_current_editor_state
    (
    self
    ):

    ed

    =

    self
    .
    gui
    .
    central
    .
    current_editor

    self
    .
    gui
    .
    cursor_position_widget
    .
```

```
update_position  
(
```

```
if
```

```
ed
```

```
is
```

```
not
```

```
None
```

```
:
```

```
actions
```

```
[  
  'editor-undo'  
]
```

```
.
```

```
setEnabled
```

```
(  
ed
```

```
.
```

```
undo_available
```

```
)
```

```
actions
```

```
[  
  'editor-redo'  
]
```

```
.
```

```
setEnabled
```

```
(  
ed
```

```
.
```

```
redo_available
```

```
)
```

```
actions
```

```
[
  'editor-copy'
]
.
setEnabled
(
  ed
.
  copy_available
)

actions
[
  'editor-cut'
]
.
setEnabled
(
  ed
.
  cut_available
)

actions
[
  'go-to-line-number'
]
.
setEnabled
(
  ed
.
  has_line_numbers
)

actions
[
  'fix-html-current'
]
.
```

```
setEnabled
```

```
(
```

```
ed
```

```
.
```

```
syntax
```

```
==
```

```
'html'
```

```
)
```

```
name
```

```
=
```

```
editor_name
```

```
(
```

```
ed
```

```
)
```

```
mime
```

```
=
```

```
current_container
```

```
()
```

```
.
```

```
mime_map
```

```
.
```

```
get
```

```
(
```

```
name
```

```
)
```

```
if
```

```
name
```

```
is
```

not

None

and

(
getattr

(
ed

,

'syntax'

,

None

)

==

'html'

or

mime

==

'image/svg+xml'

):

if

self

.

gui

.

preview

.

show

(

name

```
);  
  
# The file being displayed by the preview has changed.  
  
# Set the preview's position to the current cursor  
  
# position in the editor, in case the editors' cursor  
  
# position has not changed, since the last time it was  
  
# focused. This is not inefficient since multiple requests  
  
# to sync are de-bounced with a 100 msec wait.  
  
self  
.  
sync_preview_to_editor  
(  
  
if  
  
name  
  
is  
  
not  
  
None  
:  
  
self  
.  
gui  
.
```

```
file_list
*
mark_name_as_current
(
name
)

if

ed
*
has_line_numbers
:

self
*
gui
*
cursor_position_widget
*
update_position
(
*
ed
*
cursor_position
)

else
:

actions
[
'go-to-line-number'
]
*
setEnabled
(
False
```



```
)

self
.
gui
.
file_list
.
clear_currently_edited_name
()

def

update_cursor_position
(
self
):

ed

=

self
.
gui
.
central
.
current_editor

if

getattr
(
ed
,
'has_line_numbers'
```

```

',

False
):

    self
    *
    gui
    *
    cursor_position_widget
    *
    update_position
    (
    *
    ed
    *
    cursor_position
    )

else
:

    self
    *
    gui
    *
    cursor_position_widget
    *
    update_position
    ()

def

editor_close_requested
(
    self
    ,

```

```
editor
):

name

=

editor_name
(
editor
)

if

not

name
:

return

if

not

editor
.
is_synced_to_container
:

self
.
commit_editor_to_container
(
name
)
```

```
self
.  
close_editor  
(  
name  
)
```

[\[docs\]](#)

def

```
close_editor  
(  
self  
,  
name  
):
```

```
' Close the editor that is editing the file specified by ``name`` '
```

```
editor
```

```
=
```

```
editors
```

```
.  
pop  
(  
name  
)
```

```
self
```

```
.  
gui
```

```
.  
central
```

```
.
close_editor
(
editor
)

editor
.
break_cycles
()

if

not

editors

or

getattr
(
self
.
gui
.
central
.
current_editor
,
'syntax'
,
None
)

!=

'html'
:
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
preview
```

```
.
```

```
clear
```

```
()
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
live_css
```

```
.
```

```
clear
```

```
()
```

```
def
```

```
insert_character
```

```
(
```

```
self
```

```
):
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
insert_char
```

```
.
```

```
show
```

```
()
```

```
def
```

```
manage_snippets
```

```
(
self
):

from

calibre.gui2.tweak_book.editor.snippets

import

UserSnippets

UserSnippets
(
self
.
gui
)
.
exec
()

# Shutdown {{{

def

quit
(
self
):

if

self
.
doing_terminal_save
```

```

:

return

False

if

self
.
save_manager
.
has_tasks
:

if

question_dialog
(

self
.
gui
,

_
(
'Are you sure?'
),

_
(

'The current book is being saved in the background. Quitting now will'

' <b>abort the save process</b>! Finish saving first?'
),

```



```
yes_text
=
-
(
'Finish &saving first'
),

no_text
=
-
(
'&Quit immediately'
)):

if

self
.
save_manager
.
has_tasks
:

self
.
start_terminal_save_indicator
()

return

False

if

not
```

```
self
.  
confirm_quit  
():
```

```
return
```

```
False
```

```
self  
.  
shutdown  
()
```

```
QApplication  
.  
instance  
()  
.  
quit  
()
```

```
return
```

```
True
```

```
def
```

```
confirm_quit  
(  
self  
):
```

```
if
```

```
self
```

```
.
gui
.
action_save
.
isEnabled
():

d

=

QDialog
(
self
.
gui
)

d
.
l

=

QGridLayout
(
d
)

d
.
setLayout
(
d
.
l
)
```

```
d
.  
setTitle  
(  
    _  
(  
    'Unsaved changes'  
))
```

```
d  
.  
i  
  
=  
  
QLabel  
(  
    "  
)
```

```
d  
.  
i  
.  
setMaximumSize  
(  
    QSize  
(  
        64  
        ,  
        64  
    ))
```

```
d  
.  
i  
.  
setPixmap  
(
```

```
QIcon  
(  
I  
(  
'dialog_warning.png'  
))
```

```
.  
pixmap  
(  
d  
.   
i  
.   
maximumSize  
())
```

```
d  
.   
I  
.   
addWidget  
(  
d  
.   
i  
,  
0  
,  
0  
)
```

```
d  
.   
m  
=
```

```
QLabel  
(
```

```
-  
(  
'There are unsaved changes, if you quit without saving, you will lose them.'  
)
```

```
d  
.  
m  
.  
setWordWrap  
(  
True  
)
```

```
d  
.  
l  
.  
addWidget  
(  
d  
.  
m  
,  
0  
,  
1  
)
```

```
d  
.  
bb
```

```
=
```

```
QDialogButtonBox  
(  
QDialogButtonBox
```

```
.  
StandardButton
```

```
.  
Cancel  
)
```

```
d
```

```
.  
bb
```

```
.  
rejected
```

```
.  
connect  
(  
d
```

```
.  
reject  
)
```

```
d
```

```
.  
bb
```

```
.  
accepted
```

```
.  
connect  
(  
d
```

```
.  
accept  
)
```

```
d
```

```
.  
l
```

```
.  
addWidget  
(  
d
```

```
.
```

```
bb
```

```
,
```

```
1
```

```
,
```

```
0
```

```
,
```

```
1
```

```
,
```

```
2
```

```
)
```

```
d
```

```
.
```

```
do_save
```

```
=
```

```
None
```

```
def
```

```
endit
```

```
(
```

```
d
```

```
,
```

```
x
```

```
):
```

```
d
```

```
.
```

```
do_save
```

```
=
```



```
x

d
.
accept
()

b

=

d
.
bb
.
addButton
(
-
(
'&Save and Quit'
),

QDialogButtonBox
.
ButtonRole
.
ActionRole
)

b
.
setIcon
(
QIcon
(
I
(
'save.png'
)))
```

```
connect_lambda
(
b
.
clicked
,

d
,

lambda

d
:

endit
(
d
,

True
))

b

=

d
.
bb
.
addButton
(
-
(
'&Quit without saving'
),

QDialogButtonBox
.
ButtonRole
```

```
.
ActionRole
)

connect_lambda
(
b
.
clicked
,

d
,

lambda

d
:

endit
(
d
,

False
))

d
.
resize
(
d
.
sizeHint
())

if

d
.
```

```
exec
()

!=

QDialog
·
DialogCode
·
Accepted

or

d
·
do_save

is

None
:

return

False

if

d
·
do_save
:

self
·
gui
·
action_save
·
trigger
```

```
()  
  
self  
.  
start_terminal_save_indicator  
()
```

```
return
```

```
False
```

```
return
```

```
True
```

```
def
```

```
start_terminal_save_indicator  
(  
self  
):
```

```
self  
.  
save_state  
()
```

```
self  
.  
gui  
.  
blocking_job  
.  
set_msg  
(
```

```
-  
(  
'Saving, please wait...'  
)
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
blocking_job
```

```
.
```

```
start
```

```
()
```

```
self
```

```
.
```

```
doing_terminal_save
```

```
=
```

```
True
```

```
def
```

```
abort_terminal_save
```

```
(
```

```
self
```

```
):
```

```
self
```

```
.
```

```
doing_terminal_save
```

```
=
```

```
False
```

```
self
    .
    gui
    .
    blocking_job
    .
    stop
    ()

def

check_terminal_save
(
    self
):

if

self
    .
    doing_terminal_save

and

not

self
    .
    save_manager
    .
    has_tasks
:

# terminal save could have been aborted

self
    .
    shutdown
    ()
```

QApplication

```
.  
instance  
(  

```

```
.  
quit  
(  

```

```
def
```

```
shutdown  
(  
self  
):
```

```
self  
.save_state  
(  

```

```
completion_worker  
(  

```

```
.shutdown  
(  

```

```
self  
.save_manager  
.check_for_completion  
.disconnect  
(  

```



```
self
.  
gui  
.  
preview  
.  
stop_refresh_timer  
(
```

```
self  
.  
gui  
.  
live_css  
.  
stop_update_timer  
(
```

```
[  
x  
.  
reject  
(
```

```
for
```

```
x
```

```
in
```

```
_diff_dialogs  
]
```

```
del
```

```
_diff_dialogs  
[:]
```

```
self
```

```
.
save_manager

.
shutdown
()

parse_worker

.
shutdown
()

self

.
save_manager

.
wait
(
0.1
)

def

save_state
(
self
):

with

self

.
editor_cache
:

self

.
save_book_edit_state
```

```
()
```

```
with
```

```
tprefs
```

```
:
```

```
self
```

```
.
```

```
gui
```

```
.
```

```
save_state
```

```
()
```

```
def
```

```
save_book_edit_state
```

```
(
```

```
self
```

```
):
```

```
c
```

```
=
```

```
current_container
```

```
()
```

```
if
```

```
c
```

```
and
```

```
c
```

```
.
```

```
path_to_ebook
```

```
:

tprefs

=

self
.
editor_cache

mem

=

tprefs
[
'edit_book_state'
]

order

=

tprefs
[
'edit_book_state_order'
]

extra

=

len
(
order
)

-
```

99

if

extra

>

0

:

order

=

[

k

for

k

in

order

[

extra

:]

if

k

in

mem

]

mem

```
=  
  
{  
  k  
  :  
  mem  
  [  
    k  
  ]  
  
  for  
  
    k  
  
    in  
  
      order  
    }  
  
  mem  
  [  
    c  
    .  
    path_to_ebook  
  ]  
  
=  
  
{  
  
  'editors'  
  :{  
    name  
    :  
    ed  
    .  
    current_editing_state  
  
    for  
  
      name
```

```
,

ed

in

iteritems
(
editors
}),

'currently_editing'
:
self
.
currently_editing
,

'tab_order'
:
self
.
gui
.
central
.
tab_order
,

}

try
:

order
.
remove
(
```

```
c
.
path_to_ebook
)

except

ValueError
:

pass

order
.
append
(
c
.
path_to_ebook
)

tprefs
[
'edit_book_state'
]

=

mem

tprefs
[
'edit_book_state_order'
]

=

order
```


def

restore_book_edit_state

(

self

):

c

=

current_container

()

if

c

and

c

.

path_to_ebook

:

tprefs

=

self

.

editor_cache

state

=

```
tprefs  
[  
    'edit_book_state'  
]
```

```
.  
get  
(  
    c  
    .  
    path_to_ebook  
)
```

```
if
```

```
state
```

```
is
```

```
not
```

```
None
```

```
:
```

```
opened
```

```
=
```

```
set  
(
```

```
eds
```

```
=
```

```
state
```

```
.
```

```
get  
(  
    'editors'
```

```
,  
  
{})  
  
for  
name  
in  
state  
.  
get  
(  
    'tab_order'  
,  
  
    ()):  
  
    if  
  
        c  
        .  
        has_name  
        (  
            name  
        ):  
  
        try  
        :  
  
        editor  
  
        =  
  
        self  
        .  
        edit_file_requested  
        (  

```

```
name
)
```

```
if
```

```
editor
```

```
is
```

```
not
```

```
None
```

```
:
```

```
opened
```

```
.
```

```
add
```

```
(
```

```
name
```

```
)
```

```
es
```

```
=
```

```
eds
```

```
.
```

```
get
```

```
(
```

```
name
```

```
)
```

```
if
```

```
es
```

```
is
```

```
not
```

None

:

editor

.

current_editing_state

=

es

except

Exception

:

import

traceback

traceback

.

print_exc

()

ce

=

state

.

get

(

'currently_editing'

)

```
if
ce
in
opened
:

self
.
show_editor
(
ce
)

# }}}
```

Source code for calibre.gui2.tweak_book.plugin

```
#!/usr/bin/env python
```

```
__license__
```

```
=
```

```
'GPL v3'
```

```
__copyright__
```

```
=
```

```
'2014, Kovid Goyal <kovid at kovidgoyal.net>'
```

```
import
```

```
importlib
```

```
import
```

```
sys
```

```
from
```

```
qt.core
```

```
import
```

```
QToolButton
```

```
from
calibre

import
prints

from
calibre.customize

import
PluginInstallationType

from
calibre.customize.ui

import
all_edit_book_tool_plugins

from
calibre.gui2.tweak_book

import
current_container
,
tprefs

from
calibre.gui2.tweak_book.boss

import
get_boss
```



```
from
```

```
polyglot.builtins
```

```
import
```

```
intervalues
```

```
\[docs\]
```

```
class
```

```
Tool
```

```
:
```

```
'''
```

The base class for individual tools in an Edit Book plugin. Useful members include:

** ``self.plugin``: A reference to the :class:`calibre.customize.Plugin` object to which this tool belongs.*

** self. :attr:`boss`*

** self. :attr:`gui`*

Methods that must be overridden in sub classes:

** :meth:`create\_action`*

** :meth:`register\_shortcut`*

```
'''
```

#: Set this to a unique name it will be used as a key

name

=

None

#: If True the user can choose to place this tool in the plugins toolbar

allowed_in_toolbar

=

True

#: If True the user can choose to place this tool in the plugins menu

allowed_in_menu

=

True

#: The popup mode for the menu (if any) of the toolbar button. Possible values are 'delayed', 'instant', 'button'

toolbar_button_popup_mode

=

```
'delayed'
```

```
@property
```

```
def
```

```
boss
```

```
(
```

```
self
```

```
):
```

```
' The :class:`calibre.gui2.tweak_book.boss.Boss` object. Used to control the user interface. '
```

```
return
```

```
get_boss
```

```
()
```

```
@property
```

```
def
```

```
gui
```

```
(
```

```
self
```

```
):
```

```
' The main window of the user interface '
```

```
return
```

```
self
```

```
.  
boss
```

```
.  
gui
```

```
@property
```

```
def
```

```
current_container  
(  
    self  
):
```

```
' Return the current :class:`calibre.ebooks.oeb.polish.container.Container` object that  
represents the book being edited. '
```

```
return
```

```
current_container  
()
```

```
\[docs\]
```

```
def
```

```
register_shortcut  
(  
    self  
,
```

```
qaction  
,
```

```
unique_name
```

```

',

default_keys
=
(),

short_text
=
None
',

description
=
None
',

**
extra_data
):

'''

```

Register a keyboard shortcut that will trigger the specified ``qaction``. This keyboard shortcut

will become automatically customizable by the user in the Keyboard shortcuts section of the editor preferences.

*:param qaction: A QAction object, it will be triggered when the
configured key combination is pressed by the user.*

*:param unique_name: A unique name for this shortcut/action. It will be
used internally, it must not be shared by any other actions in this
plugin.*

*:param default_keys: A list of the default keyboard shortcuts. If not
specified no default shortcuts will be set. If the shortcuts specified*

here conflict with either builtin shortcuts or shortcuts from user configuration/other plugins, they will be ignored. In that case, users will have to configure the shortcuts manually via Preferences. For example:

```
``default_keys=('Ctrl+J', 'F9')``.
```

:param short_text: An optional short description of this action. If not specified the text from the QAction will be used.

:param description: An optional longer description of this action, it will be used in the preferences entry for this shortcut.

```
'''
```

```
short_text
```

```
=
```

```
short_text
```

```
or
```

```
str
```

```
(
```

```
qaction
```

```
.
```

```
text
```

```
()))
```

```
.
```

```
replace
```

```
(
```

```
'&&'
```

```
,
```

```
'
```

```
\0
```

```
'
```

```
)  
.  
replace  
(  
'&  
,  
  
"  
)  
.  
replace  
(  
'  
\  
'  
,  
  
'&  
)  
  
self  
.  
gui  
.  
keyboard  
.  
register_shortcut  
(  
  
self  
.  
name  
  
+  
  
' '  
_  
  
+  
  
unique_name  
,
```

```
short_text
```

```
,
```

```
default_keys
```

```
=
```

```
default_keys
```

```
,
```

```
action
```

```
=
```

```
qaction
```

```
,
```

```
description
```

```
=
```

```
description
```

```
or
```

```
"
```

```
,
```

```
group
```

```
=
```

```
—
```

```
(
```

```
'Plugins'
```

```
))
```

```
\[docs\]
```

```
def
```

```
create_action
```

```
(
```

```
self
```

```
,
```

```
for_toolbar
```

```
=
```


True

):

'''

Create a QAction that will be added to either the plugins toolbar or the plugins menu depending on ``for\_toolbar``. For example::

```
def create_action(self, for_toolbar=True):

    ac = QAction(get_icons('myicon.png'), 'Do something')

    if for_toolbar:

        # We want the toolbar button to have a popup menu

        menu = QMenu()

        ac.setMenu(menu)

        menu.addAction('Do something else')

        subaction = menu.addAction('And another')

        # Register a keyboard shortcut for this toolbar action be

        # careful to do this for only one of the toolbar action or

        # the menu action, not both.

        self.register_shortcut(ac, 'some-unique-name', default_keys=('Ctrl+K'))

    return ac

.. seealso:: Method :meth:`register_shortcut`.
```

'''

raise

NotImplementedError
(

def

load_plugin_tools
(
plugin
):

try
:

main

=

importlib
.
import_module
(
plugin
.
__class__
.
__module__
+
'main'
)

except

ImportError
:

import

traceback

traceback

.

print_exc

()

else

:

for

x

in

itervalues

(

vars

(

main

)):

if

isinstance

(

x

,

type

)

and

x

is

not

Tool

and

issubclass

(

x

,

Tool

):

ans

=

x

()

ans

.

plugin

=

plugin

yield

ans

def

plugin_action_sid

```
(
    plugin
    ,

    tool
    ,

    for_toolbar
    =
    True
):
```

```
return
```

```
plugin
    .
    name
```

```
+
```

```
tool
    .
    name
```

```
+
```

```
(
    'toolbar'
```

```
if
```

```
for_toolbar
```

```
else
```

```
'menu'
)
```

```
plugin_toolbar_actions
```

```
=  
  
[]  
  
def  
  
create_plugin_action  
(  
    plugin  
,  
  
    tool  
,  
  
    for_toolbar  
,  
  
    actions  
    =  
    None  
,  
  
    toolbar_actions  
    =  
    None  
,  
  
    plugin_menu_actions  
    =  
    None  
):  
  
try  
:  
  
    ac  
  
    =
```

```
tool
.
create_action
(
for_toolbar
=
for_toolbar
)

if

ac

is

None
:

raise

RuntimeError
(
'create_action() failed to return an action'
)

except

Exception
:

prints
(
'Failed to create action for tool:'
,

tool
.
name
)
```

import

traceback

traceback

.

print_exc

()

return

sid

=

plugin_action_sid

(

plugin

,

tool

,

for_toolbar

)

if

actions

is

not

None

and

sid

in

actions

:

prints

(

f

'The

{

tool

.

name

}

tool from the

{

plugin

.

name

}

plugin has a non unique name, ignoring'

)

else

:

if

actions

is

not

None

:

```
actions
```

```
[
```

```
sid
```

```
]
```

```
=
```

```
ac
```

```
ac
```

```
.
```

```
sid
```

```
=
```

```
sid
```

```
if
```

```
for_toolbar
```

```
:
```

```
if
```

```
toolbar_actions
```

```
is
```

```
not
```

```
None
```

```
:
```

```
toolbar_actions
```

```
[
```

```
sid
```

```
]
```

```
=  
  
ac  
  
plugin_toolbar_actions  
.  
append  
(  
ac  
)  
  
ac  
.  
popup_mode  
=  
  
{  
    'instant'  
    :  
    QPushButton  
    .  
    ToolButtonPopupMode  
    .  
    InstantPopup  
    ,  
    'button'  
    :  
    QPushButton  
    .  
    ToolButtonPopupMode  
    .  
    MenuButtonPopup  
}  
.  
get  
(  
  
tool
```

```
.  
toolbar_button_popup_mode  
,
```

```
QToolButton
```

```
.  
ToolButtonPopupMode
```

```
.  
DelayedPopup  
)
```

```
else
```

```
:
```

```
if
```

```
plugin_menu_actions
```

```
is
```

```
not
```

```
None
```

```
:
```

```
plugin_menu_actions
```

```
.  
append  
(  
ac  
)
```

```
return
```

```
ac
```

```
_tool_memory
```

```
=
```

```
[]
```

```
# Needed to prevent the tool object from being garbage collected
```

```
def
```

```
create_plugin_actions
```

```
(
```

```
actions
```

```
,
```

```
toolbar_actions
```

```
,
```

```
plugin_menu_actions
```

```
):
```

```
del
```

```
_tool_memory
```

```
[:]
```

```
del
```

```
plugin_toolbar_actions
```

```
[:]
```

```
for
```

```
plugin
```

```
in
```

```
all_edit_book_tool_plugins
```

```
():  
  
try  
:  
  
tools  
  
=  
  
tuple  
(  
load_plugin_tools  
(  
plugin  
))  
  
except  
  
Exception  
:  
  
if  
  
plugin  
.  
installation_type  
  
is  
  
PluginInstallationType  
.  
BUILTIN  
:  
  
raise  
  
print
```

```
(  
    'Failed to load third-party plugin:'  
    ,  
    plugin  
    .  
    name  
    ,  
    file  
    =  
    sys  
    .  
    stderr  
    )
```

```
import
```

```
traceback
```

```
traceback  
.  
print_exc  
( )
```

```
continue
```

```
for
```

```
tool
```

```
in
```

```
tools  
:
```

```
_tool_memory
```

```
.
```

```
append
(
tool
)

if

tool
.
allowed_in_toolbar
:

create_plugin_action
(
plugin
,

tool
,

True
,

actions
,

toolbar_actions
,

plugin_menu_actions
)

if

tool
.
allowed_in_menu
:
```



```
create_plugin_action
(
    plugin
    ,

    tool
    ,

    False
    ,

    actions
    ,

    toolbar_actions
    ,

    plugin_menu_actions
)
```

def

```
install_plugin
(
    plugin
):
```

for

```
tool
```

in

```
load_plugin_tools
(
    plugin
):
```

if

```
tool
.
allowed_in_toolbar
:

sid

=

plugin_action_sid
(
plugin
,
tool
,
True
)

if

sid

not

in

tprefs
[
'global_plugins_toolbar'
]:

tprefs
[
'global_plugins_toolbar'
]

=
```

```
tprefs  
[  
  'global_plugins_toolbar'  
]
```

```
+
```

```
[  
  sid  
]
```

Source code for calibre.utils.formatter_functions

```
#!/usr/bin/env python
```

```
'''
```

```
Created on 13 Jan 2011
```

```
@author: charles
```

```
'''
```

```
__license__
```

```
=
```

```
'GPL v3'
```

```
__copyright__
```

```
=
```

```
'2010, Kovid Goyal <kovid@kovidgoyal.net>'
```

```
__docformat__
```

```
=
```

```
'restructuredtext en'
```

```
import
inspect
,
re
,
traceback
,
numbers
from
datetime
import
datetime
,
timedelta
from
math
import
trunc
,
floor
,
ceil
,
modf
from
```

contextlib

import

suppress

from

calibre

import

human_readable

,

prints

from

calibre.constants

import

DEBUG

from

calibre.ebooks.metadata

import

title_sort

from

calibre.utils.config

import

tweaks

from

calibre.utils.titlecase

import

titlecase

from

calibre.utils.icu

import

capitalize

,

strcmp

,

sort_key

from

calibre.utils.date

import

parse_date

,

format_date

,

now

,

UNDEFINED_DATE

from

calibre.utils.localization

import

```
calibre_langcode_to_name
,

canonicalize_lang

from

polyglot.builtins

import

iteritems
,

itervalues


class

FormatterFunctions
:

error_function_body

=

(
'def evaluate(self, formatter, kwargs, mi, locals):
\n
'

,

\t
return ""

+

```



```

-
(
'Duplicate user function name
{0}
.'

'Change the name or ensure that the functions are identical'
)

+

"""
)

def

__init__
(
self
):

self
.
_builtins

=

{}

self
.
_functions

=

{}

```

```
self
.
_functions_from_library

=

{}

def

register_builtin
(
self
,

func_class
):

if

not

isinstance
(
func_class
,

FormatterFunction
):

raise

ValueError
(
'Class
%s
is not an instance of FormatterFunction'
%
(
```

```
func_class
```

```
.
```

```
__class__
```

```
.
```

```
__name__
```

```
))
```

```
name
```

```
=
```

```
func_class
```

```
.
```

```
name
```

```
if
```

```
name
```

```
in
```

```
self
```

```
.
```

```
_functions
```

```
:
```

```
raise
```

```
ValueError
```

```
(
```

```
'Name
```

```
%s
```

```
already used'
```

```
%
```

```
name
```

```
)
```

```
self
.  
_builtins  
[  
name  
]
```

```
=
```

```
func_class
```

```
self
.  
_functions  
[  
name  
]
```

```
=
```

```
func_class
```

```
for
```

```
a
```

```
in
```

```
func_class
```

```
.  
aliases  
:
```

```
self
.  
_functions  
[  
a  
]
```

```
=  
  
func_class  
  
def  
  
_register_function  
(  
    self  
,  
    func_class  
,  
    replace  
    =  
    False  
):  
  
    if  
  
        not  
  
        isinstance  
        (  
            func_class  
,  
            FormatterFunction  
        ):  
  
            raise  
  
            ValueError  
            (  
                'Class  
                %s  
                is not an instance of FormatterFunction'  
                %
```

```
(  
  
func_class  
.  
__class__  
.  
__name__  
)
```

name

=

```
func_class  
.  
name
```

if

not

replace

and

name

in

self

```
.  
_functions  
:
```

raise

```
ValueError  
(  
'Name
```

```
%s
already used'
%
name
)

self
.
_functions
[
name
]

=

func_class

def

register_functions
(
self
,

library_uuid
,

funcs
):

self
.
_functions_from_library
[
library_uuid
]

=
```

```
funcs
```

```
self
```

```
·  
_register_functions  
(
```

```
def
```

```
_register_functions  
(  
self  
):
```

```
for
```

```
compiled_funcs
```

```
in
```

```
itervalues  
(  
self  
·  
_functions_from_library  
):
```

```
for
```

```
cls
```

```
in
```

```
compiled_funcs  
:
```

```
f
```



```
=
```

```
self
```

```
.
```

```
_functions
```

```
.
```

```
get
```

```
(
```

```
cls
```

```
.
```

```
name
```

```
,
```

```
None
```

```
)
```

```
replace
```

```
=
```

```
False
```

```
if
```

```
f
```

```
is
```

```
not
```

```
None
```

```
:
```

```
existing_body
```

```
=
```

```
f
```

```
.
```

```
program_text

new_body

=

cls
.
program_text

if

new_body

!=

existing_body
:

# Change the body of the template function to one that will

# return an error message. Also change the arg count to

# -1 (variable) to avoid template compilation errors

replace

=

True

func

=

[
```

```
cls
.
name
',
"
',
-
1
',

self
.
error_function_body
.
format
(
cls
.
name
)]

cls

=

compile_user_function
(
*
func
)

else
:

continue

formatter_functions
```

```
()  
.  
_register_function  
(  
cls  
,  
  
replace  
=  
replace  
)  
  
def  
  
unregister_functions  
(  
self  
,  
  
library_uuid  
):  
  
if  
  
library_uuid  
  
in  
  
self  
.  
_functions_from_library  
:  
  
for  
  
cls  
  
in
```

```
self
.  
_functions_from_library  
[  
library_uuid  
]:
```

```
self  
.  
_functions  
.  
pop  
(  
cls  
.  
name  
,  
  
None  
)
```

```
self  
.  
_functions_from_library  
.  
pop  
(  
library_uuid  
)
```

```
self  
.  
_register_functions  
(  

```

```
def
```

```
get_builtins
```

```
(
self
):

return

self
.
_builtins

def

get_builtins_and_aliases
(
self
):

res

=

{}

for

f

in

itervalues
(
self
.
_builtins
):

res
```

```
[
f
.
name
]

=

f

for

a

in

f
.
aliases
:

res
[
a
]

=

f

return

res

def

get_functions
(
```

```
self
):

return

self
.
_functions

def

reset_to_builtins
(
self
):

self
.
_functions

=

{}

for

n
,
c

in

self
.
_builtins
.
items
():
```



```
self
.  
_functions  
[  
n  
]
```

```
=
```

```
c
```

```
for
```

```
a
```

```
in
```

```
c
```

```
.  
aliases  
:
```

```
self  
.  
_functions  
[  
a  
]
```

```
=
```

```
c
```

```
_ff
```

```
=
```

```
FormatterFunctions
()

def
formatter_functions
():

global
_ff

return
_ff

class
FormatterFunction
:

doc
=

-
(
'No documentation provided'
)

name
=
```

```
'no name provided'
```

```
category
```

```
=
```

```
'Unknown'
```

```
arg_count
```

```
=
```

```
0
```

```
aliases
```

```
=
```

```
[]
```

```
is_python
```

```
=
```

```
True
```

```
def
```

```
evaluate
```

```
(
```

```
self
```

```
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,  
  
mi  
,  
  
locals  
,  
  
*  
args  
):  
  
raise  
  
NotImplementedError  
()  
  
def  
  
eval_  
(  
self  
,  
  
formatter  
,  
  
kwargs  
,  
  
mi  
,  
  
locals  
,  
  
*  
args  
):
```

```
ret

=

self
.
evaluate
(
formatter
,

kwargs
,

mi
,

locals
,

*
args
)

if

isinstance
(
ret
,

bytes
,

str
)):

return
```

```
ret
```

```
if
```

```
isinstance
```

```
(
```

```
ret
```

```
,
```

```
list
```

```
):
```

```
return
```

```
''
```

```
,
```

```
.
```

```
join
```

```
(
```

```
ret
```

```
)
```

```
if
```

```
isinstance
```

```
(
```

```
ret
```

```
,
```

```
(
```

```
numbers
```

```
.
```

```
Number
```

```
,
```

```
bool
```

```
)):
```

```
return
```

```
str
(  
ret  
)
```

class

BuiltinFormatterFunction

```
(  
FormatterFunction  
):
```

def

```
__init__  
(  
self  
):
```

```
formatter_functions  
()
```

```
.  
register_builtin  
(  
self  
)
```

```
eval_func
```

```
=
```

```
inspect
```

```
.  
getmembers  
(  
self  
.  

```

__class__

,

lambda

x

:

inspect

.

isfunction

(

x

)

and

x

.

__name__

==

'evaluate'

)

try

:

lines

=

[

|

[

4

:]

for


```
|  
  
in  
  
inspect  
.  
getsourcelines  
(  
eval_func  
[  
0  
])[  
1  
])[  
0  
]]  
  
except  
:  
  
lines  
  
=  
  
[]  
  
self  
.  
program_text  
  
=  
  
"  
.  
join  
(  
lines  
)
```

[\[docs\]](#)

class

BuiltinStrcmp

(
BuiltinFormatterFunction
):

name

=

'strcmp'

arg_count

=

5

category

=

'Relational'

__doc__

=

doc

=

—

```
(  
'strcmp(x, y, lt, eq, gt) -- does a case-insensitive comparison of x '  
  
'and y as strings. Returns lt if x < y. Returns eq if x == y. '  
  
'Otherwise returns gt. In many cases the lexical comparison operators '  
  
'(>, <, == etc) can replace this function.'  
)
```

def

evaluate

```
(  
self
```

,

formatter

,

kwargs

,

mi

,

locals

,

x

,

y

,

lt

,

```
eq
```

```
,
```

```
gt
```

```
):
```

```
v
```

```
=
```

```
strcmp
```

```
(
```

```
x
```

```
,
```

```
y
```

```
)
```

```
if
```

```
v
```

```
<
```

```
0
```

```
:
```

```
return
```

```
It
```

```
if
```

```
v
```

```
==
```

```
0
```

```
:
```

return

eq

return

gt

[\[docs\]](#)

class

BuiltinCmp

(
BuiltinFormatterFunction
):

name

=

'cmp'

category

=

'Relational'

arg_count

=

5

__doc__

```
=
```

```
doc
```

```
=
```

```
-  
(
```

```
'cmp(x, y, lt, eq, gt) -- compares x and y after converting both to '
```

```
'numbers. Returns lt if x < y. Returns eq if x == y. Otherwise returns gt. '
```

```
'In many cases the numeric comparison operators '
```

```
'(>#, <#, ==# etc) can replace this function.'
```

```
)
```

```
def
```

```
evaluate
```

```
(
```

```
self
```

```
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
```

```
,
```

```
locals
```

```
,
```

```
x
```

```
,  
  
y  
,  
  
lt  
,  
  
eq  
,  
  
gt  
):  
  
x  
  
=  
  
float  
(  
x  
  
if  
  
x  
  
and  
  
x  
  
!=  
  
'None'  
  
else  
  
0  
)  
  
y
```

```
=  
  
float  
(  
y  
  
if  
  
y  
  
and  
  
y  
  
!=  
  
'None'  
  
else  
  
0  
)  
  
if  
  
x  
  
<  
  
y  
:  
  
return  
  
It  
  
if  
  
x
```



```
==
```

```
y  
:
```

```
return
```

```
eq
```

```
return
```

```
gt
```

```
\[docs\]  
class
```

```
BuiltinFirstMatchingCmp
```

```
(  
    BuiltinFormatterFunction  
):
```

```
name
```

```
=
```

```
'first_matching_cmp'
```

```
category
```

```
=
```

```
'Relational'
```

```
arg_count
```

```
=
```

```

-
1

__doc__

=

doc

=

-
(
'first_matching_cmp(val, [cmp1, result1,]+, else_result) -- '

'compares "val < cmpN" in sequence, returning resultN for '

'the first comparison that succeeds. Returns else_result '

'if no comparison succeeds. Example: '

'first_matching_cmp(10,5,"small",10,"middle",15,"large","giant") '

'returns "large". The same example with a first value of 16 returns "giant".'
)

def

evaluate
(
self
,
formatter
,

```

```
kwargs
,
mi
,
locals
,
*
args
):

if

(
len
(
args
)

%

2

)

!=

0
:

raise

ValueError
(
    -
    (
'first_matching_cmp requires an even number of arguments'
    ))
```

