

BOOKWORM
Open Source eBook Reader
Source Code

github.com/babluboy/bookworm
Language: Vala | License: GPL-3.0

Bookworm is a simple, focused eBook reader built for the Elementary OS desktop. It supports EPUB, MOBI, PDF, FB2, CBR, and CBZ formats and is written in Vala, a modern statically-typed language that compiles to C and uses GLib/GTK for its UI.

This document contains eight core source files chosen to show how a real desktop application is structured: entry point, application class, data model, database layer, UI components, and format-specific readers.

Files included:

main.vala	Entry Point
bookworm.vala	Application Class
book.vala	Book Model
library.vala	Library View
database.vala	Database Layer
ePubReader.vala	EPUB Reader
contentHandler.vala	Content Handler
window.vala	Main Window

```
=====
main.vala ? Entry Point
=====

1  /* Copyright 2017 Siddhartha Das (bablu.boy@gmail.com)
2  *
3  * This file is part of Bookworm and entry point to the
4  * application with the main method
5  *
6  * Bookworm is free software: you can redistribute it
7  * and/or modify it under the terms of the GNU General Public License as
8  * published by the Free Software Foundation, either version 3 of the
9  * License, or (at your option) any later version.
10 *
11 * Bookworm is distributed in the hope that it will be
12 * useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
13 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
14 * Public License for more details.
15 *
16 * You should have received a copy of the GNU General Public License along
17 * with Bookworm. If not, see http://www.gnu.org/licenses/.
18 */
19 BookwormApp.Bookworm application;
20 public static int main (string[] args) {
21     Environment.set_variable ("G_MESSAGES_DEBUG", "all", true);
22     //Get an instance of Bookworm if is running, otherwise create a new instance
23     application = BookwormApp.Bookworm.getAppInstance ();
24     //Workaround to get Granite's --about & Gtk's --help working together
25     if ("--help"      in args || "-h" in args ||
26         "--version"   in args ||
27         "--discover"  in args)
28     {
29         return application.processCommandLine (args);
30     } else {
31         Gtk.init (ref args);
32         if ("--debug"    in args) {
33             application.command_line_option_debug = true;
34         }
35         if ("--info"     in args) {
36             application.command_line_option_info = true;
37         }
38         return application.run (args);
39     }
40 }
```

```
=====
bookworm.vala ? Application Class
=====

1  /* Copyright 2017 Siddhartha Das (bablu.boy@gmail.com)
2  *
3  * This file is part of Bookworm and is the main Application class
4  *
5  * Bookworm is free software: you can redistribute it
6  * and/or modify it under the terms of the GNU General Public License as
7  * published by the Free Software Foundation, either version 3 of the
8  * License, or (at your option) any later version.
9  *
10 * Bookworm is distributed in the hope that it will be
11 * useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
12 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
13 * Public License for more details.
14 *
15 * You should have received a copy of the GNU General Public License along
16 * with Bookworm. If not, see http://www.gnu.org/licenses/.
17 */
18
19 using Gtk;
20 using Gee;
21 using Granite.Widgets;
22 using Granite.Services;
23 public const string GETTEXT_PACKAGE = "bookworm";
24
25 public class BookwormApp.Bookworm : Granite.Application {
26     private static Bookworm application;
27     private static bool isBookwormRunning = false;
28     public int exitCodeForCommand = 0;
29     public static Granite.Services.Paths app_xdg_path;
30     public static string bookworm_config_path = "";
31
32     public static string[] commandLineArgs;
33     public static bool command_line_option_version = false;
34     public static bool command_line_option_debug = false;
35     public static bool command_line_option_info = false;
36     public static bool command_line_option_discover = false;
37     private static OptionEntry[] options;
38
39     public StringBuilder spawn_async_with_pipes_output = new StringBuilder ("");
40
41     public static BookwormApp.Settings settings;
42     public static Gtk.ApplicationWindow window;
43     public static Gtk.IconTheme default_theme;
44     public static CssProvider cssProvider;
45     public static Gtk.Box bookWormUIBox;
46     public static Granite.Widgets.Welcome welcomeWidget;
47     public static Granite.Widgets.ModeButton library_mode_button;
48     public static Gtk.TreeModelFilter libraryTreeModelFilter;
49     public static Gtk.Button library_view_button;
50     public static Gtk.Button content_list_button;
51     public static Gtk.Button prefButton;
52     public static Gdk.Pixbuf bookSelectionPix;
53     public static Gdk.Pixbuf bookSelectedPix;
54     public static Gdk.Pixbuf image_selection_option_small;
55     public static Gdk.Pixbuf image_selection_checked_small;
56     public static Gdk.Pixbuf image_selection_transparent_small;
57     public static Gdk.Pixbuf image_selection_scaled;
58     public static Gdk.Pixbuf image_rating_1;
59     public static Gdk.Pixbuf image_rating_2;
60     public static Gdk.Pixbuf image_rating_3;
61     public static Gdk.Pixbuf image_rating_4;
62     public static Gdk.Pixbuf image_rating_5;
63     public static Gtk.Image select_book_image;
64     public static Gtk.Image add_book_image;
65     public static Gtk.Image remove_book_image;
66     public static Gtk.Image updateImageIcon;
67     public static Gtk.Image add_scan_directory_image;
68     public static Gtk.Image remove_scan_directory_image;
69     public static Gtk.Image library_list_button_image;
70     public static Gtk.Image library_grid_button_image;
```

```

71     public static Gtk.Image content_list_button_image;
72     public static Gtk.Image menu_icon_text_large;
73     public static Gtk.Image menu_icon;
74     public static Gtk.Image pref_menu_icon_text_large;
75     public static Gtk.Image pref_menu_icon_text_small;
76     public static Gtk.Image pref_menu_icon_align_left;
77     public static Gtk.Image pref_menu_icon_align_right;
78     public static Gtk.Image back_button_image;
79     public static Gtk.Image forward_button_image;
80     public static Gtk.Image back_page_image;
81     public static Gtk.Image forward_page_image;
82
83     public static string BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_UI_STATES[0];
84     public static Gee.HashMap<string, BookwormApp.Book> libraryViewMap = new Gee.HashMap<strin
g, BookwormApp.Book> ();
85     public static string locationOfEBookCurrentlyRead = "";
86     public static string[] pathsOfBooksToBeAdded;
87     public static int noOfBooksAddedFromCommand = 0;
88     public static bool isBookBeingAddedToLibrary = false;
89     public static bool isPageScrollRequired = false;
90     public static StringBuilder pathsOfBooksInLibraryOnLoadStr = new StringBuilder ("");
91     public static StringBuilder pathsOfBooksNotAddedStr = new StringBuilder ("");
92     public static StringBuilder onLoadJavaScript = new StringBuilder ("");
93     public static string bookwormScripts = "";
94     public static string bookwormStyles = "";
95     public static string bookTextSearchString = "";
96     public static TreeMap<string,string> searchResultsMap = new TreeMap<string,string> ();
97     public static StringBuilder aContentFileToBeSearched = new StringBuilder ("");
98     public static string[] profileColorList;
99     public static string no_of_books_per_page = "21";
100    public static ArrayList<string> paginationlist = new ArrayList<string> ();
101    public static int current_page_counter = 0;
102    public static ShortcutsAssocsHolder shortcutAssocs;
103    public static AccelGroup accel;
104
105    construct {
106        build_version = BookwormApp.Constants.bookworm_version;
107        application_id = BookwormApp.Constants.bookworm_id;
108        flags |= ApplicationFlags.HANDLES_COMMAND_LINE;
109        program_name = BookwormApp.Constants.program_name;
110        exec_name = BookwormApp.Constants.bookworm_id;
111        options = new OptionEntry[4];
112        options[0] = { "version", 0, 0, OptionArg.NONE, ref command_line_option_version, _("Display version number"), null };
113        options[1] = { "debug", 0, 0, OptionArg.NONE, ref command_line_option_debug, _("Run Bookworm in debug mode"), null };
114        options[2] = { "info", 0, 0, OptionArg.NONE, ref command_line_option_info, _("Run Bookworm in info mode"), null };
115        options[3] = { "discover", 0, 0, OptionArg.NONE, ref command_line_option_discover, _("Automatically add new books from watched folders"), null };
116        add_main_option_entries (options);
117    }
118
119    private Bookworm () {
120        Object (application_id: BookwormApp.Constants.bookworm_id, flags: ApplicationFlags.HAN
DLES_COMMAND_LINE);
121        Intl.setlocale (LocaleCategory.MESSAGES, "");
122        Intl.textdomain (GETTEXT_PACKAGE);
123        Intl.bind_textdomain_codeset (GETTEXT_PACKAGE, "utf-8");
124        //Initialize XDG Paths
125        app_xdg_path = new Granite.Services.Paths ();
126        app_xdg_path.initialize (Constants.bookworm_id, Constants.INSTALL_SCRIPTS_DIR);
127        bookworm_config_path = app_xdg_path.user_data_folder.get_path ();
128        debug ("Bookworm Install Directory: " + BookwormApp.Constants.INSTALL_PREFIX);
129        debug ("Bookworm Install Tasks Scripts Directory: " + BookwormApp.Constants.INSTALL_TA
SKS_DIR);
130        debug ("Bookworm Install Mobi Scripts Directory: " + BookwormApp.Constants.INSTALL_MOB
ILIB_DIR);
131        debug ("Bookworm User Data Directory: " + bookworm_config_path);
132    }
133
134    public static Bookworm getAppInstance () {
135        if (application == null) {
136            application = new Bookworm ();
137        } else {

```

```

138         //do nothing, return the existing instance
139     }
140     return application;
141 }
142
143 public override int command_line (ApplicationCommandLine command_line) {
144     commandLineArgs = command_line.get_arguments ();
145     activate ();
146     return 0;
147 }
148
149 public int processCommandLine (string[] args) {
150     try {
151         var opt_context = new OptionContext ("-- bookworm");
152         opt_context.set_help_enabled (true);
153         opt_context.add_main_entries (options, null);
154         unowned string[] tmpArgs = args;
155         opt_context.parse (ref tmpArgs);
156         if ("--version" in args) {
157             command_line_option_version = true;
158         }
159     } catch (OptionError e) {
160         info ("Run '%s --help' to see a full list of available command line options.\n", a
rgs[0]);
161         info ("error: %s\n", e.message);
162         return 0;
163     }
164
165     if (command_line_option_version) {
166         print ("Bookworm Version " + BookwormApp.Constants.bookworm_version + "\n");
167     }
168     if (command_line_option_discover) {
169         BookwormApp.BackgroundTasks.performTasks ();
170     }
171     return 0;
172 }
173
174 public override void activate () {
175     Logger.initialize ("com.github.babluboy.bookworm");
176     if (command_line_option_debug) {
177         Logger.DisplayLevel = LogLevel.DEBUG;
178     }
179     if (command_line_option_info) {
180         Logger.DisplayLevel = LogLevel.INFO;
181     }
182     info ("[START] [FUNCTION:activate]");
183     //proceed if Bookworm is not running already
184     if (!isBookwormRunning) {
185         debug ("No instance of Bookworm found");
186         window = new Gtk.ApplicationWindow (this);
187         default_theme = Gtk.IconTheme.get_default ();
188         //retrieve Settings
189         settings = BookwormApp.Settings.get_instance ();
190         //set window attributes
191         window.set_border_width (0);
192         window.get_style_context ().add_class ("rounded");
193         //set the minimum size of the window on minimize
194         window.set_size_request (600, 350);
195         //set css provider
196         cssProvider = new Gtk.CssProvider ();
197         loadCSSProvider (cssProvider);
198         //load images/icons
199         loadImages ();
200         //add window components
201         window.set_titlebar (BookwormApp.AppHeaderBar.create_headerbar ());
202
203         BookwormApp.AppWindow.createWelcomeScreen ();
204         bookWormUIBox = BookwormApp.AppWindow.createBoookwormUI ();
205
206         accel = new AccelGroup ();
207
208         shortcutAssocs = BookwormApp.ShortcutsAssocsHolder.readFromSettings ();
209         BookwormApp.Shortcuts.attachShortcutsToWidgets ();
210
211         /*

```

```

212         var accelMap = AccelMap.@get ();
213         accelMap.add_entry ("<Root-Window>/some-path", Gdk.Key.F10, Gdk.ModifierType.CONTR
214             OL_MASK);
214         accelMap.add_entry ("<Root-Window>/some-path", Gdk.Key.Cyrillic_TE, Gdk.ModifierTy
215             pe.CONTROL_MASK);
215         accelMap.add_entry ("<Root-Window>/some-path", Gdk.Key.Cyrillic_TE, Gdk.ModifierTy
215             pe.CONTROL_MASK | Gdk.ModifierType.SHIFT_MASK);
216         accel.connect_by_path ("<Root-Window>/some-path", () => {
217             info ("~~~~~ sunshine 2");
218             return false;
219         });
220     */
221     window.add_accel_group (accel);
222
223     //load saved books from DB and add them to Library view
224     loadBookwormState ();
225     //show welcome screen if no book is present in library instead of the normal libra
ry view
226     if (libraryViewMap.size == 0) {
227         window.add (welcomeWidget);
228     } else {
229         window.add (bookWormUIBox);
230     }
231     add_window (window);
232     window.show_all ();
233     toggleUIState ();
234     //capture window re-size events and save the window size
235     window.size_allocate.connect (() => {
236         saveWindowState ();
237     });
238     //Exit Application Event
239     window.destroy.connect (() => {
240         //Perform close down activities
241         closeBookWorm ();
242     });
243     //Add keyboard shortcuts on the window
244     window.add_events (Gdk.EventMask.KEY_PRESS_MASK);
245
246     window.window_state_event.connect (BookwormApp.AppWindow.handleWindowStateEvents);
247
248     isBookwormRunning = true;
249     debug ("Completed creating an instance of Bookworm");
250 } else {
251     window.present ();
252     debug ("An instance of Bookworm is already running");
253 }
254 //check if any books needed to be added/opened - if eBook (s) were opened from File Ex
plorer using Bookworm
255 if (commandLineArgs.length > 1) {
256     info ("Book (s) to be added/opened based on command line parameters. Size of comma
nd line attributes:" +
257         commandLineArgs.length.to_string ())
258     );
259     pathsOfBooksToBeAdded = new string[commandLineArgs.length];
260     pathsOfBooksToBeAdded = commandLineArgs;
261     //Display the progress bar
262     BookwormApp.AppWindow.bookAdditionBar.show ();
263     isBookBeingAddedToLibrary = true;
264     //handle the case if the welcome screen is shown
265     if (libraryViewMap.size == 0) {
266         //remove the welcome widget from main window
267         window.remove (welcomeWidget);
268         //add the library view to the window
269         window.add (bookWormUIBox);
270         bookWormUIBox.show_all ();
271         toggleUIState ();
272     }
273     //Update the library view with books - this returns control back immediately
274     BookwormApp.Library.addBooksToLibrary ();
275 }
276 //Perform post start up actions
277 BookwormApp.contentHandler.performStartUpActions ();
278 toggleUIState ();
279 info ("[END] [FUNCTION:activate]");

```

```

280     }
281
282     public override void open (File[] files, string hint) {
283         /* TODO */
284     }
285
286     public void loadImages () {
287         info ("[START] [FUNCTION:loadImages]");
288         try {
289             image_selection_option_small = new Gdk.Pixbuf.from_resource (BookwormApp.Constants
290 .SELECTION_OPTION_IMAGE_SMALL_LOCATION);
291             image_selection_checked_small = new Gdk.Pixbuf.from_resource (BookwormApp.Constants
292 .SELECTION_CHECKED_IMAGE_SMALL_LOCATION);
293             image_selection_transparent_small = new Gdk.Pixbuf.from_resource (BookwormApp.Constants
294 .SELECTION_CHECKED_IMAGE_SMALL_LOCATION);
295             image_selection_transparent_small.fill (0x00000000);
296             image_rating_1 = new Gdk.Pixbuf.from_resource (BookwormApp.Constants.RATING_1_IMAG
297 E_LOCATION);
298             image_rating_2 = new Gdk.Pixbuf.from_resource (BookwormApp.Constants.RATING_2_IMAG
299 E_LOCATION);
300             image_rating_3 = new Gdk.Pixbuf.from_resource (BookwormApp.Constants.RATING_3_IMAG
301 E_LOCATION);
302             image_rating_4 = new Gdk.Pixbuf.from_resource (BookwormApp.Constants.RATING_4_IMAG
303 E_LOCATION);
304             image_rating_5 = new Gdk.Pixbuf.from_resource (BookwormApp.Constants.RATING_5_IMAG
305 E_LOCATION);
306
307             if (Gtk.IconTheme.get_default ().has_icon ("object-select-symbolic")) {
308                 select_book_image = new Gtk.Image.from_icon_name ("object-select-symbolic", Gt
309 k.IconSize.MENU);
310             } else {
311                 select_book_image = new Gtk.Image.from_resource (BookwormApp.Constants.SELECT_
312 BOOK_ICON_IMAGE_LOCATION);
313             }
314             if (Gtk.IconTheme.get_default ().has_icon ("list-add-symbolic")) {
315                 add_book_image = new Gtk.Image.from_icon_name ("list-add-symbolic", Gt
316 k.IconSize.MENU);
317             } else {
318                 add_book_image = new Gtk.Image.from_resource (BookwormApp.Constants.ADD_BOOK_I
319 CON_IMAGE_LOCATION);
320             }
321             if (Gtk.IconTheme.get_default ().has_icon ("list-remove-symbolic")) {
322                 remove_book_image = new Gtk.Image.from_icon_name ("list-remove-symbolic", Gt
323 k.IconSize.MENU);
324             } else {
325                 remove_book_image = new Gtk.Image.from_resource (BookwormApp.Constants.REMOVE_
326 BOOK_ICON_IMAGE_LOCATION);
327             }
328             if (Gtk.IconTheme.get_default ().has_icon ("list-add-symbolic")) {
329                 add_scan_directory_image = new Gtk.Image.from_icon_name ("list-add-symbolic",
330 Gt
331 k.IconSize.MENU);
332             } else {
333                 add_scan_directory_image = new Gtk.Image.from_resource (BookwormApp.Constants.
334 ADD_BOOK_ICON_IMAGE_LOCATION);
335             }
336             if (Gtk.IconTheme.get_default ().has_icon ("list-remove-symbolic")) {
337                 remove_scan_directory_image = new Gtk.Image.from_icon_name ("list-remove-symb
338 lic", Gt
339 k.IconSize.MENU);
340             } else {
341                 remove_scan_directory_image = new Gtk.Image.from_resource (BookwormApp.Constants.
342 REMOVE_BOOK_ICON_IMAGE_LOCATION);
343             }
344             if (Gtk.IconTheme.get_default ().has_icon ("view-list-symbolic")) {
345                 library_list_button_image = new Gtk.Image.from_icon_name ("view-list-symbolic"
346 , Gt
347 k.IconSize.SMALL_TOOLBAR);
348             } else {
349                 library_list_button_image = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_res
350 ource_at_scale (BookwormApp.Constants.LIBRARY_VIEW_LIST_IMAGE_LOCATION, 16, 16, true));
351             }
352             if (Gtk.IconTheme.get_default ().has_icon ("view-grid-symbolic")) {
353                 library_grid_button_image = new Gtk.Image.from_icon_name ("view-grid-symbolic"
354 , Gt
355 k.IconSize.SMALL_TOOLBAR);
356             } else {
357                 library_grid_button_image = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_res
358 ource_at_scale (BookwormApp.Constants.LIBRARY_VIEW_GRID_IMAGE_LOCATION, 16, 16, true));

```

```

333         }
334         if (Gtk.IconTheme.get_default ().has_icon ("format-justify-left-symbolic")) {
335             pref_menu_icon_align_left = new Gtk.Image.from_icon_name ("format-justify-left-
336 -symbolic", Gtk.IconSize.SMALL_TOOLBAR);
337             } else {
338                 pref_menu_icon_align_left = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_res-
339 ource_at_scale (BookwormApp.Constants.TEXT_ALIGN_LEFT_ICON_LOCATION, 16, 16, true));
340             }
341             if (Gtk.IconTheme.get_default ().has_icon ("format-justify-right-symbolic")) {
342                 pref_menu_icon_align_right = new Gtk.Image.from_icon_name ("format-justify-rig-
343 ht-symbolic", Gtk.IconSize.SMALL_TOOLBAR);
344             } else {
345                 pref_menu_icon_align_right = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_re-
346 ource_at_scale (BookwormApp.Constants.TEXT_ALIGN_RIGHT_ICON_LOCATION, 16, 16, true));
347             }
348             if (Gtk.IconTheme.get_default ().has_icon ("help-info-symbolic")) {
349                 content_list_button_image = new Gtk.Image.from_icon_name ("help-info-symbolic"
350 , Gtk.IconSize.LARGE_TOOLBAR);
351             } else {
352                 content_list_button_image = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_res-
353 ource_at_scale (BookwormApp.Constants.BOOK_INFO_IMAGE_LOCATION, 24, 24, true) );
354             }
355             if (Gtk.IconTheme.get_default ().has_icon ("format-text-larger-symbolic")) {
356                 menu_icon_text_large = new Gtk.Image.from_icon_name ("format-text-larger-symbo-
357 lic", Gtk.IconSize.LARGE_TOOLBAR);
358             } else {
359                 menu_icon_text_large = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_resource_
360 _at_scale (BookwormApp.Constants.TEXT_LARGER_IMAGE_LOCATION, 24, 24, true) );
361             }
362             if (Gtk.IconTheme.get_default ().has_icon ("open-menu")) {
363                 menu_icon = new Gtk.Image.from_icon_name ("open-menu", Gtk.IconSize.LARGE_TOOL-
364 BAR);
365             } else {
366                 menu_icon = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_resource_at_scale (
367 BookwormApp.Constants.HEADERBAR_PROPERTIES_IMAGE_LOCATION, 24, 24, true) );
368             }
369             if (Gtk.IconTheme.get_default ().has_icon ("format-text-larger-symbolic")) {
370                 pref_menu_icon_text_large = new Gtk.Image.from_icon_name ("format-text-larger-
371 symbolic", Gtk.IconSize.LARGE_TOOLBAR);
372             } else {
373                 pref_menu_icon_text_large = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_res-
374 ource_at_scale (BookwormApp.Constants.TEXT_LARGER_IMAGE_LOCATION, 24, 24, true));
375             }
376             if (Gtk.IconTheme.get_default ().has_icon ("format-text-smaller-symbolic")) {
377                 pref_menu_icon_text_small = new Gtk.Image.from_icon_name ("format-text-smaller-
378 symbolic", Gtk.IconSize.LARGE_TOOLBAR);
379             } else {
380                 pref_menu_icon_text_small = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_res-
381 ource_at_scale (BookwormApp.Constants.TEXT_SMALLER_IMAGE_LOCATION, 24, 24, true));
382             }
383             if (Gtk.IconTheme.get_default ().has_icon ("go-previous-symbolic")) {
384                 back_button_image = new Gtk.Image.from_icon_name ("go-previous-symbolic", Gtk.I-
385 conSize.MENU);
386             } else {
387                 back_button_image = new Gtk.Image.from_resource (BookwormApp.Constants.PREV_PA-
388 GE_ICON_IMAGE_LOCATION);
389             }
390             if (Gtk.IconTheme.get_default ().has_icon ("go-next-symbolic")) {
391                 forward_button_image = new Gtk.Image.from_icon_name ("go-next-symbolic", Gtk.I-
392 conSize.MENU);
393             } else {
394                 forward_button_image = new Gtk.Image.from_resource (BookwormApp.Constants.NEXT-
395 _PAGE_ICON_IMAGE_LOCATION);
396             }
397             if (Gtk.IconTheme.get_default ().has_icon ("go-previous-symbolic")) {
398                 back_page_image = new Gtk.Image.from_icon_name ("go-previous-symbolic", Gtk.I-
399 conSize.MENU);
400             } else {
401                 back_page_image = new Gtk.Image.from_resource (BookwormApp.Constants.PREV_PAGE-
402 _ICON_IMAGE_LOCATION);
403             }
404             if (Gtk.IconTheme.get_default ().has_icon ("go-next-symbolic")) {
405                 forward_page_image = new Gtk.Image.from_icon_name ("go-next-symbolic", Gtk.I-
406 conSize.MENU);
407             } else {

```

```

387             forward_page_image = new Gtk.Image.from_resource (BookwormApp.Constants.NEXT_P
388             AGE_ICON_IMAGE_LOCATION);
389         }
390     } catch (GLib.Error e) {
391         warning ("Image could not be loaded. Error:" + e.message);
392     }
393     info ("[END] [FUNCTION:loadImages]");
394 }
395
396 public static void loadCSSProvider (Gtk.CssProvider cssProvider) {
397     info ("[START] [FUNCTION:loadCSSProvider] cssProvider=" + cssProvider.to_string ());
398     string dynamicCSSContent = "";
399     try {
400         profileColorList = settings.list_of_profile_colors.split ",";
401         //temp check to ensure the change to settings for colours increasing from 6 to 9 i
402         s handled
403         if (profileColorList.length < 9) {
404             profileColorList = {"#000000", "#fbfbfb", "#E8ED00", "#586e75", "#fdf6e3", "#87FF2B
405             ", "#93ala1", "#002b36", "#3465A4"};
406         }
407         dynamicCSSContent = BookwormApp.Constants.DYNAMIC_CSS_CONTENT
408             .replace ("<profile_1_color>",profileColorList[0])
409             .replace ("<profile_1_bgcolor>",profileColorList[1])
410             .replace ("<profile_2_color>",profileColorList[3])
411             .replace ("<profile_2_bgcolor>",profileColorList[4])
412             .replace ("<profile_3_color>",profileColorList[6])
413             .replace ("<profile_3_bgcolor>",profileColorList[7]);
414         debug ("CSS for Profile Buttons:" + dynamicCSSContent);
415         cssProvider.load_from_data (dynamicCSSContent, dynamicCSSContent.length);
416     } catch (GLib.Error e) {
417         warning ("Stylesheet could not be loaded from CSS Content[" + dynamicCSSContent +
418         "]. Error:" + e.message);
419     }
420
421     Gtk.StyleContext.add_provider_for_screen (
422         Gdk.Screen.get_default (), cssProvider, Gtk.STYLE_PROVIDER_PRIORITY_APPLICATION);
423     info ("[END] [FUNCTION:loadCSSProvider]");
424 }
425
426 public void loadBookwormState () {
427     info ("[START] [FUNCTION:loadBookwormState]");
428     //check and create required directory structure
429     BookwormApp.Utils.fileOperations ("CREATEDIR", BookwormApp.Constants.EBOOK_EXTRACTION_
430     LOCATION, "", "");
431     BookwormApp.Utils.fileOperations ("CREATEDIR", BookwormApp.Bookworm.bookworm_config_pa
432     th, "", "");
433     BookwormApp.Utils.fileOperations ("CREATEDIR", BookwormApp.Bookworm.bookworm_config_pa
434     th + "/covers/", "", "");
435     BookwormApp.Utils.fileOperations ("CREATEDIR", BookwormApp.Bookworm.bookworm_config_pa
436     th + "/books/", "", "");
437     //Set the window to the last saved position
438     if (settings.pos_x == 0 && settings.pos_y == 0) {
439         window.set_position (Gtk.WindowPosition.CENTER);
440     } else {
441         window.move (settings.pos_x, settings.pos_y);
442     }
443     //set window size to the last saved height/width
444     if (settings.window_is_maximized) {
445         window.maximize ();
446     } else {
447         if (settings.window_width > 0 && settings.window_height > 0) {
448             window.set_default_size (settings.window_width, settings.window_height);
449         } else {
450             window.set_default_size (1200, 700);
451         }
452     }
453     //check last state and turn on dark theme
454     if (BookwormApp.Bookworm.settings.is_dark_theme_enabled) {
455         Gtk.Settings.get_default ().gtk_application_prefer_dark_theme = true;
456     }
457     //set the number of books per library page as per user set value
458     no_of_books_per_page = settings.library_page_items.to_string ();
459
460     //check if the database exists otherwise create database and required tables
461     BookwormApp.DB.initializeBookWormDB (BookwormApp.Bookworm.bookworm_config_path);
462     //set the library view

