BOOKWORM

Open Source eBook Reader
Source Code

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

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

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

Files included:

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

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

```
/* Copyright 2017 Siddhartha Das (bablu.boy@gmail.com)
*
* This file is part of Bookworm and entry point to the
* application with the main method
*
* Bookworm is free software: you can redistribute it
* and/or modify it under the terms of the GNU General Public License as
* published by the Free Software Foundation, either version 3 of the
* License, or (at your option) any later version.
*
* Bookworm is distributed in the hope that it will be
* useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
* Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with Bookworm. If not, see http://www.gnu.org/licenses/.
*/
BookwormApp.Bookworm application;
public static int main (string[] args) {
    Environment.set_variable ("G_MESSAGES_DEBUG", "all", true);
    //Get an instance of Bookworm if is running, otherwise create a new instance
    application = BookwormApp.Bookworm.getAppInstance ();
    //Workaround to get Granite’s --about & Gtk’s --help working together
    if ("--help"     in args || "-h" in args ||
        "--version"  in args ||
        "--discover" in args)
    {
        return application.processCommandLine (args);
    } else {
        Gtk.init (ref args);
        if ("--debug" in args) {
            application.command_line_option_debug = true;
        }
        if ("--info" in args) {
            application.command_line_option_info = true;
        }
        return application.run (args);
    }
}
```

==
bookworm.vala ? Application Class
==

```
/* Copyright 2017 Siddhartha Das (bablu.boy@gmail.com)
*
* This file is part of Bookworm and is the main Application class
*
* Bookworm is free software: you can redistribute it
* and/or modify it under the terms of the GNU General Public License as
* published by the Free Software Foundation, either version 3 of the
* License, or (at your option) any later version.
*
* Bookworm is distributed in the hope that it will be
* useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
* Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with Bookworm. If not, see http://www.gnu.org/licenses/.
*/

using Gtk;
using Gee;
using Granite.Widgets;
using Granite.Services;
public const string GETTEXT_PACKAGE = "bookworm";

public class BookwormApp.Bookworm : Granite.Application {
    private static Bookworm application;
    private static bool isBookwormRunning = false;
    public int exitCodeForCommand = 0;
    public static Granite.Services.Paths app_xdg_path;
    public static string bookworm_config_path = "";

    public static string[] commandLineArgs;
    public static bool command_line_option_version = false;
    public static bool command_line_option_debug = false;
    public static bool command_line_option_info = false;
    public static bool command_line_option_discover = false;
    private static OptionEntry[] options;

    public StringBuilder spawn_async_with_pipes_output = new StringBuilder ("");

    public static BookwormApp.Settings settings;
    public static Gtk.ApplicationWindow window;
    public static Gtk.IconTheme default_theme;
    public static CssProvider cssProvider;
    public static Gtk.Box bookWormUIBox;
    public static Granite.Widgets.Welcome welcomeWidget;
    public static Granite.Widgets.ModeButton library_mode_button;
    public static Gtk.TreeModelFilter libraryTreeModelFilter;
    public static Gtk.Button library_view_button;
    public static Gtk.Button content_list_button;
    public static Gtk.Button prefButton;
    public static Gdk.Pixbuf bookSelectionPix;
    public static Gdk.Pixbuf bookSelectedPix;
    public static Gdk.Pixbuf image_selection_option_small;
    public static Gdk.Pixbuf image_selection_checked_small;
    public static Gdk.Pixbuf image_selection_transparent_small;
    public static Gdk.Pixbuf image_selection_scaled;
    public static Gdk.Pixbuf image_rating_1;
    public static Gdk.Pixbuf image_rating_2;
    public static Gdk.Pixbuf image_rating_3;
    public static Gdk.Pixbuf image_rating_4;
    public static Gdk.Pixbuf image_rating_5;
    public static Gtk.Image select_book_image;
    public static Gtk.Image add_book_image;
    public static Gtk.Image remove_book_image;
    public static Gtk.Image updateImageIcon;
    public static Gtk.Image add_scan_directory_image;
    public static Gtk.Image remove_scan_directory_image;
    public static Gtk.Image library_list_button_image;
    public static Gtk.Image library_grid_button_image;
    public static Gtk.Image content_list_button_image;
    public static Gtk.Image menu_icon_text_large;
    public static Gtk.Image menu_icon;
    public static Gtk.Image pref_menu_icon_text_large;
    public static Gtk.Image pref_menu_icon_text_small;
    public static Gtk.Image pref_menu_icon_align_left;
    public static Gtk.Image pref_menu_icon_align_right;
    public static Gtk.Image back_button_image;
    public static Gtk.Image forward_button_image;
    public static Gtk.Image back_page_image;
    public static Gtk.Image forward_page_image;

    public static string BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_UI_STATES[0];
    public static Gee.HashMap<string, BookwormApp.Book> libraryViewMap = new Gee.HashMap<string, BookwormApp.Book> ();
    public static string locationOfEBookCurrentlyRead = "";
    public static string[] pathsOfBooksToBeAdded;
    public static int noOfBooksAddedFromCommand = 0;
    public static bool isBookBeingAddedToLibrary = false;
    public static bool isPageScrollRequired = false;
    public static StringBuilder pathsOfBooksInLibraryOnLoadStr = new StringBuilder ("");
    public static StringBuilder pathsOfBooksNotAddedStr = new StringBuilder ("");
    public static StringBuilder onLoadJavaScript = new StringBuilder ("");
    public static string bookwormScripts = "";
    public static string bookwormStyles = "";
    public static string bookTextSearchString = "";
    public static TreeMap<string,string> searchResultsMap = new TreeMap<string,string> ();
    public static StringBuilder aContentFileToBeSearched = new StringBuilder ("");
    public static string[] profileColorList;
    public static string no_of_books_per_page = "21";
    public static ArrayList<string> paginationlist = new ArrayList<string> ();
    public static int current_page_counter = 0;
    public static ShortcutsAssocsHolder shortcutAssocs;
    public static AccelGroup accel;

    construct {
        build_version = BookwormApp.Constants.bookworm_version;
        application_id = BookwormApp.Constants.bookworm_id;
        flags |= ApplicationFlags.HANDLES_COMMAND_LINE;
        program_name = BookwormApp.Constants.program_name;
        exec_name = BookwormApp.Constants.bookworm_id;
        options = new OptionEntry[4];
        options[0] = { "version",  0, 0, OptionArg.NONE, ref command_line_option_version,  _("Display version number"),     null };
        options[1] = { "debug",    0, 0, OptionArg.NONE, ref command_line_option_debug,    _("Run Bookworm in debug mode"), null };
        options[2] = { "info",     0, 0, OptionArg.NONE, ref command_line_option_info,     _("Run Bookworm in info mode"),  null };
        options[3] = { "discover", 0, 0, OptionArg.NONE, ref command_line_option_discover, _("Automatically add new books from watched folders"), null };
        add_main_option_entries (options);
    }

    private Bookworm () {
        Object (application_id: BookwormApp.Constants.bookworm_id, flags: ApplicationFlags.HANDLES_COMMAND_LINE);
        Intl.setlocale (LocaleCategory.MESSAGES, "");
        Intl.textdomain (GETTEXT_PACKAGE);
        Intl.bind_textdomain_codeset (GETTEXT_PACKAGE, "utf-8");
        //Initialize XDG Paths
        app_xdg_path = new Granite.Services.Paths ();
        app_xdg_path.initialize (Constants.bookworm_id, Constants.INSTALL_SCRIPTS_DIR);
        bookworm_config_path = app_xdg_path.user_data_folder.get_path ();
        debug ("Bookworm Install Directory: " + BookwormApp.Constants.INSTALL_PREFIX);
        debug ("Bookworm Install Tasks Scripts Directory: " + BookwormApp.Constants.INSTALL_TASKS_DIR);
        debug ("Bookworm Install Mobi Scripts Directory: " + BookwormApp.Constants.INSTALL_MOBILIB_DIR);
        debug ("Bookworm User Data Directory: " + bookworm_config_path);
    }

    public static Bookworm getAppInstance () {
        if (application == null) {
            application = new Bookworm ();
        } else {
            //do nothing, return the existing instance
        }
        return application;
    }

    public override int command_line (ApplicationCommandLine command_line) {
        commandLineArgs = command_line.get_arguments ();
        activate ();
        return 0;
    }

    public int processCommandLine (string[] args) {
        try {
            var opt_context = new OptionContext ("- bookworm");
            opt_context.set_help_enabled (true);
            opt_context.add_main_entries (options, null);
            unowned string[] tmpArgs = args;
            opt_context.parse (ref tmpArgs);
            if ("--version" in args) {
                command_line_option_version = true;
            }
        } catch (OptionError e) {
            info ("Run ’%s --help’ to see a full list of available command line options.\n", args[0]);
            info ("error: %s\n", e.message);
            return 0;
        }

        if (command_line_option_version) {
            print ("Bookworm Version " + BookwormApp.Constants.bookworm_version + "\n");
        }
        if (command_line_option_discover) {
            BookwormApp.BackgroundTasks.performTasks ();
        }
        return 0;
    }

    public override void activate () {
        Logger.initialize ("com.github.babluboy.bookworm");
        if (command_line_option_debug) {
            Logger.DisplayLevel = LogLevel.DEBUG;
        }
        if (command_line_option_info) {
            Logger.DisplayLevel = LogLevel.INFO;
        }
        info ("[START] [FUNCTION:activate]");
        //proceed if Bookworm is not running already
        if (!isBookwormRunning) {
            debug ("No instance of Bookworm found");
            window = new Gtk.ApplicationWindow (this);
            default_theme = Gtk.IconTheme.get_default ();
            //retrieve Settings
            settings = BookwormApp.Settings.get_instance ();
            //set window attributes
            window.set_border_width (0);
            window.get_style_context ().add_class ("rounded");
            //set the minimum size of the window on minimize
            window.set_size_request (600, 350);
            //set css provider
            cssProvider = new Gtk.CssProvider ();
            loadCSSProvider (cssProvider);
            //load images/icons
            loadImages ();
            //add window components
            window.set_titlebar (BookwormApp.AppHeaderBar.create_headerbar ());

            BookwormApp.AppWindow.createWelcomeScreen ();
            bookWormUIBox = BookwormApp.AppWindow.createBoookwormUI ();

            accel = new AccelGroup ();

            shortcutAssocs = BookwormApp.ShortcutsAssocsHolder.readFromSettings ();
            BookwormApp.Shortcuts.attachShortcutsToWidgets ();

            /*
            var accelMap = AccelMap.@get ();
            accelMap.add_entry ("<Root-Window>/some-path", Gdk.Key.F10, Gdk.ModifierType.CONTROL_MASK);
            accelMap.add_entry ("<Root-Window>/some-path", Gdk.Key.Cyrillic_TE, Gdk.ModifierType.CONTROL_MASK);
            accelMap.add_entry ("<Root-Window>/some-path", Gdk.Key.Cyrillic_TE, Gdk.ModifierType.CONTROL_MASK | Gdk.ModifierType.SHIFT_MASK);
            accel.connect_by_path ("<Root-Window>/some-path", () => {
                info ("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ sunshine 2");
                return false;
            });
            */
            window.add_accel_group (accel);

            //load saved books from DB and add them to Library view
            loadBookwormState ();
            //show welcome screen if no book is present in library instead of the normal library view
            if (libraryViewMap.size == 0) {
                window.add (welcomeWidget);
            } else {
                window.add (bookWormUIBox);
            }
            add_window (window);
            window.show_all ();
            toggleUIState ();
            //capture window re-size events and save the window size
            window.size_allocate.connect (() => {
                saveWindowState ();
            });
            //Exit Application Event
            window.destroy.connect (() => {
                //Perform close down activities
                closeBookWorm ();
            });
            //Add keyboard shortcuts on the window
            window.add_events (Gdk.EventMask.KEY_PRESS_MASK);

            window.window_state_event.connect (BookwormApp.AppWindow.handleWindowStateEvents);

            isBookwormRunning = true;
            debug ("Completed creating an instance of Bookworm");
        } else {
            window.present ();
            debug ("An instance of Bookworm is already running");
        }
        //check if any books needed to be added/opened - if eBook (s) were opened from File Explorer using Bookworm
        if (commandLineArgs.length > 1) {
            info ("Book (s) to be added/opened based on command line parameters. Size of command line attributes:" +
                commandLineArgs.length.to_string ()
            );
            pathsOfBooksToBeAdded = new string[commandLineArgs.length];
            pathsOfBooksToBeAdded = commandLineArgs;
            //Display the progress bar
            BookwormApp.AppWindow.bookAdditionBar.show ();
            isBookBeingAddedToLibrary = true;
            //handle the case if the welcome screen is shown
            if (libraryViewMap.size == 0) {
                //remove the welcome widget from main window
                window.remove (welcomeWidget);
                //add the library view to the window
                window.add (bookWormUIBox);
                bookWormUIBox.show_all ();
                toggleUIState ();
            }
            //Update the library view with books - this returns control back immediately
            BookwormApp.Library.addBooksToLibrary ();
        }
        //Perform post start up actions
        BookwormApp.contentHandler.performStartUpActions ();
        toggleUIState ();
        info ("[END] [FUNCTION:activate]");
```

```
    }

    public override void open (File[] files, string hint) {
        /* TODO */
    }

    public void loadImages () {
        info ("[START] [FUNCTION:loadImages]");
        try {
            image_selection_option_small = new Gdk.Pixbuf.from_resource (BookwormApp.Constants.SELECTION_OPTION_IMAGE_SMALL_LOCATION);
            image_selection_checked_small = new Gdk.Pixbuf.from_resource (BookwormApp.Constants.SELECTION_CHECKED_IMAGE_SMALL_LOCATION);
            image_selection_transparent_small = new Gdk.Pixbuf.from_resource (BookwormApp.Constants.SELECTION_CHECKED_IMAGE_SMALL_LOCATION);
            image_selection_transparent_small.fill (0x00000000);
            image_rating_1 = new Gdk.Pixbuf.from_resource (BookwormApp.Constants.RATING_1_IMAGE_LOCATION);
            image_rating_2 = new Gdk.Pixbuf.from_resource (BookwormApp.Constants.RATING_2_IMAGE_LOCATION);
            image_rating_3 = new Gdk.Pixbuf.from_resource (BookwormApp.Constants.RATING_3_IMAGE_LOCATION);
            image_rating_4 = new Gdk.Pixbuf.from_resource (BookwormApp.Constants.RATING_4_IMAGE_LOCATION);
            image_rating_5 = new Gdk.Pixbuf.from_resource (BookwormApp.Constants.RATING_5_IMAGE_LOCATION);

            if (Gtk.IconTheme.get_default ().has_icon ("object-select-symbolic")) {
                select_book_image = new Gtk.Image.from_icon_name ("object-select-symbolic", Gtk.IconSize.MENU);
            } else {
                select_book_image = new Gtk.Image.from_resource (BookwormApp.Constants.SELECT_BOOK_ICON_IMAGE_LOCATION);
            }
            if (Gtk.IconTheme.get_default ().has_icon ("list-add-symbolic")) {
                add_book_image = new Gtk.Image.from_icon_name ("list-add-symbolic", Gtk.IconSize.MENU);
            } else {
                add_book_image = new Gtk.Image.from_resource (BookwormApp.Constants.ADD_BOOK_ICON_IMAGE_LOCATION);
            }
            if (Gtk.IconTheme.get_default ().has_icon ("list-remove-symbolic")) {
                remove_book_image = new Gtk.Image.from_icon_name ("list-remove-symbolic", Gtk.IconSize.MENU);
            } else {
                remove_book_image = new Gtk.Image.from_resource (BookwormApp.Constants.REMOVE_BOOK_ICON_IMAGE_LOCATION);
            }
            if (Gtk.IconTheme.get_default ().has_icon ("list-add-symbolic")) {
                add_scan_directory_image = new Gtk.Image.from_icon_name ("list-add-symbolic", Gtk.IconSize.MENU);
            } else {
                add_scan_directory_image = new Gtk.Image.from_resource (BookwormApp.Constants.ADD_BOOK_ICON_IMAGE_LOCATION);
            }
            if (Gtk.IconTheme.get_default ().has_icon ("list-remove-symbolic")) {
                remove_scan_directory_image = new Gtk.Image.from_icon_name ("list-remove-symbolic", Gtk.IconSize.MENU);
            } else {
                remove_scan_directory_image = new Gtk.Image.from_resource (BookwormApp.Constants.REMOVE_BOOK_ICON_IMAGE_LOCATION);
            }
            if (Gtk.IconTheme.get_default ().has_icon ("view-list-symbolic")) {
                library_list_button_image = new Gtk.Image.from_icon_name ("view-list-symbolic", Gtk.IconSize.SMALL_TOOLBAR);
            } else {
                library_list_button_image = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_resource_at_scale (BookwormApp.Constants.LIBRARY_VIEW_LIST_IMAGE_LOCATION, 16, 16, true));
            }
            if (Gtk.IconTheme.get_default ().has_icon ("view-grid-symbolic")) {
                library_grid_button_image = new Gtk.Image.from_icon_name ("view-grid-symbolic", Gtk.IconSize.SMALL_TOOLBAR);
            } else {
                library_grid_button_image = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_resource_at_scale  (BookwormApp.Constants.LIBRARY_VIEW_GRID_IMAGE_LOCATION, 16, 16, true));
            }
            if (Gtk.IconTheme.get_default ().has_icon ("format-justify-left-symbolic")) {
                pref_menu_icon_align_left = new Gtk.Image.from_icon_name ("format-justify-left-symbolic", Gtk.IconSize.SMALL_TOOLBAR);
            } else {
                pref_menu_icon_align_left = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_resource_at_scale  (BookwormApp.Constants.TEXT_ALIGN_LEFT_ICON_LOCATION, 16, 16, true));
            }
            if (Gtk.IconTheme.get_default ().has_icon ("format-justify-right-symbolic")) {
                pref_menu_icon_align_right = new Gtk.Image.from_icon_name ("format-justify-right-symbolic", Gtk.IconSize.SMALL_TOOLBAR);
            } else {
                pref_menu_icon_align_right = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_resource_at_scale  (BookwormApp.Constants.TEXT_ALIGN_RIGHT_ICON_LOCATION, 16, 16, true));
            }
            if (Gtk.IconTheme.get_default ().has_icon ("help-info-symbolic")) {
                content_list_button_image = new Gtk.Image.from_icon_name ("help-info-symbolic", Gtk.IconSize.LARGE_TOOLBAR);
            } else {
                content_list_button_image = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_resource_at_scale    (BookwormApp.Constants.BOOK_INFO_IMAGE_LOCATION, 24, 24, true) );
            }
            if (Gtk.IconTheme.get_default ().has_icon ("format-text-larger-symbolic")) {
                menu_icon_text_large = new Gtk.Image.from_icon_name ("format-text-larger-symbolic", Gtk.IconSize.LARGE_TOOLBAR);
            } else {
                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) );
            }
            if (Gtk.IconTheme.get_default ().has_icon ("open-menu")) {
                menu_icon = new Gtk.Image.from_icon_name ("open-menu", Gtk.IconSize.LARGE_TOOLBAR);
            } else {
                menu_icon = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_resource_at_scale (BookwormApp.Constants.HEADERBAR_PROPERTIES_IMAGE_LOCATION, 24, 24, true)  );
            }
            if (Gtk.IconTheme.get_default ().has_icon ("format-text-larger-symbolic")) {
                pref_menu_icon_text_large = new Gtk.Image.from_icon_name ("format-text-larger-symbolic", Gtk.IconSize.LARGE_TOOLBAR);
            } else {
                pref_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));
            }
            if (Gtk.IconTheme.get_default ().has_icon ("format-text-smaller-symbolic")) {
                pref_menu_icon_text_small = new Gtk.Image.from_icon_name ("format-text-smaller-symbolic", Gtk.IconSize.LARGE_TOOLBAR);
            } else {
                pref_menu_icon_text_small = new Gtk.Image.from_pixbuf (new Gdk.Pixbuf.from_resource_at_scale (BookwormApp.Constants.TEXT_SMALLER_IMAGE_ICON_LOCATION, 24, 24, true));
            }
            if (Gtk.IconTheme.get_default ().has_icon ("go-previous-symbolic")) {
                back_button_image = new Gtk.Image.from_icon_name ("go-previous-symbolic", Gtk.IconSize.MENU);
            } else {
                back_button_image = new Gtk.Image.from_resource (BookwormApp.Constants.PREV_PAGE_ICON_IMAGE_LOCATION);
            }
            if (Gtk.IconTheme.get_default ().has_icon ("go-next-symbolic")) {
                forward_button_image = new Gtk.Image.from_icon_name ("go-next-symbolic", Gtk.IconSize.MENU);
            } else {
                forward_button_image = new Gtk.Image.from_resource (BookwormApp.Constants.NEXT_PAGE_ICON_IMAGE_LOCATION);
            }
            if (Gtk.IconTheme.get_default ().has_icon ("go-previous-symbolic")) {
                back_page_image = new Gtk.Image.from_icon_name ("go-previous-symbolic", Gtk.IconSize.MENU);
            } else {
                back_page_image = new Gtk.Image.from_resource (BookwormApp.Constants.PREV_PAGE_ICON_IMAGE_LOCATION);
            }
            if (Gtk.IconTheme.get_default ().has_icon ("go-next-symbolic")) {
                forward_page_image = new Gtk.Image.from_icon_name ("go-next-symbolic", Gtk.IconSize.MENU);
            } else {
                forward_page_image = new Gtk.Image.from_resource (BookwormApp.Constants.NEXT_PAGE_ICON_IMAGE_LOCATION);
            }
        } catch (GLib.Error e) {
            warning ("Image could not be loaded. Error:" + e.message);
        }
        info ("[END] [FUNCTION:loadImages]");
    }

    public static void loadCSSProvider (Gtk.CssProvider cssProvider) {
        info ("[START] [FUNCTION:loadCSSProvider] cssProvider=" + cssProvider.to_string ());
        string dynamicCSSContent = "";
        try {
            profileColorList = settings.list_of_profile_colors.split (",");
            //temp check to ensure the change to settings for colours increasing from 6 to 9 is handled
            if (profileColorList.length < 9) {
                profileColorList = {"#000000","#fbfbfb","#E8ED00","#586e75","#fdf6e3","#87FF2B","#93a1a1","#002b36","#3465A4"};
            }
            dynamicCSSContent = BookwormApp.Constants.DYNAMIC_CSS_CONTENT
                .replace ("<profile_1_color>",profileColorList[0])
                .replace ("<profile_1_bgcolor>",profileColorList[1])
                .replace ("<profile_2_color>",profileColorList[3])
                .replace ("<profile_2_bgcolor>",profileColorList[4])
                .replace ("<profile_3_color>",profileColorList[6])
                .replace ("<profile_3_bgcolor>",profileColorList[7]);
            debug ("CSS for Profile Buttons:" + dynamicCSSContent);
            cssProvider.load_from_data (dynamicCSSContent, dynamicCSSContent.length);
        } catch (GLib.Error e) {
            warning ("Stylesheet could not be loaded from CSS Content[" + dynamicCSSContent + "]. Error:" + e.message);
        }
        Gtk.StyleContext.add_provider_for_screen (
            Gdk.Screen.get_default (), cssProvider, Gtk.STYLE_PROVIDER_PRIORITY_APPLICATION);
        info ("[END] [FUNCTION:loadCSSProvider]");
    }

    public void loadBookwormState () {
        info ("[START] [FUNCTION:loadBookwormState]");
        //check and create required directory structure
        BookwormApp.Utils.fileOperations ("CREATEDIR", BookwormApp.Constants.EBOOK_EXTRACTION_LOCATION, "", "");
        BookwormApp.Utils.fileOperations ("CREATEDIR", BookwormApp.Bookworm.bookworm_config_path, "", "");
        BookwormApp.Utils.fileOperations ("CREATEDIR", BookwormApp.Bookworm.bookworm_config_path + "/covers/", "", "");
        BookwormApp.Utils.fileOperations ("CREATEDIR", BookwormApp.Bookworm.bookworm_config_path + "/books/", "", "");
        //Set the window to the last saved position
        if (settings.pos_x == 0 && settings.pos_y == 0) {
            window.set_position (Gtk.WindowPosition.CENTER);
        } else {
            window.move (settings.pos_x, settings.pos_y);
        }
        //set window size to the last saved height/width
        if (settings.window_is_maximized) {
            window.maximize ();
        } else {
            if (settings.window_width > 0 && settings.window_height > 0) {
                window.set_default_size (settings.window_width, settings.window_height);
            } else {
                window.set_default_size (1200, 700);
            }
        }
        //check last state and turn on dark theme
        if (BookwormApp.Bookworm.settings.is_dark_theme_enabled) {
            Gtk.Settings.get_default ().gtk_application_prefer_dark_theme = true;
        }
        //set the number of books per library page as per user set value
        no_of_books_per_page = settings.library_page_items.to_string ();

        //check if the database exists otherwise create database and required tables
        BookwormApp.DB.initializeBookWormDB (BookwormApp.Bookworm.bookworm_config_path);
        //set the library view
        BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = settings.library_view_mode;
        //Fetch details of Books from the database
        BookwormApp.Bookworm.paginationlist.add ("");
        BookwormApp.Bookworm.current_page_counter = 0;
        BookwormApp.Library.paginateLibrary ("","PAGINATED_SEARCH");
        //Set the library pagination buttons based on the paginate call
        BookwormApp.AppWindow.handleLibraryPageButtons ("", false);

        info ("[END] [FUNCTION:loadBookwormState]");
    }

    public async void closeBookWorm () {
        info ("[START] [FUNCTION:closeBookWorm]");
        //If Bookworm was closed while in Reading mode - save book details for subsequent read
        if (BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[1]) {
            //Save the page scroll position of the book being read
            (libraryViewMap.get (locationOfEBookCurrentlyRead)).setBookScrollPos (BookwormApp.contentHandler.getScrollPos ());
            //Save the path to the book being read
            settings.book_being_read = locationOfEBookCurrentlyRead;
        } else {
            //Bookworm was not in reading view during close - remove path of book read last
            settings.book_being_read = "";
        }
        //release the control so that the window is closed
        Idle.add (closeBookWorm.callback);
        yield;
        //Update the book details to the database if it was opened in this session
        foreach (var book in libraryViewMap.values) {
            if (((BookwormApp.Book)book).getWasBookOpened ()) {