```
val
```

```
=
```

```
float
```

```
(
```

```
args
```

```
[
```

```
0
```

```
]
```

```
if
```

```
args
```

```
[
```

```
0
```

```
]
```

```
and
```

```
args
```

```
[
```

```
0
```

```
]
```

```
!=
```

```
'None'
```

```
else
```

```
0
```

```
)
```

```
for
```

```
i
```

```
in
```

```
range
```

```
(  
1  
,  
  
len  
(  
args  
)
```

```
-
```

```
1
```

```
,
```

```
2
```

```
):
```

```
c
```

```
=
```

```
float
```

```
(
```

```
args
```

```
[
```

```
i
```

```
]
```

```
if
```

```
args
```

```
[
```

```
i
```

```
]
```

```
and
```

```
args
```

```
[
```

```
i
```

```
]
```

```
!=
```

```
'None'
```

```
else
```

```
0
```

```
)
```

```
if
```

```
val
```

```
<
```

```
c
```

```
:
```

```
return
```

```
args
```

```
[
```

```
i
```

```
+
```

```
1
```

```
]
```

```
return
```

```
args
```

```
[
```

```
len
```

```
(
```

```
args
```

```
)
```

```
-
```

```
1
```

```
]
```

[\[docs\]](#)

class

BuiltinStrcat

(
BuiltinFormatterFunction
):

name

=

'strcat'

arg_count

=

-

1

category

=

'String manipulation'

__doc__

=

doc

=

-

(
'strcat(a [, b]*) -- can take any number of arguments. Returns the '

```
'string formed by concatenating all the arguments'  
)
```

```
def
```

```
    evaluate
```

```
    (  
    self
```

```
    ,
```

```
    formatter
```

```
    ,
```

```
    kwargs
```

```
    ,
```

```
    mi
```

```
    ,
```

```
    locals
```

```
    ,
```

```
    *
```

```
    args
```

```
    ):
```

```
    i
```

```
    =
```

```
    0
```

```
    res
```

```
    =
```

```
    "
```


for

i

in

range

(

0

,

len

(

args

)):

res

+=

args

[

i

]

return

res

[\[docs\]](#)

class

BuiltinStrlen

(

BuiltinFormatterFunction

):

name

```
=
```

```
'strlen'
```

```
arg_count
```

```
=
```

```
1
```

```
category
```

```
=
```

```
'String manipulation'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-
```

```
(
```

```
'strlen(a) -- Returns the length of the string passed as '
```

```
'the argument'
```

```
)
```

```
def
```

```
evaluate
```

```
(
```

```
self
```

```
,  
  
formatter  
,  
  
kwargs  
,  
  
mi  
,  
  
locals  
,  
  
a  
):  
  
try  
:  
  
return  
  
len  
(  
a  
)  
  
except  
:  
  
return  
  
-  
1  
  
\[docs\]  
class
```

BuiltinAdd

```
(  
    BuiltinFormatterFunction  
):
```

```
name
```

```
=
```

```
'add'
```

```
arg_count
```

```
=
```

```
-  
1
```

```
category
```

```
=
```

```
'Arithmetic'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-  
(
```

```
'add(x [, y]*) -- returns the sum of its arguments. '
```

```
'Throws an exception if an argument is not a number. '
```

'This function can often be '

'replaced with the + operator.'

)

def

evaluate

(

self

,

formatter

,

kwargs

,

mi

,

locals

,

*

args

):

res

=

0

for

v

in

args

:

v

=

float

(

v

if

v

and

v

!=

'None'

else

0

)

res

+=

v

return

str

```
(  
res  
)
```

[\[docs\]](#)

class

BuiltinSubtract

```
(  
BuiltinFormatterFunction  
):
```

name

=

'subtract'

arg_count

=

2

category

=

'Arithmetic'

__doc__

=

doc

=

```
-  
(  
'subtract(x, y) -- returns x - y. Throws an exception if '  
  
'either x or y are not numbers. This function can often be '  
  
'replaced with the - operator.'  
)
```

def

evaluate

```
(  
self  
,
```

formatter

```
,
```

kwargs

```
,
```

mi

```
,
```

locals

```
,
```

x

```
,
```

y

```
):
```

x

=


```
float
(  
x  
  
if  
  
x  
  
and  
  
x  
  
!=  
  
'None'  
  
else  
  
0  
)
```

```
y  
  
=  
  
float  
(  
y  
  
if  
  
y  
  
and  
  
y  
  
!=  
  
'None'  
  
else
```

```
0  
)
```

```
return
```

```
str
```

```
(  
x
```

```
-
```

```
y  
)
```

```
\[docs\]
```

```
class
```

```
BuiltinMultiply
```

```
(  
BuiltinFormatterFunction  
):
```

```
name
```

```
=
```

```
'multiply'
```

```
arg_count
```

```
=
```

```
-
```

```
1
```

```
category
```

```
=  
  
'Arithmetic'  
  
__doc__  
  
=  
  
doc  
  
=  
  
-  
(  
'multiply(x [, y]*) -- returns the product of its arguments. '  
  
'Throws an exception if any argument is not a number. '  
  
'This function can often be replaced with the * operator.'  
)  
  
def  
  
evaluate  
(  
self  
,  
  
formatter  
,  
  
kwargs  
,  
  
mi  
,  
  
locals
```

```
,  
  
*  
args  
):  
  
res  
  
=  
  
1  
  
for  
  
v  
  
in  
  
args  
:  
  
v  
  
=  
  
float  
(  
v  
  
if  
  
v  
  
and  
  
v  
  
!=  
  
'None'
```

else

0

)

res

*=

v

return

str

(

res

)

[\[docs\]](#)

class

BuiltinDivide

(

BuiltinFormatterFunction

):

name

=

'divide'

arg_count

=

2

```
category
```

```
=
```

```
'Arithmetic'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
(
```

```
'divide(x, y) -- returns x / y. Throws an exception if '
```

```
'either x or y are not numbers.'
```

```
' This function can often be replaced with the / operator.'
```

```
)
```

```
def
```

```
evaluate
```

```
(
```

```
self
```

```
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
',
locals
',
x
',
y
):

x

=

float
(
x

if

x

and

x

!=

'None'

else

0
)

y

=
```

```
float
(
y

if

y

and

y

!=

'None'

else

0
)

return

str
(
x

/

y
)

\[docs\]
class

BuiltInCeiling
(
    BuiltinFormatterFunction
):
```



```
name
=
'ceiling'

arg_count
=
1

category
=
'Arithmetic'

__doc__
=
doc
=
(
'ceiling(x) -- returns the smallest integer greater '

'than or equal to x. Throws an exception if x is '

'not a number.'
)

def
```

```
evaluate
(
self
,
formatter
,
kwargs
,
mi
,
locals
,
x
):

x

=

float
(
x

if
x

and
x

!=

'None'

else
```

```
0  
)
```

```
return
```

```
str  
(  
int  
(  
ceil  
(  
x  
)))
```

```
\[docs\]  
class
```

```
BuiltinFloor
```

```
(  
BuiltinFormatterFunction  
):
```

```
name
```

```
=
```

```
'floor'
```

```
arg_count
```

```
=
```

```
1
```

```
category
```

```
=
```

```
'Arithmetic'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
—  
(
```

```
'floor(x) -- returns the largest integer less '
```

```
'than or equal to x. Throws an exception if x is '
```

```
'not a number.'
```

```
)
```

```
def
```

```
evaluate
```

```
(
```

```
self
```

```
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
```

```
,
```

```
locals
```

```
,
```

```
x  
):
```

```
x
```

```
=
```

```
float
```

```
(  
x
```

```
if
```

```
x
```

```
and
```

```
x
```

```
!=
```

```
'None'
```

```
else
```

```
0  
)
```

```
return
```

```
str
```

```
(  
int
```

```
(  
floor
```

```
(  
x
```

```
)))
```

[\[docs\]](#)

class

BuiltinRound

(
BuiltinFormatterFunction
):

name

=

'round'

arg_count

=

1

category

=

'Arithmetic'

__doc__

=

doc

=

(
'round(x) -- returns the nearest integer to x. '

```
'Throws an exception if x is not a number.'  
)
```

```
def
```

```
    evaluate
```

```
    (  
    self
```

```
,
```

```
    formatter
```

```
,
```

```
    kwargs
```

```
,
```

```
    mi
```

```
,
```

```
    locals
```

```
,
```

```
    x
```

```
):
```

```
    x
```

```
    =
```

```
    float
```

```
    (  
    x
```

```
    if
```

```
        x
```

```
        and
```

```
            x
```

```
!=
```

```
'None'
```

```
else
```

```
0
```

```
)
```

```
return
```

```
str
```

```
(
```

```
int
```

```
(
```

```
round
```

```
(
```

```
x
```

```
)))
```

```
\[docs\]
```

```
class
```

```
BuiltinMod
```

```
(
```

```
BuiltinFormatterFunction
```

```
):
```

```
name
```

```
=
```

```
'mod'
```

```
arg_count
```

```
=
```



```
2
```

```
category
```

```
=
```

```
'Arithmetic'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-
```

```
(
```

```
'mod(x) -- returns floor(remainder of x / y). '
```

```
'Throws an exception if either x or y is not a number.'
```

```
)
```

```
def
```

```
evaluate
```

```
(
```

```
self
```

```
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
```

```
,
```

```
locals
```

```
,
```

```
x
```

```
,
```

```
y
```

```
):
```

```
x
```

```
=
```

```
float
```

```
(
```

```
x
```

```
if
```

```
x
```

```
and
```

```
x
```

```
!=
```

```
'None'
```

```
else
```

```
0
```

```
)
```

```
y
```

```
=
```

```
float
```

```
(
```

```
y
```

```
if
```

```
y
```

```
and
```

```
y
```

```
!=
```

```
'None'
```

```
else
```

```
0
```

```
)
```

```
return
```

```
str
```

```
(
```

```
int
```

```
(
```

```
x
```

```
%
```

```
y
```

```
))
```

```
\[docs\]
```

```
class
```

```
BuiltinFractionalPart
```

```
(
```

```
BuiltinFormatterFunction
```

```
):
```

```
name
=
'fractional_part'

arg_count
=
1

category
=
'Arithmetic'

__doc__
=
doc
=
(
'fractional_part(x) -- returns the value after the decimal '

'point. For example, fractional_part(3.14) returns 0.14. '

'Throws an exception if x is not a number.'
)

def
```

```
evaluate
(
self
,
formatter
,
kwargs
,
mi
,
locals
,
x
):

x

=

float
(
x

if
x

and
x

!=

'None'

else
```

```
0  
)
```

```
return
```

```
str  
(  
  modf  
(  
    x  
  ) [  
    0  
  ] )
```

```
\[docs\]  
class
```

```
BuiltinTemplate
```

```
(  
  BuiltinFormatterFunction  
) :
```

```
name
```

```
=
```

```
'template'
```

```
arg_count
```

```
=
```

```
1
```

```
category
```

```
=
```

'Recursion'

__doc__

=

doc

=

—
(

'template(x) -- evaluates x as a template. The evaluation is done '

'in its own context, meaning that variables are not shared between '

'the caller and the template evaluation. Because the { and } '

'characters are special, you must use [[for the { character and '

']] for the } character; they are converted automatically. '

'For example, template(

\

[[title_sort]]

\

) will evaluate the '

'template

{title_sort}

and return its value. Note also that '

'prefixes and suffixes (the `|prefix|suffix` syntax) cannot be '

```
'used in the argument to this function when using template program mode.'  
)
```

```
def
```

```
    evaluate
```

```
    (  
    self
```

```
,
```

```
    formatter
```

```
,
```

```
    kwargs
```

```
,
```

```
    mi
```

```
,
```

```
    locals
```

```
,
```

```
    template
```

```
):
```

```
    template
```

```
=
```

```
    template
```

```
.
```

```
    replace
```

```
    (  
    '['
```

```
,
```

```
    '{'
```

```
)
```



```
.
replace
(
    ']]'
,
    '}'
)

return

formatter
.
    __class__
()
.
safe_format
(
    template
,
    kwargs
,
    'TEMPLATE'
,
    mi
)

[docs]
class

BuiltinEval
(
    BuiltinFormatterFunction
):

name
```

```
=  
  
'eval'  
  
arg_count  
  
=  
  
1  
  
category  
  
=  
  
'Recursion'  
  
__doc__  
  
=  
  
doc  
  
=  
  
-  
(  
'eval(template) -- evaluates the template, passing the local '  
  
'variables (those  
'\nassign  
'\n  
'ed to) instead of the book metadata. '  
  
' This permits using the template processor to construct complex '  
  
'results from local variables. Because the { and }'
```

'characters are special, you must use [[for the { character and '

']] for the } character; they are converted automatically. '

'Note also that prefixes and suffixes (the `|prefix|suffix` syntax) '

'cannot be used in the argument to this function when using '

'template program mode.'

)

def

evaluate

(

self

,

formatter

,

kwargs

,

mi

,

locals

,

template

):

from

`calibre.utils.formatter`

import

EvalFormatter

template

=

template

.

replace

(

'[['

,

'{'

)

.

replace

(

']']

,

'}'

)

return

EvalFormatter

()

.

safe_format

(

template

,

locals

,

```
'EVAL'
```

```
,
```

```
None
```

```
)
```

```
\[docs\]
```

```
class
```

```
BuiltinAssign
```

```
(
```

```
BuiltinFormatterFunction
```

```
):
```

```
name
```

```
=
```

```
'assign'
```

```
arg_count
```

```
=
```

```
2
```

```
category
```

```
=
```

```
'Other'
```

```
\_\_doc\_\_
```

```
=
```

```
doc
```

=

(

'assign(id, val) -- assigns val to id, then returns val. '

'id must be an identifier, not an expression. '

'This function can often be replaced with the = operator.'

)

def

evaluate

(

self

,

formatter

,

kwargs

,

mi

,

locals

,

target

,

value

):

locals

```
[  
target  
]
```

```
=
```

```
value
```

```
return
```

```
value
```

```
\[docs\]
```

```
class
```

```
BuiltinListSplit
```

```
(  
BuiltinFormatterFunction  
):
```

```
name
```

```
=
```

```
'list_split'
```

```
arg_count
```

```
=
```

```
3
```

```
category
```

```
=
```

```
'List manipulation'
```

`__doc__`

`=`

`doc`

`=`

`-`
`(`

`'list_split(list_val, sep, id_prefix) -- splits the list_val '`

`"into separate values using 'sep', then assigns the values "`

`"to variables named 'id_prefix_N' where N is the position "`

`"of the value in the list. The first item has position 0 (zero). "`

`"The function returns the last element in the list. "`

`"Example: split('one:two:foo', ':', 'var') is equivalent "`

`"to var_0 = 'one'; var_1 = 'two'; var_2 = 'foo'."`

`)`

`def`

`evaluate`

`(`

`self`

`,`

`formatter`

`,`


```
kwargs
```

```
,
```

```
mi
```

```
,
```

```
locals
```

```
,
```

```
list_val
```

```
,
```

```
sep
```

```
,
```

```
id_prefix
```

```
):
```

```
|
```

```
=
```

```
[
```

```
v
```

```
.
```

```
strip
```

```
()
```

```
for
```

```
v
```

```
in
```

```
list_val
```

```
.
```

```
split
```

```
(
```

```
sep
```

```
)]
```

```
res
```

```
=
```

```
"
```

```
for
```

```
i
```

```
,
```

```
v
```

```
in
```

```
enumerate
```

```
(
```

```
l
```

```
):
```

```
res
```

```
=
```

```
locals
```

```
[
```

```
id_prefix
```

```
+
```

```
' '
```

```
—
```

```
+
```

```
str
```

```
(
```

```
i
```

```
)]
```

```
=
```

```
v
```

```
return
```

res

[\[docs\]](#)

class

BuiltinPrint

(
BuiltinFormatterFunction
):

name

=

'print'

arg_count

=

-

1

category

=

'Other'

__doc__

=

doc

=

```
-  
(  
    'print(a[, b]*) -- prints the arguments to standard output. '  
  
    'Unless you start calibre from the command line (calibre-debug -g), '  
  
    'the output will go to a black hole.'  
)
```

def

evaluate

```
(  
    self  
,
```

formatter

```
,
```

kwargs

```
,
```

mi

```
,
```

locals

```
,
```

```
*
```

args

```
):
```

print

```
(  
    args  
)
```

```
return
```

```
"
```

```
\[docs\]
```

```
class
```

```
BuiltinField
```

```
(  
    BuiltinFormatterFunction  
):
```

```
name
```

```
=
```

```
'field'
```

```
arg_count
```

```
=
```

```
1
```

```
category
```

```
=
```

```
'Get values from metadata'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
__
(
'field(lookup_name) -- returns the metadata field named by lookup_name'
)
```

def

evaluate

```
(
self
```

```
,
```

formatter

```
,
```

kwargs

```
,
```

mi

```
,
```

locals

```
,
```

name

```
):
```

return

formatter

```
.
```

get_value

```
(
```

name

```
,
```

[],

kwargs

```
)
```

```
\[docs\]
```

```
class
```

```
BuiltinRawField
```

```
(
```

```
BuiltinFormatterFunction
```

```
):
```

```
name
```

```
=
```

```
'raw_field'
```

```
arg_count
```

```
=
```

```
-
```

```
1
```

```
category
```

```
=
```

```
'Get values from metadata'
```

```
\_\_doc\_\_
```

```
=
```

```
doc
```

```
=
```

```
-
```

```
(  
'raw_field(lookup_name [, optional_default]) -- returns the '  
  
'metadata field named by lookup_name without applying any formatting. '  
  
'It evaluates and returns the optional second argument '  
  
'''default' if the field is undefined ('None')."  
)
```

def

evaluate

```
(  
self
```

,

formatter

,

kwargs

,

mi

,

locals

,

name

,

default

=

None

):


```
res
```

```
=
```

```
getattr
```

```
(
```

```
mi
```

```
,
```

```
name
```

```
,
```

```
None
```

```
)
```

```
if
```

```
res
```

```
is
```

```
None
```

```
and
```

```
default
```

```
is
```

```
not
```

```
None
```

```
:
```

```
return
```

```
default
```

```
if
```

```
isinstance
(
res
,

list
):

fm

=

mi
.
metadata_for_field
(
name
)

if
fm
is
None
:

return

', '
.
join
(
res
)

return
```

```
fm
[
'is_multiple'
][
'list_to_ui'
]
.
join
(
res
)
```

return

```
str
(
res
)
```

[\[docs\]](#)

class

BuiltinRawList

```
(
BuiltinFormatterFunction
):
```

name

=

'raw_list'

arg_count

=

2

category

=

'Get values from metadata'

__doc__

=

doc

=

(

'raw_list(lookup_name, separator) -- returns the metadata list '

'named by lookup_name without applying any formatting or sorting and '

'with items separated by separator.'

)

def

evaluate

(

self

,

formatter

,

kwargs

,

mi

```
,  
  
locals  
,  
  
name  
,  
  
separator  
):  
  
res  
  
=  
  
getattr  
(  
mi  
,  
  
name  
,  
  
None  
)  
  
if  
  
not  
  
isinstance  
(  
res  
,  
  
list  
):  
  
return
```

```
"
%s
is not a list"

%

name

return

separator

.
join
(
res
)

[docs]
class

BuiltinSubstr
(
BuiltinFormatterFunction
):

name

=

'substr'

arg_count

=

3

category
```

=

'String manipulation'

__doc__

=

doc

=

-
(

'substr(str, start, end) -- returns the start

'

th through the end

'

th '

'characters of str. The first character in str is the zero

'

th '

'character. If end is negative, then it indicates that many '

'characters counting from the right. If end is zero, then it '

'indicates the last character. For example, substr(

'

12345

'

, 1, 0) '

'returns

'

```
2345
\'
, and substr(
\'
12345
\'
, 1, -1) returns
\'
234
\'
.'
)
```

def

evaluate

(
self

,

formatter

,

kwargs

,

mi

,

locals

,

str_

,

start_

,

end_

):

return

```
str_  
[  
int  
(  
start_  
):
```

```
len  
(  
str_  
)
```

if

```
int  
(  
end_  
)
```

==

0

else

```
int  
(  
end_  
)]
```

[\[docs\]](#)

class

BuiltinLookup

```
(  
BuiltinFormatterFunction  
):
```

name

=

'lookup'

arg_count

=

-

1

category

=

'Iterating over values'

__doc__

=

doc

=

(

'lookup(val, [pattern, field,]+ else_field) -- '

'like switch, except the arguments are field (metadata) names, not '

'text. The value of the appropriate field will be fetched and used. '

'Note that because composite columns are fields, you can use this '

'function in one composite field to use the value of some other '

'composite field. This is extremely useful when constructing '

'variable save paths'

)

def

evaluate

(

self

,

formatter

,

kwargs

,

mi

,

locals

,

val

,

*

args

):

if

len

(

```
args
)

==

2
:

# here for backwards compatibility

if

val
:

return

formatter
.
vformat
(
'{'
+
args
[
0
]
.
strip
()
+
'}'
,
[],
kwargs
)

else
```

```
:  
  
return  
  
formatter  
.  
vformat  
(  
    '{'  
    +  
    args  
    [  
        1  
    ]  
    .  
    strip  
    ()  
    +  
    '}'  
    ,  
    [],  
    kwargs  
    )  
  
if  
  
    (  
        len  
        (  
            args  
        )  
        %  
        2  
    )  
  
    !=
```

```
1
:

raise

ValueError
(
-
(
'lookup requires either 2 or an odd number of arguments'
))

i

=

0

while

i

<

len
(
args
):

if

i

+

1

>=
```

```
len
(
args
):

return

formatter
.
vformat
(
{'
+
args
[
i
]
.
strip
()
+
'}
,
[],
kwargs
)

if

re
.
search
(
args
[
```

```
i
],

val
,

flags
=
re
.
|
):

return

formatter
.
vformat
(
'{'
+
args
[
i
+
1
]
.
strip
()
+
'{'
,

[]),

kwargs
)
```



```
i
+=
2
[docs]
class
BuiltinTest
(
BuiltinFormatterFunction
):

name
=
'test'

arg_count
=
3

category
=
'If-then-else'

__doc__
=
doc
```

```
=
```

```
-  
(
```

```
'test(val, text if not empty, text if empty) -- return `text if not '
```

```
'empty` if val is not empty, otherwise return `text if empty`'
```

```
)
```

```
def
```

```
evaluate
```

```
(
```

```
self
```

```
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
```

```
,
```

```
locals
```

```
,
```

```
val
```

```
,
```

```
value_if_set
```

```
,
```

```
value_not_set
```

```
):
```

```
if
```

```
val
:

return

value_if_set

else
:

return

value_not_set

\[docs\]
class

BuiltinContains
(
  BuiltinFormatterFunction
):

name

=

'contains'

arg_count

=

4

category
```

=

'If-then-else'

__doc__

=

doc

=

—
(

'contains(val, pattern, text if match, text if not match) -- checks '

'if val contains matches for the regular expression `pattern`.'

'Returns `text if match` if matches are found, otherwise it returns '

'`text if no match`'

)

def

evaluate

(

self

,

formatter

,

kwargs

,

mi

```
,  
  
locals  
,  
  
val  
,  
  
test  
,  
  
value_if_present  
,  
  
value_if_not  
):
```

if

```
re  
.  
search  
(  
test  
,  
  
val  
,  
  
flags  
=  
re  
.  
|  
):
```

return

```
value_if_present
```

else

:

return

value_if_not

[\[docs\]](#)

class

BuiltinSwitch

(
BuiltinFormatterFunction
):

name

=

'switch'

arg_count

=

-

1

category

=

'Iterating over values'

__doc__

```
=
```

```
doc
```

```
=
```

```
-  
(
```

```
'switch(val, [pattern, value,]+ else_value) -- '
```

```
'for each `pattern, value` pair, checks if `val` matches '
```

```
'the regular expression `pattern` and if so, returns that '
```

```
`value`. If no pattern matches, then `else_value` is returned. '
```

```
'You can have as many `pattern, value` pairs as you want'
```

```
)
```

```
def
```

```
evaluate
```

```
(
```

```
self
```

```
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
```

```
,
```

```
locals
```

```
,
```

```
val
,
*
args
):

if

(
  len
  (
    args
  )
  %
  2
)
!=
1
:

raise

ValueError
(
  -
  (
    'switch requires an odd number of arguments'
  ))

i
=
0
```



```
while
```

```
i
```

```
<
```

```
len
```

```
(
```

```
args
```

```
):
```

```
if
```

```
i
```

```
+
```

```
1
```

```
>=
```

```
len
```

```
(
```

```
args
```

```
):
```

```
return
```

```
args
```

```
[
```

```
i
```

```
]
```

```
if
```

```
re
```

```
.
```

```
search
```

```
(  
args  
[  
i  
],
```

```
val  
,
```

```
flags  
=  
re  
.  
|  
):
```

```
return
```

```
args  
[  
i  
+  
1  
]
```

```
i
```

```
+=
```

```
2
```

```
\[docs\]
```

```
class
```

```
BuiltinStrcatMax
```

```
(  
BuiltinFormatterFunction  
):
```

name

=

'strcat_max'

arg_count

=

-

1

category

=

'String manipulation'

__doc__

=

doc

=

(

'strcat_max(max, string1 [, prefix2, string2]*) -- '

'Returns a string formed by concatenating the arguments. The '

'returned value is initialized to string1. `Prefix, string` '

'pairs are added to the end of the value as long as the '

'resulting string length is less than `max`. String1 is returned '

'even if string1 is longer than max. You can pass as many '

`prefix, string` pairs as you wish.'

)

def

evaluate

(

self

,

formatter

,

kwargs

,

mi

,

locals

,

*

args

):

if

len

(

args

)

```
<

2
:

raise

ValueError
(
-
(
'strcat_max requires 2 or more arguments'
))

if

(
len
(
args
)
%

2
)

!=

0
:

raise

ValueError
(
-
(
'strcat_max requires an even number of arguments'
))
```

```
try
:

max

=

int
(
args
[
0
])

except
:

raise

ValueError
(
    _
    (
        'first argument to strcat_max must be an integer'
    ))

i

=

2

result

=
```

```
args  
[  
1  
]
```

```
try  
:
```

```
while
```

```
i
```

```
<
```

```
len  
(  
args  
):
```

```
if
```

```
(  
len  
(  
result  
)
```

```
+
```

```
len  
(  
args  
[  
i  
])
```

```
+
```

```
len
```

```
(  
  args  
  [  
    i  
    +  
    1  
  ]))
```

```
>
```

```
max  
:
```

```
break
```

```
result
```

```
=
```

```
result
```

```
+
```

```
args  
[  
  i  
]
```

```
+
```

```
args  
[  
  i  
  +  
  1  
]
```

```
i
```

```
+=
```


2

except

:

pass

return

result

.

strip

()

[\[docs\]](#)

class

BuiltinInList

(

BuiltinFormatterFunction

):

name

=

'in_list'

arg_count

=

-

1

```
category

=

'List lookup'

__doc__

=

doc

=

-
(
'in_list(val, separator, [ pattern, found_val, ]+ not_found_val) -- '

'treating val as a list of items separated by separator, '

'if the pattern matches any of the list values then return found_val.'

'If the pattern matches no list value then return '

'not_found_val. The pattern and found_value pairs can be repeated as '

'many times as desired. The patterns are checked in order. The '

'found_val for the first match is returned. '

'Aliases: in_list(), list_contains()'
)

aliases
```

```
=
```

```
[  
    'list_contains'  
]
```

```
def
```

```
    evaluate
```

```
    (  
        self
```

```
,
```

```
        formatter
```

```
,
```

```
        kwargs
```

```
,
```

```
        mi
```

```
,
```

```
        locals
```

```
,
```

```
        val
```

```
,
```

```
        sep
```

```
,
```

```
        *
```

```
        args
```

```
    );
```

```
if
```

```
    (  
        len
```

```
(
args
)

%

2
)

!=

1
:

raise

ValueError
(
-
(
'in_list requires an odd number of arguments'
))

|

=

[
v
.
strip
()

for

v

in

val

.
```

```
split  
(  
sep  
)
```

```
if
```

```
v  
.  
strip  
()]
```

```
i
```

```
=
```

```
0
```

```
while
```

```
i
```

```
<
```

```
len  
(  
args  
):
```

```
if
```

```
i
```

```
+
```

```
1
```

```
>=
```

```
len
```

```
(  
  args  
):
```

```
  return
```

```
  args  
  [  
    i  
  ]
```

```
  sf
```

```
  =
```

```
  args  
  [  
    i  
  ]
```

```
  fv
```

```
  =
```

```
  args  
  [  
    i  
    +  
    1  
  ]
```

```
  if
```

```
  |  
  :
```

```
  for
```

v

in

|

:

if

re

.

search

(

sf

,

v

,

flags

=

re

.

|

):

return

fv

i

+=

2

[\[docs\]](#)

class

BuiltinStrInList

```
(  
    BuiltinFormatterFunction  
):
```

name

=

'str_in_list'

arg_count

=

-

1

category

=

'List lookup'

__doc__

=

doc

=

-

```
(  
'str_in_list(val, separator, [string, found_val, ]+ not_found_val) -- '
```

'treating val as a list of items separated by separator, if the '

'string matches any of the list values then return found_val.'

'If the string matches no list value then return '

'not_found_val. The comparison is exact match (not contains) and is '

'case insensitive. The string and found_value pairs can be repeated as '

'many times as desired. The patterns are checked in order. The '

'found_val for the first match is returned.'

)

def

evaluate

(

self

,

formatter

,

kwargs

,

mi

,

locals

,

val

,

```
sep
',
*
args
):

if

(
len
(
args
)

%

2
)

!=

1
:

raise

ValueError
(
-
(
'str_in_list requires an odd number of arguments'
))

|

=

[
v
```

```
.  
strip  
( )
```

```
for
```

```
v
```

```
in
```

```
val
```

```
.  
split  
(  
sep  
)
```

```
if
```

```
v
```

```
.  
strip  
( )]
```

```
i
```

```
=
```

```
0
```

```
while
```

```
i
```

```
<
```

```
len  
(  
args  
):
```

```
if
```

```
i
```

```
+
```

```
1
```

```
>=
```

```
len
```

```
(
```

```
args
```

```
):
```

```
return
```

```
args
```

```
[
```

```
i
```

```
]
```

```
sf
```

```
=
```

```
args
```

```
[
```

```
i
```

```
]
```

```
fv
```

```
=
```

```
args
```

```
[
```

```
i
```

```
+
```

```
1  
]
```

```
c
```

```
=
```

```
[  
v
```

```
.  
strip  
(
```

```
for
```

```
v
```

```
in
```

```
sf  
.  
split  
(  
sep  
)
```

```
if
```

```
v  
.  
strip  
(  
[]
```

```
if
```

```
|  
:
```

```
for
```

```
v
```

```
in
```

```
l
```

```
:
```

```
for
```

```
t
```

```
in
```

```
c
```

```
:
```

```
if
```

```
strcmp
```

```
(
```

```
t
```

```
,
```

```
v
```

```
)
```

```
==
```

```
0
```

```
:
```

```
return
```

```
fv
```

```
i
```

```
+=
```

2

[\[docs\]](#)

class

BuiltinIdentifierInList

(
BuiltinFormatterFunction
):

name

=

'identifier_in_list'

arg_count

=

4

category

=

'List lookup'

__doc__

=

doc

=

—
(

```
'identifier_in_list(val, id, found_val, not_found_val) -- '  
  
'treat val as a list of identifiers separated by commas, '  
  
'comparing the string against each value in the list. An identifier '  
  
'has the format "identifier:value". The id parameter should be '  
  
'either "id" or "id:regex". The first case matches if there is any '  
  
'identifier with that id. The second case matches if the regex '  
  
'matches the identifier  
'\s value. If there is a match, '  
  
'return found_val, otherwise return not_found_val.'  
)
```

def

evaluate

(
self
,

formatter

,

kwargs

,

mi

,


```
locals
```

```
,
```

```
val
```

```
,
```

```
ident
```

```
,
```

```
fv
```

```
,
```

```
nfv
```

```
):
```

```
|
```

```
=
```

```
[
```

```
v
```

```
.
```

```
strip
```

```
()
```

```
for
```

```
v
```

```
in
```

```
val
```

```
.
```

```
split
```

```
(
```

```
;;
```

```
,
```

```
)
```

```
if
```

```
v
```

```
.
strip
()]

(
id
,

-
,

regexp
)

=

ident
.
partition
(
':.'
)

if

not

id
:

return

nfv

id

+=

':.'
```

if

|

:

for

v

in

|

:

if

v

.

startswith

(

id

):

if

not

regex

or

re

.

search

(

regex

,

```
v
[
len
(
id
):],

flags
=
re
.
|
):

return

fv

return

nfv

\[docs\]
class

BuiltinRe
(
BuiltinFormatterFunction
):

name

=

're'

arg_count
```

```
=  
  
3  
  
category  
  
=  
  
'String manipulation'  
  
__doc__  
  
=  
  
doc  
  
=  
  
-  
(  
're(val, pattern, replacement) -- return val after applying '  
  
'the regular expression. All instances of `pattern` are replaced '  
  
'with `replacement`. As in all of calibre, these are '  
  
'Python-compatible regular expressions'  
)  
  
def  
  
evaluate  
(  
self  
,
```

```
formatter
',
kwargs
',
mi
',
locals
',
val
',
pattern
',
replacement
):
```

```
return
```

```
re
.
sub
(
pattern
',
replacement
',
val
',
flags
=
re
.
I
```

```
)
```

```
\[docs\]
```

```
class
```

```
BuiltinReGroup
```

```
(
```

```
BuiltinFormatterFunction
```

```
):
```

```
name
```

```
=
```

```
're_group'
```

```
arg_count
```

```
=
```

```
-
```

```
1
```

```
category
```

```
=
```

```
'String manipulation'
```

```
\_\_doc\_\_
```

```
=
```

```
doc
```

```
=
```

```
-
```

```
(  
're_group(val, pattern [, template_for_group]*) -- '  
  
'return a string made by applying the regular expression pattern '  
  
'to the val and replacing each matched instance with the string '  
  
'computed by replacing each matched group by the value returned '  
  
'by the corresponding template. The original matched value for the '  
  
'group is available as $. In template program mode, like for '  
  
'the template and the eval functions, you use [[ for { and ]] for }.'  
  
