# Photobook Simplified Code Documentation

CONFIDENTIALITY NOTE:

This document is confidential and may be protected by legal privilege. You are hereby notified that any disclosure, distribution, or copying of this message, or any attachment, is prohibited. You agree to keep the contents of this document confidential and restrict distribution to those who have been given permission to see it, by David Gross, its author.

This document lists the important modules used by the "photobook" or "Mimetic Books" XML book document player and library app. It also lists some important functions in the main book display module, "slideview."

The goal of this document is to guide a programmer, to help him find the most important code, and to see how coding, caching, and modularity are accomplished. It is, of course, incomplete, but most functions do what their names say ("openChapter"), and are short and easily understood.

I have primarily included code I have written, where the approach taken (say, to caching) is significant. Of course, the app uses modules I haven't written and have no particular interest, such as XML parsing or FTP downloading.

## *Notes:*

*A "display group" is a Corona SDK display object, like a group, a picture, text, etc.*

## Important proprietary Modules

### AUDIOFUNX.LUA

*Used by: slideview.lua*

General functions for playing, pausing and loading audio.

### book.lua

*Used by: main.lua*

Reads an XML file and converts it into a book Lua table. The book table is easier to work with than an XML file, and includes a few statistics (total page count, etc.)

### BuildImageSizeFile.lua

*Used by: main.lua*

Builds a database of image sizes to speed loading of images.

### CheckFiles.lua

*Used By: main.lua*

Checks that all files found in the settings actually exist.

### download.lua

*Used By: shelves.lua*

A library of functions for downloading and unpacking tar files.

### dragger.lua

This module lets you make objects draggable, and reports their locations on screen. Useful for position things on screen during development.

### entities.lua

*Used by: textwrap.lua*

Convert numeric html entities to utf8.

### fontmetrics.lua

*Used by: textwrap.lua*

Return font metrics that tell how to position a font correctly. Reads the fontmetrics.txt file, which is built by a PHP/ImageMagick website.

### funx.lua

*Used by: all modules*

Miscellaneous functions. All functions here can stand alone, and do not depend on outside structures.

In particular, the autoWrappedText function is here, which creates a display group of wrapped, aligned, somewhat formatted text. As iOS and Corona improve, this function will have to change.

### handler.lua

*Used by: book.lua*

Part of the XML reading system.

### html.lua

Used by: textwrap.lua

Parses HTML text into a table structure.

### iosiap.lua

*Used by: main.lua*

Functions for handling in-app purchases in iOS.

### main.lua

*Used by: NA*

Loads and parses:

* settings files
* a book file
* templates
* the main touch handler for the book

Runs a slideview instance to show and play the book.

### memorymatch.lua

*Used by: slideview.lua*

A sample plugin that can play on a page, based on a Corona Labs sample game.

### motion.lua

*Used by: slideview.lua*

Motion functions for making sprites move naturually on simple paths, to allow simply animations on a page.

### navbar.lua

Create a navigation bar, used to navigate a chapter of a book.

(Depends on getting a book table, so hard to use independently of my other modules.)

local navbarLib = require("navbar")

local labelType = settings.navbar.labelType or "label"

local navbarType = settings.navbar.navbarType or "slider"

local params = {

book = book,

chapterNum = newChapterNum,

handlers = navbarHandlers,

settings = settings,

forceShow = forceShow,

navbarType = navbarType,

labelType = labelType,

}

chapter.navbar = navbarLib.new(params)

### onSwipe.lua

*Used by: main.lua, slideview.lua*

This module handles touch/swipe actions and calls specified handlers. OnSwipe is used by main.lua to add touch/swipe to the book viewer, a slideview instance. OnSwipe is also used by slideview to add touch/swipe to pictures on a page.

local actions = {

init = g.initSwipe,

swipeLeft = g.goToNextPage,

swipeRight = g.goToPrevPage,

swipeUp = g.swipeUp,

swipeDown = g.swipeDown,

cancelSwipe = g.cancelMove,

swiping = g.followSwipe,

endSwipe = g.endSwipe,

tap = zoom.toggleZoom,

}

local onSwipe = require("onSwipe")

local swipeListener = onSwipe.new(actions)

pGroup.touch = swipeListener

pGroup:addEventListener( "touch", pGroup )

### panorama.lua

*Used by: slideview.lua*

This library is for showing panoramic images larger than the screen. It can use the gyroscope for navigation.