                BookwormApp.DB.updateBookToDataBase ((BookwormApp.Book)book);
                debug ("Completed saving the book data into DB");
            }
        }
        //Run dicovery of books as a background task if not already running
        string checkBackgroundTask = BookwormApp.Utils.execute_sync_command ("ps -ef");
        if (checkBackgroundTask.index_of ("bookworm --discover") == -1) {
            BookwormApp.Utils.execute_async_multiarg_command_pipes ({"com.github.babluboy.bookworm", "--discover", "&"});
        } else {
            debug ("Bookworm discover process already running....");
        }
        info ("[END] [FUNCTION:closeBookWorm]");
    }

    public void saveWindowState () {
        int width;
        int height;
        int x;
        int y;
        window.get_size (out width, out height);
        window.get_position (out x, out y);
        if (settings.pos_x != x || settings.pos_y != y) {
            settings.pos_x = x;
            settings.pos_y = y;
        }
        if (settings.window_width != width || settings.window_height != height) {
            settings.window_width = width;
            settings.window_height = height;
        }
        if (window.is_maximized == true) {
            settings.window_is_maximized = true;
        } else {
            settings.window_is_maximized = false;
        }
        settings.zoom_level = BookwormApp.AppWindow.aWebView.get_zoom_level ();
    }

    public static void readSelectedBook (owned BookwormApp.Book aBook) {
        info ("[START] [FUNCTION:readSelectedBook] book.location=" + aBook.getBookLocation ());
        //Fetch the book meta data from the database if it is not already available
        if (aBook.getBookContentList ().size < 1) {
            //content size should be greater than 1 if the book data has been loaded
            aBook = BookwormApp.DB.getBookMetaDataFromDB (aBook);
        }
        debug ("Book details before attempting to open book for reading:" +  aBook.to_string ());
        //Handle the case when the page number of the book is not set
        if (aBook.getBookPageNumber () == -1) {
            aBook.setBookPageNumber (0);
        } else {
            //This book was previously being read, so it should be opened at the last reading position
            //Enable the flag which will scroll the page to the last read position
            isPageScrollRequired = true;
        }
        //Handle the case when the page number of the book is outside limits
        if (aBook.getBookPageNumber () >= aBook.getBookContentList ().size) {
            aBook.setBookPageNumber (aBook.getBookContentList ().size - 1);
        }
        //check if the extracted contents for the book exists
        if (BookwormApp.Bookworm.settings.is_local_storage_enabled &&
            "true" == BookwormApp.Utils.fileOperations ("DIR_EXISTS", aBook.getBookExtractionLocation (), "", "") &&
            aBook.getBookContentList () != null &&
            aBook.getBookContentList ().size > 0 &&
            aBook.getBookContentList ().size >= aBook.getBookPageNumber () &&
            "true" == BookwormApp.Utils.fileOperations (
                "EXISTS", BookwormApp.Utils.decodeHTMLChars (
                    aBook.getBookContentList ().get (aBook.getBookPageNumber ())), "", "")) {
            //extraction of book not required
            aBook.setIsBookParsed (true);
            debug ("Book has already been extracted and the extracted contents exist at:" + aBook.getBookExtractionLocation ());
        } else {
            //Extract and Parse the eBook (this will overwrite the contents already extracted)
            debug ("Extracted contents of the book was not found at expected location [" +
                aBook.getBookExtractionLocation () + "], attempting to load book from original location [" +
                aBook.getBookLocation () + "]");
            aBook = genericParser (aBook);
            //If ebook was not parsed successfully, show the warning info banner message
            if (!aBook.getIsBookParsed ()) {
                BookwormApp.AppWindow.showInfoBar (aBook, MessageType.WARNING);
            }
        }
        //progress in opening the book for reading if it has been parsed successfully
        if (aBook.getIsBookParsed ()) {
            //update the total number of pages in the book
            aBook.setBookTotalPages (aBook.getBookContentList ().size);
            //update book to mark it has been opened in this session
            aBook.setBookLastModificationDate ((new DateTime.now_utc ().to_unix ()).to_string ());
            aBook.setWasBookOpened (true);
            //update book details to libraryView Map
            libraryViewMap.set (aBook.getBookLocation (), aBook);
            locationOfEBookCurrentlyRead = aBook.getBookLocation ();
            //Update header title
            BookwormApp.AppHeaderBar.get_headerbar ().title = BookwormApp.Utils.parseMarkUp (aBook.getBookTitle ());
            //change the application view to Book Reading mode
            BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_UI_STATES[1];
            toggleUIState ();
            //reset the contents of the search entry
            BookwormApp.Info.searchresults_scroll.get_child ().destroy ();
            //set the max value and the current value of the page slider
            BookwormApp.AppWindow.pageAdjustment.set_upper (aBook.getBookContentList ().size);
            BookwormApp.AppWindow.pageAdjustment.set_value (aBook.getBookPageNumber ());
            //render the contents of the current page of book
            aBook = BookwormApp.contentHandler.renderPage (aBook, "");
            //set the focus to the webview to capture keypress events
            BookwormApp.AppWindow.aWebView.grab_focus ();
        }
        info ("[END] [FUNCTION:readSelectedBook] book.location=" + aBook.getBookLocation ());
    }

    public static void toggleUIState () {
        info ("[START] [FUNCTION:toggleUIState] bookworm current state:" + BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE);
        //hide the inforbar if there is no text in it
        if (BookwormApp.AppWindow.infobarLabel.get_text ().length < 1) {
            BookwormApp.AppWindow.infobar.hide ();
        }
        //Set-up UI specific for Library Grid View Mode
        if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[0] ||
            BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[2] ||
            BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[3])
        {
            BookwormApp.AppWindow.library_list_scroll.set_visible (false);
            BookwormApp.AppWindow.library_grid_scroll.set_visible (true);
            BookwormApp.AppWindow.library_grid.show_all ();
            BookwormApp.AppWindow.bookLibrary_ui_box.grab_focus ();
        }
        //Set-up UI specific for Library List View Mode
        if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[5] ||
            BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[6] ||
            BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[7])
        {
            BookwormApp.AppWindow.library_grid_scroll.set_visible (false);
            BookwormApp.AppWindow.library_list_scroll.set_visible (true);
            BookwormApp.AppWindow.library_table_treeview.show_all ();
            BookwormApp.AppWindow.library_table_treeview.grab_focus ();
        }
        //Set-up UI for Library View Mode (List or Grid)
        if (BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[0] ||
            BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[2] ||
            BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[3] ||
            BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[5])
        {
            BookwormApp.AppHeaderBar.headerSearchBar.set_placeholder_text (
                BookwormApp.Constants.TEXT_FOR_HEADERBAR_LIBRARY_SEARCH);
            library_mode_button.set_visible (true);
            content_list_button.set_visible (false);
            library_view_button.set_visible (false);
            BookwormApp.AppWindow.bookLibrary_ui_box.set_visible (true);
            BookwormApp.AppWindow.bookReading_ui_box.set_visible (false);
            BookwormApp.Info.info_box.set_visible (false);
            prefButton.set_visible (false);
            BookwormApp.AppHeaderBar.bookmark_active_button.set_visible (false);
            BookwormApp.AppHeaderBar.bookmark_inactive_button.set_visible (false);
            if (!isBookBeingAddedToLibrary) {
                BookwormApp.AppWindow.bookAdditionBar.hide ();
            }
        }
        //Set-up UI for Reading Mode
        if (BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[1]) {
            BookwormApp.AppHeaderBar.headerSearchBar.set_placeholder_text (
                BookwormApp.Constants.TEXT_FOR_HEADERBAR_BOOK_SEARCH);
            library_mode_button.set_visible (false);
            content_list_button.set_visible (true);
            library_view_button.set_visible (true);
            library_view_button.set_label (BookwormApp.Constants.TEXT_FOR_LIBRARY_BUTTON);
            BookwormApp.AppWindow.bookLibrary_ui_box.set_visible (false);
            BookwormApp.AppWindow.bookReading_ui_box.set_visible (true);
            BookwormApp.Info.info_box.set_visible (false);
            prefButton.set_visible (true);
            BookwormApp.contentHandler.handleBookMark ("DISPLAY");
            BookwormApp.AppWindow.bookAdditionBar.hide ();
        }
        //Set-up UI for Info Mode -  Meta Data / Table of Content
        if (BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[4]) {
            BookwormApp.AppHeaderBar.headerSearchBar.set_placeholder_text (
                BookwormApp.Constants.TEXT_FOR_HEADERBAR_LIBRARY_SEARCH);
            BookwormApp.Info.info_box.show_all ();
            library_mode_button.set_visible (false);
            content_list_button.set_visible (false);
            library_view_button.set_visible (true);
            library_view_button.set_label (BookwormApp.Constants.TEXT_FOR_RESUME_BUTTON);
            BookwormApp.AppWindow.bookLibrary_ui_box.set_visible (false);
            BookwormApp.AppWindow.bookReading_ui_box.set_visible (false);
            BookwormApp.Info.info_box.set_visible (true);
            BookwormApp.Info.stack.set_visible_child_name (settings.current_info_tab);
            prefButton.set_visible (false);
            BookwormApp.AppHeaderBar.bookmark_active_button.set_visible (false);
            BookwormApp.AppHeaderBar.bookmark_inactive_button.set_visible (false);
            BookwormApp.AppWindow.bookAdditionBar.hide ();
        }
        info ("[END] [FUNCTION:toggleUIState] bookworm current state:" + BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE);
    }

    public static BookwormApp.Book genericParser (owned BookwormApp.Book aBook) {
        info ("[START] [FUNCTION:genericParser] book.location=" + aBook.getBookLocation ());
        //set defaults for title and author - this will be over-ridden with extracted data if found
        aBook.setBookAuthor (BookwormApp.Constants.TEXT_FOR_UNKNOWN_TITLE);
        aBook.setBookTitle (BookwormApp.Constants.TEXT_FOR_UNKNOWN_TITLE);
        //check if ebook is present at provided location
        if ("false" == BookwormApp.Utils.fileOperations ("EXISTS", "", aBook.getBookLocation (), "")) {
            warning ("EBook not found at provided location:" + aBook.getBookLocation ());
            aBook.setIsBookParsed (false);
            aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_EXTRACTION_ISSUE);
            return aBook;
        }
        //determine the extension of the ebook file
        string ebookFileName = (File.new_for_path (aBook.getBookLocation ()).get_basename ());
        if (ebookFileName.index_of (".") != -1) {
            string fileExtension = ebookFileName.substring (ebookFileName.last_index_of (".")).up ();
            //parse file based on extension found
            try {
                switch (fileExtension) {
                    case ".EPUB":
                        aBook = BookwormApp.ePubReader.parseEPubBook (aBook);
                        break;
                    case ".PDF":
                        aBook = BookwormApp.pdfReader.parsePDFBook (aBook);
                        break;
                    case ".CBR":
                        aBook = BookwormApp.comicsReader.parseComicsBook (aBook, fileExtension);
                        break;
                    case ".CBZ":
                        aBook = BookwormApp.comicsReader.parseComicsBook (aBook, fileExtension);
                        break;
                    case ".MOBI":
                        aBook = BookwormApp.mobiReader.parseMobiBook (aBook);
                        break;
                    case ".PRC":
                        aBook = BookwormApp.mobiReader.parseMobiBook (aBook);
                        break;
                    case ".AZW3":
                        aBook = BookwormApp.mobiReader.parseMobiBook (aBook);
                        break;
                    case ".ZIP":
                        //check if the file is a zipped FB2 file
                        if (ebookFileName.up ().last_index_of (".FB2.ZIP") != -1) {
                            aBook = BookwormApp.fb2Reader.parseFictionBook (aBook);
                            break;
                        } else {
                            aBook.setIsBookParsed (false);
                            aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_FORMAT_NOT_SUPPORTED);
                            break;
                        }
                    case ".FB2":
                        aBook = BookwormApp.fb2Reader.parseFictionBook (aBook);
                        break;
                    default:
                        aBook.setIsBookParsed (false);
                        aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_FORMAT_NOT_SUPPO
RTED);
                            break;
                    }
                } catch (GLib.Error e) {
                    info ("Error while parsing book: %s\n", e.message);
                    aBook.setIsBookParsed (false);
                    aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_PARSING_ISSUE);
                }
            }
            //If the book could not be parsed, add the book location to the list of books which could not be loaded
            if (!aBook.getIsBookParsed () ) {
                BookwormApp.Bookworm.pathsOfBooksNotAddedStr.append (aBook.getBookLocation ()).append ("~~");
            }
            info ("[END] [FUNCTION:genericParser] book.is book parsed=" + aBook.getIsBookParsed ().to_string ());
            return aBook;
        }
    }
```

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

```
/* Copyright 2017 Siddhartha Das (bablu.boy@gmail.com)
*
* This file is part of Bookworm and has the getter/setter methods
* used for holding the state of the book
*
* Bookworm is free software: you can redistribute it
* and/or modify it under the terms of the GNU General Public License as
* published by the Free Software Foundation, either version 3 of the
* License, or (at your option) any later version.
*
* Bookworm is distributed in the hope that it will be
* useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
* Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with Bookworm. If not, see http://www.gnu.org/licenses/.
*/
using Gtk;
using Gee;
public class BookwormApp.Book {
    //These variables are persisted in the database
    private int bookId = 0;
    private bool isBookParsedCorrectly = false;
    private string parsingIssue = "";
    private string bookLocation = "";
    private string bookCoverLocation = "";
    private string bookExtractionLocation = "";
    private string bookTitle = "";
    private string bookAuthor = "";
    private string bookTags = "";
    private string annotationTags = "";
    private string bookPublishDate = "";
    private string bookCreationDate = "";
    private string bookLastModificationDate = "";
    private int bookPageNumber = -1;
    private int bookTotalPages = 1;
    private int bookScrollPosition = -1;
    private int bookRating = 0;
    private bool isBookCoverImagePresent = false;
    private StringBuilder bookmarks = new StringBuilder ("");
    private TreeMap<string,string> annotationMap = new TreeMap<string,string> ();
    private ArrayList<string> bookContentList = new ArrayList<string> ();
    private ArrayList<HashMap<string,string>> TOCMap = new ArrayList<HashMap<string,string>> ();

    //These variables are only available for the current session (not persisted)
    private string opfFileLocation = "";
    private string baseLocationOfContents = "";
    private bool ifPageForward = true;
    private bool ifPageBackward = true;
    private bool isBookSelected = false;
    private bool wasBookOpened = false;
    private string bookAnchor = "";
    private HashMap<string,Gtk.Widget> bookWidgetsList = new HashMap<string,Gtk.Widget> ();

    //getter list for book id
    public void setBookId (int aBookId) {
        bookId = aBookId;
    }
    public int getBookId () {
        return bookId;
    }

    //getter list for isBookParsedCorrectly
    public void setIsBookParsed (bool isParsed) {
        isBookParsedCorrectly = isParsed;
    }
    public bool getIsBookParsed () {
        return isBookParsedCorrectly;
    }

    //getter list for book location
    public void setParsingIssue (string aParsingIssue) {
        parsingIssue = aParsingIssue;
    }
    public string getParsingIssue () {
        return parsingIssue;
    }

    //getter list for book location
    public void setBookLocation (string aBookLocation) {
        bookLocation = aBookLocation;
    }
    public string getBookLocation () {
        return bookLocation;
    }

    //getter list for book cover image location
    public void setBookCoverLocation (string aBookCoverLocation) {
        bookCoverLocation = aBookCoverLocation;
    }
    public string getBookCoverLocation () {
        return bookCoverLocation;
    }

    //getter setter for content list of book parts
    public void setBookContentList (string contentList) {
        bookContentList.add (contentList);
    }
    public ArrayList<string> getBookContentList () {
        return bookContentList;
    }
    public void clearBookContentList () {
        bookContentList.clear ();
    }

    //getter setter for Table Of Contents
    public void setTOC (HashMap<string,string> toc) {
        TOCMap.add (toc);
    }
    public ArrayList<HashMap<string,string>> getTOC () {
        return TOCMap;
    }
    public void clearTOC () {
        TOCMap.clear ();
    }

    //getter setter for temp location of ebook contents
    public void setBookExtractionLocation (string aBookExtractionLocation) {
        bookExtractionLocation = aBookExtractionLocation;
    }
    public string getBookExtractionLocation () {
        return bookExtractionLocation;
    }

    //getter setter for book title
    public void setBookTitle (string aBookTitle) {
        bookTitle = aBookTitle;
    }
    public string getBookTitle () {
        return bookTitle;
    }

    //getter list for book rating
    public void setBookRating (int aBookRating) {
        bookRating = aBookRating;
    }
    public int getBookRating () {
        return bookRating;
    }

    //getter setter for book author
    public void setBookAuthor (string aBookAuthor) {
        bookAuthor = aBookAuthor;
    }
    public string getBookAuthor () {
        return bookAuthor;
    }

    //getter setter for book tags
    public void setBookTags (string aBookTags) {
        bookTags = aBookTags;
    }
    public string getBookTags () {
        return bookTags;
    }

    //getter setter for annotation tags
    public void setAnnotationTags (string anAnnotationTags) {
        annotationTags = anAnnotationTags;
    }
    public string getAnnotationTags () {
        return annotationTags;
    }

    //getter setter for location of books OPF file
    public void setOPFFileLocation (string aOPFFileLocation) {
        opfFileLocation = aOPFFileLocation;
    }
    public string getOPFFileLocation () {
        return opfFileLocation;
    }

    //getter setter for base location of eBook file contents
    public void setBaseLocationOfContents (string aBaseLocationOfContents) {
        baseLocationOfContents = aBaseLocationOfContents;
    }
    public string getBaseLocationOfContents () {
        return baseLocationOfContents;
    }

    //getter setter for presence of Cover Location
    public void setIsBookCoverImagePresent (bool isABookCoverImagePresent) {
        isBookCoverImagePresent = isABookCoverImagePresent;
    }
    public bool getIsBookCoverImagePresent () {
        return isBookCoverImagePresent;
    }

    //getter list for book location
    public void setBookPublishDate (string aBookPublishDate) {
        bookPublishDate = aBookPublishDate;
    }
    public string getBookPublishDate () {
        return bookPublishDate;
    }

    //getter list for book location
    public void setBookCreationDate (string aBookCreationDate) {
        bookCreationDate = aBookCreationDate;
    }
    public string getBookCreationDate () {
        return bookCreationDate;
    }

    //getter list for book location
    public void setBookLastModificationDate (string aBookLastModificationDate) {
        bookLastModificationDate = aBookLastModificationDate;
    }
    public string getBookLastModificationDate () {
        return bookLastModificationDate;
    }

    //getter setter for eBook pageNumber
    public void setBookPageNumber (int aBookPageNumber) {
        bookPageNumber = aBookPageNumber;
    }
    public int getBookPageNumber () {
        return bookPageNumber;
    }

    //getter setter for total pages in book
    public void setBookTotalPages (int aBookTotalPages) {
        bookTotalPages = aBookTotalPages;
    }
    public int getBookTotalPages () {
        return bookTotalPages;
    }

    //getter setter for eBook vertical scroll position
    public void setBookScrollPos (int aBookScrollPos) {
        bookScrollPosition = aBookScrollPos;
    }
    public int getBookScrollPos () {
        return bookScrollPosition;
    }

    //getter setter if eBook pageForward is possible
    public void setIfPageForward (bool ifBookPageForward) {
        ifPageForward = ifBookPageForward;
    }
    public bool getIfPageForward () {
        return ifPageForward;
    }

    //getter setter if eBook pageBackward is possible
    public void setIfPageBackward (bool ifBookPageBackward) {
        ifPageBackward = ifBookPageBackward;
    }
    public bool getIfPageBackward () {
        return ifPageBackward;
    }

    //getter setter for determining if the book is selected
    public void setIsBookSelected (bool aIsBookSelected) {
        isBookSelected = aIsBookSelected;
    }
    public bool getIsBookSelected () {
        return isBookSelected;
    }

    //getter setter for determining if the book was read in this session
    public void setWasBookOpened (bool aWasBookOpened) {
        wasBookOpened = aWasBookOpened;
    }
    public bool getWasBookOpened () {
        return wasBookOpened;
    }

    //getter setter for bookmarks
    public void setBookmark (int pageNumber, string action) {
        if ("ACTIVE_CLICKED" == action) {
            bookmarks.assign (bookmarks.str.replace ("**" + pageNumber.to_string () + "**", ""));
        }
        if ("INACTIVE_CLICKED" == action) {
            bookmarks.append ("**" + pageNumber.to_string () + "**");
        }
        if (pageNumber == -10) { //this is used to set the bookmark fetched from the DB
            //set -10 as the "pageNumber" and the book mark data as "action" when setting this value from the DB
            bookmarks.assign (action);
        }
    }
    public string getBookmark () {
        return bookmarks.str;
    }

    //getter setter for annotations
    public void setAnnotations (string index, string annotationText) {
        //check the value of the annotation text - add or delete accordingly
        if (annotationText != null && annotationText.length > 0) {
            //annotated text is not null/empty - update the annotation
            annotationMap.set (index, annotationText);
        } else {
            //annotated text is null/empty - remove the annotation
            if (annotationMap.has_key (index)) {
                annotationMap.unset (index);
            }
        }
    }
    public string getAnnotations (string index) {
        if (annotationMap.has_key (index)) {
            return annotationMap.get (index);
        } else {
            return "";
        }
    }
    public TreeMap<string,string> getAnnotationList () {
        return annotationMap;
    }
    public void setAnnotationList (TreeMap<string,string> aTreeMap) {
        annotationMap.set_all (aTreeMap);
    }

    //getter setter for list of Gtk Widgets used for a Book
    public void setBookWidget (string name, Gtk.Widget aWidget) {
        bookWidgetsList.set (name, aWidget);
    }
    public Gtk.Widget getBookWidget (string name) {
        return bookWidgetsList.get (name);
    }

    //getter setter for book anchor
    public void setAnchor (string aBookAnchor) {
        bookAnchor = aBookAnchor;
    }
    public string getAnchor () {
        return bookAnchor;
    }

    //print book details
    public string to_string () {
        StringBuilder bookDetails = new StringBuilder ();
        bookDetails.append ("bookId=").append (bookId.to_string ()).append (",\n")
            .append ("bookLocation=").append (bookLocation).append (",\n")
            .append ("bookCoverLocation=").append (bookCoverLocation).append (",\n")
            .append ("bookExtractionLocation=").append (bookExtractionLocation).append (",\n")
            .append ("bookTitle=" + bookTitle).append (",\n")
            .append ("opfFileLocation=").append (opfFileLocation).append (",\n")
            .append ("baseLocationOfContents=").append (baseLocationOfContents).append (",\n")
            .append ("bookPublishDate=" + bookPublishDate).append (",\n")
            .append ("isBookCoverImagePresent=").append (isBookCoverImagePresent.to_string ()).append (",\n")
            .append ("bookCreationDate=").append (bookCreationDate).append (",\n")
            .append ("bookLastModificationDate=").append (bookLastModificationDate).append (",\n")
            .append ("bookPageNumber=").append (bookPageNumber.to_string ()).append (",\n")
            .append ("ifPageForward=").append (ifPageForward.to_string ()).append (",\n")
            .append ("ifPageBackward=").append (ifPageBackward.to_string ()).append (",\n")
            .append ("bookmarks=").append (bookmarks.str).append (",\n")
            .append ("author=").append (bookAuthor).append (",\n")
            .append ("ratings=").append (bookRating.to_string ()).append (",\n")
            .append ("tags=").append (bookTags.to_string ()).append (",\n")
            .append ("annotation tags=").append (annotationTags.to_string ()).append (",\n")
            .append ("bookContentList=");
        for (int i = 0; i < bookContentList.size; i++) {
            bookDetails.append ("[" + i.to_string () + "]=" + bookContentList.get (i) + ",");
        }
        return bookDetails.str;
    }
}
```

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

```
/* Copyright 2017 Siddhartha Das (bablu.boy@gmail.com)
*
* This file is part of Bookworm and manages the library and
* views associated with the library
*
* Bookworm is free software: you can redistribute it
* and/or modify it under the terms of the GNU General Public License as
* published by the Free Software Foundation, either version 3 of the
* License, or (at your option) any later version.