' The following example in template program mode looks for series '  
  
'with more than one word and uppercases the first word: '  
  
"{series:'re_group($,  
\"  
(  
\\  
S* )(.*)  
\"  
,  
\"  
[[$:uppercase()]]  
\"  
,  
\"  
[[ $]]  
\"  
)\"}"
```



```
)

def
evaluate
(
self
,
formatter
,
kwargs
,
mi
,
locals
,
val
,
pattern
,
*
args
):

from
calibre.utils.formatter

import
EvalFormatter
```

```
def
```

```
repl
```

```
(
```

```
mo
```

```
):
```

```
res
```

```
=
```

```
"
```

```
if
```

```
mo
```

```
and
```

```
mo
```

```
.
```

```
lastindex
```

```
:
```

```
for
```

```
dex
```

```
in
```

```
range
```

```
(
```

```
0
```

```
,
```

```
mo
```

```
.
```

```
lastindex
```

```
):
```

```
gv
```

```
=
```

```
mo
```

```
.
```

```
group
```

```
(
```

```
dex
```

```
+
```

```
1
```

```
)
```

```
if
```

```
gv
```

```
is
```

```
None
```

```
:
```

```
continue
```

```
if
```

```
len
```

```
(
```

```
args
```

```
)
```

```
>
```

```
dex
```

```
:
```

```
template
```

```
=
```

```
args
```

```
[
```

```
  dex
```

```
]
```

```
.
```

```
replace
```

```
(
```

```
  '['
```

```
,
```

```
  '{'
```

```
)
```

```
.
```

```
replace
```

```
(
```

```
  '']'
```

```
,
```

```
  '}'
```

```
)
```

```
EvalFormatter
```

```
()
```

```
.
```

```
safe_format
```

```
(
```

```
  template
```

```
,
```

```
  {
```

```
    '$'
```

```
  :
```

```
gv
```

```
},
```

```
'EVAL'
```

```
,
```

```
None
```

```
,
```

```
strip_results
```

```
=
```

```
False
```

```
)
```

```
else
```

```
:
```

```
res
```

```
+=
```

```
gv
```

```
return
```

```
res
```

```
return
```

```
re
```

```
.
```

```
sub
```

```
(
```

```
pattern
```

```
,
```

```
repl
```

```
,
```

```
val
```

```
,
```

```
flags
```

```
=
```

```
re
```

```
.
```

```
|
```

```
)
```

```
\[docs\]
```

```
class
```

```
BuiltinSwapAroundComma
```

```
(
```

```
BuiltinFormatterFunction
```

```
):
```

```
name
```

```
=
```

```
'swap_around_comma'
```

```
arg_count
```

```
=
```

```
1
```

```
category
```

```
=
```

```
'String manipulation'
```

```
__doc__
```

```
=
```

doc

=

—
(

'swap_around_comma(val) -- given a value of the form '

""B, A", return "A B". This is most useful for converting names '

'in LN, FN format to FN LN. If there is no comma, the function '

'returns val unchanged'

)

def

evaluate

(

self

,

formatter

,

kwargs

,

mi

,

locals

,

val

):

return

re

.

sub

(

r

'^(.*?),\s*(.*\$)'

,

r

'\2 \1'

,

val

,

flags

=

re

.

|

)

.

strip

()

[\[docs\]](#)

class

BuiltinIfempty

(

BuiltinFormatterFunction

):

name

=

'ifempty'


```
arg_count
```

```
=
```

```
2
```

```
category
```

```
=
```

```
'If-then-else'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
(
```

```
'ifempty(val, text if empty) -- return val if val is not empty, '
```

```
'otherwise return `text if empty`'
```

```
)
```

```
def
```

```
evaluate
```

```
(
```

```
self
```

```
,
```

```
formatter
```

```
,
```

```
kwargs
',
mi
',
locals
',
val
',
value_if_empty
):
```

```
if
```

```
val
:
```

```
return
```

```
val
```

```
else
:
```

```
return
```

```
value_if_empty
```

```
\[docs\]
```

```
class
```

```
BuiltinShorten
```

```
(
BuiltinFormatterFunction
```

```
):
```

```
name
```

```
=
```

```
'shorten'
```

```
arg_count
```

```
=
```

```
4
```

```
category
```

```
=
```

```
'String manipulation'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-  
(
```

```
'shorten(val, left chars, middle text, right chars) -- Return a '
```

```
'shortened version of val, consisting of `left chars` '
```

```
'characters from the beginning of val, followed by '
```

`middle text`, followed by `right chars` characters from '

'the end of the string. `Left chars` and `right chars` must be '

'integers. For example, assume the title of the book is '

'`Ancient English Laws in the Times of Ivanhoe`, and you want '

'it to fit in a space of at most 15 characters. If you use '

'`{title:shorten(9,-,5)}`, the result will be `Ancient E-anhoe`.'

'If the field

'\n

s length is less than left chars + right chars + '

'the length of `middle text`, then the field will be used '

'intact. For example, the title `The Dome` would not be changed.'

)

def

evaluate

(

self

,

formatter

,

kwargs

,

```
mi
',

locals
',

val
',

leading
',

center_string
',

trailing
):

|

=

max
(
0
',

int
(
leading
))

t

=

max
(
0
```

```
,  
  
int  
(  
trailing  
)  
  
if  
  
len  
(  
val  
)  
  
>  
  
|  
  
+  
  
len  
(  
center_string  
)  
  
+  
  
t  
:  
  
return  
  
val  
[  
0  
:  
|  
]  
  
+
```

center_string

+

(
"

if

t

==

0

else

val

[

-

t

:])

else

:

return

val

[\[docs\]](#)

class

BuiltinCount

(

BuiltinFormatterFunction

):

name

```
=
```

```
'count'
```

```
arg_count
```

```
=
```

```
2
```

```
category
```

```
=
```

```
'List manipulation'
```

```
aliases
```

```
=
```

```
[  
'list_count'  
]
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-  
(
```

```
'count(val, separator) -- interprets the value as a list of items '
```


'separated by `separator`, returning the number of items in the '

'list. Most lists use a comma as the separator, but authors '

'uses an ampersand. Examples: {tags:count(,)}, {authors:count(&)}. '

'Aliases: count(), list_count()'

)

def

evaluate

(

self

,

formatter

,

kwargs

,

mi

,

locals

,

val

,

sep

):

return

str

```
(  
len  
([  
v  
  
for  
  
v  
  
in  
  
val  
.  
split  
(  
sep  
)  
  
if  
  
v  
]))
```

[\[docs\]](#)
class

BuiltinListCountMatching
(
BuiltinFormatterFunction
):

name

=

'list_count_matching'

arg_count

=

3

category

=

'List manipulation'

aliases

=

```
[  
'count_matching'  
]
```

__doc__

=

doc

=

```
(  
'list_count_matching(list, pattern, separator) -- '
```

"interprets 'list' as a list of items separated by 'separator', "

'returning the number of items in the list that match the regular '

"expression 'pattern'. Aliases: list_count_matching(), count_matching()"
)

```
def
```

```
    evaluate
```

```
    (
```

```
    self
```

```
    ,
```

```
    formatter
```

```
    ,
```

```
    kwargs
```

```
    ,
```

```
    mi
```

```
    ,
```

```
    locals
```

```
    ,
```

```
    list_
```

```
    ,
```

```
    pattern
```

```
    ,
```

```
    sep
```

```
):
```

```
    res
```

```
    =
```

```
    0
```

```
    for
```

```
    v
```

```
    in
```

```
[
x
.
strip
()

for

x

in

list_
.
split
(
sep
)

if

x
.
strip
()]:

if

re
.
search
(
pattern
,
v
,
flags
=
re
```

```
.  
|  
):  
  
res  
  
+=  
  
1  
  
return  
  
str  
(  
res  
)  
  
\[docs\]  
class  
  
BuiltinListitem  
(  
    BuiltinFormatterFunction  
):  
  
    name  
  
    =  
  
    'list_item'  
  
    arg_count  
  
    =  
  
    3
```

```

category

=

'List lookup'

__doc__

=

doc

=

-
(
'list_item(val, index, separator) -- interpret the value as a list of '

'items separated by `separator`, returning the `index`th item. '

'The first item is number zero. The last item can be returned '

'using `list_item(-1,separator)`. If the item is not in the list, '

'then the empty value is returned. The separator has the same '

'meaning as in the count function.'
)

def

evaluate
(
self
,

```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
```

```
,
```

```
locals
```

```
,
```

```
val
```

```
,
```

```
index
```

```
,
```

```
sep
```

```
):
```

```
if
```

```
not
```

```
val
```

```
:
```

```
return
```

```
''
```

```
index
```

```
=
```

```
int
```

```
(
```

```
index
```



```
)
```

```
val
```

```
=
```

```
val
```

```
.
```

```
split
```

```
(
```

```
sep
```

```
)
```

```
try
```

```
:
```

```
return
```

```
val
```

```
[
```

```
index
```

```
]
```

```
.
```

```
strip
```

```
()
```

```
except
```

```
:
```

```
return
```

```
""
```

```
\[docs\]
```

```
class
```

```
BuiltinSelect
```

```
(
  BuiltinFormatterFunction
):

name

=

'select'

arg_count

=

2

category

=

'List lookup'

__doc__

=

doc

=

-
(
'select(val, key) -- interpret the value as a comma-separated list '

'of items, with the items being "id:value". Find the pair with the '

'id equal to key, and return the corresponding value. Returns the '
```

```
'empty string if no match is found.'
```

```
)
```

```
def
```

```
    evaluate
```

```
    (
```

```
    self
```

```
,
```

```
    formatter
```

```
,
```

```
    kwargs
```

```
,
```

```
    mi
```

```
,
```

```
    locals
```

```
,
```

```
    val
```

```
,
```

```
    key
```

```
    );
```

```
if
```

```
not
```

```
    val
```

```
    :
```

```
return
```

```
"
```

```
vals
```

```
=
```

```
[
```

```
v
```

```
.
```

```
strip
```

```
()
```

```
for
```

```
v
```

```
in
```

```
val
```

```
.
```

```
split
```

```
(
```

```
' '
```

```
,
```

```
)]
```

```
tkey
```

```
=
```

```
key
```

```
+
```

```
'.'
```

```
]
```

```
for
```

```
v
```

```
in
```

```
vals
```

```
:
```

```
if
```

```
v
```

```
.
```

```
startswith
```

```
(
```

```
tkey
```

```
):
```

```
return
```

```
v
```

```
[
```

```
len
```

```
(
```

```
tkey
```

```
);]
```

```
return
```

```
"
```

```
\[docs\]
```

```
class
```

```
BuiltinApproximateFormats
```

```
(
```

```
BuiltinFormatterFunction
```

```
):
```

```
name
```

```
=
```

```
'approximate_formats'

arg_count

=

0

category

=

'Get values from metadata'

__doc__

=

doc

=

-
(
'approximate_formats() -- return a comma-separated '

'list of formats that at one point were associated with the '

'book. There is no guarantee that this list is correct, '

'although it probably is. '

'This function can be called in template program mode using '

'the template "{:
```

```
\approximate_formats()
\} ". '
```

'Note that format names are always uppercase, as in EPUB.'

'This function works only in the GUI. If you want to use these values '

'in save-to-disk or send-to-device templates then you '

'must make a custom "Column built from other columns", use '

'the function in that column

's template, and use that '

'column

's value in your save/send templates'

)

def

evaluate

```
(
self
,
```

```
formatter
```

,

kwargs

```
,  
  
mi  
,  
  
locals  
):  
  
if  
  
hasattr  
(  
mi  
,  
  
'_proxy_metadata'  
):  
  
fmt_data  
  
=  
  
mi  
.  
_proxy_metadata  
.  
db_approx_formats  
  
if  
  
not  
  
fmt_data  
:  
  
return  
  
"
```



```
data
```

```
=
```

```
sorted
```

```
(
```

```
fmt_data
```

```
)
```

```
return
```

```
','
```

```
.
```

```
join
```

```
(
```

```
v
```

```
.
```

```
upper
```

```
()
```

```
for
```

```
v
```

```
in
```

```
data
```

```
)
```

```
return
```

```
—
```

```
(
```

```
'This function can be used only in the GUI'
```

```
)
```

```
\[docs\]
```

```
class
```

BuiltinFormatsModtimes

```
(  
BuiltinFormatterFunction  
):
```

```
name
```

```
=
```

```
'formats_modtimes'
```

```
arg_count
```

```
=
```

```
1
```

```
category
```

```
=
```

```
'Get values from metadata'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-  
(
```

```
'formats_modtimes(date_format) -- return a comma-separated '
```

```
'list of colon-separated items representing modification times '
```

'for the formats of a book. The `date_format` parameter '

'specifies how the date is to be formatted. See the '

'`format_date` function for details. You can use the select '

'function to get the mod time for a specific '

'format. Note that format names are always uppercase, '

'as in EPUB.'

)

def

evaluate

(

self

,

formatter

,

kwargs

,

mi

,

locals

,

fmt

):

```
fmt_data

=

mi
.
get
(
'format_metadata'
,

{})

try
:

data

=

sorted
(
fmt_data
.
items
(),

key
=
lambda

x
:
x
[
1
][
'mtime'
],
```

```
reverse
=
True
)

return

','
.
join
(
k
.
upper
()
+
','
+
format_date
(
v
[
'mtime'
],

fmt
)

for
k
,
v
in
data
)
```

```
except
:

return

"""

\[docs\]
class

    BuiltinFormatsSizes
    (
    BuiltinFormatterFunction
    ):

    name

    =

    'formats_sizes'

    arg_count

    =

    0

    category

    =

    'Get values from metadata'

    __doc__

    =
```

```
doc
=
-
(
'formats_sizes() -- return a comma-separated list of '

'colon-separated items representing sizes in bytes '

'of the formats of a book. You can use the select '

'function to get the size for a specific '

'format. Note that format names are always uppercase, '

'as in EPUB.'

)
```

```
def
```

```
evaluate
```

```
(
self
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
```

```
,
```

```
locals
):

fmt_data

=

mi
.
get
(
'format_metadata'
,
{})

try
:

return

','
.
join
(
k
.
upper
()
+
':.'
+
str
(
v
[
'size'
])
```


for

k

,

v

in

iteritems

(

fmt_data

)

except

:

return

"

[\[docs\]](#)

class

BuiltinFormatsPaths

(

BuiltinFormatterFunction

):

name

=

'formats_paths'

arg_count

=

0

category

=

'Get values from metadata'

__doc__

=

doc

=

-

(

'formats_paths() -- return a comma-separated list of '

'colon-separated items representing full path to '

'the formats of a book. You can use the select '

'function to get the path for a specific '

'format. Note that format names are always uppercase, '

'as in EPUB.'

)

def

evaluate

```
(
self
,

formatter
,

kwargs
,

mi
,

locals
):

fmt_data

=

mi
.
get
(
'format_metadata'
,

{})

try
:

return

''
,

.
join
(
k
.

```

```
upper
```

```
()
```

```
+
```

```
':.'
```

```
+
```

```
str
```

```
(
```

```
v
```

```
[
```

```
'path'
```

```
])
```

```
for
```

```
k
```

```
,
```

```
v
```

```
in
```

```
iteritems
```

```
(
```

```
fmt_data
```

```
))
```

```
except
```

```
:
```

```
return
```

```
''
```

```
\[docs\]
```

```
class
```

```
BuiltinHumanReadable
```

```
(
```

```
BuiltinFormatterFunction
```

```
):
```

```
name
```

```
=
```

```
'human_readable'
```

```
arg_count
```

```
=
```

```
1
```

```
category
```

```
=
```

```
'Formatting values'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-
```

```
(
```

```
'human_readable(v) -- return a string '
```

```
'representing the number v in KB, MB, GB, etc.'
```

```
)
```

```
def
```

```
evaluate
(
    self
    ,
    formatter
    ,
    kwargs
    ,
    mi
    ,
    locals
    ,
    val
):

    try
    :

        return

    human_readable
    (
        round
        (
            float
            (
                val
            ))
    )

    except
    :

    return
```

"

[\[docs\]](#)

class

BuiltinFormatNumber

(
BuiltinFormatterFunction
):

name

=

'format_number'

arg_count

=

2

category

=

'Formatting values'

__doc__

=

doc

=

_

```
(
'format_number(v, template) -- format the number v using '

'a Python formatting template such as "
{0:5.2f}
" or '

'''
{0:.,d}
" or "$
{0:5,.2f}
". The field_name part of the '

'template must be a 0 (zero) (the "{0:" in the above examples). '

'See the template language and Python documentation for more '

'examples. You can leave off the leading "{0:" and trailing '

''' if the template contains only a format. Returns the empty '

'string if formatting fails.'

)

def
evaluate
(
self
,
formatter
```



```
,  
  
kwargs  
,  
  
mi  
,  
  
locals  
,  
  
val  
,  
  
template  
):
```

```
if
```

```
val
```

```
==
```

```
"
```

```
or
```

```
val
```

```
==
```

```
'None'
```

```
:
```

```
return
```

```
"
```

```
if
```

```
{
```

```
not
```

```
in
```

```
template
```

```
:
```

```
template
```

```
=
```

```
{0:
```

```
+
```

```
template
```

```
+
```

```
}
```

```
try
```

```
:
```

```
v1
```

```
=
```

```
float
```

```
(
```

```
val
```

```
)
```

```
except
```

```
:
```

```
return
```

```
"
```

```
try
```

```
:
```

```
# Try formatting the value as a float
```

```
return
```

```
template
```

```
.
```

```
format
```

```
(
```

```
v1
```

```
)
```

```
except
```

```
:
```

```
pass
```

```
try
```

```
:
```

```
# Try formatting the value as an int
```

```
v2
```

```
=
```

```
trunc
```

```
(
```

```
v1
```

```
)
```

if

v2

==

v1

:

return

template

.

format

(

v2

)

except

:

pass

return

""

[\[docs\]](#)

class

BuiltinSublist

(

BuiltinFormatterFunction

):

name

=

'sublist'

arg_count

=

4

category

=

'List manipulation'

__doc__

=

doc

=

—

(

'sublist(val, start_index, end_index, separator) -- interpret the '

'value as a list of items separated by `separator`, returning a '

'new list made from the `start\_index` to the `end\_index` item. '

'The first item is number zero. If an index is negative, then it '

'counts from the end of the list. As a special case, an end_index '

'of zero is assumed to be the length of the list. Examples using '

'basic template mode and assuming that the tags column (which is '

'comma-separated) contains "A, B, C": '

```
{tags:sublist(0,1,  
\\  
,)} returns "A". '
```

```
{tags:sublist(-1,0,  
\\  
,)} returns "C". '
```

```
{tags:sublist(0,-1,  
\\  
,)} returns "A, B". '
```

)

def

evaluate

(
self

,

formatter

,

kwargs

,

```
mi
,

locals
,

val
,

start_index
,

end_index
,

sep
):

if
not

val
:

return

"

si

=

int
(
start_index
)

ei
```

```
=
```

```
int  
(  
    end_index  
)
```

```
# allow empty list items so counts are what the user expects
```

```
val
```

```
=
```

```
[  
    v  
    .  
    strip  
    ()
```

```
for
```

```
v
```

```
in
```

```
val  
    .  
    split  
    (  
        sep  
    )]
```

```
if
```

```
sep
```

```
==
```



```
','  
,  
:
```

```
sep
```

```
=
```

```
','  
,
```

```
try  
:
```

```
if
```

```
ei
```

```
==
```

```
0  
:
```

```
return
```

```
sep
```

```
.  
join  
(  
val  
[  
si  
:])
```

```
else  
:
```

```
return
```

```
sep
```

```
.
```

```
join
```

```
(
```

```
val
```

```
[
```

```
si
```

```
:
```

```
ei
```

```
])
```

```
except
```

```
:
```

```
return
```

```
''
```

```
\[docs\]
```

```
class
```

```
BuiltinSubitems
```

```
(
```

```
BuiltinFormatterFunction
```

```
):
```

```
name
```

```
=
```

```
'subitems'
```

```
arg_count
```

```
=
```

```
3
```

category

=

'List manipulation'

__doc__

=

doc

=

(

'subitems(val, start_index, end_index) -- This function is used to '

'break apart lists of items such as genres. It interprets the value '

'as a comma-separated list of items, where each item is a period-'

'separated list. Returns a new list made by first finding all the '

'period-separated items, then for each such item extracting the '

'`start\_index` to the `end\_index` components, then combining '

'the results back together. The first component in a period-'

'separated list has an index of zero. If an index is negative, '

'then it counts from the end of the list. As a special case, an '

'end_index of zero is assumed to be the length of the list. '

'Example using basic template mode and assuming a #genre value of '

""A.B.C": {#genre:subitems(0,1)} returns "A". {#genre:subitems(0,2)} '

'returns "A.B". {#genre:subitems(1,0)} returns "B.C". Assuming a #genre '

'value of "A.B.C, D.E.F", {#genre:subitems(0,1)} returns "A, D". '

'{#genre:subitems(0,2)} returns "A.B, D.E"
)

period_pattern

=

re

.

compile

(

r

'(?<=[^\s])\.(?=[^\s])'

,

re

.

U

)

def

```
evaluate
(
self
,
formatter
,
kwargs
,
mi
,
locals
,
val
,
start_index
,
end_index
):
```

```
if
```

```
not
```

```
val
:
```

```
return
```

```
"
```

```
si
```

```
=
```

```
int  
(  
start_index  
)
```

```
ei
```

```
=
```

```
int  
(  
end_index  
)
```

```
has_periods
```

```
=
```

```
':'
```

```
in
```

```
val
```

```
items
```

```
=
```

```
[  
v  
.  
strip  
()]
```

```
for
```

```
v
```

```
in
```

```
val
.
split
(
    ''
)

if

v
.
strip
()]

rv

=

set
()

for

item

in

items
:

if

has_periods

and

''

in
```

```
item
:

components

=

self
.
period_pattern
.
split
(
item
)
```

```
else
:

components

=

[
item
]
```

```
try
:
```

```
if
```

```
ei
```

```
==
```

```
0
```



```
:  
  
t  
  
=  
  
'.'  
.  
join  
(  
components  
[  
si  
:])  
.  
strip  
()
```

```
else  
:  
  
t  
  
=  
  
'.'  
.  
join  
(  
components  
[  
si  
:  
ei  
])  
.  
strip  
()
```

```
if
```

```
t
```

```
:
```

```
rv
```

```
.
```

```
add
```

```
(
```

```
t
```

```
)
```

```
except
```

```
:
```

```
pass
```

```
return
```

```
' '
```

```
,
```

```
.
```

```
join
```

```
(
```

```
sorted
```

```
(
```

```
rv
```

```
,
```

```
key
```

```
=
```

```
sort_key
```

```
))
```

```
\[docs\]
```

```
class
```

```
BuiltinFormatDate
```

```
(
```

```
BuiltinFormatterFunction
```

```
):
```

```
name
```

```
=
```

```
'format_date'
```

```
arg_count
```

```
=
```

```
2
```

```
category
```

```
=
```

```
'Formatting values'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-  
(
```

```
'format_date(val, format_string) -- format the value, '
```

```
'which must be a date, using the format_string, returning a string. '
```

```
'The formatting codes are: '
```

'd : the day as number without a leading zero (1 to 31) '

'dd : the day as number with a leading zero (01 to 31) '

'ddd : the abbreviated localized day name (e.g. "Mon" to "Sun"). '

'dddd : the long localized day name (e.g. "Monday" to "Sunday"). '

'M : the month as number without a leading zero (1 to 12). '

'MM : the month as number with a leading zero (01 to 12) '

'MMM : the abbreviated localized month name (e.g. "Jan" to "Dec"). '

'MMMM : the long localized month name (e.g. "January" to "December"). '

'yy : the year as two digit number (00 to 99). '

'yyyy : the year as four digit number. '

'h : the hours without a leading 0 (0 to 11 or 0 to 23, depending on am/pm) '

'hh : the hours with a leading 0 (00 to 11 or 00 to 23, depending on am/pm) '

'm : the minutes without a leading 0 (0 to 59) '

'mm : the minutes with a leading 0 (00 to 59) '

's' : the seconds without a leading 0 (0 to 59) '

'ss' : the seconds with a leading 0 (00 to 59) '

'ap' : use a 12-hour clock instead of a 24-hour clock, with "ap" replaced by the localized string for am or pm '

'AP' : use a 12-hour clock instead of a 24-hour clock, with "AP" replaced by the localized string for AM or PM '

'iso' : the date with time and timezone. Must be the only format present '

'to_number': the date as a floating point number '

'from_number[:fmt]': format the timestamp using fmt if present otherwise iso'
)

def

evaluate

(

self

,

formatter

,

kwargs

,

mi

,

locals

,

```
val
,  
format_string  
):
```

```
if
```

```
not
```

```
val
```

```
or
```

```
val
```

```
==
```

```
'None'
```

```
:
```

```
return
```

```
"
```

```
try
```

```
:
```

```
if
```

```
format_string
```

```
==
```

```
'to_number'
```

```
:
```

```
s
=
parse_date
(
val
)
.
timestamp
()

elif

format_string
.
startswith
(
'from_number'
):

val

=

datetime
.
fromtimestamp
(
float
(
val
))

f

=

format_string
[
```

```
12
:]

s

=

format_date
(
val
,
f
if
f
else
'iso'
)

else
:

s

=

format_date
(
parse_date
(
val
),
format_string
)
```


return

s

except

:

s

=

'BAD DATE'

return

s

[\[docs\]](#)

class

BuiltinUppercase

(
BuiltinFormatterFunction
):

name

=

'uppercase'

arg_count

=

1

```
category
```

```
=
```

```
'String case changes'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-  
(
```

```
'uppercase(val) -- return val in upper case'  
)
```

```
def
```

```
evaluate
```

```
(
```

```
self
```

```
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
```

```
,
```

```
locals
```

```
,
```

```
val  
):
```

```
return
```

```
val  
.  
upper  
()
```

```
\[docs\]
```

```
class
```

```
BuiltinLowercase
```

```
(  
BuiltinFormatterFunction  
):
```

```
name
```

```
=
```

```
'lowercase'
```

```
arg_count
```

```
=
```

```
1
```

```
category
```

```
=
```

```
'String case changes'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-  
(  
  'lowercase(val) -- return val in lower case'  
)
```

```
def
```

```
  evaluate
```

```
  (  
    self
```

```
  ,
```

```
  formatter
```

```
  ,
```

```
  kwargs
```

```
  ,
```

```
  mi
```

```
  ,
```

```
  locals
```

```
  ,
```

```
  val
```

```
  ):
```

```
return
```

```
  val
```

```
  .
```

```
  lower
```

```
()
```

```
\[docs\]
```

```
class
```

```
BuiltinTitlecase
```

```
(  
    BuiltinFormatterFunction  
):
```

```
name
```

```
=
```

```
'titlecase'
```

```
arg_count
```

```
=
```

```
1
```

```
category
```

```
=
```

```
'String case changes'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-  
(
```

```
'titlecase(val) -- return val in title case'  
)
```

```
def
```

```
evaluate
```

```
(  
  self
```

```
,
```

```
  formatter
```

```
,
```

```
  kwargs
```

```
,
```

```
  mi
```

```
,
```

```
  locals
```

```
,
```

```
  val
```

```
):
```

```
return
```

```
titlecase
```

```
(  
  val  
)
```

```
\[docs\]
```

```
class
```

```
BuiltinCapitalize
```

```
(  
  BuiltinFormatterFunction  
)
```

```
name
```

```
=
```

```
'capitalize'
```

```
arg_count
```

```
=
```

```
1
```

```
category
```

```
=
```

```
'String case changes'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-
```

```
(  
'capitalize(val) -- return val capitalized'  
)
```

```
def
```

```
evaluate
```

```
(
```

```
self
```

```
,  
  
formatter  
,  
  
kwargs  
,  
  
mi  
,  
  
locals  
,  
  
val  
):  
  
return  
  
capitalize  
(  
val  
)  
  
\[docs\]  
class  
  
BuiltinBooksize  
(  
BuiltinFormatterFunction  
):  
  
name  
  
=  
  
'booksize'  
  
arg_count
```


=

0

category

=

'Get values from metadata'

__doc__

=

doc

=

(

'booksize() -- return value of the size field. '

'This function works only in the GUI. If you want to use this value '

'in save-to-disk or send-to-device templates then you '

'must make a custom "Column built from other columns", use '

'the function in that column

\'

s template, and use that '

'column

\'

s value in your save/send templates'

```
)

def
evaluate
(
self
,
formatter
,
kwargs
,
mi
,
locals
):

if
hasattr
(
mi
,
'_proxy_metadata'
):

try
:

v
=
```

```
mi
    .
    _proxy_metadata
    .
    book_size
```

```
if
```

```
v
```

```
is
```

```
not
```

```
None
```

```
:
```

```
return
```

```
str
```

```
(
```

```
mi
```

```
.
```

```
_proxy_metadata
```

```
.
```

```
book_size
```

```
)
```

```
return
```

```
''
```

```
except
```

```
:
```

```
pass
```

```
return
```

```
''
```

```
return
```

```
—  
(
```

```
'This function can be used only in the GUI'
```

```
)
```

```
\[docs\]
```

```
class
```

```
BuiltinOndevice
```

```
(
```

```
BuiltinFormatterFunction
```

```
):
```

```
name
```

```
=
```

```
'ondevice'
```

```
arg_count
```

```
=
```

```
0
```

```
category
```

```
=
```

```
'Get values from metadata'
```

```

__doc__

=

doc

=

-
(
'ondevice() -- return Yes if ondevice is set, otherwise return '

'the empty string. This function works only in the GUI. If you want to '

'use this value in save-to-disk or send-to-device templates then you '

'must make a custom "Column built from other columns", use '

'the function in that column
\'
s template, and use that '

'column
\'
s value in your save/send templates'
)

```

```

def

evaluate
(
self
,

formatter
,

```

```
kwargs
,

mi
,

locals
):

if

hasattr
(
mi
,

'_proxy_metadata'
):

if

mi
.
_proxy_metadata
.
ondevice_col
:

return

_
(
'Yes'
)

return

"
```

return

```
—  
(  
'This function can be used only in the GUI'  
)
```

[\[docs\]](#)

class

BuiltinAnnotationCount

```
(  
BuiltinFormatterFunction  
):
```

name

=

'annotation_count'

arg_count

=

0

category

=

'Get values from metadata'

__doc__

=

```
doc
=
-
(
'annotation_count() -- return the total number of annotations '

'of all types attached to the current book. '

'This function works only in the GUI.'
)
```

```
def
```

```
evaluate
```

```
(
self
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
```

```
,
```

```
locals
```

```
):
```

```
with
```

```
suppress
```

```
(
```

```
Exception
```



```
):  
  
from  
calibre.gui2.ui  
  
import  
get_gui  
  
c  
  
=  
  
get_gui  
(  
.  
current_db  
.  
new_api  
.  
annotation_count_for_book  
(  
mi  
.  
id  
)  
  
return  
  
"  
  
if  
  
c  
  
==  
  
0
```

else

```
str  
(  
c  
)
```

return

```
—  
(  
'This function can be used only in the GUI'  
)
```

[\[docs\]](#)

class

BuiltinIsMarked

```
(  
BuiltinFormatterFunction  
):
```

name

=

'is_marked'

arg_count

=

0

category

=

```
'Get values from metadata'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-
```

```
(  
"is_marked() -- check whether the book is 'marked' in "
```

```
"calibre. If it is then return the value of the mark, "
```

```
"either 'true' or the comma-separated list of named "
```

```
"marks. Returns " if the book is not marked."  
)
```

```
def
```

```
evaluate
```

```
(
```

```
self
```

```
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
```

```
,
```

```
locals
):

with

suppress
(
Exception
):

from

calibre.gui2.ui

import

get_gui

c

=

get_gui
()
.
current_db
.
data
.
get_marked
(
mi
.
id
)

return

c
```

```
if
```

```
c
```

```
else
```

```
"
```

```
return
```

```
—  
(
```

```
'This function can be used only in the GUI'
```

```
)
```

```
\[docs\]
```

```
class
```

```
BuiltinSeriesSort
```

```
(
```

```
BuiltinFormatterFunction
```

```
):
```

```
name
```

```
=
```

```
'series_sort'
```

```
arg_count
```

```
=
```

```
0
```

```
category
```

```
=
```

```
'Get values from metadata'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-  
(  
'series_sort() -- return the series sort value'  
)
```

```
def
```

```
evaluate
```

```
(  
self
```

```
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
```

```
,
```

```
locals
```

```
):
```

```
if
```

```
mi
```

```
.
series
:

return

title_sort
(
mi
.
series
)

return

''

\[docs\]
class

BuiltinHasCover
(
BuiltinFormatterFunction
):

name

=

'has_cover'

arg_count

=

0
```

```

category

=

'Get values from metadata'

__doc__

=

doc

=

-
(
'has_cover() -- return Yes if the book has a cover, '

'otherwise return the empty string'
)

def

evaluate
(
self
,

formatter
,

kwargs
,

mi
,

locals
):

```



```
if
```

```
mi
```

```
.  
has_cover  
:
```

```
return
```

```
_  
(  
'Yes'  
)
```

```
return
```

```
"
```

```
\[docs\]
```

```
class
```

```
BuiltinFirstNonEmpty
```

```
(  
BuiltinFormatterFunction  
):
```

```
name
```

```
=
```

```
'first_non_empty'
```

```
arg_count
```

```
=
```

```
-
1

category

=

'Iterating over values'

__doc__

=

doc

=

-
(
'first_non_empty(value [, value]*) -- '

'returns the first value that is not empty. If all values are '

'empty, then the empty string is returned. '

'You can have as many values as you want.'
)

def

evaluate
(
self
,
formatter
```

```
,  
  
kwargs  
,  
  
mi  
,  
  
locals  
,  
  
*  
args  
):  
  
i  
  
=  
  
0  
  
while  
  
i  
  
<  
  
len  
(  
args  
):  
  
if  
  
args  
[  
i  
]:
```

```
return
```

```
args
```

```
[
```

```
i
```

```
]
```

```
i
```

```
+=
```

```
1
```

```
return
```

```
"
```

```
\[docs\]
```

```
class
```

```
BuiltinAnd
```

```
(
```

```
BuiltinFormatterFunction
```

```
):
```

```
name
```

```
=
```

```
'and'
```

```
arg_count
```

```
=
```

```
-
```

```
1
```

```
category
```

```
=
```

```
'Boolean'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
(
```

```
'and(value [, value]*) -- '
```

```
'returns the string "1" if all values are not empty, otherwise '
```

```
'returns the empty string. This function works well with test or '
```

```
'first_non_empty. You can have as many values as you want. In many '
```

```
'cases the && operator can replace this function.'
```

```
)
```

```
def
```

```
evaluate
```

```
(
```

```
self
```

```
,
```

```
formatter
```

```
,  
  
kwargs  
,  
  
mi  
,  
  
locals  
,  
  
*  
args  
):  
  
i  
  
=  
  
0  
  
while  
  
i  
  
<  
  
len  
(  
args  
):  
  
if  
  
not  
  
args  
[  
i  
]:
```

return

"

i

+=

1

return

'1'

[\[docs\]](#)

class

BuiltinOr

(
BuiltinFormatterFunction
):

name

=

'or'

arg_count

=

-

1

```
category

=

'Boolean'

__doc__

=

doc

=

(
'or(value [, value]*) -- '

'returns the string "1" if any value is not empty, otherwise '

'returns the empty string. This function works well with test or '

'first_non_empty. You can have as many values as you want. In many '

'cases the || operator can replace this function.'
)

def

evaluate
(
self
,
formatter
,
```



```
kwargs
```

```
,
```

```
mi
```

```
,
```

```
locals
```

```
,
```

```
*
```

```
args
```

```
):
```

```
i
```

```
=
```

```
0
```

```
while
```

```
i
```

```
<
```

```
len
```

```
(
```

```
args
```

```
):
```

```
if
```

```
args
```

```
[
```

```
i
```

```
]:
```

```
return
```

'1'

i

+=

1

return

"

[\[docs\]](#)

class

BuiltinNot

(
BuiltinFormatterFunction
):

name

=

'not'

arg_count

=

1

category

=

```
'Boolean'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-
```

```
(
```

```
'not(value) -- '
```

```
'returns the string "1" if the value is empty, otherwise '
```

```
'returns the empty string. This function works well with test or '
```

```
'first_non_empty. In many cases the ! operator can replace this '
```

```
'function.'
```

```
)
```

```
def
```

```
evaluate
```

```
(
```

```
self
```

```
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
,

locals
,

val
):

return

"

if

val

else

'1'

\[docs\]
class

BuiltinListUnion
(
    BuiltinFormatterFunction
):

name

=

'list_union'

arg_count

=
```

3

category

=

'List manipulation'

__doc__

=

doc

=

—
(

'list_union(list1, list2, separator) -- '

'return a list made by merging the items in list1 and list2, '

'removing duplicate items using a case-insensitive comparison. If '

'items differ in case, the one in list1 is used. '

'The items in list1 and list2 are separated by separator, as are '

'the items in the returned list. Aliases: list_union(), merge_lists()'

)

aliases

=

```
[  
    'merge_lists'  
]
```

```
def
```

```
    evaluate
```

```
    (  
        self  
        ,
```

```
        formatter
```

```
        ,
```

```
        kwargs
```

```
        ,
```

```
        mi
```

```
        ,
```

```
        locals
```

```
        ,
```

```
        list1
```

```
        ,
```

```
        list2
```

```
        ,
```

```
        separator
```

```
    ):
```

```
        res
```

```
        =
```

```
        {
```

```
            icu_lower
```

```
            (  
                |
```

```
.
strip
()):

|
.
strip
()

for

|

in

list2

.
split
(
separator
)

if

|
.
strip
()

res

.
update
({
icu_lower
(
|
.
strip
()):

|
.

```

```
strip
()

for
|

in

list1
.
split
(
separator
)

if

|
.
strip
({})

if

separator

==

','
:

separator

=

','

return
```


separator

.

join

(

res

.

values

())

[\[docs\]](#)

class

BuiltinListRemoveDuplicates

(

BuiltinFormatterFunction

):

name

=

'list_remove_duplicates'

arg_count

=

2

category

=

'List manipulation'

__doc__

=

doc

=

—
(

'list_remove_duplicates(list, separator) -- '

'return a list made by removing duplicate items in the source list. '

'If items differ only in case, the last of them is returned. '

'The items in source list are separated by separator, as are '

'the items in the returned list.'