```

```

454     BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = settings.library_view_mode;
455     //Fetch details of Books from the database
456     BookwormApp.Bookworm.paginationlist.add ("");
457     BookwormApp.Bookworm.current_page_counter = 0;
458     BookwormApp.Library.paginateLibrary ("","PAGINATED_SEARCH");
459     //Set the library pagination buttons based on the paginate call
460     BookwormApp.AppWindow.handleLibraryPageButtons ("", false);
461
462     info ("[END] [FUNCTION:loadBookwormState]");
463 }
464
465 public async void closeBookWorm () {
466     info ("[START] [FUNCTION:closeBookWorm]");
467     //If Bookworm was closed while in Reading mode - save book details for subsequent read
468     if (BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[1]) {
469         //Save the page scroll position of the book being read
470         (libraryViewMap.get (locationOfEBookCurrentlyRead)).setBookScrollPos (BookwormApp.
471         contentHandler.getScrollPos ());
472         //Save the path to the book being read
473         settings.book_being_read = locationOfEBookCurrentlyRead;
474     } else {
475         //Bookworm was not in reading view during close - remove path of book read last
476         settings.book_being_read = "";
477     }
478     //release the control so that the window is closed
479     Idle.add (closeBookWorm.callback);
480     yield;
481     //Update the book details to the database if it was opened in this session
482     foreach (var book in libraryViewMap.values) {
483         if (((BookwormApp.Book)book).getWasBookOpened ()) {
484             BookwormApp.DB.updateBookToDataBase ((BookwormApp.Book)book);
485             debug ("Completed saving the book data into DB");
486         }
487     }
488     //Run dicovery of books as a background task if not already running
489     string checkBackgroundTask = BookwormApp.Utils.execute_sync_command ("ps -ef");
490     if (checkBackgroundTask.index_of ("bookworm --discover") == -1) {
491         BookwormApp.Utils.execute_async_multiaarg_command_pipes ({"com.github.babluboy.book
492         worm", "--discover", "&"});
493     } else {
494         debug ("Bookworm discover process already running....");
495     }
496     info ("[END] [FUNCTION:closeBookWorm]");
497 }
498
499 public void saveWindowState () {
500     int width;
501     int height;
502     int x;
503     int y;
504     window.get_size (out width, out height);
505     window.get_position (out x, out y);
506     if (settings.pos_x != x || settings.pos_y != y) {
507         settings.pos_x = x;
508         settings.pos_y = y;
509     }
510     if (settings.window_width != width || settings.window_height != height) {
511         settings.window_width = width;
512         settings.window_height = height;
513     }
514     if (window.is_maximized == true) {
515         settings.window_is_maximized = true;
516     } else {
517         settings.window_is_maximized = false;
518     }
519     settings.zoom_level = BookwormApp.AppWindow.aWebView.get_zoom_level ();
520 }
521
522 public static void readSelectedBook (owned BookwormApp.Book aBook) {
523     info ("[START] [FUNCTION:readSelectedBook] book.location=" + aBook.getBookLocation ());
524 ;
525     //Fetch the book meta data from the database if it is not already available
526     if (aBook.getBookContentList ().size < 1) {
527         //content size should be greater than 1 if the book data has been loaded
528         aBook = BookwormApp.DB.getBookMetaDataFromDB (aBook);

```

```

526     }
527     debug ("Book details before attempting to open book for reading:" + aBook.to_string (
528     ));
528     //Handle the case when the page number of the book is not set
529     if (aBook.getBookPageNumber () == -1) {
530         aBook.setBookPageNumber (0);
531     } else {
532         //This book was previously being read, so it should be opened at the last reading
533         //position
534         //Enable the flag which will scroll the page to the last read position
535         isPageScrollRequired = true;
536     }
537     //Handle the case when the page number of the book is outside limits
538     if (aBook.getBookPageNumber () >= aBook.getBookContentList ().size) {
539         aBook.setBookPageNumber (aBook.getBookContentList ().size - 1);
540     }
541     //check if the extracted contents for the book exists
542     if (BookwormApp.Bookworm.settings.is_local_storage_enabled &&
543         "true" == BookwormApp.Utils.fileOperations ("DIR_EXISTS", aBook.getBookExtractionL
ocation (), "", "") &&
544         aBook.getBookContentList () != null &&
545         aBook.getBookContentList ().size > 0 &&
546         aBook.getBookContentList ().size >= aBook.getBookPageNumber () &&
547         "true" == BookwormApp.Utils.fileOperations (
548             "EXISTS", BookwormApp.Utils.decodeHTMLChars (
549                 aBook.getBookContentList ().get (aBook.getBookPageNumber ())), "", ""));
550     //extraction of book not required
551     aBook.setIsBookParsed (true);
552     debug ("Book has already been extracted and the extracted contents exist at:" + aB
ook.getBookExtractionLocation ());
553 } else {
554     //Extract and Parse the eBook (this will overwrite the contents already extracted)
555     debug ("Extracted contents of the book was not found at expected location [" +
556         aBook.getBookExtractionLocation () + "], attempting to load book from original
location [" +
557         aBook.getBookLocation () + "]");
558     aBook = genericParser (aBook);
559     //If ebook was not parsed successfully, show the warning info banner message
560     if (!aBook.getIsBookParsed ()) {
561         BookwormApp.AppWindow.showInfoBar (aBook, MessageType.WARNING);
562     }
563     //progress in opening the book for reading if it has been parsed successfully
564     if (aBook.getIsBookParsed ()) {
565         //update the total number of pages in the book
566         aBook.setBookTotalPages (aBook.getBookContentList ().size);
567         //update book to mark it has been opened in this session
568         aBook.setBookLastModificationDate ((new DateTime.now_utc ().to_unix ()).to_string
());
569         aBook.setWasBookOpened (true);
570         //update book details to libraryView Map
571         libraryViewMap.set (aBook.getBookLocation (), aBook);
572         locationOfEBBookCurrentlyRead = aBook.getBookLocation ();
573         //Update header title
574         BookwormApp.AppHeaderBar.get_headerbar ().title = BookwormApp.Utils.parseMarkUp (a
Book.getBookTitle ());
575         //change the application view to Book Reading mode
576         BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_UI_STATES[1];
577         toggleUIState ();
578         //reset the contents of the search entry
579         BookwormApp.Info.searchresults_scroll.get_child ().destroy ();
580         //set the max value and the current value of the page slider
581         BookwormApp.AppWindow.pageAdjustment.set_upper (aBook.getBookContentList ().size);
582         BookwormApp.AppWindow.pageAdjustment.set_value (aBook.getBookPageNumber ());
583         //render the contents of the current page of book
584         aBook = BookwormApp.contentHandler.renderPage (aBook, "");
585         //set the focus to the webview to capture keypress events
586         BookwormApp.AppWindow.aWebView.grab_focus ();
587     }
588     info ("[END] [FUNCTION:readSelectedBook] book.location=" + aBook.getBookLocation ());
589 }
590
591     public static void toggleUIState () {
592         info ("[START] [FUNCTION:toggleUIState] bookworm current state:" + BookwormApp.Bookw
orm.BOOKWORM_CURRENT_STATE);

```

```

593     //hide the inforbar if there is no text in it
594     if (BookwormApp.AppWindow.infobarLabel.get_text ().length < 1) {
595         BookwormApp.AppWindow.infobar.hide ();
596     }
597     //Set-up UI specific for Library Grid View Mode
598     if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[0] ||
599         BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[2] ||
600         BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[3])
601     {
602         BookwormApp.AppWindow.library_list_scroll.set_visible (false);
603         BookwormApp.AppWindow.library_grid_scroll.set_visible (true);
604         BookwormApp.AppWindow.library_grid.show_all ();
605         BookwormApp.AppWindow.bookLibrary_ui_box.grab_focus ();
606     }
607     //Set-up UI specific for Library List View Mode
608     if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[5] ||
609         BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[6] ||
610         BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[7])
611     {
612         BookwormApp.AppWindow.library_grid_scroll.set_visible (false);
613         BookwormApp.AppWindow.library_list_scroll.set_visible (true);
614         BookwormApp.AppWindow.library_table_treeview.show_all ();
615         BookwormApp.AppWindow.library_table_treeview.grab_focus ();
616     }
617     //Set-up UI for Library View Mode (List or Grid)
618     if (BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[0] ||
619         BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[2] ||
620         BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[3] ||
621         BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[5])
622     {
623         BookwormApp.AppHeaderBar.headerSearchBar.set_placeholder_text (
624             BookwormApp.Constants.TEXT_FOR_HEADERBAR_LIBRARY_SEARCH);
625         library_mode_button.set_visible (true);
626         content_list_button.set_visible (false);
627         library_view_button.set_visible (false);
628         BookwormApp.AppWindow.bookLibrary_ui_box.set_visible (true);
629         BookwormApp.AppWindow.bookReading_ui_box.set_visible (false);
630         BookwormApp.Info.info_box.set_visible (false);
631         prefButton.set_visible (false);
632         BookwormApp.AppHeaderBar.bookmark_active_button.set_visible (false);
633         BookwormApp.AppHeaderBar.bookmark_inactive_button.set_visible (false);
634         if (!isBookBeingAddedToLibrary) {
635             BookwormApp.AppWindow.bookAdditionBar.hide ();
636         }
637     }
638     //Set-up UI for Reading Mode
639     if (BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[1]) {
640         BookwormApp.AppHeaderBar.headerSearchBar.set_placeholder_text (
641             BookwormApp.Constants.TEXT_FOR_HEADERBAR_BOOK_SEARCH);
642         library_mode_button.set_visible (false);
643         content_list_button.set_visible (true);
644         library_view_button.set_visible (true);
645         library_view_button.set_label (BookwormApp.Constants.TEXT_FOR_LIBRARY_BUTTON);
646         BookwormApp.AppWindow.bookLibrary_ui_box.set_visible (false);
647         BookwormApp.AppWindow.bookReading_ui_box.set_visible (true);
648         BookwormApp.Info.info_box.set_visible (false);
649         prefButton.set_visible (true);
650         BookwormApp.contentHandler.handleBookMark ("DISPLAY");
651         BookwormApp.AppWindow.bookAdditionBar.hide ();
652     }
653     //Set-up UI for Info Mode - Meta Data / Table of Content
654     if (BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[4]) {
655         BookwormApp.AppHeaderBar.headerSearchBar.set_placeholder_text (
656             BookwormApp.Constants.TEXT_FOR_HEADERBAR_LIBRARY_SEARCH);
657         BookwormApp.Info.info_box.show_all ();
658         library_mode_button.set_visible (false);
659         content_list_button.set_visible (false);
660         library_view_button.set_visible (true);
661         library_view_button.set_label (BookwormApp.Constants.TEXT_FOR_RESUME_BUTTON);

```

```

662         BookwormApp.AppWindow.bookLibrary_ui_box.set_visible (false);
663         BookwormApp.AppWindow.bookReading_ui_box.set_visible (false);
664         BookwormApp.Info.info_box.set_visible (true);
665         BookwormApp.Info.stack.set_visible_child_name (settings.current_info_tab);
666         prefButton.set_visible (false);
667         BookwormApp.AppHeaderBar.bookmark_active_button.set_visible (false);
668         BookwormApp.AppHeaderBar.bookmark_inactive_button.set_visible (false);
669         BookwormApp.AppWindow.bookAdditionBar.hide ();
670     }
671     info ("[END] [FUNCTION:toggleUIState] bookworm current state:" + BookwormApp.Bookworm.
672     BOOKWORM_CURRENT_STATE);
673 }
674
675     public static BookwormApp.Book genericParser (owned BookwormApp.Book aBook) {
676         info ("[START] [FUNCTION:genericParser] book.location=" + aBook.getBookLocation ());
677         //set defaults for title and author - this will be over-ridden with extracted data if
678         found
679         aBook.setBookAuthor (BookwormApp.Constants.TEXT_FOR_UNKNOWN_TITLE);
680         aBook.setBookTitle (BookwormApp.Constants.TEXT_FOR_UNKNOWN_TITLE);
681         //check if ebook is present at provided location
682         if ("false" == BookwormApp.Utils.fileOperations ("EXISTS", "", aBook.getBookLocation (
683             ), "")) {
684             warning ("EBook not found at provided location:" + aBook.getBookLocation ());
685             aBook.setIsBookParsed (false);
686             aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_EXTRACTION_ISSUE);
687             return aBook;
688         }
689         //determine the extension of the ebook file
690         string ebookFileName = (File.new_for_path (aBook.getBookLocation ()).get_basename ());
691         if (ebookFileName.index_of (".") != -1) {
692             string fileExtension = ebookFileName.substring (ebookFileName.last_index_of ("."))
693             .up ();
694             //parse file based on extension found
695             try {
696                 switch (fileExtension) {
697                     case ".EPUB":
698                         aBook = BookwormApp.ePubReader.parseEPubBook (aBook);
699                         break;
700                     case ".PDF":
701                         aBook = BookwormApp.pdfReader.parsePDFBook (aBook);
702                         break;
703                     case ".CBR":
704                         aBook = BookwormApp.comicsReader.parseComicsBook (aBook, fileExtension
705                         );
706                         break;
707                     case ".CBZ":
708                         aBook = BookwormApp.comicsReader.parseComicsBook (aBook, fileExtension
709                         );
710                         break;
711                     case ".MOBI":
712                         aBook = BookwormApp.mobiReader.parseMobiBook (aBook);
713                         break;
714                     case ".PRC":
715                         aBook = BookwormApp.mobiReader.parseMobiBook (aBook);
716                         break;
717                     case ".AZW3":
718                         aBook = BookwormApp.mobiReader.parseMobiBook (aBook);
719                         break;
720                     case ".ZIP":
721                         //check if the file is a zipped FB2 file
722                         if (ebookFileName.up ().last_index_of (".FB2.ZIP") != -1) {
723                             aBook = BookwormApp.fb2Reader.parseFictionBook (aBook);
724                             break;
725                         } else {
726                             aBook.setIsBookParsed (false);
727                             aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_FORMAT_NOT_S
728                             UPPORTED);
729                         }
730                     }
731                     case ".FB2":
732                         aBook = BookwormApp.fb2Reader.parseFictionBook (aBook);
733                         break;
734                     default:
735                         aBook.setIsBookParsed (false);
736                         aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_FORMAT_NOT_SUPPO

```

```
RTED);
730                     break;
731                 }
732             } catch (GLib.Error e) {
733                 info ("Error while parsing book: %s\n", e.message);
734                 aBook.setIsBookParsed (false);
735                 aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_PARSING_ISSUE);
736             }
737         }
738         //If the book could not be parsed, add the book location to the list of books which co
uld not be loaded
739         if (!aBook.getIsBookParsed ()) {
740             BookwormApp.Bookworm.pathsOfBooksNotAddedStr.append (aBook.getBookLocation ()).app
end ("~~");
741         }
742         info ("[END] [FUNCTION:genericParser] book.is book parsed=" + aBook.getIsBookParsed ()
.to_string ());
743         return aBook;
744     }
745 }
```

```
=====
book.vala ? Book Model
=====

1  /* Copyright 2017 Siddhartha Das (bablu.boy@gmail.com)
2  *
3  * This file is part of Bookworm and has the getter/setter methods
4  * used for holding the state of the book
5  *
6  * Bookworm is free software: you can redistribute it
7  * and/or modify it under the terms of the GNU General Public License as
8  * published by the Free Software Foundation, either version 3 of the
9  * License, or (at your option) any later version.
10 *
11 * Bookworm is distributed in the hope that it will be
12 * useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
13 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
14 * Public License for more details.
15 *
16 * You should have received a copy of the GNU General Public License along
17 * with Bookworm. If not, see http://www.gnu.org/licenses/.
18 */
19 using Gtk;
20 using Gee;
21 public class BookwormApp.Book {
22     //These variables are persisted in the database
23     private int bookId = 0;
24     private bool isBookParsedCorrectly = false;
25     private string parsingIssue = "";
26     private string bookLocation = "";
27     private string bookCoverLocation = "";
28     private string bookExtractionLocation = "";
29     private string bookTitle = "";
30     private string bookAuthor = "";
31     private string bookTags = "";
32     private string annotationTags = "";
33     private string bookPublishDate = "";
34     private string bookCreationDate = "";
35     private string bookLastModificationDate = "";
36     private int bookPageNumber = -1;
37     private int bookTotalPages = 1;
38     private int bookScrollPosition = -1;
39     private int bookRating = 0;
40     private bool isBookCoverImagePresent = false;
41     private StringBuilder bookmarks = new StringBuilder ("");
42     private TreeMap<string,string> annotationMap = new TreeMap<string,string> ();
43     private ArrayList<string> bookContentList = new ArrayList<string> ();
44     private ArrayList<HashMap<string,string>> TOCMap = new ArrayList<HashMap<string,string>> (
);
45
46     //These variables are only available for the current session (not persisted)
47     private string opffFileLocation = "";
48     private string baseLocationOfContents = "";
49     private bool ifPageForward = true;
50     private bool ifPageBackward = true;
51     private bool isBookSelected = false;
52     private bool wasBookOpened = false;
53     private string bookAnchor = "";
54     private HashMap<string,Gtk.Widget> bookWidgetsList = new HashMap<string,Gtk.Widget> ();
55
56     //getter list for book id
57     public void setBookId (int aBookId) {
58         bookId = aBookId;
59     }
60     public int getBookId () {
61         return bookId;
62     }
63
64     //getter list for isBookParsedCorrectly
65     public void setIsBookParsed (bool isParsed) {
66         isBookParsedCorrectly = isParsed;
67     }
68     public bool getIsBookParsed () {
69         return isBookParsedCorrectly;
```

```

70     }
71
72     //getter list for book location
73     public void setParsingIssue (string aParsingIssue) {
74         parsingIssue = aParsingIssue;
75     }
76     public string getParsingIssue () {
77         return parsingIssue;
78     }
79
80     //getter list for book location
81     public void setBookLocation (string aBookLocation) {
82         bookLocation = aBookLocation;
83     }
84     public string getBookLocation () {
85         return bookLocation;
86     }
87
88     //getter list for book cover image location
89     public void setBookCoverLocation (string aBookCoverLocation) {
90         bookCoverLocation = aBookCoverLocation;
91     }
92     public string getBookCoverLocation () {
93         return bookCoverLocation;
94     }
95
96     //getter setter for content list of book parts
97     public void setBookContentList (string contentList) {
98         bookContentList.add (contentList);
99     }
100    public ArrayList<string> getBookContentList () {
101        return bookContentList;
102    }
103    public void clearBookContentList () {
104        bookContentList.clear ();
105    }
106
107    //getter setter for Table Of Contents
108    public void setTOC (HashMap<string,string> toc) {
109        TOCMap.add (toc);
110    }
111    public ArrayList<HashMap<string,string>> getTOC () {
112        return TOCMap;
113    }
114    public void clearTOC () {
115        TOCMap.clear ();
116    }
117
118    //getter setter for temp location of ebook contents
119    public void setBookExtractionLocation (string aBookExtractionLocation) {
120        bookExtractionLocation = aBookExtractionLocation;
121    }
122    public string getBookExtractionLocation () {
123        return bookExtractionLocation;
124    }
125
126    //getter setter for book title
127    public void setBookTitle (string aBookTitle) {
128        bookTitle = aBookTitle;
129    }
130    public string getBookTitle () {
131        return bookTitle;
132    }
133
134    //getter list for book rating
135    public void setBookRating (int aBookRating) {
136        bookRating = aBookRating;
137    }
138    public int getBookRating () {
139        return bookRating;
140    }
141
142    //getter setter for book author
143    public void setBookAuthor (string aBookAuthor) {
144        bookAuthor = aBookAuthor;

```

```

145     }
146     public string getBookAuthor () {
147         return bookAuthor;
148     }
149
150     //getter setter for book tags
151     public void setBookTags (string aBookTags) {
152         bookTags = aBookTags;
153     }
154     public string getBookTags () {
155         return bookTags;
156     }
157
158     //getter setter for annotation tags
159     public void setAnnotationTags (string anAnnotationTags) {
160         annotationTags = anAnnotationTags;
161     }
162     public string getAnnotationTags () {
163         return annotationTags;
164     }
165
166     //getter setter for location of books OPF file
167     public void setOPFFileLocation (string aOPFFileLocation) {
168         opfFileLocation = aOPFFileLocation;
169     }
170     public string getOPFFileLocation () {
171         return opfFileLocation;
172     }
173
174     //getter setter for base location of eBook file contents
175     public void setBaseLocationOfContents (string aBaseLocationOfContents) {
176         baseLocationOfContents = aBaseLocationOfContents;
177     }
178     public string getBaseLocationOfContents () {
179         return baseLocationOfContents;
180     }
181
182     //getter setter for presence of Cover Location
183     public void setIsBookCoverImagePresent (bool isABookCoverImagePresent) {
184         isBookCoverImagePresent = isABookCoverImagePresent;
185     }
186     public bool getIsBookCoverImagePresent () {
187         return isBookCoverImagePresent;
188     }
189
190     //getter list for book location
191     public void setBookPublishDate (string aBookPublishDate) {
192         bookPublishDate = aBookPublishDate;
193     }
194     public string getBookPublishDate () {
195         return bookPublishDate;
196     }
197
198     //getter list for book location
199     public void setBookCreationDate (string aBookCreationDate) {
200         bookCreationDate = aBookCreationDate;
201     }
202     public string getBookCreationDate () {
203         return bookCreationDate;
204     }
205
206     //getter list for book location
207     public void setBookLastModificationDate (string aBookLastModificationDate) {
208         bookLastModificationDate = aBookLastModificationDate;
209     }
210     public string getBookLastModificationDate () {
211         return bookLastModificationDate;
212     }
213
214     //getter setter for eBook pageNumber
215     public void setBookPageNumber (int aBookPageNumber) {
216         bookPageNumber = aBookPageNumber;
217     }
218     public int getBookPageNumber () {
219         return bookPageNumber;

```

```

220     }
221
222     //getter setter for total pages in book
223     public void setBookTotalPages (int aBookTotalPages) {
224         bookTotalPages = aBookTotalPages;
225     }
226     public int getBookTotalPages () {
227         return bookTotalPages;
228     }
229
230     //getter setter for eBook vertical scroll position
231     public void setBookScrollPos (int aBookScrollPos) {
232         bookScrollPosition = aBookScrollPos;
233     }
234     public int getBookScrollPos () {
235         return bookScrollPosition;
236     }
237
238     //getter setter if eBook pageForward is possible
239     public void setIfPageForward (bool ifBookPageForward) {
240         ifPageForward = ifBookPageForward;
241     }
242     public bool getIfPageForward () {
243         return ifPageForward;
244     }
245
246     //getter setter if eBook pageBackward is possible
247     public void setIfPageBackward (bool ifBookPageBackward) {
248         ifPageBackward = ifBookPageBackward;
249     }
250     public bool getIfPageBackward () {
251         return ifPageBackward;
252     }
253
254     //getter setter for determining if the book is selected
255     public void setIsBookSelected (bool aIsBookSelected) {
256         isBookSelected = aIsBookSelected;
257     }
258     public bool getIsBookSelected () {
259         return isBookSelected;
260     }
261
262     //getter setter for determining if the book was read in this session
263     public void setWasBookOpened (bool aWasBookOpened) {
264         wasBookOpened = aWasBookOpened;
265     }
266     public bool getWasBookOpened () {
267         return wasBookOpened;
268     }
269
270     //getter setter for bookmarks
271     public void setBookmark (int pageNumber, string action) {
272         if ("ACTIVE_CLICKED" == action) {
273             bookmarks.assign (bookmarks.str.replace ("***" + pageNumber.to_string () + "***", ""))
274         }
275         if ("INACTIVE_CLICKED" == action) {
276             bookmarks.append ("***" + pageNumber.to_string () + "***");
277         }
278         if (pageNumber == -10) { //this is used to set the bookmark fetched from the DB
279             //set -10 as the "pageNumber" and the book mark data as "action" when setting this
280             //value from the DB
281             bookmarks.assign (action);
282         }
283     }
284     public string getBookmark () {
285         return bookmarks.str;
286     }
287
288     //getter setter for annotations
289     public void setAnnotations (string index, string annotationText) {
290         //check the value of the annotation text - add or delete accordingly
291         if (annotationText != null && annotationText.length > 0) {
292             //annotated text is not null/empty - update the annotation
293             annotationMap.set (index, annotationText);

