

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:

<code>main.vala</code>	Entry Point
<code>bookworm.vala</code>	Application Class
<code>book.vala</code>	Book Model
<code>library.vala</code>	Library View
<code>database.vala</code>	Database Layer
<code>ePubReader.vala</code>	EPUB Reader
<code>contentHandler.vala</code>	Content Handler
<code>window.vala</code>	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<string, 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.HANDLES_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_TASKS_SCRIPTS_DIR);
130        debug ("Bookworm Install Mobi Scripts Directory: " + BookwormApp.Constants.INSTALL_MOBI_SCRIPTS_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
OL_MASK);
214         accelMap.add_entry ("<Root-Window>/some-path", Gdk.Key.Cyrillic_TE, Gdk.ModifierTy
pe.CONTROL_MASK);
215         accelMap.add_entry ("<Root-Window>/some-path", Gdk.Key.Cyrillic_TE, Gdk.ModifierTy
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.Constant
292 s.SELECTION_CHECKED_IMAGE_SMALL_LOCATION);
293             image_selection_transparent_small = new Gdk.Pixbuf.from_resource (BookwormApp.Cons
294 tants.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", Gtk.IconSi
316 ze.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", Gtk.
323 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 Gtk.IconSize.MENU);
331             } else {
332                 add_scan_directory_image = new Gtk.Image.from_resource (BookwormApp.Constants.
333 ADD_BOOK_ICON_IMAGE_LOCATION);
334             }
335             if (Gtk.IconTheme.get_default ().has_icon ("list-remove-symbolic")) {
336                 remove_scan_directory_image = new Gtk.Image.from_icon_name ("list-remove-symbo
337 lic", Gtk.IconSize.MENU);
338             } else {
339                 remove_scan_directory_image = new Gtk.Image.from_resource (BookwormApp.Constan
340 ts.REMOVE_BOOK_ICON_IMAGE_LOCATION);
341             }
342             if (Gtk.IconTheme.get_default ().has_icon ("view-list-symbolic")) {
343                 library_list_button_image = new Gtk.Image.from_icon_name ("view-list-symbolic"
344 , Gtk.IconSize.SMALL_TOOLBAR);
345             } else {
346                 library_list_button_image = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_res
347 ource_at_scale (BookwormApp.Constants.LIBRARY_VIEW_LIST_IMAGE_LOCATION, 16, 16, true));
348             }
349             if (Gtk.IconTheme.get_default ().has_icon ("view-grid-symbolic")) {
350                 library_grid_button_image = new Gtk.Image.from_icon_name ("view-grid-symbolic"
351 , Gtk.IconSize.SMALL_TOOLBAR);
352             } else {
353                 library_grid_button_image = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_res
354 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
-symbolic", Gtk.IconSize.SMALL_TOOLBAR);
336         } else {
337             pref_menu_icon_align_left = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_res
ource_at_scale (BookwormApp.Constants.TEXT_ALIGN_LEFT_ICON_LOCATION, 16, 16, true));
338         }
339         if (Gtk.IconTheme.get_default ().has_icon ("format-justify-right-symbolic")) {
340             pref_menu_icon_align_right = new Gtk.Image.from_icon_name ("format-justify-rig
ht-symbolic", Gtk.IconSize.SMALL_TOOLBAR);
341         } else {
342             pref_menu_icon_align_right = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_re
source_at_scale (BookwormApp.Constants.TEXT_ALIGN_RIGHT_ICON_LOCATION, 16, 16, true));
343         }
344         if (Gtk.IconTheme.get_default ().has_icon ("help-info-symbolic")) {
345             content_list_button_image = new Gtk.Image.from_icon_name ("help-info-symbolic"
, Gtk.IconSize.LARGE_TOOLBAR);
346         } else {
347             content_list_button_image = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_res
ource_at_scale (BookwormApp.Constants.BOOK_INFO_IMAGE_LOCATION, 24, 24, true));
348         }
349         if (Gtk.IconTheme.get_default ().has_icon ("format-text-larger-symbolic")) {
350             menu_icon_text_large = new Gtk.Image.from_icon_name ("format-text-larger-symbo
lic", Gtk.IconSize.LARGE_TOOLBAR);
351         } else {
352             menu_icon_text_large = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_resource
_at_scale (BookwormApp.Constants.TEXT_LARGER_IMAGE_ICON_LOCATION, 24, 24, true));
353         }
354         if (Gtk.IconTheme.get_default ().has_icon ("open-menu")) {
355             menu_icon = new Gtk.Image.from_icon_name ("open-menu", Gtk.IconSize.LARGE_TOOL
BAR);
356         } else {
357             menu_icon = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_resource_at_scale (
BookwormApp.Constants.HEADERBAR_PROPERTIES_IMAGE_LOCATION, 24, 24, true));
358         }
359         if (Gtk.IconTheme.get_default ().has_icon ("format-text-larger-symbolic")) {
360             pref_menu_icon_text_large = new Gtk.Image.from_icon_name ("format-text-larger-
symbolic", Gtk.IconSize.LARGE_TOOLBAR);
361         } else {
362             pref_menu_icon_text_large = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_res
ource_at_scale (BookwormApp.Constants.TEXT_LARGER_IMAGE_ICON_LOCATION, 24, 24, true));
363         }
364         if (Gtk.IconTheme.get_default ().has_icon ("format-text-smaller-symbolic")) {
365             pref_menu_icon_text_small = new Gtk.Image.from_icon_name ("format-text-smaller
-symbolic", Gtk.IconSize.LARGE_TOOLBAR);
366         } else {
367             pref_menu_icon_text_small = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_res
ource_at_scale (BookwormApp.Constants.TEXT_SMALLER_IMAGE_ICON_LOCATION, 24, 24, true));
368         }
369         if (Gtk.IconTheme.get_default ().has_icon ("go-previous-symbolic")) {
370             back_button_image = new Gtk.Image.from_icon_name ("go-previous-symbolic", Gtk.
IconSize.MENU);
371         } else {
372             back_button_image = new Gtk.Image.from_resource (BookwormApp.Constants.PREV_PA
GE_ICON_IMAGE_LOCATION);
373         }
374         if (Gtk.IconTheme.get_default ().has_icon ("go-next-symbolic")) {
375             forward_button_image = new Gtk.Image.from_icon_name ("go-next-symbolic", Gtk.I
conSize.MENU);
376         } else {
377             forward_button_image = new Gtk.Image.from_resource (BookwormApp.Constants.NEXT
_PAGE_ICON_IMAGE_LOCATION);
378         }
379         if (Gtk.IconTheme.get_default ().has_icon ("go-previous-symbolic")) {
380             back_page_image = new Gtk.Image.from_icon_name ("go-previous-symbolic", Gtk.Ic
onSize.MENU);
381         } else {
382             back_page_image = new Gtk.Image.from_resource (BookwormApp.Constants.PREV_PAGE
_ICON_IMAGE_LOCATION);
383         }
384         if (Gtk.IconTheme.get_default ().has_icon ("go-next-symbolic")) {
385             forward_page_image = new Gtk.Image.from_icon_name ("go-next-symbolic", Gtk.Ico
nSize.MENU);
386         } else {

```



```

387         forward_page_image = new Gtk.Image.from_resource (BookwormApp.Constants.NEXT_P
AGE_ICON_IMAGE_LOCATION);
388     }
389     } catch (GLib.Error e) {
390         warning ("Image could not be loaded. Error:" + e.message);
391     }
392     info ("[END] [FUNCTION:loadImages]");
393 }
394
395 public static void loadCSSProvider (Gtk.CssProvider cssProvider) {
396     info ("[START] [FUNCTION:loadCSSProvider] cssProvider=" + cssProvider.to_string ());
397     string dynamicCSSContent = "";
398     try {
399         profileColorList = settings.list_of_profile_colors.split (",");
400         //temp check to ensure the change to settings for colours increasing from 6 to 9 i
s handled
401         if (profileColorList.length < 9) {
402             profileColorList = {"#000000", "#fbfbfb", "#E8ED00", "#586e75", "#fdf6e3", "#87FF2B",
"#93a1a1", "#002b36", "#3465A4"};
403         }
404         dynamicCSSContent = BookwormApp.Constants.DYNAMIC_CSS_CONTENT
405             .replace ("<profile_1_color>", profileColorList[0])
406             .replace ("<profile_1_bgcolor>", profileColorList[1])
407             .replace ("<profile_2_color>", profileColorList[3])
408             .replace ("<profile_2_bgcolor>", profileColorList[4])
409             .replace ("<profile_3_color>", profileColorList[6])
410             .replace ("<profile_3_bgcolor>", profileColorList[7]);
411         debug ("CSS for Profile Buttons:" + dynamicCSSContent);
412         cssProvider.load_from_data (dynamicCSSContent, dynamicCSSContent.length);
413     } catch (GLib.Error e) {
414         warning ("Stylesheet could not be loaded from CSS Content[" + dynamicCSSContent +
415             "]. Error:" + e.message);
416     }
417     Gtk.StyleContext.add_provider_for_screen (
418         Gdk.Screen.get_default (), cssProvider, Gtk.STYLE_PROVIDER_PRIORITY_APPLICATION);
419     info ("[END] [FUNCTION:loadCSSProvider]");
420 }
421
422 public void loadBookwormState () {
423     info ("[START] [FUNCTION:loadBookwormState]");
424     //check and create required directory structure
425     BookwormApp.Utills.fileOperations ("CREATEDIR", BookwormApp.Constants.EBOOK_EXTRACTION_
LOCATION, "", "");
426     BookwormApp.Utills.fileOperations ("CREATEDIR", BookwormApp.Bookworm.bookworm_config_pa
th, "", "");
427     BookwormApp.Utills.fileOperations ("CREATEDIR", BookwormApp.Bookworm.bookworm_config_pa
th + "/covers/", "", "");
428     BookwormApp.Utills.fileOperations ("CREATEDIR", BookwormApp.Bookworm.bookworm_config_pa
th + "/books/", "", "");
429     //Set the window to the last saved position
430     if (settings.pos_x == 0 && settings.pos_y == 0) {
431         window.set_position (Gtk.WindowPosition.CENTER);
432     } else {
433         window.move (settings.pos_x, settings.pos_y);
434     }
435     //set window size to the last saved height/width
436     if (settings.window_is_maximized) {
437         window.maximize ();
438     } else {
439         if (settings.window_width > 0 && settings.window_height > 0) {
440             window.set_default_size (settings.window_width, settings.window_height);
441         } else {
442             window.set_default_size (1200, 700);
443         }
444     }
445     //check last state and turn on dark theme
446     if (BookwormApp.Bookworm.settings.is_dark_theme_enabled) {
447         Gtk.Settings.get_default ().gtk_application_prefer_dark_theme = true;
448     }
449     //set the number of books per library page as per user set value
450     no_of_books_per_page = settings.library_page_items.to_string ();
451
452     //check if the database exists otherwise create database and required tables
453     BookwormApp.DB.initializeBookWormDB (BookwormApp.Bookworm.bookworm_config_path);
454     //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.
contentHandler.getScrollPos ());
471         //Save the path to the book being read
472         settings.book_being_read = locationOfEBookCurrentlyRead;
473     } else {
474         //Bookworm was not in reading view during close - remove path of book read last
475         settings.book_being_read = "";
476     }
477     //release the control so that the window is closed
478     Idle.add (closeBookWorm.callback);
479     yield;
480     //Update the book details to the database if it was opened in this session
481     foreach (var book in libraryViewMap.values) {
482         if (((BookwormApp.Book)book).getWasBookOpened ()) {
483             BookwormApp.DB.updateBookToDataBase ((BookwormApp.Book)book);
484             debug ("Completed saving the book data into DB");
485         }
486     }
487     //Run dicoverly of books as a background task if not already running
488     string checkBackgroundTask = BookwormApp.Utils.execute_sync_command ("ps -ef");
489     if (checkBackgroundTask.index_of ("bookworm --discover") == -1) {
490         BookwormApp.Utils.execute_async_multiarg_command_pipes ({ "com.github.babluboy.book
worm", "--discover", "&"});
491     } else {
492         debug ("Bookworm discover process already running...");
493     }
494     info ("[END] [FUNCTION:closeBookWorm]");
495 }
496
497 public void saveWindowState () {
498     int width;
499     int height;
500     int x;
501     int y;
502     window.get_size (out width, out height);
503     window.get_position (out x, out y);
504     if (settings.pos_x != x || settings.pos_y != y) {
505         settings.pos_x = x;
506         settings.pos_y = y;
507     }
508     if (settings.window_width != width || settings.window_height != height) {
509         settings.window_width = width;
510         settings.window_height = height;
511     }
512     if (window.is_maximized == true) {
513         settings.window_is_maximized = true;
514     } else {
515         settings.window_is_maximized = false;
516     }
517     settings.zoom_level = BookwormApp.AppWindow.aWebView.get_zoom_level ();
518 }
519
520 public static void readSelectedBook (owned BookwormApp.Book aBook) {
521     info ("[START] [FUNCTION:readSelectedBook] book.location=" + aBook.getBookLocation ());
522
523     //Fetch the book meta data from the database if it is not already available
524     if (aBook.getBookContentList ().size < 1) {
525         //content size should be greater than 1 if the book data has been loaded
526         aBook = BookwormApp.DB.getBookMetaDataFromDB (aBook);

```