)

def

evaluate

(

self

,

formatter

,

kwargs

,

mi

,

locals

,

```
list_  
,  
  
separator  
):  
  
res  
  
=  
  
{  
icu_lower  
(  
|  
.  
strip  
()):  
  
|  
.  
strip  
()  
  
for  
  
|  
  
in  
  
list_  
.  
split  
(  
separator  
)  
  
if  
  
|  
.  
strip  
())}
```

if

separator

==

' '

,

:

separator

=

' '

,

return

separator

.

join

(

res

.

values

()))

[\[docs\]](#)

class

BuiltinListDifference

(

BuiltinFormatterFunction

):

name

=

```
'list_difference'
```

```
arg_count
```

```
=
```

```
3
```

```
category
```

```
=
```

```
'List manipulation'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-
```

```
(  
'list_difference(list1, list2, separator) -- '
```

```
'return a list made by removing from list1 any item found in list2, '
```

```
'using a case-insensitive comparison. The items in list1 and list2 '
```

```
'are separated by separator, as are the items in the returned list.'
```

```
)
```

```
def
```

```
evaluate
(
self
,
formatter
,
kwargs
,
mi
,
locals
,
list1
,
list2
,
separator
):
```

```
l1
```

```
=
```

```
[
|
.
strip
()
```

```
for
```

```
|
```

```
in
```

```
list1
.
split
(
separator
)
```

```
if
```

```
l
.
strip
()]
```

```
l2
```

```
=
```

```
{
icu_lower
(
l
.
strip
())
```

```
for
```

```
l
```

```
in
```

```
list2
```

```
.
split
(
separator
)
```

```
if
```

```
l
.
strip
()}

res

=

[]

for

i

in

l1
:

if

icu_lower
(
i
)

not

in

l2

and

i

not
```


in

res

:

res

.

append

(

i

)

if

separator

==

' '

,

:

return

' '

,

.

join

(

res

)

return

separator

.

join

(

res

)

[\[docs\]](#)

class

BuiltinListIntersection

(
BuiltinFormatterFunction
):

name

=

'list_intersection'

arg_count

=

3

category

=

'List manipulation'

__doc__

=

doc

=

—
(

'list_intersection(list1, list2, separator) -- '

'return a list made by removing from list1 any item not found in list2, '

'using a case-insensitive comparison. The items in list1 and list2 '

'are separated by separator, as are the items in the returned list.'

)

def

evaluate

(

self

,

formatter

,

kwargs

,

mi

,

locals

,

list1

,

list2

,

separator

):

l1

=

```
[
|
.
strip
()

for

|

in

list1

.
split
(
separator
)

if

|
.
strip
()]

l2

=

{
icu_lower
(
|
.
strip
())

for

|
```

in

list2

.

split

(

separator

)

if

|

.

strip

()}

res

=

[]

for

i

in

l1

:

if

icu_lower

(

i

)

in

l2

and

i

not

in

res

:

res

.

append

(

i

)

if

separator

==

' '

,

:

return

' '

,

.

join

(

res

)

return

separator

.

join

(

res

)

[\[docs\]](#)

class

BuiltinListSort

(

BuiltinFormatterFunction

):

name

=

'list_sort'

arg_count

=

3

category

=

'List manipulation'

__doc__

```
=  
  
doc  
  
=  
  
-  
(  
'list_sort(list, direction, separator) -- '  
  
'return list sorted using a case-insensitive sort. If direction is '  
  
'zero, the list is sorted ascending, otherwise descending. The list items '  
  
'are separated by separator, as are the items in the returned list.'  
)  
  
def  
  
evaluate  
(  
self  
,  
  
formatter  
,  
  
kwargs  
,  
  
mi  
,  
  
locals  
,  
  
list1  
,
```



```
direction
```

```
,
```

```
separator
```

```
):
```

```
res
```

```
=
```

```
[
```

```
|
```

```
.
```

```
strip
```

```
()
```

```
for
```

```
|
```

```
in
```

```
list1
```

```
.
```

```
split
```

```
(
```

```
separator
```

```
)
```

```
if
```

```
|
```

```
.
```

```
strip
```

```
()]
```

```
if
```

```
separator
```

```
==
```

```
','  
:
```

```
return
```

```
','
```

```
.
```

```
join
```

```
(
```

```
sorted
```

```
(
```

```
res
```

```
,
```

```
key
```

```
=
```

```
sort_key
```

```
,
```

```
reverse
```

```
=
```

```
direction
```

```
!=
```

```
"0"
```

```
))
```

```
return
```

```
separator
```

```
.
```

```
join
```

```
(
```

```
sorted
```

```
(
```

```
res
```

```
,
```

```
key
=
sort_key
,
```

```
reverse
=
direction
```

```
!=
```

```
"0"
))
```

```
\[docs\]
class
```

```
BuiltinListEquals
```

```
(
    BuiltinFormatterFunction
):
```

```
name
```

```
=
```

```
'list_equals'
```

```
arg_count
```

```
=
```

```
6
```

```
category
```

```
=
```

```
'List manipulation'
```

__doc__

=

doc

=

—
(

'list_equals(list1, sep1, list2, sep2, yes_val, no_val) -- '

'return yes_val if list1 and list2 contain the same items, '

'otherwise return no_val. The items are determined by splitting '

'each list using the appropriate separator character (sep1 or '

'sep2). The order of items in the lists is not relevant. '

'The comparison is case insensitive.'

)

def

evaluate

(

self

,

formatter

,

kwargs

```
,  
  
mi  
,  
  
locals  
,  
  
list1  
,  
  
sep1  
,  
  
list2  
,  
  
sep2  
,  
  
yes_val  
,  
  
no_val  
):  
  
s1  
  
=  
  
{  
icu_lower  
(  
|  
.  
strip  
())  
  
for  
  
|
```

in

list1

.

split

(

sep1

)

if

|

.

strip

()}

s2

=

{

icu_lower

(

|

.

strip

())

for

|

in

list2

.

split

(

sep2

)

if

```
|  
.  
strip  
{}
```

```
if
```

```
s1
```

```
==
```

```
s2
```

```
:
```

```
return
```

```
yes_val
```

```
return
```

```
no_val
```

```
\[docs\]
```

```
class
```

```
BuiltinListRe
```

```
(  
    BuiltinFormatterFunction  
):
```

```
name
```

```
=
```

```
'list_re'
```

```
arg_count
=
4

category
=
'List manipulation'

__doc__
=
doc
=
-
(
'list_re(src_list, separator, include_re, opt_replace) -- '

'Construct a list by first separating src_list into items using '

'the separator character. For each item in the list, check if it '

'matches include_re. If it does, then add it to the list to be '

'returned. If opt_replace is not the empty string, then apply the '

'replacement before adding the item to the returned list.'
)
```



```
def
evaluate
(
self
,
formatter
,
kwargs
,
mi
,
locals
,
src_list
,
separator
,
include_re
,
opt_replace
):

|

=

[
|
.
strip
()
```

```
for
```

```
|
```

```
in
```

```
src_list
```

```
.
```

```
split
```

```
(
```

```
separator
```

```
)
```

```
if
```

```
|
```

```
.
```

```
strip
```

```
()]
```

```
res
```

```
=
```

```
[]
```

```
for
```

```
item
```

```
in
```

```
|
```

```
:
```

```
if
```

```
re
```

```
.
```

```
search
```

```
(  
include_re
```

```
,
```

```
item
```

```
,
```

```
flags
```

```
=
```

```
re
```

```
.
```

```
|
```

```
)
```

```
is
```

```
not
```

```
None
```

```
:
```

```
if
```

```
opt_replace
```

```
:
```

```
item
```

```
=
```

```
re
```

```
.
```

```
sub
```

```
(
```

```
include_re
```

```
,
```

```
opt_replace
```

```
,
```

```
item
```

```
)  
  
for  
i  
in  
[  
t  
.  
strip  
()  
for  
t  
in  
item  
.  
split  
(  
separator  
)  
if  
t  
.  
strip  
()]:  
  
if  
i  
not  
in
```

```
res
:
```

```
res
.
append
(
i
)
```

```
if
```

```
separator
```

```
==
```

```
' '
,
:
```

```
return
```

```
' '
,
.
join
(
res
)
```

```
return
```

```
separator
```

```
.
join
(
res
)
```

[\[docs\]](#)

class

BuiltinListReGroup

(
BuiltinFormatterFunction
):

name

=

'list_re_group'

arg_count

=

-

1

category

=

'List manipulation'

__doc__

=

doc

=

-

(
'list_re_group(src_list, separator, include_re, search_re [, group_template]++) -- '

```
'Like list_re except replacements are not optional. It '
```

```
'uses re_group(list_item, search_re, group_template, ...) when '
```

```
'doing the replacements on the resulting list.'
```

```
)
```

```
def
```

```
    evaluate
```

```
    (
```

```
    self
```

```
    ,
```

```
    formatter
```

```
    ,
```

```
    kwargs
```

```
    ,
```

```
    mi
```

```
    ,
```

```
    locals
```

```
    ,
```

```
    src_list
```

```
    ,
```

```
    separator
```

```
    ,
```

```
    include_re
```

```
    ,
```

```
    search_re
```

```
    ,
```

```
*
args
):

from

calibre.utils.formatter

import

EvalFormatter

|

=

[
|
.
strip
()

for

|

in

src_list
.
split
(
separator
)

if

|

.
```



```
strip  
()]
```

```
res
```

```
=
```

```
[]
```

```
for
```

```
item
```

```
in
```

```
|
```

```
:
```

```
def
```

```
repl
```

```
(
```

```
mo
```

```
):
```

```
newval
```

```
=
```

```
"
```

```
if
```

```
mo
```

```
and
```

```
mo
```

```
.  
lastindex  
:
```

```
for
```

```
dex
```

```
in
```

```
range
```

```
(  
0
```

```
,
```

```
mo
```

```
.  
lastindex  
):
```

```
gv
```

```
=
```

```
mo
```

```
.  
group
```

```
(  
dex
```

```
+
```

```
1
```

```
)
```

```
if
```

```
gv
```

```
is
```

```
None
```

```
:
```

```
continue
```

```
if
```

```
len  
(  
args  
)
```

```
>
```

```
dex  
:
```

```
template
```

```
=
```

```
args  
[  
dex  
]  
.  
replace  
(  
'['  
,  
'{'  
)  
.  
replace  
(  
']'  
,  
'}'  
)
```

newval

+=

EvalFormatter

()

.

safe_format

(

template

,

{

'\$'

:

gv

},

'EVAL'

,

None

,

strip_results

=

False

)

else

:

newval

+=

gv

return

newval

if

re

.

search

(

include_re

,

item

,

flags

=

re

.

|

)

is

not

None

:

item

=

re

.

sub

(

search_re

```
,  
  
repl  
,  
  
item  
,  
  
flags  
=  
re  
.  
|  
)  
  
for  
  
i  
  
in  
  
[  
t  
.  
strip  
()  
  
for  
  
t  
  
in  
  
item  
.  
split  
(  
separator  
)  
  
if
```

```
t
.
strip
()]:

if

i

not

in

res
:

res
.
append
(
i
)

if

separator

==

','
:

return

','
.
join
(
res
```

```
)
```

```
return
```

```
separator
```

```
.
```

```
join
```

```
(
```

```
res
```

```
)
```

```
\[docs\]
```

```
class
```

```
BuiltinToday
```

```
(
```

```
BuiltinFormatterFunction
```

```
):
```

```
name
```

```
=
```

```
'today'
```

```
arg_count
```

```
=
```

```
0
```

```
category
```

```
=
```

```
'Date functions'
```



```

__doc__

=

doc

=

-
(
'today() -- '

'return a date string for today. This value is designed for use in '

'format_date or days_between, but can be manipulated like any '

'other string. The date is in ISO format.'
)

def
evaluate
(
self
,
formatter
,
kwargs
,
mi
,
locals
):

```

return

format_date

(
now
(),

'iso'
)

[\[docs\]](#)

class

BuiltinDaysBetween

(
BuiltinFormatterFunction
):

name

=

'days_between'

arg_count

=

2

category

=

'Date functions'

__doc__

```
=
```

```
doc
```

```
=
```

```
-  
(
```

```
'days_between(date1, date2) -- '
```

```
'return the number of days between date1 and date2. The number is '
```

```
'positive if date1 is greater than date2, otherwise negative. If '
```

```
'either date1 or date2 are not dates, the function returns the '
```

```
'empty string.'
```

```
)
```

```
def
```

```
evaluate
```

```
(
```

```
self
```

```
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
```

```
,
```

```
locals
```

```
,  
  
date1  
,  
  
date2  
):  
  
try  
:  
  
d1  
  
=  
  
parse_date  
(  
date1  
)  
  
if  
  
d1  
  
==  
  
UNDEFINED_DATE  
:  
  
return  
  
"  
  
d2  
  
=  
  
parse_date
```

```
(
date2
)

if

d2

==

UNDEFINED_DATE
:

return

"

except
:

return

"

i

=

d1

-

d2

return

"
```

```
%.1f
'
%
(
i
.
days

+

(
i
.
seconds
/
(
24.0
*
60.0
*
60.0
)))
```

[\[docs\]](#)
class

BuiltinDateArithmetic
(
BuiltinFormatterFunction
):

name

=

'date_arithmetic'

arg_count

=

```
-  
1
```

```
category
```

```
=
```

```
'Date functions'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-  
(
```

```
'date_arithmetic(date, calc_spec, fmt) -- '
```

```
"Calculate a new date from 'date' using 'calc_spec'. Return the "
```

```
"new date formatted according to optional 'fmt': if not supplied "
```

```
"then the result will be in iso format. The calc_spec is a string "
```

```
"formed by concatenating pairs of 'vW' (valueWhat) where 'v' is a "
```

```
"possibly-negative number and W is one of the following letters: "
```

```
"s: add 'v' seconds to 'date' "
```

```
"m: add 'v' minutes to 'date' "
```

```
"h: add 'v' hours to 'date' "
```

```
"d: add 'v' days to 'date' "
```

```
"w: add 'v' weeks to 'date' "
```

```
"y: add 'v' years to 'date', where a year is 365 days. "
```

```
"Example: '1s3d-1m' will add 1 second, add 3 days, and subtract 1 "
```

```
"minute from 'date'."
```

```
)
```

```
calc_ops
```

```
=
```

```
{
```

```
's'
```

```
:
```

```
lambda
```

```
v
```

```
:
```

```
timedelta
```

```
(
```

```
seconds
```

```
=
```

```
v
```


),

'm'

:

lambda

v

:

timedelta

(

minutes

=

v

),

'h'

:

lambda

v

:

timedelta

(

hours

=

v

),

'd'

:

lambda

v

:

timedelta

(
days

=

v

),

'w'

:

lambda

v

:

timedelta

(
weeks

=

v

),

'y'

:

lambda

v

:

timedelta

(
days

=

v

*

365

),

```
}
```

```
def
```

```
    evaluate
```

```
    (
```

```
    self
```

```
,
```

```
    formatter
```

```
,
```

```
    kwargs
```

```
,
```

```
    mi
```

```
,
```

```
    locals
```

```
,
```

```
    date
```

```
,
```

```
    calc_spec
```

```
,
```

```
    fmt
```

```
=
```

```
    None
```

```
):
```

```
try
```

```
:
```

```
    d
```

```
=
```

```
parse_date
(
date
)

if

d

==

UNDEFINED_DATE
:

return

"

while

calc_spec
:

mo

=

re
.
match
(
r
'([-+\\d]+)([smhdwy])'
,
calc_spec
)
```

```
if
mo
is
None
:

raise

ValueError
(

_
(
"
{0}
: invalid calculation specifier '
{1}
"
)
.
format
(

'date_arithmetic'
,
calc_spec
))

d
+=

self
.
```

```
calc_ops
```

```
[  
mo
```

```
[  
2
```

```
]](  
int
```

```
(  
mo
```

```
[  
1
```

```
]))
```

```
calc_spec
```

```
=
```

```
calc_spec
```

```
[  
len
```

```
(  
mo
```

```
[  
0
```

```
]):]
```

```
return
```

```
format_date
```

```
(  
d
```

```
,
```

```
fmt
```

```
if
```

```
fmt
```

```
else
```

```
'iso'  
)  
  
except  
ValueError  
as  
e  
:  
  
raise  
e  
  
except  
Exception  
as  
e  
:  
  
traceback  
.  
print_exc  
()  
  
raise  
ValueError  
(  
_  
(  
"  
{0}
```

```
: error:
{1}
"
)
.
format
(
'date_arithmetic'
,

str
(
e
)))
```

[\[docs\]](#)
class

BuiltinLanguageStrings
(
BuiltinFormatterFunction
):

name

=

'language_strings'

arg_count

=

2

category

=


```
'Get values from metadata'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-
```

```
(  
'language_strings(lang_codes, localize) -- '
```

```
'return the strings for the language codes passed in lang_codes. '
```

```
'If localize is zero, return the strings in English. If '
```

```
'localize is not zero, return the strings in the language of '
```

```
'the current locale. Lang_codes is a comma-separated list.'
```

```
)
```

```
def
```

```
evaluate
```

```
(
```

```
self
```

```
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
',

locals
',

lang_codes
',

localize
):

retval

=

[]

for

c

in

[
c
.
strip
()

for

c

in

lang_codes
.
split
(
','
,
```

```
)  
  
if  
  
c  
.  
strip  
():  
  
try  
:  
  
n  
  
=  
  
calibre_langcode_to_name  
(  
c  
,  
  
localize  
  
!=  
  
'0'  
)  
  
if  
  
n  
:  
  
retval  
.  
append  
(  
n  
)
```

```
except
```

```
:
```

```
pass
```

```
return
```

```
','
```

```
.
```

```
join
```

```
(
```

```
retval
```

```
)
```

```
\[docs\]
```

```
class
```

```
BuiltinLanguageCodes
```

```
(
```

```
BuiltinFormatterFunction
```

```
):
```

```
name
```

```
=
```

```
'language_codes'
```

```
arg_count
```

```
=
```

```
1
```

```
category
```

```
=
```

```
'Get values from metadata'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-  
(
```

```
'language_codes(lang_strings) -- '
```

```
'return the language codes for the strings passed in lang_strings. '
```

```
'The strings must be in the language of the current locale. '
```

```
'Lang_strings is a comma-separated list.'
```

```
)
```

```
def
```

```
evaluate
```

```
(
```

```
self
```

```
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
',
locals
',
lang_strings
):
```

```
retval
```

```
=
```

```
[]
```

```
for
```

```
c
```

```
in
```

```
[
c
.
strip
()]
```

```
for
```

```
c
```

```
in
```

```
lang_strings
.
split
(
', '
)
)
```

```
if
```

```
c
.
strip
[]):

try
:

cv

=

canonicalize_lang
(
c
)

if

cv
:

retval
.
append
(
canonicalize_lang
(
cv
))

except
:

pass
```

```
return
```

```
','
```

```
.
```

```
join
```

```
(
```

```
retval
```

```
)
```

```
\[docs\]
```

```
class
```

```
BuiltinCurrentLibraryName
```

```
(
```

```
BuiltinFormatterFunction
```

```
):
```

```
name
```

```
=
```

```
'current_library_name'
```

```
arg_count
```

```
=
```

```
0
```

```
category
```

```
=
```

```
'Get values from metadata'
```

```
__doc__
```



```
=

doc

=

-
(
'current_library_name() -- '

'return the last name on the path to the current calibre library. '

'This function can be called in template program mode using the '

'template "{:
\'
current_library_name()
\'
}":'
)

def

evaluate
(
self
,

formatter
,

kwargs
,

mi
,

locals
```

```
):
```

```
from
```

```
calibre.library
```

```
import
```

```
current_library_name
```

```
return
```

```
current_library_name
```

```
()
```

```
\[docs\]
```

```
class
```

```
BuiltinCurrentLibraryPath
```

```
(
```

```
BuiltinFormatterFunction
```

```
):
```

```
name
```

```
=
```

```
'current_library_path'
```

```
arg_count
```

```
=
```

```
0
```

```
category
```

```

=

'Get values from metadata'

__doc__

=

doc

=

-
(
'current_library_path() -- '

'return the path to the current calibre library. This function can '

'be called in template program mode using the template '

'''{
\
current_library_path()
\
}''
)

def

evaluate
(
self
,

formatter
,

```

```
kwargs
,

mi
,

locals
):

from

calibre.library

import

current_library_path

return

current_library_path
()

\[docs\]
class

BuiltinFinishFormatting
(
    BuiltinFormatterFunction
):

    name

    =

    'finish_formatting'

    arg_count
```

=

4

category

=

'Formatting values'

__doc__

=

doc

=

(

'finish_formatting(val, fmt, prefix, suffix) -- apply the '

'format, prefix, and suffix to a value in the same way as '

'done in a template like `{series\_index:05.2f| - |-}`. For '

'example, the following program produces the same output '

'as the above template: '

'program: finish_formatting(field("series_index"), "05.2f", " - ", " - ")'

)

def

```
evaluate
(
self
,
formatter
,
kwargs
,
mi
,
locals_
,
val
,
fmt
,
prefix
,
suffix
):
```

```
if
```

```
not
```

```
val
:
```

```
return
```

```
val
```

return

prefix

+

formatter

.

_do_format

(

val

,

fmt

)

+

suffix

[\[docs\]](#)

class

BuiltinVirtualLibraries

(

BuiltinFormatterFunction

):

name

=

'virtual_libraries'

arg_count

=

0

category

=

'Get values from metadata'

__doc__

=

doc

=

-
(

'virtual_libraries() -- return a comma-separated list of '

'Virtual libraries that contain this book. This function '

'works only in the GUI. If you want to use these values '

'in save-to-disk or send-to-device templates then you '

'must make a custom "Column built from other columns", use '

'the function in that column

'
s template, and use that '

'column

'
s value in your save/send templates'
)


```
def
```

```
evaluate
```

```
(  
self
```

```
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
```

```
,
```

```
locals_  
):
```

```
with
```

```
suppress
```

```
(  
Exception  
):
```

```
try
```

```
:
```

```
from
```

```
calibre.gui2.ui
```

```
import
```

```
get_gui
```

```

a
=

get_gui
()
.
current_db
.
data
.
get_virtual_libraries_for_books
((
mi
.
id
,))

return

','
.
join
(
a
[
mi
.
id
])

except

ValueError

as

v
:
```

return

str

(
v
)

return

—
(
'This function can be used only in the GUI'
)

[\[docs\]](#)

class

BuiltinCurrentVirtualLibraryName

(
BuiltinFormatterFunction
):

name

=

'current_virtual_library_name'

arg_count

=

0

category

=

```
'Get values from metadata'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
(
```

```
'current_virtual_library_name() -- '
```

```
'return the name of the current virtual library if there is one, '
```

```
'otherwise the empty string. Library name case is preserved. '
```

```
'Example: "program: current_virtual_library_name()".'
```

```
)
```

```
def
```

```
evaluate
```

```
(
```

```
self
```

```
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
```

```
,
```

```
locals
):

with

suppress
(
Exception
):

from

calibre.gui2.ui

import

get_gui

return

get_gui
()
.
current_db
.
data
.
get_base_restriction_name
()

return

—
(
'This function can be used only in the GUI'
)
```

[\[docs\]](#)

class

BuiltinUserCategories

(
BuiltinFormatterFunction
):

name

=

'user_categories'

arg_count

=

0

category

=

'Get values from metadata'

__doc__

=

doc

=

—
(

'user_categories()' -- return a comma-separated list of '

'the user categories that contain this book. This function '

'works only in the GUI. If you want to use these values '

'in save-to-disk or send-to-device templates then you '

'must make a custom "Column built from other columns", use '

'the function in that column

\'
s template, and use that '

'column

\'
s value in your save/send templates'
)

def

evaluate

(
self
,

formatter

,

kwargs

,

mi

,

locals_

):

```
if
hasattr
(
mi
,
'_proxy_metadata'
):
```

```
cats
```

```
=
```

```
{
k
```

```
for
```

```
k
```

```
,
```

```
v
```

```
in
```

```
iteritems
```

```
(
```

```
mi
```

```
.
```

```
_proxy_metadata
```

```
.
```

```
user_categories
```

```
)
```

```
if
```

```
v
```

```
}
```



```
cats

=

sorted
(
cats
,
key
=
sort_key
)

return

', '
.
join
(
cats
)

return

-
(
'This function can be used only in the GUI'
)
```

[\[docs\]](#)

class

BuiltinTransliterate

```
(
BuiltinFormatterFunction
):
```

name

=

'transliterate'

arg_count

=

1

category

=

'String manipulation'

__doc__

=

doc

=

—

(

'transliterate(a) -- Returns a string in a latin alphabet '

'formed by approximating the sound of the words in the '

'source string. For example, if the source is "

{0}

""

' the function returns "

{1}

```
"'
)
.
format
(

"Фёдор Миха́йлович Достоевский"
,

'Fiodor Mikhailovich Dostoievskii'
)
```

```
def
```

```
evaluate
```

```
(
self
```

```
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
```

```
,
```

```
locals
```

```
,
```

```
source
```

```
):
```

```
from
```

```
calibre.utils filenames
```

```
import
```

```
ascii_text
```

```
return
```

```
ascii_text  
(  
    source  
)
```

```
\[docs\]
```

```
class
```

```
BuiltinAuthorLinks
```

```
(  
    BuiltinFormatterFunction  
):
```

```
name
```

```
=
```

```
'author_links'
```

```
arg_count
```

```
=
```

```
2
```

```
category
```

```
=
```

```
'Get values from metadata'
```

```
__doc__
```

```
=  
  
doc  
  
=  
  
-  
(  
'author_links(val_separator, pair_separator) -- returns '  
  
'a string containing a list of authors and that author  
'  
s '  
  
'link values in the '  
  
'form author1 val_separator author1link pair_separator '  
  
'author2 val_separator author2link etc. An author is '  
  
'separated from its link value by the val_separator string '  
  
'with no added spaces. author:linkvalue pairs are separated '  
  
'by the pair_separator string argument with no added spaces. '  
  
'It is up to you to choose separator strings that do '  
  
'not occur in author names or links. An author is '  
  
'included even if the author link is empty.'  
)
```

```
def
```

```
    evaluate
```

```
    (
```

```
    self
```

```
    ,
```

```
    formatter
```

```
    ,
```

```
    kwargs
```

```
    ,
```

```
    mi
```

```
    ,
```

```
    locals
```

```
    ,
```

```
    val_sep
```

```
    ,
```

```
    pair_sep
```

```
    ):
```

```
    if
```

```
        hasattr
```

```
        (
```

```
        mi
```

```
        ,
```

```
        '_proxy_metadata'
```

```
    ):
```

```
        link_data
```

```
        =
```

```
mi
·
_proxy_metadata
·
author_link_map
```

```
if
```

```
not
```

```
link_data
:
```

```
return
```

```
""
```

```
names
```

```
=
```

```
sorted
(
link_data
·
keys
(),
key
=
sort_key
)
```

```
return
```

```
pair_sep
·
join
```

```
(
n

+

val_sep

+

link_data
[
n
]

for

n

in

names
)

return

—
(
'This function can be used only in the GUI'
)
```

[\[docs\]](#)

class

BuiltinAuthorSorts

```
(
BuiltinFormatterFunction
):
```

name


```
=  
  
'author_sorts'  
  
arg_count  
  
=  
  
1  
  
category  
  
=  
  
'Get values from metadata'  
  
__doc__  
  
=  
  
doc  
  
=  
  
_  
(  
'author_sorts(val_separator) -- returns a string '  
  
'containing a list of author  
'\n'  
s sort values for the '  
  
'authors of the book. The sort is the one in the author '  
  
'metadata (different from the author_sort in books). The '
```

'returned list has the form author sort 1 val_separator '

'author sort 2 etc. The author sort values in this list '

'are in the same order as the authors of the book. If '

'you want spaces around val_separator then include them '

'in the separator string'

)

def

evaluate

(

self

,

formatter

,

kwargs

,

mi

,

locals

,

val_sep

):

sort_data

=

```
mi
.  
author_sort_map
```

```
if
```

```
not
```

```
sort_data  
:
```

```
return
```

```
''
```

```
names
```

```
=
```

```
[  
sort_data  
.  
get  
(  
n  
)
```

```
for
```

```
n
```

```
in
```

```
mi  
.  
authors
```

```
if
```

```
n
.
strip
()]

return

val_sep
.
join
(
n

for

n

in

names
)

\[docs\]
class

BuiltinConnectedDeviceName
(
    BuiltinFormatterFunction
):

    name

    =

    'connected_device_name'

    arg_count

    =
```

1

category

=

'Get values from metadata'

__doc__

=

doc

=

-

(
"connected_device_name(storage_location) -- if a device is "

"connected then return the device name, otherwise return "

"the empty string. Each storage location on a device can "

"have a different name. The location names are 'main', "

""carda' and 'cardb'. This function works only in the GUI."

)

def

evaluate

(

self

```
,
formatter
,
kwargs
,
mi
,
locals
,
storage_location
):

with

suppress
(
Exception
):

# Do the import here so that we don't entangle the GUI when using

# command line functions

from

calibre.gui2.ui

import

get_gui

info
```

```
=  
  
get_gui  
(  
.  
device_manager  
.  
get_current_device_information  
(  
  
if  
  
info  
  
is  
  
None  
:  
  
return  
  
"  
  
try  
:  
  
if  
  
storage_location  
  
not  
  
in  
  
{  
'main'  
,  
  
'carda'
```

```
,  
  
'cardb'  
}:  
  
raise  
  
ValueError  
(  
  
    _  
(  
    'connected_device_name: invalid storage location "  
    {}  
    "'  
  
    .  
    format  
    (  
    storage_location  
    )))  
  
info  
  
=  
  
info  
[  
    'info'  
][  
    4  
]  
  
if  
  
storage_location  
  
not
```


in

info

:

return

"

return

info

[

storage_location

][

'device_name'

]

except

Exception

:

traceback

.

print_exc

()

raise

return

—

(

'This function can be used only in the GUI'

```
)
```

```
\[docs\]
```

```
class
```

```
BuiltinConnectedDeviceUUID
```

```
(
```

```
BuiltinFormatterFunction
```

```
):
```

```
name
```

```
=
```

```
'connected_device_uuid'
```

```
arg_count
```

```
=
```

```
1
```

```
category
```

```
=
```

```
'Get values from metadata'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
-
```

```
(
```

```
"connected_device_uuid(storage_location) -- if a device is "
```

```
"connected then return the device uuid (unique id), "
```

```
"otherwise return the empty string. Each storage location "
```

```
"on a device has a different uuid. The location names are "
```

```
""main', 'carda' and 'cardb'. This function works only in "
```

```
"the GUI."
```

```
)
```

```
def
```

```
evaluate
```

```
(
```

```
self
```

```
,
```

```
formatter
```

```
,
```

```
kwargs
```

```
,
```

```
mi
```

```
,
```

```
locals
```

```
,
```

```
storage_location
```

```
):
```

with

suppress

(

Exception

):

Do the import here so that we don't entangle the GUI when using

command line functions

from

calibre.gui2.ui

import

get_gui

info

=

get_gui

()

.

device_manager

.

get_current_device_information

()

if

info

is

None

```
:  
  
return  
  
"  
  
try  
:  
  
if  
  
storage_location  
  
not  
  
in  
  
{  
'main'  
,  
  
'carda'  
,  
  
'cardb'  
}:  
  
raise  
  
ValueError  
(  
  
_  
(  
'connected_device_name: invalid storage location "  
{  
"'
```

```
.  
format  
(  
storage_location  
)))
```

```
info
```

```
=
```

```
info  
[  
'info'  
][  
4  
]
```

```
if
```

```
storage_location
```

```
not
```

```
in
```

```
info  
:
```

```
return
```

```
""
```

```
return
```

```
info  
[  
storage_location  
][
```

```
'device_store_uuid'
]

except

Exception
:

traceback
.
print_exc
()

raise

return

_
(
'This function can be used only in the GUI'
)

\[docs\]
class

BuiltinCheckYesNo
(
BuiltinFormatterFunction
):

name

=

'check_yes_no'
```

arg_count

=

4

category

=

'If-then-else'

__doc__

=

doc

=

(

'check_yes_no(field_name, is_undefined, is_false, is_true) '

'-- checks the value of the yes/no field named by the '

'lookup key field_name for a value specified by the '

'parameters, returning "yes" if a match is found, otherwise '

'returning an empty string. Set the parameter is_undefined, '

'is_false, or is_true to 1 (the number) to check that '

'condition, otherwise set it to 0. Example: '

'check_yes_no("#bool", 1, 0, 1) returns "yes" if the '

'yes/no field "#bool" is either undefined (neither True '

'nor False) or True. More than one of is_undefined, '

'is_false, or is_true can be set to 1. This function '

'is usually used by the test() or is_empty() functions.'
)

def

evaluate

(
self
,

formatter
,

kwargs
,

mi
,

locals
,

field
,

is_undefined
,

```
is_false
,

is_true
):

res

=

getattr
(
mi
,
field
,
None
)

if

res

is

None

:

if

is_undefined

==

'1'
:
```

return

'yes'

return

"""

if

not

isinstance

(

res

,

bool

):

raise

ValueError

(

—

(

'check_yes_no requires the field be a Yes/No custom column'

))

if

is_false

==

'1'

and

not

res

:

return

'yes'

if

is_true

==

'1'

and

res

:

return

'yes'

return

"""

[\[docs\]](#)

class

BuiltinRatingToStars

(

BuiltinFormatterFunction

):

```
name
```

```
=
```

```
'rating_to_stars'
```

```
arg_count
```

```
=
```

```
2
```

```
category
```

```
=
```

```
'Formatting values'
```

```
__doc__
```

```
=
```

```
doc
```

```
=
```

```
(
```

```
'rating_to_stars(value, use_half_stars) '
```

```
'-- Returns the rating as string of star characters. '
```

```
'The value is a number between 0 and 5. Set use_half_stars '
```

```
'to 1 if you want half star characters for custom ratings '
```

```
'columns that support non-integer ratings, for example 2.5.'  
)
```

```
def
```

```
    evaluate
```

```
    (  
    self
```

```
,
```

```
    formatter
```

```
,
```

```
    kwargs
```

```
,
```

```
    mi
```

```
,
```

```
    locals
```

```
,
```

```
    value
```

```
,
```

```
    use_half_stars
```

```
):
```

```
if
```

```
not
```

```
    value
```

```
:
```

```
return
```

```
"  
  
err_msg  
  
=  
  
_  
(  
'The rating must be a number between 0 and 5'  
)  
  
try  
:  
  
v  
  
=  
  
float  
(  
value  
)  
  
*  
  
2  
  
except  
:  
  
raise  
  
ValueError  
(  
err_msg  
)
```

```
if
v
<
0
or
v
>
10
:

raise

ValueError
(
err_msg
)

from

calibre.ebooks.metadata

import

rating_to_stars

return

rating_to_stars
(
v
,
use_half_stars
```



```
==
```

```
'1'
```

```
)
```

```
\[docs\]
```

```
class
```

```
BuiltinSwapAroundArticles
```

```
(
```

```
BuiltinFormatterFunction
```

```
):
```

```
name
```

```
=
```

```
'swap_around_articles'
```

```
arg_count
```

```
=
```

```
2
```

```
category
```

```
=
```

```
'String manipulation'
```

```
__doc__
```

```
=
```

```
doc
```

```
=  
  
_  
(  
'swap_around_articles(val, separator) '  
  
'-- returns the val with articles moved to the end. '  
  
'The value can be a list, in which case each member '  
  
'of the list is processed. If the value is a list then '  
  
'you must provide the list value separator. If no '  
  
'separator is provided then the value is treated as '  
  
'being a single value, not a list.'  
)
```

def

evaluate

(
self

,

formatter

,

kwargs

,

mi

,

```
locals
```

```
,
```

```
val
```

```
,
```

```
separator
```

```
):
```

```
if
```

```
not
```

```
val
```

```
:
```

```
return
```

```
"
```

```
if
```

```
not
```

```
separator
```

```
:
```

```
return
```

```
title_sort
```

```
(
```

```
val
```

```
)
```

```
.
```

```
replace
```

```
(
```

```
' '
```

```
,
```

```
,
```

```
    ),
)

result

=

[]

try
:

for
v
in
[
x
.
strip
()
for
x
in
val
.
split
(
separator
)]:

result
.
```

```
append
(  
title_sort  
(  
v  
)
```

```
.  
replace  
(  
' '  
,  
  
'.'  
,  
)  
)
```

```
except  
:
```

```
traceback  
.  
print_exc  
(  
)
```

```
return
```

```
separator  
.  
join  
(  
sorted  
(  
result  
,  
  
key  
=  
sort_key  
)  
)
```

[\[docs\]](#)

class

BuiltinArguments

(
BuiltinFormatterFunction
):

name

=

'arguments'

arg_count

=

-

1

category

=

'other'

__doc__

=

doc

=

-

(
'arguments(id[=expression] [, id[=expression]]*) '

'-- Used in a stored template to retrieve the arguments '

'passed in the call. It both declares and initializes '

'local variables, effectively parameters. The variables '

'are positional; they get the value of the value given '

'in the call in the same position. If the corresponding '

'parameter is not provided in the call then arguments '

'assigns that variable the provided default value. If '

'there is no default value then the variable is set to '

'the empty string.'