```

```

293     } else {
294         //annotated text is null/empty - remove the annotation
295         if (annotationMap.has_key (index)) {
296             annotationMap.unset (index);
297         }
298     }
299 }
300 public string getAnnotations (string index) {
301     if (annotationMap.has_key (index)) {
302         return annotationMap.get (index);
303     } else {
304         return "";
305     }
306 }
307 public TreeMap<string,string> getAnnotationList () {
308     return annotationMap;
309 }
310 public void setAnnotationList (TreeMap<string,string> aTreeMap) {
311     annotationMap.set_all (aTreeMap);
312 }
313
314 //getter setter for list of Gtk Widgets used for a Book
315 public void setBookWidget (string name, Gtk.Widget aWidget) {
316     bookWidgetsList.set (name, aWidget);
317 }
318 public Gtk.Widget getBookWidget (string name) {
319     return bookWidgetsList.get (name);
320 }
321
322 //getter setter for book anchor
323 public void setAnchor (string aBookAnchor) {
324     bookAnchor = aBookAnchor;
325 }
326 public string getAnchor () {
327     return bookAnchor;
328 }
329
330 //print book details
331 public string to_string () {
332     StringBuilder bookDetails = new StringBuilder ();
333     bookDetails.append ("bookId=").append (bookId.to_string ()).append (",\n")
334         .append ("bookLocation=").append (bookLocation).append (",\n")
335         .append ("bookCoverLocation=").append (bookCoverLocation).append (",\n")
336         .append ("bookExtractionLocation=").append (bookExtractionLocation).append (",\n")
337         .append ("bookTitle=" + bookTitle).append (",\n")
338         .append ("opfFileLocation=").append (opfFileLocation).append (",\n")
339         .append ("baseLocationOfContents=").append (baseLocationOfContents).append (",\n")
340         .append ("bookPublishDate=" + bookPublishDate).append (",\n")
341         .append ("isBookCoverImagePresent=").append (isBookCoverImagePresent.to_string ())
342         .append (",\n")
343         .append ("bookCreationDate=").append (bookCreationDate).append (",\n")
344         .append ("bookLastModificationDate=").append (bookLastModificationDate).append (",
345 \n")
346         .append ("bookPageNumber=").append (bookPageNumber.to_string ()).append (",\n")
347         .append ("ifPageForward=").append (ifPageForward.to_string ()).append (",\n")
348         .append ("ifPageBackward=").append (ifPageBackward.to_string ()).append (",\n")
349         .append ("bookmarks=").append (bookmarks.str).append (",\n")
350         .append ("author=").append (bookAuthor).append (",\n")
351         .append ("ratings=").append (bookRating.to_string ()).append (",\n")
352         .append ("tags=").append (bookTags.to_string ()).append (",\n")
353         .append ("annotation tags=").append (annotationTags.to_string ()).append (",\n")
354         .append ("bookContentList=");
355     for (int i = 0; i < bookContentList.size; i++) {
356         bookDetails.append ("[" + i.to_string () + "]=" + bookContentList.get (i) + ",");
357     }
358 }

```

```
=====
library.vala ? Library View
=====

1  /* Copyright 2017 Siddhartha Das (bablu.boy@gmail.com)
2  *
3  * This file is part of Bookworm and manages the library and
4  * views associated with the library
5  *
6  * Bookworm is free software: you can redistribute it
7  * and/or modify it under the terms of the GNU General Public License as
8  * published by the Free Software Foundation, either version 3 of the
9  * License, or (at your option) any later version.
10 *
11 * Bookworm is distributed in the hope that it will be
12 * useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
13 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
14 * Public License for more details.
15 *
16 * You should have received a copy of the GNU General Public License along
17 * with Bookworm. If not, see http://www.gnu.org/licenses/.
18 */
19 using Gtk;
20 using Gee;
21 public class BookwormApp.Library {
22     public static ArrayList<BookwormApp.Book> listOfBooksInLibraryOnLoad = new ArrayList<Bookw
ormApp.Book> ();
23
24     public static void updateLibraryView (owned BookwormApp.Book aBook) {
25         info ("[START] [FUNCTION:updateLibraryView]");
26         //add book details to libraryView Map
27         BookwormApp.Bookworm.libraryViewMap.set (aBook.getBookLocation (), aBook);
28         //update the library views
29         updateLibraryListView (aBook);
30         updateLibraryGridView (aBook);
31         info ("[END] [FUNCTION:updateLibraryView]");
32     }
33
34     public static void updateLibraryListView (owned BookwormApp.Book aBook) {
35         debug ("[START] [FUNCTION:updateLibraryListView] book.location=" + aBook.getBookLocati
on ());
36         //if (aBook.getBookTitle != null && aBook.getBookTitle ().length > 1) {
37             debug ("Started updating Library List View for book:" + aBook.getBookLocation ());
38             //set the rating image
39             Gdk.Pixbuf image_rating;
40             string modifiedElapsedTime = "";
41             switch (aBook.getBookRating ().to_string ()) {
42                 case "1":
43                     image_rating = BookwormApp.Bookworm.image_rating_1;
44                     break;
45                 case "2":
46                     image_rating = BookwormApp.Bookworm.image_rating_2;
47                     break;
48                 case "3":
49                     image_rating = BookwormApp.Bookworm.image_rating_3;
50                     break;
51                 case "4":
52                     image_rating = BookwormApp.Bookworm.image_rating_4;
53                     break;
54                 case "5":
55                     image_rating = BookwormApp.Bookworm.image_rating_5;
56                     break;
57                 default:
58                     image_rating = null;
59                     break;
60             }
61             //calculate the time elapsed from last modified DateTime
62             TimeSpan timespan = (new DateTime.now_local ()).difference (
63                 new DateTime.from_unix_local (int64.parse (aBook.getBookLastModificationDate (
64 ))));
64             int64 daysElapsed = timespan/ (86400000000);
65             if (timespan < TimeSpan.DAY) {
66                 modifiedElapsedTime = BookwormApp.Constants.TEXT_FOR_TIME_TODAY;
67             } else if (timespan < 2 * TimeSpan.DAY) {
```

```

68             modifiedElapsed = BookwormApp.Constants.TEXT_FOR_TIME_YESTERDAY;
69         } else if (timespan < 30 * TimeSpan.DAY) {
70             modifiedElapsed = daysElapsed.to_string () + " " + BookwormApp.Constants.T
EXT_FOR_TIME_DAYS;
71         } else {
72             modifiedElapsed = new DateTime.from_unix_local (int64.parse (aBook.getBook
LastModificationDate ())).format ("%d %m %Y");
73         }
74         BookwormApp.AppWindow.library_table_liststore.append (out BookwormApp.AppWindow.li
brary_table_iter);
75         BookwormApp.AppWindow.library_table_liststore.set (BookwormApp.AppWindow.library_t
able_iter,
76             0, null,
77             1, BookwormApp.Utils.parseMarkUp (aBook.getBookTitle ()),
78             2, aBook.getBookAuthor (),
79             3, modifiedElapsed,
80             4, image_rating,
81             5, aBook.getBookTags (),
82             6, aBook.getBookRating ().to_string (),
83             7, aBook.getBookLocation ());
84         BookwormApp.Bookworm.libraryTreeModelFilter = new Gtk.TreeModelFilter (BookwormApp
.AppWindow.library_table_liststore, null);
85         //BookwormApp.Bookworm.libraryTreeModelFilter.set_visible_func (filterTree);
86         Gtk.TreeModelSort aTreeModelSort = new TreeModelSort.with_model (BookwormApp.Bookw
orm.libraryTreeModelFilter);
87         BookwormApp.AppWindow.library_table_treeview.set_model (aTreeModelSort);
88         //set treeview columns for sorting
89         BookwormApp.AppWindow.library_table_treeview.get_column (1).set_sort_column_id (1)
;
90         BookwormApp.AppWindow.library_table_treeview.get_column (1).set_sort_order (SortTy
pe.DESCENDING);
91         BookwormApp.AppWindow.library_table_treeview.get_column (2).set_sort_column_id (2)
;
92         BookwormApp.AppWindow.library_table_treeview.get_column (2).set_sort_order (SortTy
pe.DESCENDING);
93         BookwormApp.AppWindow.library_table_treeview.get_column (3).set_sort_column_id (3)
;
94         BookwormApp.AppWindow.library_table_treeview.get_column (3).set_sort_order (SortTy
pe.DESCENDING);
95         //6th item is the rating value corresponding to the image on the 4th item
96         BookwormApp.AppWindow.library_table_treeview.get_column (4).set_sort_column_id (6)
;
97         BookwormApp.AppWindow.library_table_treeview.get_column (4).set_sort_order (SortTy
pe.DESCENDING);
98         BookwormApp.AppWindow.library_table_treeview.get_column (5).set_sort_column_id (5)
;
99         BookwormApp.AppWindow.library_table_treeview.get_column (5).set_sort_order (SortTy
pe.DESCENDING);
100        //}
101        debug ("[END] [FUNCTION:updateLibraryListView] book.location=" + aBook.getBookLocation
());
102    }
103
104    public static void updateLibraryGridView (owned BookwormApp.Book aBook) {
105        debug ("[START] [FUNCTION:updateLibraryGridView] book.location=" + aBook.getBookLocati
on ());
106        debug ("Started updating Library Grid View for book:" + aBook.getBookLocation ());
107        Gtk.Image aCoverImage = new Gtk.Image ();
108        Gtk.Label titleTextLabel = new Gtk.Label ("");
109        Gtk.Image bookSelectionImage;
110        Gtk.Image bookSelectedImage;
111        string bookCoverLocation;
112        Gdk.Pixbuf aBookCover;
113        Gtk.Image bookPlaceholderCoverImage = null;
114        try {
115            Gdk.Pixbuf bookPlaceholderCoverPix = new Gdk.Pixbuf.from_resource_at_scale (
116                BookwormApp.Constants.PLACEHOLDER_COVER_IMAGE_LOCATION, 10, 200, false);
117            bookPlaceholderCoverImage = new Gtk.Image.from_pixbuf (bookPlaceholderCoverPix);
118        } catch (GLib.Error e) {
119            warning ("Error loading the placeholder cover image from location[" +
120                BookwormApp.Constants.PLACEHOLDER_COVER_IMAGE_LOCATION + "] : " +
121                e.message );
122        }
123        Gtk.ProgressBar bookProgressBar = new Gtk.ProgressBar ();
124        //Add a default cover selected at random if no cover exists

```

```

125         if (aBook.getBookCoverLocation () == null ||
126             aBook.getBookCoverLocation ().length < 1 ||
127             "true" != BookwormApp.Utils.fileOperations ("EXISTS", "", aBook.getBookCoverLocati
on (), ""))
128     {
129         aBook.setIsBookCoverImagePresent (false);
130         //default Book Cover Image not set - select at random from the default covers
131         bookCoverLocation = BookwormApp.Constants.DEFAULT_COVER_IMAGE_LOCATION
132             .replace ("N", GLib.Random.int_range (1, 6).to_string ());
133         aBook.setBookCoverLocation (bookCoverLocation);
134         try {
135             aBookCover = new Gdk.Pixbuf.from_resource_at_scale (aBook.getBookCoverLocation
(), 150, 200, false);
136                 aCoverImage = new Gtk.Image.from_pixbuf (aBookCover);
137             } catch (GLib.Error e) {
138                 warning ("Error in loading default cover image at location [" + aBook.getBookC
overLocation () + "]: " + e.message);
139             }
140         } else {
141             try {
142                 aBookCover = new Gdk.Pixbuf.from_file_at_scale (aBook.getBookCoverLocation (), 150, 200, false);
143                     aCoverImage = new Gtk.Image.from_pixbuf (aBookCover);
144                 } catch (GLib.Error e) {
145                     //Sometimes the path to the image selected by the parser is not a image
146                     //This catch block assigns a default cover selected at random to cover this is
sue
147                     bookCoverLocation = BookwormApp.Constants.DEFAULT_COVER_IMAGE_LOCATION
148                         .replace ("N", GLib.Random.int_range (1, 6).to_string ());
149                     aBook.setBookCoverLocation (bookCoverLocation);
150                     aCoverImage = null;
151                     try {
152                         aBookCover = new Gdk.Pixbuf.from_resource_at_scale (aBook.getBookCoverLoca
tion (), 150, 200, false);
153                             aCoverImage = new Gtk.Image.from_pixbuf (aBookCover);
154                             //set cover image present flag to false - this will add title text to the
default cover
155                             aBook.setIsBookCoverImagePresent (false);
156                             aCoverImage.set_halign (Align.CENTER);
157                             aCoverImage.set_valign (Align.CENTER);
158                         } catch (GLib.Error e) {
159                             warning ("Error in loading cover image at location [" + aBook.getBookCover
Location () + "]: " + e.message);
160                         }
161                     }
162                 }
163                 //Add title of the book if Default Cover is being used
164                 if (!aBook.getIsBookCoverImagePresent ()) {
165                     string title = aBook.getBookTitle ();
166                     if (title == null || title.length < 1) {
167                         title = BookwormApp.Constants.TEXT_FOR_UNKNOWN_TITLE;
168                     }
169                     //replace special chars from title
170                     title = title.replace ("&", "and");
171                     title = BookwormApp.Utils.minimizeStringLength (title, 4 * BookwormApp.Constants.M
AX_NUMBER_OF_CHARS_FOR_BOOK_TITLE);
172                     titleTextLabel.set_text (
173                         "<b>" + BookwormApp.Utils.breakString (title, BookwormApp.Constants.MAX_NUMBER
_OF_CHARS_FOR_BOOK_TITLE, "\n") + "</b>");
174                     titleTextLabel.set_use_markup (true);
175                     titleTextLabel.set_line_wrap (true);
176                     titleTextLabel.set_justify (Justification.CENTER);
177                     titleTextLabel.set_margin_start (BookwormApp.Constants.SPACING_WIDGETS);
178                     titleTextLabel.set_margin_end (BookwormApp.Constants.SPACING_WIDGETS);
179                 } else {
180                     //remove the title label if the book has a cover image available
181                     titleTextLabel.set_text ("");
182                 }
183                 //Add selection option badge to the book for later use
184                 Gdk.Pixbuf bookSelectionPix = null;
185                 try {
186                     bookSelectionPix = new Gdk.Pixbuf.from_resource (BookwormApp.Constants.SELECTION_O
PTION_IMAGE_LOCATION);
187                 } catch (GLib.Error e) {
188                     warning ("Error in loading Book selection image from location [" +

```

```

189             BookwormApp.Constants.SELECTION_OPTION_IMAGE_LOCATION + "]: " + e.message);
190         }
191         bookSelectionImage = new Gtk.Image.from_pixbuf (bookSelectionPix);
192         bookSelectionImage.set_halign (Align.CENTER);
193         bookSelectionImage.set_valign (Align.START);
194         //Add selection checked badge to the book for later use
195         Gdk.Pixbuf bookSelectedPix = null;
196         try {
197             bookSelectedPix = new Gdk.Pixbuf.from_resource (BookwormApp.Constants.SELECTION_CHECKED_IMAGE_LOCATION);
198         } catch (GLib.Error e) {
199             warning ("Error in loading Book Selection Checked image from location[" +
200                     BookwormApp.Constants.SELECTION_CHECKED_IMAGE_LOCATION + "] :" + e.message);
201         }
202         bookSelectedImage = new Gtk.Image.from_pixbuf (bookSelectedPix);
203         bookSelectedImage.set_halign (Align.CENTER);
204         bookSelectedImage.set_valign (Align.START);
205         //Set the value of the progress bar
206         double progress = 0.0;
207         bookProgressBar.set_halign (Align.CENTER);
208         bookProgressBar.set_valign (Align.END);
209         bookProgressBar.set_visible (false);
210         //protect the progress bar against the show_all called on the library view
211         bookProgressBar.set_no_show_all (true);
212         //Create a Overlay to hold the images in the right order
213         Gtk.Overlay aOverlayImage = new Gtk.Overlay ();
214         aOverlayImage.add (bookPlaceholderCoverImage);
215         aOverlayImage.add_overlay (bookSelectionImage);
216         aOverlayImage.add_overlay (bookSelectedImage);
217         aOverlayImage.add_overlay (aCoverImage);
218         aOverlayImage.add_overlay (titleTextLabel);
219         aOverlayImage.add_overlay (bookProgressBar); //this will be invisible until mouse enters
220         //Add the overlaid images to a EventBox to allow mouse click actions to be captured
221         Gtk.EventBox aEventBox = new Gtk.EventBox ();
222         aEventBox.set_border_width (BookwormApp.Constants.SPACING_WIDGETS/2);
223         aEventBox.set_name (aBook.getBookLocation ());
224         aEventBox.add (aOverlayImage);
225         //register the book with the filter function
226         var aFlowBoxChild = new Gtk.FlowBoxChild ();
227         aFlowBoxChild.add (aEventBox);
228         //add the book to the library view
229         BookwormApp.AppWindow.library_grid.add (aFlowBoxChild);
230         //set gtk widgets into the Book object for later manipulation
231         aBook.setBookWidget ("PLACEHOLDER_COVER_IMAGE", bookPlaceholderCoverImage);
232         aBook.setBookWidget ("COVER_IMAGE", aCoverImage);
233         aBook.setBookWidget ("TITLE_TEXT_LABEL", titleTextLabel);
234         aBook.setBookWidget ("SELECTED_BADGE_IMAGE", bookSelectedImage);
235         aBook.setBookWidget ("SELECTION_BADGE_IMAGE", bookSelectionImage);
236         aBook.setBookWidget ("BOOK_EVENTBOX", aEventBox);
237         aBook.setBookWidget ("BOOK_OVERLAY_IMAGE", aOverlayImage);
238         //Create a popover context menu for the book
239         Gtk.Popover bookPopover = BookwormApp.AppDialog.createBookContextMenu (aBook);
240         //add mouse enter listener for book object
241         aEventBox.enter_notify_event.connect ((event) => {
242             //calculate the progress of the book
243             progress = ((double)aBook.getBookPageNumber () + 1)/aBook.getBookTotalPages ();
244             bookProgressBar.set_fraction (progress);
245             bookProgressBar.set_visible (true);
246             return false;
247         });
248         //add mouse exit listener for book object
249         aEventBox.leave_notify_event.connect ((event) => {
250             //Checking for Gdk.NotifyType.INFERIOR resolves the unwanted leave event fired due
to the cover being a default type image
251             if (event.detail != Gdk.NotifyType.INFERIOR) {
252                 bookProgressBar.set_visible (false);
253             }
254             return false;
255         });
256         //add mouse click listener for book objects based on mode
257         aEventBox.button_press_event.connect ((event) => {
258             //capture which mouse button was clicked on the book in the library
259             uint mouseButtonClicked;
260             event.get_button (out mouseButtonClicked);
261             //handle right button click for context menu

```

```

262         if (event.get_event_type () == Gdk.EventType.BUTTON_PRESS && mouseButtonClicked ==
263     3) {
264             bookPopover.set_visible (true);
265             bookPopover.show_all ();
266             return true;
267         } else {
268             //left button click for reading or selection of book
269             if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
270                 ORM_UI_STATES[0]) {
271                 aBook = BookwormApp.Bookworm.libraryViewMap.get (aEventBox.get_name ());
272                 BookwormApp.Bookworm.readSelectedBook (aBook);
273             }
274             if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
275                 ORM_UI_STATES[2] || BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
276                 ORM_UI_STATES[3])
277             {
278                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKW
279                 ORM_UI_STATES[3];
280                 aBook = BookwormApp.Bookworm.libraryViewMap.get (aEventBox.get_name ());
281                 updateGridViewForSelection (aBook);
282             }
283             return true;
284         });
285         //Show the grid view based on state
286         if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
287             TATES[0] || BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
288             TATES[2] || BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
289             TATES[3])
290         {
291             BookwormApp.AppWindow.library_grid.show_all ();
292         }
293         debug ("Completed updating Library View for book:" + aBook.getBookLocation ());
294     }
295
296     public static void replaceCoverImageOnBook (owned BookwormApp.Book? book) {
297         debug ("[START] [FUNCTION:replaceCoverImageOnBook]");
298         //remove the existing overlay image
299         Gtk.Overlay oldOverlayImage = (Gtk.Overlay) book.getBookWidget ("BOOK_OVERLAY_IMAGE");
300         oldOverlayImage.destroy ();
301         //create a new overlay image
302         Gtk.Overlay lOverlayImage = new Gtk.Overlay ();
303         lOverlayImage.add (book.getBookWidget ("PLACEHOLDER_COVER_IMAGE"));
304         lOverlayImage.add_overlay (book.getBookWidget ("SELECTION_BADGE_IMAGE"));
305         lOverlayImage.add_overlay (book.getBookWidget ("SELECTED_BADGE_IMAGE"));
306         lOverlayImage.add_overlay (book.getBookWidget ("COVER_IMAGE"));
307         lOverlayImage.add_overlay (book.getBookWidget ("TITLE_TEXT_LABEL"));
308         book.setBookWidget ("BOOK_OVERLAY_IMAGE", lOverlayImage);
309         //associate the eventbox with the new overlay image
310         Gtk.EventBox aEventBox = (Gtk.EventBox) book.getBookWidget ("BOOK_EVENTBOX");
311         aEventBox.add (lOverlayImage);
312         book.setBookWidget ("BOOK_EVENTBOX", aEventBox);
313         //update the libraryview map with the book object
314         BookwormApp.Bookworm.libraryViewMap.set (book.getBookLocation (), book);
315         debug ("[END] [FUNCTION:replaceCoverImageOnBook]");
316     }
317
318     public static void updateListViewForSelection (owned BookwormApp.Book? lBook) {
319         debug ("[START] [FUNCTION:updateListViewForSelection] " +
320             "Updating List View Selection Badges for mode:" +
321             BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE);
322         Gtk.TreeModelForeachFunc print_row = (model, path, iter) => {
323             GLib.Value bookLocationAtRow;
324             BookwormApp.AppWindow.library_table_liststore.get_value (iter, 7, out bookLoc
325             atRow);
326             BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap.get ((string) bookLoc
327             atRow);
328             if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_
329                 UI_STATES[5]) {
330                 BookwormApp.AppWindow.library_table_liststore.set_value (iter, 0, BookwormApp.
331                 Bookworm.image_selection_transparent_small);
332                 aBook.setIsBookSelected (false);

```

```

325             BookwormApp.AppWindow.controlDeletionButton (false);
326         }
327         if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_
    UI_STATES[6]) {
328             BookwormApp.AppWindow.library_table_liststore.set_value (iter, 0, BookwormApp.
    Bookworm.image_selection_option_small);
329             aBook.setIsBookSelected (false);
330             BookwormApp.AppWindow.controlDeletionButton (false);
331         }
332         if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_
    UI_STATES[7] &&
333             (string) bookLocationAtRow == lBook.getBookLocation ())
334         {
335             if (!lBook.getIsBookSelected ()) {
336                 BookwormApp.AppWindow.library_table_liststore.set_value (iter, 0, Bookworm
    App.Bookworm.image_selection_checked_small);
337                 aBook.setIsBookSelected (true);
338                 BookwormApp.AppWindow.controlDeletionButton (true);
339             } else {
340                 BookwormApp.AppWindow.library_table_liststore.set_value (iter, 0, Bookworm
    App.Bookworm.image_selection_option_small);
341                 aBook.setIsBookSelected (false);
342                 BookwormApp.AppWindow.controlDeletionButton (false);
343             }
344         }
345         //update the book into the Library view HashMap
346         BookwormApp.Bookworm.libraryViewMap.set (aBook.getBookLocation (), aBook);
347         return false;
348     };
349     BookwormApp.AppWindow.library_table_liststore.foreach (print_row);
350     debug ("[END] [FUNCTION:updateListViewForSelection]");
351 }
352
353     public static void updateGridViewForSelection (owned BookwormApp.Book? lBook) {
354         debug ("[START] [FUNCTION:updateGridViewForSelection]");
355         if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
    TATES[0]) {
356             debug ("Updating Library View for Selection Badges BOOKWORM_UI_STATES[0]");
357             Gee.HashMap<string, BookwormApp.Book> temp_libraryViewMap = new Gee.HashMap<string
    , BookwormApp.Book> ();
358             //loop over HashMap of Book Objects and overlay selection image
359             foreach (BookwormApp.Book book in BookwormApp.Bookworm.libraryViewMap.values) {
360                 if (BookwormApp.AppWindow.library_grid_scroll.get_visible ()) {
361                     Gtk.Overlay aOverlayImage = (Gtk.Overlay) book.getBookWidget ("BOOK_OVERLAY_IM
    AGE");
362                     //Align the selection badges to the center so that they are not visible
363                     book.getBookWidget ("SELECTION_BADGE_IMAGE").set_halign (Align.CENTER);
364                     book.getBookWidget ("SELECTED_BADGE_IMAGE").set_halign (Align.CENTER);
365                     //set the order of the widgets to put the selection/selected badges at bot
    tom
366                     aOverlayImage.reorder_overlay (book.getBookWidget ("SELECTION_BADGE_IMAGE"
    ), 1);
367                     aOverlayImage.reorder_overlay (book.getBookWidget ("SELECTED_BADGE_IMAGE")
    , 2);
368                     aOverlayImage.reorder_overlay (book.getBookWidget ("COVER_IMAGE"), 3);
369                     aOverlayImage.reorder_overlay (book.getBookWidget ("TITLE_TEXT_LABEL"), 4)
    ;
370                 }
371                 temp_libraryViewMap.set (book.getBookLocation (), book);
372             }
373             //Iterate over all books and make the selection flag for each book as false
374             //This is to cover the scenario when a book was selected and the selection mode wa
    s changed without deleting the book
375             foreach (BookwormApp.Book aBook in temp_libraryViewMap.values) {
376                 aBook.setIsBookSelected (false);
377                 BookwormApp.Bookworm.libraryViewMap.set (aBook.getBookLocation (), aBook);
378             }
379         }
380         if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
    TATES[2]) {
381             debug ("Updating Library View for Selection Badges BOOKWORM_UI_STATES[2]");
382             Gee.HashMap<string, BookwormApp.Book> temp_libraryViewMap = new Gee.HashMap<string
    , BookwormApp.Book> ();
383             //loop over HashMap of Book Objects and overlay selection badge
384             foreach (BookwormApp.Book book in BookwormApp.Bookworm.libraryViewMap.values) {