```

526     }
527     debug ("Book details before attempting to open book for reading:" + aBook.to_string (
    ));
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
        position
533         //Enable the flag which will scroll the page to the last read position
534         isPageScrollRequired = true;
535     }
536     //Handle the case when the page number of the book is outside limits
537     if (aBook.getBookPageNumber () >= aBook.getBookContentList ().size) {
538         aBook.setBookPageNumber (aBook.getBookContentList ().size - 1);
539     }
540     //check if the extracted contents for the book exists
541     if (BookwormApp.Bookworm.settings.is_local_storage_enabled &&
542         "true" == BookwormApp.Utils.fileOperations ("DIR_EXISTS", aBook.getBookExtractionL
        ocation (), "", "") &&
543         aBook.getBookContentList () != null &&
544         aBook.getBookContentList ().size > 0 &&
545         aBook.getBookContentList ().size >= aBook.getBookPageNumber () &&
546         "true" == BookwormApp.Utils.fileOperations (
547             "EXISTS", BookwormApp.Utils.decodeHTMLChars (
548                 aBook.getBookContentList ().get (aBook.getBookPageNumber ()), "", "")) {
549         //extraction of book not required
550         aBook.setIsBookParsed (true);
551         debug ("Book has already been extracted and the extracted contents exist at:" + aB
        ook.getBookExtractionLocation ());
552     } else {
553         //Extract and Parse the eBook (this will overwrite the contents already extracted)
554         debug ("Extracted contents of the book was not found at expected location [" +
555             aBook.getBookExtractionLocation () + "], attempting to load book from original
        location [" +
556             aBook.getBookLocation () + "]);
557         aBook = genericParser (aBook);
558         //If ebook was not parsed successfully, show the warning info banner message
559         if (!aBook.getIsBookParsed ()) {
560             BookwormApp.AppWindow.showInfoBar (aBook, MessageType.WARNING);
561         }
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         locationOfEBookCurrentlyRead = 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.Bookwor
        m.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_S
TATES[0] ||
599             BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
TATES[2] ||
600             BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
TATES[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_S
TATES[5] ||
609             BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
TATES[6] ||
610             BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
TATES[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.
BOOKWORM_CURRENT_STATE);
672 }
673
674     public static BookwormApp.Book genericParser (owned BookwormApp.Book aBook) {
675         info ("[START] [FUNCTION:genericParser] book.location=" + aBook.getBookLocation ());
676         //set defaults for title and author - this will be over-ridden with extracted data if
found
677         aBook.setBookAuthor (BookwormApp.Constants.TEXT_FOR_UNKNOWN_TITLE);
678         aBook.setBookTitle (BookwormApp.Constants.TEXT_FOR_UNKNOWN_TITLE);
679         //check if ebook is present at provided location
680         if ("false" == BookwormApp.Utils.fileOperations ("EXISTS", "", aBook.getBookLocation (
), "")) {
681             warning ("EBook not found at provided location:" + aBook.getBookLocation ());
682             aBook.setIsBookParsed (false);
683             aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_EXTRACTION_ISSUE);
684             return aBook;
685         }
686         //determine the extension of the ebook file
687         string ebookFileName = (File.new_for_path (aBook.getBookLocation ()).get_basename ());
688         if (ebookFileName.index_of (".") != -1) {
689             string fileExtension = ebookFileName.substring (ebookFileName.last_index_of ("."))
.up ();
690             //parse file based on extension found
691             try {
692                 switch (fileExtension) {
693                     case ".EPUB":
694                         aBook = BookwormApp.ePubReader.parseEPubBook (aBook);
695                         break;
696                     case ".PDF":
697                         aBook = BookwormApp.pdfReader.parsePDFBook (aBook);
698                         break;
699                     case ".CBR":
700                         aBook = BookwormApp.comicsReader.parseComicsBook (aBook, fileExtension
);
701                         break;
702                     case ".CBZ":
703                         aBook = BookwormApp.comicsReader.parseComicsBook (aBook, fileExtension
);
704                         break;
705                     case ".MOBI":
706                         aBook = BookwormApp.mobiReader.parseMobiBook (aBook);
707                         break;
708                     case ".PRC":
709                         aBook = BookwormApp.mobiReader.parseMobiBook (aBook);
710                         break;
711                     case ".AZW3":
712                         aBook = BookwormApp.mobiReader.parseMobiBook (aBook);
713                         break;
714                     case ".ZIP":
715                         //check if the file is a zipped FB2 file
716                         if (ebookFileName.up ().last_index_of (".FB2.ZIP") != -1) {
717                             aBook = BookwormApp.fb2Reader.parseFictionBook (aBook);
718                             break;
719                         } else {
720                             aBook.setIsBookParsed (false);
721                             aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_FORMAT_NOT_S
UPPORTED);
722                             break;
723                         }
724                     case ".FB2":
725                         aBook = BookwormApp.fb2Reader.parseFictionBook (aBook);
726                         break;
727                     default:
728                         aBook.setIsBookParsed (false);
729                         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
47     //These variables are only available for the current session (not persisted)
48     private string opfFileLocation = "";
49     private string baseLocationOfContents = "";
50     private bool ifPageForward = true;
51     private bool ifPageBackward = true;
52     private bool isBookSelected = false;
53     private bool wasBookOpened = false;
54     private string bookAnchor = "";
55     private HashMap<string,Gtk.Widget> bookWidgetsList = new HashMap<string,Gtk.Widget> ();
56
57     //getter list for book id
58     public void setBookId (int aBookId) {
59         bookId = aBookId;
60     }
61     public int getBookId () {
62         return bookId;
63     }
64
65     //getter list for isBookParsedCorrectly
66     public void setIsBookParsed (bool isParsed) {
67         isBookParsedCorrectly = isParsed;
68     }
69     public bool getIsBookParsed () {
70         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         }
276         if ("INACTIVE_CLICKED" == action) {
277             bookmarks.append ("***" + pageNumber.to_string () + "***");
278         }
279         if (pageNumber == -10) { //this is used to set the bookmark fetched from the DB
280             //set -10 as the "pageNumber" and the book mark data as "action" when setting this
281             value from the DB
282             bookmarks.assign (action);
283         }
284     }
285     public string getBookmark () {
286         return bookmarks.str;
287     }
288
289     //getter setter for annotations
290     public void setAnnotations (string index, string annotationText) {
291         //check the value of the annotation text - add or delete accordingly
292         if (annotationText != null && annotationText.length > 0) {
293             //annotated text is not null/empty - update the annotation
294             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 (",\n")
345             .append ("bookPageNumber=").append (bookPageNumber.to_string ().append (",\n")
346             .append ("ifPageForward=").append (ifPageForward.to_string ().append (",\n")
347             .append ("ifPageBackward=").append (ifPageBackward.to_string ().append (",\n")
348             .append ("bookmarks=").append (bookmarks.str).append (",\n")
349             .append ("author=").append (bookAuthor).append (",\n")
350             .append ("ratings=").append (bookRating.to_string ().append (",\n")
351             .append ("tags=").append (bookTags.to_string ().append (",\n")
352             .append ("annotation tags=").append (annotationTags.to_string ().append (",\n")
353             .append ("bookContentList=");
354         for (int i = 0; i < bookContentList.size; i++) {
355             bookDetails.append ("[" + i.to_string () + "]= " + bookContentList.get (i) + ","");
356         }
357         return bookDetails.str;
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<BookwormApp.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.getBookLocation () +
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 (
on ()))));
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         modifiedElapsedTime = BookwormApp.Constants.TEXT_FOR_TIME_YESTERDAY;
69     } else if (timespan < 30 * TimeSpan.DAY) {
70         modifiedElapsedTime = daysElapsed.to_string () + " " + BookwormApp.Constants.T
EXT_FOR_TIME_DAYS;
71     } else {
72         modifiedElapsedTime = 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, modifiedElapsedTime,
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.DESENDING);
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.DESENDING);
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.DESENDING);
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.DESENDING);
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.DESENDING);
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 (
BookwormApp.Constants.PLACEHOLDER_COVER_IMAGE_LOCATION, 10, 200, false);
116         bookPlaceholderCoverImage = new Gtk.Image.from_pixbuf (bookPlaceholderCoverPix);
117     } catch (GLib.Error e) {
118         warning ("Error loading the placeholder cover image from location[" +
BookwormApp.Constants.PLACEHOLDER_COVER_IMAGE_LOCATION + "] : " +
119             e.message );
120     }
121 }
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.Utills.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.Utills.minimizeStringLength (title, 4 * BookwormApp.Constants.M
AX_NUMBER_OF_CHARS_FOR_BOOK_TITLE);
172             titleTextLabel.set_text (
173                 "<b>" + BookwormApp.Utills.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 ==
3) {
263             bookPopover.set_visible (true);
264             bookPopover.show_all ();
265             return true;
266         } else {
267             //left button click for reading or selection of book
268             if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
ORM_UI_STATES[0]) {
269                 aBook = BookwormApp.Bookworm.libraryViewMap.get (aEventBox.get_name ());
270                 BookwormApp.Bookworm.readSelectedBook (aBook);
271             }
272             if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
ORM_UI_STATES[2] ||
273                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
ORM_UI_STATES[3])
274             {
275                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWO
RM_UI_STATES[3];
276                 aBook = BookwormApp.Bookworm.libraryViewMap.get (aEventBox.get_name ());
277                 updateGridViewForSelection (aBook);
278             }
279             return true;
280         }
281     });
282     //Show the grid view based on state
283     if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
TATES[0] ||
284         BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
TATES[2] ||
285         BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
TATES[3])
286     {
287         BookwormApp.AppWindow.library_grid.show_all ();
288     }
289     debug ("Completed updating Library View for book:" + aBook.getBookLocation ());
290 }
291
292 public static void replaceCoverImageOnBook (owned BookwormApp.Book? book) {
293     debug ("[START] [FUNCTION:replaceCoverImageOnBook]");
294     //remove the existing overlay image
295     Gtk.Overlay oldOverlayImage = (Gtk.Overlay) book.getBookWidget ("BOOK_OVERLAY_IMAGE");
296     oldOverlayImage.destroy ();
297     //create a new overlay image
298     Gtk.Overlay lOverlayImage = new Gtk.Overlay ();
299     lOverlayImage.add (book.getBookWidget ("PLACEHOLDER_COVER_IMAGE"));
300     lOverlayImage.add_overlay (book.getBookWidget ("SELECTION_BADGE_IMAGE"));
301     lOverlayImage.add_overlay (book.getBookWidget ("SELECTED_BADGE_IMAGE"));
302     lOverlayImage.add_overlay (book.getBookWidget ("COVER_IMAGE"));
303     lOverlayImage.add_overlay (book.getBookWidget ("TITLE_TEXT_LABEL"));
304     book.setBookWidget ("BOOK_OVERLAY_IMAGE", lOverlayImage);
305     //associate the eventbox with the new overlay image
306     Gtk.EventBox aEventBox = (Gtk.EventBox) book.getBookWidget ("BOOK_EVENTBOX");
307     aEventBox.add (lOverlayImage);
308     book.setBookWidget ("BOOK_EVENTBOX", aEventBox);
309     //update the libraryview map with the book object
310     BookwormApp.Bookworm.libraryViewMap.set (book.getBookLocation (), book);
311     debug ("[END] [FUNCTION:replaceCoverImageOnBook]");
312 }
313
314 public static void updateListViewForSelection (owned BookwormApp.Book? lBook) {
315     debug ("[START] [FUNCTION:updateListViewForSelection] " +
316         "Updating List View Selection Badges for mode:" +
317         BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE);
318     Gtk.TreeModelForeachFunc print_row = (model, path, iter) => {
319         GLib.Value bookLocationAtRow;
320         BookwormApp.AppWindow.library_table_liststore.get_value (iter, 7, out bookLocation
AtRow);
321         BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap.get ((string) bookLoc
ationAtRow);
322         if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_
UI_STATES[5]) {
323             BookwormApp.AppWindow.library_table_liststore.set_value (iter, 0, BookwormApp.
Bookworm.image_selection_transparent_small);
324             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
Y_IMAGE");
387             //Align the selection badges to the right to make visible but the selected
badges should be centered to keep hidden
388             book.getBookWidget ("SELECTION_BADGE_IMAGE").set_halign (Align.START);
389             book.getBookWidget ("SELECTED_BADGE_IMAGE").set_halign (Align.CENTER);