*
* Bookworm is distributed in the hope that it will be
* useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
* Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with Bookworm. If not, see http://www.gnu.org/licenses/.
*/
using Gtk;
using Gee;
public class BookwormApp.Library {
    public static ArrayList<BookwormApp.Book> listOfBooksInLibraryOnLoad = new ArrayList<BookwormApp.Book> ();

    public static void updateLibraryView (owned BookwormApp.Book aBook) {
        info ("[START] [FUNCTION:updateLibraryView]");
        //add book details to libraryView Map
        BookwormApp.Bookworm.libraryViewMap.set (aBook.getBookLocation (), aBook);
        //update the library views
        updateLibraryListView (aBook);
        updateLibraryGridView (aBook);
        info ("[END] [FUNCTION:updateLibraryView]");
    }

    public static void updateLibraryListView (owned BookwormApp.Book aBook) {
        debug ("[START] [FUNCTION:updateLibraryListView] book.location=" + aBook.getBookLocation ());
        //if (aBook.getBookTitle != null && aBook.getBookTitle ().length > 1) {
            debug ("Started updating Library List View for book:" + aBook.getBookLocation ());
            //set the rating image
            Gdk.Pixbuf image_rating;
            string modifiedElapsedTime = "";
            switch (aBook.getBookRating ().to_string ()) {
                case "1":
                    image_rating = BookwormApp.Bookworm.image_rating_1;
                    break;
                case "2":
                    image_rating = BookwormApp.Bookworm.image_rating_2;
                    break;
                case "3":
                    image_rating = BookwormApp.Bookworm.image_rating_3;
                    break;
                case "4":
                    image_rating = BookwormApp.Bookworm.image_rating_4;
                    break;
                case "5":
                    image_rating = BookwormApp.Bookworm.image_rating_5;
                    break;
                default:
                    image_rating = null;
                    break;
            }
            //calculate the time elapsed from last modified DateTime
            TimeSpan timespan = (new DateTime.now_local ()).difference (
                new DateTime.from_unix_local (int64.parse (aBook.getBookLastModificationDate ())));
            int64 daysElapsed = timespan/ (86400000000);
            if (timespan < TimeSpan.DAY) {
                modifiedElapsedTime = BookwormApp.Constants.TEXT_FOR_TIME_TODAY;
            } else if (timespan < 2 * TimeSpan.DAY) {
                modifiedElapsedTime = BookwormApp.Constants.TEXT_FOR_TIME_YESTERDAY;
            } else if (timespan < 30 * TimeSpan.DAY) {
                modifiedElapsedTime = daysElapsed.to_string () + " " + BookwormApp.Constants.TEXT_FOR_TIME_DAYS;
            } else {
                modifiedElapsedTime = new DateTime.from_unix_local (int64.parse (aBook.getBookLastModificationDate ())).format ("%d %m %Y");
            }
            BookwormApp.AppWindow.library_table_liststore.append (out BookwormApp.AppWindow.library_table_iter);
            BookwormApp.AppWindow.library_table_liststore.set (BookwormApp.AppWindow.library_table_iter,
                0, null,
                1, BookwormApp.Utils.parseMarkUp (aBook.getBookTitle ()),
                2, aBook.getBookAuthor (),
                3, modifiedElapsedTime,
                4, image_rating,
                5, aBook.getBookTags (),
                6, aBook.getBookRating ().to_string (),
                7, aBook.getBookLocation ());
            BookwormApp.Bookworm.libraryTreeModelFilter = new Gtk.TreeModelFilter (BookwormApp.AppWindow.library_table_liststore, null);
            //BookwormApp.Bookworm.libraryTreeModelFilter.set_visible_func (filterTree);
            Gtk.TreeModelSort aTreeModelSort = new TreeModelSort.with_model (BookwormApp.Bookworm.libraryTreeModelFilter);
            BookwormApp.AppWindow.library_table_treeview.set_model (aTreeModelSort);
            //set treeview columns for sorting
            BookwormApp.AppWindow.library_table_treeview.get_column (1).set_sort_column_id (1);
            BookwormApp.AppWindow.library_table_treeview.get_column (1).set_sort_order (SortType.DESCENDING);
            BookwormApp.AppWindow.library_table_treeview.get_column (2).set_sort_column_id (2);
            BookwormApp.AppWindow.library_table_treeview.get_column (2).set_sort_order (SortType.DESCENDING);
            BookwormApp.AppWindow.library_table_treeview.get_column (3).set_sort_column_id (3);
            BookwormApp.AppWindow.library_table_treeview.get_column (3).set_sort_order (SortType.DESCENDING);
            //6th item is the rating value corresponding to the image on the 4th item
            BookwormApp.AppWindow.library_table_treeview.get_column (4).set_sort_column_id (6);
            BookwormApp.AppWindow.library_table_treeview.get_column (4).set_sort_order (SortType.DESCENDING);
            BookwormApp.AppWindow.library_table_treeview.get_column (5).set_sort_column_id (5);
            BookwormApp.AppWindow.library_table_treeview.get_column (5).set_sort_order (SortType.DESCENDING);
        //}
        debug ("[END] [FUNCTION:updateLibraryListView] book.location=" + aBook.getBookLocation ());
    }

    public static void updateLibraryGridView (owned BookwormApp.Book aBook) {
        debug ("[START] [FUNCTION:updateLibraryGridView] book.location=" + aBook.getBookLocation ());
        debug ("Started updating Library Grid View for book:" + aBook.getBookLocation ());
        Gtk.Image aCoverImage = new Gtk.Image ();
        Gtk.Label titleTextLabel = new Gtk.Label ("");
        Gtk.Image bookSelectionImage;
        Gtk.Image bookSelectedImage;
        string bookCoverLocation;
        Gdk.Pixbuf aBookCover;
        Gtk.Image bookPlaceholderCoverImage = null;
        try {
            Gdk.Pixbuf bookPlaceholderCoverPix = new Gdk.Pixbuf.from_resource_at_scale (
                BookwormApp.Constants.PLACEHOLDER_COVER_IMAGE_LOCATION, 10, 200, false);
            bookPlaceholderCoverImage = new Gtk.Image.from_pixbuf (bookPlaceholderCoverPix);
        } catch (GLib.Error e) {
            warning ("Error loading the placeholder cover image from location[" +
                BookwormApp.Constants.PLACEHOLDER_COVER_IMAGE_LOCATION + "] : " +
                e.message );
        }
        Gtk.ProgressBar bookProgressBar = new Gtk.ProgressBar ();
        //Add a default cover selected at random if no cover exists
        if (aBook.getBookCoverLocation () == null ||
            aBook.getBookCoverLocation ().length < 1 ||
            "true" != BookwormApp.Utils.fileOperations ("EXISTS", "", aBook.getBookCoverLocation (), ""))
        {
            aBook.setIsBookCoverImagePresent (false);
            //default Book Cover Image not set - select at random from the default covers
            bookCoverLocation = BookwormApp.Constants.DEFAULT_COVER_IMAGE_LOCATION
                .replace ("N", GLib.Random.int_range (1, 6).to_string ());
            aBook.setBookCoverLocation (bookCoverLocation);
            try {
                aBookCover = new Gdk.Pixbuf.from_resource_at_scale (aBook.getBookCoverLocation (), 150, 200, false);
                aCoverImage = new Gtk.Image.from_pixbuf (aBookCover);
            } catch (GLib.Error e) {
                warning ("Error in loading default cover image at location [" + aBook.getBookCoverLocation () + "]: " + e.message);
            }
        } else {
            try {
                aBookCover = new Gdk.Pixbuf.from_file_at_scale (aBook.getBookCoverLocation (), 150, 200, false);
                aCoverImage = new Gtk.Image.from_pixbuf (aBookCover);
            } catch (GLib.Error e) {
                //Sometimes the path to the image selected by the parser is not a image
                //This catch block assigns a default cover selected at random to cover this issue
                bookCoverLocation = BookwormApp.Constants.DEFAULT_COVER_IMAGE_LOCATION
                    .replace ("N", GLib.Random.int_range (1, 6).to_string ());
                aBook.setBookCoverLocation (bookCoverLocation);
                aCoverImage = null;
                try {
                    aBookCover = new Gdk.Pixbuf.from_resource_at_scale (aBook.getBookCoverLocation (), 150, 200, false);
                    aCoverImage = new Gtk.Image.from_pixbuf (aBookCover);
                    //set cover image present flag to false - this will add title text to the default cover
                    aBook.setIsBookCoverImagePresent (false);
                    aCoverImage.set_halign (Align.CENTER);
                    aCoverImage.set_valign (Align.CENTER);
                } catch (GLib.Error e) {
                    warning ("Error in loading cover image at location [" + aBook.getBookCoverLocation () + "]: " + e.message);
                }
            }
        }
        //Add title of the book if Default Cover is being used
        if (!aBook.getIsBookCoverImagePresent ()) {
            string title = aBook.getBookTitle ();
            if (title == null || title.length < 1) {
                title = BookwormApp.Constants.TEXT_FOR_UNKNOWN_TITLE;
            }
            //replace special chars from title
            title = title.replace ("&", "and");
            title = BookwormApp.Utils.minimizeStringLength (title, 4 * BookwormApp.Constants.MAX_NUMBER_OF_CHARS_FOR_BOOK_TITLE);
            titleTextLabel.set_text (
                "<b>" + BookwormApp.Utils.breakString (title, BookwormApp.Constants.MAX_NUMBER_OF_CHARS_FOR_BOOK_TITLE, "\n") + "</b>");
            titleTextLabel.set_use_markup (true);
            titleTextLabel.set_line_wrap (true);
            titleTextLabel.set_justify (Justification.CENTER);
            titleTextLabel.set_margin_start (BookwormApp.Constants.SPACING_WIDGETS);
            titleTextLabel.set_margin_end (BookwormApp.Constants.SPACING_WIDGETS);
        } else {
            //remove the title label if the book has a cover image available
            titleTextLabel.set_text ("");
        }
        //Add selection option badge to the book for later use
        Gdk.Pixbuf bookSelectionPix = null;
        try {
            bookSelectionPix = new Gdk.Pixbuf.from_resource (BookwormApp.Constants.SELECTION_OPTION_IMAGE_LOCATION);
        } catch (GLib.Error e) {
            warning ("Error in loading Book selection image from location [" +
                BookwormApp.Constants.SELECTION_OPTION_IMAGE_LOCATION + "]: " + e.message);
        }
        bookSelectionImage = new Gtk.Image.from_pixbuf (bookSelectionPix);
        bookSelectionImage.set_halign (Align.CENTER);
        bookSelectionImage.set_valign (Align.START);
        //Add selection checked badge to the book for later use
        Gdk.Pixbuf bookSelectedPix = null;
        try {
            bookSelectedPix = new Gdk.Pixbuf.from_resource (BookwormApp.Constants.SELECTION_CHECKED_IMAGE_LOCATION);
        } catch (GLib.Error e) {
            warning ("Error in loading Book Selection Checked image from location[" +
                BookwormApp.Constants.SELECTION_CHECKED_IMAGE_LOCATION + "] :" + e.message);
        }
        bookSelectedImage = new Gtk.Image.from_pixbuf (bookSelectedPix);
        bookSelectedImage.set_halign (Align.CENTER);
        bookSelectedImage.set_valign (Align.START);
        //Set the value of the progress bar
        double progress = 0.0;
        bookProgressBar.set_halign (Align.CENTER);
        bookProgressBar.set_valign (Align.END);
        bookProgressBar.set_visible (false);
        //protect the progress bar against the show_all called on the library view
        bookProgressBar.set_no_show_all (true);
        //Create a Overlay to hold the images in the right order
        Gtk.Overlay aOverlayImage = new Gtk.Overlay ();
        aOverlayImage.add (bookPlaceholderCoverImage);
        aOverlayImage.add_overlay (bookSelectionImage);
        aOverlayImage.add_overlay (bookSelectedImage);
        aOverlayImage.add_overlay (aCoverImage);
        aOverlayImage.add_overlay (titleTextLabel);
        aOverlayImage.add_overlay (bookProgressBar);//this will be invisble until mouse enters
        //Add the overlaid images to a EventBox to allow mouse click actions to be captured
        Gtk.EventBox aEventBox = new Gtk.EventBox ();
        aEventBox.set_border_width (BookwormApp.Constants.SPACING_WIDGETS/2);
        aEventBox.set_name (aBook.getBookLocation ());
        aEventBox.add (aOverlayImage);
        //register the book with the filter function
        var aFlowBoxChild = new Gtk.FlowBoxChild ();
        aFlowBoxChild.add (aEventBox);
        //add the book to the library view
        BookwormApp.AppWindow.library_grid.add (aFlowBoxChild);
        //set gtk widgets into the Book object for later manipulation
        aBook.setBookWidget ("PLACEHOLDER_COVER_IMAGE", bookPlaceholderCoverImage);
        aBook.setBookWidget ("COVER_IMAGE", aCoverImage);
        aBook.setBookWidget ("TITLE_TEXT_LABEL", titleTextLabel);
        aBook.setBookWidget ("SELECTED_BADGE_IMAGE", bookSelectedImage);
        aBook.setBookWidget ("SELECTION_BADGE_IMAGE", bookSelectionImage);
        aBook.setBookWidget ("BOOK_EVENTBOX", aEventBox);
        aBook.setBookWidget ("BOOK_OVERLAY_IMAGE", aOverlayImage);
        //Create a popover context menu for the book
        Gtk.Popover bookPopover = BookwormApp.AppDialog.createBookContextMenu (aBook);
        //add mouse enter listener for book object
        aEventBox.enter_notify_event.connect ((event) => {
            //calculate the progress of the book
            progress = ((double)aBook.getBookPageNumber () + 1)/aBook.getBookTotalPages ();
            bookProgressBar.set_fraction (progress);
            bookProgressBar.set_visible (true);
            return false;
        });
        //add mouse exit listener for book object
        aEventBox.leave_notify_event.connect ((event) => {
            //Checking for Gdk.NotifyType.INFERIOR resolves the unwanted leave event fired due to the cover being a default type image
            if (event.detail != Gdk.NotifyType.INFERIOR) {
                bookProgressBar.set_visible (false);
            }
            return false;
        });
        //add mouse click listener for book objects based on mode
        aEventBox.button_press_event.connect ((event) => {
            //capture which mouse button was clicked on the book in the library
            uint mouseButtonClicked;
            event.get_button (out mouseButtonClicked);
            //handle right button click for context menu
            if (event.get_event_type () == Gdk.EventType.BUTTON_PRESS && mouseButtonClicked == 3) {
                bookPopover.set_visible (true);
                bookPopover.show_all ();
                return true;
            } else {
                //left button click for reading or selection of book
                if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[0]) {
                    aBook = BookwormApp.Bookworm.libraryViewMap.get (aEventBox.get_name ());
                    BookwormApp.Bookworm.readSelectedBook (aBook);
                }
                if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[2] ||
                    BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[3])
                {
                    BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_UI_STATES[3];
                    aBook = BookwormApp.Bookworm.libraryViewMap.get (aEventBox.get_name ());
                    updateGridViewForSelection (aBook);
                }
                return true;
            }
        });
        //Show the grid view based on state
        if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[0] ||
            BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[2] ||
            BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[3])
        {
            BookwormApp.AppWindow.library_grid.show_all ();
        }
        debug ("Completed updating Library View for book:" + aBook.getBookLocation ());
    }

    public static void replaceCoverImageOnBook (owned BookwormApp.Book? book) {
        debug ("[START] [FUNCTION:replaceCoverImageOnBook]");
        //remove the existing overlay image
        Gtk.Overlay oldOverlayImage = (Gtk.Overlay) book.getBookWidget ("BOOK_OVERLAY_IMAGE");
        oldOverlayImage.destroy ();
        //create a new overlay image
        Gtk.Overlay lOverlayImage = new Gtk.Overlay ();
        lOverlayImage.add (book.getBookWidget ("PLACEHOLDER_COVER_IMAGE"));
        lOverlayImage.add_overlay (book.getBookWidget ("SELECTION_BADGE_IMAGE"));
        lOverlayImage.add_overlay (book.getBookWidget ("SELECTED_BADGE_IMAGE"));
        lOverlayImage.add_overlay (book.getBookWidget ("COVER_IMAGE"));
        lOverlayImage.add_overlay (book.getBookWidget ("TITLE_TEXT_LABEL"));
        book.setBookWidget ("BOOK_OVERLAY_IMAGE", lOverlayImage);
        //associate the eventbox with the new overlay image
        Gtk.EventBox aEventBox = (Gtk.EventBox) book.getBookWidget ("BOOK_EVENTBOX");
        aEventBox.add (lOverlayImage);
        book.setBookWidget ("BOOK_EVENTBOX", aEventBox);
        //update the libraryview map with the book object
        BookwormApp.Bookworm.libraryViewMap.set (book.getBookLocation (), book);
        debug ("[END] [FUNCTION:replaceCoverImageOnBook]");
    }

    public static void updateListViewForSelection (owned BookwormApp.Book? lBook) {
        debug ("[START] [FUNCTION:updateListViewForSelection] " +
            "Updating List View Selection Badges for mode:" +
            BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE);
        Gtk.TreeModelForeachFunc print_row = (model, path, iter) => {
            GLib.Value bookLocationAtRow;
            BookwormApp.AppWindow.library_table_liststore.get_value (iter, 7, out bookLocationAtRow);
            BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap.get ((string) bookLocationAtRow);
            if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[5]) {
                BookwormApp.AppWindow.library_table_liststore.set_value (iter, 0, BookwormApp.Bookworm.image_selection_transparent_small);
                aBook.setIsBookSelected (false);
                BookwormApp.AppWindow.controlDeletionButton (false);
            }
            if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[6]) {
                BookwormApp.AppWindow.library_table_liststore.set_value (iter, 0, BookwormApp.Bookworm.image_selection_option_small);
                aBook.setIsBookSelected (false);
                BookwormApp.AppWindow.controlDeletionButton (false);
            }
            if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[7] &&
                (string) bookLocationAtRow == lBook.getBookLocation ())
            {
                if (!lBook.getIsBookSelected ()) {
                    BookwormApp.AppWindow.library_table_liststore.set_value (iter, 0, BookwormApp.Bookworm.image_selection_checked_small);
                    aBook.setIsBookSelected (true);
                    BookwormApp.AppWindow.controlDeletionButton (true);
                } else {
                    BookwormApp.AppWindow.library_table_liststore.set_value (iter, 0, BookwormApp.Bookworm.image_selection_option_small);
                    aBook.setIsBookSelected (false);
                    BookwormApp.AppWindow.controlDeletionButton (false);
                }
            }
            //update the book into the Library view HashMap
            BookwormApp.Bookworm.libraryViewMap.set (aBook.getBookLocation (), aBook);
            return false;
        };
        BookwormApp.AppWindow.library_table_liststore.foreach (print_row);
        debug ("[END] [FUNCTION:updateListViewForSelection]");
    }

    public static void updateGridViewForSelection (owned BookwormApp.Book? lBook) {
        debug ("[START] [FUNCTION:updateGridViewForSelection]");
        if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[0]) {
            debug ("Updating Library View for Selection Badges BOOKWORM_UI_STATES[0]");
            Gee.HashMap<string, BookwormApp.Book> temp_libraryViewMap = new Gee.HashMap<string, BookwormApp.Book> ();
            //loop over HashMap of Book Objects and overlay selection image
            foreach (BookwormApp.Book book in BookwormApp.Bookworm.libraryViewMap.values) {
                if (BookwormApp.AppWindow.library_grid_scroll.get_visible ()) {
                Gtk.Overlay aOverlayImage = (Gtk.Overlay) book.getBookWidget ("BOOK_OVERLAY_IMAGE");
                    //Align the selection badges to the center so that they are not visible
                    book.getBookWidget ("SELECTION_BADGE_IMAGE").set_halign (Align.CENTER);
                    book.getBookWidget ("SELECTED_BADGE_IMAGE").set_halign (Align.CENTER);
                    //set the order of the widgets to put the selection/selected badges at bottom
                    aOverlayImage.reorder_overlay (book.getBookWidget ("SELECTION_BADGE_IMAGE"), 1);
                    aOverlayImage.reorder_overlay (book.getBookWidget ("SELECTED_BADGE_IMAGE"), 2);
                    aOverlayImage.reorder_overlay (book.getBookWidget ("COVER_IMAGE"), 3);
                    aOverlayImage.reorder_overlay (book.getBookWidget ("TITLE_TEXT_LABEL"), 4);
                }
                temp_libraryViewMap.set (book.getBookLocation (), book);
            }
            //Iterate over all books and make the selection flag for each book as false
            //This is to cover the scenario when a book was selected and the selection mode was changed without deleting the book
            foreach (BookwormApp.Book aBook in temp_libraryViewMap.values) {
                aBook.setIsBookSelected (false);
                BookwormApp.Bookworm.libraryViewMap.set (aBook.getBookLocation (), aBook);
            }
        }
        if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[2]) {
            debug ("Updating Library View for Selection Badges BOOKWORM_UI_STATES[2]");
            Gee.HashMap<string, BookwormApp.Book> temp_libraryViewMap = new Gee.HashMap<string, BookwormApp.Book> ();
            //loop over HashMap of Book Objects and overlay selection badge
            foreach (BookwormApp.Book book in BookwormApp.Bookworm.libraryViewMap.values) {
                if (BookwormApp.AppWindow.library_grid_scroll.get_visible ()) {
                    Gtk.Overlay aOverlayImage = (Gtk.Overlay) book.getBookWidget ("BOOK_OVERLAY_IMAGE");
                    //Align the selection badges to the right to make visible but the selected badges should be centered to keep hidden
                    book.getBookWidget ("SELECTION_BADGE_IMAGE").set_halign (Align.START);
                    book.getBookWidget ("SELECTED_BADGE_IMAGE").set_halign (Align.CENTER);
                    //set the order of the widgets to put the selection badge on top
                    aOverlayImage.reorder_overlay (book.getBookWidget ("SELECTED_BADGE_IMAGE"), 1);
                    aOverlayImage.reorder_overlay (book.getBookWidget ("COVER_IMAGE"), 2);
                    aOverlayImage.reorder_overlay (book.getBookWidget ("TITLE_TEXT_LABEL"), 3);
                    aOverlayImage.reorder_overlay (book.getBookWidget ("SELECTION_BADGE_IMAGE"), 4);
                }
                temp_libraryViewMap.set (book.getBookLocation (), book);
            }
            //Iterate over all books and make the selection flag for each book as false
            //This is to cover the scenario when a book was selected and the selection mode was changed without deleting the book
            foreach (BookwormApp.Book aBook in temp_libraryViewMap.values) {
                aBook.setIsBookSelected (false);
