Corona Comics SDK 2.0

David Fox
Electric Eggplant and Ansca Mobile.
November 11, 2011

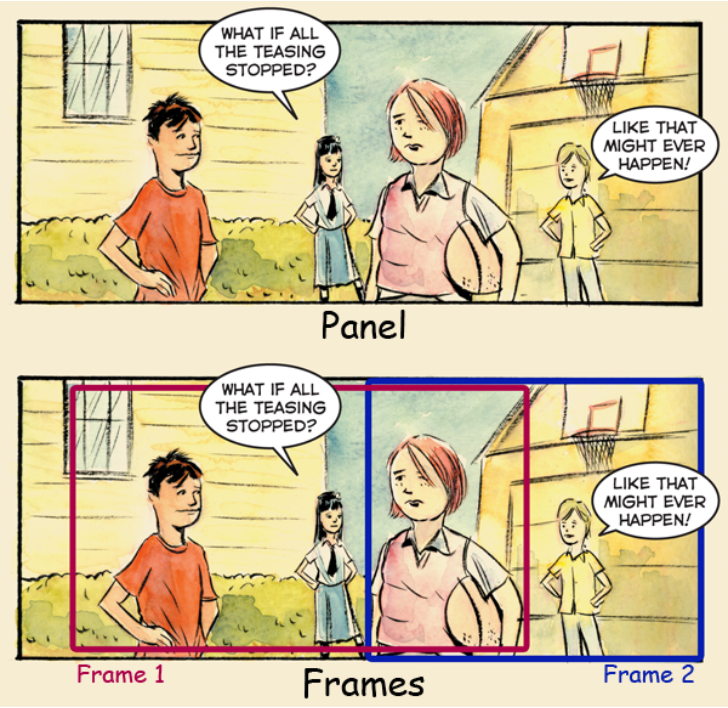
The Corona Comics SDK 2.0 is based on the 1.0 version from November, 2010. In truth, this code is not as much an SDK as a demo of a fully functioning app using the code. It is a Lite version of Electric Eggplant's iPad app, "Be Confident in Who You Are: A Middle School Confidential™ Graphic Novel" with all the functionality intact. It contains the opening animation and all of chapter 1, plus the Info page, and several About pages.

Note that since it wasn't intended as an SDK, there's some hard-coded special functionality still in place that will most likely need to be commented out or removed if you use this code.

Terminology

In this document we're using two different terms for what you see on a graphic novel/comic page:

- **Panel** — each individual image, usually contained within a rectangle, which you see on a page.
- **Frame** — the camera framing within a panel. In many cases, the frame encompasses the entire panel, so they're the same. Other times, though, we do several camera moves within a panel. Each of these moves is a *frame*.



We're always dealing with frames in the code rather than panels. There are at least as many frames on a page as there are panels, but often many more. This is especially the case when there are multiple word balloons in a panel. We found the story moves better when panels with many balloons are broken into multiple frames.

Features

Here's a partial feature list of this code:

- **Zoom in/out via double-tap** – double-tap on a frame and you'll zoom into it. Double-tap again, and you zoom out. You can continue reading the graphic novel in either mode, but there are a lot more sound effects in zoomed-in mode. It also makes the experience much more like watching an animatic.
- **Hires images** – the artwork we're using is 2x the size of an iPad screen, or 1536x2048. This means that you can zoom into an image and it won't get blurry.
- **Separate word balloons layer** – the word balloons for each page are stored in a sprite sheet. Because they're on a separate layer, they can overlap adjacent panels. For panels with multiple word balloons, we often do additional "camera moves" inside that panel, showing/hiding word balloons as the story progresses. There's also an option to turn off this functionality if your balloons are "baked in" to the background art by setting `_G.hasBalloons = false` in main.lua.
- **Auto bookmark** – remember which page (or frame) you're on when you exit out of the app so you can continue where you left off.
- **Table of contents** – there's an Info icon in the lower right corner. Tapping on it takes you to a screen with buttons for the About pages, any of the 8 chapters, or to restart.
- **Sound** – there's a table of sound effects describing which sounds to trigger for the entire page, and which to trigger for a specific frame. There's support for 32 simultaneous sounds (though we never use more than a half dozen of them at the same time). Also support for fade in/out, delay start of sound, loop sound x times, transition to a new volume level, etc.
- **Orientation** – works fine in any orientation. Advance with a swipe or a tap in the margin.
- **Animated cover page** – of course, our intro wouldn't work for any other book, but there's a place to drop in your animation if you like. This can be omitted by setting the `_G.coverAnimation = false` in main.lua.
- **Tutorial** – on the first page, we tell you how to zoom in/out and use the Info button.
- **Memory management** – catches iOS out of memory warnings and clears all page. Also clears pages as you move through.