390             //set the order of the widgets to put the selection badge on top
391             aOverlayImage.reorder_overlay (book.getBookWidget ("SELECTED_BADGE_IMAGE")
, 1);
392             aOverlayImage.reorder_overlay (book.getBookWidget ("COVER_IMAGE"), 2);
393             aOverlayImage.reorder_overlay (book.getBookWidget ("TITLE_TEXT_LABEL"), 3)
;
394             aOverlayImage.reorder_overlay (book.getBookWidget ("SELECTION_BADGE_IMAGE"
), 4);
395         }
396         temp_libraryViewMap.set (book.getBookLocation (), book);
397     }
398     //Iterate over all books and make the selection flag for each book as false
399     //This is to cover the scenario when a book was selected and the selection mode wa
s changed without deleting the book
400     foreach (BookwormApp.Book aBook in temp_libraryViewMap.values) {
401         aBook.setIsBookSelected (false);
402         BookwormApp.Bookworm.libraryViewMap.set (aBook.getBookLocation (), aBook);
403     }
404 }
405 if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
TATES[3]) {
406     debug ("Updating Library View for Selection Badges BOOKWORM_UI_STATES[3]");
407     if (lBook != null) {
408         if (BookwormApp.AppWindow.library_grid_scroll.get_visible ()) {
409             Gtk.Overlay aOverlayImage = (Gtk.Overlay) lBook.getBookWidget ("BOOK_OVERL
AY_IMAGE");
410             if (!lBook.getIsBookSelected ()) { //Do work for selecting this book
411                 //Align the selected badges to the right to make visible but keep the
selection badge centered to keep hidden
412                 lBook.getBookWidget ("SELECTED_BADGE_IMAGE").set_halign (Align.START);
413                 lBook.getBookWidget ("SELECTION_BADGE_IMAGE").set_halign (Align.CENTER
);
414                 //set the order of the widgets to put the selected badge on top
415                 aOverlayImage.reorder_overlay (lBook.getBookWidget ("SELECTION_BADGE_I
MAGE"), 1);
416                 aOverlayImage.reorder_overlay (lBook.getBookWidget ("COVER_IMAGE"), 2)
;
417                 aOverlayImage.reorder_overlay (lBook.getBookWidget ("TITLE_TEXT_LABEL"
), 3);
418                 aOverlayImage.reorder_overlay (lBook.getBookWidget ("SELECTED_BADGE_IM
AGE"), 4);
419                 lBook.setIsBookSelected (true);
420                 BookwormApp.AppWindow.controlDeletionButton (true);
421             } else { //Do work for de-selecting this book
422                 //set the order of the widgets to put the selection badge on top
423                 //Align the selection badges to the right to make visible but keep the
selected badges centered to keep hidden
424                 lBook.getBookWidget ("SELECTION_BADGE_IMAGE").set_halign (Align.START)
;
425                 lBook.getBookWidget ("SELECTED_BADGE_IMAGE").set_halign (Align.CENTER)
;
426                 aOverlayImage.reorder_overlay (lBook.getBookWidget ("SELECTED_BADGE_IM
AGE"), 1);
427                 aOverlayImage.reorder_overlay (lBook.getBookWidget ("COVER_IMAGE"), 2)
;
428                 aOverlayImage.reorder_overlay (lBook.getBookWidget ("TITLE_TEXT_LABEL"
), 3);
429                 aOverlayImage.reorder_overlay (lBook.getBookWidget ("SELECTION_BADGE_I
MAGE"), 4);
430                 lBook.setIsBookSelected (false);
431                 BookwormApp.AppWindow.controlDeletionButton (false);
432             }
433         }
434         //update the book into the Library view HashMap
435         BookwormApp.Bookworm.libraryViewMap.set (lBook.getBookLocation (), lBook);
436     }
437 }
438 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
View Model
445         foreach (BookwormApp.Book book in BookwormApp.Bookworm.libraryViewMap.values) {
446             //check if the book selection flag to true and add it to removal list
447             if (book.getIsBookSelected ()) {
448                 //hold the books to be deleted in a list
449                 listOfBooksToBeRemoved.add (book.getBookLocation ());
450                 Gtk.EventBox lEventBox = (Gtk.EventBox) book.getBookWidget ("BOOK_EVENTBOX");
451                 //destroy the EventBox parent widget - this removes the book from the library
grid
452                 lEventBox.get_parent ().destroy ();
453                 //destroy the EventBox widget
454                 lEventBox.destroy ();
455                 //remove the cover image if it exists (ignore default covers)
456                 if (book.getBookCoverLocation ().index_of (
457                     BookwormApp.Constants.DEFAULT_COVER_IMAGE_LOCATION.replace ("-cover-N.svg"
, "")) == -1)
458                 {
459                     BookwormApp.Utils.execute_sync_command ("rm \"\" + book.getBookCoverLocatio
n () + \"\"");
460                 }
461                 //update the onloadBookList - this is to enable re-adding the book within
the same session
462                 BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr
463                     .assign (BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr.str
464                         .replace (book.getBookLocation (), ""));
465                 BookwormApp.Library.listOfBooksInLibraryOnLoad.remove (book);
466             }
467         }
468         if (listOfBooksToBeRemoved.size > 0) {
469             //loop through the rows in the treeview and remove the selected books
470             ArrayList<Gtk.TreeIter?> listOfIterstoBeRemoved = new ArrayList<Gtk.TreeIter?> ();
471             Gtk.TreeModelForeachFunc print_row = (model, path, iter) => {
472                 GLib.Value bookLocationAtRow;
473                 BookwormApp.AppWindow.library_table_liststore.get_value (iter, 7, out bookLoca
tionAtRow);
474                 if ((string) bookLocationAtRow in listOfBooksToBeRemoved) {
475                     listOfIterstoBeRemoved.add (iter);
476                 }
477                 return false;
478             };
479             BookwormApp.AppWindow.library_table_liststore.foreach (print_row);
480             foreach (Gtk.TreeIter iterToBeRemoved in listOfIterstoBeRemoved) {
481                 //remove item for list store - vala_36 compatibility wrapper
482                 #if VALA_0_36
483                     BookwormApp.AppWindow.library_table_liststore.remove (ref iterToBeRemoved)
;
484                 #else
485                     BookwormApp.AppWindow.library_table_liststore.remove (iterToBeRemoved);
486                 #endif
487             }
488         }
489         //loop through the removed books and remove them from the Library View Hashmap, local
cache and Database
490         foreach (string bookLocation in listOfBooksToBeRemoved) {
491             BookwormApp.DB.removeBookFromDB (BookwormApp.Bookworm.libraryViewMap.get (bookLoca
tion));
492             BookwormApp.Bookworm.libraryViewMap.unset (bookLocation);
493         }
494         //Set to normal grid view if the current view is in any of the Grid View State
495         if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
TATES[0] ||
496             BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
TATES[2] ||
497             BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
TATES[3])
498         {
499             BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_UI_ST
ATES[0];
500             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
    TATES[5] ||
504         BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
    TATES[6] ||
505         BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
    TATES[7])
506     {
507         BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_UI_ST
    ATES[5];
508         BookwormApp.Library.updateListViewForSelection (null);
509     }
510     BookwormApp.Bookworm.toggleUIState ();
511     debug ("[END] [FUNCTION:removeSelectedBooksFromLibrary]");
512 }
513
514 public static async void updateLibraryViewFromDB () {
515     debug ("[START] [FUNCTION:updateLibraryViewFromDB]");
516     foreach (BookwormApp.Book book in listOfBooksInLibraryOnLoad) {
517         //add the book to the UI - both grid and list view
518         BookwormApp.Library.updateLibraryView (book);
519         Idle.add (updateLibraryViewFromDB.callback);
520         yield;
521     }
522     debug ("[END] [FUNCTION:updateLibraryViewFromDB]");
523 }
524
525 public static async void addBooksToLibrary () {
526     debug ("[START] [FUNCTION:addBooksToLibrary]");
527     debug ("books to be added=" + BookwormApp.Bookworm.pathsOfBooksToBeAdded.length.to_str
    ing ());
528     double progress = 0d;
529     //loop through the command line and add books to library
530     foreach (string pathToSelectedBook in BookwormApp.Bookworm.pathsOfBooksToBeAdded) {
531         debug ("Attempting to add book from path:" + pathToSelectedBook);
532         //Set async callback only if multiple books are being added
533         //If only one book is being added, complete parsing and adding the book,
534         //so that it will be added to the BookwormApp.Bookworm.libraryViewMap and opened o
    n the
535         //BookwormApp.contentHandler.performStartUpActions method
536         if (BookwormApp.Bookworm.pathsOfBooksToBeAdded.length > 2) {
537             Idle.add (addBooksToLibrary.callback);
538         }
539         BookwormApp.Bookworm.noOfBooksAddedFromCommand++;
540         if (BookwormApp.Constants.bookworm_id != pathToSelectedBook.strip ()) {
541             //set progress for the UI Book addition progress bar
542             progress = (((double) (BookwormApp.Bookworm.noOfBooksAddedFromCommand))/((doub
    le) (BookwormApp.Bookworm.pathsOfBooksToBeAdded.length)));
543             BookwormApp.AppWindow.bookAdditionBar.set_text (_,("Adding ") +
544                 ((int) (progress * 100)).to_string () + "% : " + File.new_for_path (pathTo
    SelectedBook).get_basename ());
545             BookwormApp.AppWindow.bookAdditionBar.set_fraction (progress);
546         }
547         //Return control back for any further actions only if multiple books are being add
    ed
548         //If only one book is being added, complete parsing and adding the book,
549         //so that it will be added to the BookwormApp.Bookworm.libraryViewMap and opened o
    n the
550         //BookwormApp.contentHandler.performStartUpActions method
551         if (BookwormApp.Bookworm.pathsOfBooksToBeAdded.length > 2) {
552             yield;
553         }
554         //ignore the first command which is the application name
555         if (BookwormApp.Constants.bookworm_id != pathToSelectedBook.strip ()) {
556             //check if book already exists in the library
557             if (BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr.str.index_of (pathToSe
    lectedBook.strip ()) != -1) {
558                 debug ("Book already exists in library..." + BookwormApp.Bookworm.pathsOfB
    ooksInLibraryOnLoadStr.str);
559                 //Enable the flag which will scroll the page to the last read position
560                 BookwormApp.Bookworm.isPageScrollRequired = true;
561                 //set the name of the book being currently read
562                 BookwormApp.Bookworm.locationOfEBookCurrentlyRead = pathToSelectedBook.str
    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 addBookToLibra
ry function
568             //however the libraryViewMap will only be fully populated when all books a
re added to it
569             addBookToLibrary (aBookBeingAdded);
570             //update the onloadBookList - this is to prevent re-adding the book within
the same session
571             BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr.append (aBookBeingAdde
d.getBookLocation ());
572             BookwormApp.Library.listOfBooksInLibraryOnLoad.add (aBookBeingAdded);
573         }
574     }
575 }
576 //Hide the progress bar on completion of adding books
577 BookwormApp.AppWindow.bookAdditionBar.hide ();
578 BookwormApp.Bookworm.isBookBeingAddedToLibrary = false;
579 BookwormApp.Bookworm.noOfBooksAddedFromCommand = 0;
580 debug ("[END] [FUNCTION:addBooksToLibrary]");
581 }
582
583 public static void addBookToLibrary (owned BookwormApp.Book aBook) {
584     debug ("[START] [FUNCTION:addBookToLibrary] book.location=" + aBook.getBookLocation ()
);
585     //check if the selected eBook exists
586     string eBookLocation = aBook.getBookLocation ();
587     File eBookFile = File.new_for_path (eBookLocation);
588     if (eBookFile.query_exists () && eBookFile.query_file_type (0) != FileType.DIRECTORY)
    {
589         //insert book details to database and fetch the ID
590         int bookID = BookwormApp.DB.addBookToDataBase (aBook);
591         aBook.setBookId (bookID);
592         /*Other than location, nothing is inserted into the DB for the book at this time.
593         Mark book as opened in the session so that details for book are updated
594         into DB when the application is closed - eBook parsing happens after the initial i
nsert
595         */
596         aBook.setBookLastModificationDate ((new DateTime.now_utc ().to_unix ().to_string
()));
597         aBook.setWasBookOpened (true);
598         //parse eBook to populate cache and book meta data
599         aBook = BookwormApp.Bookworm.genericParser (aBook);
600         if (!aBook.getIsBookParsed ()) {
601             BookwormApp.DB.removeBookFromDB (aBook);
602             BookwormApp.AppWindow.showInfoBar (aBook, MessageType.WARNING);
603         } else {
604             //add eBook cover image to library view
605             BookwormApp.Library.updateLibraryView (aBook);
606             //Set to normal grid view if the current view is in any of the Grid View State
607             if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
ORM_UI_STATES[0] ||
608                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
ORM_UI_STATES[2] ||
609                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
ORM_UI_STATES[3])
        {
610             BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWO
RM_UI_STATES[0];
611             BookwormApp.Library.updateGridViewForSelection (null);
612         }
613         //Set to normal list view if the current view is in any of the List View State
614         if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
ORM_UI_STATES[5] ||
615             BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
ORM_UI_STATES[6] ||
616             BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
ORM_UI_STATES[7])
        {
617             BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWO
RM_UI_STATES[5];
618             BookwormApp.Library.updateListViewForSelection (null);
619         }
620     }
621 }

```