                BookwormApp.Bookworm.libraryViewMap.set (aBook.getBookLocation (), aBook);
            }
        }
        if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[3]) {
            debug ("Updating Library View for Selection Badges BOOKWORM_UI_STATES[3]");
            if (lBook != null) {
                if (BookwormApp.AppWindow.library_grid_scroll.get_visible ()) {
                    Gtk.Overlay aOverlayImage = (Gtk.Overlay) lBook.getBookWidget ("BOOK_OVERLAY_IMAGE");
                    if (!lBook.getIsBookSelected ()) { //Do work for selecting this book
                        //Align the selected badges to the right to make visible but keep the selection badge centered to keep hidden
                        lBook.getBookWidget ("SELECTED_BADGE_IMAGE").set_halign (Align.START);
                        lBook.getBookWidget ("SELECTION_BADGE_IMAGE").set_halign (Align.CENTER);
                        //set the order of the widgets to put the selected badge on top
                        aOverlayImage.reorder_overlay (lBook.getBookWidget ("SELECTION_BADGE_IMAGE"), 1);
                        aOverlayImage.reorder_overlay (lBook.getBookWidget ("COVER_IMAGE"), 2);
                        aOverlayImage.reorder_overlay (lBook.getBookWidget ("TITLE_TEXT_LABEL"), 3);
                        aOverlayImage.reorder_overlay (lBook.getBookWidget ("SELECTED_BADGE_IMAGE"), 4);
                        lBook.setIsBookSelected (true);
                        BookwormApp.AppWindow.controlDeletionButton (true);
                    } else { //Do work for de-selecting this book
                        //set the order of the widgets to put the selection badge on top
                        //Align the selection badges to the right to make visible but keep the selected badges centered to keep hidden
                        lBook.getBookWidget ("SELECTION_BADGE_IMAGE").set_halign (Align.START);
                        lBook.getBookWidget ("SELECTED_BADGE_IMAGE").set_halign (Align.CENTER);
                        aOverlayImage.reorder_overlay (lBook.getBookWidget ("SELECTED_BADGE_IMAGE"), 1);
                        aOverlayImage.reorder_overlay (lBook.getBookWidget ("COVER_IMAGE"), 2);
                        aOverlayImage.reorder_overlay (lBook.getBookWidget ("TITLE_TEXT_LABEL"), 3);
                        aOverlayImage.reorder_overlay (lBook.getBookWidget ("SELECTION_BADGE_IMAGE"), 4);
                        lBook.setIsBookSelected (false);
                        BookwormApp.AppWindow.controlDeletionButton (false);
                    }
                }
                //update the book into the Library view HashMap
                BookwormApp.Bookworm.libraryViewMap.set (lBook.getBookLocation (), lBook);
            }
        }
        debug ("[END] [FUNCTION:updateGridViewForSelection]");
```

```
    }

    public static void removeSelectedBooksFromLibrary () {
            debug ("[START] [FUNCTION:removeSelectedBooksFromLibrary]");
        ArrayList<string> listOfBooksToBeRemoved = new ArrayList<string> ();
        //loop through the Library View Hashmap and remove the selected books from the Library View Model
        foreach (BookwormApp.Book book in BookwormApp.Bookworm.libraryViewMap.values) {
            //check if the book selection flag to true and add it to removal list
            if (book.getIsBookSelected ()) {
                //hold the books to be deleted in a list
                listOfBooksToBeRemoved.add (book.getBookLocation ());
                Gtk.EventBox lEventBox = (Gtk.EventBox) book.getBookWidget ("BOOK_EVENTBOX");
                //destroy the EventBox parent widget - this removes the book from the library grid
                lEventBox.get_parent ().destroy ();
                //destroy the EventBox widget
                lEventBox.destroy ();
                //remove the cover image if it exists (ignore default covers)
                if (book.getBookCoverLocation ().index_of (
                    BookwormApp.Constants.DEFAULT_COVER_IMAGE_LOCATION.replace ("-cover-N.svg", "")) == -1)
                {
                    BookwormApp.Utils.execute_sync_command ("rm \"" + book.getBookCoverLocation () + "\"");
                }
                    //update the onloadBookList - this is to enable re-adding the book within the same session
                    BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr
                        .assign (BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr.str
                        .replace (book.getBookLocation (), ""));
                    BookwormApp.Library.listOfBooksInLibraryOnLoad.remove (book);
            }
        }
        if (listOfBooksToBeRemoved.size > 0) {
            //loop through the rows in the treeview and remove the selected books
            ArrayList<Gtk.TreeIter?> listOfItersToBeRemoved = new ArrayList<Gtk.TreeIter?> ();
            Gtk.TreeModelForeachFunc print_row = (model, path, iter) => {
                GLib.Value bookLocationAtRow;
                BookwormApp.AppWindow.library_table_liststore.get_value (iter, 7, out bookLocationAtRow);
                if ((string) bookLocationAtRow in listOfBooksToBeRemoved) {
                    listOfItersToBeRemoved.add (iter);
                }
                return false;
            };
            BookwormApp.AppWindow.library_table_liststore.foreach (print_row);
            foreach (Gtk.TreeIter iterToBeRemoved in listOfItersToBeRemoved) {
                //remove item for list store - vala_36 compatibility wrapper
                #if VALA_0_36
                    BookwormApp.AppWindow.library_table_liststore.remove (ref iterToBeRemoved);
                #else
                    BookwormApp.AppWindow.library_table_liststore.remove (iterToBeRemoved);
                #endif
            }
        }
        //loop through the removed books and remove them from the Library View Hashmap, local cache and Database
        foreach (string bookLocation in listOfBooksToBeRemoved) {
            BookwormApp.DB.removeBookFromDB (BookwormApp.Bookworm.libraryViewMap.get (bookLocation));
            BookwormApp.Bookworm.libraryViewMap.unset (bookLocation);
        }
        //Set to normal grid view if the current view is in any of the Grid View State
        if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[0] ||
            BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[2] ||
            BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[3])
        {
            BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_UI_STATES[0];
            BookwormApp.Library.updateGridViewForSelection (null);
        }
        //Set to normal list view if the current view is in any of the List View State
        if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[5] ||
            BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[6] ||
            BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[7])
        {
            BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_UI_STATES[5];
            BookwormApp.Library.updateListViewForSelection (null);
        }
        BookwormApp.Bookworm.toggleUIState ();
        debug ("[END] [FUNCTION:removeSelectedBooksFromLibrary]");
    }

    public static async void updateLibraryViewFromDB () {
        debug ("[START] [FUNCTION:updateLibraryViewFromDB]");
        foreach (BookwormApp.Book book in listOfBooksInLibraryOnLoad) {
            //add the book to the UI - both grid and list view
            BookwormApp.Library.updateLibraryView (book);
            Idle.add (updateLibraryViewFromDB.callback);
            yield;
        }
        debug ("[END] [FUNCTION:updateLibraryViewFromDB]");
    }

    public static async void addBooksToLibrary () {
        debug ("[START] [FUNCTION:addBooksToLibrary]");
        debug ("books to be added=" + BookwormApp.Bookworm.pathsOfBooksToBeAdded.length.to_string ());
        double progress = 0d;
        //loop through the command line and add books to library
        foreach (string pathToSelectedBook in BookwormApp.Bookworm.pathsOfBooksToBeAdded) {
            debug ("Attempting to add book from path:" + pathToSelectedBook);
            //Set async callback only if multiple books are being added
            //If only one book is being added, complete parsing and adding the book,
            //so that it will be added to the BookwormApp.Bookworm.libraryViewMap and opened on the
            //BookwormApp.contentHandler.performStartUpActions method
            if (BookwormApp.Bookworm.pathsOfBooksToBeAdded.length > 2) {
                Idle.add (addBooksToLibrary.callback);
            }
            BookwormApp.Bookworm.noOfBooksAddedFromCommand++;
            if (BookwormApp.Constants.bookworm_id != pathToSelectedBook.strip ()) {
                //set progress for the UI Book addition progress bar
                progress = (((double) (BookwormApp.Bookworm.noOfBooksAddedFromCommand))/((double) (BookwormApp.Bookworm.pathsOfBooksToBeAdded.length)));
                BookwormApp.AppWindow.bookAdditionBar.set_text (_("Adding ") +
                    ((int) (progress * 100)).to_string () + "% : " + File.new_for_path (pathToSelectedBook).get_basename ());
                BookwormApp.AppWindow.bookAdditionBar.set_fraction (progress);
            }
            //Return control back for any further actions only if multiple books are being added
            //If only one book is being added, complete parsing and adding the book,
            //so that it will be added to the BookwormApp.Bookworm.libraryViewMap and opened on the
            //BookwormApp.contentHandler.performStartUpActions method
            if (BookwormApp.Bookworm.pathsOfBooksToBeAdded.length > 2) {
                yield;
            }
            //ignore the first command which is the application name
            if (BookwormApp.Constants.bookworm_id != pathToSelectedBook.strip ()) {
                //check if book already exists in the library
                if (BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr.str.index_of (pathToSelectedBook.strip ()) != -1) {
                    debug ("Book already exists in library..." + BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr.str);
                    //Enable the flag which will scroll the page to the last read position
                    BookwormApp.Bookworm.isPageScrollRequired = true;
                    //set the name of the book being currently read
                    BookwormApp.Bookworm.locationOfEBookCurrentlyRead = pathToSelectedBook.strip ();
                } else {
                    //book does not exist in library - create a new instance for the book
                    BookwormApp.Book aBookBeingAdded = new BookwormApp.Book ();
                    aBookBeingAdded.setBookLocation (pathToSelectedBook.strip ());
                    //the book will be updated to the libraryViewMap within the addBookToLibrary function
                    //however the libraryViewMap will only be fully populated when all books are added to it
                    addBookToLibrary (aBookBeingAdded);
                    //update the onloadBookList - this is to prevent re-adding the book within the same session
                    BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr.append (aBookBeingAdded.getBookLocation ());
                    BookwormApp.Library.listOfBooksInLibraryOnLoad.add (aBookBeingAdded);
                }
            }
        }
        //Hide the progress bar on completion of adding books
        BookwormApp.AppWindow.bookAdditionBar.hide ();
        BookwormApp.Bookworm.isBookBeingAddedToLibrary = false;
        BookwormApp.Bookworm.noOfBooksAddedFromCommand = 0;
        debug ("[END] [FUNCTION:addBooksToLibrary]");
    }

    public static void addBookToLibrary (owned BookwormApp.Book aBook) {
        debug ("[START] [FUNCTION:addBookToLibrary] book.location=" + aBook.getBookLocation ());
        //check if the selected eBook exists
        string eBookLocation = aBook.getBookLocation ();
        File eBookFile = File.new_for_path (eBookLocation);
        if (eBookFile.query_exists () && eBookFile.query_file_type (0) != FileType.DIRECTORY) {
            //insert book details to database and fetch the ID
            int bookID = BookwormApp.DB.addBookToDataBase (aBook);
            aBook.setBookId (bookID);
            /*Other than location, nothing is inserted into the DB for the book at this time.
            Mark book as opened in the session so that details for book are updated
            into DB when the application is closed - eBook parsing happens after the initial insert
            */
            aBook.setBookLastModificationDate ((new DateTime.now_utc ().to_unix ()).to_string ());
            aBook.setWasBookOpened (true);
            //parse eBook to populate cache and book meta data
            aBook = BookwormApp.Bookworm.genericParser (aBook);
            if (!aBook.getIsBookParsed ()) {
                BookwormApp.DB.removeBookFromDB (aBook);
                BookwormApp.AppWindow.showInfoBar (aBook, MessageType.WARNING);
            } else {
                //add eBook cover image to library view
                BookwormApp.Library.updateLibraryView (aBook);
                //Set to normal grid view if the current view is in any of the Grid View State
                if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[0] ||
                    BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[2] ||
                    BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[3])
                {
                    BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_UI_STATES[0];
                    BookwormApp.Library.updateGridViewForSelection (null);
                }
                //Set to normal list view if the current view is in any of the List View State
                if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[5] ||
                    BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[6] ||
                    BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[7])
                {
                    BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_UI_STATES[5];
                    BookwormApp.Library.updateListViewForSelection (null);
                }
                BookwormApp.Bookworm.toggleUIState ();
                //set the name of the book being currently read
                BookwormApp.Bookworm.locationOfEBookCurrentlyRead = eBookLocation;
                debug ("Completed adding book to ebook library. Number of books in library:" +
                    BookwormApp.Bookworm.libraryViewMap.size.to_string ());
            }
        } else {
            debug ("No ebook found for adding to library");
        }
        debug ("[END] [FUNCTION:addBookToLibrary] book.location=" + aBook.getBookLocation ());
    }

    public static void paginateLibrary (string library_search_data, string mode) {
        //update the current set of books into the library db
        foreach (var book in BookwormApp.Bookworm.libraryViewMap.values) {
            if (((BookwormApp.Book)book).getWasBookOpened ()) {
                BookwormApp.DB.updateBookToDataBase ((BookwormApp.Book)book);
                debug ("Completed saving the book data into DB");
            }
        }
        if (mode == "LIBRARY_SEARCH") {
            //Perform library search for results
            debug ("Executing library search query with criteria: " + library_search_data);
            BookwormApp.Library.listOfBooksInLibraryOnLoad = BookwormApp.DB.getBooksFromDB (library_search_data, mode);
        } else {
            //Query DB for the next/prev page
            debug ("Executing paginated query for books with " +
                "current_page_counter: " + BookwormApp.Bookworm.current_page_counter.to_string () +
                " on paginationlist: " + string.joinv (", ", (BookwormApp.Bookworm.paginationlist.to_array ())));
            BookwormApp.Library.listOfBooksInLibraryOnLoad = BookwormApp.DB.getBooksFromDB (
                BookwormApp.Bookworm.paginationlist.get (BookwormApp.Bookworm.current_page_counter), mode);
        }
        //check for the condition where no books are returned from the DB for the page criteria
        if (BookwormApp.Library.listOfBooksInLibraryOnLoad.size != 0) {
            //Remove books currently on grid view
            GLib.List<weak Gtk.Widget> children_grid = BookwormApp.AppWindow.library_grid.get_children ();
            foreach (Gtk.Widget element in children_grid) {
                BookwormApp.AppWindow.library_grid.remove (element);
            }
            //Remove boooks currently on list view
            ArrayList<Gtk.TreeIter?> listOfItersToBeRemoved = new ArrayList<Gtk.TreeIter?> ();
            Gtk.TreeModelForeachFunc print_row = (model, path, iter) => {
                listOfItersToBeRemoved.add (iter);
                return false;
            };
            BookwormApp.AppWindow.library_table_liststore.foreach (print_row);
            foreach (Gtk.TreeIter iterToBeRemoved in listOfItersToBeRemoved) {
                //remove item for list store - vala_36 compatibility wrapper
                #if VALA_0_36
                    BookwormApp.AppWindow.library_table_liststore.remove (ref iterToBeRemoved);
                #else
                    BookwormApp.AppWindow.library_table_liststore.remove (iterToBeRemoved);
                #endif
            }
            //Clear the paths of the loaded books
            BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr.erase (0, -1);
            //Update the library view
            BookwormApp.Library.updateLibraryViewFromDB ();
        }
        //set the status of the library buttons based on the paginate query results
        ////false : will prevent another paginate call
        BookwormApp.AppWindow.handleLibraryPageButtons ("", false);
    }
}
```

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

```
/* Copyright 2017 Siddhartha Das (bablu.boy@gmail.com)
*
* This file is part of Bookworm and manages all the Database interactions
*
* Bookworm is free software: you can redistribute it
* and/or modify it under the terms of the GNU General Public License as
* published by the Free Software Foundation, either version 3 of the
* License, or (at your option) any later version.
*
* Bookworm is distributed in the hope that it will be
* useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
* Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with Bookworm. If not, see http://www.gnu.org/licenses/.
*/

using Sqlite;
using Gee;

public class BookwormApp.DB {
    public const string BOOKWORM_TABLE_BASE_NAME = "BOOK_LIBRARY_TABLE";
    public const string BOOKWORM_TABLE_VERSION = "6"; //Only integers allowed
    public const string BOOKWORM_TABLE_NAME = BOOKWORM_TABLE_BASE_NAME + BOOKWORM_TABLE_VERSION;
    public const string BOOKMETADATA_TABLE_BASE_NAME = "BOOK_METADATA_TABLE";
    public const string BOOKMETADATA_TABLE_VERSION = "1"; //Only integers allowed
    public const string BOOKMETADATA_TABLE_NAME = BOOKMETADATA_TABLE_BASE_NAME + BOOKMETADATA_TABLE_VERSION;
    public const string VERSION_TABLE_BASE_NAME = "VERSION_TABLE";
    public const string VERSION_TABLE_VERSION = "1"; //Only integers allowed
    private static Sqlite.Database bookwormDB;
    private static string errmsg;
    private static string queryString;
    private static int executionStatus;

    public static bool initializeBookWormDB (string bookworm_config_path) {
        info ("[START] [FUNCTION:initializeBookWormDB] bookworm_config_path=" + bookworm_config_path);
        Statement stmt;
        debug ("Checking BookWorm DB or creating it if the DB does not exist...");
        int dbOpenStatus = Database.open_v2 (
            bookworm_config_path + "/bookworm.db", out bookwormDB, Sqlite.OPEN_READWRITE | Sqlite.OPEN_CREATE);
        if (dbOpenStatus != Sqlite.OK) {
            warning ("Error in opening database[" + bookworm_config_path + "/bookworm.db" + "]: %d: %s\n", bookwormDB.errcode (), bookwormDB.errmsg ());
            return false;
        } else {
            debug ("Successfully checked/created DB for Bookworm.....");
        }

        debug ("Creating latest version for Library table if it does not exists");
        queryString = "CREATE TABLE IF NOT EXISTS " + BOOKWORM_TABLE_NAME + " (" +
            "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
            "BOOK_LOCATION TEXT NOT NULL DEFAULT '', " +
            "BOOK_TITLE TEXT NOT NULL DEFAULT '', " +
            "BOOK_AUTHOR TEXT NOT NULL DEFAULT '', " +
            "BOOK_COVER_IMAGE_LOCATION TEXT NOT NULL DEFAULT '', " +
            "IS_BOOK_COVER_IMAGE_PRESENT TEXT NOT NULL DEFAULT '', " +
            "BOOK_PUBLISH_DATE TEXT NOT NULL DEFAULT '', " +
            "BOOK_TOTAL_NUMBER_OF_PAGES TEXT NOT NULL DEFAULT '', " +
            "BOOK_LAST_READ_PAGE_NUMBER TEXT NOT NULL DEFAULT '', " +
            "BOOK_TOTAL_PAGES TEXT NOT NULL DEFAULT '', " + //Added in table v6
            "TAGS TEXT NOT NULL DEFAULT '', " + //Added in table v3
            "ANNOTATION_TAGS TEXT NOT NULL DEFAULT '', " + //Added in table v7
            "RATINGS TEXT NOT NULL DEFAULT '', " + //Added in table v3
            "CONTENT_EXTRACTION_LOCATION TEXT NOT NULL DEFAULT '', " + //Added in table v4
            "creation_date INTEGER," +
            "modification_date INTEGER)";
        executionStatus = bookwormDB.exec (queryString, null, out errmsg);
        if (executionStatus != Sqlite.OK) {
            report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errms
g ());
            return false;
        } else {
            debug ("Successfully checked/created table:" + BOOKWORM_TABLE_NAME);
        }

        debug ("Creating latest version for Book Metadata table if it does not exists");
        queryString = "CREATE TABLE IF NOT EXISTS " + BOOKMETADATA_TABLE_NAME + " (" +
            "id INTEGER PRIMARY KEY, " +
            "BOOK_TOC_DATA TEXT NOT NULL DEFAULT '', " +
            "BOOKMARKS TEXT NOT NULL DEFAULT '', " +
            "CONTENT_DATA_LIST TEXT NOT NULL DEFAULT '', " +
            "BOOK_LAST_SCROLL_POSITION TEXT NOT NULL DEFAULT '', " +
            "BOOK_ANNOTATIONS TEXT NOT NULL DEFAULT '', " +
            "creation_date INTEGER," +
            "modification_date INTEGER)";
        executionStatus = bookwormDB.exec (queryString, null, out errmsg);
        if (executionStatus != Sqlite.OK) {
            report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errms
g ());
            return false;
        } else {
            debug ("Successfully checked/created table:" + BOOKMETADATA_TABLE_NAME);
        }

        //Check details of tables in DB
        ArrayList<string> listOfTables = new ArrayList<string> ();
        queryString = "SELECT NAME FROM SQLITE_MASTER WHERE TYPE='table' ORDER BY NAME";
        executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
        if (executionStatus != Sqlite.OK) {
            report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errms
g ());
        }
        while (stmt.step () == ROW) {
            listOfTables.add (stmt.column_text (0).strip ());
        }
        stmt.reset ();

        //Remove the current tables (latest versions) from the list
        listOfTables.remove (BOOKWORM_TABLE_NAME);
        listOfTables.remove (BOOKMETADATA_TABLE_NAME);

        //Loop over any remaning old versions of tables and delete
        //them after ensuring data is migrated to the latest versions of the tables
        foreach (string old_table_name in listOfTables) {
            //BOOK_LIBRARY_TABLE5 : Migrate data and drop table
            if (old_table_name == "BOOK_LIBRARY_TABLE5") {