### settings.lua

*Used by: main.lua*

This module reads a system settings file (e.g. settings.xml). In main we read both a system file, and the book file, then merge the tables together.

### shelves.lua

*Used by: main.lua*

Shelves manages the in-app library of books (but not the UI for that library).

### shelvesdisplay.lua

*Used by: main.lua, shelves.lua*

ShelvesDisplay is the UI for the shelves library. It is the UI which lists library pages, lets the reader choose a book to read, delete a book, etc.

### slider.lua

*Used by: navbar.lua*

Used by the navbar module, this module creates a slider for navigating the pages. The slider can be formatted in various ways, and there are options, like grouping by date, or showing page labels in the slider.

### slideview.lua

*Used by: main.lua*

This is the main viewer module for Photobook.

### textinput.lua

*Used by: slideview.lua, shelvesdisplay.lua*

Simple functions for user input of text.

### textwrap.lua

*Used by: many modules*

A text formatting engine that reads simplified HTML text, and/or uses a simple paragraph format encoding. Most importantly, it wraps text.

### zoom.lua

*Used by: slideview.lua*

Enlarge or reduce a picture. This is the single-finger gesture version. You MUST pass the mainTouchTarget if you want gestures not used, e.g. swipe, to be passed on. If you don't, then such gestures on the obj will be ignored.

Actually, this module is *transitioning* back and forth between two states of an object. It only uses scale, rotation, and alpha right now, but it could easily use other attributes of objects, such as color. However, there aren't many other attributes of Corona display objects, are there?

# slideview.lua

The module "slideview.lua " is the main module for drawing and using an XML document. I have documented some of the important functions

## Important Functions

##### monitorMem()

Monitors memory usage and prints current memory usage. Important for finding memory leaks.

##### slideview:new(book, chapterNum, startpage, templates, settings)

Returns a book viewer object. In theory, you could invoke this more than once to have multiple viewers on the screen, even a book inside a book. However, I don't think I've got it working in a 100% modular fashion, so in practice, I don't think it would work right now.

###### Parameters:

book : a book table, created from an XML file, by the book.lua module

chapterNum : The book chapter to open with, normally 1. (default = 1)

startpage : The starting page of the chapter, normally 1. (default = 1)

templates : The templates table, created from a templates XML file by the templates.lua module.

settings : The settings table, create from settings file using the settings.lua module.

##### checkScale(displayObject)

If a Corona display object, e.g. an image, is larger than the screen, then resize it to fit the screen. Useful for code to work on different size screens, e.g. iPhone, Android.

##### resizeFromIpad(displayObject)

If a the current screen is NOT an iPad screen, then resize the Corona display object to fit the current screen. This assumes the object was designed for the iPad.

##### buildFacebookMessage()

Build a message which can be posted on Facebook, including name, caption, description, footer, and a picture link.

##### buildTwitterMessage()

Build a message to be posted on Twitter.

##### buildCaptionObject (page, pageNumber, addShadow)

Build a display group which is a caption bar, including text and buttons.

##### buildTextObject (textblockObject, pageValues, params)

Returns a display group of a block of wrapped, formatted block of text. This version automatically handles a multi-line textblock. Default is no shadow

pageNumber is pageNumber in the chapter, not in the book. That is page.number

###### Parameters:

textblockObject : A text object from the book table. Typically, the could be something like book.chapter[x].page[x].textblock. Here is an example in the XML book file of a textblock object:

<textblock [attributes] >

<title>mytitle</title>

<text>my text</text>

</textblock>

pageValues : This is a page table, taken from a book table. For example, you could pass book.chapter[x].page[x] as the pageValues parameter.

params : The params are based on the settings files. The variable names come from the names of settings, such as "overlayTextColor". Normally, my variables don't begin with uppercase letters, but in this function they might. Why, you ask? Because the params table can be built from sets of values in the settings files, such as all entries beginning with the text, "overlay". This allows us to create sets of parameters in the settings file. Of course, if the settings reader could handle depth in the XML file, we could simply have the sets that way...but it can't so for now, we are doing it this way. *For example, the setting "overlayTextColor" can be read and added to an "overlayParams" table as "TextColor", then easily passed to the function.*

##### buildLinksObject(page)

Returns a display group of tappable text links, to jump to a chapter and page in the book. Links are text only. Buttons are different: they don't jump to pages in the book, and a button can be a picture or text.