```

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_VERSION;
26
27     N;
28     public const string BOOKMETADATA_TABLE_BASE_NAME = "BOOK_METADATA_TABLE";
29     public const string BOOKMETADATA_TABLE_VERSION = "1"; //Only integers allowed
30     public const string BOOKMETADATA_TABLE_NAME = BOOKMETADATA_TABLE_BASE_NAME + BOOKMETADATA_
31     TABLE_VERSION;
32     public const string VERSION_TABLE_BASE_NAME = "VERSION_TABLE";
33     public const string VERSION_TABLE_VERSION = "1"; //Only integers allowed
34     private static Sqlite.Database bookwormDB;
35     private static string errmsg;
36     private static string queryString;
37     private static int executionStatus;
38
39     public static bool initializeBookWormDB (string bookworm_config_path) {
40         info ("[START] [FUNCTION:initializeBookWormDB] bookworm_config_path=" + bookworm_conf
41         g_path);
42         Statement stmt;
43         debug ("Checking BookWorm DB or creating it if the DB does not exist...");
44         int dbOpenStatus = Database.open_v2 (
45             bookworm_config_path + "/bookworm.db", out bookwormDB, Sqlite.OPEN_READWRITE | Sql
46             ite.OPEN_CREATE);
47         if (dbOpenStatus != Sqlite.OK) {
48             warning ("Error in opening database[" + bookworm_config_path + "/bookworm.db" + "]
49             : %d: %s\n", bookwormDB.errcode (), bookwormDB.errmsg ());
50             return false;
51         } else {
52             debug ("Successfully checked/created DB for Bookworm.....");
53         }
54
55         debug ("Creating latest version for Library table if it does not exists");
56         queryString = "CREATE TABLE IF NOT EXISTS " + BOOKWORM_TABLE_NAME + " (" +
57             "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
58             "BOOK_LOCATION TEXT NOT NULL DEFAULT '', " +
59             "BOOK_TITLE TEXT NOT NULL DEFAULT '', " +
60             "BOOK_AUTHOR TEXT NOT NULL DEFAULT '', " +
61             "BOOK_COVER_IMAGE_LOCATION TEXT NOT NULL DEFAULT '', " +
62             "IS_BOOK_COVER_IMAGE_PRESENT TEXT NOT NULL DEFAULT '', " +
63             "BOOK_PUBLISH_DATE TEXT NOT NULL DEFAULT '', " +
64             "BOOK_TOTAL_NUMBER_OF_PAGES TEXT NOT NULL DEFAULT '', " +
65             "BOOK_LAST_READ_PAGE_NUMBER TEXT NOT NULL DEFAULT '', " +
66             "BOOK_TOTAL_PAGES TEXT NOT NULL DEFAULT '', " + //Added in table v6
67             "TAGS TEXT NOT NULL DEFAULT '', " + //Added in table v3
68             "ANNOTATION_TAGS TEXT NOT NULL DEFAULT '', " + //Added in table v7
69             "RATINGS TEXT NOT NULL DEFAULT '', " + //Added in table v3
70             "CONTENT_EXTRACTION_LOCATION TEXT NOT NULL DEFAULT '', " + //Added in table v4
71             "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 remaning 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
    ITION, " +
132         "creation_date, modification_date) SELECT id, BOOK_TOC_DATA, BOOKMARKS
    , " +
133         "CONTENT_DATA_LIST, BOOK_LAST_SCROLL_POSITION, creation_date, " +
134         "modification_date FROM BOOK_LIBRARY_TABLE5";
135     executionStatus = bookwormDB.exec (queryString, null, out errmsg);
136     if (executionStatus != SQLite.OK) {
137         report_query_execution_error (queryString, bookwormDB.errcode (), book
wormDB.errmsg ());
138     } else {
139         debug ("Successfully migrated " + bookwormDB.changes ().to_string () +
140             " rows from BOOK_LIBRARY_TABLE5 into" + BOOKMETADATA_TABLE_NAME);
141         //drop the old table
142         queryString = "DROP TABLE IF EXISTS BOOK_LIBRARY_TABLE5";
143         executionStatus = bookwormDB.exec (queryString, null, out errmsg);
144         if (executionStatus != SQLite.OK) {
145             report_query_execution_error (queryString, bookwormDB.errcode (),
bookwormDB.errmsg ());
146         } else {
147             debug ("Successfully dropped old table LIBRARY_TABLE5");
148         }
149     }
150 }
151 }
152 //VERSION_TABLE : Drop table
153 if (old_table_name == "VERSION_TABLE") {
154     //drop the old table
155     queryString = "DROP TABLE IF EXISTS VERSION_TABLE";
156     executionStatus = bookwormDB.exec (queryString, null, out errmsg);
157     if (executionStatus != SQLite.OK) {
158         report_query_execution_error (queryString, bookwormDB.errcode (), bookworm
DB.errmsg ());
159     } else {
160         debug ("Successfully dropped old table VERSION_TABLE");
161     }
162 }
163 }
164 //All DB loading operations completed
165 info ("[END] [FUNCTION:initializeBookWormDB]");
166 return true;
167 }
168
169 public static ArrayList<BookwormApp.Book> getBooksFromDB (string criteria, string mode) {
170     info ("[START] [FUNCTION:getBooksFromDB] Querying with mode[" + mode + "] and criteria[
" + criteria + "]);
171     ArrayList<BookwormApp.Book> listOfBooks = new ArrayList<BookwormApp.Book> ();
172     Statement stmt;
173     string last_modification_date = "-1";
174     queryString = "SELECT id, BOOK_LOCATION, BOOK_TITLE, BOOK_AUTHOR, BOOK_COVER_IMAGE_LOC
ATION, " +
175         "IS_BOOK_COVER_IMAGE_PRESENT, BOOK_LAST_READ_PAGE_NUMBER, BOOK_PUBLISH_DATE, TAGS,
" +
176         "ANNOTATION_TAGS, RATINGS, CONTENT_EXTRACTION_LOCATION, BOOK_TOTAL_PAGES, creation
_date, " +
177         "modification_date FROM " + BOOKWORM_TABLE_NAME;
178     if (criteria == "" && mode == "PAGINATED_SEARCH") {
179         //initial query on app load without pagination criteria
180         queryString = queryString + " ORDER BY modification_date DESC LIMIT " + BookwormAp
p.Bookworm.no_of_books_per_page;
181         debug ("Paginated Query with last_modification_date[" + criteria + "]: " + queryStr
ing);
182     } else if (mode == "LIBRARY_SEARCH") {
183         //query db for matching search criteria on all book meta data
184         queryString = queryString + " WHERE " +
185             " BOOK_TITLE LIKE '%" + criteria + "%' OR " +
186             " BOOK_LOCATION LIKE '%" + criteria + "%' OR " +
187             " BOOK_AUTHOR LIKE '%" + criteria + "%' OR " +
188             " TAGS LIKE '%" + criteria + "%' OR " +
189             " ANNOTATION_TAGS LIKE '%" + criteria + "%'";
190         debug ("Library Search Query with criteria[" + criteria + "]: " + queryString);
191     } else if (mode == "PAGINATED_SEARCH") {
192         //query for pagination criteria
193         queryString = queryString + " where modification_date < CAST ('" + criteria + "' A
S INT) " +

```