                //copy data to new library table
                queryString = "INSERT INTO " + BOOKWORM_TABLE_NAME + " (" +
                    "id, BOOK_LOCATION, BOOK_TITLE, BOOK_AUTHOR, BOOK_COVER_IMAGE_LOCATION, "
 +
                    "IS_BOOK_COVER_IMAGE_PRESENT, BOOK_PUBLISH_DATE, BOOK_TOTAL_NUMBER_OF_PAGE
S, " +
                    "BOOK_LAST_READ_PAGE_NUMBER, TAGS, RATINGS,CONTENT_EXTRACTION_LOCATION, "
 +
                    "creation_date, modification_date) SELECT id, BOOK_LOCATION, BOOK_TITLE, "
 +
                    "BOOK_AUTHOR, BOOK_COVER_IMAGE_LOCATION, IS_BOOK_COVER_IMAGE_PRESENT, " +
                    "BOOK_PUBLISH_DATE, BOOK_TOTAL_NUMBER_OF_PAGES, BOOK_LAST_READ_PAGE_NUMBER
, " +
                    "TAGS, RATINGS, CONTENT_EXTRACTION_LOCATION, creation_date, " +
                    "modification_date FROM BOOK_LIBRARY_TABLE5";
                executionStatus = bookwormDB.exec (queryString, null, out errmsg);
                if (executionStatus != Sqlite.OK) {
                    report_query_execution_error (queryString, bookwormDB.errcode (), bookworm
DB.errmsg ());
                } else {
                    debug ("Successfully migrated " + bookwormDB.changes ().to_string () + " r
ows from BOOK_LIBRARY_TABLE5 into " + BOOKWORM_TABLE_NAME);
                    //copy data to new meta data table
                    queryString = "INSERT INTO " + BOOKMETADATA_TABLE_NAME + " (" +
                        "id, BOOK_TOC_DATA, BOOKMARKS, CONTENT_DATA_LIST, BOOK_LAST_SCROLL_POSITION, " +
                        "creation_date, modification_date) SELECT id, BOOK_TOC_DATA, BOOKMARKS, " +
                        "CONTENT_DATA_LIST, BOOK_LAST_SCROLL_POSITION, creation_date, " +
                        "modification_date FROM BOOK_LIBRARY_TABLE5";
                    executionStatus = bookwormDB.exec (queryString, null, out errmsg);
                    if (executionStatus != Sqlite.OK) {
                        report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errmsg ());
                    } else {
                        debug ("Successfully migrated " + bookwormDB.changes ().to_string () +
                            " rows from BOOK_LIBRARY_TABLE5 into" + BOOKMETADATA_TABLE_NAME);
                        //drop the old table
                        queryString = "DROP TABLE IF EXISTS BOOK_LIBRARY_TABLE5";
                        executionStatus = bookwormDB.exec (queryString, null, out errmsg);
                        if (executionStatus != Sqlite.OK) {
                            report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errmsg ());
                        } else {
                            debug ("Successfully dropped old table LIBRARY_TABLE5");
                        }
                    }
                }
            }
            //VERSION_TABLE : Drop table
            if (old_table_name == "VERSION_TABLE") {
                //drop the old table
                queryString = "DROP TABLE IF EXISTS VERSION_TABLE";
                executionStatus = bookwormDB.exec (queryString, null, out errmsg);
                if (executionStatus != Sqlite.OK) {
                    report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errmsg ());
                } else {
                    debug ("Successfully dropped old table VERSION_TABLE");
                }
            }
        }
        //All DB loading operations completed
        info ("[END] [FUNCTION:initializeBookWormDB]");
        return true;
    }

    public static ArrayList<BookwormApp.Book> getBooksFromDB (string criteria, string mode) {
        info ("[START] [FUNCTION:getBooksFromDB] Quering with mode[" + mode + "] and criteria[" + criteria + "]");
        ArrayList<BookwormApp.Book> listOfBooks = new ArrayList<BookwormApp.Book> ();
        Statement stmt;
        string last_modification_date = "-1";
        queryString = "SELECT id, BOOK_LOCATION, BOOK_TITLE, BOOK_AUTHOR, BOOK_COVER_IMAGE_LOCATION, " +
            "IS_BOOK_COVER_IMAGE_PRESENT, BOOK_LAST_READ_PAGE_NUMBER, BOOK_PUBLISH_DATE, TAGS, " +
            "ANNOTATION_TAGS, RATINGS, CONTENT_EXTRACTION_LOCATION, BOOK_TOTAL_PAGES, creation_date, " +
            "modification_date FROM " + BOOKWORM_TABLE_NAME;
        if (criteria == "" && mode == "PAGINATED_SEARCH") {
            //initial query on app load without pagination criteria
            queryString = queryString + " ORDER BY modification_date DESC LIMIT " + BookwormApp.Bookworm.no_of_books_per_page;
            debug ("Paginated Query with last_modification_date[" + criteria + "]:" + queryString);
        } else if (mode == "LIBRARY_SEARCH") {
            //query db for matching search criteria on all book meta data
            queryString = queryString + " WHERE " +
                " BOOK_TITLE LIKE ’%" + criteria + "%’ OR " +
                " BOOK_LOCATION LIKE ’%" + criteria + "%’ OR " +
                " BOOK_AUTHOR LIKE ’%" + criteria + "%’ OR " +
                " TAGS LIKE ’%" + criteria + "%’ OR " +
                " ANNOTATION_TAGS LIKE ’%" + criteria + "%’";
            debug ("Library Search Query with criteria[" + criteria + "]:" + queryString);
        } else if (mode == "PAGINATED_SEARCH") {
            //query for pagination criteria
            queryString = queryString + " where modification_date < CAST (’" + criteria + "’ AS INT) " +
                "ORDER BY modification_date DESC LIMIT " + BookwormApp.Bookworm.no_of_books_per_page;
            debug ("Paginated Query with last_modification_date[" + criteria + "]:" + queryString);
        }
        executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
        if (executionStatus != Sqlite.OK) {
            report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errmsg ());
        } else {
            while (stmt.step () == ROW) {
                BookwormApp.Book aBook = new BookwormApp.Book ();
                aBook.setBookId (stmt.column_int (0));
                aBook.setBookLocation (stmt.column_text (1));
                aBook.setBookTitle (stmt.column_text (2));
                aBook.setBookAuthor (stmt.column_text (3));
                aBook.setBookCoverLocation (stmt.column_text (4));
                aBook.setIsBookCoverImagePresent ( (stmt.column_text (5) == "true") ? true:false);
                aBook.setBookPageNumber (int.parse (stmt.column_text (6)));
                aBook.setBookPublishDate (stmt.column_text (7));
                aBook.setBookTags (stmt.column_text (8));
                aBook.setAnnotationTags (stmt.column_text (9));
                aBook.setBookRating (int.parse (stmt.column_text (10)));
                aBook.setBookExtractionLocation (stmt.column_text (11));
                aBook.setBookTotalPages (int.parse (stmt.column_text (12)));
                aBook.setBookCreationDate (stmt.column_text (13));
                aBook.setBookLastModificationDate (stmt.column_text (14));
                debug ("Book details fetched from DB: id=" + stmt.column_int (0).to_string () +
                    ",BOOK_LOCATION=" + stmt.column_text (1) +
                    ",BOOK_TITLE=" + stmt.column_text (2) +
                    ",BOOK_AUTHOR=" + stmt.column_text (3) +
                    ",BOOK_COVER_IMAGE_LOCATION=" + stmt.column_text (4) +
                    ",IS_BOOK_COVER_IMAGE_PRESENT=" + stmt.column_text (5) +
                    ",BOOK_LAST_READ_PAGE_NUMBER=" + stmt.column_text (6) +
                    ",BOOK_PUBLISH_DATE=" + stmt.column_text (7) +
                    ",TAGS=" + stmt.column_text (8) +
                    ",ANNOTATION_TAGS=" + stmt.column_text (9) +
                    ",RATINGS=" + stmt.column_text (10) +
                    ",CONTENT_EXTRACTION_LOCATION=" + stmt.column_text (11) +
                    ",BOOK_TOTAL_PAGES=" + stmt.column_text (12) +
                    ",creation_date=" + stmt.column_text (13) +
                    ",modification_date=" + stmt.column_text (14));
                //add book details to list
                listOfBooks.add (aBook);
                //build the string of book paths in the library
                BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr.append (aBook.getBookLocation ());
                //capture the last_modification_date of the book
                last_modification_date = aBook.getBookLastModificationDate ();
            }
            if (mode == "PAGINATED_SEARCH") {
                //Only capture the last modification date if the results are equal to the page size
                if (listOfBooks.size == int.parse (BookwormApp.Bookworm.no_of_books_per_page)) {
                    //set the last book’s modification date for pagination
                    BookwormApp.Bookworm.paginationlist.add (last_modification_date);
                } else {
                    BookwormApp.Bookworm.paginationlist.add ("-1");
                }
            }
            stmt.reset ();
        }
        info ("[END] [FUNCTION:getBooksFromDB] no. of books fetched [" + listOfBooks.size.to_string () + "], last_modification_date of books fetched[" + last_modification_date + "]");
        return listOfBooks;
    }

    public static BookwormApp.Book getBookFromDB (string book_location) {
        info ("[START] [FUNCTION:getBookFromDB] Attempting to search DB for book.location=" + book_location);
        Statement stmt;
        BookwormApp.Book aBook = new BookwormApp.Book ();
        queryString = "SELECT id, BOOK_LOCATION, BOOK_TITLE, BOOK_AUTHOR, BOOK_COVER_IMAGE_LOCATION, " +
            "IS_BOOK_COVER_IMAGE_PRESENT, BOOK_LAST_READ_PAGE_NUMBER, BOOK_PUBLISH_DATE, TAGS, " +
            "ANNOTATION_TAGS, RATINGS, CONTENT_EXTRACTION_LOCATION, BOOK_TOTAL_PAGES, creation_date, " +
            "modification_date FROM " + BOOKWORM_TABLE_NAME;
        //query db for matching book location
        queryString = queryString + " WHERE " + " BOOK_LOCATION LIKE ’" + book_location + "’";
        debug ("Library Search Query with criteria[" + book_location + "]:" + queryString);
        executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
        if (executionStatus != Sqlite.OK) {
            report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errmsg ());
        } else {
            while (stmt.step () == ROW) {
                aBook.setBookId (stmt.column_int (0));
                aBook.setBookLocation (stmt.column_text (1));
                aBook.setBookTitle (stmt.column_text (2));
                aBook.setBookAuthor (stmt.column_text (3));
                aBook.setBookCoverLocation (stmt.column_text (4));
                aBook.setIsBookCoverImagePresent ( (stmt.column_text (5) == "true") ? true : false);
                aBook.setBookPageNumber (int.parse (stmt.column_text (6)));
                aBook.setBookPublishDate (stmt.column_text (7));
                aBook.setBookTags (stmt.column_text (8));
                aBook.setAnnotationTags (stmt.column_text (9));
                aBook.setBookRating (int.parse (stmt.column_text (10)));
                aBook.setBookExtractionLocation (stmt.column_text (11));
                aBook.setBookTotalPages (int.parse (stmt.column_text (12)));
                aBook.setBookCreationDate (stmt.column_text (13));
                aBook.setBookLastModificationDate (stmt.column_text (14));
                debug ("Book details fetched from DB: id=" + stmt.column_int (0).to_string () +
                    ",BOOK_LOCATION=" + stmt.column_text (1) +
                    ",BOOK_TITLE=" + stmt.column_text (2) +
                    ",BOOK_AUTHOR=" + stmt.column_text (3) +
                    ",BOOK_COVER_IMAGE_LOCATION=" + stmt.column_text (4) +
                    ",IS_BOOK_COVER_IMAGE_PRESENT=" + stmt.column_text (5) +
                    ",BOOK_LAST_READ_PAGE_NUMBER=" + stmt.column_text (6) +
                    ",BOOK_PUBLISH_DATE=" + stmt.column_text (7) +
                    ",TAGS=" + stmt.column_text (8) +
                    ",ANNOTATION_TAGS=" + stmt.column_text (9) +
                    ",RATINGS=" + stmt.column_text (10) +
                    ",CONTENT_EXTRACTION_LOCATION=" + stmt.column_text (11) +
                    ",BOOK_TOTAL_PAGES=" + stmt.column_text (12) +
                    ",creation_date=" + stmt.column_text (13) +
                    ",modification_date=" + stmt.column_text (14));
            }
        }
        stmt.reset ();
        info ("[END] [FUNCTION:getBookFromDB] Book fetched [" + aBook.getBookLocation () + "]");
        return aBook;
    }

    public static BookwormApp.Book getBookMetaDataFromDB (owned BookwormApp.Book aBook) {
        info ("[START] [FUNCTION:getBookMetaDataFromDB] book.location=" + aBook.getBookLocation ());
        Statement stmt;
        queryString = "SELECT BOOK_TOC_DATA, BOOKMARKS, CONTENT_DATA_LIST, BOOK_LAST_SCROLL_POSITION, " +
            "BOOK_ANNOTATIONS FROM " + BOOKMETADATA_TABLE_NAME + " WHERE id = ?";
        executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
        if (executionStatus != Sqlite.OK) {
            report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errmsg ());
        }
        stmt.bind_int (1, aBook.getBookId ());
        while (stmt.step () == ROW) {
            aBook = BookwormApp.Utils.convertStringToTOC (aBook, stmt.column_text (0));
            aBook.setBookmark (-10, stmt.column_text (1)); //-10 is a flag to set the bookmark string into the object
            aBook = BookwormApp.Utils.convertStringToContentList (aBook, stmt.column_text (2));
            aBook.setBookScrollPos (int.parse (stmt.column_text (3)));
            aBook.setAnnotationList (BookwormApp.Utils.convertStringToTreeMap (stmt.column_text (4)));
            debug ("Book MetaData details fetched from DB: id=" + aBook.getBookId ().to_string () +
                ", BOOK_TOC_DATA=" + stmt.column_text (0) +
                ", BOOKMARKS=" + stmt.column_text (1) +
                ", CONTENT_DATA_LIST=" + stmt.column_text (2) +
                ", BOOK_LAST_SCROLL_POSITION=" + stmt.column_text (3) +
                ", BOOK_ANNOTATIONS=" + stmt.column_text (4));
        }
        stmt.reset ();
        info ("[END] [FUNCTION:getBookMetaDataFromDB] book.location=" + aBook.getBookLocation ());
        return aBook;
    }

    public static int addBookToDataBase (BookwormApp.Book aBook) {
        info ("[START] [FUNCTION:addBookToDataBase] book.location=" + aBook.getBookLocation ());
        Sqlite.Statement stmt;
        int insertedBookID = 0;
        queryString = "INSERT INTO " + BOOKWORM_TABLE_NAME + " (BOOK_LOCATION, BOOK_TITLE, BOOK_AUTHOR, " +
            "BOOK_COVER_IMAGE_LOCATION, IS_BOOK_COVER_IMAGE_PRESENT, CONTENT_EXTRACTION_LOCATION, " +
            "creation_date, modification_date) " + "VALUES (?, ?, ?, ?, ?, ?, CAST (strftime (’%s’, ’now’) " +
            "AS INT), CAST (strftime (’%s’, ’now’) AS INT))";
        executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
        if (executionStatus != Sqlite.OK) {
            report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errmsg ());
            return -1;
        }
        stmt.bind_text (1, aBook.getBookLocation ());
        stmt.bind_text (2, aBook.getBookTitle ());
        stmt.bind_text (3, aBook.getBookAuthor ());
        stmt.bind_text (4, aBook.getBookCoverLocation ());
        stmt.bind_text (5, aBook.getIsBookCoverImagePresent ().to_string ());
        stmt.bind_text (6, aBook.getBookExtractionLocation ());

        stmt.step ();
        stmt.reset ();
        //fetch the id of the book just inserted into the DB
        queryString = "SELECT id FROM " + BOOKWORM_TABLE_NAME + " WHERE BOOK_LOCATION = ?";
        executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
        if (executionStatus != Sqlite.OK) {
            report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errmsg ());
        }
        stmt.bind_text (1, aBook.getBookLocation ());
        while (stmt.step () == ROW) {
            insertedBookID = stmt.column_int (0);
        }
        stmt.reset ();
        info ("[END] [FUNCTION:addBookToDataBase] book.location=" + aBook.getBookLocation ());
        return insertedBookID;
    }

    public static bool removeBookFromDB (BookwormApp.Book aBook) {
        info ("[START] [FUNCTION:removeBookFromDB] book.location=" + aBook.getBookLocation ());
        Sqlite.Statement stmt;
        //delete book from library table
        queryString = "DELETE FROM " + BOOKWORM_TABLE_NAME + " WHERE id = ?";
        executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
        if (executionStatus != Sqlite.OK) {
            report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errmsg ());
            return false;
        } else {
            stmt.bind_int (1, aBook.getBookId ());
            stmt.step ();
            stmt.reset ();
            debug ("Removed this book from library table:" + aBook.getBookTitle () + "[" + aBook.getBookId ().to_string () + "]");
            //delete book meta data from meta data table
            queryString = "DELETE FROM " + BOOKMETADATA_TABLE_NAME + " WHERE id = ?";
            executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
            if (executionStatus != Sqlite.OK) {
                report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errmsg ());
                return false;
            } else {
                stmt.bind_int (1, aBook.getBookId ());
                stmt.step ();
                stmt.reset ();
            }
        }
        info ("[END] [FUNCTION:removeBookFromDB] book.location=" + aBook.getBookLocation ());
        return true;
    }

    public static bool updateBookToDataBase (BookwormApp.Book aBook) {
        info ("[START] [FUNCTION:updateBookToDataBase] Updating book to DB for the following details:" + aBook.to_string ());
        Sqlite.Statement stmt;
        queryString = "UPDATE " + BOOKWORM_TABLE_NAME + " SET BOOK_LAST_READ_PAGE_NUMBER = ?, " +
            "BOOK_TITLE = ?, BOOK_AUTHOR = ?, BOOK_COVER_IMAGE_LOCATION = ?, " +
            "IS_BOOK_COVER_IMAGE_PRESENT = ?, TAGS = ?, ANNOTATION_TAGS = ?, RATINGS = ?, " +
            "CONTENT_EXTRACTION_LOCATION = ?, BOOK_TOTAL_PAGES = ?, " +
            "modification_date = CAST (? AS INT) WHERE ID = ? ";
        executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
        if (executionStatus != Sqlite.OK) {
            report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errmsg ());
            return false;
        }
        stmt.bind_text (1, aBook.getBookPageNumber ().to_string ());
        stmt.bind_text (2, aBook.getBookTitle ());
        stmt.bind_text (3, aBook.getBookAuthor ());
        stmt.bind_text (4, aBook.getBookCoverLocation ());
        stmt.bind_text (5, aBook.getIsBookCoverImagePresent ().to_string ());
        stmt.bind_text (6, aBook.getBookTags ());
        stmt.bind_text (7, aBook.getAnnotationTags ());
        stmt.bind_text (8, aBook.getBookRating ().to_string ());
        stmt.bind_text (9, aBook.getBookExtractionLocation ());
        stmt.bind_text (10, aBook.getBookTotalPages ().to_string ());
        stmt.bind_text (11, aBook.getBookLastModificationDate ());
        stmt.bind_int (12, aBook.getBookId ());
        stmt.step ();
        stmt.reset ();
        debug ("Updated library details to " + BOOKWORM_TABLE_NAME + " for book:" + aBook.getBookTitle () + "[" + aBook.getBookId ().to_string () + "]");
        //Attempt to insert book meta data
        queryString = "INSERT OR IGNORE INTO " + BOOKMETADATA_TABLE_NAME + " (BOOK_TOC_DATA, BOOKMARKS, " +
            "CONTENT_DATA_LIST, BOOK_LAST_SCROLL_POSITION, BOOK_ANNOTATIONS, modification_date, id) " +
            "VALUES (?, ?, ?, ?, ?, CAST (strftime ('%s', 'now') AS INT), ?);";
        executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
        if (executionStatus != Sqlite.OK) {
            report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errmsg ());
            return false;
        }
        stmt.bind_text (1, BookwormApp.Utils.convertTOCToString (aBook));
        stmt.bind_text (2, aBook.getBookmark ());
        stmt.bind_text (3, BookwormApp.Utils.convertContentListToString (aBook));
        stmt.bind_text (4, aBook.getBookScrollPos ().to_string ());
        stmt.bind_text (5, BookwormApp.Utils.convertTreeMapToString (aBook.getAnnotationList ()));
        stmt.bind_int (6, aBook.getBookId ());
        stmt.step ();
        stmt.reset ();
        if (bookwormDB.changes () == 0) {
            //Book already present, update the meta data
            queryString = "UPDATE " + BOOKMETADATA_TABLE_NAME + " SET BOOK_TOC_DATA = ?, BOOKMARKS = ?, " +
                "CONTENT_DATA_LIST = ?, BOOK_LAST_SCROLL_POSITION = ?, BOOK_ANNOTATIONS = ?, " +
                "modification_date = CAST (strftime ('%s', 'now') AS INT) WHERE id = ? ";
            executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
            if (executionStatus != Sqlite.OK) {
                report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errmsg ());
                return false;
            }
            stmt.bind_text (1, BookwormApp.Utils.convertTOCToString (aBook));
            stmt.bind_text (2, aBook.getBookmark ());
            stmt.bind_text (3, BookwormApp.Utils.convertContentListToString (aBook));
            stmt.bind_text (4, aBook.getBookScrollPos ().to_string ());
            stmt.bind_text (5, BookwormApp.Utils.convertTreeMapToString (aBook.getAnnotationList ()));
            stmt.bind_int (6, aBook.getBookId ());
            stmt.step ();
            stmt.reset ();
            debug ("Updated book meta data details to " + BOOKMETADATA_TABLE_NAME + " for book:" + aBook.getBookTitle () + "[" + aBook.getBookId ().to_string () + "]");
        } else {
            debug ("Inserted book meta data details to " + BOOKMETADATA_TABLE_NAME + " for book:" + aBook.getBookTitle () + "[" + aBook.getBookId ().to_string () + "]");
        }
        info ("[END] [FUNCTION:updateBookToDataBase] book.location=" + aBook.getBookLocation ());
        return true;
    }

    public static ArrayList<string> getBookIDListFromDB () {
        info ("[START] [FUNCTION:getBookIDListFromDB]");
        ArrayList<string> bookIDList = new ArrayList<string> ();
        Statement stmt;
        queryString = "SELECT id,BOOK_LOCATION FROM " + BOOKWORM_TABLE_NAME + " ORDER BY id DESC";
        executionStatus = bookwormDB.prepare_v2 (queryString, queryString.length, out stmt);
        if (executionStatus != Sqlite.OK) {
            report_query_execution_error (queryString, bookwormDB.errcode (), bookwormDB.errmsg ());
        }
        while (stmt.step () == ROW) {
            bookIDList.add (stmt.column_int (0).to_string () + "::" + stmt.column_text (1));
        }
        stmt.reset ();
        info ("[END] [FUNCTION:getBookIDListFromDB] bookIDList.size" + bookIDList.size.to_string ());
        return bookIDList;
    }

    private static void report_query_execution_error (string query, int errcode, string errmsg) {
        debug   ("Error on executing Query: %s\n", query);
        warning ("Error (%d) details: %s\n", errcode, errmsg);
    }
}
```

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

```
/* Copyright 2017 Siddhartha Das (bablu.boy@gmail.com)
*
* This file is part of Bookworm and is used for parsing EPUB file formats
*
* Bookworm is free software: you can redistribute it
* and/or modify it under the terms of the GNU General Public License as
* published by the Free Software Foundation, either version 3 of the
* License, or (at your option) any later version.
*
* Bookworm is distributed in the hope that it will be
* useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
* Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with Bookworm. If not, see http://www.gnu.org/licenses/.