- **Links** – you can include web links, links to the iTunes store, or email links. In the case of web links, a web popup opens on the page or you can have the URL open in Safari instead. The other two take you to the relevant app.
- **Ratings prompt** – after you finish the graphic novel, we prompt you to rate it with an alert. The options include "review now", "later" and "never".
- **Message from a website** – when you go into the Info screen, it checks a file on our web server (you can use your own server, of course) and displays it in a panel that slides up. If the message hasn't changed, it doesn't show it again, only new messages.
- **Android support** – we do have this on the NOOK Color, so at least one Android platform works. Not tested on others yet.
- **iPhone support** – not fully tested, but it should be mostly ready. Our app has some About screens with small text that wouldn't be legible on an iPhone, so we haven't actually released a universal version yet. There is already code in place to handle both 2x and 1x sized images, but we're not using the standard Corona code for this.

Files Overview

The files included in the demo app are:

- main.lua — includes initialization code, info page
- balloons.lua — coordinate and size information for all the word balloons
- cover.lua — animation of the book cover. This can be omitted by setting the `_G.coverAnimation = false` in main.lua.
- coordinates.lua — where should the "camera" move for each frame
- device.lua — just lists which OS or device. This should really be retrieved by checking the device hardware, but we found it easier to manage multiple versions this way.
- frames.lua — has the coordinate data for the double-tap (to zoom) hotspot rectangles for each panel. Also includes hotspots for web URLs, iTunes URLs and email links.
- PageN.lua, PageN@2x.lua — holds the sprite sheet data for page N. Needed for any page with 2 or more frames and you're using the word balloon layers (which are stored as sprites). These sheets are generated by the Zwoptex program (http://zwoptexapp.com/). The @2x ones are the hires versions of these sheets.
- pageSize.lua — lists which pages are available at 2x size for iPad and iPhone 4. This is for the background images only. You'll still need 2x sprites.
- promote.lua — Promotion Library for Corona SDK, written by Reflare. We use this to display a "news" update when you visit the Info page.
- Reader.lua — the framework code, or the guts of the comic reader, loads pages, handles all camera moves and page changes, has sound routines, and more
- sound.lua — loads the sound files, sets up the tables for controlling all the sound effects

- ui.lua — Corona UI code, with a few mods for hires text buttons

images/

This folder has the following subfolders:

- cover — art assets for the cover animation
- pages — has both the lores (768x1024) images for each page, and in some case also hires images (1536x2048, with @2x in the file name) if we'll be zooming in on that page. Page0.jpg is a static version of the cover page. Page1.jpg, the first page of the actual book, is the only one that's in horizontal orientation (in this case, 1024x707). The end-of-chapter summary pages (e.g., Page9.jpg) are all smaller in size and have their word balloons embedded into the art (since there's no zooming on these pages). For the iPhone 3 version, we'd use the lores art for all pages.
- promote — assets for the promote.lua code
- sprites — sprite sheets containing the word balloons, created with the Zwoptex program.
- ui — assets for the user interface, including the info button, the navigation buttons on the Info page, etc.

sound/

Holds the sound files. We used stereo files for the ambient looping sounds and most of the music, and mono for the sound effects. The m4a sounds are for iOS devices. We use mp3 for Android. You're welcome to use the UI sound effects in your own code:

```
ui_beep_1.m4a
ui_click_1.m4a
ui_click_2.m4a
ui_page_turn_1.m4a
zoom1_in.m4a
zoom1_out.m4a
```

The rest of the sounds and music are proprietary.