```

```

385             if (BookwormApp.AppWindow.library_grid_scroll.get_visible ()) {
386                 Gtk.Overlay aOverlayImage = (Gtk.Overlay) book.getBookWidget ("BOOK_OVERLA
387                     Y_IMAGE");
388                     //Align the selection badges to the right to make visible but the selected
389                     badges should be centered to keep hidden
390                         book.getBookWidget ("SELECTION_BADGE_IMAGE").set_halign (Align.START);
391                         book.getBookWidget ("SELECTED_BADGE_IMAGE").set_halign (Align.CENTER);
392                         //set the order of the widgets to put the selection badge on top
393                         aOverlayImage.reorder_overlay (book.getBookWidget ("SELECTED_BADGE_IMAGE")
394 , 1);
395                         aOverlayImage.reorder_overlay (book.getBookWidget ("COVER_IMAGE"), 2);
396                         aOverlayImage.reorder_overlay (book.getBookWidget ("TITLE_TEXT_LABEL"), 3)
397 ;
398                         aOverlayImage.reorder_overlay (book.getBookWidget ("SELECTION_BADGE_IMAGE"
399 ), 4);
400         }
401         temp_libraryViewMap.set (book.getBookLocation (), book);
402     }
403     //Iterate over all books and make the selection flag for each book as false
404     //This is to cover the scenario when a book was selected and the selection mode wa
405     s changed without deleting the book
406     foreach (BookwormApp.Book aBook in temp_libraryViewMap.values) {
407         aBook.setIsBookSelected (false);
408         BookwormApp.Bookworm.libraryViewMap.set (aBook.getBookLocation (), aBook);
409     }
410     if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
411     TATES[3]) {
412         debug ("Updating Library View for Selection Badges BOOKWORM_UI_STATES[3]");
413         if (lBook != null) {
414             if (BookwormApp.AppWindow.library_grid_scroll.get_visible ()) {
415                 Gtk.Overlay aOverlayImage = (Gtk.Overlay) lBook.getBookWidget ("BOOK_OVERL
416                     AY_IMAGE");
417                     if (!lBook.getIsBookSelected ()) { //Do work for selecting this book
418                         //Align the selected badges to the right to make visible but keep the
419                         selection badge centered to keep hidden
420                             lBook.getBookWidget ("SELECTED_BADGE_IMAGE").set_halign (Align.START);
421                             lBook.getBookWidget ("SELECTION_BADGE_IMAGE").set_halign (Align.CENTER
422 );
423                         //set the order of the widgets to put the selected badge on top
424                         aOverlayImage.reorder_overlay (lBook.getBookWidget ("SELECTION_BADGE_I
425                     MAGE"), 1);
426                         aOverlayImage.reorder_overlay (lBook.getBookWidget ("COVER_IMAGE"), 2)
427 ;
428                         aOverlayImage.reorder_overlay (lBook.getBookWidget ("TITLE_TEXT_LABEL"
429 ), 3);
430                         aOverlayImage.reorder_overlay (lBook.getBookWidget ("SELECTED_BADGE_IM
431                     AGE"), 4);
432                         lBook.setIsBookSelected (true);
433                         BookwormApp.AppWindow.controlDeletionButton (true);
434                     } else { //Do work for de-selecting this book
435                         //set the order of the widgets to put the selection badge on top
436                         //Align the selection badges to the right to make visible but keep the
437                         selected badges centered to keep hidden
438                             lBook.getBookWidget ("SELECTION_BADGE_IMAGE").set_halign (Align.START)
439 ;
440                             lBook.getBookWidget ("SELECTED_BADGE_IMAGE").set_halign (Align.CENTER)
441 ;
442                             aOverlayImage.reorder_overlay (lBook.getBookWidget ("SELECTED_BADGE_IM
443                     AGE"), 1);
444                             aOverlayImage.reorder_overlay (lBook.getBookWidget ("COVER_IMAGE"), 2)
445 ;
446                             aOverlayImage.reorder_overlay (lBook.getBookWidget ("TITLE_TEXT_LABEL"
447 ), 3);
448                             aOverlayImage.reorder_overlay (lBook.getBookWidget ("SELECTION_BADGE_I
449                     MAGE"), 4);
450                         lBook.setIsBookSelected (false);
451                         BookwormApp.AppWindow.controlDeletionButton (false);
452                     }
453                 }
454                 //update the book into the Library view HashMap
455                 BookwormApp.Bookworm.libraryViewMap.set (lBook.getBookLocation (), lBook);
456             }
457         }
458     }
459     debug ("[END] [FUNCTION:updateGridViewForSelection]");

```

```

439     }
440
441     public static void removeSelectedBooksFromLibrary () {
442         debug ("[START] [FUNCTION:removeSelectedBooksFromLibrary]");
443         ArrayList<string> listOfBooksToBeRemoved = new ArrayList<string> ();
444         //loop through the Library View Hashmap and remove the selected books from the Library
445         View Model
446         foreach (BookwormApp.Book book in BookwormApp.Bookworm.libraryViewMap.values) {
447             //check if the book selection flag to true and add it to removal list
448             if (book.getIsBookSelected ()) {
449                 //hold the books to be deleted in a list
450                 listOfBooksToBeRemoved.add (book.getBookLocation ());
451                 Gtk.EventBox lEventBox = (Gtk.EventBox) book.getBookWidget ("BOOK_EVENTBOX");
452                 //destroy the EventBox parent widget - this removes the book from the library
453                 grid
454                 lEventBox.get_parent ().destroy ();
455                 //destroy the EventBox widget
456                 lEventBox.destroy ();
457                 //remove the cover image if it exists (ignore default covers)
458                 if (book.getBookCoverLocation ().index_of (
459                     BookwormApp.Constants.DEFAULT_COVER_IMAGE_LOCATION.replace ("-cover-N.svg"
460                     , "") ) == -1)
461                 {
462                     BookwormApp.Utils.execute_sync_command ("rm \"\" + book.getBookCoverLocatio
463                     n () + "\"");
464                 }
465                 //update the onloadBookList - this is to enable re-adding the book within
466                 the same session
467                 BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr
468                     .assign (BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr.str
469                     .replace (book.getBookLocation (), ""));
470                 BookwormApp.Library.listOfBooksInLibraryOnLoad.remove (book);
471             }
472         }
473         if (listOfBooksToBeRemoved.size > 0) {
474             //loop through the rows in the treeview and remove the selected books
475             ArrayList<Gtk.TreeIter?> listOfIterstoBeRemoved = new ArrayList<Gtk.TreeIter?> ();
476             Gtk.TreeModelForeachFunc print_row = (model, path, iter) => {
477                 GLib.Value bookLocationAtRow;
478                 BookwormApp.AppWindow.library_table_liststore.get_value (iter, 7, out bookLoca
479                 tionAtRow);
480                 if ((string) bookLocationAtRow in listOfBooksToBeRemoved) {
481                     listOfIterstoBeRemoved.add (iter);
482                 }
483                 return false;
484             };
485             BookwormApp.AppWindow.library_table_liststore.foreach (print_row);
486             foreach (Gtk.TreeIter iterToBeRemoved in listOfIterstoBeRemoved) {
487                 //remove item for list store - vala_36 compatibility wrapper
488                 #if VALA_0_36
489                     BookwormApp.AppWindow.library_table_liststore.remove (ref iterToBeRemoved)
490                 ;
491                 #else
492                     BookwormApp.AppWindow.library_table_liststore.remove (iterToBeRemoved);
493                 #endif
494             }
495             //loop through the removed books and remove them from the Library View Hashmap, local
496             //cache and Database
497             foreach (string bookLocation in listOfBooksToBeRemoved) {
498                 BookwormApp.DB.removeBookFromDB (BookwormApp.Bookworm.libraryViewMap.get (bookLoca
499                 tion));
500                 BookwormApp.Bookworm.libraryViewMap.unset (bookLocation);
501             }
502             //Set to normal grid view if the current view is in any of the Grid View State
503             if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
504                 TATES[0] ||
505                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
506                 TATES[2] ||
507                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
508                 TATES[3])
509             {
510                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_UI_ST
511                 ATES[0];
512                 BookwormApp.Library.updateGridViewForSelection (null);

```

```

501         }
502         //Set to normal list view if the current view is in any of the List View State
503         if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
504             TATES[5] || BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
505             TATES[6] || BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
506             TATES[7]) {
507             BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_UI_ST
508             ATES[5];
509             BookwormApp.Library.updateListViewForSelection (null);
510             }
511             BookwormApp.Bookworm.toggleUIState ();
512             debug ("[END] [FUNCTION:removeSelectedBooksFromLibrary]");
513         }
514
515         public static async void updateLibraryViewFromDB () {
516             debug ("[START] [FUNCTION:updateLibraryViewFromDB]");
517             foreach (BookwormApp.Book book in listOfBooksInLibraryOnLoad) {
518                 //add the book to the UI - both grid and list view
519                 BookwormApp.Library.updateLibraryView (book);
520                 Idle.add (updateLibraryViewFromDB.callback);
521                 yield;
522             }
523             debug ("[END] [FUNCTION:updateLibraryViewFromDB]");
524         }
525
526         public static async void addBooksToLibrary () {
527             debug ("[START] [FUNCTION:addBooksToLibrary]");
528             debug ("books to be added=" + BookwormApp.Bookworm.pathsOfBooksToBeAdded.length.to_s
529             ing ());
530             double progress = 0d;
531             //loop through the command line and add books to library
532             foreach (string pathToSelectedBook in BookwormApp.Bookworm.pathsOfBooksToBeAdded) {
533                 debug ("Attempting to add book from path:" + pathToSelectedBook);
534                 //Set async callback only if multiple books are being added
535                 //If only one book is being added, complete parsing and adding the book,
536                 //so that it will be added to the BookwormApp.Bookworm.libraryViewMap and opened o
537                 n the
538                 //BookwormApp.contentHandler.performStartUpActions method
539                 if (BookwormApp.Bookworm.pathsOfBooksToBeAdded.length > 2) {
540                     Idle.add (addBooksToLibrary.callback);
541                 }
542                 BookwormApp.Bookworm.noOfBooksAddedFromCommand++;
543                 if (BookwormApp.Constants.bookworm_id != pathToSelectedBook.strip ()) {
544                     //set progress for the UI Book addition progress bar
545                     progress = (((double) (BookwormApp.Bookworm.noOfBooksAddedFromCommand)) / ((doub
546 le) (BookwormApp.Bookworm.pathsOfBooksToBeAdded.length)));
547                     BookwormApp.AppWindow.bookAdditionBar.set_text (_("Adding ") +
548                     ((int) (progress * 100)).to_string () + "% : " + File.new_for_path (pathTo
549 SelectedBook).get_basename ());
550                     BookwormApp.AppWindow.bookAdditionBar.set_fraction (progress);
551                 }
552                 //Return control back for any further actions only if multiple books are being add
553                 ed
554                 //If only one book is being added, complete parsing and adding the book,
555                 //so that it will be added to the BookwormApp.Bookworm.libraryViewMap and opened o
556                 n the
557                 //BookwormApp.contentHandler.performStartUpActions method
558                 if (BookwormApp.Bookworm.pathsOfBooksToBeAdded.length > 2) {
559                     yield;
560                 }
561                 //ignore the first command which is the application name
562                 if (BookwormApp.Constants.bookworm_id != pathToSelectedBook.strip ()) {
563                     //check if book already exists in the library
564                     if (BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr.str.index_of (pathToSe
565 lectedBook.strip ()) != -1) {
566                         debug ("Book already exists in library..." + BookwormApp.Bookworm.pathsOfB
567 ooksInLibraryOnLoadStr.str);
568                         //Enable the flag which will scroll the page to the last read position
569                         BookwormApp.Bookworm.isPageScrollRequired = true;
570                         //set the name of the book being currently read
571                         BookwormApp.Bookworm.locationOfEBookCurrentlyRead = pathToSelectedBook.str
572 ip ();

```

```

563         } else {
564             //book does not exist in library - create a new instance for the book
565             BookwormApp.Book aBookBeingAdded = new BookwormApp.Book ();
566             aBookBeingAdded.setBookLocation (pathToSelectedBook.strip ());
567             //the book will be updated to the libraryViewMap within the addBookToLibrary
568             try function
569                 //however the libraryViewMap will only be fully populated when all books are
570                 //added to it
571                 addBookToLibrary (aBookBeingAdded);
572                 //update the onloadBookList - this is to prevent re-adding the book within
573                 //the same session
574                 BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr.append (aBookBeingAdded
575                     .getBookLocation ());
576                 BookwormApp.Library.listOfBooksInLibraryOnLoad.add (aBookBeingAdded);
577             }
578         }
579     }
580     //Hide the progress bar on completion of adding books
581     BookwormApp.AppWindow.bookAdditionBar.hide ();
582     BookwormApp.Bookworm.isBookBeingAddedToLibrary = false;
583     BookwormApp.Bookworm.noOfBooksAddedFromCommand = 0;
584     debug ("[END] [FUNCTION:addBooksToLibrary]");
585 }
586
587 public static void addBookToLibrary (owned BookwormApp.Book aBook) {
588     debug ("[START] [FUNCTION:addBookToLibrary] book.location=" + aBook.getBookLocation ());
589     //check if the selected eBook exists
590     string eBookLocation = aBook.getBookLocation ();
591     File eBookFile = File.new_for_path (eBookLocation);
592     if (eBookFile.query_exists () && eBookFile.query_file_type (0) != FileType.DIRECTORY)
593     {
594         //insert book details to database and fetch the ID
595         int bookID = BookwormApp.DB.addBookToDataBase (aBook);
596         aBook.setBookId (bookID);
597         /*Other than location, nothing is inserted into the DB for the book at this time.
598         Mark book as opened in the session so that details for book are updated
599         into DB when the application is closed - eBook parsing happens after the initial insert
600         */
601         aBook.setWasBookOpened (true);
602         //parse eBook to populate cache and book meta data
603         aBook = BookwormApp.Bookworm.genericParser (aBook);
604         if (!aBook.getIsBookParsed ()) {
605             BookwormApp.DB.removeBookFromDB (aBook);
606             BookwormApp.AppWindow.showInfoBar (aBook, MessageType.WARNING);
607         } else {
608             //add eBook cover image to library view
609             BookwormApp.Library.updateLibraryView (aBook);
610             //Set to normal grid view if the current view is in any of the Grid View State
611             if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
612                 _ORM_UI_STATES[0] ||
613                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
614                 _ORM_UI_STATES[2] ||
615                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
616                 _ORM_UI_STATES[3])
617                 {
618                     BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKW
619                     _RM_UI_STATES[0];
620                     BookwormApp.Library.updateGridViewForSelection (null);
621                 }
622                 //Set to normal list view if the current view is in any of the List View State
623                 if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
624                     _ORM_UI_STATES[5] ||
625                     BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
626                     _ORM_UI_STATES[6] ||
627                     BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
628                     _ORM_UI_STATES[7])
629                     {
630                         BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKW
631                         _RM_UI_STATES[5];
632                         BookwormApp.Library.updateListViewForSelection (null);
633                     }

```

```

622         BookwormApp.Bookworm.toggleUIState ();
623         //set the name of the book being currently read
624         BookwormApp.Bookworm.locationOfEBookCurrentlyRead = eBookLocation;
625         debug ("Completed adding book to ebook library. Number of books in library:" +
626             BookwormApp.Bookworm.libraryViewMap.size.to_string ());
627     }
628 } else {
629     debug ("No ebook found for adding to library");
630 }
631 debug ("[END] [FUNCTION:addBookToLibrary] book.location=" + aBook.getBookLocation ());
632 }
633
634 public static void paginateLibrary (string library_search_data, string mode) {
635     //update the current set of books into the library db
636     foreach (var book in BookwormApp.Bookworm.libraryViewMap.values) {
637         if (((BookwormApp.Book)book).getWasBookOpened ()) {
638             BookwormApp.DB.updateBookToDataBase ((BookwormApp.Book)book);
639             debug ("Completed saving the book data into DB");
640         }
641     }
642     if (mode == "LIBRARY_SEARCH") {
643         //Perform library search for results
644         debug ("Executing library search query with criteria: " + library_search_data);
645         BookwormApp.Library.listOfBooksInLibraryOnLoad = BookwormApp.DB.getBooksFromDB (li
646         brary_search_data, mode);
647     } else {
648         //Query DB for the next/prev page
649         debug ("Executing paginated query for books with " +
650             "current_page_counter: " + BookwormApp.Bookworm.current_page_counter.to_string
651             () +
652             " on paginationlist: " + string.joinv ("", ", ", (BookwormApp.Bookworm.paginationl
653             ist.to_array ()));
654         BookwormApp.Library.listOfBooksInLibraryOnLoad = BookwormApp.DB.getBooksFromDB (
655             BookwormApp.Bookworm.paginationlist.get (BookwormApp.Bookworm.current_page_cou
656             nter), mode);
657     }
658     //check for the condition where no books are returned from the DB for the page criteri
659     a
660     if (BookwormApp.Library.listOfBooksInLibraryOnLoad.size != 0) {
661         //Remove books currently on grid view
662         GLib.List<weak Gtk.Widget> children_grid = BookwormApp.AppWindow.library_grid.get_
663         children ();
664         foreach (Gtk.Widget element in children_grid) {
665             BookwormApp.AppWindow.library_grid.remove (element);
666         }
667         //Remove boooks currently on list view
668         ArrayList<Gtk.TreeIter?> listOfIterstoBeRemoved = new ArrayList<Gtk.TreeIter?> ();
669         Gtk.TreeModelForeachFunc print_row = (model, path, iter) => {
670             listOfIterstoBeRemoved.add (iter);
671             return false;
672         };
673         BookwormApp.AppWindow.library_table_liststore.foreach (print_row);
674         foreach (Gtk.TreeIter iterToBeRemoved in listOfIterstoBeRemoved) {
675             //remove item for list store - vala_36 compatibility wrapper
676             #if VALA_0_36
677                 BookwormApp.AppWindow.library_table_liststore.remove (ref iterToBeRemoved)
678             ;
679             #else
680                 BookwormApp.AppWindow.library_table_liststore.remove (iterToBeRemoved);
681             #endif
682         }
683         //Clear the paths of the loaded books
684         BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr.erase (0, -1);
685         //Update the library view
686         BookwormApp.Library.updateLibraryViewFromDB ();
687     }
688     //set the status of the library buttons based on the paginate query results
689     //false : will prevent another paginate call
690     BookwormApp.AppWindow.handleLibraryPageButtons ("", false);
691 }
692 }
```

```
=====
database.vala ? Database Layer
=====

1  /* Copyright 2017 Siddhartha Das (bablu.boy@gmail.com)
2  *
3  * This file is part of Bookworm and manages all the Database interactions
4  *
5  * Bookworm is free software: you can redistribute it
6  * and/or modify it under the terms of the GNU General Public License as
7  * published by the Free Software Foundation, either version 3 of the
8  * License, or (at your option) any later version.
9  *
10 * Bookworm is distributed in the hope that it will be
11 * useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
12 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
13 * Public License for more details.
14 *
15 * You should have received a copy of the GNU General Public License along
16 * with Bookworm. If not, see http://www.gnu.org/licenses/.
17 */
18
19 using Sqlite;
20 using Gee;
21
22 public class BookwormApp.DB {
23     public const string BOOKWORM_TABLE_BASE_NAME = "BOOK_LIBRARY_TABLE";
24     public const string BOOKWORM_TABLE_VERSION = "6"; //Only integers allowed
25     public const string BOOKWORM_TABLE_NAME = BOOKWORM_TABLE_BASE_NAME + BOOKWORM_TABLE_VERSIO
N;
26     public const string BOOKMETADATA_TABLE_BASE_NAME = "BOOK_METADATA_TABLE";
27     public const string BOOKMETADATA_TABLE_VERSION = "1"; //Only integers allowed
28     public const string BOOKMETADATA_TABLE_NAME = BOOKMETADATA_TABLE_BASE_NAME + BOOKMETADATA_
TABLE_VERSION;
29     public const string VERSION_TABLE_BASE_NAME = "VERSION_TABLE";
30     public const string VERSION_TABLE_VERSION = "1"; //Only integers allowed
31     private static Sqlite.Database bookwormDB;
32     private static string errmsg;
33     private static string queryString;
34     private static int executionStatus;
35
36     public static bool initializeBookWormDB (string bookworm_config_path) {
37         info ("[START] [FUNCTION:initializeBookWormDB] bookworm_config_path=" + bookworm_config_
path);
38         Statement stmt;
39         debug ("Checking BookWorm DB or creating it if the DB does not exist..."); 
40         int dbOpenStatus = Database.open_v2 (
41             bookworm_config_path + "/bookworm.db", out bookwormDB, Sqlite.OPEN_READWRITE | Sql
ite.OPEN_CREATE);
42         if (dbOpenStatus != Sqlite.OK) {
43             warning ("Error in opening database[" + bookworm_config_path + "/bookworm.db" + "]"
: %d: %s\n", bookwormDB.errcode (), bookwormDB errmsg ());
44             return false;
45         } else {
46             debug ("Successfully checked/created DB for Bookworm.....");
47         }
48
49         debug ("Creating latest version for Library table if it does not exists");
50         queryString = "CREATE TABLE IF NOT EXISTS " + BOOKWORM_TABLE_NAME + " (" +
51             "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
52             "BOOK_LOCATION TEXT NOT NULL DEFAULT '', " +
53             "BOOK_TITLE TEXT NOT NULL DEFAULT '', " +
54             "BOOK_AUTHOR TEXT NOT NULL DEFAULT '', " +
55             "BOOK_COVER_IMAGE_LOCATION TEXT NOT NULL DEFAULT '', " +
56             "IS_BOOK_COVER_IMAGE_PRESENT TEXT NOT NULL DEFAULT '', " +
57             "BOOK_PUBLISH_DATE TEXT NOT NULL DEFAULT '', " +
58             "BOOK_TOTAL_NUMBER_OF_PAGES TEXT NOT NULL DEFAULT '', " +
59             "BOOK_LAST_READ_PAGE_NUMBER TEXT NOT NULL DEFAULT '', " +
60             "BOOK_TOTAL_PAGES TEXT NOT NULL DEFAULT ''" + //Added in table v6
61             ", TAGS TEXT NOT NULL DEFAULT ''" + //Added in table v3
62             ", ANNOTATION_TAGS TEXT NOT NULL DEFAULT ''" + //Added in table v7
63             ", RATINGS TEXT NOT NULL DEFAULT ''" + //Added in table v3
64             ", CONTENT_EXTRACTION_LOCATION TEXT NOT NULL DEFAULT ''" + //Added in table v4
65             ", creation_date INTEGER," +
```

```

66         "modification_date INTEGER)";
67     executionStatus = bookwormDB.exec (queryString, null, out errmsg);
68     if (executionStatus != Sqlite.OK) {
69         report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errms
g ());
70         return false;
71     } else {
72         debug ("Successfully checked/created table:" + BOOKWORM_TABLE_NAME);
73     }
74
75     debug ("Creating latest version for Book Metadata table if it does not exists");
76     queryString = "CREATE TABLE IF NOT EXISTS " + BOOKMETADATA_TABLE_NAME + " (" +
77         "id INTEGER PRIMARY KEY, " +
78         "BOOK_TOC_DATA TEXT NOT NULL DEFAULT '', " +
79         "BOOKMARKS TEXT NOT NULL DEFAULT '', " +
80         "CONTENT_DATA_LIST TEXT NOT NULL DEFAULT '', " +
81         "BOOK_LAST_SCROLL_POSITION TEXT NOT NULL DEFAULT '', " +
82         "BOOK_ANNOTATIONS TEXT NOT NULL DEFAULT '', " +
83         "creation_date INTEGER," +
84         "modification_date INTEGER)";
85     executionStatus = bookwormDB.exec (queryString, null, out errmsg);
86     if (executionStatus != Sqlite.OK) {
87         report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errms
g ());
88         return false;
89     } else {
90         debug ("Successfully checked/created table:" + BOOKMETADATA_TABLE_NAME);
91     }
92
93     //Check details of tables in DB
94     ArrayList<string> listOfTables = new ArrayList<string> ();
95     queryString = "SELECT NAME FROM SQLITE_MASTER WHERE TYPE='table' ORDER BY NAME";
96     executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
97     if (executionStatus != Sqlite.OK) {
98         report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errms
g ());
99     }
100    while (stmt.step () == ROW) {
101        listOfTables.add (stmt.column_text (0).strip ());
102    }
103    stmt.reset ();
104
105    //Remove the current tables (latest versions) from the list
106    listOfTables.remove (BOOKWORM_TABLE_NAME);
107    listOfTables.remove (BOOKMETADATA_TABLE_NAME);
108
109    //Loop over any remainng old versions of tables and delete
110    //them after ensuring data is migrated to the latest versions of the tables
111    foreach (string old_table_name in listOfTables) {
112        //BOOK_LIBRARY_TABLE5 : Migrate data and drop table
113        if (old_table_name == "BOOK_LIBRARY_TABLE5") {
114            //copy data to new library table
115            queryString = "INSERT INTO " + BOOKWORM_TABLE_NAME + " (" +
116                "id, BOOK_LOCATION, BOOK_TITLE, BOOK_AUTHOR, BOOK_COVER_IMAGE_LOCATION, " +
117                "IS_BOOK_COVER_IMAGE_PRESENT, BOOK_PUBLISH_DATE, BOOK_TOTAL_NUMBER_OF_PAGE
S, " +
118                "BOOK_LAST_READ_PAGE_NUMBER, TAGS, RATINGS, CONTENT_EXTRACTION_LOCATION, " +
119                "creation_date, modification_date) SELECT id, BOOK_LOCATION, BOOK_TITLE, " +
120                "BOOK_AUTHOR, BOOK_COVER_IMAGE_LOCATION, IS_BOOK_COVER_IMAGE_PRESENT, " +
121                "BOOK_PUBLISH_DATE, BOOK_TOTAL_NUMBER_OF_PAGES, BOOK_LAST_READ_PAGE_NUMBER
, " +
122                "TAGS, RATINGS, CONTENT_EXTRACTION_LOCATION, creation_date, " +
123                "modification_date FROM BOOK_LIBRARY_TABLE5";
124            executionStatus = bookwormDB.exec (queryString, null, out errmsg);
125            if (executionStatus != Sqlite.OK) {
126                report_query_execution_error (queryString, bookwormDB.errcode (), bookworm
DB.errmsg ());
127            } else {
128                debug ("Successfully migrated " + bookwormDB.changes ().to_string () + " r
ows from BOOK_LIBRARY_TABLE5 into " + BOOKWORM_TABLE_NAME);
129                //copy data to new meta data table
130                queryString = "INSERT INTO " + BOOKMETADATA_TABLE_NAME + " (" +