)

def

evaluate

(

self

,

formatter

,

kwargs

,

```
mi
',

locals
',

*
args
):

# The arguments function is implemented in-line in the formatter

raise

NotImplementedError
()

class

BuiltinGlobals
(
BuiltinFormatterFunction
):

name

=

'globals'

arg_count

=

-
1
```



```
category
=
'other'

__doc__
=
doc
=
-
(
'globals(id[=expression] [, id[=expression]])*'
'-- Retrieves "global variables" that can be passed into '
'the formatter. It both declares and initializes local '
'variables with the names of the global variables passed '
'in. If the corresponding variable is not provided in '
'the passed-in globals then it assigns that variable the '
'provided default value. If there is no default value '
'then the variable is set to the empty string.'
)
```

```
def
evaluate
(
self
,
formatter
,
kwargs
,
mi
,
locals
,
*
args
):

# The globals function is implemented in-line in the formatter

raise

NotImplementedError
()

\[docs\]
class

BuiltinSetGlobals
(
BuiltinFormatterFunction
):
```

name

=

'set_globals'

arg_count

=

-

1

category

=

'other'

__doc__

=

doc

=

-

('globals(id[=expression] [, id[=expression]]*) ')

'-- Retrieves "global variables" that can be passed into '

'the formatter. It both declares and initializes local '

'variables with the names of the global variables passed '

'in. If the corresponding variable is not provided in '

'the passed-in globals then it assigns that variable the '

'provided default value. If there is no default value '

'then the variable is set to the empty string.'

)

def

evaluate

(

self

,

formatter

,

kwargs

,

mi

,

locals

,

*

args

):

The globals function is implemented in-line in the formatter

raise

NotImplementedError

()

[\[docs\]](#)

class

BuiltinFieldExists

(

BuiltinFormatterFunction

):

name

=

'field_exists'

arg_count

=

1

category

=

'If-then-else'

__doc__

=

doc

```

=

_
(
'field_exists(field_name) -- checks if a field '

'(column) named field_name exists, returning '

'"1' if so and " if not."
)

def
evaluate
(
self
,
formatter
,
kwargs
,
mi
,
locals
,
field_name
):

if
field_name
.
lower

```

```
()

in

mi
.
all_field_keys
():

return

'1'

return

"

[docs]
class

BuiltinCharacter
(
BuiltinFormatterFunction
):

name

=

'character'

arg_count

=

1
```

```
category

=

'String manipulation'

__doc__

=

doc

=

(
'character(character_name) -- returns the '

'character named by character_name. For example, '

r
"character('newline') returns a newline character ('\n'). "

"The supported character names are 'newline', 'return', "

"tab', and 'backslash'."
)

def

evaluate
(
self
,
formatter
```



```
,  
  
kwargs  
,  
  
mi  
,  
  
locals  
,  
  
character_name  
):  
  
# The globals function is implemented in-line in the formatter  
  
raise  
  
NotImplementedError  
()  
  
_formatter_builtins  
  
=  
  
[  
  
BuiltinAdd  
(),  
  
BuiltinAnd  
(),  
  
BuiltinApproximateFormats  
(),  
  
BuiltinArguments  
(),
```

BuiltinAssign
(),

BuiltinAuthorLinks
(),

BuiltinAuthorSorts
(),

BuiltinBooksize
(),

BuiltinCapitalize
(),

BuiltinCharacter
(),

BuiltinCheckYesNo
(),

BuiltinCeiling
(),

BuiltinCmp
(),

BuiltinConnectedDeviceName
(),

BuiltinConnectedDeviceUUID
(),

BuiltinContains
(),

BuiltinCount
(),

BuiltinCurrentLibraryName
(),

BuiltinCurrentLibraryPath
(),

BuiltinCurrentVirtualLibraryName
(),

BuiltinDateArithmetic
(),

BuiltinDaysBetween
(),

BuiltinDivide
(),

BuiltinEval
(),

BuiltinFirstNonEmpty
(),

BuiltinField
(),

BuiltinFieldExists
(),

BuiltinFinishFormatting
(),

BuiltinFirstMatchingCmp
(),

BuiltinFloor
(),

BuiltinFormatDate

(),

BuiltinFormatNumber

(),

BuiltinFormatsModtimes

(),

BuiltinFormatsPaths

(),

BuiltinFormatsSizes

(),

BuiltinFractionalPart

(),

BuiltinGlobals

(),

BuiltinHasCover

(),

BuiltinHumanReadable

(),

BuiltinIdentifierInList

(),

BuiltinIfempty

(),

BuiltinLanguageCodes

(),

BuiltinLanguageStrings

`()`,

`BuiltinInList`

`()`,

`BuiltinIsMarked`

`()`,

`BuiltinListCountMatching`

`()`,

`BuiltinListDifference`

`()`,

`BuiltinListEquals`

`()`,

`BuiltinListIntersection`

`()`,

`BuiltinListitem`

`()`,

`BuiltinListRe`

`()`,

`BuiltinListReGroup`

`()`,

`BuiltinListRemoveDuplicates`

`()`,

`BuiltinListSort`

`()`,

`BuiltinListSplit`

`()`,

BuiltinListUnion

(),

BuiltinLookup

(),

BuiltinLowercase

(),

BuiltinMod

(),

BuiltinMultiply

(),

BuiltinNot

(),

BuiltinOndevice

(),

BuiltinOr

(),

BuiltinPrint

(),

BuiltinRatingToStars

(),

BuiltinRawField

(),

BuiltinRawList

(),

BuiltinRe

(),

BuiltinReGroup

(),

BuiltinRound
(),

BuiltinSelect
(),

BuiltinSeriesSort
(),

BuiltinSetGlobals
(),

BuiltinShorten
(),

BuiltinStrcat
(),

BuiltinStrcatMax
(),

BuiltinStrcmp
(),

BuiltinStrInList
(),

BuiltinStrlen
(),

BuiltinSubitems
(),

BuiltinSublist
(),
BuiltinSubstr
(),

BuiltinSubtract

```
(),  
  
BuiltinSwapAroundArticles  
(),  
  
BuiltinSwapAroundComma  
(),  
  
BuiltinSwitch  
(),  
  
BuiltinTemplate  
(),  
  
BuiltinTest  
(),  
  
BuiltinTitlecase  
(),  
  
BuiltinToday  
(),  
  
BuiltinTransliterate  
(),  
  
BuiltinUppercase  
(),  
  
BuiltinUserCategories  
(),  
  
BuiltinVirtualLibraries  
(),  
  
BuiltinAnnotationCount  
(  
]
```


class

FormatterUserFunction

```
(  
    FormatterFunction  
):
```

def

__init__

```
(  
    self  
,  
  
    name  
,  
  
    doc  
,  
  
    arg_count  
,  
  
    program_text  
,  
  
    is_python  
):
```

self

is_python

=

is_python

```
self
.  
name
```

```
=  
  
name
```

```
self  
.  
doc
```

```
=  
  
doc
```

```
self  
.  
arg_count  
  
=  
  
arg_count
```

```
self  
.  
program_text  
  
=  
  
program_text
```

```
self  
.  
cached_parse_tree  
  
=
```

None

def

to_pref

(
self
):

return

[
self
.
name
,

self
.
doc
,

self
.
arg_count
,

self
.
program_text
]

tabs

=

re
.

```
compile
```

```
(
```

```
r
```

```
'^\t*'
```

```
)
```

```
def
```

```
function_pref_is_python
```

```
(
```

```
pref
```

```
):
```

```
if
```

```
isinstance
```

```
(
```

```
pref
```

```
,
```

```
list
```

```
):
```

```
pref
```

```
=
```

```
pref
```

```
[
```

```
3
```

```
]
```

```
if
```

```
pref
```

```
.
```

```
startswith
```

```
(
```

```
'def'
):

return

True

if

pref
.
startswith
(
'program'
):

return

False

raise

ValueError
(
'Unknown program type in formatter function pref'
)

def

function_pref_name
(
pref
):

return
```

```
pref  
[  
0  
]
```

```
def
```

```
compile_user_function
```

```
(  
name  
,
```

```
doc  
,
```

```
arg_count  
,
```

```
eval_func  
):
```

```
if
```

```
not
```

```
function_pref_is_python  
(  
eval_func  
):
```

```
return
```

```
FormatterUserFunction
```

```
(  
name  
,
```

```
doc  
,
```

```
arg_count
```

```
,
```

```
eval_func
```

```
,
```

```
False
```

```
)
```

```
def
```

```
replace_func
```

```
(
```

```
mo
```

```
):
```

```
return
```

```
mo
```

```
.
```

```
group
```

```
()
```

```
.
```

```
replace
```

```
(
```

```
'
```

```
\t
```

```
'
```

```
,
```

```
' '
```

```
)
```

```
func
```

```
=
```

```

'
+
'
\n
'
.
join
([
tabs
.
sub
(
replace_func
,
line
)

for
line
in
eval_func
.
splitlines
())

prog
=
'''

from calibre.utils.formatter_functions import FormatterUserFunction

from calibre.utils.formatter_functions import formatter_functions

```



```
class UserFunction(FormatterUserFunction):
```

```
'''
```

```
+
```

```
func
```

```
locals_
```

```
=
```

```
{}
```

```
if
```

```
DEBUG
```

```
and
```

```
tweaks
```

```
.
```

```
get
```

```
(
```

```
'enable_template_debug_printing'
```

```
,
```

```
False
```

```
):
```

```
print
```

```
(
```

```
prog
```

```
)
```

```
exec
```

```
(
```

```
prog
```

```
,
```

```
locals_  
)  
  
cls  
  
=  
  
locals_  
[  
  'UserFunction'  
](  
  name  
  ,  
  doc  
  ,  
  arg_count  
  ,  
  eval_func  
  ,  
  True  
)
```

```
return
```

```
cls
```

```
def
```

```
compile_user_template_functions  
(  
  funcs  
):
```

```
compiled_funcs

=

{}

for

func

in

funcs
:

try
:

# Force a name conflict to test the logic

# if func[0] == 'myFunc2':

#   func[0] = 'myFunc3'

# Compile the function so that the tab processing is done on the

# source. This helps ensure that if the function already is defined

# then white space differences don't cause them to compare differently

cls
```

```
=  
  
compile_user_function  
(  
*  
func  
)
```

```
cls  
.  
is_python
```

```
=  
  
function_pref_is_python  
(  
func  
)
```

```
compiled_funcs  
[  
cls  
.  
name  
]
```

```
=  
  
cls
```

```
except
```

```
Exception  
:
```

```
try  
:
```

```
func_name

=

func
[
0
]

except

Exception
:

func_name

=

'Unknown'

prints
(
'**** Compilation errors in user template function "
%s
" ****'

%

func_name
)

traceback
.
print_exc
(
limit
=
10
)
```

```
prints
(
    """ End compilation errors in
    %s
    """

    %

    func_name
)

return

compiled_funcs

def

load_user_template_functions
(
    library_uuid
,

    funcs
,

    precompiled_user_functions
=
    None
):

unload_user_template_functions
(
    library_uuid
)

if
```

```
precompiled_user_functions  
:
```

```
compiled_funcs
```

```
=
```

```
precompiled_user_functions
```

```
else
```

```
:
```

```
compiled_funcs
```

```
=
```

```
compile_user_template_functions
```

```
(
```

```
funcs
```

```
)
```

```
formatter_functions
```

```
()
```

```
.
```

```
register_functions
```

```
(
```

```
library_uuid
```

```
,
```

```
list
```

```
(
```

```
compiled_funcs
```

```
.
```

```
values
```

```
()))
```

def

unload_user_template_functions

(

library_uuid

):

formatter_functions

()

.

unregister_functions

(

library_uuid

)

Source code for calibre.web.feeds.news

```
__license__

=

'GPL v3'

__copyright__

=

'2008, Kovid Goyal <kovid at kovidgoyal.net>'

'''

Defines various abstract base classes that can be subclassed to create powerful news fetching recipes.

'''

__docformat__

=

"restructuredtext en"


import

io

import

os
```

import

re

import

sys

import

time

import

traceback

from

collections

import

defaultdict

from

contextlib

import

closing

from

urllib.parse

import

urlparse

,

urlsplit

```
from
calibre

import

(

__appname__
,

as_unicode
,

browser
,

force_unicode
,

iswindows
,

preferred_encoding
,

random_user_agent
,

strftime
)

from
calibre.ebooks.BeautifulSoup

import

BeautifulSoup
```

```
,  
  
CData  
,  
  
NavigableString  
,  
  
Tag  
  
from  
  
calibre.ebooks.metadata  
  
import  
  
MetaInformation  
  
from  
  
calibre.ebooks.metadata.opf2  
  
import  
  
OPFCreator  
  
from  
  
calibre.ebooks.metadata.toc  
  
import  
  
TOC  
  
from  
  
calibre.ptempfile  
  
import  
  
PersistentTemporaryFile  
  
from
```

calibre.utils.date

import

now

as

nowf

from

calibre.utils.icu

import

numeric_sort_key

from

calibre.utils.img

import

add_borders_to_image

,

image_to_data

,

save_cover_data_to

from

calibre.utils.localization

import

canonicalize_lang

from

calibre.utils.logging

import

ThreadSafeWrapper

from

calibre.utils.threadpool

import

NoResultsPending

,

ThreadPool

,

WorkRequest

from

calibre.web

import

Recipe

from

calibre.web.feeds

import

Feed

,

feed_from_xml

,

feeds_from_index

,

```
templates

from
calibre.web.fetch.simple

import

(

AbortArticle
,

RecursiveFetcher
,

option_parser

as

web2disk_option_parser

)

from
calibre.web.fetch.utils

import

prepare_masthead_image

from
polyglot.builtins

import

string_or_bytes
```

```
def
```

```
classes
```

```
(
```

```
classes
```

```
):
```

```
q
```

```
=
```

```
frozenset
```

```
(
```

```
classes
```

```
.
```

```
split
```

```
(
```

```
' '
```

```
))
```

```
return
```

```
dict
```

```
(
```

```
attrs
```

```
=
```

```
{
```

```
'class'
```

```
:
```

```
lambda
```

```
x
```

```
:
```

```
x
```

```
and
```



```
frozenset
```

```
(
```

```
x
```

```
.
```

```
split
```

```
())
```

```
.
```

```
intersection
```

```
(
```

```
q
```

```
))
```

```
def
```

```
prefixed_classes
```

```
(
```

```
classes
```

```
):
```

```
q
```

```
=
```

```
frozenset
```

```
(
```

```
classes
```

```
.
```

```
split
```

```
(
```

```
    )
```

```
))
```

```
def
```

```
matcher
```

```
(
```

```
x
```

```
):
```

```
if
```

```
x
```

```
:
```

```
for
```

```
candidate
```

```
in
```

```
frozenset
```

```
(
```

```
x
```

```
.
```

```
split
```

```
()):
```

```
for
```

```
x
```

```
in
```

```
q
```

```
:
```

```
if
```

```
candidate
```

```
.
```

```
startswith
```

```
(
```

```
x
```

```
):
```

```
return
```

True

return

False

return

```
{  
'attrs'  
:
```

```
{  
'class'  
:
```

```
matcher  
}}
```

class

LoginFailed

```
(  
ValueError  
):
```

pass

class

DownloadDenied

```
(  
ValueError  
):
```

```
pass
```

```
[docs]
```

```
class
```

```
BasicNewsRecipe
```

```
(  
Recipe  
):
```

```
'''
```

```
Base class that contains logic needed in all recipes. By overriding  
progressively more of the functionality in this class, you can make  
progressively more customized/powerful recipes. For a tutorial introduction  
to creating recipes, see :doc:`news`.
```

```
'''
```

```
#: The title to use for the e-book
```

```
title
```

```
=
```

```
—  
(
```

```
'Unknown News Source'
```

```
)
```

#: A couple of lines that describe the content this recipe downloads.

#: This will be used primarily in a GUI that presents a list of recipes.

description

=

"

#: The author of this recipe

__author__

=

__appname__

#: Minimum calibre version needed to use this recipe

requires_version

=

(

0

,

6

,

0

)

#: The language that the news is in. Must be an ISO-639 code either

#: two or three characters long

language

=

'und'

#: Maximum number of articles to download from each feed. This is primarily

#: useful for feeds that don't have article dates. For most feeds, you should

#: use :attr:`BasicNewsRecipe.oldest\_article`

max_articles_per_feed

=

100

#: Oldest article to download from this news source. In days.

oldest_article

=

7.0

#: Number of levels of links to follow on article webpages

recursions

=

0

#: Delay between consecutive downloads in seconds. The argument may be a

#: floating point number to indicate a more precise time.

delay

=

0

#: Publication type

#: Set to newspaper, magazine or blog. If set to None, no publication type

#: metadata will be written to the opf file.

publication_type

=

'unknown'

#: Number of simultaneous downloads. Set to 1 if the server is picky.

#: Automatically reduced to 1 if :attr:`BasicNewsRecipe.delay` > 0

simultaneous_downloads

=

5

#: Timeout for fetching files from server in seconds

timeout

=

120.0

#: The format string for the date shown on the first page.

#: By default: Day_Name, Day_Number Month_Name Year

timefmt

=

' [
%a
,
%d
%b %Y]'

#: List of feeds to download.

#: Can be either ``[url1, url2, ...]`` or ``[('title1', url1), ('title2', url2),...]``

feeds

=

None

#: Max number of characters in the short description

summary_length

=

500

#: Convenient flag to disable loading of stylesheets for websites

#: that have overly complex stylesheets unsuitable for conversion

#: to e-book formats.

#: If True stylesheets are not downloaded and processed

no_stylesheets

=

False

#: Convenient flag to strip all JavaScript tags from the downloaded HTML

remove_javascript

=

True

#: If True the GUI will ask the user for a username and password

#: to use while downloading.

#: If set to "optional" the use of a username and password becomes optional

needs_subscription

=

False

#: If True the navigation bar is center aligned, otherwise it is left aligned

center_navbar

=

True

#: Specify an override encoding for sites that have an incorrect

#: charset specification. The most common being specifying ``latin1`` and

#: using ``cp1252``. If None, try to detect the encoding. If it is a

#: callable, the callable is called with two arguments: The recipe object

#: and the source to be decoded. It must return the decoded source.

encoding

=

None

#: Normally we try to guess if a feed has full articles embedded in it

#: based on the length of the embedded content. If `None`, then the

#: default guessing is used. If `True` then the we always assume the feeds has

#: embedded content and if `False` we always assume the feed does not have

#: embedded content.

use_embedded_content

=

None

#: Set to True and implement :meth:`get\_obfuscated\_article` to handle

#: websites that try to make it difficult to scrape content.

articles_are_obfuscated

=

False

#: Reverse the order of articles in each feed

reverse_article_order

=

False

#: Automatically extract all the text from downloaded article pages. Uses

#: the algorithms from the readability project. Setting this to True, means

#: that you do not have to worry about cleaning up the downloaded HTML

#: manually (though manual cleanup will always be superior).

auto_cleanup

=

False

#: Specify elements that the auto cleanup algorithm should never remove.

#: The syntax is a XPath expression. For example::

#:

#: auto_cleanup_keep = '//div[@id="article-image"]' will keep all divs with

#: id="article-image"

#: auto_cleanup_keep = '//[@class="important"]' will keep all elements*

#: with class="important"

#: auto_cleanup_keep = '//div[@id="article-image"]//span[@class="important"]'

#: will keep all divs with id="article-image" and spans

#: with class="important"

#:

auto_cleanup_keep

=

None

#: Specify any extra `:term:`CSS`` that should be added to downloaded `:term:`HTML`` files.

#: It will be inserted into `<style>` tags, just before the closing

#: `</head>` tag thereby overriding all `:term:`CSS`` except that which is

#: declared using the style attribute on individual `:term:`HTML`` tags.

#: Note that if you want to programmatically generate the `extra_css` override

#: the `:meth:`get_extra_css()`` method instead.

#: For example::

#:

#: `extra_css = '.heading { font: serif x-large }'`

#:

`extra_css`

`=`

None

#: If `True` empty feeds are removed from the output.

#: This option has no effect if `parse_index` is overridden in

#: the sub class. It is meant only for recipes that return a list

#: of feeds using `feeds` or :meth:`get\_feeds`. It is also used if you use

#: the ignore_duplicate_articles option.

remove_empty_feeds

=

False

#: List of regular expressions that determines which links to follow.

#: If empty, it is ignored. Used only if is_link_wanted is

#: not implemented. For example::

#:

#: match_regexp = [r'page=[0-9]+']

#:

#: will match all URLs that have `page=some number` in them.

#:

#: Only one of :attr:`BasicNewsRecipe.match\_regexps` or

#: :attr:`BasicNewsRecipe.filter\_regexps` should be defined.

`match_regexps`

`=`

`[]`

#: List of regular expressions that determines which links to ignore.

#: If empty it is ignored. Used only if `is_link_wanted` is not

#: implemented. For example::

#:

#: `filter_regexps = [r'ads\.doubleclick\.net']`

#:

#: will remove all URLs that have `ads.doubleclick.net` in them.

#:

#: Only one of :attr:`BasicNewsRecipe.match\_regexps` or

#: :attr:`BasicNewsRecipe.filter\_regexps` should be defined.


```
filter_regexp
```

```
=
```

```
[]
```

```
#: Recipe specific options to control the conversion of the downloaded
```

```
#: content into an e-book. These will override any user or plugin specified
```

```
#: values, so only use if absolutely necessary. For example::
```

```
#: 
```

```
#: conversion_options = {
```

```
#: 'base_font_size' : 16,
```

```
#: 'linearize_tables' : True,
```

```
#: }
```

```
#: 
```

```
conversion_options
```

```
=
```

```
{}
```

#: List of tags to be removed. Specified tags are removed from downloaded HTML.

#: A tag is specified as a dictionary of the form::

#:

#: {

#: name : 'tag name', #e.g. 'div'

#: attrs : a dictionary, #e.g. {'class': 'advertisement'}

#: }

#:

#: All keys are optional. For a full explanation of the search criteria, see

#: `Beautiful Soup <<https://www.crummy.com/software/BeautifulSoup/bs4/doc/#searching-the-tree>>`__

#: A common example::

#:

#: remove_tags = [dict(name='div', class_='advert')]

#:

#: This will remove all `

#: their children from the downloaded :term:`HTML`.

remove_tags

=

[]

#: Remove all tags that occur after the specified tag.

#: For the format for specifying a tag see :attr:`BasicNewsRecipe.remove\_tags`.

#: For example::

#:

#: remove_tags_after = [dict(id='content')]

#:

#: will remove all

#: tags after the first element with `id="content"`.

remove_tags_after

=

None

#: Remove all tags that occur before the specified tag.

#: For the format for specifying a tag see :attr:`BasicNewsRecipe.remove\_tags`.

#: For example::

#:

#: remove_tags_before = dict(id='content')

#:

#: will remove all

#: tags before the first element with `id="content"`.

remove_tags_before

=

None

#: List of attributes to remove from all tags.

#: For example::

#:

```
#: remove_attributes = ['style', 'font']
```

```
remove_attributes
```

```
=
```

```
[]
```

```
#: Keep only the specified tags and their children.
```

```
#: For the format for specifying a tag see :attr:`BasicNewsRecipe.remove_tags`.
```

```
#: If this list is not empty, then the `<body>` tag will be emptied and re-filled with
```

```
#: the tags that match the entries in this list. For example::
```

```
#:
```

```
#: keep_only_tags = [dict(id=['content', 'heading'])]
```

```
#:
```

```
#: will keep only tags that have an `id` attribute of `"content"` or `"heading"`.
```

```
keep_only_tags
```

```
=
```

```
[]
```

#: List of :term:`regexp` substitution rules to run on the downloaded :term:`HTML`.

#: Each element of the

#: list should be a two element tuple. The first element of the tuple should

#: be a compiled regular expression and the second a callable that takes

#: a single match object and returns a string to replace the match. For example::

#:

#: preprocess_regexps = [

#: (re.compile(r'<!--Article ends here-->.</body>', re.DOTALL|re.IGNORECASE),*

#: lambda match: '</body>'),

#:]

#:

#: will remove everything from `<!--Article ends here-->` to `</body>`.

preprocess_regexps

=

[]

#: The CSS that is used to style the templates, i.e., the navigation bars and

#: the Tables of Contents. Rather than overriding this variable, you should

#: use `extra\_css` in your recipe to customize look and feel.

template_css

=

'''

```
.article_date {  
    color: gray; font-family: monospace;  
}
```

```
.article_description {  
    text-indent: 0pt;  
}
```

```
a.article {  
    font-weight: bold; text-align:left;  
}
```

```
a.feed {  
    font-weight: bold;
```

```
}

.calibre_navbar {

    font-family:monospace;

}

'''
```

#: By default, calibre will use a default image for the masthead (Kindle only).

#: Override this in your recipe to provide a url to use as a masthead.

masthead_url

=

None

#: By default, the cover image returned by get_cover_url() will be used as

#: the cover for the periodical. Overriding this in your recipe instructs

#: calibre to render the downloaded cover into a frame whose width and height

#: are expressed as a percentage of the downloaded cover.

#: cover_margins = (10, 15, 'ffffff') pads the cover with a white margin

#: 10px on the left and right, 15px on the top and bottom.

#: Color names are defined `here` <<https://www.imagemagick.org/script/color.php>>`_.

#: Note that for some reason, white does not always work in Windows. Use

#: #ffffff instead

cover_margins

=

(

0

,

0

,

'#ffffff'

)

#: Set to a non empty string to disable this recipe.

#: The string will be used as the disabled message

recipe_disabled

=

None

#: Ignore duplicates of articles that are present in more than one section.

#: A duplicate article is an article that has the same title and/or URL.

#: To ignore articles with the same title, set this to::

#:

#: ignore_duplicate_articles = {'title'}

#:

#: To use URLs instead, set it to::

#:

#: ignore_duplicate_articles = {'url'}

#:

#: To match on title or URL, set it to::

#:

#: ignore_duplicate_articles = {'title', 'url'}

ignore_duplicate_articles

=

None

The following parameters control how the recipe attempts to minimize

JPEG image sizes

#: Set this to False to ignore all scaling and compression parameters and

#: pass images through unmodified. If True and the other compression

#: parameters are left at their default values, JPEG images will be scaled to fit

#: in the screen dimensions set by the output profile and compressed to size at

#: most $(w * h)/16$ where $w \times h$ are the scaled image dimensions.

`compress_news_images`

`=`

False

#: The factor used when auto compressing JPEG images. If set to None,

#: auto compression is disabled. Otherwise, the images will be reduced in size to

#: $(w * h)/\text{compress\_news\_images\_auto\_size}$ bytes if possible by reducing

#: the quality level, where $w \times h$ are the image dimensions in pixels.

#: The minimum JPEG quality will be 5/100 so it is possible this constraint

#: will not be met. This parameter can be overridden by the parameter

#: compress_news_images_max_size which provides a fixed maximum size for images.

#: Note that if you enable scale_news_images_to_device then the image will

#: first be scaled and then its quality lowered until its size is less than

*#: $(w * h)/\text{factor}$ where w and h are now the *scaled* image dimensions. In*

#: other words, this compression happens after scaling.

compress_news_images_auto_size

=

16

#: Set JPEG quality so images do not exceed the size given (in KBytes).

#: If set, this parameter overrides auto compression via compress_news_images_auto_size.

#: The minimum JPEG quality will be 5/100 so it is possible this constraint

#: will not be met.

compress_news_images_max_size

=

None

#: Rescale images to fit in the device screen dimensions set by the output profile.

#: Ignored if no output profile is set.

scale_news_images_to_device

=

True

#: Maximum dimensions (w,h) to scale images to. If scale_news_images_to_device is True

#: this is set to the device screen dimensions set by the output profile unless

#: there is no profile set, in which case it is left at whatever value it has been

#: assigned (default None).

scale_news_images

=

None

#: If set to True then links in downloaded articles that point to other downloaded articles are

#: changed to point to the downloaded copy of the article rather than its original web URL. If you

#: set this to True, you might also need to implement :meth:`canonicalize\_internal\_url` to work

#: with the URL scheme of your particular website.

resolve_internal_links

=

False

#: Set to False if you dont want to use gzipped transfers. Note that some old servers flake out with gzip

handle_gzip

=

True

See the built-in recipes for examples of these settings.

def

short_title

(

self

):

```
return
```

```
force_unicode
```

```
(  
self
```

```
.  
title
```

```
,
```

```
preferred_encoding
```

```
)
```

```
\[docs\]
```

```
def
```

```
is_link_wanted
```

```
(  
self
```

```
,
```

```
url
```

```
,
```

```
tag
```

```
):
```

```
'''
```

Return True if the link should be followed or False otherwise. By

default, raises NotImplementedError which causes the downloader to

ignore it.

:param url: The URL to be followed

:param tag: The tag from which the URL was derived

```
'''

raise

NotImplementedError
()

\[docs\]

def

get_extra_css
(
    self
):

'''

By default returns `self.extra_css`. Override if you want to programmatically generate the
extra_css.

'''

return

self
.
extra_css

\[docs\]

def

get_cover_url
(
    self
```



```

):

'''
    Return a :term:`URL` to the cover image for this issue or `None`.

    By default it returns the value of the member `self.cover_url` which
    is normally `None`. If you want your recipe to download a cover for the e-book
    override this method in your subclass, or set the member variable `self.cover_url`
    before this method is called.

'''

return

getattr
(
self
,
'cover_url'
,
None
)

[docs]

def
get_masthead_url
(
self
):

'''

```

Return a :term:`URL` to the masthead image for this issue or `None`.

By default it returns the value of the member `self.masthead\_url` which

is normally `None`. If you want your recipe to download a masthead for the e-book

override this method in your subclass, or set the member variable `self.masthead\_url`

before this method is called.

Masthead images are used in Kindle MOBI files.

'''

return

getattr

(

self

,

'masthead_url'

,

None

)

[\[docs\]](#)

def

get_feeds

(

self

):

'''

Return a list of :term:`RSS` feeds to fetch for this profile. Each element of the list

must be a 2-element tuple of the form (title, url). If title is None or an empty string, the title from the feed is used. This method is useful if your recipe needs to do some processing to figure out the list of feeds to download. If so, override in your subclass.

'''

if

not

self

.

feeds

:

raise

NotImplementedError

()

if

self

.

test

:

return

self

.

feeds

[:

self

```
.  
test  
[  
0  
]]
```

```
return
```

```
self
```

```
.  
feeds
```

```
\[docs\]
```

```
@classmethod
```

```
def
```

```
print_version
```

```
(  
cls  
,
```

```
url  
):
```

```
'''
```

Take a `url` pointing to the webpage with article content and return the

:term:`URL` pointing to the print version of the article. By default does

nothing. For example::

```
def print_version(self, url):
```

```
    return url + '?&pagewanted=print'
```

```
'''

raise

NotImplementedError
()

[docs]

@classmethod

def

image_url_processor
(
cls
,
baseurl
,
url
):

'''

    Perform some processing on image urls (perhaps removing size restrictions for
    dynamically generated images, etc.) and return the precessed URL.

    '''

return

url
```

[\[docs\]](#)

def

preprocess_image

(
self

,

img_data

,

image_url

):

'''

Perform some processing on downloaded image data. This is called on the raw

data before any resizing is done. Must return the processed raw data. Return

None to skip the image.

'''

return

img_data

[\[docs\]](#)

def

get_browser

(

self

,

*

args

```
,
```

```
**
```

```
kwargs
```

```
):
```

```
'''
```

Return a browser instance used to fetch documents from the web. By default

it returns a `mechanize <<https://mechanize.readthedocs.io/en/latest/>>`_

browser instance that supports cookies, ignores robots.txt, handles

refreshes and has a mozilla firefox user agent.

If your recipe requires that you login first, override this method

in your subclass. For example, the following code is used in the New York

Times recipe to login for full access::

```
def get_browser(self):
```

```
    br = BasicNewsRecipe.get_browser(self)
```

```
    if self.username is not None and self.password is not None:
```

```
        br.open('https://www.nytimes.com/auth/login')
```

```
        br.select_form(name='login')
```

```
        br['USERID'] = self.username
```

```
        br['PASSWORD'] = self.password
```

```
        br.submit()
```

```
    return br
```

```
'''

if
    'user_agent'
    not
    in
    kwargs
    :

    # More and more news sites are serving JPEG XR images to IE

    ua
    =
    getattr
    (
    self
    ,
    'last_used_user_agent'
    ,
    None
    )
    or
    self
    .
    calibre_most_common_ua
    or
    random_user_agent
```



```
(
allow_ie
=
False
)

kwargs
[
'user_agent'
]

=

self
.
last_used_user_agent

=

ua

self
.
log
(
'Using user agent:'
,

kwargs
[
'user_agent'
])

br

=

browser
(
*
```

```
args
,
**
kwargs
)

br
.
addheaders

+=

[(
'Accept'
,
'*/'
)]

if

self
.
handle_gzip
:

br
.
set_handle_gzip
(
True
)

return

br
```

[\[docs\]](#)

def

clone_browser

(
self
,

br
):

'''

*Clone the browser br. Cloned browsers are used for multi-threaded
downloads, since mechanize is not thread safe. The default cloning
routines should capture most browser customization, but if you do
something exotic in your recipe, you should override this method in
your recipe and clone manually.*

*Cloned browser instances use the same, thread-safe CookieJar by
default, unless you have customized cookie handling.*

'''

if

callable

(
getattr
(
br
,

'clone_browser'

```
,  
  
None  
)):  
  
return  
  
br  
.  
clone_browser  
()  
  
# Uh-oh recipe using something exotic, call get_browser  
  
return  
  
self  
.  
get_browser  
()  
  
@property  
  
def  
  
cloned_browser  
(  
self  
):  
  
if  
  
hasattr  
(  
self  
.  

```

```
get_browser
',

'is_base_class_implementation'
):

# We are using the default get_browser, which means no need to

# clone

br

=

BasicNewsRecipe
.
get_browser
(
self
)

else
:

br

=

self
.
clone_browser
(
self
.
browser
)
```

return

br

[\[docs\]](#)

def

get_article_url

(

self

,

article

):

'''

Override in a subclass to customize extraction of the :term:`URL` that points

to the content for each article. Return the

article URL. It is called with `article`, an object representing a parsed article

from a feed. See `feedparser` <<https://pythonhosted.org/feedparser/>>`_.

By default it looks for the original link (for feeds syndicated via a

service like FeedBurner or Pheedo) and if found,

returns that or else returns

`article.link` <<https://pythonhosted.org/feedparser/reference-entry-link.html>>`_.