##### buildPicturesObject(pictures, params, margins, absolute)

Returns a display group of pictures. The pictures can "zoom" when tapped, meaning they can change in scale, location, and transparency.

###### Parameters:

pictures : A pictures object from a book table. The object contains <pictures> elements. Here is an example in the XML of a pictures object:

<pictures x="right" y="110">

<picture zoomedScale="1" filename="\_user/page/image/javier\_arcenillas\_portrait.jpg" />

</pictures>

params : (ONLY USED FOR DEBUGGING FOR NOW)  
Typically, this would be values from the page which determine how the pictures block is formatted. Only the page number is used right now (params["number"]). However, you can pass the entire book page (book.chapter[x].page[x]) knowing the most of the info will be ignored but that the main attributes (such as the page number) are there. AND, this is only used to warn the developer of a missing image!

margins : A table of margins for positioning the pictures. Typically, these are the page margins so it is easy to set margins for pages. The default is 0 all around. The margins are a table:

margins = { top=20, bottom=40, left=40, right=30}

absolute : If true, then the margins are set to 0 all around.

##### buildButtonsObject(page)

Returns a display group of buttons. A button can be a picture or text. Buttons can do many things, such as play sound or video, show a map, etc.

###### Parameters:

page : A book page table, e.g. book.chapter[x].page[x]. The buttons are created from the <buttons> object. Here is an example of a buttons object:

<buttons x="0" y="512" absolute="true">

<button x="20" y="0" id="goToURL" text="http://books.mimetic.com/" params="http://books.mimetic.com/" action="goToURL" />

<button x="10" y="57" id="goToURL" params="http://www.fotoevidence.com/" action="goToURL" imagefile="\_user/button/button-fotoevidence.png" over="\_user/button/button-fotoevidence-over.png" />

<button x="18" y="+100" id="emailAuthor" params="FotoEvidence@FotoEvidence.com" action="sendEmail" imagefile="\_ui/button-mail.png" over="\_ui/button-mail-over.png" />

<button x="68" y="168" id="sendEmail" size="14" text="FotoEvidence@FotoEvidence.com" params="FotoEvidence@FotoEvidence.com" action="sendEmail" />

<button x="68" y="+18" id="goToURL" size="14" text="http://www.FotoEvidence.com/" params="http://www.FotoEvidence.com/" action="goToURL" />

</buttons>

##### addTableOfContents(book)

Automatically build a table of contents for the book. Build the contents for the book based on the chapters. The chapter title is set in the chapter's <title> element. The chapter <index> element preceeds the title, and typically the chapter number. The <index> let's you label a chapter as, for example, "A" instead of "1" in the contents.

Example:

<chapter>

<title>My Chapter Title</title>

<index>1.</index>

<page contents="true" ...>

<tableofcontents x="center" align="center" />

</page>

...

</chapter>

##### loadPage (pageNum, fadeSetting, transitionSpeed)

Render a page and surrounding pages. This is a short function that calls the main render function, loadOnePage.

##### loadOnePage (pageNum, offset, fadeInSetting, transitionSpeed )

This is the main function for rendering a page. It creates the "screen" display tables, which are part of the added to the book table, as an element of a page. The table structure that is created is:

book.chapter.page.screen

This allows us to dump the .screen part of a page to clear memory in one step.

This function currently builds layers of objects, grouped by type. In the future, we won't group by type — that is a legacy of having only one of each type of layer, e.g. only one type block, one picture, etc.

##### leavePage(pageNum)

A handler for leaving a page. Turns off audio if required, stops video, etc.

##### clearPageCache(pageNum, cnum)

Clear a page display object from memory.

##### applyTemplateToPage(templateID, sourcePageValues)

Apply a template from the templates.txt file to a page from the book.xml file. It's a bit tricky — it has to fill in values it knows from the book.xml yet not create empty objects, such as picture entries, if the book has no info for the template.

##### goToNextPage, goToPrevPage (nextPageOffset)

This function moves the view to the next page, where the page is *nextPageOffset* pages away, default is 1. It does this by moving the display object that holds the current chapter, sliding it left or right. The code could as easily move pages up and down with little modification.

##### unloadDistantPages (pageNum, oldPageNum)

When the reader jumps to a page, the cached pages around where he was reading before must be unloaded to save memory.

##### loadChapter (book, newChapterNum, startpage)

Take a chapter from the book table, which is purely XML, and add screen and display elements to the chapter. This prebuilds the basic display objects for a chapter and adds touch handlers, navbar, and toolbar.