Fonts

One font is also included, SF_Slapstick_Comic.ttf. The license information for this font is in "ShyFonts Font License - SF_Slapstick_Comic.txt" — you're free to use it in your app, just follow the instructions. (This isn't the font we're using in our published app.)

Page Organization

The demo has a total of 18 pages, numbered 0-17:

- Page 0 – static version of the cover. When the cover animation completes, this page loads and the cover animation files are removed from memory.
- Pages 1-9 – actual story pages
- Page 10 – ad to "buy this app". This code was intended as a lite version of the app but was rejected by Apple because it was too close to the full version.

- Pages 11-16 – About pages, featuring the "cast of characters", credits, and a page for the other books in the series.
- Page 17 – Info page. The code assumes the info page is always the last page defined in the coordinates.lua table.

Take a look at lines 180-188 in main.lua and you'll see variables for the above page types and ranges for this "lite" version. For the full version, look at lines 169-174. Of course, you'll adjust these values to match your own book.

Next look at line 867 in startBook(), also in main.lua:

```
reader:initialize( imgPageDir .. "Page", data, firstPage, { loadLastPage =
bookmarking, minPage = 0, maxPage = _G.lastPage, frameNum = firstFrame,
fadeIn = bookmark, doubleTapped = doubleTapped, infoTapped = infoTapped  } )
```

This sets up the reader code and displays the first page of the book (`firstPage`) We're also setting `minPage=0` and `maxPage=_G.lastPage` (which is we've set to 16). This means you can't move to a page earlier than `minPage` or a page beyond `maxPage`. This means you can run through all the pages except the Info page just by moving forward through the book. We could have set it up so the only way to get to the About pages is through the info page by setting `maxPage= buyAppPage`.

We use a similar page range-setting code on the Info page to bring up the About pages. For example, on line 317 in main.lua, when the Credits button is pressed:

```
reader:gotoPage(aboutCreditsPage, 0, { minPage = firstAboutPage, maxPage =
_G.lastPage, aboutFlag = true })
```

This is goes to the Credits page (the following '0' is for frame 0), and sets the range of pages to be constrained to all the About pages. (We're also setting `aboutFlag=true`, but this parameter isn't currently being used for anything in the code, so you could disregard.)

In fact, going to the Info page uses the same routine on line 438 of main.lua:

```
local page = reader:gotoPage(_G.infoPage, 0, { minPage = _G.infoPage,
maxPage = _G.infoPage })
```

Here the range is constrained to just the Info page, so you can't swipe left or right off of this page.

You could have several ranges of pages in a book, each isolated from all other pages. One use might be for special activities (e.g, quizzes) that you might get to via additional buttons.

Assembling a Comics Page in Photoshop

We use Photoshop to assemble our pages, including the word balloons, frames and touchable areas. We then use built-in Photoshop functions plus a script to export the assets into separate files (see below).

Open the Photoshop psd file we've included with this project for page 7, called "Be Confident Demo - page 7.psd". Notice that this 1536x2048 file has 3 groups (folders): Frames 7, Coordinates 7, and Page 7.

Let's look inside the Page 7 group first. It has 8 layers, numbered 1-8. Each one contains at least one word balloon. It also contains another group, Background, which contains the background art plus any text that we want "burned into" the background. In other words, everything in the Background folder gets saved as the final background image for this page.

Now look at the Coordinates 7 group. This holds the frame definition information for this page's camera moves. We're following the example in the Corona Comics demo code and using a semi-transparent rectangle to define each frame. The names of the layers in this group are ignored. What's important is their order.