'''

for

```
key
```

```
in
```

```
article
```

```
.
```

```
keys
```

```
():
```

```
if
```

```
key
```

```
.
```

```
endswith
```

```
(
```

```
'_origlink'
```

```
):
```

```
url
```

```
=
```

```
article
```

```
[
```

```
key
```

```
]
```

```
if
```

```
url
```

```
and
```

```
(
```

```
url
```

```
.
```

```
startswith
```

```
(
```

```
'http://'
```

```
)
```

or

url

.

startswith

(

'https://'

)):

return

url

ans

=

article

.

get

(

'link'

,

None

)

if

not

ans

and

getattr

(

article

,


```
'links'
,
None
):

for
item

in
article
.
links
:

if
item
.
get
(
'rel'
,
'alternate'
)
==
'alternate'
:

ans
=
item
```

```
[  
'href'  
]
```

```
break
```

```
return
```

```
ans
```

```
\[docs\]
```

```
def
```

```
skip_ad_pages
```

```
(  
self  
,
```

```
soup  
):
```

```
'''
```

This method is called with the source of each downloaded :term:`HTML` file, before

any of the cleanup attributes like `remove_tags`, `keep_only_tags` are

applied. Note that `preprocess_regexps` will have already been applied.

It is meant to allow the recipe to skip ad pages. If the soup represents

an ad page, return the HTML of the real page. Otherwise return

None.

`soup`: A `BeautifulSoup` <<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>>`__

```

        instance containing the downloaded :term:`HTML`.

        """

    return

    None

\[docs\]

    def

    abort_article
    (
    self
    ,

    msg
    =
    None
    ):

        """ Call this method inside any of the preprocess methods to abort the

            download for the current article. Useful to skip articles that contain

            inappropriate content, such as pure video articles. """

    raise

    AbortArticle
    (
    msg

    or

    _
    (
    'Article download aborted'

```

```
))
```

```
\[docs\]
```

```
def
```

```
    preprocess_raw_html
```

```
    (
```

```
    self
```

```
    ,
```

```
    raw_html
```

```
    ,
```

```
    url
```

```
    ):
```

```
    """
```

This method is called with the source of each downloaded :term:`HTML` file, before

it is parsed into an object tree. raw_html is a unicode string

representing the raw HTML downloaded from the web. url is the URL from

which the HTML was downloaded.

*Note that this method acts ***before*** preprocess_regexps.*

This method must return the processed raw_html as a unicode object.

```
    """
```

```
return
```

```
    raw_html
```

```
def
preprocess_raw_html_
(
self
,
raw_html
,
url
):

raw_html

=

self
.
preprocess_raw_html
(
raw_html
,
url
)

if
self
.
auto_cleanup
:

try
:

raw_html
```

```
=  
  
self  
.  
extract_readable_article  
(  
raw_html  
,  
  
url  
)
```

```
except  
:
```

```
self  
.  
log  
.  
exception  
(  
'Auto cleanup of URL:  
%r  
failed'  
%  
url  
)
```

```
return
```

```
raw_html
```

[\[docs\]](#)

```
def
```

```
preprocess_html
```

```
(  
self
```

```
,
```

```
soup
```

```
):
```

```
'''
```

This method is called with the source of each downloaded :term:`HTML` file, before it is parsed for links and images. It is called after the cleanup as specified by `remove_tags` etc.

It can be used to do arbitrarily powerful pre-processing on the :term:`HTML`.

It should return `soup` after processing it.

`soup`: A `BeautifulSoup` <<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>>`\_\_ instance containing the downloaded :term:`HTML`.

```
'''
```

```
return
```

```
soup
```

```
\[docs\]
```

```
def
```

```
postprocess_html
```

```
(
```

```
self
```

```
,
```

```
soup
```

```
,
```

```
first_fetch
```

```
):
```

```
'''
```

This method is called with the source of each downloaded :term:`HTML` file, after

it is parsed for links and images.

It can be used to do arbitrarily powerful post-processing on the :term:`HTML`.

It should return `soup` after processing it.

:param soup: A `BeautifulSoup

<<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>>`\_\_ instance containing the downloaded :term:`HTML`.

:param first_fetch: True if this is the first page of an article.

```
'''
```

```
return
```

```
soup
```

```
\[docs\]
```

```
def
```

```
cleanup
```

```
(
```

```
self
```

```
):
```



```
'''  
  
    Called after all articles have been download. Use it to do any cleanup like  
  
    logging out of subscription sites, etc.  
  
    '''
```

pass

[\[docs\]](#)

def

canonicalize_internal_url

(
self

,

url

,

is_link

=

True

):

```
'''  
  
    Return a set of canonical representations of ``url``. The default  
  
    implementation uses just the server hostname and path of the URL,  
  
    ignoring any query parameters, fragments, etc. The canonical  
  
    representations must be unique across all URLs for this news source. If  
  
    they are not, then internal links may be resolved incorrectly.  
  
    '''
```

*:param is_link: Is True if the URL is coming from an internal link in
an HTML file. False if the URL is the URL used to
download an article.*

'''

try

:

parts

=

urlparse

(

url

)

except

Exception

:

self

.

log

.

error

(

'Failed to parse url:

%r

, ignoring'

%

url

)

```
return
```

```
frozenset
```

```
()
```

```
nl
```

```
=
```

```
parts
```

```
.
```

```
netloc
```

```
path
```

```
=
```

```
parts
```

```
.
```

```
path
```

```
or
```

```
"
```

```
if
```

```
isinstance
```

```
(
```

```
nl
```

```
,
```

```
bytes
```

```
):
```

```
nl
```

```
=  
  
nl  
.  
decode  
(  
    'utf-8'  
,  
    'replace'  
)  
  
if  
  
isinstance  
(  
    path  
,  
    bytes  
):  
  
    path  
  
    =  
  
    path  
    .  
    decode  
    (  
        'utf-8'  
    ,  
        'replace'  
    )  
  
    return  
  
    frozenset  
    ({
```

```
nl
',
path
.
rstrip
(
'/'
)))
```

[\[docs\]](#)

def

```
index_to_soup
(
self
,
url_or_raw
,
raw
=
False
,
as_tree
=
False
,
save_raw
=
None
):
```

```
'''
```

Convenience method that takes an URL to the index page and returns

a BeautifulSoup <<https://www.crummy.com/software/BeautifulSoup/bs4/doc>>`__

of it.

``url_or_raw``: Either a URL or the downloaded index page as a string

'''

if

re

.

match

((

br

'\w+:/'

if

isinstance

(

url_or_raw

,

bytes

)

else

r

'\w+:/'

),

url_or_raw

):

We may be called in a thread (in the skip_ad_pages method), so

clone the browser to be safe. We cannot use self.cloned_browser

as it may or may not actually clone the browser, depending on if

the recipe implements get_browser() or not

br

=

self

.

clone_browser

(

self

.

browser

)

open_func

=

getattr

(

br

,

'open_novisit'

,

br

.

open

)

with

closing

```
(
    open_func
    (
        url_or_raw
        ,

        timeout
        =
        self
        .
        timeout
    ))

as

f
:

    _raw

    =

    f
    .
    read
    ()

if

not

    _raw
    :

raise

RuntimeError
(
    'Could not fetch index from
    %s
```



```
|  
%  
url_or_raw  
)
```

```
else  
:
```

```
_raw
```

```
=
```

```
url_or_raw
```

```
if
```

```
raw  
:
```

```
return
```

```
_raw
```

```
if
```

```
not
```

```
isinstance
```

```
(  
_raw  
,
```

```
str  
)
```

```
and
```

```
self
```

```
.  
encoding  
:
```

```
if
```

```
callable  
(  
    self  
    .  
    encoding  
):
```

```
    _raw
```

```
    =
```

```
    self
```

```
    .  
    encoding  
    (  
        _raw  
    )
```

```
else
```

```
:
```

```
    _raw
```

```
    =
```

```
    _raw
```

```
    .  
    decode  
    (  
        self
```

```
    .  
    encoding  
    ,
```

```
'replace'
)

from
calibre.ebooks.chardet

import

(

strip_encoding_declarations
,

xml_to_unicode

)

from
calibre.utils.cleantext

import

clean_xml_chars

if

isinstance
(
_raw
,

str
):
```

```
_raw  
  
=  
  
strip_encoding_declarations  
(  
_raw  
)
```

```
else  
:
```

```
_raw  
  
=  
  
xml_to_unicode  
(  
_raw  
,  
  
strip_encoding_pats  
=  
True  
,  
  
resolve_entities  
=  
True  
)[  
0  
]
```

```
_raw  
  
=  
  
clean_xml_chars  
(  
_raw
```

```
)
```

```
if
```

```
save_raw
```

```
:
```

```
with
```

```
lopen
```

```
(
```

```
save_raw
```

```
,
```

```
'wb'
```

```
)
```

```
as
```

```
f
```

```
:
```

```
f
```

```
.
```

```
write
```

```
(
```

```
_raw
```

```
.
```

```
encode
```

```
(
```

```
'utf-8'
```

```
))
```

```
if
```

```
as_tree
```

```
:
```

```
from
```

```
html5_parser
```

```
import
```

```
parse
```

```
return
```

```
parse
```

```
(  
_raw  
)
```

```
return
```

```
BeautifulSoup
```

```
(  
_raw  
)
```

```
\[docs\]
```

```
def
```

```
extract_readable_article
```

```
(  
self  
,
```

```
html
```

```
,
```

```
url
```

```
):
```

```
'''
```

Extracts main article content from 'html', cleans up and returns as a (article_html, extracted_title) tuple.

Based on the original readability algorithm by Arc90.

```
'''  
  
from  
  
lxml.html  
  
import  
  
document_fromstring  
,  
  
fragment_fromstring  
,  
  
tostring  
  
from  
  
calibre.ebooks.readability  
  
import  
  
readability  
  
doc  
  
=  
  
readability  
.  
Document  
(  
html
```

```
,  
  
self  
.  
log  
,  
  
url  
=  
url  
,  
  
keep_elements  
=  
self  
.  
auto_cleanup_keep  
)  
  
article_html  
  
=  
  
doc  
.  
summary  
()  
  
extracted_title  
  
=  
  
doc  
.  
title  
()  
  
try
```



```
:  
  
frag  
  
=  
  
fragment_fromstring  
(  
    article_html  
)
```

```
except  
:
```

```
doc  
  
=  
  
document_fromstring  
(  
    article_html  
)
```

```
frag  
  
=  
  
doc  
.  
xpath  
(  
    '//body'  
)[  
    -  
    1  
]
```

```
if
```

```
frag
```

```
.
```

```
tag
```

```
==
```

```
'html'
```

```
:
```

```
root
```

```
=
```

```
frag
```

```
elif
```

```
frag
```

```
.
```

```
tag
```

```
==
```

```
'body'
```

```
:
```

```
root
```

```
=
```

```
document_fromstring
```

```
(
```

```
'<html><head><title>
```

```
%s
```

```
</title></head></html>'
```

```
%
```

```
extracted_title  
)
```

```
root  
.  
append  
(  
frag  
)
```

```
else  
:
```

```
root
```

```
=
```

```
document_fromstring  
(
```

```
'<html><head><title>  
%s  
</title></head><body/></html>'
```

```
%
```

```
extracted_title  
)
```

```
root  
.  
xpath  
(  
'//body'  
)[
```

```
0
]
.
append
(
frag
)

body

=

root
.
xpath
(
'//body'
)[
0
]

has_title

=

False

for

x

in

body
.
iterdescendants
():
```

```
if
x
.
text

==

extracted_title
:

has_title

=

True

inline_titles

=

body
.
xpath
(
'//h1|//h2'
)

if
not
has_title
and
not
inline_titles
:
```

heading

=

body

.

makeelement

(

'h2'

)

heading

.

text

=

extracted_title

body

.

insert

(

0

,

heading

)

raw_html

=

toString

(

root

,

```
encoding
=
'unicode'
)
```

```
return
```

```
raw_html
```

```
\[docs\]
```

```
def
```

```
sort_index_by
```

```
(
self
,
```

```
index
```

```
,
```

```
weights
```

```
):
```

```
"""
```

Convenience method to sort the titles in `index` according to `weights`.

`index` is sorted in place. Returns `index`.

`index`: A list of titles.

`weights`: A dictionary that maps weights to titles. If any titles

in index are not in weights, they are assumed to have a weight of 0.

```
'''  
  
weights
```

```
=
```

```
defaultdict
```

```
(  
    lambda  
    :
```

```
    0
```

```
,
```

```
weights
```

```
)
```

```
index
```

```
.
```

```
sort
```

```
(  
    key  
    =  
    lambda
```

```
    x
```

```
    :
```

```
weights
```

```
[
```

```
    x
```

```
])
```

```
return
```

```
index
```

```
\[docs\]
```



```
def
```

```
    parse_index
```

```
    (
```

```
    self
```

```
):
```

```
    """
```

This method should be implemented in recipes that parse a website instead of feeds to generate a list of articles. Typical uses are for news sources that have a "Print Edition" webpage that lists all the articles in the current print edition. If this function is implemented, it will be used in preference to :meth:`BasicNewsRecipe.parse\_feeds`.

It must return a list. Each element of the list must be a 2-element tuple of the form ``('feed title', list of articles)``.

Each list of articles must contain dictionaries of the form::

```
    {
```

```
        'title'    : article title,
```

```
        'url'      : URL of print version,
```

```
        'date'     : The publication date of the article as a string,
```

```
        'description' : A summary of the article
```

```
        'content'   : The full article (can be an empty string). Obsolete
```

do not use, instead save the content to a temporary

*file and pass a file:///path/to/temp/file.html as
the URL.*

}

For an example, see the recipe for downloading `The Atlantic`.

In addition, you can add 'author' for the author of the article.

*If you want to abort processing for some reason and have
calibre show the user a simple message instead of an error, call
:meth:`abort\_recipe\_processing`.*

'''

raise

NotImplementedError

()

[\[docs\]](#)

def

abort_recipe_processing

(

self

,

msg

):

'''

Causes the recipe download system to abort the download of this recipe,

displaying a simple feedback message to the user.

'''

from

calibre.ebooks.conversion

import

ConversionUserFeedBack

raise

ConversionUserFeedBack

(

—
(

'Failed to download

%s

,

)

%

self

.

title

,

msg

)

[\[docs\]](#)

def

get_obfuscated_article

(

self

```
,
```

```
url
```

```
):
```

```
'''
```

If you set `articles\_are\_obfuscated` this method is called with every article URL. It should return the path to a file on the filesystem that contains the article HTML. That file is processed by the recursive HTML fetching engine, so it can contain links to pages/images on the web.

This method is typically useful for sites that try to make it difficult to access article content automatically.

```
'''
```

```
raise
```

```
NotImplementedError
```

```
()
```

```
\[docs\]
```

```
def
```

```
add_toc_thumbnail
```

```
(
```

```
self
```

```
,
```

```
article
```

```
,
```

```
src
```

```
):
```

```
'''
```

```
Call this from populate_article_metadata with the src attribute of an  
<img> tag from the article that is appropriate for use as the thumbnail  
representing the article in the Table of Contents. Whether the  
thumbnail is actually used is device dependent (currently only used by  
the Kindles). Note that the referenced image must be one that was  
successfully downloaded, otherwise it will be ignored.
```

```
'''
```

```
if
```

```
not
```

```
src
```

```
or
```

```
not
```

```
hasattr
```

```
(
```

```
article
```

```
,
```

```
'toc_thumbnail'
```

```
):
```

```
return
```

```
src

=

src
.
replace
(
'
\\
'
,
'/'
)

if

re
.
search
(
r
'feed_\d+/article_\d+/images/img'
,

src
,

flags
=
re
.
I
)

is

None
:
```

```
self
.
log
.
warn
(
'Ignoring invalid TOC thumbnail image:
%r'
,
%
src
)

return

article
.
toc_thumbnail
=

re
.
sub
(
r
'^.*?feed'
,

'feed'
,

src
,

flags
=
re
.
IGNORECASE
```

```
)
```

[\[docs\]](#)

```
def
```

```
    populate_article_metadata
```

```
    (
```

```
        self
```

```
        ,
```

```
        article
```

```
        ,
```

```
        soup
```

```
        ,
```

```
        first
```

```
    ):
```

```
    """
```

Called when each HTML page belonging to article is downloaded.

Intended to be used to get article metadata like author/summary/etc.

from the parsed HTML (soup).

:param article: A object of class :class:`calibre.web.feeds.Article`.

If you change the summary, remember to also change the text_summary

:param soup: Parsed HTML belonging to this article

:param first: True iff the parsed HTML is the first page of the article.

```
    """
```



```
pass
```

```
\[docs\]
```

```
def
```

```
postprocess_book
```

```
(
```

```
self
```

```
,
```

```
oeb
```

```
,
```

```
opts
```

```
,
```

```
log
```

```
):
```

```
'''
```

```
Run any needed post processing on the parsed downloaded e-book.
```

```
:param oeb: An OEBook object
```

```
:param opts: Conversion options
```

```
'''
```

```
pass
```

```
def
```

```
\_\_init\_\_
```

```
(
```

```
self
```

```
,
```

```

options
,

log
,

progress_reporter
):

'''

    Initialize the recipe.

    :param options: Parsed commandline options

    :param log: Logging object

    :param progress_reporter: A Callable that takes two arguments: progress (a number
between 0 and 1) and a string message. The message should be optional.

'''

self
.
log
=

ThreadSafeWrapper
(
log
)

if

not

isinstance
(

```

```
self
.
title
,

str
):

self
.
title

=

str
(
self
.
title
,

'utf-8'
,

'replace'
)

self
.
debug

=

options
.
verbose

>

1
```

```
self
.  
output_dir
```

```
=  
  
os  
.  
path  
.  
abspath  
(  
os  
.  
getcwd  
())
```

```
self  
.  
verbose
```

```
=  
  
options  
.  
verbose
```

```
self  
.  
test
```

```
=  
  
options  
.  
test
```

```
if
```

```
self
.
test

and

not

isinstance
(
self
.
test
,
tuple
):

self
.
test

=

(
2
,
2
)

self
.
username

=

options
.
username
```

```
self
```

```
.
```

```
password
```

```
=
```

```
options
```

```
.
```

```
password
```

```
self
```

```
.
```

```
lrf
```

```
=
```

```
options
```

```
.
```

```
lrf
```

```
self
```

```
.
```

```
output_profile
```

```
=
```

```
options
```

```
.
```

```
output_profile
```

```
self
```

```
.
```

```
touchscreen
```

```
=
```

```
getattr
```

```
(
```

```
self
    .
    output_profile
    ,
    'touchscreen'
    ,
    False
)

if

self
    .
    touchscreen
    :

self
    .
    template_css
    +=

self
    .
    output_profile
    .
    touchscreen_news_css

if

self
    .
    test
    :

self
```

```
.
max_articles_per_feed

=

self
.
test
[
1
]

self
.
simultaneous_downloads

=

min
(
4
,
self
.
simultaneous_downloads
)

if
self
.
debug
:

self
.
verbose
```


=

True

self

.

report_progress

=

progress_reporter

if

self

.

needs_subscription

and

(

self

.

username

is

None

or

self

.

password

is

None

or

(

not

self

.

username

and

not

self

.

password

)):

if

self

.

needs_subscription

!=

'optional'

:

raise

ValueError

(

—

(

'The "

%s

" recipe needs a username and password.'

```
)  
%  
self  
.  
title  
)  
  
self  
.  
browser  
  
=  
  
self  
.  
get_browser  
()  
  
self  
.  
image_map  
,  
  
self  
.  
image_counter  
  
=  
  
{},  
  
1  
  
self  
.  
css_map  
  
=
```

```
}
```

```
web2disk_cmdline
```

```
=
```

```
[  
  'web2disk'
```

```
,
```

```
  '--timeout'
```

```
,
```

```
  str  
  (  
    self  
    .  
    timeout  
  ),
```

```
  '--max-recursions'
```

```
,
```

```
  str  
  (  
    self  
    .  
    recursions  
  ),
```

```
  '--delay'
```

```
,
```

```
  str  
  (  
    self  
    .  
    delay
```

```
),
```

```
]
```

```
if
```

```
self
```

```
.
```

```
verbose
```

```
:
```

```
web2disk_cmdline
```

```
.
```

```
append
```

```
(
```

```
'--verbose'
```

```
)
```

```
if
```

```
self
```

```
.
```

```
no_stylesheets
```

```
:
```

```
web2disk_cmdline
```

```
.
```

```
append
```

```
(
```

```
'--dont-download-stylesheets'
```

```
)
```

```
for
```

```
reg

in

self
.
match_regexps
:

web2disk_cmdline
.
extend
([
'--match-regex'
,
reg
])

for

reg

in

self
.
filter_regexps
:

web2disk_cmdline
.
extend
([
'--filter-regex'
,
reg
])
```

```
if
    options
    .
    output_profile
    .
    short_name

in

(
'default'
,
'tablet'
):

self
.
scale_news_images_to_device

=

False

elif

self
.
scale_news_images_to_device
:

self
.
scale_news_images

=
```

```
options
```

```
.
```

```
output_profile
```

```
.
```

```
screen_size
```

```
self
```

```
.
```

```
web2disk_options
```

```
=
```

```
web2disk_option_parser
```

```
()
```

```
.
```

```
parse_args
```

```
(
```

```
web2disk_cmdline
```

```
)[
```

```
0
```

```
]
```

```
for
```

```
extra
```

```
in
```

```
(
```

```
'keep_only_tags'
```

```
,
```

```
'remove_tags'
```

```
,
```

```
'preprocess_regexp'
```

```
,
```



```
'skip_ad_pages'
,

'preprocess_html'
,

'remove_tags_after'
,

'remove_tags_before'
,

'is_link_wanted'
,

'compress_news_images'
,

'compress_news_images_max_size'
,

'compress_news_images_auto_size'
,

'scale_news_images'
):
```

```
setattr
(
self
.
web2disk_options
,

extra
,

getattr
(
```

```
self
```

```
,
```

```
extra
```

```
))
```

```
self
```

```
.
```

```
web2disk_options
```

```
.
```

```
postprocess_html
```

```
=
```

```
self
```

```
.
```

```
_postprocess_html
```

```
self
```

```
.
```

```
web2disk_options
```

```
.
```

```
preprocess_image
```

```
=
```

```
self
```

```
.
```

```
preprocess_image
```

```
self
```

```
.
```

```
web2disk_options
```

```
.
```

```
encoding
```

```
=
```

```
self
```

```
.
encoding

self
.
web2disk_options
.
preprocess_raw_html

=

self
.
preprocess_raw_html_

if

self
.
delay

>

0
:

self
.
simultaneous_downloads

=

1

self
.
navbar
```

```
=  
  
templates  
.  
TouchscreenNavBarTemplate  
(  
  
    if  
  
        self  
        .  
touchscreen  
  
    else  
    \  
  
templates  
.  
NavBarTemplate  
(  
  
    self  
    .  
failed_downloads  
  
    =  
  
    []  
  
    self  
    .  
partial_failures  
  
    =  
  
    []  
  
    def
```

```
_postprocess_html
```

```
(
```

```
self
```

```
,
```

```
soup
```

```
,
```

```
first_fetch
```

```
,
```

```
job_info
```

```
):
```

```
if
```

```
self
```

```
.
```

```
no_stylesheets
```

```
:
```

```
for
```

```
link
```

```
in
```

```
soup
```

```
.
```

```
findAll
```

```
(
```

```
'link'
```

```
):
```

```
if
```

```
(
```

```
link
```

```
.
```

```
get
(
  'type'
)

or

'text/css'
)
.
lower
()

==

'text/css'

and

'stylesheet'

in

(
link
.
get
(
  'rel'
)

or

(
  'stylesheet'
,)):

link
.
extract
()
```

for

style

in

soup

.

findAll

(

'style'

):

style

.

extract

()

head

=

soup

.

find

(

'head'

)

if

not

head

:

head

```
=  
  
soup  
.find  
(  
'body'  
)  
  
if  
  
not  
  
head  
:  
  
head  
  
=  
  
soup  
.find  
(  
True  
)  
  
css  
  
=  
  
self  
.template_css  
  
+  
  
'  
\n\n'  
,
```



```
+  
  
(  
self  
.  
get_extra_css  
)  
  
or  
  
"  
)  
  
style  
  
=  
  
soup  
.  
new_tag  
(  
'style'  
,  
  
type  
=  
'text/css'  
,  
  
title  
=  
'override_css'  
)  
  
style  
.  
append  
(  
css  
)
```

```
head
```

```
.
```

```
append
```

```
(
```

```
style
```

```
)
```

```
if
```

```
first_fetch
```

```
and
```

```
job_info
```

```
:
```

```
url
```

```
,
```

```
f
```

```
,
```

```
a
```

```
,
```

```
feed_len
```

```
=
```

```
job_info
```

```
body
```

```
=
```

```
soup
```

```
.
```

```
find
```

```
(  
'body'  
)
```

```
if
```

```
body
```

```
is
```

```
not
```

```
None
```

```
:
```

```
templ
```

```
=
```

```
self
```

```
.
```

```
navbar
```

```
.
```

```
generate
```

```
(
```

```
False
```

```
,
```

```
f
```

```
,
```

```
a
```

```
,
```

```
feed_len
```

```
,
```

```
not
```

```
self
```

```
.  
has_single_feed
```

```
,
```

```
url
```

```
,
```

```
__appname__
```

```
,
```

```
center
```

```
=
```

```
self
```

```
.
```

```
center_navbar
```

```
,
```

```
extra_css
```

```
=
```

```
self
```

```
.
```

```
get_extra_css
```

```
()
```

```
or
```

```
''
```

```
)
```

```
elem
```

```
=
```

```
BeautifulSoup
```

```
(
```

```
templ
```

```
.
```

```
render
```

```
(
```

```
doctype
=
'xhtml'
)
.
decode
(
'utf-8'
))
.
find
(
'div'
)

body
.
insert
(
0
,
elem
)

# This is needed because otherwise inserting elements into

# the soup breaks find()

soup
=
BeautifulSoup
(
soup
.
decode_contents
())
```

```
if
```

```
self
```

```
.  
remove_javascript  
:
```

```
for
```

```
script
```

```
in
```

```
list  
(  
soup  
.findAll  
(  
'script'  
)):
```

```
script  
.extract  
()
```

```
for
```

```
o
```

```
in
```

```
soup  
.findAll  
(  
onload
```

```
=
True
):

del

o
[
'onload'
]

for

attr

in

self
.
remove_attributes
:

for

x

in

soup
.
findAll
(
attrs
=
{
attr
:
True
}):
```

del

```
x  
[  
  attr  
]
```

for

bad_tag

in

```
list  
(  
  soup  
  .  
  findAll  
  ([  
    'base'  
    ,  
    'iframe'  
    ,  
    'canvas'  
    ,  
    'embed'  
    ,  
    'command'  
    ,  
    'datalist'  
    ,  
    'video'  
    ,
```



```
'audio'  
,  
'noscript'  
,  
'link'  
,  
'meta'  
]]):
```

link tags can be used for preloading causing network activity in

calibre viewer. meta tags can do all sorts of crazy things,

including http-equiv refresh, viewport shenanigans, etc.

```
bad_tag  
.  
extract  
()
```

srcset causes some viewers, like calibre's to load images from the

web, and it also possible causes iBooks on iOS to barf, see

<https://bugs.launchpad.net/bugs/1713986>

for

img

in

```
soup
.
findAll
(
'img'
,
srcset
=
True
):

del

img
[
'srcset'
]

ans

=

self
.
postprocess_html
(
soup
,
first_fetch
)

# Nuke HTML5 tags

for
```

```
x
```

```
in
```

```
ans
```

```
.
```

```
findAll
```

```
([
```

```
'article'
```

```
,
```

```
'aside'
```

```
,
```

```
'header'
```

```
,
```

```
'footer'
```

```
,
```

```
'nav'
```

```
,
```

```
'figcaption'
```

```
,
```

```
'figure'
```

```
,
```

```
'section'
```

```
]):
```

```
x
```

```
.
```

```
name
```

```
=
```

```
'div'
```

```
if
```

```
job_info
```

```
:
```

```
url
```

```
,
```

```
f
```

```
,
```

```
a
```

```
,
```

```
feed_len
```

```
=
```

```
job_info
```

```
try
```

```
:
```

```
article
```

```
=
```

```
self
```

```
.
```

```
feed_objects
```

```
[
```

```
f
```

```
]
```

```
.
```

```
articles
```

```
[
```

```
a
```

```
]
```

```
except
:

self
.
log
.
exception
(
'Failed to get article object for postprocessing'
)

pass

else
:

self
.
populate_article_metadata
(
article
,

ans
,

first_fetch
)

return

ans
```

[\[docs\]](#)

def

download

(
self
):

"""

Download and pre-process all articles from the feeds in this recipe.

This method should be called only once on a particular Recipe instance.

Calling it more than once will lead to undefined behavior.

:return: Path to index.html

"""

try

:

res

=

self

.
build_index
()

self

.
report_progress

(

1

,

```
-  
(  
  'Download finished'  
)  
  
if  
  
  self  
  .  
  failed_downloads  
  :  
  
  self  
  .  
  log  
  .  
  warning  
  (  
    -  
    (  
      'Failed to download the following articles:'  
    ))  
  
  for  
  
    feed  
    ,  
  
    article  
    ,  
  
    debug  
  
  in  
  
  self  
  .  
  failed_downloads  
  :
```

```
self
.  
log  
.  
warning  
(  
article  
.  
title  
,  
'from'  
,  
feed  
.  
title  
)
```

```
self  
.  
log  
.  
debug  
(  
article  
.  
url  
)
```

```
self  
.  
log  
.  
debug  
(  
debug  
)
```



```
if
self
·
partial_failures
:

self
·
log
·
warning
(
  _
  (
    'Failed to download parts of the following articles:'
  )
)

for
feed
,
atitle
,
aurl
,
debug

in
self
·
partial_failures
:

self
```

```
.
log

.
warning
(
atitle

+

-
(
' from '
)

+

feed
)

self

.
log

.
debug
(
aurl
)

self

.
log

.
warning
(
-
(
'
\t
Failed links:'
))
```

for

|

,

tb

in

debug

:

self

.

log

.

warning

(

|

)

self

.

log

.

debug

(

tb

)

return

res

finally

:

```
self
.
cleanup
()

@property

def
lang_for_html
(
self
):

try
:

lang

=

self
.
language
.
replace
(
'_'
,
'_'
)
.
partition
(
'_'
)[
0
]
```

```
.  
lower  
(  
  
if  
  
lang  
  
==  
  
'und'  
:  
  
lang  
  
=  
  
None  
  
except  
:  
  
lang  
  
=  
  
None  
  
return  
  
lang  
  
def  
  
feeds2index  
(
```

```
self
',

feeds
):

templ

=

(
templates
.
TouchscreenIndexTemplate

if

self
.
touchscreen

else

templates
.
IndexTemplate
)

templ

=

templ
(
lang
=
self
.
lang_for_html
)
```

```
css
=
self
.
template_css

+

'
\n\n
'

+
(
self
.
get_extra_css
()

or

"
)

timefmt

=

self
.
timefmt

return

templ
.
generate
```

```
(
self
.
title
,

"mastheadImage.jpg"
,

timefmt
,

feeds
,

extra_css
=
css
)
.
render
(
doctype
=
'xhtml'
)
```

@classmethod

def

```
description_limiter
(
cls
,

src
):
```



```
if
not
src
:
return
"
src
=
force_unicode
(
src
,
'utf-8'
)
pos
=
cls
.
summary_length
fuzz
=
50
```

```
si
```

```
=
```

```
src
```

```
.
```

```
find
```

```
(
```

```
'.'
```

```
,
```

```
,
```

```
pos
```

```
)
```

```
if
```

```
si
```

```
>
```

```
0
```

```
and
```

```
si
```

```
-
```

```
pos
```

```
>
```

```
fuzz
```

```
:
```

```
si
```

```
=
```

```
-
```

```
1
```

```
gi
```

```
=
```

```
src
```

```
.
```

```
find
```

```
(
```

```
'>'
```

```
,
```

```
pos
```

```
)
```

```
if
```

```
gi
```

```
>
```

```
0
```

```
and
```

```
gi
```

```
-
```

```
pos
```

```
>
```

```
fuzz
```

```
:
```

```
gi
```

```
=
```

```
-
```

```
1
```

```
npos
```

```
=
```

```
max
```

```
(
```

```
si
```

```
,
```

```
gi
```

```
)
```

```
if
```

```
npos
```

```
<
```

```
0
```

```
:
```

```
npos
```

```
=
```

```
pos
```

```
ans
```

```
=
```

```
src
```

```
[:
```

```
npos
```

```
+
```

```
1
```

```
]
```

```
if
```

```
len  
(  
ans  
)
```

```
<
```

```
len  
(  
src  
):
```

```
from
```

```
calibre.utils.cleantext
```

```
import
```

```
clean_xml_chars
```

```
# Truncating the string could cause a dangling UTF-16 half-surrogate, which will cause lxml to  
barf, clean it
```

```
ans
```

```
=
```

```
clean_xml_chars  
(  
ans  
)
```

```
+
```

```
,
```

```
\u2026  
,
```

```
return
```

```
ans
```

```
def
```

```
feed2index
```

```
(
```

```
self
```

```
,
```

```
f
```

```
,
```

```
feeds
```

```
):
```

```
feed
```

```
=
```

```
feeds
```

```
[
```

```
f
```

```
]
```

```
if
```

```
feed
```

```
.
```

```
image_url
```

```
is
```

```
not
```

```
None
```

```
:
```

```
# Download feed image
```

```
imgdir
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
join
```

```
(
```

```
self
```

```
.
```

```
output_dir
```

```
,
```

```
'images'
```

```
)
```

```
if
```

```
not
```

```
os
```

```
.
```

```
path
```

```
.
```

```
isdir
```

```
(
```

```
imgdir
```

```
):
```

```
os
```

```
.
```

```
makedirs
```

```
(
```

```
imgdir
```

```
)
```

```
if
```

```
    feed
```

```
    .
```

```
    image_url
```

```
in
```

```
    self
```

```
    .
```

```
    image_map
```

```
    :
```

```
    feed
```

```
    .
```

```
    image_url
```

```
    =
```

```
    self
```

```
    .
```

```
    image_map
```

```
    [
```

```
    feed
```

```
    .
```

```
    image_url
```

```
    ]
```

```
else
```

```
    :
```

```
    bn
```

```
    =
```

```
    urlsplit
```

```
    (
```



```
feed
    .
    image_url
    )
    .
    path
```

```
if
```

```
    bn
    :
```

```
    bn
```

```
    =
```

```
    bn
```

```
    .
    rpartition
    (
    '/'
    )[
    -
    1
    ]
```

```
if
```

```
    bn
    :
```

```
    img
```

```
    =
```

```
    os
```

```
    .
    path
```

```
    .
```

```
join
(
imgdir
,

'feed_image_
%d%s'
,
%
(
self
.
image_counter
,

os
.
path
.
splitext
(
bn
)))

try
:

with

open
(
img
,

'wb'
)

as

fi
,
```

```
closing
(
self
.
browser
.
open
(
feed
.
image_url
,

timeout
=
self
.
timeout
))

as

r
:

fi
.
write
(
r
.
read
())

self
.
image_counter
+=
```

```
1

feed
.
image_url

=

img

self
.
image_map
[
feed
.
image_url
]

=

img

except
:

pass

if

isinstance
(
feed
.
image_url
,
bytes
```

```
):  
  
feed  
.  
image_url  
  
=  
  
feed  
.  
image_url  
.  
decode  
(  
sys  
.  
getfilesystemencoding  
()),  
  
'strict'  
)  
  
templ  
  
=  
  
(  
templates  
.  
TouchscreenFeedTemplate  
  
if  
  
self  
.  
touchscreen  
  
else
```

```
templates
.  
FeedTemplate  
)
```

```
templ
```

```
=
```

```
templ  
(  
lang  
=  
self  
.  
lang_for_html  
)
```

```
css
```

```
=
```

```
self  
.  
template_css
```

```
+
```

```
'  
\n\n'
```

```
+
```

```
(  
self  
.  
get_extra_css  
)
```

```
or
```

```
"
)

return

templ
.
generate
(
f
,

feeds
,

self
.
description_limiter
,

extra_css
=
css
)
.
render
(
doctype
=
'xhtml'
)

def

_fetch_article
(
self
,
```

```
url
,

dir_
,

f
,

a
,

num_of_feeds
):

br

=

self
.
browser

if

hasattr
(
self
.
get_browser
,

'is_base_class_implementation'
):

# We are using the default get_browser, which means no need to

# clone
```


br

=

BasicNewsRecipe

.

get_browser

(

self

)

else

:

br

=

self

.

clone_browser

(

self

.

browser

)

self

.

web2disk_options

.

browser

=

br

```
fetcher

=

RecursiveFetcher
(
  self
  .
  web2disk_options
  ,

  self
  .
  log
  ,

  self
  .
  image_map
  ,

  self
  .
  css_map
  ,

  (
    url
    ,

    f
    ,

    a
    ,

    num_of_feeds
  ))

fetcher
```

```
.
browser

=

br

fetcher

.
base_dir

=

dir_

fetcher

.
current_dir

=

dir_

fetcher

.
show_progress

=

False

fetcher

.
image_url_processor

=

self

.
```

```
image_url_processor
```

```
res
```

```
,
```

```
path
```

```
,
```

```
failures
```

```
=
```

```
fetcher
```

```
.
```

```
start_fetch
```

```
(
```

```
url
```

```
),
```

```
fetcher
```

```
.
```

```
downloaded_paths
```

```
,
```

```
fetcher
```

```
.
```

```
failed_links
```

```
if
```

```
not
```

```
res
```

```
or
```

```
not
```

```
os
```

```
.
```

```
path
```

```
.
exists
(
res
):

msg

=

-
(
'Could not fetch article.'
)

+

''

if

self

.
debug
:

msg

+=

-
(
'The debug traceback is available earlier in this log'
)

else
:
```

```
msg
```

```
+=
```

```
-
```

```
(
```

```
'Run with -vv to see the reason'
```

```
)
```

```
raise
```

```
Exception
```

```
(
```

```
msg
```

```
)
```

```
return
```

```
res
```

```
,
```

```
path
```

```
,
```

```
failures
```

```
def
```

```
fetch_article
```

```
(
```

```
self
```

```
,
```

```
url
```

```
,
```

```
dir
```

```
,
```

```
f
',

a
',

num_of_feeds
):
```

```
return
```

```
self
.
_fetch_article
(
url
,

dir
,

f
,

a
,

num_of_feeds
)
```

```
def
```

```
fetch_obfuscated_article
(
self
,

url
,
```

```
dir
,

f
,

a
,

num_of_feeds
):

path

=

os
.
path
.
abspath
(
self
.
get_obfuscated_article
(
url
))

url

=

(
'file:'
+
path
)

if
```



```
iswindows
```

```
else
```

```
(  
    'file:/'  
    +  
    path  
)
```

```
return
```

```
self  
.  
_fetch_article  
(  
    url  
    ,  
    dir  
    ,  
    f  
    ,  
    a  
    ,  
    num_of_feeds  
)
```

```
def
```

```
fetch_embedded_article  
(  
    self  
    ,  
    article
```

```
,  
  
dir  
,  
  
f  
,  
  
a  
,  
  
num_of_feeds  
):  
  
templ  
  
=  
  
templates  
.  
EmbeddedContent  
()  
  
raw  
  
=  
  
templ  
.  
generate  
(  
    article  
)  
.  
render  
(  
    'html'  
)  
  
with
```

```
PersistentTemporaryFile
```

```
(  
    '_feeds2disk.html'  
)
```

```
as
```

```
pt  
:
```

```
pt  
.  
write  
(  
    raw  
)
```

```
url
```

```
=
```

```
(  
    'file:'  
    +  
    pt  
    .  
    name  
)
```

```
if
```

```
iswindows
```

```
else
```

```
(  
    'file:/'  
    +  
    pt  
    .
```

```
name
)

return

self
.
_fetch_article
(
url
,

dir
,

f
,

a
,

num_of_feeds
)

def

remove_duplicate_articles
(
self
,

feeds
):

seen_keys

=

defaultdict
```

```
(
set
)

remove

=

[]

for

f

in

feeds
:

for

article

in

f
:

for

key

in

self
.
ignore_duplicate_articles
:
```

```
val
=
getattr
(
article
,
key
)

seen
=
seen_keys
[
key
]

if
val
:

if
val
in
seen
:

remove
.
append
```

```
((
f
,

article
))

else
:

seen
.
add
(
val
)

for

feed
,

article

in

remove
:

self
.
log
.
debug
(
'Removing duplicate article:
%s
from section:
```

```
%s
',
%
(

article
.
title
',

feed
.
title
))

feed
.
remove_article
(
article
)

if

self
.
remove_empty_feeds
:

feeds

=

[
f

for
```



```
f
in
feeds
if
len
(
f
)
>
0
]

return
feeds

def
build_index
(
self
):

self
.
report_progress
(
0
,

—
(
'Fetching feeds...'