```

```

131             "id, BOOK_TOC_DATA, BOOKMARKS, CONTENT_DATA_LIST, BOOK_LAST_SCROLL_POS
132             ITION, " +
133             "creation_date, modification_date) SELECT id, BOOK_TOC_DATA, BOOKMARKS
134             , " +
135             "CONTENT_DATA_LIST, BOOK_LAST_SCROLL_POSITION, creation_date, " +
136             "modification_date FROM BOOK_LIBRARY_TABLE5";
137             executionStatus = bookwormDB.exec (queryString, null, out errmsg);
138             if (executionStatus != Sqlite.OK) {
139                 report_query_execution_error (queryString, bookwormDB.errcode (), book
140                 wormDBerrmsg ());
141             } else {
142                 debug ("Successfully migrated " + bookwormDB.changes ().to_string () +
143                     " rows from BOOK_LIBRARY_TABLE5 into" + BOOKMETADATA_TABLE_NAME);
144                 //drop the old table
145                 queryString = "DROP TABLE IF EXISTS BOOK_LIBRARY_TABLE5";
146                 executionStatus = bookwormDB.exec (queryString, null, out errmsg);
147                 if (executionStatus != Sqlite.OK) {
148                     report_query_execution_error (queryString, bookwormDB.errcode (), book
149                     wormDBerrmsg ());
150                 } else {
151                     debug ("Successfully dropped old table LIBRARY_TABLE5");
152                 }
153             }
154         }
155         //VERSION_TABLE : Drop table
156         if (old_table_name == "VERSION_TABLE") {
157             //drop the old table
158             queryString = "DROP TABLE IF EXISTS VERSION_TABLE";
159             executionStatus = bookwormDB.exec (queryString, null, out errmsg);
160             if (executionStatus != Sqlite.OK) {
161                 report_query_execution_error (queryString, bookwormDB.errcode (), bookworm
162                 DB errmsg ());
163             } else {
164                 debug ("Successfully dropped old table VERSION_TABLE");
165             }
166         }
167         //All DB loading operations completed
168         info ("[END] [FUNCTION:initializeBookWormDB]");
169         return true;
170     }
171     public static ArrayList<BookwormApp.Book> getBooksFromDB (String criteria, String mode) {
172         info ("[START] [FUNCTION:getBooksFromDB] Quering with mode[" + mode + "] and criteria[
173             " + criteria + "]");
174         ArrayList<BookwormApp.Book> listOfBooks = new ArrayList<BookwormApp.Book> ();
175         Statement stmt;
176         String last_modification_date = "-1";
177         queryString = "SELECT id, BOOK_LOCATION, BOOK_TITLE, BOOK_AUTHOR, BOOK_COVER_IMAGE_LOC
178             ATION, " +
179             "IS_BOOK_COVER_IMAGE_PRESENT, BOOK_LAST_READ_PAGE_NUMBER, BOOK_PUBLISH_DATE, TAGS,
180             " +
181             "ANNOTATION_TAGS, RATINGS, CONTENT_EXTRACTION_LOCATION, BOOK_TOTAL_PAGES, creation
182             _date, " +
183             "modification_date FROM " + BOOKWORM_TABLE_NAME;
184         if (criteria == "" && mode == "PAGINATED_SEARCH") {
185             //initial query on app load without pagination criteria
186             queryString = queryString + " ORDER BY modification_date DESC LIMIT " + BookwormAp
187             p.Bookworm.no_of_books_per_page;
188             debug ("Paginated Query with last_modification_date[" + criteria + "]: " + queryStr
189             ing);
190         } else if (mode == "LIBRARY_SEARCH") {
191             //query db for matching search criteria on all book meta data
192             queryString = queryString + " WHERE " +
193                 " BOOK_TITLE LIKE '%" + criteria + "%' OR " +
194                 " BOOK_LOCATION LIKE '%" + criteria + "%' OR " +
195                 " BOOK_AUTHOR LIKE '%" + criteria + "%' OR " +
196                 " TAGS LIKE '%" + criteria + "%' OR " +
197                 " ANNOTATION_TAGS LIKE '%" + criteria + "%'";
198             debug ("Library Search Query with criteria[" + criteria + "]: " + queryString);
199         } else if (mode == "PAGINATED_SEARCH") {
200             //query for pagination criteria
201             queryString = queryString + " where modification_date < CAST ('" + criteria + "' A
202             S INT) " +

```

```

194             "ORDER BY modification_date DESC LIMIT " + BookwormApp.Bookworm.no_of_books_per_page;
195         debug ("Paginated Query with last_modification_date[" + criteria + "]:" + queryString);
196     }
197     executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
198     if (executionStatus != Sqlite.OK) {
199         report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errmsg);
200     } else {
201         while (stmt.step () == ROW) {
202             BookwormApp.Book aBook = new BookwormApp.Book ();
203             aBook.setBookId (stmt.column_int (0));
204             aBook.setBookLocation (stmt.column_text (1));
205             aBook.setBookTitle (stmt.column_text (2));
206             aBook.setBookAuthor (stmt.column_text (3));
207             aBook.setBookCoverLocation (stmt.column_text (4));
208             aBook.setIsBookCoverImagePresent ( (stmt.column_text (5) == "true") ? true:false);
209         }
210         aBook.setBookPageNumber (int.parse (stmt.column_text (6)));
211         aBook.setBookPublishDate (stmt.column_text (7));
212         aBook.setBookTags (stmt.column_text (8));
213         aBook.setAnnotationTags (stmt.column_text (9));
214         aBook.setBookRating (int.parse (stmt.column_text (10)));
215         aBook.setBookExtractionLocation (stmt.column_text (11));
216         aBook.setBookTotalPages (int.parse (stmt.column_text (12)));
217         aBook.setBookCreationDate (stmt.column_text (13));
218         aBook.setBookLastModificationDate (stmt.column_text (14));
219         debug ("Book details fetched from DB: id=" + stmt.column_int (0).to_string ());
220         +
221         ",BOOK_LOCATION=" + stmt.column_text (1) +
222         ",BOOK_TITLE=" + stmt.column_text (2) +
223         ",BOOK_AUTHOR=" + stmt.column_text (3) +
224         ",BOOK_COVER_IMAGE_LOCATION=" + stmt.column_text (4) +
225         ",IS_BOOK_COVER_IMAGE_PRESENT=" + stmt.column_text (5) +
226         ",BOOK_LAST_READ_PAGE_NUMBER=" + stmt.column_text (6) +
227         ",BOOK_PUBLISH_DATE=" + stmt.column_text (7) +
228         ",TAGS=" + stmt.column_text (8) +
229         ",ANNOTATION_TAGS=" + stmt.column_text (9) +
230         ",RATINGS=" + stmt.column_text (10) +
231         ",CONTENT_EXTRACTION_LOCATION=" + stmt.column_text (11) +
232         ",BOOK_TOTAL_PAGES=" + stmt.column_text (12) +
233         ",creation_date=" + stmt.column_text (13) +
234         ",modification_date=" + stmt.column_text (14));
235         //add book details to list
236         listOfBooks.add (aBook);
237         //build the string of book paths in the library
238         BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr.append (aBook.getBookLocation ());
239     }
240     //capture the last_modification_date of the book
241     last_modification_date = aBook.getBookLastModificationDate ();
242 }
243 if (mode == "PAGINATED_SEARCH") {
244     //Only capture the last modification date if the results are equal to the page
245     size
246     if (listOfBooks.size == int.parse (BookwormApp.Bookworm.no_of_books_per_page))
247     {
248         //set the last book's modification date for pagination
249         BookwormApp.Bookworm.paginationlist.add (last_modification_date);
250     } else {
251         BookwormApp.Bookworm.paginationlist.add ("-1");
252     }
253 }
254
255 public static BookwormApp.Book getBookFromDB (string book_location) {
256     info ("[START] [FUNCTION:getBookFromDB] Attempting to search DB for book.location=" +
book_location);
257     Statement stmt;
258     BookwormApp.Book aBook = new BookwormApp.Book ();

```

```

259         queryString = "SELECT id, BOOK_LOCATION, BOOK_TITLE, BOOK_AUTHOR, BOOK_COVER_IMAGE_LOC
260             ATION, " +
261             "IS_BOOK_COVER_IMAGE_PRESENT, BOOK_LAST_READ_PAGE_NUMBER, BOOK_PUBLISH_DATE, TAGS,
262             " +
263             "ANNOTATION_TAGS, RATINGS, CONTENT_EXTRACTION_LOCATION, BOOK_TOTAL_PAGES, creation
264             _date, " +
265             "modification_date FROM " + BOOKWORM_TABLE_NAME;
266             //query db for matching book location
267             queryString = queryString + " WHERE " + "BOOK_LOCATION LIKE '" + book_location + "'";
268             debug ("Library Search Query with criteria[" + book_location + "]:" + queryString);
269             executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
270             if (executionStatus != Sqlite.OK) {
271                 report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errms
272                 g ());
273             } else {
274                 while (stmt.step () == ROW) {
275                     aBook.setBookId (stmt.column_int (0));
276                     aBook.setBookLocation (stmt.column_text (1));
277                     aBook.setBookTitle (stmt.column_text (2));
278                     aBook.setBookAuthor (stmt.column_text (3));
279                     aBook.setBookCoverLocation (stmt.column_text (4));
280                     aBook.setIsBookCoverImagePresent ( (stmt.column_text (5) == "true") ? true : f
281                     alse);
282                     aBook.setBookPageNumber (int.parse (stmt.column_text (6)));
283                     aBook.setBookPublishDate (stmt.column_text (7));
284                     aBook.setBookTags (stmt.column_text (8));
285                     aBook.setAnnotationTags (stmt.column_text (9));
286                     aBook.setBookRating (int.parse (stmt.column_text (10)));
287                     aBook.setBookExtractionLocation (stmt.column_text (11));
288                     aBook.setBookTotalPages (int.parse (stmt.column_text (12)));
289                     aBook.setBookCreationDate (stmt.column_text (13));
290                     aBook.setBookLastModificationDate (stmt.column_text (14));
291                     debug ("Book details fetched from DB: id=" + stmt.column_int (0).to_string ()
292                     +
293                         ",BOOK_LOCATION=" + stmt.column_text (1) +
294                         ",BOOK_TITLE=" + stmt.column_text (2) +
295                         ",BOOK_AUTHOR=" + stmt.column_text (3) +
296                         ",BOOK_COVER_IMAGE_LOCATION=" + stmt.column_text (4) +
297                         ",IS_BOOK_COVER_IMAGE_PRESENT=" + stmt.column_text (5) +
298                         ",BOOK_LAST_READ_PAGE_NUMBER=" + stmt.column_text (6) +
299                         ",BOOK_PUBLISH_DATE=" + stmt.column_text (7) +
300                         ",TAGS=" + stmt.column_text (8) +
301                         ",ANNOTATION_TAGS=" + stmt.column_text (9) +
302                         ",RATINGS=" + stmt.column_text (10) +
303                         ",CONTENT_EXTRACTION_LOCATION=" + stmt.column_text (11) +
304                         ",BOOK_TOTAL_PAGES=" + stmt.column_text (12) +
305                         ",creation_date=" + stmt.column_text (13) +
306                         ",modification_date=" + stmt.column_text (14));
307                 }
308             }
309             stmt.reset ();
310             info ("[END] [FUNCTION:getBookFromDB] Book fetched [" + aBook.getBookLocation () + "]"
311             );
312             return aBook;
313         }
314     }
315
316     public static BookwormApp.Book getBookMetaDataTableFromDB (owned BookwormApp.Book aBook) {
317         info ("[START] [FUNCTION:getBookMetaDataTableFromDB] book.location=" + aBook.getBookLocatio
318         n ());
319         Statement stmt;
320         queryString = "SELECT BOOK_TOC_DATA, BOOKMARKS, CONTENT_DATA_LIST, BOOK_LAST_SCROLL_PO
321             SITION, " +
322             "BOOK_ANNOTATIONS FROM " + BOOKMETADATA_TABLE_NAME + " WHERE id = ?";
323         executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
324         if (executionStatus != Sqlite.OK) {
325             report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errms
326             g ());
327         }
328         stmt.bind_int (1, aBook.getBookId ());
329         while (stmt.step () == ROW) {
330             aBook = BookwormApp.Utils.convertStringToTOC (aBook, stmt.column_text (0));
331             aBook.setBookmark (-10, stmt.column_text (1)); // -10 is a flag to set the bookmark
332             string into the object
333             aBook = BookwormApp.Utils.convertStringToContentList (aBook, stmt.column_text (2))
334         ;

```

```

322         aBook.setBookScrollPos (int.parse (stmt.column_text (3)));
323         aBook.setAnnotationList (BookwormApp.Utils.convertStringToTreeMap (stmt.column_text
324             (4)));
324         debug ("Book MetaData details fetched from DB: id=" + aBook.getBookId ().to_string
324             ());
325         ", BOOK_TOC_DATA=" + stmt.column_text (0) +
326         ", BOOKMARKS=" + stmt.column_text (1) +
327         ", CONTENT_DATA_LIST=" + stmt.column_text (2) +
328         ", BOOK_LAST_SCROLL_POSITION=" + stmt.column_text (3) +
329         ", BOOK_ANNOTATIONS=" + stmt.column_text (4));
330     }
331     stmt.reset ();
332     info ("[END] [FUNCTION:getBookMetaDataFromDB] book.location=" + aBook.getBookLocation
332         ());
333     return aBook;
334 }
335
336 public static int addBookToDataBase (BookwormApp.Book aBook) {
337     info ("[START] [FUNCTION:addBookToDataBase] book.location=" + aBook.getBookLocation ())
337         ;
338     Sqlite.Statement stmt;
339     int insertedBookID = 0;
340     queryString = "INSERT INTO " + BOOKWORM_TABLE_NAME + " (BOOK_LOCATION, BOOK_TITLE, BOO
K_AUTHOR, " +
341         "BOOK_COVER_IMAGE_LOCATION, IS_BOOK_COVER_IMAGE_PRESENT, CONTENT_EXTRACTION_LOCATI
ON, " +
342         "creation_date, modification_date) " + "VALUES (?, ?, ?, ?, ?, ?, ?, CAST (strftime (
'%"s', 'now') " +
343         "AS INT), CAST (strftime ('%"s', 'now') AS INT))";
344     executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
345     if (executionStatus != Sqlite.OK) {
346         report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errms
g ());
347         return -1;
348     }
349     stmt.bind_text (1, aBook.getBookLocation ());
350     stmt.bind_text (2, aBook.getBookTitle ());
351     stmt.bind_text (3, aBook.getBookAuthor ());
352     stmt.bind_text (4, aBook.getBookCoverLocation ());
353     stmt.bind_text (5, aBook.getIsBookCoverImagePresent ().to_string ());
354     stmt.bind_text (6, aBook.getBookExtractionLocation ());
355
356     stmt.step ();
357     stmt.reset ();
358     //fetch the id of the book just inserted into the DB
359     queryString = "SELECT id FROM " + BOOKWORM_TABLE_NAME + " WHERE BOOK_LOCATION = ?";
360     executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
361     if (executionStatus != Sqlite.OK) {
362         report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errms
g ());
363     }
364     stmt.bind_text (1, aBook.getBookLocation ());
365     while (stmt.step () == ROW) {
366         insertedBookID = stmt.column_int (0);
367     }
368     stmt.reset ();
369     info ("[END] [FUNCTION:addBookToDataBase] book.location=" + aBook.getBookLocation ());
370     return insertedBookID;
371 }
372
373 public static bool removeBookFromDB (BookwormApp.Book aBook) {
374     info ("[START] [FUNCTION:removeBookFromDB] book.location=" + aBook.getBookLocation ())
374         ;
375     Sqlite.Statement stmt;
376     //delete book from library table
377     queryString = "DELETE FROM " + BOOKWORM_TABLE_NAME + " WHERE id = ?";
378     executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
379     if (executionStatus != Sqlite.OK) {
380         report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errms
g ());
381         return false;
382     } else {
383         stmt.bind_int (1, aBook.getBookId ());
384         stmt.step ();
385         stmt.reset ();

```

```

386         debug ("Removed this book from library table:" + aBook.getBookTitle () + "[" + aBo
ok.getBookId ().to_string () + "]");
387         //delete book meta data from meta data table
388         queryString = "DELETE FROM " + BOOKMETADATA_TABLE_NAME + " WHERE id = ?";
389         executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt
);
390         if (executionStatus != Sqlite.OK) {
391             report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.e
rrmsg ());
392             return false;
393         } else {
394             stmt.bind_int (1, aBook.getBookId ());
395             stmt.step ();
396             stmt.reset ();
397         }
398     }
399     info ("[END] [FUNCTION:removeBookFromDB] book.location=" + aBook.getBookLocation ());
400     return true;
401 }
402
403 public static boolean updateBookToDataBase (BookwormApp.Book aBook) {
404     info ("[START] [FUNCTION:updateBookToDataBase] Updating book to DB for the following d
etails:" + aBook.to_string ());
405     Sqlite.Statement stmt;
406     queryString = "UPDATE " + BOOKWORM_TABLE_NAME + " SET BOOK_LAST_READ_PAGE_NUMBER = ?, "
+
407         "BOOK_TITLE = ?, BOOK_AUTHOR = ?, BOOK_COVER_IMAGE_LOCATION = ?, " +
408         "IS_BOOK_COVER_IMAGE_PRESENT = ?, TAGS = ?, ANNOTATION_TAGS = ?, RATINGS = ?, " +
409         "CONTENT_EXTRACTION_LOCATION = ?, BOOK_TOTAL_PAGES = ?, " +
410         "modification_date = CAST (? AS INT) WHERE ID = ? ";
411     executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
412     if (executionStatus != Sqlite.OK) {
413         report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errms
g ());
414         return false;
415     }
416     stmt.bind_text (1, aBook.getBookPageNumber ().to_string ());
417     stmt.bind_text (2, aBook.getBookTitle ());
418     stmt.bind_text (3, aBook.getBookAuthor ());
419     stmt.bind_text (4, aBook.getBookCoverLocation ());
420     stmt.bind_text (5, aBook.getIsBookCoverImagePresent ().to_string ());
421     stmt.bind_text (6, aBook.getBookTags ());
422     stmt.bind_text (7, aBook.getAnnotationTags ());
423     stmt.bind_text (8, aBook.getBookRating ().to_string ());
424     stmt.bind_text (9, aBook.getBookExtractionLocation ());
425     stmt.bind_text (10, aBook.getBookTotalPages ().to_string ());
426     stmt.bind_text (11, aBook.getBookLastModificationDate ());
427     stmt.bind_int (12, aBook.getBookId ());
428     stmt.step ();
429     stmt.reset ();
430     debug ("Updated library details to " + BOOKWORM_TABLE_NAME + " for book:" + aBook.getB
ookTitle () + "[" + aBook.getBookId ().to_string () + "]");
431     //Attempt to insert book meta data
432     queryString = "INSERT OR IGNORE INTO " + BOOKMETADATA_TABLE_NAME + " (BOOK_TOC_DATA, B
OOKMARKS, " +
433         "CONTENT_DATA_LIST, BOOK_LAST_SCROLL_POSITION, BOOK_ANNOTATIONS, modification_date
, id) " +
434         "VALUES (?, ?, ?, ?, ?, CAST (strftime ('%s', 'now') AS INT), ?);";
435     executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
436     if (executionStatus != Sqlite.OK) {
437         report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errms
g ());
438         return false;
439     }
440     stmt.bind_text (1, BookwormApp.Utils.convertTOCToString (aBook));
441     stmt.bind_text (2, aBook.getBookmark ());
442     stmt.bind_text (3, BookwormApp.Utils.convertContentListToString (aBook));
443     stmt.bind_text (4, aBook.getBookScrollPos ().to_string ());
444     stmt.bind_text (5, BookwormApp.Utils.convertTreeMapToString (aBook.getAnnotationList (
)));
445     stmt.bind_int (6, aBook.getBookId ());
446     stmt.step ();
447     stmt.reset ();
448     if (bookwormDB.changes () == 0) {
449         //Book already present, update the meta data

```

```

450         queryString = "UPDATE " + BOOKMETADATA_TABLE_NAME + " SET BOOK_TOC_DATA = ?, BOOKM
ARKS = ?, " +
451             "CONTENT_DATA_LIST = ?, BOOK_LAST_SCROLL_POSITION = ?, BOOK_ANNOTATIONS = ?, "
+
452             "modification_date = CAST (strftime ('%s', 'now') AS INT) WHERE id = ? ";
453             executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt
);
454             if (executionStatus != Sqlite.OK) {
455                 report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.e
rrmsg ());
456                 return false;
457             }
458             stmt.bind_text (1, BookwormApp.Utils.convertTOCToString (aBook));
459             stmt.bind_text (2, aBook.getBookmark ());
460             stmt.bind_text (3, BookwormApp.Utils.convertContentListToString (aBook));
461             stmt.bind_text (4, aBook.getBookScrollPos ().to_string ());
462             stmt.bind_text (5, BookwormApp.Utils.convertTreeMapToString (aBook.getAnnotationLi
st ()));
463             stmt.bind_int (6, aBook.getBookId ());
464             stmt.step ();
465             stmt.reset ();
466             debug ("Updated book meta data details to " + BOOKMETADATA_TABLE_NAME + " for book
: " + aBook.getBookTitle () + "[" + aBook.getBookId ().to_string () + "]");
467             } else {
468                 debug ("Inserted book meta data details to " + BOOKMETADATA_TABLE_NAME + " for boo
k: " + aBook.getBookTitle () + "[" + aBook.getBookId ().to_string () + "]");
469             }
470             info ("[END] [FUNCTION:updateBookToDataBase] book.location=" + aBook.getBookLocation (
));
471             return true;
472     }
473
474     public static ArrayList<string> getBookIDListFromDB () {
475         info ("[START] [FUNCTION:getBookIDListFromDB]");
476         ArrayList<string> bookIDList = new ArrayList<string> ();
477         Statement stmt;
478         queryString = "SELECT id,BOOK_LOCATION FROM " + BOOKWORM_TABLE_NAME + " ORDER BY id DE
SC";
479         executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
480         if (executionStatus != Sqlite.OK) {
481             report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errms
g ());
482         }
483         while (stmt.step () == ROW) {
484             bookIDList.add (stmt.column_int (0).to_string () + ":" + stmt.column_text (1));
485         }
486         stmt.reset ();
487         info ("[END] [FUNCTION:getBookIDListFromDB] bookIDList.size" + bookIDList.size.to_stri
ng ());
488         return bookIDList;
489     }
490
491     private static void report_query_execution_error (String query, int errcode, String errmsg
) {
492         debug ("Error on executing Query: %s\n", query);
493         warning ("Error (%d) details: %s\n", errcode, errmsg);
494     }
495 }
```

```
=====
ePubReader.vala ? EPUB Reader
=====

1  /* Copyright 2017 Siddhartha Das (bablu.boy@gmail.com)
2  *
3  * This file is part of Bookworm and is used for parsing EPUB file formats
4  *
5  * Bookworm is free software: you can redistribute it
6  * and/or modify it under the terms of the GNU General Public License as
7  * published by the Free Software Foundation, either version 3 of the
8  * License, or (at your option) any later version.
9  *
10 * Bookworm is distributed in the hope that it will be
11 * useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
12 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
13 * Public License for more details.
14 *
15 * You should have received a copy of the GNU General Public License along
16 * with Bookworm. If not, see http://www.gnu.org/licenses/.
17 */
18
19 using Gee;
20 public class BookwormApp.ePubReader {
21     public static string NCXRefInSpineData = "";
22     public static BookwormApp.Book parseEPubBook (owned BookwormApp.Book aBook) {
23         info ("[START] [FUNCTION:parseEPubBook] book.location=" + aBook.getBookLocation ());
24         //Only parse the eBook if it has not been parsed already
25         if (!aBook.getIsBookParsed ()) {
26             debug ("Starting to parse EPub Book located at:" + aBook.getBookLocation ());
27             //Extract the content of the EPub
28             string extractionLocation = extractEBook (aBook.getBookLocation ());
29             if ("false" == extractionLocation) { //handle error condition
30                 aBook.setIsBookParsed (false);
31                 aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_EXTRACTION_ISSUE);
32                 return aBook;
33             } else {
34                 aBook.setBookExtractionLocation (extractionLocation);
35             }
36             //Check if the EPUB mime type is correct
37             bool isEPubFormat = isEPubFormat (extractionLocation);
38             if (!isEPubFormat) { //handle error condition
39                 aBook.setIsBookParsed (false);
40                 aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_MIMETYPE_ISSUE);
41                 return aBook;
42             }
43             //Determine the location of OPF File
44             string locationOfOPFFile = getOPFFfileLocation (extractionLocation);
45             if ("false" == locationOfOPFFile) { //handle error condition
46                 aBook.setIsBookParsed (false);
47                 aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_CONTENT_ISSUE);
48                 return aBook;
49             }
50             string baseLocationOfContents = locationOfOPFFile.replace (File.new_for_path (locationOfOPFFile).get_basename (), "");
51             aBook.setBaseLocationOfContents (baseLocationOfContents);
52             //Populate content list for EPub Book
53             aBook = determineToC (aBook, locationOfOPFFile);
54             if (aBook.getBookContentList ().size < 1) {
55                 aBook.setIsBookParsed (false);
56                 aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_CONTENT_ISSUE);
57                 return aBook;
58             }
59             //Try to determine Book Cover Image if it is not already available
60             if (!aBook.getIsBookCoverImagePresent ()) {
61                 aBook = setCoverImage (aBook, locationOfOPFFile);
62             }
63             //Determine Book Meta Data like Title, Author, etc
64             aBook = setBookMetaData (aBook, locationOfOPFFile);
65             aBook.setIsBookParsed (true);
66         }
67         info ("[END] [FUNCTION:parseEPubBook]");
68         return aBook;
69     }
}
```

```

70
71     public static string extractEBook (string eBookLocation) {
72         info ("[START] [FUNCTION:extractEBook] eBookLocation=" + eBookLocation);
73         string extractionLocation = "false";
74         debug ("Initiated process for content extraction of ePub Book located at:" + eBookLoca
tion);
75         //create a location for extraction of eBook based on local storage preference
76         if (BookwormApp.Bookworm.settings == null) {
77             BookwormApp.Bookworm.settings = BookwormApp.Settings.get_instance ();
78         }
79         if (BookwormApp.Bookworm.settings.is_local_storage_enabled) {
80             extractionLocation = BookwormApp.Bookworm.bookworm_config_path + "/books/" + File.
new_for_path (eBookLocation).get_basename ();
81         } else {
82             extractionLocation = BookwormApp.Constants.EBOOK_EXTRACTION_LOCATION + File.new_fo
r_path (eBookLocation).get_basename ();
83         }
84         //check and create directory for extracting contents of ebook
85         BookwormApp.Utils.fileOperations ("CREATEDIR", extractionLocation, "", "");
86         //unzip eBook contents into extraction location
87         string status = BookwormApp.Utils.execute_sync_command ("unzip -o \\" + eBookLocation
+ "\\\" -d \\\"" + extractionLocation + "\\\"");
88         if ("false" == status) {
89             extractionLocation = "false";
90         }
91         info ("[END] [FUNCTION:extractEBook] extractionLocation=" + extractionLocation);
92         return extractionLocation;
93     }
94
95     public static bool isEPubFormat (string extractionLocation) {
96         info ("[START] [FUNCTION:isEPubFormat] extractionLocation=" + extractionLocation);
97         bool ePubFormat = false;
98         debug ("Checking if mime type is valid ePub for contents at:" + extractionLocation);
99         string ePubMimeContents = BookwormApp.Utils.fileOperations (
100             "READ", extractionLocation, BookwormApp.Constants.EPUB_MIME_SPECIFICATION_FILENAME
, "");
101        if ("false" == ePubMimeContents) {
102            //Mime Content File was not found at expected location
103            warning ("Mime Content file could not be located at expected location:" +
extractionLocation + "/" + BookwormApp.Constants.EPUB_MIME_SPECIFICATION_FILEN
AME);
104            return false;
105        }
106        debug ("Mime Contents found in file :" + extractionLocation + "/" +
BookwormApp.Constants.EPUB_MIME_SPECIFICATION_FILENAME + " is:" + ePubMimeContents
);
107        if (ePubMimeContents.strip () != BookwormApp.Constants.EPUB_MIME_SPECIFICATION_CONTENT
) {
108            debug ("Mime Contents in file :" + extractionLocation + "/" +
BookwormApp.Constants.EPUB_MIME_SPECIFICATION_FILENAME + " is not :" +
BookwormApp.Constants.EPUB_MIME_SPECIFICATION_CONTENT + ". No further parsing
will be done.");
109            return false;
110        } else {
111            //mime content is as expected
112            ePubFormat = true;
113        }
114        info ("[END] [FUNCTION:isEPubFormat] ePubFormat=" + ePubFormat.to_string ());
115        return ePubFormat;
116    }
117
118    public static string getOPFFileLocation (string extractionLocation) {
119        info ("[START] [FUNCTION:getOPFFileLocation] extractionLocation=" + extractionLocation
);
120        string locationOfOPFFile = "false";
121        //Form the path to the META-INF/container.xml file
122        string pathToXMLfile = extractionLocation + "/" + BookwormApp.Constants.EPUB_META_INF_
FILENAME;
123        //Parse META-INF/container.xml file to locate the path to the OPF file
124        ArrayList<XMLData> inputDataTable = new ArrayList<XMLData> ();
125        inputDataTable.add (new XMLData () {
126            containerTagName = "rootfiles",
127            inputTagName = "rootfile",
128            inputAttributeName = "full-path"
129        });
130
131
132
133