```

194         "ORDER BY modification_date DESC LIMIT " + BookwormApp.Bookworm.no_of_books_pe
r_page;
195         debug ("Paginated Query with last_modification_date[" + criteria + "]: " + queryStr
ing);
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.errms
g ());
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:fal
se);
209             aBook.setBookPageNumber (int.parse (stmt.column_text (6)));
210             aBook.setBookPublishDate (stmt.column_text (7));
211             aBook.setBookTags (stmt.column_text (8));
212             aBook.setAnnotationTags (stmt.column_text (9));
213             aBook.setBookRating (int.parse (stmt.column_text (10)));
214             aBook.setBookExtractionLocation (stmt.column_text (11));
215             aBook.setBookTotalPages (int.parse (stmt.column_text (12)));
216             aBook.setBookCreationDate (stmt.column_text (13));
217             aBook.setBookLastModificationDate (stmt.column_text (14));
218             debug ("Book details fetched from DB: id=" + stmt.column_int (0).to_string ()
+
219                 ",BOOK_LOCATION=" + stmt.column_text (1) +
220                 ",BOOK_TITLE=" + stmt.column_text (2) +
221                 ",BOOK_AUTHOR=" + stmt.column_text (3) +
222                 ",BOOK_COVER_IMAGE_LOCATION=" + stmt.column_text (4) +
223                 ",IS_BOOK_COVER_IMAGE_PRESENT=" + stmt.column_text (5) +
224                 ",BOOK_LAST_READ_PAGE_NUMBER=" + stmt.column_text (6) +
225                 ",BOOK_PUBLISH_DATE=" + stmt.column_text (7) +
226                 ",TAGS=" + stmt.column_text (8) +
227                 ",ANNOTATION_TAGS=" + stmt.column_text (9) +
228                 ",RATINGS=" + stmt.column_text (10) +
229                 ",CONTENT_EXTRACTION_LOCATION=" + stmt.column_text (11) +
230                 ",BOOK_TOTAL_PAGES=" + stmt.column_text (12) +
231                 ",creation_date=" + stmt.column_text (13) +
232                 ",modification_date=" + stmt.column_text (14));
233             //add book details to list
234             listOfBooks.add (aBook);
235             //build the string of book paths in the library
236             BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr.append (aBook.getBookLocat
ion ());
237             //capture the last_modification_date of the book
238             last_modification_date = aBook.getBookLastModificationDate ();
239         }
240         if (mode == "PAGINATED_SEARCH") {
241             //Only capture the last modification date if the results are equal to the page
size
242             if (listOfBooks.size == int.parse (BookwormApp.Bookworm.no_of_books_per_page))
{
243                 //set the last book's modification date for pagination
244                 BookwormApp.Bookworm.paginationlist.add (last_modification_date);
245             } else {
246                 BookwormApp.Bookworm.paginationlist.add ("-1");
247             }
248         }
249         stmt.reset ();
250     }
251     info ("[END] [FUNCTION:getBooksFromDB] no. of books fetched [" + listOfBooks.size.to_s
tring () + "], last_modification_date of books fetched[" + last_modification_date + "]);
252     return listOfBooks;
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
    ATION, " +
260             "IS_BOOK_COVER_IMAGE_PRESENT, BOOK_LAST_READ_PAGE_NUMBER, BOOK_PUBLISH_DATE, TAGS,
    " +
261             "ANNOTATION_TAGS, RATINGS, CONTENT_EXTRACTION_LOCATION, BOOK_TOTAL_PAGES, creation
    _date, " +
262             "modification_date FROM " + BOOKWORM_TABLE_NAME;
263         //query db for matching book location
264         queryString = queryString + " WHERE " + " BOOK_LOCATION LIKE '" + book_location + "'";
265         debug ("Library Search Query with criteria[" + book_location + "]: " + queryString);
266         executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
267         if (executionStatus != SQLite.OK) {
268             report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errms
    g ());
269         } else {
270             while (stmt.step () == ROW) {
271                 aBook.setBookId (stmt.column_int (0));
272                 aBook.setBookLocation (stmt.column_text (1));
273                 aBook.setBookTitle (stmt.column_text (2));
274                 aBook.setBookAuthor (stmt.column_text (3));
275                 aBook.setBookCoverLocation (stmt.column_text (4));
276                 aBook.setIsBookCoverImagePresent ( (stmt.column_text (5) == "true") ? true : f
    else);
277                 aBook.setBookPageNumber (int.parse (stmt.column_text (6)));
278                 aBook.setBookPublishDate (stmt.column_text (7));
279                 aBook.setBookTags (stmt.column_text (8));
280                 aBook.setAnnotationTags (stmt.column_text (9));
281                 aBook.setBookRating (int.parse (stmt.column_text (10)));
282                 aBook.setBookExtractionLocation (stmt.column_text (11));
283                 aBook.setBookTotalPages (int.parse (stmt.column_text (12)));
284                 aBook.setBookCreationDate (stmt.column_text (13));
285                 aBook.setBookLastModificationDate (stmt.column_text (14));
286                 debug ("Book details fetched from DB: id=" + stmt.column_int (0).to_string ()
    +
287                     ",BOOK_LOCATION=" + stmt.column_text (1) +
288                     ",BOOK_TITLE=" + stmt.column_text (2) +
289                     ",BOOK_AUTHOR=" + stmt.column_text (3) +
290                     ",BOOK_COVER_IMAGE_LOCATION=" + stmt.column_text (4) +
291                     ",IS_BOOK_COVER_IMAGE_PRESENT=" + stmt.column_text (5) +
292                     ",BOOK_LAST_READ_PAGE_NUMBER=" + stmt.column_text (6) +
293                     ",BOOK_PUBLISH_DATE=" + stmt.column_text (7) +
294                     ",TAGS=" + stmt.column_text (8) +
295                     ",ANNOTATION_TAGS=" + stmt.column_text (9) +
296                     ",RATINGS=" + stmt.column_text (10) +
297                     ",CONTENT_EXTRACTION_LOCATION=" + stmt.column_text (11) +
298                     ",BOOK_TOTAL_PAGES=" + stmt.column_text (12) +
299                     ",creation_date=" + stmt.column_text (13) +
300                     ",modification_date=" + stmt.column_text (14));
301             }
302         }
303         stmt.reset ();
304         info ("[END] [FUNCTION:getBookFromDB] Book fetched [" + aBook.getBookLocation () + "]
    ");
305         return aBook;
306     }
307
308     public static BookwormApp.Book getBookMetaDataFromDB (owned BookwormApp.Book aBook) {
309         info ("[START] [FUNCTION:getBookMetaDataFromDB] book.location=" + aBook.getBookLocatio
    n ());
310         Statement stmt;
311         queryString = "SELECT BOOK_TOC_DATA, BOOKMARKS, CONTENT_DATA_LIST, BOOK_LAST_SCROLL_PO
    SITION, " +
312             "BOOK_ANNOTATIONS FROM " + BOOKMETADATA_TABLE_NAME + " WHERE id = ?";
313         executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
314         if (executionStatus != SQLite.OK) {
315             report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errms
    g ());
316         }
317         stmt.bind_int (1, aBook.getBookId ());
318         while (stmt.step () == ROW) {
319             aBook = BookwormApp.Utils.convertStringToTOC (aBook, stmt.column_text (0));
320             aBook.setBookmark (-10, stmt.column_text (1)); //-10 is a flag to set the bookmark
    string into the object
321             aBook = BookwormApp.Utils.convertStringToContentList (aBook, stmt.column_text (2))
    ;

```

```

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

```

```

386         debug ("Removed this book from library table:" + aBook.getBookTitle () + "[" + aBook.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     };
391     if (executionStatus != Sqlite.OK) {
392         report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errmsg ());
393         return false;
394     } else {
395         stmt.bind_int (1, aBook.getBookId ());
396         stmt.step ();
397         stmt.reset ();
398     }
399     info ("[END] [FUNCTION:removeBookFromDB] book.location=" + aBook.getBookLocation ());
400     return true;
401 }
402
403 public static bool updateBookToDataBase (BookwormApp.Book aBook) {
404     info ("[START] [FUNCTION:updateBookToDataBase] Updating book to DB for the following details:" + 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.errmsg ());
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.getBookTitle () + "[" + aBook.getBookId ().to_string () + "]");
431     //Attempt to insert book meta data
432     queryString = "INSERT OR IGNORE INTO " + BOOKMETADATA_TABLE_NAME + " (BOOK_TOC_DATA, BOOKMARKS, " +
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.errmsg ());
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 = getOPFFileLocation (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:" +
104                 extractionLocation + "/" + BookwormApp.Constants.EPUB_MIME_SPECIFICATION_FILEN
AME);
105             return false;
106         }
107         debug ("Mime Contents found in file :" + extractionLocation + "/" +
108             BookwormApp.Constants.EPUB_MIME_SPECIFICATION_FILENAME + " is:" + ePubMimeContents
);
109         if (ePubMimeContents.strip () != BookwormApp.Constants.EPUB_MIME_SPECIFICATION_CONTENT
) {
110             debug ("Mime Contents in file :" + extractionLocation + "/" +
111                 BookwormApp.Constants.EPUB_MIME_SPECIFICATION_FILENAME + " is not :" +
112                 BookwormApp.Constants.EPUB_MIME_SPECIFICATION_CONTENT + ". No further parsing
will be done.");
113             return false;
114         } else {
115             //mime content is as expected
116             ePubFormat = true;
117         }
118         info ("[END] [FUNCTION:isEPubFormat] ePubFormat=" + ePubFormat.to_string ());
119         return ePubFormat;
120     }
121
122     public static string getOPFFileLocation (string extractionLocation) {
123         info ("[START] [FUNCTION:getOPFFileLocation] extractionLocation=" + extractionLocation
);
124         string locationOfOPFFile = "false";
125         //Form the path to the META-INF/container.xml file
126         string pathToXMLFile = extractionLocation + "/" + BookwormApp.Constants.EPUB_META_INF_
FILENAME;
127         //Parse META-INF/container.xml file to locate the path to the OPF file
128         ArrayList<XMLData> inputDataList = new ArrayList<XMLData> ();
129         inputDataList.add (new XMLData () {
130             containerTagName = "rootfiles",
131             inputTagName = "rootfile",
132             inputAttributeName = "full-path"
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                 tivePath);
201                 string ncxFilePath = (
202                     BookwormApp.Utils.getFullPathFromFilename (
203                         aBook.getBaseLocationOfContents (), NCXFileRelativePath.strip ()).str
204                     ip ());
205                 if ("true" == BookwormApp.Utils.fileOperations ("EXISTS", "", ncxFilePath, ""))
206                 ) {
207                     debug ("Successfully determined NCX File Path as:" + ncxFilePath);
208                     //Parse NCX xml file to read the ToC data (id, href, media-type)

```