```

```
))
```

```
feeds
```

```
=
```

```
None
```

```
try
```

```
:
```

```
feeds
```

```
=
```

```
feeds_from_index
```

```
(
```

```
self
```

```
.
```

```
parse_index
```

```
()),
```

```
oldest_article
```

```
=
```

```
self
```

```
.
```

```
oldest_article
```

```
,
```

```
max_articles_per_feed
```

```
=
```

```
self
```

```
.
```

```
max_articles_per_feed
```

```
,
```

```
log
```

```
=
```

```
self
.
log
)

self
.
report_progress
(
0
,

—
(
'Got feeds from index page'
))

except

NotImplementedError
:

pass

if

feeds

is

None
:

feeds

=
```

```
self
.  
parse_feeds  
(  
  
  
if  
  
not  
  
feeds  
:  
  
raise  
  
ValueError  
(  
    'No articles found, aborting'  
)  
  
  
if  
  
self  
.  
ignore_duplicate_articles  
  
is  
  
not  
  
None  
:  
  
feeds  
  
=  
  
self
```

```
.
remove_duplicate_articles
(
feeds
)
```

```
self
```

```
.
report_progress
(
0
,
_
(
'Trying to download cover...'
))
```

```
self
```

```
.
download_cover
()
```

```
self
```

```
.
report_progress
(
0
,
_
(
'Generating masthead...'
))
```

```
self
```

```
.
resolve_masthead
```

```
()

if
self
.
test
:

feeds

=

feeds
[:
self
.
test
[
0
]]

self
.
has_single_feed

=

len
(
feeds
)

==

1
```

```
index

=

os
.
path
.
join
(
self
.
output_dir
,

'index.html'
)
```

```
html

=

self
.
feeds2index
(
feeds
)
```

with

```
open
(
index
,

'wb'
)
```

as

```
fi
```

```
:
```

```
fi
```

```
.
```

```
write
```

```
(
```

```
html
```

```
)
```

```
self
```

```
.
```

```
jobs
```

```
=
```

```
[]
```

```
if
```

```
self
```

```
.
```

```
reverse_article_order
```

```
:
```

```
for
```

```
feed
```

```
in
```

```
feeds
```

```
:
```

```
if
```



```
hasattr
(
feed
,

'reverse'
):

feed
.
reverse
()

self
.
feed_objects
=

feeds

for

f
,

feed

in

enumerate
(
feeds
):

feed_dir
```

```
=  
  
os  
.path  
.join  
(  
self  
.output_dir  
,  
  
'feed_  
%d'  
,  
%  
f  
)  
  
if  
  
not  
  
os  
.path  
.isdir  
(  
feed_dir  
):  
  
os  
.mkdirs  
(  
feed_dir  
)
```

```
for
```

```
    a
```

```
    ,
```

```
    article
```

```
in
```

```
    enumerate
```

```
    (
```

```
    feed
```

```
):
```

```
if
```

```
    a
```

```
    >=
```

```
    self
```

```
    .
```

```
    max_articles_per_feed
```

```
    :
```

```
break
```

```
    art_dir
```

```
    =
```

```
    os
```

```
    .
```

```
    path
```

```
    .
```

```
    join
```

```
    (
```

```
    feed_dir
```

```
    ,
```

```
'article_  
%d'  
,  
%  
a  
)  
  
if  
  
not  
  
os  
.  
path  
.  
isdir  
(  
art_dir  
):  
  
os  
.  
mkdirs  
(  
art_dir  
)  
  
try  
:  
  
url  
  
=  
  
self  
.  
print_version  
(
```

```
article
    .
    url
)

except

NotImplementedError
:

url

=

article
    .
    url

except
:

self

    .
    log

    .
    exception
    (
    'Failed to find print version for: '
    +
    article
    .
    url
    )

url

=
```

None

if

not

url
:

continue

func

,

arg

=

(

self

.

fetch_embedded_article

,

article

)

\

if

self

.

use_embedded_content

or

(

self

.

use_embedded_content

is

None

and

feed

.

has_embedded_content

(())

\

else

\

((

self

.

fetch_obfuscated_article

if

self

.

articles_are_obfuscated

else

self

.

fetch_article

),

url

)

req

=

```
WorkRequest
(
func
,

(
arg
,

art_dir
,

f
,

a
,

len
(
feed
)),

{}),

(
f
,

a
),

self
.
article_downloaded
,

self
.
error_in_article_download
)
```



```
req
.  
feed
```

```
=  
  
feed
```

```
req  
.  
article
```

```
=  
  
article
```

```
req  
.  
feed_dir
```

```
=  
  
feed_dir
```

```
self  
.  
jobs  
.  
append  
(  
req  
)
```

```
self  
.  
jobs_done
```

```
=  
  
0  
  
tp  
  
=  
  
ThreadPool  
(  
self  
.  
simultaneous_downloads  
)  
  
for  
req  
in  
self  
.  
jobs  
:  
  
    tp  
    .  
    putRequest  
    (  
    req  
    ,  
  
    block  
    =  
    True  
    ,  
  
    timeout
```

```
=
0
)

self
.
report_progress
(
0
,

gettext
(

'Starting download in a single thread...'
,

'Starting download [
{}
threads]...'
,

self
.
simultaneous_downloads
)
.
format
(
self
.
simultaneous_downloads
))

while

True
:
```

```
try
```

```
:
```

```
    tp
```

```
    .
```

```
    poll
```

```
    ()
```

```
    time
```

```
    .
```

```
    sleep
```

```
    (
```

```
    0.1
```

```
    )
```

```
except
```

```
NoResultsPending
```

```
:
```

```
break
```

```
for
```

```
    f
```

```
    ,
```

```
    feed
```

```
in
```

```
    enumerate
```

```
    (
```

```
    feeds
```

```
    );
```

```
html
```

```
=
```

```
self
```

```
.
```

```
feed2index
```

```
(
```

```
f
```

```
,
```

```
feeds
```

```
)
```

```
feed_dir
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
join
```

```
(
```

```
self
```

```
.
```

```
output_dir
```

```
,
```

```
'feed_
```

```
%d
```

```
'
```

```
%
```

```
f
```

```
)
```

```
with
```

```
open
```

```
(
```

```
os
.
path
.
join
(
feed_dir
,

'index.html'
),

'wb'
)

as

fi
:

fi
.
write
(
html
)

self
.
create_opf
(
feeds
)

self
.
report_progress
(
1
,
```

```
-  
(  
    'Feeds downloaded to  
    %s  
,  
)  
    %  
index  
)
```

return

index

def

_download_cover
(
 self
):

self

.
cover_path

=

None

try
:

cu

=

```
self
.
get_cover_url
()

except
Exception

as

err
:

self
.
log
.
error
(
-
(
'Could not download cover:
%s
'
)
%
as_unicode
(
err
))

self
.
log
.
debug
(
traceback
```



```
.  
format_exc  
(())
```

```
else  
:
```

```
if
```

```
not
```

```
cu  
:
```

```
return
```

```
cdata
```

```
=
```

```
None
```

```
if
```

```
hasattr  
(  
cu  
,
```

```
'read'  
):
```

```
cdata
```

```
=
```

```
cu
```

```
.
read
()

cu

=

getattr
(
cu
,
'name'
,
'cover.jpg'
)

elif

os

.
access
(
cu
,

os
.
R_OK
):

with

open
(
cu
,
```

```
'rb'
)

as

f
:

cdata

=

f
.
read
()

else
:

self
.
report_progress
(
1
,
—
(
'Downloading cover from
%s
'
)
%
cu
)

with
```

```
closing
(  
self  
.  
browser  
.  
open  
(  
cu  
,  
  
timeout  
=  
self  
.  
timeout  
)
```

```
as
```

```
r  
:
```

```
cdata
```

```
=
```

```
r  
.  
read  
(
```

```
if
```

```
not
```

```
cdata  
:
```

```
return
```

```
ext
```

```
=
```

```
cu
```

```
.
```

```
split
```

```
(
```

```
'/'
```

```
)[
```

```
-
```

```
1
```

```
]
```

```
.
```

```
rpartition
```

```
(
```

```
''
```

```
.)
```

```
)[
```

```
-
```

```
1
```

```
]
```

```
.
```

```
lower
```

```
()
```

```
.
```

```
strip
```

```
()
```

```
if
```

```
ext
```

```
==
```

```
'pdf'
```

```
:
```

```
from
```

[calibre.ebooks.metadata.pdf](#)

import

get_metadata

stream

=

io

.

BytesIO

(

cdata

)

cdata

=

None

mi

=

get_metadata

(

stream

)

if

mi

.

cover_data

and

```
mi
.
cover_data
[
1
]:

cdata

=

mi
.
cover_data
[
1
]

if
not
cdata
:

return

if
self
.
cover_margins
[
0
]
or
```

```
self
.
cover_margins
[
1
]:

cdata

=

image_to_data
(
add_borders_to_image
(
cdata
,

left
=
self
.
cover_margins
[
0
],
right
=
self
.
cover_margins
[
0
],

top
=
self
.
cover_margins
```



```
[
1
],
bottom
=
self
.
cover_margins
[
1
],

border_color
=
self
.
cover_margins
[
2
]))

cpath

=

os
.
path
.
join
(
self
.
output_dir
,

'cover.jpg'
)
```

```
save_cover_data_to  
(  
    cdata  
    ,  
    cpath  
)
```

```
self  
.  
cover_path  
  
=  
  
cpath
```

```
def
```

```
download_cover  
(  
    self  
):
```

```
self  
.  
cover_path  
  
=  
  

```

```
None
```

```
try  
:
```

```
self  
.  
_download_cover
```

```
()

except
:

self
.
log
.
exception
(
'Failed to download cover'
)

self
.
cover_path
=

None

def
_download_masthead
(
self
,
mu
):

if
hasattr
(
mu
```

```
,  
  
'rpartition'  
):
```

```
ext
```

```
=
```

```
mu
```

```
.
```

```
rpartition
```

```
(
```

```
''
```

```
)[
```

```
-
```

```
1
```

```
]
```

```
if
```

```
'?'
```

```
in
```

```
ext
```

```
:
```

```
ext
```

```
=
```

```
"
```

```
else
```

```
:
```

```
ext
```

```
=
```

```
mu
```

```
.
```

```
name
```

```
.
```

```
rpartition
```

```
(
```

```
['
```

```
']
```

```
)[
```

```
-
```

```
1
```

```
]
```

```
ext
```

```
=
```

```
ext
```

```
.
```

```
lower
```

```
()
```

```
if
```

```
ext
```

```
else
```

```
'jpg'
```

```
mpath
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
join
```

```
(
self
.
output_dir
,

'masthead_source.'
+
ext
)

outfile

=

os
.
path
.
join
(
self
.
output_dir
,

'mastheadImage.jpg'
)

if

hasattr
(
mu
,

'read'
):

with
```

```
open  
(  
  mpath  
,  
  'wb'  
)
```

```
as
```

```
mfile  
:
```

```
mfile
```

```
.  
write  
(  
  mu
```

```
.  
read  
())
```

```
elif
```

```
os
```

```
.  
access  
(  
  mu
```

```
,
```

```
os
```

```
.  
R_OK  
):
```

```
with
```

```
open
```

```
(  
  mpath  
,  
  'wb'  
)
```

as

```
mfile  
:
```

```
mfile
```

```
.
```

```
write
```

```
(
```

```
  open
```

```
(
```

```
    mu
```

```
,
```

```
  'rb'
```

```
)
```

```
.
```

```
read
```

```
(())
```

else

```
:
```

with

```
open
```

```
(
```

```
  mpath
```

```
,
```

```
  'wb'
```

```
)
```


as

mfile

,

closing

(

self

.

browser

.

open

(

mu

,

timeout

=

self

.

timeout

))

as

r

:

mfile

.

write

(

r

.

read

()))

self

.

report_progress

(

```
1
,
-
(
'Masthead image downloaded'
))
```

```
self
.
prepare_masthead_image
(
mpath
,
outfile
)
```

```
self
.
masthead_path
```

```
=
outfile
```

```
if
os
.
path
.
exists
(
mpath
):
```

```
os
```

```
.
```

```
remove
(
    mpath
)

def

download_masthead
(
    self
,
    url
):

try
:

    self
    .
    _download_masthead
    (
        url
    )

except
:

    self
    .
    log
    .
    exception
    (
        "Failed to download supplied masthead_url"
    )
```

```
def

resolve_masthead
(
self
):
```

```
self
.
masthead_path
```

```
=
```

```
None
```

```
try
:
```

```
murl
```

```
=
```

```
self
.
get_masthead_url
()
```

```
except
:
```

```
self
```

```
.
log
```

```
.
exception
(
```

```
'Failed to get masthead url'  
)
```

```
murl
```

```
=
```

```
None
```

```
if
```

```
murl
```

```
is
```

```
not
```

```
None
```

```
:
```

```
# Try downloading the user-supplied masthead_url
```

```
# Failure sets self.masthead_path to None
```

```
self
```

```
.
```

```
download_masthead
```

```
(
```

```
murl
```

```
)
```

```
if
```

```
self
```

```
.
```

```
masthead_path
```

```
is
```

```
None
```

```
:
```

```
self
```

```
.
```

```
log
```

```
.
```

```
info
```

```
(
```

```
"Synthesizing mastheadImage"
```

```
)
```

```
self
```

```
.
```

```
masthead_path
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
join
```

```
(
```

```
self
```

```
.
```

```
output_dir
```

```
,
```

```
'mastheadImage.jpg'
```

```
)
```

```
try
```

```
:
```

```
self
```

```
.
default_masthead_image
(
self
.
masthead_path
)

except
:

self
.
log
.
exception
(
'Failed to generate default masthead image'
)

self
.
masthead_path
=
None

\[docs\]

def
default_cover
(
self
,

```

```
cover_file
):

'''
    Create a generic cover for recipes that don't have a cover
'''

try
:

from
calibre.ebooks.covers

import
create_cover

title

=

self
.
title

if

isinstance
(
self
.
title
,
str
)
)
```



```
else
\

self
.
title
.
decode
(
preferred_encoding
,

'replace'
)

date

=

strftime
(
self
.
timefmt
)
.
replace
(
'['
,
"
)
.
replace
(
']'
,
"
)
```

```
img_data

=

create_cover
(
title
,

[
date
])

cover_file
.
write
(
img_data
)

cover_file
.
flush
()

except
:

self
.
log
.
exception
(
'Failed to generate default cover'
)
```

```
return
```

```
False
```

```
return
```

```
True
```

```
\[docs\]
```

```
def
```

```
get_masthead_title
```

```
(
```

```
self
```

```
):
```

```
'Override in subclass to use something other than the recipe title'
```

```
return
```

```
self
```

```
.
```

```
title
```

```
MI_WIDTH
```

```
=
```

```
600
```

```
MI_HEIGHT
```

```
=
```

```
60
```

```
def
```

```
default_masthead_image
```

```
(
```

```
self
```

```
,
```

```
out_path
```

```
):
```

```
from
```

```
calibre.ebooks
```

```
import
```

```
generate_masthead
```

```
generate_masthead
```

```
(
```

```
self
```

```
.
```

```
get_masthead_title
```

```
()),
```

```
output_path
```

```
=
```

```
out_path
```

```
,
```

```
width
```

```
=
```

```
self
```

```
.
```

```
MI_WIDTH
```

```
,
```

```
height
=
self
.
MI_HEIGHT
)
```

```
def
```

```
prepare_masthead_image
(
self
,

path_to_image
,

out_path
):
```

```
prepare_masthead_image
(
path_to_image
,

out_path
,

self
.
MI_WIDTH
,

self
.
MI_HEIGHT
)
```

```
def
publication_date
(
self
):
```

```
return
```

```
nowf
()
```

```
def
create_opf
(
self
,
feeds
,
dir
=
None
):
```

```
if
```

```
dir
```

```
is
```

```
None
:
```

```
dir
```

```
=  
  
self  
.  
output_dir  
  
title  
  
=  
  
self  
.  
short_title  
(  
  
if  
  
self  
.  
output_profile  
.  
periodical_date_in_title  
:  
  
title  
  
+=  
  
strftime  
(  
self  
.  
timefmt  
)  
  
mi  
  
=
```

```
MetaInformation
```

```
(  
title
```

```
,
```

```
[  
__appname__  
)
```

```
mi
```

```
.
```

```
publisher
```

```
=
```

```
__appname__
```

```
mi
```

```
.
```

```
author_sort
```

```
=
```

```
__appname__
```

```
if
```

```
self
```

```
.
```

```
publication_type
```

```
:
```

```
mi
```

```
.
```

```
publication_type
```

```
=
```

```
'periodical:'
```



```
+
self
.
publication_type
+
':.'
+
self
.
short_title
()
```

```
mi
.
timestamp
```

```
=
nowf
()
```

```
article_titles
,
```

```
aseen
```

```
=
```

```
[],
```

```
set
()
```

```
for
```

```
f
```

```
in
```

```
feeds
```

:

for

a

in

f

:

if

a

.

title

and

a

.

title

not

in

aseen

:

aseen

.

add

(

a

.

title

)

```
article_titles
.  
append  
(  
force_unicode  
(  
a  
.  
title  
,  
  
'utf-8'  
)  
)
```

```
desc  
  
=  
  
self  
.  
description
```

```
if  
  
not  
  
isinstance  
(  
desc  
,  
  
str  
):
```

```
desc  
  
=  
  
desc
```

```
.  
decode  
(  
    'utf-8'  
,  
  
    'replace'  
)  
  
mi  
.  
comments  
  
=  
  
(  
    -  
(  
        'Articles in this issue:'  
  
    )  
  
    +  
  
    '  
    \n\n'  
  
    +  
  
    '  
    \n\n'  
  
    .  
    join  
    (  
        article_titles  
    )  
  
    +
```

```
'  
\\n\\n'  
  
+  
  
desc  
  
language  
  
=  
  
canonicalize_lang  
(  
self  
.  
language  
)  
  
if  
  
language  
  
is  
  
not  
  
None  
:  
  
mi  
.  
language  
  
=  
  
language
```

```
mi
.
pubdate

=

self
.
publication_date
()

opf_path

=

os
.
path
.
join
(
dir
,
'index.opf'
)

ncx_path

=

os
.
path
.
join
(
dir
,
'index.ncx'
```

```
)
```

```
opf
```

```
=
```

```
OPFCreator
```

```
(
```

```
dir
```

```
,
```

```
mi
```

```
)
```

```
# Add mastheadImage entry to <guide> section
```

```
mp
```

```
=
```

```
getattr
```

```
(
```

```
self
```

```
,
```

```
'masthead_path'
```

```
,
```

```
None
```

```
)
```

```
if
```

```
mp
```

```
is
```

```
not
```

None

and

os

.

access

(

mp

,

os

.

R_OK

):

from

[calibre.ebooks.metadata.opf2](#)

import

Guide

ref

=

Guide

.

Reference

(

os

.

path

.

basename

(

self

.


```
masthead_path  
)
```

```
os
```

```
.  
getcwd  
(())
```

```
ref
```

```
.  
type
```

```
=
```

```
'masthead'
```

```
ref
```

```
.  
title
```

```
=
```

```
'Masthead Image'
```

```
opf
```

```
.  
guide
```

```
.  
append  
(  
  ref  
)
```

```
manifest
```

```
=
```

```
[
```

```
os
.
path
.
join
(
dir
,

'feed_
%d
'
%
i
)

for

i

in

range
(
len
(
feeds
))]

manifest
.
append
(
os
.
path
.
join
(
dir
,
```

```
'index.html'  
)
```

```
manifest
```

```
.  
append  
(  
os
```

```
.  
path
```

```
.  
join
```

```
(  
dir
```

```
,
```

```
'index.ncx'  
)
```

```
# Get cover
```

```
cpath
```

```
=
```

```
getattr
```

```
(  
self
```

```
,
```

```
'cover_path'
```

```
,
```

```
None
```

```
)
```

```
if
```

```
    cpath

    is

    None
    :

    pf

    =

    open
    (
    os
    .
    path
    .
    join
    (
    dir
    ,

    'cover.jpg'
    ),

    'wb'
    )

    if

    self
    .
    default_cover
    (
    pf
    ):

    cpath

    =
```

```
pf
.  
name
```

```
if
```

```
cpath
```

```
is
```

```
not
```

```
None
```

```
and
```

```
os  
.  
access  
(  
  cpath  
,  
  
  os  
  .  
  R_OK  
):
```

```
opf  
.  
cover
```

```
=
```

```
cpath
```

```
manifest  
.  
append
```

```
(
  cpath
)

# Get masthead

mpath

=

getattr
(
  self
,
  'masthead_path'
,
  None
)

if
  mpath
  is
  not
  None
  and
  os
  .
  access
  (
    mpath
  ,
```

```
os
```

```
.
```

```
R_OK
```

```
):
```

```
manifest
```

```
.
```

```
append
```

```
(
```

```
mpath
```

```
)
```

```
opf
```

```
.
```

```
create_manifest_from_files_in
```

```
(
```

```
manifest
```

```
)
```

```
for
```

```
mani
```

```
in
```

```
opf
```

```
.
```

```
manifest
```

```
:
```

```
if
```

```
mani
```

```
.
```

```
path
```

```
.
```

```
endswith
```

```
(  
  '.ncx'  
):  
  
mani  
.  
id  
  
=  
  
'ncx'  
  
if  
  
mani  
.  
path  
.  
endsWith  
(  
  'mastheadImage.jpg'  
):  
  
mani  
.  
id  
  
=  
  
'masthead-image'  
  
entries  
  
=  
  
[  
  'index.html'  
]
```



```
toc
```

```
=
```

```
TOC
```

```
(
```

```
base_path
```

```
=
```

```
dir
```

```
)
```

```
self
```

```
.
```

```
play_order_counter
```

```
=
```

```
0
```

```
self
```

```
.
```

```
play_order_map
```

```
=
```

```
{}
```

```
self
```

```
.
```

```
article_url_map
```

```
=
```

```
aumap
```

```
=
```

```
defaultdict  
(  
    set  
)
```

```
def
```

```
feed_index  
(  
    num  
    ,  
    parent  
):
```

```
    f
```

```
    =
```

```
    feeds  
    [  
        num  
    ]
```

```
for
```

```
    j  
    ,
```

```
    a
```

```
in
```

```
enumerate  
(  
    f  
):
```

```
if
    getattr
    (
    a
    ,
    'downloaded'
    ,
    False
    ):

```

```
adir
=
'feed_
%d
/article_
%d
/'
%
(
num
,
j
)

```

```
auth
=
a
.
author

```

```
if
```

not

auth

:

auth

=

None

desc

=

a

.

text_summary

if

not

desc

:

desc

=

None

else

:

desc

```
=  
  
self  
.  
description_limiter  
(  
desc  
)
```

```
tt
```

```
=
```

```
a  
.  
toc_thumbnail
```

```
if
```

```
a  
.  
toc_thumbnail
```

```
else
```

```
None
```

```
entries
```

```
.  
append  
(  
'  
%s  
index.html'  
%  
adir  
)
```

```
po
```

```
=  
  
self  
.  
play_order_map  
.  
get  
(  
entries  
[  
-  
1  
],  
  
None  
)  
  
if  
  
po  
  
is  
  
None  
:  
  
self  
.  
play_order_counter  
  
+=  
  
1  
  
po  
  
=  
  
self  
.  

```

```
play_order_counter
```

```
arelpath
```

```
=
```

```
,
```

```
%s
```

```
index.html'
```

```
%
```

```
adir
```

```
for
```

```
curl
```

```
in
```

```
self
```

```
.
```

```
canonicalize_internal_url
```

```
(
```

```
a
```

```
.
```

```
orig_url
```

```
,
```

```
is_link
```

```
=
```

```
False
```

```
):
```

```
aumap
```

```
[
```

```
curl
```

```
]
```

```
.
```

```
add
```

```
(
```

```
arelpath
```

```
)

article_toc_entry

=

parent
.
add_item
(
arelpah
,

None
,

a
.
title

if

a
.
title

else

_
(
'Untitled article'
),

play_order
=
po
,

author
=
```



```
auth
,

description
=
desc
,

toc_thumbnail
=
tt
)

for
entry
in
a
.
internal_toc_entries
:

    anchor
    =
    entry
    .
    get
    (
    'anchor'
    )

    if
    anchor
    :
```

```
self
.
play_order_counter

+=

1

po

+=

1

article_toc_entry
.
add_item
(

arelpah
,

entry
[
'anchor'
],

entry
[
'title'
]

or

-
(
'Unknown section'
),
```

```
play_order
```

```
=
```

```
po
```

```
)
```

```
last
```

```
=
```

```
os
```

```
.
```

```
path
```

```
.
```

```
join
```

```
(
```

```
self
```

```
.
```

```
output_dir
```

```
,
```

```
(
```

```
'
```

```
%s
```

```
index.html'
```

```
%
```

```
adir
```

```
)
```

```
.
```

```
replace
```

```
(
```

```
'/'
```

```
,
```

```
os
```

```
.
```

```
sep
```

```
))
```

for

sp

in

a

.
sub_pages

:

prefix

=

os

.
path

.
commonprefix

(
opf_path

,

sp

])

relp

=

sp

[
len

(
prefix
):]

entries

```
.  
append  
(  
relp
```

```
.  
replace  
(  
os
```

```
.  
sep  
,
```

```
'/'  
))
```

```
last
```

```
=
```

```
sp
```

```
if
```

```
os
```

```
.  
path
```

```
.  
exists  
(  
last  
):
```

```
with
```

```
open  
(  
last  
,
```

```
'rb'  
)  
  
as  
  
fi  
:  
  
src  
  
=  
  
fi  
.  
read  
(  
.  
decode  
(  
'utf-8'  
)  
  
soup  
  
=  
  
BeautifulSoup  
(  
src  
)  
  
body  
  
=  
  
soup  
.  
find  
(  
'body'
```

```
)  
  
if  
  
body  
  
is  
  
not  
  
None  
:  
  
prefix  
  
=  
  
 '/'  
.  
join  
(  
  '..'  
for  
  
i  
  
in  
  
range  
(  
  2  
  *  
  len  
(  
    re  
    .  
    findall  
(  
      r  
      'link\d+'  
    ,
```

```
last
))))
```

```
templ
```

```
=
```

```
self
```

```
.
```

```
navbar
```

```
.
```

```
generate
```

```
(
```

```
True
```

```
,
```

```
num
```

```
,
```

```
j
```

```
,
```

```
len
```

```
(
```

```
f
```

```
),
```

```
not
```

```
self
```

```
.
```

```
has_single_feed
```

```
,
```

```
a
```

```
.
```

```
orig_url
```

```
,
```



```
__appname__
```

```
,
```

```
prefix
```

```
=
```

```
prefix
```

```
,
```

```
center
```

```
=
```

```
self
```

```
.
```

```
center_navbar
```

```
)
```

```
elem
```

```
=
```

```
BeautifulSoup
```

```
(
```

```
templ
```

```
.
```

```
render
```

```
(
```

```
doctype
```

```
=
```

```
'xhtml'
```

```
)
```

```
.
```

```
decode
```

```
(
```

```
'utf-8'
```

```
))
```

```
.
```

```
find
```

```
(
```

```
'div'
```

```
)
```

```
body
.  
insert  
(  
  len  
(  
    body  
    .  
    contents  
  ),  
  
  elem  
)
```

with

```
open  
(  
  last  
  ,  
  
  'wb'  
)
```

as

```
fi  
:
```

```
fi  
.  
write  
(  
  str  
(  
    soup  
  )  
  .  
  encode  
(  
    'utf-8'
```

```
))
```

```
if
```

```
len
```

```
(
```

```
feeds
```

```
)
```

```
==
```

```
0
```

```
:
```

```
raise
```

```
Exception
```

```
(
```

```
'All feeds are empty, aborting.'
```

```
)
```

```
if
```

```
len
```

```
(
```

```
feeds
```

```
)
```

```
>
```

```
1
```

```
:
```

```
for
```

```
i
```

```
,
```

```
f
in
enumerate
(
feeds
):

entries
.
append
(
'feed_
%d
/index.html'
%
i
)

po
=

self
.
play_order_map
.
get
(
entries
[
-
1
],
None
)

if
```

```
po
```

```
is
```

```
None
```

```
:
```

```
self
```

```
.
```

```
play_order_counter
```

```
+=
```

```
1
```

```
po
```

```
=
```

```
self
```

```
.
```

```
play_order_counter
```

```
auth
```

```
=
```

```
getattr
```

```
(
```

```
f
```

```
,
```

```
'author'
```

```
,
```

```
None
```

```
)
```

```
if
    not
    auth
    :
    auth
    =
    None
    desc
    =
    getattr
    (
    f
    ,
    'description'
    ,
    None
    )
    if
    not
    desc
    :
    desc
    =
```

None

feed_index

(

i

,

toc

.

add_item

(

'feed_

%d

/index.html'

%

i

,

None

,

f

.

title

,

play_order

=

po

,

description

=

desc

,

author

=

auth

))

else

:

entries

.

append

(

'feed_

%d

/index.html'

%

0

)

feed_index

(

0

,

toc

)

for

i

,

p

in

enumerate

(

entries

):


```
entries
[
i
]

=

os
.
path
.
join
(
dir
,

p
.
replace
(
'/'
,

os
.
sep
))

opf
.
create_spine
(
entries
)

opf
.
set_toc
(
toc
)
```

with

```
open  
(  
opf_path  
,
```

```
'wb'  
)
```

as

```
opf_file  
,
```

```
open  
(  
ncx_path  
,
```

```
'wb'  
)
```

as

```
ncx_file  
:
```

```
opf  
.  
render  
(  
opf_file  
,
```

```
ncx_file  
)
```

```
def
article_downloaded
(
self
,
request
,
result
):

index
=

os
.
path
.
join
(
os
.
path
.
dirname
(
result
[
0
]),
'index.html'
)

if
index
```

```
!=
```

```
result
```

```
[
```

```
0
```

```
]:
```

```
if
```

```
os
```

```
.
```

```
path
```

```
.
```

```
exists
```

```
(
```

```
index
```

```
):
```

```
os
```

```
.
```

```
remove
```

```
(
```

```
index
```

```
)
```

```
os
```

```
.
```

```
rename
```

```
(
```

```
result
```

```
[
```

```
0
```

```
],
```

```
index
```

```
)
```

```
a
```

```
=
```

```
request
```

```
.
```

```
requestID
```

```
[
```

```
1
```

```
]
```

```
article
```

```
=
```

```
request
```

```
.
```

```
article
```

```
self
```

```
.
```

```
log
```

```
.
```

```
debug
```

```
(
```

```
'Downloaded article:'
```

```
,
```

```
article
```

```
.
```

```
title
```

```
,
```

```
'from'
```

```
,
```

```
article
```

```
.
```

```
url
```

```
)
```

```
article
.
orig_url
```

```
=
```

```
article
.
url
```

```
article
.
url
```

```
=
```

```
'article_
%d
/index.html'
%
```

```
a
```

```
article
.
downloaded
```

```
=
```

```
True
```

```
article
.
sub_pages
```

```
=
```

```
result
```

```
[
```

```
1
```

```
][
1
:]

self
.
jobs_done

+=

1

self
.
report_progress
(
float
(
self
.
jobs_done
)
/
len
(
self
.
jobs
),

_
(
'Article downloaded:
%s
'
)
%
force_unicode
(
article
```

```
.
title
))

if

result
[
2
]:

self

.
partial_failures

.
append
((
request

.
feed

.
title
,

article

.
title
,

article

.
url
,

result
[
2
]))
```



```
def
error_in_article_download
(
self
,
request
,
traceback
):

self
.
jobs_done
+=
1

if
traceback
and
re
.
search
(
'^AbortArticle:'
,
traceback
,
flags
=
re
.