```

```

134     XmlParser thisParser = new XmlParser ();
135     ArrayList<XMLData> extractedDataList = new ArrayList<XMLData> ();
136     extractedDataList = thisParser.extractDataFromXML (pathToXMLFile, inputDataList);
137
138     foreach (XMLData aExtractedData in extractedDataList) {
139         foreach (string aAttributeValue in aExtractedData.extractedTagAttributes) {
140             string OPFFfilePath = aAttributeValue;
141             locationOfOPFFfile = extractionLocation + "/" + OPFFfilePath;
142         }
143     }
144     info ("[END] [FUNCTION:getOPFFfileLocation] locationOfOPFFfile=" + locationOfOPFFfile);
145     return locationOfOPFFfile;
146 }
147
148     public static BookwormApp.Book determineToC (owned BookwormApp.Book aBook, string location
149     OfOPFFfile) {
150         info ("[START] [FUNCTION:determineToC] book.location=" + aBook.getBookLocation () + ", "
151         locationOfOPFFfile=" + locationOfOPFFfile);
152         //Parse OPF xml file to read the MANIFEST data (id, href, media-type)
153         ArrayList<XMLData> inputDataList = new ArrayList<XMLData> ();
154         inputDataList.add (new XMLData () {
155             containerTagName = "manifest",
156             inputTagName = "item",
157             inputAttributeName = "id"
158         });
159         inputDataList.add (new XMLData () {
160             containerTagName = "manifest",
161             inputTagName = "item",
162             inputAttributeName = "href"
163         });
164         inputDataList.add (new XMLData () {
165             containerTagName = "manifest",
166             inputTagName = "item",
167             inputAttributeName = "media-type"
168         });
169         inputDataList.add (new XMLData () {
170             containerTagName = "spine",
171             inputTagName = "itemref",
172             inputAttributeName = "idref"
173         });
174         inputDataList.add (new XMLData () {
175             containerTagName = "",
176             inputTagName = "spine",
177             inputAttributeName = "toc"
178         });
179         XmlParser thisParser = new XmlParser ();
180         ArrayList<XMLData> opfItemsList = new ArrayList<XMLData> ();
181         opfItemsList = thisParser.extractDataFromXML (locationOfOPFFfile, inputDataList);
182
183         if (opfItemsList.size > 3 && opfItemsList.get (4).extractedTagAttributes.size>0) {
184             debug ("Successfully extracted SPINE data..");
185             //Get the reference of the NCX file in the SPINE data
186             string spineNCXReference = opfItemsList.get (4).extractedTagAttributes.get (0);
187             debug ("Successfully determined NCX File Reference as:" + spineNCXReference);
188             //Get the position of NCX Reference in MANIFEST data
189             if (opfItemsList.size>0 && opfItemsList.get (0).extractedTagAttributes.contains (s
190             pineNCXReference)) {
191                 debug ("Successfully extracted MANIFEST data");
192                 int spineNCXPosition = opfItemsList.get (0).extractedTagAttributes.index_of (s
193                 pineNCXReference);
194                 debug ("Successfully matched NCX File path information on MANIFEST data at pos
195                 ition:" + spineNCXPosition.to_string ());
196                 //Get the location of the NCX file from the MANIFEST href attribute
197                 string NCXfileRelativePath = opfItemsList.get (1).extractedTagAttributes.get (
198                 spineNCXPosition);
199                 debug ("Extracted relative NCX file path from MANIFEST data as:" + NCXfileRela

```

```

200         ArrayList<XMLData> inputDataListForToC = new ArrayList<XMLData> ();
201         inputDataListForToC.add (new XMLData () {
202             containerTagName = "navLabel",
203             inputTagName = "text",
204             inputAttributeName = ""
205         });
206         inputDataListForToC.add (new XMLData () {
207             containerTagName = "",
208             inputTagName = "content",
209             inputAttributeName = "src"
210         });
211         XmlParser ncxParser = new XmlParser ();
212         ArrayList<XMLData> ncxDataExtractedList = new ArrayList<XMLData> ();
213         ncxDataExtractedList = ncxParser.extractDataFromXML (ncxFilePath, inputDataListForToC);
214         if (ncxDataExtractedList.get (0).extractedTagValues.size > 0 &&
215             ncxDataExtractedList.get (1).extractedTagAttributes.size > 0 &&
216             ncxDataExtractedList.get (0).extractedTagValues.size == ncxDataExtractedList.get (1).extractedTagAttributes.size)
217         {
218             for (int count=0; count<ncxDataExtractedList.get (0).extractedTagValue
219             .size; count++) {
220                 HashMap<string,string> TOCMapItem = new HashMap<string,string> ();
221                 string tocLocation = ncxDataExtractedList.get (1).extractedTagAttr
222                 ibutes.get (count);
223                 //Handle the links with anchor elements
224                 string anchorValue = "";
225                 if (tocLocation.index_of ("#") != -1 ) {
226                     anchorValue = tocLocation.slice (tocLocation.index_of ("#"), t
227                     ocLocation.length);
228                     tocLocation = tocLocation.slice (0, tocLocation.index_of ("#")
229                 );
230                 }
231                 tocLocation = BookwormApp.Utils.getFullPathFromFilename (aBook.get
232                 BaseLocationOfContents (), tocLocation);
233                 TOCMapItem.set (tocLocation + anchorValue, ncxDataExtractedList.get
234                 (0).extractedTagValues.get (count));
235                 aBook.setTOC (TOCMapItem);
236                 debug ("Extracted ToC Chapter Name:" + ncxDataExtractedList.get (0
237                 ).extractedTagValues.get (count) + " at location:" + tocLocatio
238                 n + anchorValue);
239             }
240         }
241         // Create the content list - clear the content list of any previous items
242         aBook.clearBookContentList ();
243         //loop over all idref attributes in spine data
244         foreach (string spineIDREF in opfItemsList[3].extractedTagAttributes) {
245             //check if the SPINE IDREF exists in the MANIFEST Attributes
246             if (opfItemsList[0].extractedTagAttributes.contains (spineIDREF)) {
247                 int positionOfIDREF = opfItemsList[0].extractedTagAttributes.index_of (spineID
248                 REF);
249                 //extract the HREF from MANIFEST corresponding to the SPINE IDREF
250                 string locationOfContentData = opfItemsList[1].extractedTagAttributes.get (pos
251                 itionOfIDREF);
252                 aBook.setBookContentList (aBook.getBaseLocationOfContents () + locationOfConte
253                 ntData);
254                 debug ("Book content data :" + aBook.getBaseLocationOfContents () + locationOf
255                 ContentData);
256             }
257         }
258         info ("[END] [FUNCTION:determineToC]");
259         return aBook;
260     }
261
262     public static BookwormApp.Book setCoverImage (owned BookwormApp.Book aBook, string locatio
263     nOfOPFFile) {
264         info ("[START] [FUNCTION:setCoverImage] book.location=" + aBook.getBookLocation () + "
265         , locationOfOPFFile=" + locationOfOPFFile);
266         string bookCoverLocation = "";
267         //Parse OPF xml file to read the MANIFEST data

```

```

259     ArrayList<XMLData> inputDataList = new ArrayList<XMLData> ();
260     inputDataList.add (new XMLData () {
261         containerTagName = "manifest",
262         inputTagName = "item",
263         inputAttributeName = "id"
264     });
265     inputDataList.add (new XMLData () {
266         containerTagName = "manifest",
267         inputTagName = "item",
268         enforceAttributeData = true,
269         inputAttributeName = "media-type"
270     });
271     inputDataList.add (new XMLData () {
272         containerTagName = "manifest",
273         inputTagName = "item",
274         inputAttributeName = "href"
275     });
276     inputDataList.add (new XMLData () {
277         containerTagName = "manifest",
278         inputTagName = "item",
279         enforceAttributeData = true,
280         inputAttributeName = "properties"
281     });
282     XmlParser thisParser = new XmlParser ();
283     ArrayList<XMLData> opfItemsList = new ArrayList<XMLData> ();
284     opfItemsList = thisParser.extractDataFromXML (locationOfOPFFile, inputDataList);
285     int count = 0;
286     //epub3.1 : Check for a MANIFEST item with "properties" attribute containing the word "
287     cover-image"
288     foreach (string properties in opfItemsList[3].extractedTagAttributes) {
289         if (properties.contains ("cover-image")) {
290             //Get media type for the cover items
291             string coverMediaType = opfItemsList[1].extractedTagAttributes.get (count);
292             //get cover location if media type matches "image"
293             if (coverMediaType.index_of ("image") != -1) {
294                 bookCoverLocation = opfItemsList[2].extractedTagAttributes.get (count);
295                 bookCoverLocation = aBook.getBaseLocationOfContents () + bookCoverLocation
296 ;
297                 break;
298             }
299             count++;
300         }
301         //If cover could not be located in properties="cover-image" :
302         //Check for a MANIFEST item with "id" attribute containing the word "cover"
303         if (bookCoverLocation.length < 1 &&
304             "true" == BookwormApp.Utils.fileOperations ("EXISTS", "", bookCoverLocation, ""))
305         {
306             count = 0;
307             foreach (string id in opfItemsList[0].extractedTagAttributes) {
308                 if (id.contains ("cover")) {
309                     //Get media type for the cover items
310                     string coverMediaType = opfItemsList[1].extractedTagAttributes.get (count)
311 ;
312                     //get cover location if media type matches "image"
313                     if (coverMediaType.index_of ("image") != -1) {
314                         bookCoverLocation = opfItemsList[2].extractedTagAttributes.get (count)
315 ;
316                         bookCoverLocation = aBook.getBaseLocationOfContents () + bookCoverLoca
317 tion;
318                         break;
319                     }
320                 }
321                 count++;
322             }
323         }
324         //check if cover was still not found and assign flag for default cover to be used
325         if (bookCoverLocation.length < 1 &&
326             "true" == BookwormApp.Utils.fileOperations ("EXISTS", "", bookCoverLocation, ""))
327         {
328             aBook.setIsBookCoverImagePresent (false);
329             debug ("Cover image not found for book located at:" + aBook.getBookExtractionLocat
ion ());
330         } else {
331             //copy cover image to bookworm cover image cache

```

```

328         aBook = BookwormApp.Utils.setBookCoverImage (aBook, bookCoverLocation);
329     }
330     info ("[END] [FUNCTION:setCoverImage] book.location=" + aBook.getBookLocation () + " ,
bookCoverLocation=" + bookCoverLocation);
331     return aBook;
332 }
333
334 public static BookwormApp.Book setBookMetaData (owned BookwormApp.Book aBook, string locat
ionOfOPFFfile) {
335     info ("[START] [FUNCTION:setBookMetaData] book.location=" +
336           aBook.getBookLocation () + ", locationOfOPFFfile=" + locationOfOPFFfile);
337     //Parse OPF xml file to read the book meta data
338     ArrayList<XMLData> inputDownList = new ArrayList<XMLData> ();
339     inputDownList.add (new XMLData () {
340         containerTagName = "",
341         inputTagName = "title",
342         inputAttributeName = ""
343     });
344     inputDownList.add (new XMLData () {
345         containerTagName = "",
346         inputTagName = "creator",
347         inputAttributeName = ""
348     });
349     XmlParser thisParser = new XmlParser ();
350     ArrayList<XMLData> opfItemsList = new ArrayList<XMLData> ();
351     opfItemsList = thisParser.extractDataFromXML (locationOfOPFFfile, inputDownList);
352     if (opfItemsList[0].extractedTagValues.size > 0) {
353         string bookTitle = opfItemsList[0].extractedTagValues.get (0);
354         if (bookTitle.length > 0) {
355             aBook.setBookTitle (BookwormApp.Utils.decodeHTMLChars (bookTitle));
356             debug ("Determined eBook Title as:" + bookTitle);
357         } else {
358             //If the book title has not been determined, use the file name as book title
359             if (aBook.getBookTitle () != null && (
360                 aBook.getBookTitle () == BookwormApp.Constants.TEXT_FOR_UN
KNOWN_TITLE ||
361                 aBook.getBookTitle ().length < 1
362             )
363         ) {
364             bookTitle = File.new_for_path (aBook.getBookExtractionLocation ()).get_bas
ename ();
365             if (bookTitle.last_index_of (".") != -1) {
366                 bookTitle = bookTitle.slice (0, bookTitle.last_index_of ("."));
367             }
368             aBook.setBookTitle (bookTitle);
369             debug ("File name set as Title:" + bookTitle);
370         }
371     }
372 }
373 //determine the author of the book
374 if (opfItemsList[1].extractedTagValues.size > 0) {
375     string bookAuthor = opfItemsList[1].extractedTagValues.get (0);
376     if (bookAuthor.length > 0) {
377         aBook.setBookAuthor (BookwormApp.Utils.decodeHTMLChars (bookAuthor));
378         debug ("Determined eBook Author as:" + bookAuthor);
379     } else {
380         //If the book author has not been determined, use a default text for author
381         aBook.setBookAuthor (BookwormApp.Constants.TEXT_FOR_UNKNOWN_TITLE);
382         debug ("Could not determine eBook Author, default Author set");
383     }
384 }
385 info ("[END] [FUNCTION:setBookMetaData]");
386 return aBook;
387 }
388 }
```

```

contentHandler.vala ? Content Handler
=====

1  /* Copyright 2017 Siddhartha Das (bablu.boy@gmail.com)
2  *
3  * This file is part of Bookworm and is used for handling the eBook contents
4  * The prerequisite for the content handler is for the eBook contents to have
5  * been parsed into HTML format
6  *
7  * Bookworm is free software: you can redistribute it
8  * and/or modify it under the terms of the GNU General Public License as
9  * published by the Free Software Foundation, either version 3 of the
10 * License, or (at your option) any later version.
11 *
12 * Bookworm is distributed in the hope that it will be
13 * useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
14 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
15 * Public License for more details.
16 *
17 * You should have received a copy of the GNU General Public License along
18 * with Bookworm. If not, see http://www.gnu.org/licenses/.
19 */
20
21 using Gee;
22 public class BookwormApp.contentHandler {
23     public static BookwormApp.Settings settings;
24
25     public static BookwormApp.Book renderPage (owned BookwormApp.Book aBook, owned string dire
ction) {
26         debug ("[START] [FUNCTION:renderPage] book.location=" + aBook.getBookLocation () + ", "
direction=" + direction);
27         int currentContentLocation = aBook.getBookPageNumber ();
28         bool shouldReloadPage = true;
29         //set page number based on direction of navigation
30         switch (direction) {
31             case "FORWARD"://This is for moving the book forward
32                 if (aBook.getIfPageForward ()) {
33                     currentContentLocation++;
34                     aBook.setBookPageNumber (currentContentLocation);
35                 }
36                 break;
37             case "BACKWARD"://This is for moving the book backwards
38                 if (aBook.getIfPageBackward ()) {
39                     currentContentLocation--;
40                     aBook.setBookPageNumber (currentContentLocation);
41                 }
42                 break;
43             case "SEARCH"://Load the page and scroll to the search text
44                 break;
45             default://This is for opening the current page of the book
46                 //Do not change the page number
47                 break;
48         }
49         string bookContent = contentHandler.provideContent (aBook, currentContentLocation, dire
ction);
50         //render the content on webview
51         BookwormApp.AppWindow.aWebView.load_html (bookContent, BookwormApp.Constants.PREFIX_FO
R_FILE_URL);
52         //set the bookmark icon on the header
53         handleBookMark ("DISPLAY");
54         //set the navigation controls
55         aBook = controlNavigation (aBook);
56         //set the current value of the page slider
57         BookwormApp.AppWindow.pageAdjustment.set_value (currentContentLocation + 1);
58         debug ("[END] [FUNCTION:renderPage]");
59         return aBook;
60     }
61
62     public static string provideContent (owned BookwormApp.Book aBook, int contentLocation, st
ring mode) {
63         debug ("[START] [FUNCTION:provideContent] book.location=" + aBook.getBookLocation () +
64             ", contentLocation=" + contentLocation.to_string () + ", mode=" + mode);
65         StringBuilder contents = new StringBuilder ();

```

```

66         if (aBook.getBookContentList () != null) {
67             string bookLocationToRead = "";
68             if (contentLocation > -1 && aBook.getBookContentList ().size > contentLocation) {
69                 bookLocationToRead = aBook.getBookContentList ().get (contentLocation);
70                 if("true" != BookwormApp.Utils.fileOperations ("EXISTS", bookLocationToRead, "
71                     "", ""));
72                     //handle the case when the content list has html escape chars for the URI
73                     bookLocationToRead = BookwormApp.Utils.decodeHTMLChars (aBook.getBookContentList ().get (contentLocation));
74                     if("true" != BookwormApp.Utils.fileOperations ("EXISTS", bookLocationToRead,
75                         "", ""));
75                     aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_NAVIGATION_ISSUE);
76             };
77             BookwormApp.AppWindow.showInfoBar (aBook, Gtk.MessageType.WARNING);
78             warning ("[END] [FUNCTION:provideContent] Page could not be loaded from
79             location:"+bookLocationToRead);
80             return contents.str;
81         }
82         //fetch content from extracted book
83         contents.assign (BookwormApp.Utils.fileOperations ("READ_FILE", bookLocationTo
84             Read, "", ""));
85         //find list of relative urls with src, href, etc and convert them to absolute
86         ones
87         foreach (string tagname in BookwormApp.Constants.TAG_NAME_WITH_PATHS) {
88             string[] srcList = BookwormApp.Utils.multiExtractBetweenTwoStrings (con
89             tents.str, tagname, "\\");
90             StringBuider srcItemFullPath = new StringBuider ();
91             foreach (string srcItem in srcList) {
92                 srcItemFullPath.assign (
93                     BookwormApp.Utils.getFullPathFromFilename (
94                         aBook.getBookExtractionLocation (), srcItem)
95                 );
96                 contents.assign (
97                     contents.str.replace (tagname + srcItem + "\\",
98                         BookwormApp.Utils.encodeHTMLChars (tagname + srcItemFullPath.str
99                         + "\\")));
100            }
101            //update the content for required manipulation
102            contents.assign (adjustPageContent (aBook, contents.str, mode));
103            //handle the case for contentLocation set to -1 when the book is added to the DB
104            } else if (contentLocation == -1 && aBook.getBookContentList ().size > 0) {
105                provideContent (aBook, 0, mode);
106            } else {
107                //requested content not available
108                aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_NAVIGATION_ISSUE);
109                BookwormApp.AppWindow.showInfoBar (aBook, Gtk.MessageType.WARNING);
110            }
111        }
112        debug ("[END] [FUNCTION:provideContent] contents.length=" + contents.str.length.to_s
113        ting ());
114        return contents.str;
115    }
116
117    public static void handleBookMark (string action) {
118        debug ("[START] [FUNCTION:handleBookMark] action=" + action);
119        //get the book being currently read
120        BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap.get (BookwormApp.Bookworm
121        .locationOfEBookCurrentlyRead);
122        switch (action) {
123            case "DISPLAY":
124                if (aBook != null && aBook.getBookmark () != null && aBook.getBookmark ().inde
125                    x_of (aBook.getPageNumber ().to_string ()) != -1) {
126                        //display bookmark as active
127                        BookwormApp.AppHeaderBar.bookmark_active_button.set_visible (true);
128                        BookwormApp.AppHeaderBar.bookmark_inactive_button.set_visible (false);
129                    } else {
130                        //display bookmark as inactive
131                        BookwormApp.AppHeaderBar.bookmark_active_button.set_visible (false);

```

```

129             BookwormApp.AppHeaderBar.bookmark_inactive_button.set_visible (true);
130         }
131         break;
132     case "ACTIVE_CLICKED":
133         BookwormApp.AppHeaderBar.bookmark_active_button.set_visible (false);
134         BookwormApp.AppHeaderBar.bookmark_inactive_button.set_visible (true);
135         //set the bookmark
136         aBook.setBookmark (aBook.getBookPageNumber (), action);
137         break;
138     case "INACTIVE_CLICKED":
139         BookwormApp.AppHeaderBar.bookmark_active_button.set_visible (true);
140         BookwormApp.AppHeaderBar.bookmark_inactive_button.set_visible (false);
141         //set the bookmark
142         aBook.setBookmark (aBook.getBookPageNumber (), action);
143         break;
144     default:
145         break;
146     }
147     //update book details to libraryView Map
148     if (aBook != null) {
149         debug ("updating libraryViewMap with bookmark info...");
150         BookwormApp.Bookworm.libraryViewMap.set (BookwormApp.Bookworm.locationOfEBookCurre
ntlyRead, aBook);
151     }
152     debug ("[END] [FUNCTION:handleBookMark]");
153 }
154
155     public static string adjustPageContent (BookwormApp.Book aBook, owned string pageContentSt
r, string mode) {
156         debug ("[START] [FUNCTION:adjustPageContent] book.location=" + aBook.getBookLocation (
) +
157             ", pageContentStr.length=" + pageContentStr.length.to_string () + ", mode=" + mode
);
158         //wrap the html content in a div tag for pagination
159         StringBuilder pageContent = new StringBuilder (pageContentStr);
160         //Remove the empty title if it is present
161         pageContent.assign (pageContent.str
162             .replace ("<title/>", ""))
163         ;
164         constructOnLoadJavascript(aBook, mode);
165         string currentBookwormStyles = createPageStyles(aBook, pageContentStr);
166
167         BookwormApp.AppWindow.aWebView.get_user_content_manager().remove_all_style_sheets();
168         BookwormApp.AppWindow.aWebView.get_user_content_manager().add_style_sheet(
169             new WebKit.UserStyleSheet(currentBookwormStyles,
170                 WebKit.UserContentInjectedFrames.ALL_FRAMES,
171                 WebKit.UserStyleLevel.AUTHOR,
172                 null, null)
173         );
174         debug ("[END] [FUNCTION:adjustPageContent] pageContent.length=" + pageContent.str.leng
th.to_string ());
175         debug ("\n\n\n" + pageContent.str);
176         return pageContent.str;
177     }
178
179     private static string createPageStyles(BookwormApp.Book aBook, owned string pageContentStr
) {
180         if (BookwormApp.Bookworm.bookwormStyles == null || BookwormApp.Bookworm.bookwormStyles
.length < 1) {
181             uint8[] bookwormStylesData;
182             GLib.File.new_for_uri (BookwormApp.Constants.HTML_SCRIPT_STYLES_RESOURCE_LOCATION)
183                 .load_contents (null, out bookwormStylesData, null);
184             BookwormApp.Bookworm.bookwormStyles = (string)bookwormStylesData;
185             debug ("Loaded styles data from resource:\n" + BookwormApp.Bookworm.bookwormStyles
);
186         }
187         string currentBookwormStyles = BookwormApp.Bookworm.bookwormStyles;
188         string cssForTextAndBackgroundColor = "";
189
190         //For the Title Page (first or second page), resize height and width of images
191         if (aBook.getBookPageNumber () < 2 && (pageContentStr.contains ("<image") || pageConte
ntStr.contains ("<img")))) {
192             currentBookwormStyles = currentBookwormStyles.replace ("$TITLE_PAGE_IMAGE", "img,
193             image");
194         }

```

```

194         //Set background and font colour based on profile
195         if (BookwormApp.Constants.BOOKWORM_READING_MODE[4] == BookwormApp.Bookworm.settings.reading_profile) {
196             //default dark profile
197             cssForTextAndBackgroundColor = " background-color: #002b36 !important; color: #93a1a1 !important;";
198             currentBookwormStyles = currentBookwormStyles
199                 .replace ("$SCROLLBAR_BACKGROUND", "#002b36")
200                 .replace ("$HIGHLIGHT_COLOR", "#3465A4");
201         } else if (BookwormApp.Constants.BOOKWORM_READING_MODE[3] == BookwormApp.Bookworm.settings.reading_profile) {
202             //default light profile
203             cssForTextAndBackgroundColor = " background-color: #fbfbfb !important; color: #000000 !important;";
204             currentBookwormStyles = currentBookwormStyles
205                 .replace ("$SCROLLBAR_BACKGROUND", "#fbfbfb")
206                 .replace ("$HIGHLIGHT_COLOR", "#E8ED00");
207         } else if (BookwormApp.Constants.BOOKWORM_READING_MODE[2] == BookwormApp.Bookworm.settings.reading_profile) {
208             cssForTextAndBackgroundColor =
209                 " background-color: " + BookwormApp.Bookworm.profileColorList[7] + " !important;
t;" +
210                 " color: " + BookwormApp.Bookworm.profileColorList[6] + " !important;
t;";
211             currentBookwormStyles = currentBookwormStyles
212                 .replace ("$SCROLLBAR_BACKGROUND", BookwormApp.Bookworm.profileColorList[7])
213                 .replace ("$HIGHLIGHT_COLOR", BookwormApp.Bookworm.profileColorList[8]);
214         } else if (BookwormApp.Constants.BOOKWORM_READING_MODE[1] == BookwormApp.Bookworm.settings.reading_profile) {
215             cssForTextAndBackgroundColor =
216                 " background-color: " + BookwormApp.Bookworm.profileColorList[4] + " !important;
t;" +
217                 " color: " + BookwormApp.Bookworm.profileColorList[3] + " !important;
t;";
218             currentBookwormStyles = currentBookwormStyles
219                 .replace ("$SCROLLBAR_BACKGROUND", BookwormApp.Bookworm.profileColorList[4])
220                 .replace ("$HIGHLIGHT_COLOR", BookwormApp.Bookworm.profileColorList[5]);
221         } else {
222             cssForTextAndBackgroundColor =
223                 " background-color: " + BookwormApp.Bookworm.profileColorList[1] + " !important;
t;" +
224                 " color: " + BookwormApp.Bookworm.profileColorList[0] + " !important;
t;";
225             currentBookwormStyles = currentBookwormStyles
226                 .replace ("$SCROLLBAR_BACKGROUND", BookwormApp.Bookworm.profileColorList[1])
227                 .replace ("$HIGHLIGHT_COLOR", BookwormApp.Bookworm.profileColorList[2]);
228         }
229         //Set up CSS for book as per preference settings - this will override any css in the book contents
230         currentBookwormStyles = currentBookwormStyles
231             .replace ("$READING_LINE_HEIGHT", BookwormApp.Bookworm.settings.reading_line_height)
232             .replace ("$READING_WIDTH", (100 - int.parse (BookwormApp.Bookworm.settings.reading_width)).to_string ())
233             .replace ("$FONT_FAMILY", BookwormApp.Bookworm.settings.reading_font_name_family)
234             .replace ("$FONT_SIZE", BookwormApp.Bookworm.settings.reading_font_size.to_string ())
235             .replace ("$READING_TEXT_ALIGN", BookwormApp.Bookworm.settings.text_alignment)
236             .replace ("$TEXT_AND_BACKGROUND_COLOR", cssForTextAndBackgroundColor);
237         return currentBookwormStyles;
238     }
239
240     private static void constructOnLoadJavascript(BookwormApp.Book aBook, string mode) {
241         settings = BookwormApp.Settings.get_instance ();
242         //BookwormApp.Bookworm.onLoadJavaScript.assign ("onload=\"\"");
243         BookwormApp.Bookworm.onLoadJavaScript.assign ("");
244
245         //Scroll to the previous vertical position - this should be used:
246         // (1) when the book is re-opened from the library and
247         // (2) when a book existing in the library is opened from File Explorer using Bookworm
248         // (3) when clicking on a link in the TableOfContents which has an anchor
249         //The flag for applying the javascript is set from the above locations
250         if (BookwormApp.Bookworm.isPageScrollRequired) {
251             //check if an Anchor is present and set up the javascript for the same
252             if (aBook.getAnchor ().length > 0) {