```

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).extractedTagValues.size; count++) {
219                 HashMap<string,string> TOCMapItem = new HashMap<string,string> ();
220                 string tocLocation = ncxDataExtractedList.get (1).extractedTagAttributes.get (count);
221                 //Handle the links with anchor elements
222                 string anchorValue = "";
223                 if (tocLocation.index_of ("#") != -1 ) {
224                     anchorValue = tocLocation.slice (tocLocation.index_of ("#"), tocLocation.length);
225                     tocLocation = tocLocation.slice (0, tocLocation.index_of ("#"));
226                 }
227                 tocLocation = BookwormApp.Utils.getFullPathFromFilename (aBook.getBaseLocationOfContents (), tocLocation);
228                 TOCMapItem.set (tocLocation + anchorValue, ncxDataExtractedList.get (0).extractedTagValues.get (count));
229                 aBook.setTOC (TOCMapItem);
230                 debug ("Extracted ToC Chapter Name:" + ncxDataExtractedList.get (0).extractedTagValues.get (count) + " at location:" + tocLocation + anchorValue);
231             }
232         }
233     }
234 }
235 }
236 }
237
238 // Create the content list - clear the content list of any previous items
239 aBook.clearBookContentList ();
240 //loop over all idref attributes in spine data
241 foreach (string spineIDREF in opfItemsList[3].extractedTagAttributes) {
242     //check if the SPINE IDREF exists in the MANIFEST Attributes
243     if (opfItemsList[0].extractedTagAttributes.contains (spineIDREF)) {
244         int positionOfIDREF = opfItemsList[0].extractedTagAttributes.index_of (spineIDREF);
245         //extract the HREF from MANIFEST corresponding to the SPINE IDREF
246         string locationOfContentData = opfItemsList[1].extractedTagAttributes.get (positionOfIDREF);
247         aBook.setBookContentList (aBook.getBaseLocationOfContents () + locationOfContentData);
248         debug ("Book content data :" + aBook.getBaseLocationOfContents () + locationOfContentData);
249     }
250 }
251 info ("[END] [FUNCTION:determineToC]");
252 return aBook;
253 }
254
255 public static BookwormApp.Book setCoverImage (owned BookwormApp.Book aBook, string locationOfOPFFFile) {
256     info ("[START] [FUNCTION:setCoverImage] book.location=" + aBook.getBookLocation () + " , locationOfOPFFFile=" + locationOfOPFFFile);
257     string bookCoverLocation = "";
258     //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 (locationOfOPFFFile, inputDataList);
285     int count = 0;
286     //epub3.1 : Check for a MANIFEST item with "properties" attribute containng the word "
cover-image"
287     foreach (string properties in opfItemsList[3].extractedTagAttributes) {
288         if (properties.contains ("cover-image")) {
289             //Get media type for the cover items
290             string coverMediaType = opfItemsList[1].extractedTagAttributes.get (count);
291             //get cover location if media type matches "image"
292             if (coverMediaType.index_of ("image") != -1) {
293                 bookCoverLocation = opfItemsList[2].extractedTagAttributes.get (count);
294                 bookCoverLocation = aBook.getBaseLocationOfContents () + bookCoverLocation
;
295                 break;
296             }
297         }
298         count++;
299     }
300     //If cover could not be located in properties="cover-image" :
301     //Check for a MANIFEST item with "id" attribute containng the word "cover"
302     if (bookCoverLocation.length < 1 &&
303         "true" == BookwormApp.Utills.fileOperations ("EXISTS", "", bookCoverLocation, ""))
304     {
305         count = 0;
306         foreach (string id in opfItemsList[0].extractedTagAttributes) {
307             if (id.contains ("cover")) {
308                 //Get media type for the cover items
309                 string coverMediaType = opfItemsList[1].extractedTagAttributes.get (count)
;
310                 //get cover location if media type matches "image"
311                 if (coverMediaType.index_of ("image") != -1) {
312                     bookCoverLocation = opfItemsList[2].extractedTagAttributes.get (count)
;
313                     bookCoverLocation = aBook.getBaseLocationOfContents () + bookCoverLoca
tion;
314                     break;
315                 }
316             }
317             count++;
318         }
319     }
320     //check if cover was still not found and assign flag for default cover to be used
321     if (bookCoverLocation.length < 1 &&
322         "true" == BookwormApp.Utills.fileOperations ("EXISTS", "", bookCoverLocation, ""))
323     {
324         aBook.setIsBookCoverImagePresent (false);
325         debug ("Cover image not found for book located at:" + aBook.getBookExtractionLocat
ion ());
326     } else {
327         //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> inputDataList = new ArrayList<XMLData> ();
339         inputDataList.add (new XMLData () {
340             containerTagName = "",
341             inputTagName = "title",
342             inputAttributeName = ""
343         });
344         inputDataList.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, inputDataList);
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 direction) {
26         debug ("[START] [FUNCTION:renderPage] book.location=" + aBook.getBookLocation () + ",
27             direction=" + direction);
28         int currentContentLocation = aBook.getBookPageNumber ();
29         bool shouldReloadPage = true;
30         //set page number based on direction of navigation
31         switch (direction) {
32             case "FORWARD"://This is for moving the book forward
33                 if (aBook.getIfPageForward ()) {
34                     currentContentLocation++;
35                     aBook.setBookPageNumber (currentContentLocation);
36                 }
37                 break;
38             case "BACKWARD"://This is for moving the book backwards
39                 if (aBook.getIfPageBackward ()) {
40                     currentContentLocation--;
41                     aBook.setBookPageNumber (currentContentLocation);
42                 }
43                 break;
44             case "SEARCH"://Load the page and scroll to the search text
45                 break;
46             default://This is for opening the current page of the book
47                 //Do not change the page number
48                 break;
49         }
50         string bookContent = contentHandler.provideContent (aBook,currentContentLocation, direction);
51         //render the content on webview
52         BookwormApp.AppWindow.aWebView.load_html (bookContent, BookwormApp.Constants.PREFIX_FOLDER_FILE_URL);
53         //set the bookmark icon on the header
54         handleBookMark ("DISPLAY");
55         //set the navigation controls
56         aBook = controlNavigation (aBook);
57         //set the current value of the page slider
58         BookwormApp.AppWindow.pageAdjustment.set_value (currentContentLocation + 1);
59         debug ("[END] [FUNCTION:renderPage]");
60         return aBook;
61     }
62
63     public static string provideContent (owned BookwormApp.Book aBook, int contentLocation, string mode) {
64         debug ("[START] [FUNCTION:provideContent] book.location=" + aBook.getBookLocation () +
65             ", contentLocation=" + contentLocation.to_string () + ", mode=" + mode);
66         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.Utills.fileOperations ("EXISTS", bookLocationToRead, "
", "")){
71                     //handle the case when the content list has html escape chars for the URI
72                     bookLocationToRead = BookwormApp.Utills.decodeHTMLChars (aBook.getBookContentList ().get (contentLocation));
73                     if("true" != BookwormApp.Utills.fileOperations ("EXISTS", bookLocationToRead, "
", "")){
74                         //requested content not available
75                         aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_NAVIGATION_ISSUE);
76                         BookwormApp.AppWindow.showInfoBar (aBook, Gtk.MessageType.WARNING);
77                         warning ("[END] [FUNCTION:provideContent] Page could not be loaded from location:"+bookLocationToRead);
78                         return contents.str;
79                     }
80                 }
81                 //fetch content from extracted book
82                 contents.assign (BookwormApp.Utills.fileOperations ("READ_FILE", bookLocationToRead, "", ""));
83                 //find list of relative urls with src, href, etc and convert them to absolute ones
84                 foreach (string tagname in BookwormApp.Constants.TAG_NAME_WITH_PATHS) {
85                     string[] srcList = BookwormApp.Utills.multiExtractBetweenTwoStrings (contents.str, tagname, "\"");
86                     StringBuilder srcItemFullPath = new StringBuilder ();
87                     foreach (string srcItem in srcList) {
88                         srcItemFullPath.assign (
89                             BookwormApp.Utills.getFullPathFromFilename (
90                                 aBook.getBookExtractionLocation (), srcItem)
91                         );
92                         contents.assign (
93                             contents.str.replace (tagname + srcItem + "\"",
94                                 BookwormApp.Utills.encodeHTMLChars (tagname + srcItemFullPath.str
95                                 + "\""));
96                     }
97                     //update the content for required manipulation
98                     contents.assign (adjustPageContent (aBook, contents.str, mode));
99                     //handle the case for contentLocation set to -1 when the book is added to the DB
100                 } else if (contentLocation == -1 && aBook.getBookContentList ().size > 0) {
101                     provideContent (aBook, 0, mode);
102                 } else {
103                     //requested content not available
104                     aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_NAVIGATION_ISSUE);
105                     BookwormApp.AppWindow.showInfoBar (aBook, Gtk.MessageType.WARNING);
106                 }
107             } else {
108                 //requested content not available
109                 aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_CONTENT_NOT_FOUND_ISSUE);
110                 BookwormApp.AppWindow.showInfoBar (aBook, Gtk.MessageType.WARNING);
111             }
112             debug ("[END] [FUNCTION:provideContent] contents.length=" + contents.str.length.to_string ());
113             return contents.str;
114         }
115     }
116     public static void handleBookMark (string action) {
117         debug ("[START] [FUNCTION:handleBookMark] action=" + action);
118         //get the book being currently read
119         BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap.get (BookwormApp.Bookworm.locationOfEBookCurrentlyRead);
120         switch (action) {
121             case "DISPLAY":
122                 if (aBook != null && aBook.getBookmark () != null && aBook.getBookmark ().index_of (aBook.getBookPageNumber ().to_string ()) != -1) {
123                     //display bookmark as active
124                     BookwormApp.AppHeaderBar.bookmark_active_button.set_visible (true);
125                     BookwormApp.AppHeaderBar.bookmark_inactive_button.set_visible (false);
126                 } else {
127                     //display bookmark as inactive
128                     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,
image");
193     }

```

```

194         //Set background and font colour based on profile
195         if (BookwormApp.Constants.BOOKWORM_READING_MODE[4] == BookwormApp.Bookworm.settings.re
    ading_profile) {
196             //default dark profile
197             cssForTextAndBackgroundColor = " background-color: #002b36 !important; color: #93a
    1a1 !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.sett
    ings.reading_profile) {
202                 //default light profile
203                 cssForTextAndBackgroundColor = " background-color: #fbfbfb !important; color: #000
    000 !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.sett
    ings.reading_profile) {
208                 cssForTextAndBackgroundColor =
209                     " background-color: " + BookwormApp.Bookworm.profileColorList[7] + " !importan
    t;" +
210                     " color: " + BookwormApp.Bookworm.profileColorList[6] + " !importan
    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.sett
    ings.reading_profile) {
215                 cssForTextAndBackgroundColor =
216                     " background-color: " + BookwormApp.Bookworm.profileColorList[4] + " !importan
    t;" +
217                     " color: " + BookwormApp.Bookworm.profileColorList[3] + " !importan
    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] + " !importan
    t;" +
224                     " color: " + BookwormApp.Bookworm.profileColorList[0] + " !importan
    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 b
    ook contents
230             currentBookwormStyles = currentBookwormStyles
231                 .replace ("$$READING_LINE_HEIGHT", BookwormApp.Bookworm.settings.reading_line_heigh
    t)
232                 .replace ("$$READING_WIDTH", (100 - int.parse (BookwormApp.Bookworm.settings.readin
    g_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             }
320             stringToBeHighlighted = stringToBeHighlighted.replace ("\"", "&quot;").rep
    lace ("'", "&#39;");

```