```

```
M
)

is

not

None
:

self
.
log
.
warn
(
'Aborted download of article:'
,
request
.
article
.
title
,
'from'
,
request
.
article
.
url
)

self
.
report_progress
(
```

```
float
(
self
.
jobs_done
)
/
len
(
self
.
jobs
),

_
(
'Article download aborted:
%s
'
)
%
force_unicode
(
request
.
article
.
title
))

else
:

self
.
log
.
error
(
'Failed to download article:'
```

```
,  
  
request  
.  
article  
.  
title  
,  
  
'from'  
,  
  
request  
.  
article  
.  
url  
)  
  
self  
.  
log  
.  
debug  
(  
    traceback  
)  
  
self  
.  
log  
.  
debug  
(  
    '  
    \n  
'  
)
```

```
self
.
report_progress
(
float
(
self
.
jobs_done
)
/
len
(
self
.
jobs
),

_
(
'Article download failed:
%s
'
)
%
force_unicode
(
request
.
article
.
title
))

self
.
failed_downloads
.
append
((
request
```

```
.  
feed  
,  
  
request  
.  
article  
,  
  
traceback  
)
```

[\[docs\]](#)

def

```
parse_feeds  
(  
self  
):
```

```
'''
```

*Create a list of articles from the list of feeds returned by
:meth:`BasicNewsRecipe.get\_feeds`.*

Return a list of :class:`Feed` objects.

```
'''
```

```
feeds
```

```
=
```

```
self
```

```
.  
get_feeds  
(
```

```
parsed_feeds
```

```
=
```

```
[]
```

```
br
```

```
=
```

```
self
```

```
.
```

```
browser
```

```
for
```

```
obj
```

```
in
```

```
feeds
```

```
:
```

```
if
```

```
isinstance
```

```
(
```

```
obj
```

```
,
```

```
string_or_bytes
```

```
):
```

```
title
```

```
,
```

```
url
```

```
=
```

```
None
```

```
,
```

```
obj
```

```
else
```

```
:
```

```
title
```

```
,
```

```
url
```

```
=
```

```
obj
```

```
if
```

```
isinstance
```

```
(
```

```
title
```

```
,
```

```
bytes
```

```
):
```

```
title
```

```
=
```

```
title
```

```
.
```

```
decode
```

```
(
```

```
'utf-8'
```



```
)  
  
if  
  
isinstance  
(  
url  
,  
  
bytes  
):  
  
url  
  
=  
  
url  
.  
decode  
(  
'utf-8'  
)  
  
if  
  
url  
.  
startswith  
(  
'feed:/'  
):  
  
url  
  
=  
  
'http'  
+  
url
```

```
[
4
:]

self
.
report_progress
(
0
,

_
(
'Fetching feed'
)
+
,
%s
...!
%
(
title

if

title

else

url
))

try
:

purl

=

urlparse
```

```
(  
url  
,  
  
allow_fragments  
=  
False  
)
```

if

```
purl  
.  
username
```

or

```
purl  
.  
password  
:
```

```
hostname
```

```
=
```

```
purl  
.  
hostname
```

if

```
purl  
.  
port  
:
```

```
hostname
```

```
+=
```

```
f
```

```
':
```

```
{
```

```
purl
```

```
.
```

```
port
```

```
}
```

```
'
```

```
url
```

```
=
```

```
purl
```

```
.
```

```
_replace
```

```
(
```

```
netloc
```

```
=
```

```
hostname
```

```
)
```

```
.
```

```
geturl
```

```
()
```

```
if
```

```
purl
```

```
.
```

```
username
```

```
and
```

```
purl
```

```
.
```

```
password
```

```
:
```

```
br
.  
add_password  
(  
url  
,  
  
purl  
.  
username  
,  
  
purl  
.  
password  
)
```

with

```
closing  
(  
br  
.  
open_novisit  
(  
url  
,  
  
timeout  
=  
self  
.  
timeout  
))
```

as

```
f  
:
```

```
raw
```

```
=  
  
f  
.  
read  
(  
  
    parsed_feeds  
    .  
    append  
    (  
        feed_from_xml  
        (  
  
            raw  
            ,  
  
            title  
            =  
            title  
            ,  
  
            log  
            =  
            self  
            .  
            log  
            ,  
  
            oldest_article  
            =  
            self  
            .  
            oldest_article  
            ,  
  
            max_articles_per_feed  
            =
```

```
self
    .
    max_articles_per_feed
    ,
```

```
get_article_url
=
self
    .
    get_article_url
```

```
))
```

```
except
```

```
Exception
```

```
as
```

```
err
:
```

```
feed
```

```
=
```

```
Feed
()
```

```
msg
```

```
=
```

```
'Failed feed:
```

```
%s
```

```
,
```

```
%
```

```
(
```

```
title
```

```
if
```

```
title
```

```
else
```

```
url
```

```
)
```

```
feed
```

```
.
```

```
populate_from_prepared_feed
```

```
(
```

```
msg
```

```
,
```

```
[])
```

```
feed
```

```
.
```

```
description
```

```
=
```

```
as_unicode
```

```
(
```

```
err
```

```
)
```

```
parsed_feeds
```

```
.
```

```
append
```

```
(
```

```
feed
```

```
)
```

```
self
```



```
.  
log  
.  
exception  
(  
msg  
)
```

```
if
```

```
self
```

```
.  
delay
```

```
>
```

```
0  
:
```

```
time
```

```
.  
sleep  
(  
self  
.  
delay  
)
```

```
remove
```

```
=
```

```
[  
fl
```

```
for
```

```
fl
```

```
in

parsed_feeds

if

len
(
fl
)

==

0

and

self
.
remove_empty_feeds
]

for

f

in

remove
:

parsed_feeds
.
remove
(
f
)

return
```

parsed_feeds

[\[docs\]](#)

@classmethod

def

tag_to_string

(
self
,

tag
,

use_alt
=

True
,

normalize_whitespace
=

True
):

'''

Convenience method to take a

`BeautifulSoup` <<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>>`_

:code:`Tag` and extract the text from it recursively, including any CDATA sections

and alt tag attributes. Return a possibly empty Unicode string.

`use\_alt`: If `True` try to use the alt attribute for tags that don't

have any textual content

``tag`: `BeautifulSoup <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>`_`

`:code:`Tag``

`'''`

if

tag

is

None

:

return

"

if

isinstance

(

tag

,

string_or_bytes

):

return

tag

if

```
callable
(
  getattr
  (
    tag
    ,
    'xpath'
    ,
    None
  )
)

and

not

hasattr
(
  tag
  ,
  'contents'
):

# a lxml tag

from

lxml.etree

import

tostring

ans

=

tostring
(
```

```
tag
,
method
=
'text'
,
encoding
=
'unicode'
,
with_tail
=
False
)
```

```
else
:
```

```
strings
=
[]
```

```
for
```

```
item
```

```
in
```

```
tag
.
contents
:
```

```
if
```

```
isinstance
(
item
,
(
NavigableString
,
CData
)):
```

```
strings
.
append
(
item
.
string
)
```

elif

```
isinstance
(
item
,
Tag
):
```

```
res
```

```
=
```

```
self
.
tag_to_string
(
```

```
item
)
```

```
if
```

```
res
:
```

```
strings
.
append
(
res
)
```

```
elif
```

```
use_alt
:
```

```
try
:
```

```
strings
.
append
(
item
[
'alt'
])
```

```
except
```

```
KeyError
:
```



```
pass
```

```
ans
```

```
=
```

```
"
```

```
.
```

```
join
```

```
(
```

```
strings
```

```
)
```

```
if
```

```
normalize_whitespace
```

```
:
```

```
ans
```

```
=
```

```
re
```

```
.
```

```
sub
```

```
(
```

```
r
```

```
'\s+'
```

```
,
```

```
" "
```

```
" "
```

```
,
```

```
" "
```

```
ans
```

```
)
```

```
return
```

ans

@classmethod

def

soup

(

cls

,

raw

):

return

BeautifulSoup

(

raw

)

[\[docs\]](#)

@classmethod

def

adeify_images

(

cls

,

soup

):

```
'''  
  
    If your recipe when converted to EPUB has problems with images when  
  
    viewed in Adobe Digital Editions, call this method from within  
  
    :meth:`postprocess_html`.  
  
    '''
```

```
for
```

```
item
```

```
in
```

```
soup
```

```
.  
findAll
```

```
(  
    'img'  
):
```

```
for
```

```
attrib
```

```
in
```

```
[  
    'height'  
    ,  
    'width'  
    ,  
    'border'  
    ,  
    'align'  
    ,  
    'style'  
]:
```

```
try
```

```
:
```

```
del
```

```
item
```

```
[
```

```
attrib
```

```
]
```

```
except
```

```
KeyError
```

```
:
```

```
pass
```

```
oldParent
```

```
=
```

```
item
```

```
.
```

```
parent
```

```
myIndex
```

```
=
```

```
oldParent
```

```
.
```

```
contents
```

```
.
```

```
index
```

```
(
```

```
item
```

```
)
```

```
item
```

```
.
```

```
extract
```

```
()
```

```
divtag
```

```
=
```

```
soup
```

```
.
```

```
new_tag
```

```
(
```

```
'div'
```

```
)
```

```
brtag
```

```
=
```

```
soup
```

```
.
```

```
new_tag
```

```
(
```

```
'br'
```

```
)
```

```
oldParent
```

```
.
```

```
insert
```

```
(
```

```
myIndex
```

```
,
```

```
divtag
```

```
)
```

```
divtag
```

```
.
append
(
item
)

divtag
.
append
(
brtag
)

return

soup

def

internal_postprocess_book
(
self
,

oeb
,

opts
,

log
):

if

self
.
resolve_internal_links
```

```
and

self
.
article_url_map
:

seen

=

set
()

for

item

in

oeb
.
spine
:

for

a

in

item
.
data
.
xpath
(
'//*[local-name()="a" and @href]'
):
```

```
if
a
.
get
(
'rel'
)

==

'calibre-downloaded-from'
:

continue

url

=

a
.
get
(
'href'
)

for
curl

in
self
.
canonicalize_internal_url
(
url
);
```



```
articles
```

```
=
```

```
self
```

```
.
```

```
article_url_map
```

```
.
```

```
get
```

```
(
```

```
curl
```

```
)
```

```
if
```

```
articles
```

```
:
```

```
arelpath
```

```
=
```

```
sorted
```

```
(
```

```
articles
```

```
,
```

```
key
```

```
=
```

```
numeric_sort_key
```

```
)[
```

```
0
```

```
]
```

```
a
```

```
.
```

```
set
```

```
(
```

```
'href'
,
item
.
relhref
(
arelpah
))

if

url

not

in

seen
:

log
.
debug
(
f
'Resolved internal URL:
{
url
}
->
{
arelpah
}
'
)

seen
.
add
```

```
(  
url  
)
```

class

CustomIndexRecipe

```
(  
BasicNewsRecipe  
):
```

def

custom_index

```
(  
self  
):
```

```
'''
```

Return the filesystem path to a custom HTML document that will serve as the index for this recipe. The index document will typically contain many `<a href="...">` tags that point to resources on the internet that should be downloaded.

```
'''
```

raise

NotImplementedError

def

create_opf

```
(
```

```
self
):

mi

=

MetaInformation
(
self
.
title

+

strftime
(
self
.
timefmt
),

[
__appname__
])

mi
.
publisher

=

__appname__

mi
.
author_sort

=
```

__appname__

mi

=

OPFCreator

(

self

.

output_dir

,

mi

)

mi

.

create_manifest_from_files_in

([

self

.

output_dir

])

mi

.

create_spine

([

os

.

path

.

join

(

self

.

output_dir

,

```
'index.html'  
))
```

```
with
```

```
open  
(  
os  
.  
path  
.  
join  
(  
self  
.  
output_dir  
,
```

```
'index.opf'  
),
```

```
'wb'  
)
```

```
as
```

```
opf_file  
:
```

```
mi  
.  
render  
(  
opf_file  
)
```

```
def
```

```
download
```

```
(
self
):

index

=

os
.
path
.
abspath
(
self
.
custom_index
())

url

=

'file:'
+
index

if

iswindows

else

'file:/'
+
index

self
.
web2disk_options
```

```
.
browser

=

self
.
clone_browser
(
self
.
browser
)

fetcher

=

RecursiveFetcher
(
self
.
web2disk_options
,
self
.
log
)

fetcher

.
base_dir

=

self
.
output_dir
```



```
fetcher
.
current_dir

=

self
.
output_dir

fetcher
.
show_progress

=

False

res

=

fetcher
.
start_fetch
(
url
)

self
.
create_opf
()

return

res
```

```
class
```

```
AutomaticNewsRecipe
```

```
(  
BasicNewsRecipe  
):
```

```
auto_cleanup
```

```
=
```

```
True
```

```
class
```

```
CalibrePeriodical
```

```
(  
BasicNewsRecipe  
):
```

```
#: Set this to the slug for the calibre periodical
```

```
calibre_periodicals_slug
```

```
=
```

```
None
```

```
LOG_IN
```

```
=
```

```
'https://news.calibre-ebook.com/accounts/login'
```

```
needs_subscription
```

```
=
```

```
True
```

```
__author__
```

```
=
```

```
'calibre Periodicals'
```

```
def
```

```
get_browser
```

```
(
```

```
self
```

```
):
```

```
br
```

```
=
```

```
BasicNewsRecipe
```

```
.
```

```
get_browser
```

```
(
```

```
self
```

```
)
```

```
br
```

```
.
```

```
open
```

```
(
```

```
self
```

```
.
```

```
LOG_IN
```

```
)
```

```
br
```

```
.
```

```
select_form
```

```
(
```

```
name
```

```
=
```

```
'login'
```

```
)
```

```
br
```

```
[
```

```
'username'
```

```
]
```

```
=
```

```
self
```

```
.
```

```
username
```

```
br
```

```
[
```

```
'password'
```

```
]
```

```
=
```

```
self
```

```
.
```

```
password
```

```
raw
```

```
=
```

```
br
```

```
.
submit
()
.
read
()

if

'href="/my-account"'

not

in

raw
:

raise

LoginFailed
(

_
(
'Failed to log in, check your username and password for'

' the calibre Periodicals service.'
))

return

br

get_browser
.
```

```
is_base_class_implementation
```

```
=
```

```
True
```

```
def
```

```
download
```

```
(  
self  
):
```

```
self
```

```
.  
log  
(  
'Fetching downloaded recipe'  
)
```

```
try
```

```
:
```

```
raw
```

```
=
```

```
self
```

```
.  
browser
```

```
.  
open_novisit  
(
```

```
'https://news.calibre-ebook.com/subscribed_files/  
%s  
'/0/temp.downloaded_recipe'
```

```
%

self
.
calibre_periodicals_slug

)
.
read
()

except
Exception

as

e
:

if

hasattr
(
e
,
'getcode'
)

and

e
.
getcode
()

==
```

403

:

raise

DownloadDenied

(

—
(

'You do not have permission to download this issue.'

' Either your subscription has expired or you have'

' exceeded the maximum allowed downloads for today.'

))

raise

f

=

io

.

BytesIO

(

raw

)

from

calibre.utils.zipfile

import


```
ZipFile
```

```
zf
```

```
=
```

```
ZipFile
```

```
(  
f  
)
```

```
zf
```

```
.
```

```
extractall
```

```
()
```

```
zf
```

```
.
```

```
close
```

```
()
```

```
from
```

```
glob
```

```
import
```

```
glob
```

```
from
```

```
calibre.web.feeds.recipes
```

```
import
```

```
compile_recipe
```

```
try
:

recipe

=

compile_recipe
(
open
(
glob
(
'*.recipe'
)[
0
],

'rb'
)
.read
())

self
.conversion_options

=

recipe
.conversion_options

except
:
```

```
self
.  
log  
.  
exception  
(  
    'Failed to compile downloaded recipe'  
)  
  
return  
  
os  
.  
path  
.  
abspath  
(  
    'index.html'  
)
```

Table of Contents

1. [calibre User Manual](#)
2. [The Graphical User Interface](#)
 1. [Actions](#)
 1. [Add books](#)
 2. [Edit metadata](#)
 3. [Convert books](#)
 4. [View](#)
 5. [Send to device](#)
 6. [Fetch news](#)
 7. [Library](#)
 8. [Device](#)
 9. [Save to disk](#)
 10. [Connect/share](#)
 11. [Remove books](#)
 2. [Preferences](#)
 3. [Catalogs](#)
 4. [Search & sort](#)
 5. [The search interface](#)
 6. [Saving searches](#)
 7. [Virtual libraries](#)
 8. [Guessing metadata from file names](#)
 9. [Book details](#)

10. [Tag browser](#)
11. [Cover grid](#)
12. [Cover browser](#)
13. [Quickview](#)
14. [Jobs](#)
15. [Keyboard shortcuts](#)
3. [Adding your favorite news website](#)
 1. [Completely automatic fetching](#)
 1. [The calibre blog](#)
 2. [bbc.co.uk](#)
 2. [Customizing the fetch process](#)
 1. [Using the print version of bbc.co.uk](#)
 2. [Replacing article styles](#)
 3. [Slicing and dicing](#)
 4. [Real life example](#)
 3. [Tips for developing new recipes](#)
 4. [Further reading](#)
 5. [API documentation](#)
 1. [API documentation for recipes](#)
4. [The E-book viewer](#)
 1. [Starting the E-book viewer](#)
 2. [Navigating around an e-book](#)
 1. [Bookmarks](#)
 2. [Table of Contents](#)
 3. [Navigating by location](#)

4. [Reference mode](#)
 3. [Highlighting text](#)
 4. [Read aloud](#)
 5. [Following links using only the keyboard](#)
 6. [Customizing the look and feel of your reading experience](#)
 7. [Dictionary lookup](#)
 8. [Copying text and images](#)
 9. [Zooming in on images](#)
 10. [Non re-flowable content](#)
 11. [Designing your book to work well with the calibre viewer](#)
5. [E-book conversion](#)
 1. [Introduction](#)
 2. [Look & feel](#)
 1. [Fonts](#)
 2. [Text](#)
 3. [Layout](#)
 4. [Styling](#)
 5. [Transform styles](#)
 6. [Transform HTML](#)
 3. [Page setup](#)
 4. [Heuristic processing](#)
 5. [Search & replace](#)
 6. [Structure detection](#)
 1. [Chapters and page breaks](#)
 2. [Miscellaneous](#)

7. [Table of Contents](#)
8. [Using images as chapter titles when converting HTML input documents](#)
9. [Using tag attributes to supply the text for entries in the Table of Contents](#)
10. [How options are set/saved for conversion](#)
11. [Format specific tips](#)
 1. [Convert Microsoft Word documents](#)
 1. [Older .doc files](#)
 2. [Convert TXT documents](#)
 3. [Convert PDF documents](#)
 4. [Comic book collections](#)
 5. [EPUB advanced formatting demo](#)
 6. [Convert ODT documents](#)
 7. [Converting to PDF](#)
 1. [Headers and Footers](#)
 2. [Printable Table of Contents](#)
 3. [Custom page margins for individual HTML files](#)
6. [Editing e-books](#)
 1. [Basic workflow](#)
 2. [The File browser](#)
 1. [Renaming files](#)
 2. [Merging files](#)
 3. [Changing text file order](#)
 4. [Marking the cover](#)
 5. [Deleting files](#)
 6. [Exporting files](#)

7. [Adding new images/fonts/etc. or creating new blank files](#)
 8. [Replacing files](#)
 9. [Linking stylesheets to HTML files efficiently](#)
3. [Search & replace](#)
 1. [Saved searches](#)
 2. [Function mode](#)
 3. [Search ignoring HTML tags](#)
4. [Automated tools](#)
 1. [Editing the Table of Contents](#)
 2. [Checking the book](#)
 3. [Adding a cover](#)
 4. [Embedding referenced fonts](#)
 5. [Subsetting embedded fonts](#)
 6. [Smartening punctuation](#)
 7. [Transforming CSS properties](#)
 8. [Removing unused CSS rules](#)
 9. [Fixing HTML](#)
 10. [Beautifying files](#)
 11. [Inserting an inline Table of Contents](#)
 12. [Setting Semantics](#)
 13. [Filtering style information](#)
 14. [Upgrading the book's internals](#)
5. [Checkpoints](#)
6. [The Live preview panel](#)
 1. [Splitting HTML files](#)

7. [The Live CSS panel](#)
8. [Miscellaneous tools](#)
 1. [The Table of Contents view](#)
 2. [Checking the spelling of words in the book](#)
 1. [Adding new dictionaries](#)
 3. [Inserting special characters](#)
 4. [The code inspector view](#)
 5. [Checking external links](#)
 6. [Downloading external resources](#)
 7. [Arranging files into folders by type](#)
 8. [Importing files in other e-book formats as EPUB](#)
 9. [The Reports tool](#)
9. [Special features in the code editor](#)
 1. [Syntax highlighting](#)
 2. [Context sensitive help](#)
 3. [Auto-complete](#)
 4. [Snippets](#)
7. [The calibre Content server](#)
 1. [Accessing the Content server from other devices](#)
 1. [Accessing the server from devices on your home network](#)
 1. [Trouble-shooting the home network connection](#)
 2. [Accessing the server from anywhere on the internet](#)
 2. [The server interface](#)
 1. [The book list](#)
 2. [The book reader](#)

3. [Browser support](#)
4. [Enabling offline support](#)
5. [Managing user accounts from the command-line only](#)
6. [Integrating the calibre Content server into other servers](#)
 1. [Using a full virtual host](#)
 2. [Using a URL prefix](#)
7. [Creating a service for the calibre server on a modern Linux system](#)
8. [Comparing e-books](#)
 1. [Understanding the comparison view](#)
 2. [Launching the comparison tool](#)
 1. [Comparing two e-book files](#)
 2. [Comparing the ORIGINAL FMT to FMT](#)
 3. [Comparing a checkpoint to the current state of the book while editing](#)
9. [Editing e-book metadata](#)
 1. [Editing the metadata of one book at a time](#)
 1. [Downloading metadata](#)
 2. [Managing book formats](#)
 3. [All about covers](#)
 2. [Editing the metadata of many books at a time](#)
 1. [Search and replace](#)
 2. [Bulk downloading of metadata](#)
10. [Frequently Asked Questions](#)
 1. [E-book format conversion](#)
 1. [What formats does calibre support conversion to/from?](#)
 2. [What are the best source formats to convert?](#)

3. [I converted a PDF file, but the result has various problems?](#)
4. [How do I convert my file containing non-English characters, or smart quotes?](#)
5. [What's the deal with Table of Contents in MOBI files?](#)
6. [The covers for my MOBI files have stopped showing up in Kindle for PC/Kindle for Android/iPad etc.](#)
7. [How do I convert a collection of HTML files in a specific order?](#)
8. [The EPUB I produced with calibre is not valid?](#)
9. [How do I use some of the advanced features of the conversion tools?](#)
2. [Device integration](#)
 1. [What devices does calibre support?](#)
 2. [How can I help get my device supported in calibre?](#)
 3. [My device is not being detected by calibre?](#)
 4. [My device is non-standard or unusual. What can I do to connect to it?](#)
 5. [How do I use calibre with my iPad/iPhone/iPod touch?](#)
 1. [Using the Content server](#)
 6. [How do I use calibre with my Android phone/tablet or Kindle Fire HD?](#)
 1. [Using a USB cable](#)
 2. [Over the air](#)
 7. [Can I access my calibre books using the web browser in my Kindle or other reading device?](#)
 8. [I cannot send emails using calibre?](#)
 9. [My device is getting mounted read-only in Linux, so calibre cannot connect to it?](#)
 10. [Why does calibre not support collections on the Kindle or shelves on the Nook?](#)

11. [I am getting an error when I try to use calibre with my Kobo Touch/Glo/etc.?](#)
 12. [Covers for books I send to my e-ink Kindle show up momentarily and then are replaced by a generic cover?](#)
 13. [I transferred some books to my Kindle using calibre and they did not show up?](#)
3. [Library management](#)
1. [Where are the book files stored?](#)
 2. [How does calibre manage author names and sorting?](#)
 3. [Why doesn't calibre let me store books in my own folder structure?](#)
 4. [Why doesn't calibre have a column for foo?](#)
 5. [Can I have a column showing the formats or the ISBN?](#)
 6. [How do I move my calibre data from one computer to another?](#)
 7. [The list of books in calibre is blank!](#)
 8. [I am getting errors with my calibre library on a networked drive/NAS?](#)
4. [Miscellaneous](#)
1. [I want calibre to download news from my favorite news website.](#)
 2. [Why the name calibre?](#)
 3. [Why does calibre show only some of my fonts on macOS?](#)
 4. [calibre is not starting on Windows?](#)
 5. [calibre freezes/crashes occasionally?](#)
 6. [The calibre E-book viewer and Edit book tools do not work on Windows?](#)
 7. [Using the viewer or doing any conversions results in a permission denied error on Windows](#)
 8. [calibre is not starting/crashing on macOS?](#)
 9. [I downloaded the installer, but it is not working?](#)

10. [My antivirus program claims calibre is a virus/trojan?](#)
11. [How do I backup calibre?](#)
12. [How do I use purchased EPUB books with calibre \(or what do I do with .acsm files\)?](#)
13. [I am getting a "Permission Denied" error?](#)
14. [Can I have the comment metadata show up on my reader?](#)
15. [How do I get calibre to use my HTTP proxy?](#)
16. [I want some feature added to calibre. What can I do?](#)
17. [Why doesn't calibre have an automatic update?](#)
18. [How is calibre licensed?](#)
19. [How do I run calibre from my USB stick?](#)
20. [How do I run parts of calibre like news download and the Content server on my own Linux server?](#)

11. [Tutorials](#)

1. [Adding your favorite news website](#)
 1. [Completely automatic fetching](#)
 1. [The calibre blog](#)
 2. [bbc.co.uk](#)
 2. [Customizing the fetch process](#)
 1. [Using the print version of bbc.co.uk](#)
 2. [Replacing article styles](#)
 3. [Slicing and dicing](#)
 4. [Real life example](#)
 3. [Tips for developing new recipes](#)
 4. [Further reading](#)

5. [API documentation](#)
 1. [API documentation for recipes](#)
2. [Managing subgroups of books, for example "genre"](#)
 1. [Setup](#)
 2. [Searching](#)
 3. [Restrictions](#)
 4. [Useful template functions](#)
3. [XPath tutorial](#)
 1. [Selecting by tag name](#)
 2. [Selecting by attributes](#)
 3. [Selecting by tag content](#)
 4. [Sample e-book](#)
 5. [XPath built-in functions](#)
4. [The calibre template language](#)
 1. [Basic templates](#)
 2. [Advanced formatting](#)
 3. [Using templates to define custom columns](#)
 4. [Using functions in templates - Single Function Mode](#)
 5. [General Program Mode](#)
 6. [More complex programs in template expressions - Template Program Mode](#)
 7. [Stored general program mode templates](#)
 8. [Providing additional information to templates](#)
 9. [Notes on the difference between modes](#)
 10. [User-defined Python template functions](#)

11. [Special notes for save/send templates](#)
12. [Templates and plugboards](#)
13. [Tips](#)
14. [Function reference](#)
 1. [Reference for all built-in template language functions](#)
5. [All about using regular expressions in calibre](#)
 1. [First, a word of warning and a word of courage](#)
 2. [Where in calibre can you use regular expressions?](#)
 3. [What on earth is a regular expression?](#)
 4. [Care to explain?](#)
 5. [That doesn't sound too bad. What's next?](#)
 6. [Hey, neat! This is starting to make sense!](#)
 7. [Well, these special characters are very neat and all, but what if I wanted to match a dot or a question mark?](#)
 8. [So, what are the most useful sets?](#)
 9. [But if I had a few varying strings I wanted to match, things get complicated?](#)
 10. [You missed...](#)
 11. [In the beginning, you said there was a way to make a regular expression case insensitive?](#)
 12. [I think I'm beginning to understand these regular expressions now... how do I use them in calibre?](#)
 1. [Conversions](#)
 2. [Adding books](#)
 3. [Bulk editing metadata](#)
 13. [Quick reference](#)

1. [Quick reference for regexp syntax](#)
14. [Credits](#)
6. [Writing your own plugins to extend calibre's functionality](#)
 1. [Anatomy of a calibre plugin](#)
 2. [A User Interface plugin](#)
 1. [__init__.py](#)
 2. [ui.py](#)
 3. [main.py](#)
 4. [Getting resources from the plugin ZIP file](#)
 5. [Enabling user configuration of your plugin](#)
 3. [Edit book plugins](#)
 1. [main.py](#)
 4. [Running User Interface plugins in a separate process](#)
 5. [Adding translations to your plugin](#)
 6. [The plugin API](#)
 7. [Debugging plugins](#)
 8. [More plugin examples](#)
 9. [Sharing your plugins with others](#)
7. [Typesetting mathematics in e-books](#)
 1. [A simple HTML file with mathematics](#)
 2. [More information](#)
8. [Creating AZW3 • EPUB • MOBI catalogs](#)
 1. [Selecting books to catalog](#)
 2. [Included sections](#)
 3. [Prefixes](#)

4. [Excluded books](#)
5. [Excluded genres](#)
6. [Other options](#)
7. [Custom catalog covers](#)
8. [Additional help resources](#)
9. [Virtual libraries](#)
 1. [Creating Virtual libraries](#)
 1. [Examples of useful Virtual libraries](#)
 2. [Working with Virtual libraries](#)
 3. [Using Virtual libraries in searches](#)
 4. [Using additional restrictions](#)
12. [The calibre:// URL scheme](#)
 1. [Switch to a specific library](#)
 2. [Show a specific book in calibre](#)
 3. [Open a specific book in the E-book viewer at a specific position](#)
 4. [Searching for books](#)
 5. [Hex encoding of URL parameters](#)
13. [Customizing calibre](#)
 1. [Environment variables](#)
 2. [Tweaks](#)
 3. [Overriding icons, templates, et cetera](#)
 4. [Creating your own icon theme for calibre](#)
 5. [Customizing calibre with plugins](#)
14. [Command Line Interface](#)
 1. [Documented commands](#)

1. [calibre](#)
 1. [\[options\]](#)
2. [calibre-customize](#)
 1. [\[options\]](#)
3. [calibre-debug](#)
 1. [\[options\]](#)
4. [calibre-server](#)
 1. [\[options\]](#)
5. [calibre-smtp](#)
 1. [\[options\]](#)
6. [calibredb](#)
 1. [Global Options](#)
 2. [list](#)
 3. [add](#)
 4. [remove](#)
 5. [add format](#)
 6. [remove format](#)
 7. [show metadata](#)
 8. [set metadata](#)
 9. [export](#)
 10. [catalog](#)
 11. [saved searches](#)
 12. [add custom column](#)
 13. [custom columns](#)
 14. [remove custom column](#)

15. [set custom](#)
 16. [restore database](#)
 17. [check library](#)
 18. [list categories](#)
 19. [backup metadata](#)
 20. [clone](#)
 21. [embed metadata](#)
 22. [search](#)
7. [ebook-convert](#)
1. [Look and Feel](#)
 2. [Heuristic Processing](#)
 3. [Search and Replace](#)
 4. [Structure Detection](#)
 5. [Table of Contents](#)
 6. [Metadata](#)
 7. [Debug](#)
 8. [AZW4 Input Options](#)
 9. [CHM Input Options](#)
 10. [Comic Input Options](#)
 11. [DJVU Input Options](#)
 12. [DOCX Input Options](#)
 13. [EPUB Input Options](#)
 14. [FB2 Input Options](#)
 15. [HTLZ Input Options](#)
 16. [HTML Input Options](#)

17. [LIT Input Options](#)
18. [LRF Input Options](#)
19. [MOBI Input Options](#)
20. [ODT Input Options](#)
21. [PDB Input Options](#)
22. [PDF Input Options](#)
23. [PML Input Options](#)
24. [RB Input Options](#)
25. [RTF Input Options](#)
26. [Recipe Input Options](#)
27. [SNB Input Options](#)
28. [TCR Input Options](#)
29. [TXT Input Options](#)
30. [AZW3 Output Options](#)
31. [DOCX Output Options](#)
32. [EPUB Output Options](#)
33. [FB2 Output Options](#)
34. [HTML Output Options](#)
35. [HTMLZ Output Options](#)
36. [LIT Output Options](#)
37. [LRF Output Options](#)
38. [MOBI Output Options](#)
39. [OEB Output Options](#)
40. [PDB Output Options](#)
41. [PDF Output Options](#)

- 42. [PML Output Options](#)
- 43. [RB Output Options](#)
- 44. [RTF Output Options](#)
- 45. [SNB Output Options](#)
- 46. [TCR Output Options](#)
- 47. [TXT Output Options](#)
- 48. [TXTZ Output Options](#)
- 8. [ebook-edit](#)
 - 1. [\[options\]](#)
- 9. [ebook-meta](#)
 - 1. [\[options\]](#)
- 10. [ebook-polish](#)
 - 1. [\[options\]](#)
- 11. [ebook-viewer](#)
 - 1. [\[options\]](#)
- 12. [fetch-ebook-metadata](#)
 - 1. [\[options\]](#)
- 13. [lrf2lrs](#)
 - 1. [\[options\]](#)
- 14. [lrfviewer](#)
 - 1. [\[options\]](#)
- 15. [lrs2lrf](#)
 - 1. [\[options\]](#)
- 16. [web2disk](#)
 - 1. [\[options\]](#)

2. [Undocumented commands](#)
15. [Setting up a calibre development environment](#)
 1. [Design philosophy](#)
 1. [Code layout](#)
 2. [Getting the code](#)
 1. [Submitting your changes to be included](#)
 3. [Windows development environment](#)
 4. [macOS development environment](#)
 5. [Linux development environment](#)
 6. [Having separate “normal” and “development” calibre installs on the same computer](#)
 7. [Debugging tips](#)
 1. [Using print statements](#)
 2. [Using an interactive Python interpreter](#)
 3. [Using the Python debugger as a remote debugger](#)
 4. [Using the debugger in your favorite Python IDE](#)
 5. [Executing arbitrary scripts in the calibre Python environment](#)
 8. [Using calibre in your projects](#)
 1. [Binary install of calibre](#)
 2. [Source install on Linux](#)
 9. [API documentation for various parts of calibre](#)
 1. [API documentation for recipes](#)
 2. [API documentation for plugins](#)
 1. [Plugin](#)
 2. [FileTypePlugin](#)

3. [Metadata plugins](#)
4. [Catalog plugins](#)
5. [Metadata download plugins](#)
6. [Conversion plugins](#)
7. [Device drivers](#)
8. [User interface actions](#)
9. [Preferences plugins](#)
3. [API documentation for the database interface](#)
4. [API documentation for the e-book editing tools](#)
 1. [The Container object](#)
 2. [Managing component files in a container](#)
 3. [Pretty printing and auto fixing parse errors](#)
 4. [Managing book jackets](#)
 5. [Splitting and merging of files](#)
 6. [Managing covers](#)
 7. [Working with CSS](#)
 8. [Working with the Table of Contents](#)
 9. [Edit book tool](#)
 10. [Controlling the editor's user interface](#)

16. [Digital Rights Management \(DRM\)](#)

1. [What does DRM imply for me personally?](#)
2. [What does DRM do for authors?](#)
3. [DRM and freedom](#)
4. [Why does calibre not support DRM?](#)
5. [What is calibre's view on content providers?](#)

6. [How can I help fight DRM?](#)

17. [Glossary](#)