```

```

253         BookwormApp.Bookworm.onLoadJavaScript.append (
254             " document.getElementById('" + aBook.getAnchor () + "').scrollIntoView();"
255         );
256         aBook.setAnchor("");
257     } else { //set up the javascript for scrolling to last read position
258         BookwormApp.Bookworm.onLoadJavaScript.append (" window.scrollTo (0," + (
259             BookwormApp.Bookworm.libraryViewMap.get (
260                 BookwormApp.Bookworm.locationOfEBookCurrentlyRead)
261                 .getBookScrollPos ().to_string () + ")");
262     }
263     BookwormApp.Bookworm.isPageScrollRequired = false; // stop this function being cal
led subsequently
264 }
265 //If two page view id required - add a script to set the CSS for two-page if there are
more than 500 chars
266 if (BookwormApp.Bookworm.settings.is_two_page_enabled) {
267     BookwormApp.Bookworm.onLoadJavaScript.append (" setTwoPageView ();");
268 }
269 //Overlay any Annotated text
270 foreach (var entry in aBook.getAnnotationList ().entries) {
271     if (aBook.getBookPageNumber ().to_string () == entry.key.split ("#~~#")[0]) {
272         BookwormApp.Bookworm.onLoadJavaScript.append (" overlayAnnotation ('" + entry.
key.split ("#~~#")[1] + "'');");
273     }
274 }
275
276 //Highlight and Scroll To Search String on page if required
277 if ("SEARCH" == mode) {
278     if (BookwormApp.Bookworm.bookTextSearchString.length > 1) {
279         string[] searchTokens = BookwormApp.Bookworm.bookTextSearchString.split ("#~~#
");
280         if (searchTokens.length == 2) {
281             //limit the search string to one word on either side of search text
282             int startPosOfSearchString = searchTokens[1].index_of (searchTokens[0]);
283             int endPosOfSearchString = startPosOfSearchString + searchTokens[0].length
;
284             int lengthOfLineWithSearchString = searchTokens[1].length;
285             int countSpaces = 0;
286             int startPosOfStringToBeHighlighted = 0;
287             int endPosOfStringToBeHighlighted = 0;
288             string stringToBeHighlighted = "";
289             if (startPosOfSearchString != -1) {
290                 //get the position of the word before the searched phrase
291                 for (int i=startPosOfSearchString; i>1; i--) {
292                     //match the second space before the search string
293                     if (" " == searchTokens[1].slice (i, i + 1)) {
294                         countSpaces++;
295                     }
296                     if (countSpaces == 2) {
297                         startPosOfStringToBeHighlighted = i + 1;
298                         break;
299                     }
300                 }
301                 //get the position of the word after the searched phrase
302                 countSpaces = 0;
303                 for (int j=endPosOfSearchString; j<lengthOfLineWithSearchString; j++)
{
304                     //match the second space before the search string
305                     if (" " == searchTokens[1].slice (j, j + 1)) {
306                         countSpaces++;
307                     }
308                     if (countSpaces == 2) {
309                         endPosOfStringToBeHighlighted = j;
310                         break;
311                     }
312                 }
313                 //form the string to be highlighted
314                 if (endPosOfStringToBeHighlighted > startPosOfStringToBeHighlighted) {
315                     stringToBeHighlighted = searchTokens[1].slice (
316                         startPosOfStringToBeHighlighted, endPosOfStringToBeHighlighted
);
317                 }
318             }
319             stringToBeHighlighted = stringToBeHighlighted.replace ("\\"", """).rep
lace ("\"", "\\"");
320

```

```

321             debug ("Searching to highlight the phrase:" + stringToBeHighlighted);
322             BookwormApp.Bookworm.onLoadJavaScript
323                 .append (" highlightText(encodeURIComponent('" + stringToBeHighlighted
324                     + "''));" );
325             }
326         }
327     //complete the onload javascript string
328     //BookwormApp.Bookworm.append ("\\" );
329 }
330
331 public static void searchHTMLContents () {
332     debug ("[START] [FUNCTION:searchHTMLContents]");
333     StringBuilder bookSearchResults = new StringBuilder ("");
334     int searchResultCount = 1;
335     BookwormApp.Bookworm.searchResultsMap.clear ();
336     //execute search
337     bookSearchResults.assign (
338         BookwormApp.Utils.execute_sync_command (
339             BookwormApp.Constants.SEARCH_SCRIPT_LOCATION +
340             "\\" + BookwormApp.Bookworm.aContentFileToBeSearched.str + "\\" +
341             BookwormApp.AppHeaderBar.headerSearchBar.get_text () + "\\");
342     //process search results
343     if (bookSearchResults.str.strip ().length > 0 && bookSearchResults.str != "false") {
344         string[] individualLines = bookSearchResults.str.strip ().split ("\n",-1);
345         foreach (string individualLine in individualLines) {
346             BookwormApp.Bookworm.searchResultsMap.set (
347                 searchResultCount.to_string () + "~" +
348                 BookwormApp.Bookworm.aContentFileToBeSearched.str,
349                 individualLine.strip ());
350             searchResultCount++;
351         }
352     }
353     debug ("[END] [FUNCTION:searchHTMLContents]");
354 }
355
356 public static BookwormApp.Book controlNavigation (owned BookwormApp.Book aBook) {
357     info ("[START] [FUNCTION:controlNavigation] book.location=" + aBook.getBookLocation ());
358     int currentContentLocation = aBook.getBookPageNumber ();
359     debug ("In controlNavigation with currentContentLocation=" + currentContentLocation.to
360     _string ());
361     //check if Book can be moved back and disable back button otherwise
362     if (currentContentLocation > 0) {
363         aBook.setIfPageBackward (true);
364         BookwormApp.AppWindow.back_button.set_sensitive (true);
365     } else {
366         aBook.setIfPageBackward (false);
367         BookwormApp.AppWindow.back_button.set_sensitive (false);
368     }
369     //check if Book can be moved forward and disable forward button otherwise
370     if (currentContentLocation < (aBook.getBookContentList ().size - 1)) {
371         aBook.setIfPageForward (true);
372         BookwormApp.AppWindow.forward_button.set_sensitive (true);
373     } else {
374         aBook.setIfPageForward (false);
375         BookwormApp.AppWindow.forward_button.set_sensitive (false);
376     }
377     info ("[END] [FUNCTION:controlNavigation] book.location=" + aBook.getBookLocation ());
378     return aBook;
379 }
380
381 public static void refreshCurrentPage () {
382     debug ("[START] [FUNCTION:refreshCurrentPage]");
383     if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
384     TATES[1]) {
385         BookwormApp.Book currentBookForRefresh = BookwormApp.Bookworm.libraryViewMap.get (
386             BookwormApp.Bookworm.locationOfEBookCurrentlyRead);
387         BookwormApp.Bookworm.isPageScrollRequired = true; //set up the flag to scroll to t
388         he last read position
389         currentBookForRefresh = renderPage (
390             BookwormApp.Bookworm.libraryViewMap.get (
391                 BookwormApp.Bookworm.locationOfEBookCurrentlyRead), "");
392         BookwormApp.Bookworm.libraryViewMap.set (
393             BookwormApp.Bookworm.locationOfEBookCurrentlyRead,

```

```

391             currentBookForRefresh);
392         }
393     debug ("[END] [FUNCTION:refreshCurrentPage]");
394 }
395
396 public static int getScrollPos () {
397     debug ("[START] [FUNCTION:getScrollPos]");
398     //This function is responsible for returning the vertical scroll position of the webvi
ew
399     //This should be called when the user leaves reading a book :
400     // (1) Return to Library button on Header Bar
401     // (2) Close Bookworm while in reading mode
402     // (3) Move to info view using Info button on Header Bar
403     int scrollPos = -1;
404     scrollPos = int.parse (BookwormApp.Utils.setWebViewTitle ("document.title = window.scr
ollY;"));
405     debug ("[START] [FUNCTION:getScrollPos] scrollPos=" + scrollPos.to_string ());
406     return scrollPos;
407 }
408
409 public static void performStartUpActions () {
410     debug ("[START] [FUNCTION:performStartUpActions]");
411     //load javascript data from resource if it has not been loaded already
412     if (BookwormApp.Bookworm.bookwormScripts == null || BookwormApp.Bookworm.bookwormScrip
ts.length < 1) {
413         uint8[] bookwormScriptsData;
414         GLib.File.new_for_uri (BookwormApp.Constants.HTML_SCRIPT_FUNCTIONS_RESOURCE_LOCATI
ON)
415         .load_contents (null, out bookwormScriptsData, null);
416         BookwormApp.Bookworm.bookwormScripts = (string)bookwormScriptsData;
417         WebKit.UserScript userScript = new WebKit.UserScript(BookwormApp.Bookworm.bookworm
Scripts, WebKit.UserContentInjectedFrames.ALL_FRAMES, WebKit.UserScriptInjectionTime.END, null,
null);
418         BookwormApp.AppWindow.aWebView.get_user_content_manager().add_script(userScript);
419         debug ("Loaded javascript data from resource:\n" + BookwormApp.Bookworm.bookwormSc
ripts);
420     }
421     //open the book added, if only one book path is present on command line
422     //if this book was not in the library, then the library view will be shown
423     if (BookwormApp.Bookworm.pathsOfBooksToBeAdded.length == 2 && //check if only one book
is on the command line
424         //check if first parameter is bookworm
425         BookwormApp.Constants.bookworm_id == BookwormApp.Bookworm.pathsOfBooksToBeAdded[0]
&&
426         //check if book has not already failed to load
427         BookwormApp.Bookworm.pathsOfBooksNotAddedStr.str.index_of (BookwormApp.Bookworm.pa
thsOfBooksToBeAdded[1]) == -1)
428     {
429         BookwormApp.Book requestedBook = null;
430         //Check if the requested book is available in the library
431         if (BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr.str.index_of (
BookwormApp.Bookworm.commandLineArgs[1].strip ()) != -1)
432     {
433         //pick the book from the Initial ArrayList used for holding the books in the l
ibrary
435         //as the BookwormApp.Bookworm.libraryViewMap would not have finished loading
foreach (BookwormApp.Book aBook in BookwormApp.Library.listOfBooksInLibraryOnL
oad) {
437             if (BookwormApp.Bookworm.commandLineArgs[1].strip () == aBook.getBookLocat
ion ()) {
438                 requestedBook = aBook;
439                 break;
440             }
441         }
442     } else {
443         //pick the book from the BookwormApp.Bookworm.libraryViewMap as it would have
been added
444         //as part of the code above to create a new book
445         requestedBook = BookwormApp.Bookworm.libraryViewMap.get (BookwormApp.Bookworm.
commandLineArgs[1].strip ());
446     }
447     debug ("Bookworm opened for single book[" + requestedBook.getBookLocation () + "]"
- proceed to reading view...);
448     if (requestedBook != null) {
449         //set the name of the book being currently read

```

```
450             BookwormApp.Bookworm.locationOfEBookCurrentlyRead = BookwormApp.Bookworm.comma
451             ndLineArgs[1].strip ();
452                 //Initiate Reading the book
453                 BookwormApp.Bookworm.readSelectedBook (requestedBook);
454             }
455         } else {
456             //check and continue the last book being read - if "Always show library on start i
s false"
457             if ((!BookwormApp.Bookworm.settings.is_show_library_on_start) && (BookwormApp.Book
worm.settings.book_being_read != ""))
458                 //check if the library contains the book being read last
459                 BookwormApp.Book lastReadBook = BookwormApp.DB.getBookFromDB (BookwormApp.Book
worm.settings.book_being_read);
460                 if (lastReadBook.getBookLocation () != null && lastReadBook.getBookLocation ()
.length > 1) {
461                     debug ("Opening the last read book [" + BookwormApp.Bookworm.settings.book
_being_read + "]");
462                     BookwormApp.Bookworm.locationOfEBookCurrentlyRead = BookwormApp.Bookworm.s
ettings.book_being_read;
463                     //Initiate Reading the book
464                     BookwormApp.Bookworm.readSelectedBook (lastReadBook);
465                 } else {
466                     warning ("The last read book [" + BookwormApp.Bookworm.settings.book_being
_read + "] " +
467                         "was not found in the library, so showing the library view instead of
opening the last read book");
468                 }
469             }
470             debug ("[END] [FUNCTION:performStartUpActions]");
471         }
472     }
```

```
=====
window.vala ? Main Window
=====

1  /* Copyright 2017 Siddhartha Das (bablu.boy@gmail.com)
2  *
3  * This file is part of Bookworm and is used for drawing the
4  * window components for both the library view and the reading view
5  *
6  * Bookworm is free software: you can redistribute it
7  * and/or modify it under the terms of the GNU General Public License as
8  * published by the Free Software Foundation, either version 3 of the
9  * License, or (at your option) any later version.
10 *
11 * Bookworm is distributed in the hope that it will be
12 * useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
13 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
14 * Public License for more details.
15 *
16 * You should have received a copy of the GNU General Public License along
17 * with Bookworm. If not, see http://www.gnu.org/licenses/.
18 */
19 using Gtk;
20 using Gee;
21 using Granite.Widgets;
22
23 public class BookwormApp.AppWindow {
24     public static Gtk.InfoBar infobar;
25     public static Gtk.Box bookLibrary_ui_box;
26     public static Gtk.FlowBox library_grid;
27     public static Gtk.ListStore library_table_liststore;
28     public static TreeIter library_table_iter;
29     public static Gtk.TreeView library_table_treeview;
30     public static Gtk.Label infobarLabel;
31     public static ScrolledWindow library_grid_scroll;
32     public static ScrolledWindow library_list_scroll;
33     public static WebKit.WebView aWebView;
34     public static WebKit.Settings webkitSettings;
35     public static Gtk.EventBox book_reading_footer_eventbox;
36     public static Gtk.ActionBar book_reading_footer_box;
37     public static Gtk.Box bookReading_ui_box;
38     public static Gtk.Button forward_button;
39     public static Gtk.GestureSwipe gesture_swipe;
40     public static Gtk.Button back_button;
41     public static Gtk.ProgressBar bookAdditionBar;
42     public static Adjustment pageAdjustment;
43     public static Scale pageSlider;
44     public static BookwormApp.Settings settings;
45     public static bool isWebViewRequestCompleted = true;
46     public static Gtk.Button remove_book_button;
47     public static Gtk.Button page_button_prev;
48     public static Gtk.Button page_button_next;
49     public static int noOfBooksSelected = 0;
50
51
52     public static Gtk.Box createBoookwormUI () {
53         info ("[START] [FUNCTION:createBoookwormUI]");
54         settings = BookwormApp.Settings.get_instance ();
55
56         //Create a grid to display the books cover images in library
57         library_grid = new Gtk.FlowBox ();
58         library_grid.set_border_width (BookwormApp.Constants.SPACING_WIDGETS);
59         library_grid.column_spacing = BookwormApp.Constants.SPACING_WIDGETS;
60         library_grid.row_spacing = BookwormApp.Constants.SPACING_WIDGETS;
61         library_grid.set_valign (Gtk.Align.START);
62
63         library_grid_scroll = new ScrolledWindow (null, null);
64         library_grid_scroll.set_policy (PolicyType.AUTOMATIC, PolicyType.AUTOMATIC);
65         library_grid_scroll.add (library_grid);
66
67         //Create a treeview and Liststore to display the list of books in the library
68         library_table_liststore = new Gtk.ListStore (8,
69             typeof (Gdk.Pixbuf), typeof (string), typeof (string), typeof (string),
70             typeof (Gdk.Pixbuf), typeof (string), typeof (string), typeof (string)));
71 }
```

```

71     library_table_treeview = new Gtk.TreeView ();
72     library_table_treeview.activate_on_single_click = true;
73     //Set up the various cell types for the library metadata
74     CellRendererPixbuf selection_cell_pix = new CellRendererPixbuf ();
75     CellRendererText non_editable_cell_txt = new CellRendererText ();
76     CellRendererText title_cell_txt = new CellRendererText ();
77     title_cell_txt.editable = true;
78     CellRendererText author_cell_txt = new CellRendererText ();
79     author_cell_txt.editable = true;
80     CellRendererPixbuf rating_cell_pix = new CellRendererPixbuf ();
81     CellRendererText tags_cell_txt = new CellRendererText ();
82     tags_cell_txt.editable = true;
83     //Set up Treeview columns
84     library_table_treeview.insert_column_with_attributes (-1, " ", selection_cell_pix, "pi
xbuf", 0);
85     library_table_treeview.insert_column_with_attributes (-1, BookwormApp.Constants.TEXT_F
OR_LIST_VIEW_COLUMN_NAME_TITLE, title_cell_txt, "text", 1);
86     library_table_treeview.insert_column_with_attributes (-1, BookwormApp.Constants.TEXT_F
OR_LIST_VIEW_COLUMN_NAME_AUTHOR, author_cell_txt, "text", 2);
87     library_table_treeview.insert_column_with_attributes (-1, BookwormApp.Constants.TEXT_F
OR_LIST_VIEW_COLUMN_NAME_MODIFIED_DATE, non_editable_cell_txt, "text", 3);
88     library_table_treeview.insert_column_with_attributes (-1, BookwormApp.Constants.TEXT_F
OR_LIST_VIEW_COLUMN_NAME_RATING, rating_cell_pix, "pixbuf", 4);
89     library_table_treeview.insert_column_with_attributes (-1, BookwormApp.Constants.TEXT_F
OR_LIST_VIEW_COLUMN_NAME_TAGS, tags_cell_txt, "text", 5);
90
91     library_list_scroll = new ScrolledWindow (null, null);
92     library_list_scroll.set_policy (PolicyType.AUTOMATIC, PolicyType.AUTOMATIC);
93     library_list_scroll.add (library_table_treeview);
94
95     //Create a box to hold the grid view and list view - only one is visible at a time
96     Gtk.Box library_view_box = new Gtk.Box (Orientation.VERTICAL, 0);
97     library_view_box.set_border_width (0);
98     library_view_box.pack_start (library_grid_scroll, true, true, 0);
99     library_view_box.pack_start (library_list_scroll, true, true, 0);
100    //Set up Button for selecting books
101    Gtk.Button select_book_button = new Gtk.Button ();
102    select_book_button.set_image (BookwormApp.Bookworm.select_book_image);
103    select_book_button.set_relief (ReliefStyle.NONE);
104    select_book_button.set_tooltip_markup (BookwormApp.Constants.TOOLTIP_TEXT_FOR_SELECT_B
OOK);
105
106    //Set up Button for adding books
107    Gtk.Button add_book_button = new Gtk.Button ();
108    add_book_button.set_image (BookwormApp.Bookworm.add_book_image);
109    add_book_button.set_relief (ReliefStyle.NONE);
110    add_book_button.set_tooltip_markup (BookwormApp.Constants.TOOLTIP_TEXT_FOR_ADD_BOOK);
111
112    //Set up Button for removing books
113    remove_book_button = new Gtk.Button ();
114    remove_book_button.set_image (BookwormApp.Bookworm.remove_book_image);
115    remove_book_button.set_relief (ReliefStyle.NONE);
116    //set the button as disabled - it will be enabled only if books are selected
117    remove_book_button.set_sensitive (false);
118    remove_book_button.set_tooltip_markup (BookwormApp.Constants.TOOLTIP_TEXT_FOR_REMOVE_B
OOK_UNSELECTED);
119
120    //Set up buttons for paginating the library
121    Gtk.Box library_page_switcher_box = new Gtk.Box (Orientation.HORIZONTAL, 0);
122    library_page_switcher_box.set_border_width (0);
123
124    page_button_prev = new Gtk.Button ();
125    page_button_prev.set_image (BookwormApp.Bookworm.back_page_image);
126    page_button_prev.set_relief (ReliefStyle.NONE);
127    page_button_prev.set_tooltip_markup (BookwormApp.Constants.TOOLTIP_TEXT_FOR_PREV_PAGE)
;
128    library_page_switcher_box.pack_start (page_button_prev);
129    page_button_prev.set_sensitive (false); //disable the prev button on first time load
130
131    page_button_next = new Gtk.Button ();
132    page_button_next.set_image (BookwormApp.Bookworm.forward_page_image);
133    page_button_next.set_relief (ReliefStyle.NONE);
134    page_button_next.set_tooltip_markup (BookwormApp.Constants.TOOLTIP_TEXT_FOR_NEXT_PAGE)
;
135    library_page_switcher_box.pack_start (page_button_next);

```

```

136
137     //Set up the progress bar for addition of books to library
138     bookAdditionBar = new Gtk.ProgressBar ();
139     bookAdditionBar.set_valign (Gtk.Align.CENTER);
140     bookAdditionBar.set_show_text (true);
141
142     //Create a footer and add widgets for select/add/remove books
143     ActionBar add_remove_footer_box = new ActionBar ();
144     add_remove_footer_box.pack_start (select_book_button);
145     add_remove_footer_box.pack_start (add_book_button);
146     add_remove_footer_box.pack_start (remove_book_button);
147     add_remove_footer_box.pack_start (bookAdditionBar);
148     add_remove_footer_box.pack_end (library_page_switcher_box);
149
150     //Create a MessageBar to show status messages
151     infobar = new Gtk.InfoBar ();
152     infobarLabel = new Gtk.Label ("");
153     Gtk.Container infobarContent = infobar.get_content_area ();
154     infobar.set_message_type (MessageType.INFO);
155     infobarContent.add (infobarLabel);
156     infobar.set_show_close_button (true);
157     infobar.response.connect (on_info_bar_closed);
158     infobar.hide ();
159
160     //Create the UI for library view and add all components to ui box for library view
161     bookLibrary_ui_box = new Gtk.Box (Orientation.VERTICAL, 0);
162     bookLibrary_ui_box.set_border_width (0);
163     bookLibrary_ui_box.pack_start (library_view_box, true, true, 0);
164     bookLibrary_ui_box.pack_start (add_remove_footer_box, false, true, 0);
165
166     //create the webview to display page content
167     webkitSettings = new WebKit.Settings ();
168     webkitSettings.set_allow_file_access_from_file_urls (true);
169     webkitSettings.set_allow_universal_access_from_file_urls (true); //this gives launchpa
d build error for Yaketty
170     webkitSettings.set_auto_load_images (true);
171     aWebView = new WebKit.WebView.with_settings (webkitSettings);
172     aWebView.set_zoom_level (BookwormApp.Settings.get_instance ().zoom_level);
173     aWebView.load_changed.connect((loadEvent) => {
174         switch (loadEvent) {
175             case WebKit.LoadEvent.STARTED:
176                 break;
177             case WebKit.LoadEvent.REDIRECTED:
178                 break;
179             case WebKit.LoadEvent.COMMITTED:
180                 break;
181             case WebKit.LoadEvent.FINISHED:
182                 aWebView.run_javascript (BookwormApp.Bookworm.onLoadJavaScript.str, null);
183                 break;
184         }
185     });
186     webkitSettings.set_enable_javascript (true);
187     //This is for setting the font to the system font - Is this required ?
188     //webkitSettings.set_default_font_family (aWebView.get_style_context ().get_font (Stat
eFlags.NORMAL).get_family ());
189     webkitSettings.set_default_font_size (BookwormApp.Bookworm.settings.reading_font_size)
;
190     webkitSettings.set_default_font_family (BookwormApp.Bookworm.settings.reading_font_nam
e);
191     gesture_swipe = new Gtk.GestureSwipe(aWebView);
192     gesture_swipe.set_propagation_phase(Gtk.PropagationPhase.CAPTURE);
193
194     //Set up Button for previous page
195     back_button = new Gtk.Button ();
196     back_button.set_image (BookwormApp.Bookworm.back_button_image);
197     back_button.set_relief (ReliefStyle.NONE);
198
199     //Set up Button for next page
200     forward_button = new Gtk.Button ();
201     forward_button.set_image (BookwormApp.Bookworm.forward_button_image);
202     forward_button.set_relief (ReliefStyle.NONE);
203
204     //Set up a slider for jumping pages
205     pageAdjustment = new Adjustment (0, 1, 100, 1, 0, 0);
206     pageSlider = new Gtk.Scale (Gtk.Orientation.HORIZONTAL, pageAdjustment);