```

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.onLoadJavaScript.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
_string ());
360     //check if Book can be moved back and disable back button otherwise
361     if (currentContentLocation > 0) {
362         aBook.setIfPageBackward (true);
363         BookwormApp.AppWindow.back_button.set_sensitive (true);
364     } else {
365         aBook.setIfPageBackward (false);
366         BookwormApp.AppWindow.back_button.set_sensitive (false);
367     }
368     //check if Book can be moved forward and disable forward button otherwise
369     if (currentContentLocation < (aBook.getBookContentList ().size - 1)) {
370         aBook.setIfPageForward (true);
371         BookwormApp.AppWindow.forward_button.set_sensitive (true);
372     } else {
373         aBook.setIfPageForward (false);
374         BookwormApp.AppWindow.forward_button.set_sensitive (false);
375     }
376     info ("[END] [FUNCTION:controlNavigation] book.location=" + aBook.getBookLocation ());
377     return aBook;
378 }
379
380 public static void refreshCurrentPage () {
381     debug ("[START] [FUNCTION:refreshCurrentPage]");
382     if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_S
TATES[1]) {
383         BookwormApp.Book currentBookForRefresh = BookwormApp.Bookworm.libraryViewMap.get (
384             BookwormApp.Bookworm.locationOfEBookCurrentlyRead);
385         BookwormApp.Bookworm.isPageScrollRequired = true; //set up the flag to scroll to t
he last read position
386         currentBookForRefresh = renderPage (
387             BookwormApp.Bookworm.libraryViewMap.get (
388                 BookwormApp.Bookworm.locationOfEBookCurrentlyRead), "");
389         BookwormApp.Bookworm.libraryViewMap.set (
390             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.Utills.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 (
432             BookwormApp.Bookworm.commandLineArgs[1].strip ()) != -1)
433         {
434             //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
436             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
ndLineArgs[1].strip ();
451         //Initiate Reading the book
452         BookwormApp.Bookworm.readSelectedBook (requestedBook);
453     }
454 } else {
455     //check and continue the last book being read - if "Always show library on start i
s false"
456     if ((!BookwormApp.Bookworm.settings.is_show_library_on_start) && (BookwormApp.Book
worm.settings.book_being_read != "")) {
457         //check if the library contains the book being read last
458         BookwormApp.Book lastReadBook = BookwormApp.DB.getBookFromDB (BookwormApp.Book
worm.settings.book_being_read);
459         if (lastReadBook.getBookLocation () != null && lastReadBook.getBookLocation ()
.length > 1) {
460             debug ("Opening the last read book [" + BookwormApp.Bookworm.settings.book
_being_read + "]");
461             BookwormApp.Bookworm.locationOfEBookCurrentlyRead = BookwormApp.Bookworm.s
ettings.book_being_read;
462             //Initiate Reading the book
463             BookwormApp.Bookworm.readSelectedBook (lastReadBook);
464         } else {
465             warning ("The last read book [" + BookwormApp.Bookworm.settings.book_being
_read + "] " +
466                 "was not found in the library, so showing the library view instead of
opening the last read book");
467         }
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     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, "pixbuf", 0);
85     library_table_treeview.insert_column_with_attributes (-1, BookwormApp.Constants.TEXT_FOR_LIST_VIEW_COLUMN_NAME_TITLE, title_cell_txt, "text", 1);
86     library_table_treeview.insert_column_with_attributes (-1, BookwormApp.Constants.TEXT_FOR_LIST_VIEW_COLUMN_NAME_AUTHOR, author_cell_txt, "text", 2);
87     library_table_treeview.insert_column_with_attributes (-1, BookwormApp.Constants.TEXT_FOR_LIST_VIEW_COLUMN_NAME_MODIFIED_DATE, non_editable_cell_txt, "text", 3);
88     library_table_treeview.insert_column_with_attributes (-1, BookwormApp.Constants.TEXT_FOR_LIST_VIEW_COLUMN_NAME_RATING, rating_cell_pix, "pixbuf", 4);
89     library_table_treeview.insert_column_with_attributes (-1, BookwormApp.Constants.TEXT_FOR_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_BOOK);
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    Gtk.Button 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_BOOK_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    Gtk.Button 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
129    library_page_switcher_box.pack_start (page_button_prev);
130    page_button_prev.set_sensitive (false); //disable the prev button on first time load
131
132    Gtk.Button page_button_next = new Gtk.Button ();
133    page_button_next.set_image (BookwormApp.Bookworm.forward_page_image);
134    page_button_next.set_relief (ReliefStyle.NONE);
135    page_button_next.set_tooltip_markup (BookwormApp.Constants.TOOLTIP_TEXT_FOR_NEXT_PAGE);
136
137    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
240             ation);
241         if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_
242             UI_STATES[6] ||
243             BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_
244             UI_STATES[7])
245         {
246             BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_U
247             I_STATES[7];
248             BookwormApp.Library.updateListViewForSelection (aBook);
249         }
250         if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_
251             UI_STATES[5]) {
252             BookwormApp.Bookworm.readSelectedBook (aBook);
253         }
254     });
255     //Add action to update tree view when editing is Completed
256     title_cell_txt.edited.connect ((path, new_text) => {
257         updateLibraryListViewData (path, new_text, 1);
258     });
259     author_cell_txt.edited.connect ((path, new_text) => {
260         updateLibraryListViewData (path, new_text, 2);
261     });
262     tags_cell_txt.edited.connect ((path, new_text) => {
263         updateLibraryListViewData (path, new_text, 5);
264     });
265     //Add action to open the context menu on right click of tree view
266     library_table_treeview.button_press_event.connect ((event) => {
267         //capture which mouse button was clicked on the book in the library
268         uint mouseButtonClicked;
269         event.get_button (out mouseButtonClicked);
270         //handle right button click for context menu
271         if (event.get_event_type () == Gdk.EventType.BUTTON_PRESS && mouseButtonClicked ==
272             3) {
273             /*TreeIter iter;
274             TreeModel model;
275             Value bookLocation;
276             TreeSelection selection = library_table_treeview.get_selection ();
277             selection.get_selected (out model, out iter);
278             model.get_value (iter, 0, out bookLocation);
279             BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap.get ((string) boo
280                 kLocation);
281             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 reponse to a finger
280 // swipe gesture from right to left to or left to right respectively
281 gesture_swipe.swipe.connect((x, y) => {
282     // Avoid triggering nagivation 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.Bookworm.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.Utills.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] = pathToSele
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_
    UI_STATES[5] ||
347             BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_
    UI_STATES[6] ||
348             BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_
    UI_STATES[7]))
349         {
350             //check if the mode is already in selection mode
351             if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
    ORM_UI_STATES[6] ||
352                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
    ORM_UI_STATES[7])
353             {
354                 //UI is already in selection/selected mode - second click puts the view in
    normal mode
355                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWO
    RM_UI_STATES[5];
356                 BookwormApp.Library.updateListViewForSelection (null);
357             } else {
358                 //UI is not in selection/selected mode - set the view mode to selection mo
    de
359                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWO
    RM_UI_STATES[6];
360                 BookwormApp.Library.updateListViewForSelection (null);
361             }
362         } else {
363             //check if the mode is already in selection mode
364             if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
    ORM_UI_STATES[2] ||
365                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKW
    ORM_UI_STATES[3])
366             {
367                 //UI is already in selection/selected mode - second click puts the view in
    normal mode
368                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = settings.library_view_mode;
369                 BookwormApp.Library.updateGridViewForSelection (null);
370             } else {
371                 //UI is not in selection/selected mode - set the view mode to selection mo
    de
372                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWO
    RM_UI_STATES[2];
373                 BookwormApp.Library.updateGridViewForSelection (null);
374             }
375         }
376     });
377     //Add action for removing a selected book on the library view
378     remove_book_button.clicked.connect (() => {
379         BookwormApp.Library.removeSelectedBooksFromLibrary ();
380     });
381     //handle mouse click on webview (reading mode)
382     aWebView.button_press_event.connect ((event) => {
383         if (!settings.is_leaf_over_page_by_edge_enabled) {
384             return false;
385         }
386         int width;
387         int height;
388         //capture the current window size
389         BookwormApp.Bookworm.window.get_size (out width, out height);
390         //capture which mouse button was clicked on the book in the library
391         uint mouseButtonClicked;
392         event.get_button (out mouseButtonClicked);
393         //handle left button click for page navigation if the click is near the left and r
    ight margins
394         if (event.get_event_type () == Gdk.EventType.BUTTON_PRESS && mouseButtonClicked ==
    1) {
395             //check if mouse is clicked near the right margin 10% of page width and go to
    previous page
396             if(event.x < ((BookwormApp.Constants.PERCENTAGE_WIDTH_FOR_PAGE_NAVIGATION_ON_C
    LICK/100) * width)){

```

```

397         handleBookNavigation ("PREV");
398     }
399     if(event.x > (width - ((BookwormApp.Constants.PERCENTAGE_WIDTH_FOR_PAGE_NAVIGA
TION_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_VI
EW", null);
409     SimpleAction pageActionFullScreenExit = new SimpleAction ("FULL_SCREEN_READING_VIE
W", null);
410     SimpleAction pageActionWordMeaning = new SimpleAction ("WORD_MEANING", null);
411     SimpleAction pageActionAnnotateSelection = new SimpleAction ("ANNOTATE_SELECTION",
null);
412     WebKit.ContextMenuItem pageContextMenuItemWordMeaning = new WebKit.ContextMenuIte
m.from_gaction (
413         pageActionWordMeaning, BookwormApp.Constants.TEXT_FOR_PAGE_CONTEXTMENU_WORD_ME
ANING, null);
414     WebKit.ContextMenuItem pageContextMenuItemFullScreenEntry = new WebKit.ContextMenu
Item.from_gaction (
415         pageActionFullScreenEntry, BookwormApp.Constants.TEXT_FOR_PAGE_CONTEXTMENU_FUL
L_SCREEN_ENTRY, null);
416     WebKit.ContextMenuItem pageContextMenuItemFullScreenExit = new WebKit.ContextMenuI
tem.from_gaction (
417         pageActionFullScreenExit, BookwormApp.Constants.TEXT_FOR_PAGE_CONTEXTMENU_FULL
_SCREEN_EXIT, null);
418     WebKit.ContextMenuItem pageContextMenuItemAnnotateSelection = new WebKit.ContextMe
nuItem.from_gaction (
419         pageActionAnnotateSelection, BookwormApp.Constants.TEXT_FOR_PAGE_CONTEXTMENU_A
NNOTATE_SELECTION, null);
420     context_menu.append (pageContextMenuItemWordMeaning);
421     context_menu.append (pageContextMenuItemAnnotateSelection);
422     if (!settings.is_fullscreen) {
423         context_menu.append (pageContextMenuItemFullScreenEntry);
424     } else {
425         context_menu.append (pageContextMenuItemFullScreenExit);
426     }
427     //Set Context menu items
428     pageActionWordMeaning.activate.connect (() => {
429         string selected_text = BookwormApp.Utils.setWebViewTitle ("document.title = ge
tSelectionText (");
430         if (selected_text != null && selected_text.length > 0) {
431             //Save the page scroll position of the book being read
432             BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap
433                 .get (BookwormApp.Bookworm.locationOfEBookCurrentlyRead);
434             aBook.setBookScrollPos (BookwormApp.contentHandler.getScrollPos ());
435
436             BookwormApp.Info.populateDictionaryResults (selected_text);
437         }
438     });
439     pageActionAnnotateSelection.activate.connect (() => {
440         string selected_text = BookwormApp.Utils.setWebViewTitle ("document.title = ge
tSelectionText (");
441         if (selected_text != null && selected_text.length > 0) {
442             BookwormApp.AppDialog.createAnnotationDialog (selected_text);
443         }
444     });
445     pageActionFullScreenEntry.activate.connect (() => {
446         book_reading_footer_box.hide ();
447         BookwormApp.Bookworm.window.fullscreen ();
448     });
449     pageActionFullScreenExit.activate.connect (() => {
450         book_reading_footer_box.show ();
451         BookwormApp.Bookworm.window.unfullscreen ();
452     });
453     return false;
454 });
455 //capture the url clicked on the webview and action the navigation type clicks
456 aWebView.decide_policy.connect ((decision, type) => {
457     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
ACTION");
463         isWebViewRequestCompleted = true;
464     }
465     if (type == WebKit.PolicyDecisionType.NAVIGATION_ACTION && isWebViewRequestComple
ed) {
466         debug ("Signal captured for Policy type WebKit.PolicyDecisionType.NAVIGATION_A
ACTION");
467         //set the webview request flag to false to prevent re-trigger of this function
468         //the webview request flag will be set to true when the response is received
469         isWebViewRequestCompleted = false;
470         WebKit.NavigationPolicyDecision aNavDecision = (WebKit.NavigationPolicyDecisio
n)decision;
471         WebKit.NavigationAction aNavAction = aNavDecision.get_navigation_action ();
472         WebKit.URIRequest aURIReq = aNavAction.get_request ();
473         string url_clicked_on_webview = BookwormApp.Utils.decodeHTMLChars (aURIReq.get
_uri ().strip ());
474         url_clicked_on_webview = Soup.URI.decode (url_clicked_on_webview);
475         debug ("URL Captured:" + url_clicked_on_webview);
476         //Handle external links (not file://) by opening the default browser i.e. http
://, ftp://
477         if (url_clicked_on_webview.index_of ("file://") == -1) {
478             BookwormApp.Utils.execute_sync_command ("xdg-open " + url_clicked_on_webvi
ew);
479             decision.ignore ();
480             return true;
481         }
482         //Handle Bookworm type links i.e. Annotation Overlay
483         debug ("Window Title:" + BookwormApp.AppWindow.aWebView.get_title ());
484         if (BookwormApp.AppWindow.aWebView.get_title () != null &&
BookwormApp.AppWindow.aWebView.get_title ().length > 1 &&
BookwormApp.AppWindow.aWebView.get_title ().index_of ("annotation:") != -1
)
485         {
486             //Open the annotation dialog
487             BookwormApp.AppDialog.createAnnotationDialog (
BookwormApp.AppWindow.aWebView.get_title ().replace ("annotation:", ""
));
488             isWebViewRequestCompleted = true;
489         }
490         //Handle file:/// type links to other content of the book i.e. Table of Conten
ts
491         string anchor = "";
492         if (url_clicked_on_webview.index_of ("#") != -1) {
493             string[] urlSplittedByHashtag = url_clicked_on_webview.split("#", 2);
494             url_clicked_on_webview = urlSplittedByHashtag[0];
495             anchor = urlSplittedByHashtag[1];
496         }
497         url_clicked_on_webview = File.new_for_path (url_clicked_on_webview).get_basena
me ();
498         int contentLocationPosition = 0;
499         BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap
.get (BookwormApp.Bookworm.locationOfEBookCurrentlyRead);
500         foreach (string aBookContent in aBook.getBookContentList ()) {
501             if (BookwormApp.Utils.decodeHTMLChars (aBookContent).index_of (url_clicked
_on_webview) != -1) {
502                 debug ("Matched Link Clicked to book content:" +
BookwormApp.Utils.decodeHTMLChars (aBookContent));
503                 aBook.setBookPageNumber (contentLocationPosition);
504                 //update book details to libraryView Map
505                 BookwormApp.Bookworm.libraryViewMap.set (aBook.getBookLocation (), aBo
ok);
506                 if (anchor.len() > 0) { // anchor - id in link after # symbol
507                     BookwormApp.Bookworm.isPageScrollRequired = true;
508                     aBook.setAnchor(anchor);
509                 }
510                 aBook = BookwormApp.contentHandler.renderPage (aBook, "");
511                 //Set the mode back to Reading mode
512                 BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BO
OKWORM_UI_STATES[1];
513                 BookwormApp.Bookworm.toggleUIState ();

```