*/

using Gee;
public class BookwormApp.ePubReader {
    public static string NCXRefInSpineData = "";
    public static BookwormApp.Book parseEPubBook (owned BookwormApp.Book aBook) {
        info ("[START] [FUNCTION:parseEPubBook] book.location=" + aBook.getBookLocation ());
        //Only parse the eBook if it has not been parsed already
        if (!aBook.getIsBookParsed ()) {
            debug ("Starting to parse EPub Book located at:" + aBook.getBookLocation ());
            //Extract the content of the EPub
            string extractionLocation = extractEBook (aBook.getBookLocation ());
            if ("false" == extractionLocation) { //handle error condition
                aBook.setIsBookParsed (false);
                aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_EXTRACTION_ISSUE);
                return aBook;
            } else {
                aBook.setBookExtractionLocation (extractionLocation);
            }
            //Check if the EPUB mime type is correct
            bool isEPubFormat = isEPubFormat (extractionLocation);
            if (!isEPubFormat) { //handle error condition
                aBook.setIsBookParsed (false);
                aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_MIMETYPE_ISSUE);
                return aBook;
            }
            //Determine the location of OPF File
            string locationOfOPFFile = getOPFFileLocation (extractionLocation);
            if ("false" == locationOfOPFFile) { //handle error condition
                aBook.setIsBookParsed (false);
                aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_CONTENT_ISSUE);
                return aBook;
            }
            string baseLocationOfContents = locationOfOPFFile.replace (File.new_for_path (loca
tionOfOPFFile).get_basename (), "");
            aBook.setBaseLocationOfContents (baseLocationOfContents);
            //Populate content list for EPub Book
            aBook = determineToC (aBook, locationOfOPFFile);
            if (aBook.getBookContentList ().size < 1) {
                aBook.setIsBookParsed (false);
                aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_CONTENT_ISSUE);
                return aBook;
            }
            //Try to determine Book Cover Image if it is not already available
            if (!aBook.getIsBookCoverImagePresent ()) {
                aBook = setCoverImage (aBook, locationOfOPFFile);
            }
            //Determine Book Meta Data like Title, Author, etc
            aBook = setBookMetaData (aBook, locationOfOPFFile);
            aBook.setIsBookParsed (true);
        }
        info ("[END] [FUNCTION:parseEPubBook]");
        return aBook;
    }

    public static string extractEBook (string eBookLocation) {
        info ("[START] [FUNCTION:extractEBook] eBookLocation=" + eBookLocation);
        string extractionLocation = "false";
        debug ("Initiated process for content extraction of ePub Book located at:" + eBookLocation);
        //create a location for extraction of eBook based on local storage prefference
        if (BookwormApp.Bookworm.settings == null) {
            BookwormApp.Bookworm.settings = BookwormApp.Settings.get_instance ();
        }
        if (BookwormApp.Bookworm.settings.is_local_storage_enabled) {
            extractionLocation = BookwormApp.Bookworm.bookworm_config_path + "/books/" + File.new_for_path (eBookLocation).get_basename ();
        } else {
            extractionLocation = BookwormApp.Constants.EBOOK_EXTRACTION_LOCATION + File.new_for_path (eBookLocation).get_basename ();
        }
        //check and create directory for extracting contents of ebook
        BookwormApp.Utils.fileOperations ("CREATEDIR", extractionLocation, "", "");
        //unzip eBook contents into extraction location
        string status = BookwormApp.Utils.execute_sync_command ("unzip -o \"" + eBookLocation + "\" -d \"" + extractionLocation + "\"");
        if ("false" == status) {
            extractionLocation = "false";
        }
        info ("[END] [FUNCTION:extractEBook] extractionLocation=" + extractionLocation);
        return extractionLocation;
    }

    public static bool isEPubFormat (string extractionLocation) {
        info ("[START] [FUNCTION:isEPubFormat] extractionLocation=" + extractionLocation);
        bool ePubFormat = false;
        debug ("Checking if mime type is valid ePub for contents at:" + extractionLocation);
        string ePubMimeContents = BookwormApp.Utils.fileOperations (
            "READ", extractionLocation, BookwormApp.Constants.EPUB_MIME_SPECIFICATION_FILENAME, "");
        if ("false" == ePubMimeContents) {
            //Mime Content File was not found at expected location
            warning ("Mime Content file could not be located at expected location:" +
                extractionLocation + "/" + BookwormApp.Constants.EPUB_MIME_SPECIFICATION_FILENAME);
            return false;
        }
        debug ("Mime Contents found in file :" + extractionLocation + "/" +
            BookwormApp.Constants.EPUB_MIME_SPECIFICATION_FILENAME + " is:" + ePubMimeContents);
        if (ePubMimeContents.strip () != BookwormApp.Constants.EPUB_MIME_SPECIFICATION_CONTENT) {
            debug ("Mime Contents in file :" + extractionLocation + "/" +
                BookwormApp.Constants.EPUB_MIME_SPECIFICATION_FILENAME + " is not :" +
                BookwormApp.Constants.EPUB_MIME_SPECIFICATION_CONTENT + ". No further parsing will be done.");
            return false;
        } else {
            //mime content is as expected
            ePubFormat = true;
        }
        info ("[END] [FUNCTION:isEPubFormat] ePubFormat=" + ePubFormat.to_string ());
        return ePubFormat;
    }

    public static string getOPFFileLocation (string extractionLocation) {
        info ("[START] [FUNCTION:getOPFFileLocation] extractionLocation=" + extractionLocation);
        string locationOfOPFFile = "false";
        //Form the path to the META-INF/container.xml file
        string pathToXMLFile = extractionLocation + "/" + BookwormApp.Constants.EPUB_META_INF_FILENAME;
        //Parse META-INF/container.xml file to locate the path to the OPF file
        ArrayList<XMLData> inputDataList = new ArrayList<XMLData> ();
        inputDataList.add (new XMLData () {
            containerTagName = "rootfiles",
            inputTagName = "rootfile",
            inputAttributeName = "full-path"
        });
        XmlParser thisParser = new XmlParser ();
        ArrayList<XMLData> extractedDataList = new ArrayList<XMLData> ();
        extractedDataList = thisParser.extractDataFromXML (pathToXMLFile, inputDataList);

        foreach (XMLData aExtractedData in extractedDataList) {
            foreach (string aAttributeValue in aExtractedData.extractedTagAttributes) {
                string OPFFilePath = aAttributeValue;
                locationOfOPFFile = extractionLocation + "/" + OPFFilePath;
            }
        }
        info ("[END] [FUNCTION:getOPFFileLocation] locationOfOPFFile=" + locationOfOPFFile);
        return locationOfOPFFile;
    }

    public static BookwormApp.Book determineToC (owned BookwormApp.Book aBook, string locationOfOPFFile) {
        info ("[START] [FUNCTION:determineToC] book.location=" + aBook.getBookLocation () + ", locationOfOPFFile=" + locationOfOPFFile);
        //Parse OPF xml file to read the MANIFEST data (id, href, media-type)
        ArrayList<XMLData> inputDataList = new ArrayList<XMLData> ();
        inputDataList.add (new XMLData () {
            containerTagName = "manifest",
            inputTagName = "item",
            inputAttributeName = "id"
        });
        inputDataList.add (new XMLData () {
            containerTagName = "manifest",
            inputTagName = "item",
            inputAttributeName = "href"
        });
        inputDataList.add (new XMLData () {
            containerTagName = "manifest",
            inputTagName = "item",
            inputAttributeName = "media-type"
        });
        inputDataList.add (new XMLData () {
            containerTagName = "spine",
            inputTagName = "itemref",
            inputAttributeName = "idref"
        });
        inputDataList.add (new XMLData () {
            containerTagName = "",
            inputTagName = "spine",
            inputAttributeName = "toc"
        });
        XmlParser thisParser = new XmlParser ();
        ArrayList<XMLData> opfItemsList = new ArrayList<XMLData> ();
        opfItemsList = thisParser.extractDataFromXML (locationOfOPFFile, inputDataList);

        if (opfItemsList.size > 3 && opfItemsList.get (4).extractedTagAttributes.size>0) {
            debug ("Successfully extracted SPINE data..");
            //Get the reference of the NCX file in the SPINE data
            string spineNCXReference = opfItemsList.get (4).extractedTagAttributes.get (0);
            debug ("Successfully determined NCX File Reference as:" + spineNCXReference);
            //Get the position of NCX Reference in MANIFEST data
            if (opfItemsList.size>0 && opfItemsList.get (0).extractedTagAttributes.contains (spineNCXReference)) {
                debug ("Successfully extracted MANIFEST data");
                int spineNCXPosition = opfItemsList.get (0).extractedTagAttributes.index_of (spineNCXReference);
                debug ("Successfully matched NCX File path information on MANIFEST data at position:" + spineNCXPosition.to_string ());
                //Get the location of the NCX file from the MANIFEST href attribute
                string NCXFileRelativePath = opfItemsList.get (1).extractedTagAttributes.get (spineNCXPosition);
                debug ("Extracted relative NCX file path from MANIFEST data as:" + NCXFileRelativePath);
                string ncxFilePath = (
                    BookwormApp.Utils.getFullPathFromFilename (
                        aBook.getBaseLocationOfContents (), NCXFileRelativePath.strip ())).strip ();
                if ("true" == BookwormApp.Utils.fileOperations ("EXISTS", "", ncxFilePath, "")) {
                    debug ("Successfully determined NCX File Path as:" + ncxFilePath);
                    //Parse NCX xml file to read the ToC data (id, href, media-type)
                    ArrayList<XMLData> inputDataListForToC = new ArrayList<XMLData> ();
                    inputDataListForToC.add (new XMLData () {
                        containerTagName = "navLabel",
                        inputTagName = "text",
                        inputAttributeName = ""
                    });
                    inputDataListForToC.add (new XMLData () {
                        containerTagName = "",
                        inputTagName = "content",
                        inputAttributeName = "src"
                    });
                    XmlParser ncxParser = new XmlParser ();
                    ArrayList<XMLData> ncxDataExtractedList = new ArrayList<XMLData> ();
                    ncxDataExtractedList = ncxParser.extractDataFromXML (ncxFilePath, inputDataListForToC);
                    if (ncxDataExtractedList.get (0).extractedTagValues.size > 0 &&
                        ncxDataExtractedList.get (1).extractedTagAttributes.size > 0 &&
                        ncxDataExtractedList.get (0).extractedTagValues.size == ncxDataExtractedList.get (1).extractedTagAttributes.size)
                    {
                        for (int count=0; count<ncxDataExtractedList.get (0).extractedTagValues.size; count++) {
                            HashMap<string,string> TOCMapItem = new HashMap<string,string> ();
                            string tocLocation = ncxDataExtractedList.get (1).extractedTagAttributes.get (count);
                            //Handle the links with anchor elements
                            string anchorValue = "";
                            if (tocLocation.index_of ("#") != -1 ) {
                                anchorValue = tocLocation.slice (tocLocation.index_of ("#"), tocLocation.length);
                                tocLocation = tocLocation.slice (0, tocLocation.index_of ("#"));
                            }
                            tocLocation = BookwormApp.Utils.getFullPathFromFilename (aBook.getBaseLocationOfContents (), tocLocation);
                            TOCMapItem.set (tocLocation + anchorValue, ncxDataExtractedList.get (0).extractedTagValues.get (count));
                            aBook.setTOC (TOCMapItem);
                            debug ("Extracted ToC Chapter Name:" + ncxDataExtractedList.get (0)
                                .extractedTagValues.get (count) + " at location:" + tocLocation + anchorValue);
                        }
                    }
                }
            }
        }

        // Create the content list - clear the content list of any previous items
        aBook.clearBookContentList ();
        //loop over all idref attributes in spine data
        foreach (string spineIDREF in opfItemsList[3].extractedTagAttributes) {
            //check if the SPINE IDREF exists in the MANIFEST Attributes
            if (opfItemsList[0].extractedTagAttributes.contains (spineIDREF)) {
                int positionOfIDREF = opfItemsList[0].extractedTagAttributes.index_of (spineIDREF);
                //extract the HREF from MANIFEST corresponding to the SPINE IDREF
                string locationOfContentData = opfItemsList[1].extractedTagAttributes.get (positionOfIDREF);
                aBook.setBookContentList (aBook.getBaseLocationOfContents () + locationOfContentData);
                debug ("Book content data :" + aBook.getBaseLocationOfContents () + locationOfContentData);
            }
        }
        info ("[END] [FUNCTION:determineToC]");
        return aBook;
    }

    public static BookwormApp.Book setCoverImage (owned BookwormApp.Book aBook, string locationOfOPFFile) {
        info ("[START] [FUNCTION:setCoverImage] book.location=" + aBook.getBookLocation () + ", locationOfOPFFile=" + locationOfOPFFile);
        string bookCoverLocation = "";
        //Parse OPF xml file to read the MANIFEST data
        ArrayList<XMLData> inputDataList = new ArrayList<XMLData> ();
        inputDataList.add (new XMLData () {
            containerTagName = "manifest",
            inputTagName = "item",
            inputAttributeName = "id"
        });
        inputDataList.add (new XMLData () {
            containerTagName = "manifest",
            inputTagName = "item",
            enforceAttributeData = true,
            inputAttributeName = "media-type"
        });
        inputDataList.add (new XMLData () {
            containerTagName = "manifest",
            inputTagName = "item",
            inputAttributeName = "href"
        });
        inputDataList.add (new XMLData () {
            containerTagName = "manifest",
            inputTagName = "item",
            enforceAttributeData = true,
            inputAttributeName = "properties"
        });
        XmlParser thisParser = new XmlParser ();
        ArrayList<XMLData> opfItemsList = new ArrayList<XMLData> ();
        opfItemsList = thisParser.extractDataFromXML (locationOfOPFFile, inputDataList);
        int count = 0;
        //epub3.1 : Check for a MANIFEST item with "properties" attribute contaning the word "cover-image"
        foreach (string properties in opfItemsList[3].extractedTagAttributes) {
            if (properties.contains ("cover-image")) {
                //Get media type for the cover items
                string coverMediaType = opfItemsList[1].extractedTagAttributes.get (count);
                //get cover location if media type matches "image"
                if (coverMediaType.index_of ("image") != -1) {
                    bookCoverLocation = opfItemsList[2].extractedTagAttributes.get (count);
                    bookCoverLocation = aBook.getBaseLocationOfContents () + bookCoverLocation;
                    break;
                }
            }
            count++;
        }
        //If cover could not be located in properties="cover-image" :
        //Check for a MANIFEST item with "id" attribute contaning the word "cover"
        if (bookCoverLocation.length < 1 &&
            "true" == BookwormApp.Utils.fileOperations ("EXISTS", "", bookCoverLocation, "") )
        {
            count = 0;
            foreach (string id in opfItemsList[0].extractedTagAttributes) {
                if (id.contains ("cover") ) {
                    //Get media type for the cover items
                    string coverMediaType = opfItemsList[1].extractedTagAttributes.get (count);
                    //get cover location if media type matches "image"
                    if (coverMediaType.index_of ("image") != -1) {
                        bookCoverLocation = opfItemsList[2].extractedTagAttributes.get (count);
                        bookCoverLocation = aBook.getBaseLocationOfContents () + bookCoverLocation;
                        break;
                    }
                }
                count++;
            }
        }
        //check if cover was still not found and assign flag for default cover to be used
        if (bookCoverLocation.length < 1 &&
            "true" == BookwormApp.Utils.fileOperations ("EXISTS", "", bookCoverLocation, ""))
        {
            aBook.setIsBookCoverImagePresent (false);
            debug ("Cover image not found for book located at:" + aBook.getBookExtractionLocation ());
        } else {
            //copy cover image to bookworm cover image cache
            aBook = BookwormApp.Utils.setBookCoverImage (aBook, bookCoverLocation);
        }
        info ("[END] [FUNCTION:setCoverImage] book.location=" + aBook.getBookLocation () + ", bookCoverLocation=" + bookCoverLocation);
        return aBook;
    }

    public static BookwormApp.Book setBookMetaData (owned BookwormApp.Book aBook, string locationOfOPFFile) {
        info ("[START] [FUNCTION:setBookMetaData] book.location=" +
            aBook.getBookLocation () + ", locationOfOPFFile=" + locationOfOPFFile);
        //Parse OPF xml file to read the book meta data
        ArrayList<XMLData> inputDataList = new ArrayList<XMLData> ();
        inputDataList.add (new XMLData () {
            containerTagName = "",
            inputTagName = "title",
            inputAttributeName = ""
        });
        inputDataList.add (new XMLData () {
            containerTagName = "",
            inputTagName = "creator",
            inputAttributeName = ""
        });
        XmlParser thisParser = new XmlParser ();
        ArrayList<XMLData> opfItemsList = new ArrayList<XMLData> ();
        opfItemsList = thisParser.extractDataFromXML (locationOfOPFFile, inputDataList);
        if (opfItemsList[0].extractedTagValues.size > 0) {
            string bookTitle = opfItemsList[0].extractedTagValues.get (0);
            if (bookTitle.length > 0) {
                aBook.setBookTitle (BookwormApp.Utils.decodeHTMLChars (bookTitle));
                debug ("Determined eBook Title as:" + bookTitle);
            } else {
                //If the book title has not been determined, use the file name as book title
                if (aBook.getBookTitle () != null && (
                                    aBook.getBookTitle () == BookwormApp.Constants.TEXT_FOR_UNKNOWN_TITLE ||
                                    aBook.getBookTitle ().length < 1
                                )
                ) {
                    bookTitle = File.new_for_path (aBook.getBookExtractionLocation ()).get_basename ();
                    if (bookTitle.last_index_of (".") != -1) {
                        bookTitle = bookTitle.slice (0, bookTitle.last_index_of ("."));
                    }
                    aBook.setBookTitle (bookTitle);
                    debug ("File name set as Title:" + bookTitle);
                }
            }
        }
        //determine the author of the book
        if (opfItemsList[1].extractedTagValues.size > 0) {
            string bookAuthor = opfItemsList[1].extractedTagValues.get (0);
            if (bookAuthor.length > 0) {
                aBook.setBookAuthor (BookwormApp.Utils.decodeHTMLChars (bookAuthor));
                debug ("Determined eBook Author as:" + bookAuthor);
            } else {
                //If the book author has not been determined, use a default text for author
                aBook.setBookAuthor (BookwormApp.Constants.TEXT_FOR_UNKNOWN_TITLE);
                debug ("Could not determine eBook Author, default Author set");
            }
        }
        info ("[END] [FUNCTION:setBookMetaData]");
        return aBook;
    }
}
```

```
======================================================================
contentHandler.vala  ?  Content Handler
======================================================================
```

```
/* Copyright 2017 Siddhartha Das (bablu.boy@gmail.com)
*
* This file is part of Bookworm and is used for handling the eBook contents
* The prerequisite for the content handler is for the eBook contents to have
* been parsed into HTML format
*
* Bookworm is free software: you can redistribute it
* and/or modify it under the terms of the GNU General Public License as
* published by the Free Software Foundation, either version 3 of the
* License, or (at your option) any later version.
*
* Bookworm is distributed in the hope that it will be
* useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
* Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with Bookworm. If not, see http://www.gnu.org/licenses/.
*/

using Gee;
public class BookwormApp.contentHandler {
    public static BookwormApp.Settings settings;

    public static BookwormApp.Book renderPage (owned BookwormApp.Book aBook, owned string direction) {
        debug ("[START] [FUNCTION:renderPage] book.location=" + aBook.getBookLocation () + ", direction=" + direction);
        int currentContentLocation = aBook.getBookPageNumber ();
        bool shouldReloadPage = true;
        //set page number based on direction of navigation
        switch (direction) {
            case "FORWARD"://This is for moving the book forward
                if (aBook.getIfPageForward ()) {
                    currentContentLocation++;
                    aBook.setBookPageNumber (currentContentLocation);
                }
                break;
            case "BACKWARD"://This is for moving the book backwards
                if (aBook.getIfPageBackward ()) {
                    currentContentLocation--;
                    aBook.setBookPageNumber (currentContentLocation);
                }
                break;
            case "SEARCH"://Load the page and scroll to the search text
                break;
            default://This is for opening the current page of the book
                    //Do not change the page number
                break;
        }
        string bookContent = contentHandler.provideContent (aBook,currentContentLocation, direction);
        //render the content on webview
        BookwormApp.AppWindow.aWebView.load_html (bookContent, BookwormApp.Constants.PREFIX_FOR_FILE_URL);
        //set the bookmak icon on the header
        handleBookMark ("DISPLAY");
        //set the navigation controls
        aBook = controlNavigation (aBook);
        //set the current value of the page slider
        BookwormApp.AppWindow.pageAdjustment.set_value (currentContentLocation + 1);
        debug ("[END] [FUNCTION:renderPage]");
        return aBook;
    }

    public static string provideContent (owned BookwormApp.Book aBook, int contentLocation, string mode) {
        debug ("[START] [FUNCTION:provideContent] book.location=" + aBook.getBookLocation () +
            ", contentLocation=" + contentLocation.to_string () + ", mode=" + mode);
        StringBuilder contents = new StringBuilder ();
        if (aBook.getBookContentList () != null) {
            string bookLocationToRead = "";
            if (contentLocation > -1 && aBook.getBookContentList ().size > contentLocation) {
                bookLocationToRead = aBook.getBookContentList ().get (contentLocation);
                if("true" != BookwormApp.Utils.fileOperations ("EXISTS", bookLocationToRead, "", "")){
                    //handle the case when the content list has html escape chars for the URI
                    bookLocationToRead = BookwormApp.Utils.decodeHTMLChars (aBook.getBookContentList ().get (contentLocation));
                    if("true" != BookwormApp.Utils.fileOperations ("EXISTS", bookLocationToRead, "", "")){
                        //requested content not available
                        aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_NAVIGATION_ISSUE);
                        BookwormApp.AppWindow.showInfoBar (aBook, Gtk.MessageType.WARNING);
                        warning ("[END] [FUNCTION:provideContent] Page could not be loaded from location:"+bookLocationToRead);
                        return contents.str;
                    }
                }
                //fetch content from extracted book
                contents.assign (BookwormApp.Utils.fileOperations ("READ_FILE", bookLocationToRead, "", ""));
                //find list of relative urls with src, href, etc and convert them to absolute ones
                foreach (string tagname in BookwormApp.Constants.TAG_NAME_WITH_PATHS) {
                    string[] srcList = BookwormApp.Utils.multiExtractBetweenTwoStrings (contents.str, tagname, "\"");
                    StringBuilder srcItemFullPath = new StringBuilder ();
                    foreach (string srcItem in srcList) {
                        srcItemFullPath.assign (
                                BookwormApp.Utils.getFullPathFromFilename (
                                            aBook.getBookExtractionLocation (), srcItem)
                        );
                        contents.assign (
                            contents.str.replace (tagname + srcItem + "\"",
                            BookwormApp.Utils.encodeHTMLChars (tagname + srcItemFullPath.str) + "\""));
                    }
                }
                //update the content for required manipulation
                contents.assign (adjustPageContent (aBook, contents.str, mode));
            //handle the case for contentLocation set to -1 when the book is added to the DB
            } else if (contentLocation == -1 && aBook.getBookContentList ().size > 0) {
                provideContent (aBook, 0, mode);
            } else {
                //requested content not available
                aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_NAVIGATION_ISSUE);
                BookwormApp.AppWindow.showInfoBar (aBook, Gtk.MessageType.WARNING);
            }
        } else {
            //requested content not available
            aBook.setParsingIssue (BookwormApp.Constants.TEXT_FOR_CONTENT_NOT_FOUND_ISSUE);
            BookwormApp.AppWindow.showInfoBar (aBook, Gtk.MessageType.WARNING);
        }
        debug ("[END] [FUNCTION:provideContent] contents.length=" + contents.str.length.to_string ());
        return contents.str;
    }

    public static void handleBookMark (string action) {
        debug ("[START] [FUNCTION:handleBookMark] action=" + action);
        //get the book being currently read
        BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap.get (BookwormApp.Bookworm.locationOfEBookCurrentlyRead);
        switch (action) {
            case "DISPLAY":
                if (aBook != null && aBook.getBookmark () != null && aBook.getBookmark ().index_of (aBook.getBookPageNumber ().to_string ()) != -1) {
                    //display bookmark as active
                    BookwormApp.AppHeaderBar.bookmark_active_button.set_visible (true);
                    BookwormApp.AppHeaderBar.bookmark_inactive_button.set_visible (false);
                } else {
                    //display bookmark as inactive
                    BookwormApp.AppHeaderBar.bookmark_active_button.set_visible (false);
                    BookwormApp.AppHeaderBar.bookmark_inactive_button.set_visible (true);
                }
                break;
            case "ACTIVE_CLICKED":
                BookwormApp.AppHeaderBar.bookmark_active_button.set_visible (false);
                BookwormApp.AppHeaderBar.bookmark_inactive_button.set_visible (true);
                //set the bookmark
                aBook.setBookmark (aBook.getBookPageNumber (), action);
                break;
            case "INACTIVE_CLICKED":
                BookwormApp.AppHeaderBar.bookmark_active_button.set_visible (true);
                BookwormApp.AppHeaderBar.bookmark_inactive_button.set_visible (false);
                //set the bookmark
                aBook.setBookmark (aBook.getBookPageNumber (), action);
                break;
            default:
                break;
        }
        //update book details to libraryView Map
        if (aBook != null) {
            debug ("updating libraryViewMap with bookmark info...");
            BookwormApp.Bookworm.libraryViewMap.set (BookwormApp.Bookworm.locationOfEBookCurrentlyRead, aBook);
        }
        debug ("[END] [FUNCTION:handleBookMark]");
    }

    public static string adjustPageContent (BookwormApp.Book aBook, owned string pageContentStr, string mode) {
        debug ("[START] [FUNCTION:adjustPageContent] book.location=" + aBook.getBookLocation () +
            ", pageContentStr.length=" + pageContentStr.length.to_string () + ", mode=" + mode);
        //wrap the html content in a div tag for pagination
        StringBuilder pageContent = new StringBuilder (pageContentStr);
        //Remove the empty title if it is present
        pageContent.assign (pageContent.str
            .replace ("<title/>","")
        );
        constructOnLoadJavascript(aBook, mode);
        string currentBookwormStyles = createPageStyles(aBook, pageContentStr);

        BookwormApp.AppWindow.aWebView.get_user_content_manager().remove_all_style_sheets();
        BookwormApp.AppWindow.aWebView.get_user_content_manager().add_style_sheet(
        new WebKit.UserStyleSheet(currentBookwormStyles,
                                  WebKit.UserContentInjectedFrames.ALL_FRAMES,
                                  WebKit.UserStyleLevel.AUTHOR,
                                  null, null)
        );
        debug ("[END] [FUNCTION:adjustPageContent] pageContent.length=" + pageContent.str.length.to_string ());
        debug ("\n\n\n" + pageContent.str);
        return pageContent.str;
    }

    private static string createPageStyles(BookwormApp.Book aBook, owned string pageContentStr) {
        if (BookwormApp.Bookworm.bookwormStyles == null || BookwormApp.Bookworm.bookwormStyles.length < 1) {
            uint8[] bookwormStylesData;
            GLib.File.new_for_uri (BookwormApp.Constants.HTML_SCRIPT_STYLES_RESOURCE_LOCATION)
                .load_contents (null, out bookwormStylesData, null);
            BookwormApp.Bookworm.bookwormStyles = (string)bookwormStylesData;
            debug ("Loaded styles data from resource:\n" + BookwormApp.Bookworm.bookwormStyles);
        }
        string currentBookwormStyles = BookwormApp.Bookworm.bookwormStyles;
        string cssForTextAndBackgroundColor = "";

        //For the Title Page (first or second page), resize height and width of images
        if (aBook.getBookPageNumber () < 2 && (pageContentStr.contains ("<image") || pageContentStr.contains ("<img"))) {
            currentBookwormStyles = currentBookwormStyles.replace ("$TITLE_PAGE_IMAGE", "img, image");
        }
        //Set background and font colour based on profile
        if (BookwormApp.Constants.BOOKWORM_READING_MODE[4] == BookwormApp.Bookworm.settings.reading_profile) {
            //default dark profile
            cssForTextAndBackgroundColor = " background-color: #002b36 !important; color: #93a1a1 !important;";
            currentBookwormStyles = currentBookwormStyles
                .replace ("$SCROLLBAR_BACKGROUND", "#002b36")
                .replace ("$HIGHLIGHT_COLOR", "#3465A4");