```

```

207     pageSlider.set_digits (0);
208     pageSlider.set_valign (Gtk.Align.START);
209     pageSlider.set_value_pos (Gtk.PositionType.RIGHT);
210     pageSlider.set_hexpand (true);
211
212     //Set up contents of the footer
213     book_reading_footer_box = new ActionBar ();
214     book_reading_footer_box.pack_start (back_button);
215     book_reading_footer_box.pack_start (pageSlider);
216     book_reading_footer_box.pack_end (forward_button);
217
218     //Create the Gtk Box to hold components for reading a selected book
219     bookReading_ui_box = new Gtk.Box (Orientation.VERTICAL, 0);
220     bookReading_ui_box.set_border_width (0);
221     bookReading_ui_box.pack_start (aWebView, true, true, 0);
222     bookReading_ui_box.pack_start (book_reading_footer_box, false, true, 0);
223
224     //Add all ui components to the main UI box
225     Gtk.Box main_ui_box = new Gtk.Box (Orientation.VERTICAL, 0);
226     main_ui_box.set_border_width (0);
227     main_ui_box.pack_start (infobar, false, true, 0);
228     main_ui_box.pack_start (bookLibrary_ui_box, true, true, 0);
229     main_ui_box.pack_start (BookwormApp.Info.createBookInfo (), true, true, 0);
230     main_ui_box.pack_end (bookReading_ui_box, true, true, 0);
231
232     //Add action to open a book for clicking on row in library list view
233     library_table_treeview.row_activated.connect ((path, column) => {
234         Gtk.TreeIter iter;
235         Value bookLocation;
236         TreeModel aTreeModel = library_table_treeview.get_model ();
237         aTreeModel.get_iter (out iter, path);
238         aTreeModel.get_value (iter, 7, out bookLocation);
239         BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap.get ((string) bookLoc
ation);
240         if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_
UI_STATES[6] ||
241             BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_
UI_STATES[7])
242             {
243                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_U
I_STATES[7];
244                 BookwormApp.Library.updateListViewForSelection (aBook);
245             }
246         if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_
UI_STATES[5])
247             BookwormApp.Bookworm.readSelectedBook (aBook);
248         }
249     });
250     //Add action to update tree view when editing is Completed
251     title_cell_txt.edited.connect ((path, new_text) => {
252         updateLibraryListViewData (path, new_text, 1);
253     });
254     author_cell_txt.edited.connect ((path, new_text) => {
255         updateLibraryListViewData (path, new_text, 2);
256     });
257     tags_cell_txt.edited.connect ((path, new_text) => {
258         updateLibraryListViewData (path, new_text, 5);
259     });
260     //Add action to open the context menu on right click of tree view
261     library_table_treeview.button_press_event.connect ((event) => {
262         //capture which mouse button was clicked on the book in the library
263         uint mouseButtonClicked;
264         event.get_button (out mouseButtonClicked);
265         //handle right button click for context menu
266         if (event.get_event_type () == Gdk.EventType.BUTTON_PRESS && mouseButtonClicked ==
3) {
267             /*TreeIter iter;
268             TreeModel model;
269             Value bookLocation;
270             TreeSelection selection = library_table_treeview.get_selection ();
271             selection.get_selected (out model, out iter);
272             model.get_value (iter, 0, out bookLocation);
273             BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap.get ((string) boo
kLocation);
274             TODO: Set up the right click context

```

```

275         */
276     };
277     return false; //return false to propagate the action further i.e. row activation
278 });
279 // Add action to go to next or previous page in response to a finger
280 // swipe gesture from right to left to or left to right respectively
281 gesture_swipe.connect((x, y) => {
282     // Avoid triggering navigation actions on mostly vertical swipes that
283     // should scroll up or down the page rather than flip it.
284     // The x and y-values here are relatively arbitrary but seems to feel
285     // right in testing.
286     if (y.abs() > 800 || x.abs() < 800) {
287         return;
288     }
289
290     // x == 0 on tap, so we ignore that
291     if (x > 0) {
292         handleBookNavigation("PREV");
293     } else if (x < 0) {
294         handleBookNavigation("NEXT");
295     }
296 });
297 //Add action on the forward button for reading
298 forward_button.clicked.connect (() => {
299     handleBookNavigation ("NEXT");
300 });
301 //Add action on the backward button for reading
302 back_button.clicked.connect (() => {
303     handleBookNavigation ("PREV");
304 });
305 //Add action for moving the pages for the page slider
306 pageSlider.change_value.connect ((scroll, new_value) => {
307     debug ("Page Slider value change [" + new_value.to_string () +
308           "] Initiated for book at location:" + BookwormApp.Bookworm.locationOfEBookCurr
309           entlyRead);
310     BookwormApp.Book currentBookForSlider = new BookwormApp.Book ();
311     currentBookForSlider = BookwormApp.Bookworm.libraryViewMap.get (BookwormApp.Bookwo
312     rm.locationOfEBookCurrentlyRead);
313     if ((int.parse (new_value.to_string ())-1) > (currentBookForSlider.getBookContentL
314     ist ().size)) {
315         //this is for the scenario where the slider crosses the max value
316         currentBookForSlider.setBookPageNumber (currentBookForSlider.getBookContentLis
317         t ().size-1);
318         } else {
319             currentBookForSlider.setBookPageNumber (int.parse (new_value.to_string ())-1);
320             }
321             //update book details to libraryView Map
322             currentBookForSlider = BookwormApp.contentHandler.renderPage (currentBookForSlider
323             , "");
324             BookwormApp.Bookworm.libraryViewMap.set (currentBookForSlider.getBookLocation (),
325             currentBookForSlider);
326             BookwormApp.Bookworm.locationOfEBookCurrentlyRead = currentBookForSlider.getBookLo
327             cation ();
328             debug ("Page Slider value change action completed for book at location:" +
329                 BookwormApp.Bookworm.locationOfEBookCurrentlyRead +
330                 " and rendering completed for page number:" + currentBookForSlider.getBookPage
331                 Number ().to_string ());
332                 return true;
333             });
334             //Add action for adding book (s) on the library view
335             add_book_button.clicked.connect (() => {
336                 ArrayList<string> selectedEBooks = BookwormApp.Utils.selectFileChooser (
337                     Gtk.FileChooserAction.OPEN, _("Select eBook"), BookwormApp.Bookworm.window, tr
338                     ue, "EBOOKS");
339                 BookwormApp.Bookworm.pathsOfBooksToBeAdded = new string[selectedEBooks.size];
340                 int countOfBooksToBeAdded = 0;
341                 foreach (string pathToSelectedBook in selectedEBooks) {
342                     BookwormApp.Bookworm.pathsOfBooksToBeAdded[countOfBooksToBeAdded] = pathToSelec
343                     ctedBook;
344                     countOfBooksToBeAdded++;
345                 }
346                 //Display the progress bar
347                 BookwormApp.AppWindow.bookAdditionBar.show ();
348                 BookwormApp.Bookworm.isBookBeingAddedToLibrary = true;
349                 BookwormApp.Library.addBooksToLibrary ();

```

```

340     });
341     //Add action for putting library in select view
342     select_book_button.clicked.connect (() => {
343         //initialize the counter to check how many books are selected
344         noOfBooksSelected = 0;
345         //check if the library is in List View mode
346         if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_
347             UI_STATES[5] || BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_
348             UI_STATES[6] || BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_
349             UI_STATES[7]) {
350             //check if the mode is already in selection mode
351             if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW_
352                 ORM_UI_STATES[6] || BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW_
353                 ORM_UI_STATES[7]) {
354                 //UI is already in selection/selected mode - second click puts the view in
355                 //normal mode
356                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKW_
357                 RM_UI_STATES[5];
358                 BookwormApp.Library.updateListViewForSelection (null);
359             } else {
360                 //UI is not in selection/selected mode - set the view mode to selection mo
361                 de
362                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKW_
363                 RM_UI_STATES[6];
364                 BookwormApp.Library.updateListViewForSelection (null);
365             }
366         } else {
367             //check if the mode is already in selection mode
368             if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW_
369                 ORM_UI_STATES[2] || BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW_
370                 ORM_UI_STATES[3]) {
371                 //UI is already in selection/selected mode - second click puts the view in
372                 //normal mode
373                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = settings.library_view_mode;
374                 BookwormApp.Library.updateGridViewForSelection (null);
375             } else {
376                 //UI is not in selection/selected mode - set the view mode to selection mo
377                 de
378                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKW_
379                 RM_UI_STATES[2];
380                 BookwormApp.Library.updateGridViewForSelection (null);
381             }
382         }
383     });
384     //Add action for removing a selected book on the library view
385     remove_book_button.clicked.connect (() => {
386         BookwormApp.Library.removeSelectedBooksFromLibrary ();
387     });
388     //handle mouse click on webview (reading mode)
389     aWebView.button_press_event.connect ((event) => {
390         if (!settings.is_leaf_over_page_by_edge_enabled) {
391             return false;
392         }
393         int width;
394         int height;
395         //capture the current window size
396         BookwormApp.Bookworm.window.get_size (out width, out height);
397         //capture which mouse button was clicked on the book in the library
398         uint mouseButtonClicked;
399         event.get_button (out mouseButtonClicked);
400         //handle left button click for page navigation if the click is near the left and r
401         ight margins
402         if (event.get_event_type () == Gdk.EventType.BUTTON_PRESS && mouseButtonClicked ==
403             1) {
404             //check if mouse is clicked near the right margin 10% of page width and go to
405             previous page
406             if(event.x < ((BookwormApp.Constants.PERCENTAGE_WIDTH_FOR_PAGE_NAVIGATION_ON_C
407             LICK/100) * width)){

```

```

397             handleBookNavigation ("PREV");
398         }
399         if(event.x > (width - ((BookwormApp.Constants.PERCENTAGE_WIDTH_FOR_PAGE_NAVIGATION_ON_CLICK/100) * width))){
400             handleBookNavigation ("NEXT");
401         }
402     };
403     return false; //return false to propagate the action further
404 });
405 //handle context menu on the webview reader
406 aWebView.context_menu.connect ((context_menu, event, hit_test_result) => {
407     context_menu.remove_all ();
408     SimpleAction pageActionFullScreenEntry = new SimpleAction ("FULL_SCREEN_READING_VIEW", null);
409     SimpleAction pageActionFullScreenExit = new SimpleAction ("FULL_SCREEN_READING_VIEW", null);
410     SimpleAction pageActionWordMeaning = new SimpleAction ("WORD_MEANING", null);
411     SimpleAction pageActionAnnotateSelection = new SimpleAction ("ANNOTATE_SELECTION", null);
412     WebKit.ContextMenuItem pageContextMenuWordMeaning = new WebKit.ContextMenuItem
413         .from_gaction (
414             pageActionWordMeaning, BookwormApp.Constants.TEXT_FOR_PAGE_CONTEXTMENU_WORD_MEANING, null);
415     WebKit.ContextMenuItem pageContextMenuFullScreenEntry = new WebKit.ContextMenuItem
416         .from_gaction (
417             pageActionFullScreenEntry, BookwormApp.Constants.TEXT_FOR_PAGE_CONTEXTMENU_FULLSCREEN_ENTRY, null);
418     WebKit.ContextMenuItem pageContextMenuFullScreenExit = new WebKit.ContextMenuItem
419         .from_gaction (
420             pageActionFullScreenExit, BookwormApp.Constants.TEXT_FOR_PAGE_CONTEXTMENU_FULLSCREEN_EXIT, null);
421     WebKit.ContextMenuItem pageContextMenuAnnotateSelection = new WebKit.ContextMenuItem
422         .from_gaction (
423             pageActionAnnotateSelection, BookwormApp.Constants.TEXT_FOR_PAGE_CONTEXTMENU_ANNOTATE_SELECTION, null);
424     context_menu.append (pageContextMenuWordMeaning);
425     context_menu.append (pageContextMenuAnnotateSelection);
426     if (!settings.is_fullscreen) {
427         context_menu.append (pageContextMenuFullScreenEntry);
428     } else {
429         context_menu.append (pageContextMenuFullScreenExit);
430     }
431     //Set Context menu items
432     pageActionWordMeaning.activate.connect (() => {
433         string selected_text = BookwormApp.Utils.setWebViewTitle ("document.title = getSelectionText ()");
434         if (selected_text != null && selected_text.length > 0) {
435             //Save the page scroll position of the book being read
436             BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap
437                 .get (BookwormApp.Bookworm.locationOfEBookCurrentlyRead);
438             aBook.setBookScrollPos (BookwormApp.contentHandler.getScrollPos ());
439
440             BookwormApp.Info.populateDictionaryResults (selected_text);
441         }
442     });
443     pageActionAnnotateSelection.activate.connect (() => {
444         string selected_text = BookwormApp.Utils.setWebViewTitle ("document.title = getSelectionText ()");
445         if (selected_text != null && selected_text.length > 0) {
446             BookwormApp.AppDialog.createAnnotationDialog (selected_text);
447         }
448     });
449     pageActionFullScreenEntry.activate.connect (() => {
450         book_reading_footer_box.hide ();
451         BookwormApp.Bookworm.window.fullscreen ();
452     });
453     pageActionFullScreenExit.activate.connect (() => {
454         book_reading_footer_box.show ();
455         BookwormApp.Bookworm.window.unfullscreen ();
456     });
457     return false;
458 });
459 //capture the url clicked on the webview and action the navigation type clicks
460 aWebView.decide_policy.connect ((decision, type) => {
461     if (type == WebKit.PolicyDecisionType.RESPONSE) {

```

```

458         debug ("Signal captured for Policy type WebKit.PolicyDecisionType.RESPONSE");
459         isWebViewRequestCompleted = true;
460     }
461     if (type == WebKit.PolicyDecisionType.NEW_WINDOW_ACTION) {
462         debug ("Signal captured for Policy type WebKit.PolicyDecisionType.NEW_WINDOW_A
463             CTION");
464     }
465     if (type == WebKit.PolicyDecisionType.NAVIGATION_ACTION && isWebViewRequestComplet
466 ed) {
467         debug ("Signal captured for Policy type WebKit.PolicyDecisionType.NAVIGATION_A
468             CTION");
469         //set the webview request flag to false to prevent re-trigger of this function
470         //the webview request flag will be set to true when the response is received
471         isWebViewRequestCompleted = false;
472         WebKit.NavigationPolicyDecision aNavDecision = (WebKit.NavigationPolicyDecisio
473 n)decision;
474         WebKit.NavigationAction aNavAction = aNavDecision.get_navigation_action ();
475         WebKit.URIRequest aURIReq = aNavAction.get_request ();
476         string url_clicked_on_webview = BookwormApp.Utils.decodeHTMLChars (aURIReq.get
477 _uri ().strip ());
478         url_clicked_on_webview = Soup.URI.decode (url_clicked_on_webview);
479         debug ("URL Captured:" + url_clicked_on_webview);
480         //Handle external links (not file:/// by opening the default browser i.e. http
481         ://, ftp://
482         if (url_clicked_on_webview.index_of ("file://") == -1) {
483             BookwormApp.Utils.execute_sync_command ("xdg-open " + url_clicked_on_webvi
484 ew);
485             decision.ignore ();
486             return true;
487         }
488         //Handle Bookworm type links i.e. Annotation Overlay
489         debug ("Window Title:" + BookwormApp.AppWindow.aWebView.get_title ());
490         if (BookwormApp.AppWindow.aWebView.get_title () != null &&
491             BookwormApp.AppWindow.aWebView.get_title ().length > 1 &&
492             BookwormApp.AppWindow.aWebView.get_title ().index_of ("annotation:") != -1
493         )
494         {
495             //Open the annotation dialog
496             BookwormApp.AppDialog.createAnnotationDialog (
497                 BookwormApp.AppWindow.aWebView.get_title ().replace ("annotation:", ""))
498         );
499         isWebViewRequestCompleted = true;
500     }
501     //Handle file:/// type links to other content of the book i.e. Table of Conten
502 ts
503     string anchor = "";
504     if (url_clicked_on_webview.index_of ("#") != -1) {
505         string[] url_splitted_by_hashtag = url_clicked_on_webview.split("#", 2);
506         url_clicked_on_webview = url_splitted_by_hashtag[0];
507         anchor = url_splitted_by_hashtag[1];
508     }
509     url_clicked_on_webview = File.new_for_path (url_clicked_on_webview).get_basena
510 me ();
511     int contentLocationPosition = 0;
512     BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap
513         .get (BookwormApp.Bookworm.locationOfEBookCurrentlyRead);
514     foreach (string aBookContent in aBook.getBookContentList ()) {
515         if (BookwormApp.Utils.decodeHTMLChars (aBookContent).index_of (url_clicked
516 _on_webview) != -1) {
517             debug ("Matched Link Clicked to book content:" +
518                 BookwormApp.Utils.decodeHTMLChars (aBookContent));
519             aBook.setBookPageNumber (contentLocationPosition);
520             //update book details to libraryView Map
521             BookwormApp.Bookworm.libraryViewMap.set (aBook.getBookLocation (), aBo
522 ok);
523             if (anchor.len() > 0) { // anchor - id in link after # symbol
524                 BookwormApp.Bookworm.isPageScrollRequired = true;
525                 aBook.setAnchor(anchor);
526             }
527             aBook = BookwormApp.contentHandler.renderPage (aBook, "");
528             //Set the mode back to Reading mode
529             BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BO
530 OKWORM_UI_STATES[1];
531     }
532     BookwormApp.Bookworm.toggleUIState ();

```

```

519                 debug ("URL is initiated from Bookworm Contents, Book page number set
520                 at:" +
521                     aBook.getBookPageNumber ().to_string ());
522                 break;
523             }
524         contentLocationPosition++;
525     }
526     isWebViewRequestCompleted = true;
527     return true;
528 });
529 //Add action for paginating the library
530 page_button_next.clicked.connect (() => {
531     handleLibraryPageButtons ("NEXT_PAGE", true);
532 });
533 page_button_prev.clicked.connect (() => {
534     handleLibraryPageButtons ("PREV_PAGE", true);
535 });
536 info ("[END] [FUNCTION:createBoookwormUI]");
537 return main_ui_box;
538 }
539
540 public static void handleBookNavigation (string action){
541     //action for NEXT page
542     if(action == "NEXT") {
543         //get object for this ebook and call the next page
544         BookwormApp.Book currentBookForForward = new BookwormApp.Book ();
545         currentBookForForward = BookwormApp.Bookworm.libraryViewMap.get (BookwormApp.Bookw
orm.locationOfEBookCurrentlyRead);
546         debug ("Initiating read forward for eBook:" + currentBookForForward.getBookLocatio
n ());
547         currentBookForForward = BookwormApp.contentHandler.renderPage (currentBookForForwa
rd, "FORWARD");
548         //update book details to libraryView Map
549         BookwormApp.Bookworm.libraryViewMap.set (currentBookForForward.getBookLocation (),c
urrentBookForForward);
550         BookwormApp.Bookworm.locationOfEBookCurrentlyRead = currentBookForForward.getBookL
ocation ();
551     }
552     //action for PREV page
553     if(action == "PREV") {
554         //get object for this ebook and call the next page
555         BookwormApp.Book currentBookForReverse = new BookwormApp.Book ();
556         currentBookForReverse = BookwormApp.Bookworm.libraryViewMap.get (BookwormApp.Bookw
orm.locationOfEBookCurrentlyRead);
557         debug ("Initiating read previous for eBook:" + currentBookForReverse.getBookLocati
on ());
558         currentBookForReverse = BookwormApp.contentHandler.renderPage (currentBookForRever
se, "BACKWARD");
559         //update book details to libraryView Map
560         BookwormApp.Bookworm.libraryViewMap.set (currentBookForReverse.getBookLocation (),c
urrentBookForReverse);
561         BookwormApp.Bookworm.locationOfEBookCurrentlyRead = currentBookForReverse.getBookL
ocation ();
562     }
563 }
564
565 public static void handleLibraryPageButtons (string mode, bool isPaginateRequired) {
566     if (mode == "NEXT_PAGE" && isPaginateRequired) {
567         //activate the previous page button if it is disabled
568         if (!page_button_prev.get_sensitive ()) {
569             page_button_prev.set_sensitive (true);
570         }
571         //move the counter to the next position
572         BookwormApp.Bookworm.current_page_counter = BookwormApp.Bookworm.current_page_coun
ter + 1;
573         BookwormApp.Library.paginateLibrary ("", "PAGINATED_SEARCH");
574         //disable the forward button if the last modification date returned -1 for this pa
ge position
575         if (BookwormApp.Bookworm.paginationlist.contains ("-1")) {
576             page_button_next.set_sensitive (false);
577         }
578     }
579     if (mode == "PREV_PAGE" && isPaginateRequired) {
580         if (BookwormApp.Bookworm.current_page_counter > 0) {

```

```

581         //activate the next page button if it is disabled
582         if (!page_button_next.get_sensitive ()) {
583             page_button_next.set_sensitive (true);
584         }
585         //remove -1 from the paginated list if present of last modification dates to a
586         //allow the forward button to work
587         if (BookwormApp.Bookworm.paginationlist.contains ("-1")) {
588             BookwormApp.Bookworm.paginationlist.remove ("-1");
589         }
590         BookwormApp.Bookworm.current_page_counter = BookwormApp.Bookworm.current_page_
591         counter - 1;
592         BookwormApp.Library.paginateLibrary ("", "PAGINATED_SEARCH");
593     } else {
594         //disable the prev button as the counter is on the first page
595         page_button_prev.set_sensitive (false);
596     }
597     if (!isPaginateRequired) { //set the button status without doing pagination
598         if (BookwormApp.Bookworm.current_page_counter < 1) {
599             page_button_prev.set_sensitive (false);
600         }
601         if (BookwormApp.Bookworm.paginationlist.contains ("-1")) {
602             page_button_next.set_sensitive (false);
603         }
604     }
605 }
606
607 public static bool updateLibraryListViewData (string path, string new_text, int column) {
608     info ("[START] [FUNCTION:updateLibraryListViewData] updating metadata in List View on
609     row:" + +
610         path + " for change:" + new_text + " on column:" + column.to_string ());
611     //Determine the book whose meta data is being updated
612     Gtk.TreeIter sortedIter;
613     Value bookLocation;
614     TreeModel aTreeModel = library_table_treeview.get_model ();
615     Gtk.TreePath aTreePath = new Gtk.TreePath.from_string (path);
616     aTreeModel.get_iter (out sortedIter, aTreePath);
617     aTreeModel.get_value (sortedIter, 7, out bookLocation);
618     //iterate over the list store
619     Gtk.TreeIter iter;
620     string bookLocationforCurrentRow;
621     bool iterExists = true;
622     iterExists = library_table_liststore.get_iter_first (out iter);
623     while (iterExists) {
624         library_table_liststore.get (iter, 7, out bookLocationforCurrentRow);
625         if ((string)bookLocation == bookLocationforCurrentRow) {
626             library_table_liststore.set (iter, column, new_text);
627             BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap.get ((string) boo
628             kLocation);
629             if (column == 1) {
630                 aBook.setBookTitle (new_text);
631             }
632             if (column == 2) {
633                 aBook.setBookAuthor (new_text);
634             }
635             if (column == 5) {
636                 aBook.setBookTags (new_text);
637             }
638             aBook.setWasBookOpened (true);
639             BookwormApp.Bookworm.libraryViewMap.set (aBook.getBookLocation (), aBook);
640             debug ("Completed updating metadata in List View for book:" + (string) bookLoc
641             ation);
642             return true; //break out of the iterations
643         }
644     }
645 }
646
647 public static Granite.Widgets.Welcome createWelcomeScreen () {
648     info ("[START] [FUNCTION:createWelcomeScreen]");
649     //Create a welcome screen for view of library with no books
650     BookwormApp.Bookworm.welcomeWidget = new Granite.Widgets.Welcome (
651         BookwormApp.Constants.TEXT_FOR_WELCOME_MESSAGE_TITLE,

```

```

651         BookwormApp.Constants.TEXT_FOR_WELCOME_MESSAGE_SUBTITLE);
652     Gtk.Image? openFolderImage = new Gtk.Image.from_icon_name ("document-open", Gtk.IconSize.DIALOG);
653     BookwormApp.Bookworm.welcomeWidget.append_with_image (
654         openFolderImage, "Open", BookwormApp.Constants.TEXT_FOR_WELCOME_OPENDIR_MESSAGE);
655     //Add action for adding a book on the library view
656     BookwormApp.Bookworm.welcomeWidget.activated.connect (() => {
657         ArrayList<String> selectedEBooks = BookwormApp.Utils.selectFileChooser (
658             Gtk.FileChooserAction.OPEN, _("Select eBook"), BookwormApp.Bookworm.window, true,
659             "EBOOKS");
660         //If ebooks were selected, remove the welcome widget from main window and show the
661         //library view
662         if (selectedEBooks.size > 0) {
663             BookwormApp.Bookworm.window.remove (BookwormApp.Bookworm.welcomeWidget);
664             BookwormApp.Bookworm.window.add (BookwormApp.Bookworm.bookWormUIBox);
665             BookwormApp.Bookworm.bookWormUIBox.show_all ();
666             BookwormApp.Bookworm.toggleUIState ();
667             BookwormApp.Bookworm.pathsOfBooksToBeAdded = new string[selectedEBooks.size];
668             int countOfBooksToBeAdded = 0;
669             foreach (string pathToSelectedBook in selectedEBooks) {
670                 BookwormApp.Bookworm.pathsOfBooksToBeAdded[countOfBooksToBeAdded] = pathTo
671                 SelectedBook;
672                 countOfBooksToBeAdded++;
673             }
674             //Display the progress bar
675             BookwormApp.AppWindow.bookAdditionBar.show ();
676             BookwormApp.Bookworm.isBookBeingAddedToLibrary = true;
677             BookwormApp.Library.addBooksToLibrary ();
678         }
679     });
680     info ("[END] [FUNCTION:createWelcomeScreen] ");
681     return BookwormApp.Bookworm.welcomeWidget;
682 }
683
684 public static void showInfoBar (BookwormApp.Book aBook, MessageType aMessageType) {
685     debug ("[START] [FUNCTION:showInfoBar] ");
686     StringBuilder message = new StringBuilder ("");
687     message.append (aBook.getParsingIssue ()).append (aBook.getBookLocation ());
688     BookwormApp.AppWindow.infobarLabel.set_text (message.str);
689     BookwormApp.AppWindow.infobar.set_message_type (aMessageType);
690     BookwormApp.AppWindow.infobar.show ();
691     debug ("[END] [FUNCTION:showInfoBar] with message:" + message.str);
692 }
693
694 //Handle action for close of the InfoBar
695 public static void on_info_bar_closed () {
696     BookwormApp.AppWindow.infobar.hide ();
697 }
698
699 public static bool handleWindowStateEvents (Gdk.EventWindowState ev) {
700     if (ev.type == Gdk.EventType.WINDOW_STATE) {
701         if ((ev.window.get_state () & Gdk.WindowState.FULLSCREEN) == 0) {
702             settings.is_fullscreen = false;
703         } else {
704             settings.is_fullscreen = true;
705         }
706     }
707     return false;
708 }
709
710 public static void controlDeletionButton (bool selectionState) {
711     if (selectionState) {
712         //Enable the Deletion Button as a book is selected for potential removal
713         noOfBooksSelected++;
714         remove_book_button.set_sensitive (true);
715         remove_book_button.set_tooltip_markup (BookwormApp.Constants.TOOLTIP_TEXT_FOR_REMOVE_BOOK);
716     } else {
717         //Check and Disable the Deletion Button if no books are selected after this de-sel
718         noOfBooksSelected--;
719         if (noOfBooksSelected < 1) {
720             remove_book_button.set_sensitive (false);
721             remove_book_button.set_tooltip_markup (
722                 BookwormApp.Constants.TOOLTIP_TEXT_FOR_REMOVE_BOOK_UNSELECTED);

```

```
721         }
722     }
723 }
724
725
726 }
```