```

519         debug ("URL is initiated from Bookworm Contents, Book page number set
at:" +
520             aBook.getBookPageNumber ().to_string ());
521         break;
522     }
523     contentLocationPosition++;
524 }
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:createBookwormUI]");
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.Bookworm.locationOfEBookCurrentlyRead);
546         debug ("Initiating read forward for eBook:" + currentBookForForward.getBookLocation ());
547         currentBookForForward = BookwormApp.contentHandler.renderPage (currentBookForForward, "FORWARD");
548         //update book details to libraryView Map
549         BookwormApp.Bookworm.libraryViewMap.set (currentBookForForward.getBookLocation (), currentBookForForward);
550         BookwormApp.Bookworm.locationOfEBookCurrentlyRead = currentBookForForward.getBookLocation ();
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.Bookworm.locationOfEBookCurrentlyRead);
557         debug ("Initiating read previous for eBook:" + currentBookForReverse.getBookLocation ());
558         currentBookForReverse = BookwormApp.contentHandler.renderPage (currentBookForReverse, "BACKWARD");
559         //update book details to libraryView Map
560         BookwormApp.Bookworm.libraryViewMap.set (currentBookForReverse.getBookLocation (), currentBookForReverse);
561         BookwormApp.Bookworm.locationOfEBookCurrentlyRead = currentBookForReverse.getBookLocation ();
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_counter + 1;
573         BookwormApp.Library.paginateLibrary ("", "PAGINATED_SEARCH");
574         //disable the forward button if the last modification date returned -1 for this page 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
        llow the forward button to work
586         if (BookwormApp.Bookworm.paginationlist.contains ("-1")) {
587             BookwormApp.Bookworm.paginationlist.remove ("-1");
588         }
589         BookwormApp.Bookworm.current_page_counter = BookwormApp.Bookworm.current_page_
        counter - 1;
590         BookwormApp.Library.paginateLibrary ("", "PAGINATED_SEARCH");
591     } else {
592         //disable the prev button as the counter is on the first page
593         page_button_prev.set_sensitive (false);
594     }
595 }
596 if (!isPaginateRequired) { //set the button status without doing pagination
597     if (BookwormApp.Bookworm.current_page_counter < 1) {
598         page_button_prev.set_sensitive (false);
599     }
600     if (BookwormApp.Bookworm.paginationlist.contains ("-1")) {
601         page_button_next.set_sensitive (false);
602     }
603 }
604 }
605
606 public static bool updateLibraryListViewData (string path, string new_text, int column) {
607     info ("[START] [FUNCTION:updateLibraryListViewData] updating metadata in List View on
        row:" +
608         path + " for change:" + new_text + " on column:" + column.to_string ());
609     //Determine the book whose meta data is being updated
610     Gtk.TreeIter sortedIter;
611     Value bookLocation;
612     TreeModel aTreeModel = library_table_treeview.get_model ();
613     Gtk.TreePath aTreePath = new Gtk.TreePath.from_string (path);
614     aTreeModel.get_iter (out sortedIter, aTreePath);
615     aTreeModel.get_value (sortedIter, 7, out bookLocation);
616     //iterate over the list store
617     Gtk.TreeIter iter;
618     string bookLocationforCurrentRow;
619     bool iterExists = true;
620     iterExists = library_table_liststore.get_iter_first (out iter);
621     while (iterExists) {
622         library_table_liststore.get (iter, 7, out bookLocationforCurrentRow);
623         if ((string)bookLocation == bookLocationforCurrentRow) {
624             library_table_liststore.set (iter, column, new_text);
625             BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap.get ((string) boo
                kLocation);
626             if (column == 1) {
627                 aBook.setBookTitle (new_text);
628             }
629             if (column == 2) {
630                 aBook.setBookAuthor (new_text);
631             }
632             if (column == 5) {
633                 aBook.setBookTags (new_text);
634             }
635             aBook.setWasBookOpened (true);
636             BookwormApp.Bookworm.libraryViewMap.set (aBook.getBookLocation (), aBook);
637             debug ("Completed updating metadata in List View for book:" + (string) bookLoc
                ation);
638             return true; //break out of the iterations
639         }
640         iterExists = library_table_liststore.iter_next (ref iter);
641     }
642     info ("[END] [FUNCTION:updateLibraryListViewData]");
643     return true;
644 }
645
646 public static Granite.Widgets.Welcome createWelcomeScreen () {
647     info ("[START] [FUNCTION:createWelcomeScreen]");
648     //Create a welcome screen for view of library with no books
649     BookwormApp.Bookworm.welcomeWidget = new Granite.Widgets.Welcome (
        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.IconSi
ze.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.Utills.selectFileChooser (
658                 Gtk.FileChooserAction.OPEN, _("Select eBook"), BookwormApp.Bookworm.window, tr
ue, "EBOOKS");
659             //If ebooks were selected, remove the welcome widget from main window and show the
library view
660             if (selectedEBooks.size > 0) {
661                 BookwormApp.Bookworm.window.remove (BookwormApp.Bookworm.welcomeWidget);
662                 BookwormApp.Bookworm.window.add (BookwormApp.Bookworm.bookWormUIBox);
663                 BookwormApp.Bookworm.bookWormUIBox.show_all ();
664                 BookwormApp.Bookworm.toggleUIState ();
665                 BookwormApp.Bookworm.pathsOfBooksToBeAdded = new string[selectedEBooks.size];
666                 int countOfBooksToBeAdded = 0;
667                 foreach (string pathToSelectedBook in selectedEBooks) {
668                     BookwormApp.Bookworm.pathsOfBooksToBeAdded[countOfBooksToBeAdded] = pathTo
SelectedBook;
669                     countOfBooksToBeAdded++;
670                 }
671                 //Display the progress bar
672                 BookwormApp.AppWindow.bookAdditionBar.show ();
673                 BookwormApp.Bookworm.isBookBeingAddedToLibrary = true;
674                 BookwormApp.Library.addBooksToLibrary ();
675             }
676         });
677         info ("[END] [FUNCTION:createWelcomeScreen] ");
678         return BookwormApp.Bookworm.welcomeWidget;
679     }
680
681     public static void showInfoBar (BookwormApp.Book aBook, MessageType aMessageType) {
682         debug ("[START] [FUNCTION:showInfoBar] ");
683         StringBuilder message = new StringBuilder ("");
684         message.append (aBook.getParsingIssue ().append (aBook.getBookLocation ());
685         BookwormApp.AppWindow.infoBarLabel.set_text (message.str);
686         BookwormApp.AppWindow.infoBar.set_message_type (aMessageType);
687         BookwormApp.AppWindow.infoBar.show ();
688         debug ("[END] [FUNCTION:showInfoBar] with message:" + message.str);
689     }
690
691     //Handle action for close of the InfoBar
692     public static void on_info_bar_closed () {
693         BookwormApp.AppWindow.infoBar.hide ();
694     }
695
696     public static bool handleWindowStateEvents (Gdk.EventWindowState ev) {
697         if (ev.type == Gdk.EventType.WINDOW_STATE) {
698             if ((ev.window.get_state () & Gdk.WindowState.FULLSCREEN) == 0) {
699                 settings.is_fullscreen = false;
700             } else {
701                 settings.is_fullscreen = true;
702             }
703         }
704         return false;
705     }
706
707     public static void controlDeletionButton (bool selectionState) {
708         if (selectionState) {
709             //Enable the Deletion Button as a book is selected for potential removal
710             noOfBooksSelected++;
711             remove_book_button.set_sensitive (true);
712             remove_book_button.
713                 set_tooltip_markup (BookwormApp.Constants.TOOLTIP_TEXT_FOR_REMOVE_BOOK);
714         } else {
715             //Check and Disable the Deletion Button if no books are selected after this de-sel
ection
716             noOfBooksSelected--;
717             if (noOfBooksSelected < 1) {
718                 remove_book_button.set_sensitive (false);
719                 remove_book_button.set_tooltip_markup (
720                     BookwormApp.Constants.TOOLTIP_TEXT_FOR_REMOVE_BOOK_UNSELECTED);

```

721 }
722 }
723 }
724
725
726 }