        } else if (BookwormApp.Constants.BOOKWORM_READING_MODE[3] == BookwormApp.Bookworm.settings.reading_profile) {
            //default light profile
            cssForTextAndBackgroundColor = " background-color: #fbfbfb !important; color: #000000 !important;";
            currentBookwormStyles = currentBookwormStyles
                .replace ("$SCROLLBAR_BACKGROUND", "#fbfbfb")
                .replace ("$HIGHLIGHT_COLOR", "#E8ED00");
        } else if (BookwormApp.Constants.BOOKWORM_READING_MODE[2] == BookwormApp.Bookworm.settings.reading_profile) {
            cssForTextAndBackgroundColor =
                " background-color: " + BookwormApp.Bookworm.profileColorList[7] + " !important;" +
                           " color: " + BookwormApp.Bookworm.profileColorList[6] + " !important;";
            currentBookwormStyles = currentBookwormStyles
                .replace ("$SCROLLBAR_BACKGROUND", BookwormApp.Bookworm.profileColorList[7])
                .replace ("$HIGHLIGHT_COLOR",      BookwormApp.Bookworm.profileColorList[8]);
        } else if (BookwormApp.Constants.BOOKWORM_READING_MODE[1] == BookwormApp.Bookworm.settings.reading_profile) {
            cssForTextAndBackgroundColor =
                " background-color: " + BookwormApp.Bookworm.profileColorList[4] + " !important;" +
                           " color: " + BookwormApp.Bookworm.profileColorList[3] + " !important;";
            currentBookwormStyles = currentBookwormStyles
                .replace ("$SCROLLBAR_BACKGROUND", BookwormApp.Bookworm.profileColorList[4])
                .replace ("$HIGHLIGHT_COLOR",      BookwormApp.Bookworm.profileColorList[5]);
        } else {
            cssForTextAndBackgroundColor =
                " background-color: " + BookwormApp.Bookworm.profileColorList[1] + " !important;" +
                           " color: " + BookwormApp.Bookworm.profileColorList[0] + " !important;";
            currentBookwormStyles = currentBookwormStyles
                .replace ("$SCROLLBAR_BACKGROUND", BookwormApp.Bookworm.profileColorList[1])
                .replace ("$HIGHLIGHT_COLOR",      BookwormApp.Bookworm.profileColorList[2]);
        }
        //Set up CSS for book as per preference settings - this will override any css in the book contents
        currentBookwormStyles = currentBookwormStyles
            .replace ("$READING_LINE_HEIGHT", BookwormApp.Bookworm.settings.reading_line_height)
            .replace ("$READING_WIDTH", (100 - int.parse (BookwormApp.Bookworm.settings.reading_width)).to_string ())
            .replace ("$FONT_FAMILY", BookwormApp.Bookworm.settings.reading_font_name_family)
            .replace ("$FONT_SIZE", BookwormApp.Bookworm.settings.reading_font_size.to_string ())
            .replace ("$READING_TEXT_ALIGN", BookwormApp.Bookworm.settings.text_alignment)
            .replace ("$TEXT_AND_BACKGROUND_COLOR", cssForTextAndBackgroundColor);
        return currentBookwormStyles;
    }

    private static void constructOnLoadJavascript(BookwormApp.Book aBook, string mode) {
        settings = BookwormApp.Settings.get_instance ();
        //BookwormApp.Bookworm.onLoadJavaScript.assign ("onload=\"");
        BookwormApp.Bookworm.onLoadJavaScript.assign ("");

        //Scroll to the previous vertical position - this should be used:
        // (1) when the book is re-opened from the library and
        // (2) when a book existing in the library is opened from File Explorer using Bookworm
        // (3) when clicking on a link in the TableOfContents which has an anchor
        //The flag for applying the javascript is set from the above locations
        if (BookwormApp.Bookworm.isPageScrollRequired) {
            //check if an Anchor is present and set up the javascript for the same
            if (aBook.getAnchor ().length > 0) {
                BookwormApp.Bookworm.onLoadJavaScript.append (
                " document.getElementById('" + aBook.getAnchor () + "').scrollIntoView();"
                );
                aBook.setAnchor("");
            } else { //set up the javascript for scrolling to last read position
                BookwormApp.Bookworm.onLoadJavaScript.append (" window.scrollTo (0," + (
                    BookwormApp.Bookworm.libraryViewMap.get (
                        BookwormApp.Bookworm.locationOfEBookCurrentlyRead))
                        .getBookScrollPos ().to_string () + ");");
            }
            BookwormApp.Bookworm.isPageScrollRequired = false; // stop this function being called subsequently
        }
        //If two page view id required - add a script to set the CSS for two-page if there are more than 500 chars
        if (BookwormApp.Bookworm.settings.is_two_page_enabled) {
            BookwormApp.Bookworm.onLoadJavaScript.append (" setTwoPageView ();");
        }
        //Overlay any Annotated text
        foreach (var entry in aBook.getAnnotationList ().entries) {
            if (aBook.getBookPageNumber ().to_string () == entry.key.split ("#~~#")[0]) {
                BookwormApp.Bookworm.onLoadJavaScript.append (" overlayAnnotation ('" + entry.key.split ("#~~#")[1] + "');");
            }
        }

        //Highlight and Scroll To Search String on page if required
        if ("SEARCH" == mode) {
            if (BookwormApp.Bookworm.bookTextSearchString.length > 1) {
                string[] searchTokens = BookwormApp.Bookworm.bookTextSearchString.split ("#~~#");
                if (searchTokens.length == 2) {
                    //limit the search string to one word on either side of search text
                    int startPosOfSearchString = searchTokens[1].index_of (searchTokens[0]);
                    int endPosOfSearchString = startPosOfSearchString + searchTokens[0].length;
                    int lengthOfLineWithSearchString = searchTokens[1].length;
                    int countSpaces = 0;
                    int startPosOfStringToBeHighlighted = 0;
                    int endPosOfStringToBeHighlighted = 0;
                    string stringToBeHighlighted = "";
                    if (startPosOfSearchString != -1) {
                        //get the position of the word before the searched phrase
                        for (int i=startPosOfSearchString; i>1; i--) {
                            //match the second space before the search string
                            if (" " == searchTokens[1].slice (i, i + 1)) {
                                countSpaces++;
                            }
                            if (countSpaces == 2) {
                                startPosOfStringToBeHighlighted = i + 1;
                                break;
                            }
                        }
                        //get the position of the word after the searched phrase
                        countSpaces = 0;
                        for (int j=endPosOfSearchString; j<lengthOfLineWithSearchString; j++) {
                            //match the second space before the search string
                            if (" " == searchTokens[1].slice (j, j + 1)) {
                                countSpaces++;
                            }
                            if (countSpaces == 2) {
                                endPosOfStringToBeHighlighted = j;
                                break;
                            }
                        }
                        //form the string to be highlighted
                        if (endPosOfStringToBeHighlighted > startPosOfStringToBeHighlighted) {
                            stringToBeHighlighted = searchTokens[1].slice (
                                startPosOfStringToBeHighlighted, endPosOfStringToBeHighlighted
                            );
                        }
                    }
                    stringToBeHighlighted = stringToBeHighlighted.replace ("\"", "&quot;").replace ("'", "&#39;");
                    debug ("Searching to highlight the phrase:" + stringToBeHighlighted);
                    BookwormApp.Bookworm.onLoadJavaScript
                        .append (" highlightText(encodeURIComponent(’" + stringToBeHighlighted + "’));");
                }
            }
        }
        //complete the onload javascript string
        //BookwormApp.Bookworm.onLoadJavaScript.append ("\"");
    }

    public static void searchHTMLContents () {
        debug ("[START] [FUNCTION:searchHTMLContents]");
        StringBuilder bookSearchResults = new StringBuilder ("");
        int searchResultCount = 1;
        BookwormApp.Bookworm.searchResultsMap.clear ();
        //execute search
        bookSearchResults.assign (
            BookwormApp.Utils.execute_sync_command (
                BookwormApp.Constants.SEARCH_SCRIPT_LOCATION +
                " \"" + BookwormApp.Bookworm.aContentFileToBeSearched.str + "\" \"" +
                BookwormApp.AppHeaderBar.headerSearchBar.get_text () + "\""));
        //process search results
        if (bookSearchResults.str.strip ().length > 0 && bookSearchResults.str != "false") {
            string[] individualLines = bookSearchResults.str.strip ().split ("\n",-1);
            foreach (string individualLine in individualLines) {
                BookwormApp.Bookworm.searchResultsMap.set (
                    searchResultCount.to_string () + "~~" +
                    BookwormApp.Bookworm.aContentFileToBeSearched.str,
                    individualLine.strip ());
                searchResultCount++;
            }
        }
        debug ("[END] [FUNCTION:searchHTMLContents]");
    }

    public static BookwormApp.Book controlNavigation (owned BookwormApp.Book aBook) {
        info ("[START] [FUNCTION:controlNavigation] book.location=" + aBook.getBookLocation ());
        int currentContentLocation = aBook.getBookPageNumber ();
        debug ("In controlNavigation with currentContentLocation=" + currentContentLocation.to_string ());
        //check if Book can be moved back and disable back button otherwise
        if (currentContentLocation > 0) {
            aBook.setIfPageBackward (true);
            BookwormApp.AppWindow.back_button.set_sensitive (true);
        } else {
            aBook.setIfPageBackward (false);
            BookwormApp.AppWindow.back_button.set_sensitive (false);
        }
        //check if Book can be moved forward and disable forward button otherwise
        if (currentContentLocation < (aBook.getBookContentList ().size - 1)) {
            aBook.setIfPageForward (true);
            BookwormApp.AppWindow.forward_button.set_sensitive (true);
        } else {
            aBook.setIfPageForward (false);
            BookwormApp.AppWindow.forward_button.set_sensitive (false);
        }
        info ("[END] [FUNCTION:controlNavigation] book.location=" + aBook.getBookLocation ());
        return aBook;
    }

    public static void refreshCurrentPage () {
        debug ("[START] [FUNCTION:refreshCurrentPage]");
        if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[1]) {
            BookwormApp.Book currentBookForRefresh = BookwormApp.Bookworm.libraryViewMap.get (
                BookwormApp.Bookworm.locationOfEBookCurrentlyRead);
            BookwormApp.Bookworm.isPageScrollRequired = true; //set up the flag to scroll to the last read position
            currentBookForRefresh = renderPage (
                BookwormApp.Bookworm.libraryViewMap.get (
                BookwormApp.Bookworm.locationOfEBookCurrentlyRead), "");
            BookwormApp.Bookworm.libraryViewMap.set (
                BookwormApp.Bookworm.locationOfEBookCurrentlyRead,
                currentBookForRefresh);
        }
        debug ("[END] [FUNCTION:refreshCurrentPage]");
    }

    public static int getScrollPos () {
        debug ("[START] [FUNCTION:getScrollPos]");
        //This function is responsible for returning the vertical scroll position of the webview
        //This should be called when the user leaves reading a book :
        // (1) Return to Library button on Header Bar
        // (2) Close Bookworm while in reading mode
        // (3) Move to info view using Info button on Header Bar
        int scrollPos = -1;
        scrollPos = int.parse (BookwormApp.Utils.setWebViewTitle ("document.title = window.scrollY;"));
        debug ("[START] [FUNCTION:getScrollPos] scrollPos=" + scrollPos.to_string ());
        return scrollPos;
    }

    public static void performStartUpActions () {
        debug ("[START] [FUNCTION:performStartUpActions]");
        //load javascript data from resource if it has not been loaded already
        if (BookwormApp.Bookworm.bookwormScripts == null || BookwormApp.Bookworm.bookwormScripts.length < 1) {
            uint8[] bookwormScriptsData;
            GLib.File.new_for_uri (BookwormApp.Constants.HTML_SCRIPT_FUNCTIONS_RESOURCE_LOCATION)
                .load_contents (null, out bookwormScriptsData, null);
            BookwormApp.Bookworm.bookwormScripts = (string)bookwormScriptsData;
            WebKit.UserScript userScript = new WebKit.UserScript(BookwormApp.Bookworm.bookwormScripts, WebKit.UserContentInjectedFrames.ALL_FRAMES, WebKit.UserScriptInjectionTime.END, null, null);
            BookwormApp.AppWindow.aWebView.get_user_content_manager().add_script(userScript);
            debug ("Loaded javascript data from resource:\n" + BookwormApp.Bookworm.bookwormScripts);
        }
        //open the book added, if only one book path is present on command line
        //if this book was not in the library, then the library view will be shown
        if (BookwormApp.Bookworm.pathsOfBooksToBeAdded.length == 2 && //check if only one book is on the command line
            //check if first parameter is bookworm
            BookwormApp.Constants.bookworm_id == BookwormApp.Bookworm.pathsOfBooksToBeAdded[0] &&
            //check if book has not already failed to load
            BookwormApp.Bookworm.pathsOfBooksNotAddedStr.str.index_of (BookwormApp.Bookworm.pathsOfBooksToBeAdded[1]) == -1)
        {
            BookwormApp.Book requestedBook = null;
            //Check if the requested book is available in the library
            if (BookwormApp.Bookworm.pathsOfBooksInLibraryOnLoadStr.str.index_of (
                BookwormApp.Bookworm.commandLineArgs[1].strip ()) != -1)
            {
                //pick the book from the Initial ArrayList used for holding the books in the library
                //as the BookwormApp.Bookworm.libraryViewMap would not have finished loading
                foreach (BookwormApp.Book aBook in BookwormApp.Library.listOfBooksInLibraryOnLoad) {
                    if (BookwormApp.Bookworm.commandLineArgs[1].strip () == aBook.getBookLocation ()) {
                        requestedBook = aBook;
                        break;
                    }
                }
            } else {
                //pick the book from the BookwormApp.Bookworm.libraryViewMap as it would have been added
                //as part of the code above to create a new book
                requestedBook = BookwormApp.Bookworm.libraryViewMap.get (BookwormApp.Bookworm.commandLineArgs[1].strip ());
            }
            debug ("Bookworm opened for single book[" + requestedBook.getBookLocation () + "] - proceed to reading view...");
            if (requestedBook != null) {
                //set the name of the book being currently read
                BookwormApp.Bookworm.locationOfEBookCurrentlyRead = BookwormApp.Bookworm.commandLineArgs[1].strip ();
                //Initiate Reading the book
                BookwormApp.Bookworm.readSelectedBook (requestedBook);
            }
        } else {
            //check and continue the last book being read - if "Always show library on start is false"
            if ((!BookwormApp.Bookworm.settings.is_show_library_on_start) && (BookwormApp.Bookworm.settings.book_being_read != "")) {
                //check if the library contains the book being read last
                BookwormApp.Book lastReadBook = BookwormApp.DB.getBookFromDB (BookwormApp.Bookworm.settings.book_being_read);
                if (lastReadBook.getBookLocation () != null && lastReadBook.getBookLocation ().length > 1) {
                    debug ("Opening the last read book [" + BookwormApp.Bookworm.settings.book_being_read + "]");
                    BookwormApp.Bookworm.locationOfEBookCurrentlyRead = BookwormApp.Bookworm.settings.book_being_read;
                    //Initiate Reading the book
                    BookwormApp.Bookworm.readSelectedBook (lastReadBook);
                } else {
                    warning ("The last read book [" + BookwormApp.Bookworm.settings.book_being_read + "] " +
                        "was not found in the library, so showing the library view instead of opening the last read book");
                }
            }
        }
        debug ("[END] [FUNCTION:performStartUpActions]");
    }
}
```

==
window.vala ? Main Window
==

```
/* Copyright 2017 Siddhartha Das (bablu.boy@gmail.com)
*
* This file is part of Bookworm and is used for drawing the
* window components for both the library view and the reading view
*
* Bookworm is free software: you can redistribute it
* and/or modify it under the terms of the GNU General Public License as
* published by the Free Software Foundation, either version 3 of the
* License, or (at your option) any later version.
*
* Bookworm is distributed in the hope that it will be
* useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
* Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with Bookworm. If not, see http://www.gnu.org/licenses/.
*/
using Gtk;
using Gee;
using Granite.Widgets;

public class BookwormApp.AppWindow {
    public static Gtk.InfoBar infobar;
    public static Gtk.Box bookLibrary_ui_box;
    public static Gtk.FlowBox library_grid;
    public static Gtk.ListStore library_table_liststore;
    public static TreeIter library_table_iter;
    public static Gtk.TreeView library_table_treeview;
    public static Gtk.Label infobarLabel;
    public static ScrolledWindow library_grid_scroll;
    public static ScrolledWindow library_list_scroll;
    public static WebKit.WebView aWebView;
    public static WebKit.Settings webkitSettings;
    public static Gtk.EventBox book_reading_footer_eventbox;
    public static Gtk.ActionBar book_reading_footer_box;
    public static Gtk.Box bookReading_ui_box;
    public static Gtk.Button forward_button;
    public static Gtk.GestureSwipe gesture_swipe;
    public static Gtk.Button back_button;
    public static Gtk.ProgressBar bookAdditionBar;
    public static Adjustment pageAdjustment;
    public static Scale pageSlider;
    public static BookwormApp.Settings settings;
    public static bool isWebViewRequestCompleted = true;
    public static Gtk.Button remove_book_button;
    public static Gtk.Button page_button_prev;
    public static Gtk.Button page_button_next;
    public static int noOfBooksSelected = 0;


    public static Gtk.Box createBoookwormUI () {
        info ("[START] [FUNCTION:createBoookwormUI]");
        settings = BookwormApp.Settings.get_instance ();

        //Create a grid to display the books cover images in library
        library_grid = new Gtk.FlowBox ();
        library_grid.set_border_width (BookwormApp.Constants.SPACING_WIDGETS);
        library_grid.column_spacing = BookwormApp.Constants.SPACING_WIDGETS;
        library_grid.row_spacing = BookwormApp.Constants.SPACING_WIDGETS;
        library_grid.set_valign (Gtk.Align.START);

        library_grid_scroll = new ScrolledWindow (null, null);
        library_grid_scroll.set_policy (PolicyType.AUTOMATIC, PolicyType.AUTOMATIC);
        library_grid_scroll.add (library_grid);

        //Create a treeview and Liststore to display the list of books in the library
        library_table_liststore = new Gtk.ListStore (8,
            typeof (Gdk.Pixbuf), typeof (string), typeof (string), typeof (string),
            typeof (Gdk.Pixbuf), typeof (string), typeof (string), typeof (string));
        library_table_treeview = new Gtk.TreeView ();
        library_table_treeview.activate_on_single_click = true;
        //Set up the various cell types for the library metadata
        CellRendererPixbuf selection_cell_pix = new CellRendererPixbuf ();
        CellRendererText non_editable_cell_txt = new CellRendererText ();
        CellRendererText title_cell_txt = new CellRendererText ();
        title_cell_txt.editable = true;
        CellRendererText author_cell_txt = new CellRendererText ();
        author_cell_txt.editable = true;
        CellRendererPixbuf rating_cell_pix = new CellRendererPixbuf ();
        CellRendererText tags_cell_txt = new CellRendererText ();
        tags_cell_txt.editable = true;
        //Set up Treeview columns
        library_table_treeview.insert_column_with_attributes (-1, " ", selection_cell_pix, "pixbuf", 0);
        library_table_treeview.insert_column_with_attributes (-1, BookwormApp.Constants.TEXT_FOR_LIST_VIEW_COLUMN_NAME_TITLE, title_cell_txt, "text", 1);
        library_table_treeview.insert_column_with_attributes (-1, BookwormApp.Constants.TEXT_FOR_LIST_VIEW_COLUMN_NAME_AUTHOR, author_cell_txt, "text", 2);
        library_table_treeview.insert_column_with_attributes (-1, BookwormApp.Constants.TEXT_FOR_LIST_VIEW_COLUMN_NAME_MODIFIED_DATE, non_editable_cell_txt, "text", 3);
        library_table_treeview.insert_column_with_attributes (-1, BookwormApp.Constants.TEXT_FOR_LIST_VIEW_COLUMN_NAME_RATING, rating_cell_pix, "pixbuf", 4);
        library_table_treeview.insert_column_with_attributes (-1, BookwormApp.Constants.TEXT_FOR_LIST_VIEW_COLUMN_NAME_TAGS, tags_cell_txt, "text", 5);

        library_list_scroll = new ScrolledWindow (null, null);
        library_list_scroll.set_policy (PolicyType.AUTOMATIC, PolicyType.AUTOMATIC);
        library_list_scroll.add (library_table_treeview);

        //Create a box to hold the grid view and list view - only one is visible at a time
        Gtk.Box library_view_box = new Gtk.Box (Orientation.VERTICAL, 0);
        library_view_box.set_border_width (0);
        library_view_box.pack_start (library_grid_scroll, true, true, 0);
        library_view_box.pack_start (library_list_scroll, true, true, 0);
        //Set up Button for selecting books
        Gtk.Button select_book_button = new Gtk.Button ();
        select_book_button.set_image (BookwormApp.Bookworm.select_book_image);
        select_book_button.set_relief (ReliefStyle.NONE);
        select_book_button.set_tooltip_markup (BookwormApp.Constants.TOOLTIP_TEXT_FOR_SELECT_BOOK);

        //Set up Button for adding books
        Gtk.Button add_book_button = new Gtk.Button ();
        add_book_button.set_image (BookwormApp.Bookworm.add_book_image);
        add_book_button.set_relief (ReliefStyle.NONE);
        add_book_button.set_tooltip_markup (BookwormApp.Constants.TOOLTIP_TEXT_FOR_ADD_BOOK);

        //Set up Button for removing books
        remove_book_button = new Gtk.Button ();
        remove_book_button.set_image (BookwormApp.Bookworm.remove_book_image);
        remove_book_button.set_relief (ReliefStyle.NONE);
        //set the button as disabled - it will be enabled only if books are selected
        remove_book_button.set_sensitive (false);
        remove_book_button.set_tooltip_markup (BookwormApp.Constants.TOOLTIP_TEXT_FOR_REMOVE_BOOK_UNSELECTED);

        //Set up buttons for paginating the library
        Gtk.Box library_page_switcher_box = new Gtk.Box (Orientation.HORIZONTAL, 0);
        library_page_switcher_box.set_border_width (0);

        page_button_prev = new Gtk.Button ();
        page_button_prev.set_image (BookwormApp.Bookworm.back_page_image);
        page_button_prev.set_relief (ReliefStyle.NONE);
        page_button_prev.set_tooltip_markup (BookwormApp.Constants.TOOLTIP_TEXT_FOR_PREV_PAGE);
        library_page_switcher_box.pack_start (page_button_prev);
        page_button_prev.set_sensitive (false); //disable the prev button on first time load

        page_button_next = new Gtk.Button ();
        page_button_next.set_image (BookwormApp.Bookworm.forward_page_image);
        page_button_next.set_relief (ReliefStyle.NONE);
        page_button_next.set_tooltip_markup (BookwormApp.Constants.TOOLTIP_TEXT_FOR_NEXT_PAGE);
        library_page_switcher_box.pack_start (page_button_next);

        //Set up the progress bar for addition of books to library
        bookAdditionBar = new Gtk.ProgressBar ();
        bookAdditionBar.set_valign (Gtk.Align.CENTER);
        bookAdditionBar.set_show_text (true);

        //Create a footer and add widgets for select/add/remove books
        ActionBar add_remove_footer_box = new ActionBar ();
        add_remove_footer_box.pack_start (select_book_button);
        add_remove_footer_box.pack_start (add_book_button);
        add_remove_footer_box.pack_start (remove_book_button);
        add_remove_footer_box.pack_start (bookAdditionBar);
        add_remove_footer_box.pack_end (library_page_switcher_box);

        //Create a MessageBar to show status messages
        infobar = new Gtk.InfoBar ();
        infobarLabel = new Gtk.Label ("");
        Gtk.Container infobarContent = infobar.get_content_area ();
        infobar.set_message_type (MessageType.INFO);
        infobarContent.add (infobarLabel);
        infobar.set_show_close_button (true);
        infobar.response.connect (on_info_bar_closed);
        infobar.hide ();

        //Create the UI for library view and add all components to ui box for library view
        bookLibrary_ui_box = new Gtk.Box (Orientation.VERTICAL, 0);
        bookLibrary_ui_box.set_border_width (0);
        bookLibrary_ui_box.pack_start (library_view_box, true, true, 0);
        bookLibrary_ui_box.pack_start (add_remove_footer_box, false, true, 0);

        //create the webview to display page content
        webkitSettings = new WebKit.Settings ();
        webkitSettings.set_allow_file_access_from_file_urls (true);
        webkitSettings.set_allow_universal_access_from_file_urls (true); //this gives launchpad build error for Yaketty
        webkitSettings.set_auto_load_images (true);
        aWebView = new WebKit.WebView.with_settings (webkitSettings);
        aWebView.set_zoom_level (BookwormApp.Settings.get_instance ().zoom_level);
        aWebView.load_changed.connect((loadEvent) => {
            switch (loadEvent) {
                case WebKit.LoadEvent.STARTED:
                    break;
                case WebKit.LoadEvent.REDIRECTED:
                    break;
                case WebKit.LoadEvent.COMMITTED:
                    break;
                case WebKit.LoadEvent.FINISHED:
                    aWebView.run_javascript(BookwormApp.Bookworm.onLoadJavaScript.str, null);
                    break;
            }
        });
        webkitSettings.set_enable_javascript (true);
        //This is for setting the font to the system font - Is this required ?
        //webkitSettings.set_default_font_family (aWebView.get_style_context ().get_font (StateFlags.NORMAL).get_family ());
        webkitSettings.set_default_font_size (BookwormApp.Bookworm.settings.reading_font_size);
        webkitSettings.set_default_font_family (BookwormApp.Bookworm.settings.reading_font_name);
        gesture_swipe = new Gtk.GestureSwipe(aWebView);
        gesture_swipe.set_propagation_phase(Gtk.PropagationPhase.CAPTURE);

        //Set up Button for previous page
        back_button = new Gtk.Button ();
        back_button.set_image (BookwormApp.Bookworm.back_button_image);
        back_button.set_relief (ReliefStyle.NONE);

        //Set up Button for next page
        forward_button = new Gtk.Button ();
        forward_button.set_image (BookwormApp.Bookworm.forward_button_image);
        forward_button.set_relief (ReliefStyle.NONE);

        //Set up a slider for jumping pages
        pageAdjustment = new Adjustment (0, 1, 100, 1, 0, 0);
        pageSlider = new Gtk.Scale (Gtk.Orientation.HORIZONTAL, pageAdjustment);
        pageSlider.set_digits (0);
        pageSlider.set_valign (Gtk.Align.START);
        pageSlider.set_value_pos (Gtk.PositionType.RIGHT);
        pageSlider.set_hexpand (true);

        //Set up contents of the footer
        book_reading_footer_box = new ActionBar ();
        book_reading_footer_box.pack_start (back_button);
        book_reading_footer_box.pack_start (pageSlider);
        book_reading_footer_box.pack_end (forward_button);

        //Create the Gtk Box to hold components for reading a selected book
        bookReading_ui_box = new Gtk.Box (Orientation.VERTICAL, 0);
        bookReading_ui_box.set_border_width (0);
        bookReading_ui_box.pack_start (aWebView, true, true, 0);
        bookReading_ui_box.pack_start (book_reading_footer_box, false, true, 0);

        //Add all ui components to the main UI box
        Gtk.Box main_ui_box = new Gtk.Box (Orientation.VERTICAL, 0);
        main_ui_box.set_border_width (0);
        main_ui_box.pack_start (infobar, false, true, 0);
        main_ui_box.pack_start (bookLibrary_ui_box, true, true, 0);
        main_ui_box.pack_start (BookwormApp.Info.createBookInfo (), true, true, 0);
        main_ui_box.pack_end (bookReading_ui_box, true, true, 0);

        //Add action to open a book for clicking on row in library list view
        library_table_treeview.row_activated.connect ((path, column) => {
            Gtk.TreeIter iter;
            Value bookLocation;
            TreeModel aTreeModel = library_table_treeview.get_model ();
            aTreeModel.get_iter (out iter, path);
            aTreeModel.get_value (iter, 7, out bookLocation);
            BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap.get ((string) bookLoc
ation);
            if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_
UI_STATES[6] ||
                BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_
UI_STATES[7])
            {
                BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_U
I_STATES[7];
                BookwormApp.Library.updateListViewForSelection (aBook);
            }
            if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_
UI_STATES[5]) {
                BookwormApp.Bookworm.readSelectedBook (aBook);
            }
        });
        //Add action to update tree view when editing is Completed
        title_cell_txt.edited.connect ((path, new_text) => {
            updateLibraryListViewData (path, new_text, 1);
        });
        author_cell_txt.edited.connect ((path, new_text) => {
            updateLibraryListViewData (path, new_text, 2);
        });
        tags_cell_txt.edited.connect ((path, new_text) => {
            updateLibraryListViewData (path, new_text, 5);
        });
        //Add action to open the context menu on right click of tree view
        library_table_treeview.button_press_event.connect ((event) => {
            //capture which mouse button was clicked on the book in the library
            uint mouseButtonClicked;
            event.get_button (out mouseButtonClicked);
            //handle right button click for context menu
            if (event.get_event_type () == Gdk.EventType.BUTTON_PRESS && mouseButtonClicked ==
 3) {
                /*TreeIter iter;
                TreeModel model;
                Value bookLocation;
                TreeSelection selection = library_table_treeview.get_selection ();
                selection.get_selected (out model, out iter);
                model.get_value (iter, 0, out bookLocation);
                BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap.get ((string) boo
kLocation);
                TODO: Set up the right click context
                */
            };
            return false; //return false to propagate the action further i.e. row activation
        });
        // Add action to go to next or previous page in reponse to a finger
        // swipe gesture from right to left to or left to right respectively
        gesture_swipe.swipe.connect((x, y) => {
          // Avoid triggering nagivation actions on mostly vertical swipes that
          // should scroll up or down the page rather then flip it.
          // The x and y-values here are relatively arbitrary but seems to feel
          // right in testing.
          if (y.abs() > 800 || x.abs() < 800) {
            return;
          }

          // x == 0 on tap, so we ignore that
          if (x > 0) {
            handleBookNavigation("PREV");
          } else if (x < 0) {
            handleBookNavigation("NEXT");
          }
        });
        //Add action on the forward button for reading
        forward_button.clicked.connect (() => {
            handleBookNavigation ("NEXT");
        });
        //Add action on the backward button for reading
        back_button.clicked.connect (() => {
            handleBookNavigation ("PREV");
        });
        //Add action for moving the pages for the page slider
        pageSlider.change_value.connect ((scroll, new_value) => {
            debug ("Page Slider value change [" + new_value.to_string () +
                "] Initiated for book at location:" + BookwormApp.Bookworm.locationOfEBookCurrentlyRead);
            BookwormApp.Book currentBookForSlider = new BookwormApp.Book ();
            currentBookForSlider = BookwormApp.Bookworm.libraryViewMap.get (BookwormApp.Bookworm.locationOfEBookCurrentlyRead);
            if ((int.parse (new_value.to_string ())-1) > (currentBookForSlider.getBookContentList ().size)) {
                //this is for the scenario where the slider crosses the max value
                currentBookForSlider.setBookPageNumber (currentBookForSlider.getBookContentList ().size-1);
            } else {
                currentBookForSlider.setBookPageNumber (int.parse (new_value.to_string ())-1);
            }
            //update book details to libraryView Map
            currentBookForSlider = BookwormApp.contentHandler.renderPage (currentBookForSlider, "");
            BookwormApp.Bookworm.libraryViewMap.set (currentBookForSlider.getBookLocation (), currentBookForSlider);
            BookwormApp.Bookworm.locationOfEBookCurrentlyRead = currentBookForSlider.getBookLocation ();
            debug ("Page Slider value change action completed for book at location:" +
                BookwormApp.Bookworm.locationOfEBookCurrentlyRead +
                " and rendering completed for page number:" + currentBookForSlider.getBookPageNumber ().to_string ());
            return true;
        });
        //Add action for adding book (s) on the library view
        add_book_button.clicked.connect (() => {
            ArrayList<string> selectedEBooks = BookwormApp.Utils.selectFileChooser (
                Gtk.FileChooserAction.OPEN, _("Select eBook"), BookwormApp.Bookworm.window, true, "EBOOKS");
            BookwormApp.Bookworm.pathsOfBooksToBeAdded = new string[selectedEBooks.size];
            int countOfBooksToBeAdded = 0;
            foreach (string pathToSelectedBook in selectedEBooks) {
                BookwormApp.Bookworm.pathsOfBooksToBeAdded[countOfBooksToBeAdded] = pathToSelectedBook;
                countOfBooksToBeAdded++;
            }
            //Display the progress bar
            BookwormApp.AppWindow.bookAdditionBar.show ();
            BookwormApp.Bookworm.isBookBeingAddedToLibrary = true;
            BookwormApp.Library.addBooksToLibrary ();
        });
        //Add action for putting library in select view
        select_book_button.clicked.connect (() => {
            //initialize the counter to check how many books are selected
            noOfBooksSelected = 0;
            //check if the library is in List View mode
            if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[5] ||
                BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[6] ||
                BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[7])
            {
                //check if the mode is already in selection mode
                if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[6] ||
                    BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[7])
                {
                    //UI is already in selection/selected mode - second click puts the view in normal mode
                    BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_UI_STATES[5];
                    BookwormApp.Library.updateListViewForSelection (null);
                } else {
                    //UI is not in selection/selected mode - set the view mode to selection mode
                    BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_UI_STATES[6];
                    BookwormApp.Library.updateListViewForSelection (null);
                }
            } else {
                //check if the mode is already in selection mode
                if (BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[2] ||
                    BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE == BookwormApp.Constants.BOOKWORM_UI_STATES[3])
                {
                    //UI is already in selection/selected mode - second click puts the view in normal mode
                    BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = settings.library_view_mode;
                    BookwormApp.Library.updateGridViewForSelection (null);
                } else {
                    //UI is not in selection/selected mode - set the view mode to selection mode
                    BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_UI_STATES[2];
                    BookwormApp.Library.updateGridViewForSelection (null);
                }
            }
        });
        //Add action for removing a selected book on the library view
        remove_book_button.clicked.connect (() => {
            BookwormApp.Library.removeSelectedBooksFromLibrary ();
        });
        //handle mouse click on webview (reading mode)
        aWebView.button_press_event.connect ((event) => {
            if (!settings.is_leaf_over_page_by_edge_enabled) {
                return false;
            }
            int width;
            int height;
            //capture the current window size
            BookwormApp.Bookworm.window.get_size (out width, out height);
            //capture which mouse button was clicked on the book in the library
            uint mouseButtonClicked;
            event.get_button (out mouseButtonClicked);
            //handle left button click for page navigation if the click is near the left and right margins
            if (event.get_event_type () == Gdk.EventType.BUTTON_PRESS && mouseButtonClicked == 1) {
                //check if mouse is clicked near the right margin 10% of page width and go to previous page
                if(event.x < ((BookwormApp.Constants.PERCENTAGE_WIDTH_FOR_PAGE_NAVIGATION_ON_CLICK/100) * width)){
                    handleBookNavigation ("PREV");
                }
                if(event.x > (width - ((BookwormApp.Constants.PERCENTAGE_WIDTH_FOR_PAGE_NAVIGATION_ON_CLICK/100) * width))){
                    handleBookNavigation ("NEXT");
                }
            };
            return false; //return false to propagate the action further
        });
        //handle context menu on the webview reader
        aWebView.context_menu.connect ((context_menu, event, hit_test_result) => {
            context_menu.remove_all ();
            SimpleAction pageActionFullScreenEntry = new SimpleAction ("FULL_SCREEN_READING_VIEW", null);
            SimpleAction pageActionFullScreenExit = new SimpleAction ("FULL_SCREEN_READING_VIEW", null);
            SimpleAction pageActionWordMeaning = new SimpleAction ("WORD_MEANING", null);
            SimpleAction pageActionAnnotateSelection = new SimpleAction ("ANNOTATE_SELECTION", null);
            WebKit.ContextMenuItem pageContextMenuItemWordMeaning = new WebKit.ContextMenuItem.from_gaction (
                pageActionWordMeaning, BookwormApp.Constants.TEXT_FOR_PAGE_CONTEXTMENU_WORD_MEANING, null);
            WebKit.ContextMenuItem pageContextMenuItemFullScreenEntry = new WebKit.ContextMenuItem.from_gaction (
                pageActionFullScreenEntry, BookwormApp.Constants.TEXT_FOR_PAGE_CONTEXTMENU_FULL_SCREEN_ENTRY, null);
            WebKit.ContextMenuItem pageContextMenuItemFullScreenExit = new WebKit.ContextMenuItem.from_gaction (
                pageActionFullScreenExit, BookwormApp.Constants.TEXT_FOR_PAGE_CONTEXTMENU_FULL_SCREEN_EXIT, null);
            WebKit.ContextMenuItem pageContextMenuItemAnnotateSelection = new WebKit.ContextMenuItem.from_gaction (
                pageActionAnnotateSelection, BookwormApp.Constants.TEXT_FOR_PAGE_CONTEXTMENU_ANNOTATE_SELECTION, null);
            context_menu.append (pageContextMenuItemWordMeaning);
            context_menu.append (pageContextMenuItemAnnotateSelection);
            if (!settings.is_fullscreen) {
                context_menu.append (pageContextMenuItemFullScreenEntry);
            } else {
                context_menu.append (pageContextMenuItemFullScreenExit);
            }
            //Set Context menu items
            pageActionWordMeaning.activate.connect (() => {
                string selected_text = BookwormApp.Utils.setWebViewTitle ("document.title = getSelectionText ()");
                if (selected_text != null && selected_text.length > 0) {
                    //Save the page scroll position of the book being read
                    BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap
                        .get (BookwormApp.Bookworm.locationOfEBookCurrentlyRead);
                    aBook.setBookScrollPos (BookwormApp.contentHandler.getScrollPos ());

                    BookwormApp.Info.populateDictionaryResults (selected_text);
                }
            });
            pageActionAnnotateSelection.activate.connect (() => {
                string selected_text = BookwormApp.Utils.setWebViewTitle ("document.title = getSelectionText ()");
                if (selected_text != null && selected_text.length > 0) {
                    BookwormApp.AppDialog.createAnnotationDialog (selected_text);
                }
            });
            pageActionFullScreenEntry.activate.connect (() => {
                book_reading_footer_box.hide ();
                BookwormApp.Bookworm.window.fullscreen ();
            });
            pageActionFullScreenExit.activate.connect (() => {
                book_reading_footer_box.show ();
                BookwormApp.Bookworm.window.unfullscreen ();
            });
            return false;
        });
        //capture the url clicked on the webview and action the navigation type clicks
        aWebView.decide_policy.connect ((decision, type) => {
            if (type == WebKit.PolicyDecisionType.RESPONSE) {
```

```
                debug ("Signal captured for Policy type WebKit.PolicyDecisionType.RESPONSE");
                isWebViewRequestCompleted = true;
            }
            if (type == WebKit.PolicyDecisionType.NEW_WINDOW_ACTION) {
                debug ("Signal captured for Policy type WebKit.PolicyDecisionType.NEW_WINDOW_ACTION");
                isWebViewRequestCompleted = true;
            }
            if (type == WebKit.PolicyDecisionType.NAVIGATION_ACTION && isWebViewRequestCompleted) {
                debug ("Signal captured for Policy type WebKit.PolicyDecisionType.NAVIGATION_ACTION");
                //set the webview request flag to false to prevent re-trigger of this function
                //the webview request flag will be set to true when the response is received
                isWebViewRequestCompleted = false;
                WebKit.NavigationPolicyDecision aNavDecision = (WebKit.NavigationPolicyDecision)decision;
                WebKit.NavigationAction aNavAction = aNavDecision.get_navigation_action ();
                WebKit.URIRequest aURIReq = aNavAction.get_request ();
                string url_clicked_on_webview = BookwormApp.Utils.decodeHTMLChars (aURIReq.get_uri ().strip ());
                url_clicked_on_webview = Soup.URI.decode (url_clicked_on_webview);
                debug ("URL Captured:" + url_clicked_on_webview);
                //Handle external links (not file://) by opening the default browser i.e. http://, ftp://
                if (url_clicked_on_webview.index_of ("file://") == -1) {
                    BookwormApp.Utils.execute_sync_command ("xdg-open " + url_clicked_on_webview);
                    decision.ignore ();
                    return true;
                }
                //Handle Bookworm type links i.e. Annotation Overlay
                debug ("Window Title:" + BookwormApp.AppWindow.aWebView.get_title ());
                if (BookwormApp.AppWindow.aWebView.get_title () != null &&
                    BookwormApp.AppWindow.aWebView.get_title ().length > 1 &&
                    BookwormApp.AppWindow.aWebView.get_title ().index_of ("annotation:") != -1)
                {
                    //Open the annotation dialog
                    BookwormApp.AppDialog.createAnnotationDialog (
                        BookwormApp.AppWindow.aWebView.get_title ().replace ("annotation:", ""));
                    isWebViewRequestCompleted = true;
                }
                //Handle file:/// type links to other content of the book i.e. Table of Contents
                string anchor = "";
                if (url_clicked_on_webview.index_of ("#") != -1) {
                    string[] url_splitted_by_hashtag = url_clicked_on_webview.split("#", 2);
                    url_clicked_on_webview = url_splitted_by_hashtag[0];
                    anchor = url_splitted_by_hashtag[1];
                }
                url_clicked_on_webview = File.new_for_path (url_clicked_on_webview).get_basename ();
                int contentLocationPosition = 0;
                BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap
                    .get (BookwormApp.Bookworm.locationOfEBookCurrentlyRead);
                foreach (string aBookContent in aBook.getBookContentList ()) {
                    if (BookwormApp.Utils.decodeHTMLChars (aBookContent).index_of (url_clicked_on_webview) != -1) {
                        debug ("Matched Link Clicked to book content:" +
                            BookwormApp.Utils.decodeHTMLChars (aBookContent));
                        aBook.setBookPageNumber (contentLocationPosition);
                        //update book details to libraryView Map
                        BookwormApp.Bookworm.libraryViewMap.set (aBook.getBookLocation (), aBook);
                        if (anchor.len() > 0) { // anchor - id in link after # symbol
                            BookwormApp.Bookworm.isPageScrollRequired = true;
                            aBook.setAnchor(anchor);
                        }
                        aBook = BookwormApp.contentHandler.renderPage (aBook, "");
                        //Set the mode back to Reading mode
                        BookwormApp.Bookworm.BOOKWORM_CURRENT_STATE = BookwormApp.Constants.BOOKWORM_UI_STATES[1];
                        BookwormApp.Bookworm.toggleUIState ();
                        debug ("URL is initiated from Bookworm Contents, Book page number set at:" +
                            aBook.getBookPageNumber ().to_string ());
                        break;
                    }
                    contentLocationPosition++;
                }
            }
            isWebViewRequestCompleted = true;
            return true;
        });
        //Add action for paginating the library
        page_button_next.clicked.connect (() => {
            handleLibraryPageButtons ("NEXT_PAGE", true);
        });
        page_button_prev.clicked.connect (() => {
            handleLibraryPageButtons ("PREV_PAGE", true);
        });
        info ("[END] [FUNCTION:createBoookwormUI]");
        return main_ui_box;
    }

    public static void handleBookNavigation (string action){
        //action for NEXT page
        if(action == "NEXT") {
            //get object for this ebook and call the next page
            BookwormApp.Book currentBookForForward = new BookwormApp.Book ();
            currentBookForForward = BookwormApp.Bookworm.libraryViewMap.get (BookwormApp.Bookworm.locationOfEBookCurrentlyRead);
            debug ("Initiating read forward for eBook:" + currentBookForForward.getBookLocation ());
            currentBookForForward = BookwormApp.contentHandler.renderPage (currentBookForForward, "FORWARD");
            //update book details to libraryView Map
            BookwormApp.Bookworm.libraryViewMap.set (currentBookForForward.getBookLocation (), currentBookForForward);
            BookwormApp.Bookworm.locationOfEBookCurrentlyRead = currentBookForForward.getBookLocation ();
        }
        //action for PREV page
        if(action == "PREV") {
            //get object for this ebook and call the next page
            BookwormApp.Book currentBookForReverse = new BookwormApp.Book ();
            currentBookForReverse = BookwormApp.Bookworm.libraryViewMap.get (BookwormApp.Bookworm.locationOfEBookCurrentlyRead);
            debug ("Initiating read previous for eBook:" + currentBookForReverse.getBookLocation ());
            currentBookForReverse = BookwormApp.contentHandler.renderPage (currentBookForReverse, "BACKWARD");
            //update book details to libraryView Map
            BookwormApp.Bookworm.libraryViewMap.set (currentBookForReverse.getBookLocation (), currentBookForReverse);
            BookwormApp.Bookworm.locationOfEBookCurrentlyRead = currentBookForReverse.getBookLocation ();
        }
    }

    public static void handleLibraryPageButtons (string mode, bool isPaginateRequired) {
        if (mode == "NEXT_PAGE" && isPaginateRequired) {
            //activate the previous page button if it is disabled
            if (!page_button_prev.get_sensitive ()) {
                page_button_prev.set_sensitive (true);
            }
            //move the counter to the next position
            BookwormApp.Bookworm.current_page_counter = BookwormApp.Bookworm.current_page_counter + 1;
            BookwormApp.Library.paginateLibrary ("", "PAGINATED_SEARCH");
            //disable the forward button if the last modification date returned -1 for this page position
            if (BookwormApp.Bookworm.paginationlist.contains ("-1")) {
                page_button_next.set_sensitive (false);
            }
        }
        if (mode == "PREV_PAGE" && isPaginateRequired) {
            if (BookwormApp.Bookworm.current_page_counter > 0) {
                //activate the next page button if it is disabled
                if (!page_button_next.get_sensitive ()) {
                    page_button_next.set_sensitive (true);
                }
                //remove -1 from the paginated list if present of last modification dates to allow the forward button to work
                if (BookwormApp.Bookworm.paginationlist.contains ("-1")) {
                    BookwormApp.Bookworm.paginationlist.remove ("-1");
                }
                BookwormApp.Bookworm.current_page_counter = BookwormApp.Bookworm.current_page_counter - 1;
                BookwormApp.Library.paginateLibrary ("", "PAGINATED_SEARCH");
            } else {
                //disable the prev button as the counter is on the first page
                page_button_prev.set_sensitive (false);
            }
        }
        if (!isPaginateRequired) { //set the button status without doing pagination
            if (BookwormApp.Bookworm.current_page_counter < 1) {
                page_button_prev.set_sensitive (false);
            }
            if (BookwormApp.Bookworm.paginationlist.contains ("-1")) {
                page_button_next.set_sensitive (false);
            }
        }
    }

    public static bool updateLibraryListViewData (string path, string new_text, int column) {
        info ("[START] [FUNCTION:updateLibraryListViewData] updating metadata in List View on row:" +
            path + " for change:" + new_text + " on column:" + column.to_string ());
        //Determine the book whose meta data is being updated
        Gtk.TreeIter sortedIter;
        Value bookLocation;
        TreeModel aTreeModel = library_table_treeview.get_model ();
        Gtk.TreePath aTreePath = new Gtk.TreePath.from_string (path);
        aTreeModel.get_iter (out sortedIter, aTreePath);
        aTreeModel.get_value (sortedIter, 7, out bookLocation);
        //iterate over the list store
        Gtk.TreeIter iter;
        string bookLocationforCurrentRow;
        bool iterExists = true;
        iterExists = library_table_liststore.get_iter_first (out iter);
        while (iterExists) {
            library_table_liststore.get (iter, 7, out bookLocationforCurrentRow);
            if ((string)bookLocation == bookLocationforCurrentRow) {
                library_table_liststore.set (iter, column, new_text);
                BookwormApp.Book aBook = BookwormApp.Bookworm.libraryViewMap.get ((string) bookLocation);
                if (column == 1) {
                    aBook.setBookTitle (new_text);
                }
                if (column == 2) {
                    aBook.setBookAuthor (new_text);
                }
                if (column == 5) {
                    aBook.setBookTags (new_text);
                }
                aBook.setWasBookOpened (true);
                BookwormApp.Bookworm.libraryViewMap.set (aBook.getBookLocation (), aBook);
                debug ("Completed updating metadata in List View for book:" + (string) bookLocation);
                return true; //break out of the iterations
            }
            iterExists = library_table_liststore.iter_next (ref iter);
        }
        info ("[END] [FUNCTION:updateLibraryListViewData]");
        return true;
    }

    public static Granite.Widgets.Welcome createWelcomeScreen () {
        info ("[START] [FUNCTION:createWelcomeScreen]");
        //Create a welcome screen for view of library with no books
        BookwormApp.Bookworm.welcomeWidget = new Granite.Widgets.Welcome (
            BookwormApp.Constants.TEXT_FOR_WELCOME_MESSAGE_TITLE,
            BookwormApp.Constants.TEXT_FOR_WELCOME_MESSAGE_SUBTITLE);
        Gtk.Image? openFolderImage = new Gtk.Image.from_icon_name ("document-open", Gtk.IconSize.DIALOG);
        BookwormApp.Bookworm.welcomeWidget.append_with_image (
            openFolderImage, "Open", BookwormApp.Constants.TEXT_FOR_WELCOME_OPENDIR_MESSAGE);
        //Add action for adding a book on the library view
        BookwormApp.Bookworm.welcomeWidget.activated.connect (() => {
            ArrayList<string> selectedEBooks = BookwormApp.Utils.selectFileChooser (
                Gtk.FileChooserAction.OPEN, _("Select eBook"), BookwormApp.Bookworm.window, true, "EBOOKS");
            //If ebooks were selected, remove the welcome widget from main window and show the library view
            if (selectedEBooks.size > 0) {
                BookwormApp.Bookworm.window.remove (BookwormApp.Bookworm.welcomeWidget);
                BookwormApp.Bookworm.window.add (BookwormApp.Bookworm.bookWormUIBox);
                BookwormApp.Bookworm.bookWormUIBox.show_all ();
                BookwormApp.Bookworm.toggleUIState ();
                BookwormApp.Bookworm.pathsOfBooksToBeAdded = new string[selectedEBooks.size];
                int countOfBooksToBeAdded = 0;
                foreach (string pathToSelectedBook in selectedEBooks) {
                    BookwormApp.Bookworm.pathsOfBooksToBeAdded[countOfBooksToBeAdded] = pathToSelectedBook;
                    countOfBooksToBeAdded++;
                }
                //Display the progress bar
                BookwormApp.AppWindow.bookAdditionBar.show ();
                BookwormApp.Bookworm.isBookBeingAddedToLibrary = true;
                BookwormApp.Library.addBooksToLibrary ();
            }
        });
        info ("[END] [FUNCTION:createWelcomeScreen] ");
        return BookwormApp.Bookworm.welcomeWidget;
    }

    public static void showInfoBar (BookwormApp.Book aBook, MessageType aMessageType) {
        debug ("[START] [FUNCTION:showInfoBar] ");
        StringBuilder message = new StringBuilder ("");
        message.append (aBook.getParsingIssue ()).append (aBook.getBookLocation ());
        BookwormApp.AppWindow.infobarLabel.set_text (message.str);
        BookwormApp.AppWindow.infobar.set_message_type (aMessageType);
        BookwormApp.AppWindow.infobar.show ();
        debug ("[END] [FUNCTION:showInfoBar] with message:" + message.str);
    }

    //Handle action for close of the InfoBar
    public static void on_info_bar_closed () {
        BookwormApp.AppWindow.infobar.hide ();
    }

    public static bool handleWindowStateEvents (Gdk.EventWindowState ev) {
        if (ev.type == Gdk.EventType.WINDOW_STATE) {
            if ((ev.window.get_state () & Gdk.WindowState.FULLSCREEN) == 0) {
                settings.is_fullscreen = false;
            } else {
                settings.is_fullscreen = true;
            }
        }
        return false;
    }

    public static void controlDeletionButton (bool selectionState) {
        if (selectionState) {
            //Enable the Deletion Button as a book is selected for potential removal
            noOfBooksSelected++;
            remove_book_button.set_sensitive (true);
            remove_book_button.
                set_tooltip_markup (BookwormApp.Constants.TOOLTIP_TEXT_FOR_REMOVE_BOOK);
        } else {
            //Check and Disable the Deletion Button if no books are selected after this de-selection
            noOfBooksSelected--;
            if (noOfBooksSelected < 1) {
                remove_book_button.set_sensitive (false);
                remove_book_button.set_tooltip_markup (
                    BookwormApp.Constants.TOOLTIP_TEXT_FOR_REMOVE_BOOK_UNSELECTED);
                }
            }
        }


}
```