Even though you can define a very small frame, we won't scale up the image past a certain point, now set as 2.5. You can adjust this in line 1234 of Reader.lua, but increasing this number will result in blurrier images:

```
local maxScale = 2.5    -- we don't want to zoom in more than 2.5x
                        -- or the 2x images will degrade too much
```

The Frames 7 group is very similar to the Coordinates group (and on some pages it's identical). It's used to indicate the "hot spot" where you can double-tap to enable a zoom-in to that frame. When there's a difference between the Frames and Coordinates groups' layers, it's usually because we have multiple moves within a panel, and we want to make sure that when you double-tap, what you expect to happen does happen. For example, notice that there 2 camera moves in the first panel of page 7. The first camera position is centered on the 3 kids on the left. The second includes just the 2 kids on the right. Here we decided that anywhere you tap on this panel should zoom you into the first camera position to avoid confusion.

Also, because placing a smaller hot spot below a larger one would make the lower one inaccessible, sometimes we have to flip the order of the layers in the Frames group to get it all to work properly.

To handle both of these instances, we can add an extra value in a row of data in the frames.lua file that indicates exactly which coordinates group (frame/camera position) to zoom into:

```
[7] = {
    { 49,87,674,280,1},     -- zoom into frame 1
    { 46,384,218,285,3},    -- zoom into frame 3
    { 269,374,233,305,4},   -- zoom into frame 4
    { 514,384,211,281,5},   -- etc.
    { 45,685,225,302,6},
    { 287,685,209,305,7},
    { 509,684,215,306,8},
},
```

Exporting Image Assets

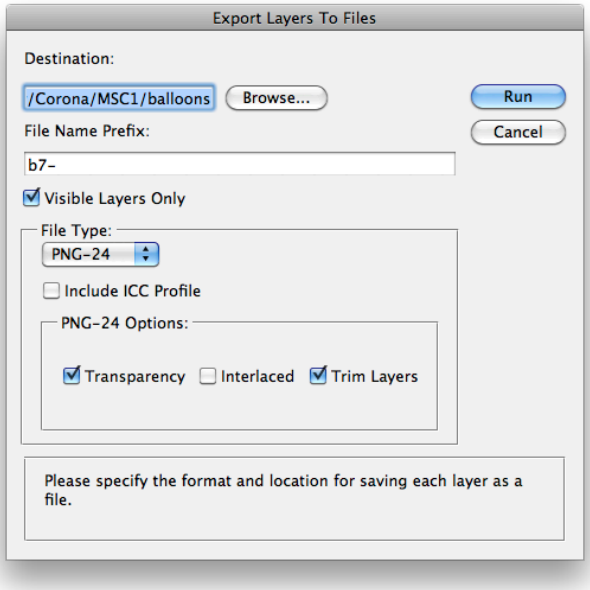
All of the image assets can be saved at full 1536x2048 resolution as well as ½ resolution, 768x1024. We're using full resolution background images as well as word balloons for iPad and iPhone 4/4S. For iPhone 3 we're using ½ resolution backgrounds and word balloons. For NOOK Color we're using full ½ resolution backgrounds and full resolution word balloons.

There's an exception for word balloons when a single balloon is greater than 1024 pixels wide and we know that we won't be zooming in on that panel. In that case, we're saving that balloon at ½ resolution, and later we'll mark this balloon in the table indicating it's already been scaled to ½ resolution. See balloons.lua, and look at the third row in the table and you'll see the "true" as the final value on that row:

```
{2,55,412,598,163, true},  -- This balloon does not need to be displayed at 2x,
                           -- it already fills the page.
```

Word Balloons

Balloons are saved as transparent PNGs using Photoshop's File:Scripts:Export Layers to Files… available in Photoshop CS4 and later (see inset). Make sure you hide all layers besides the balloons and that Trim Layers is selected. (Trim will not work if you're running this on the bottom layer (background).) We then removed the extra numbering info for each file so they were named b7-1.png, b7-2.png, etc. (where 7 is the page number for these balloons).



Then we use Zwoptex to merge all the balloons for a page into a single sprite sheet (and if we also need them for a lores device, also save them at ½ resolution in a second sprite sheet). Zwoptex exports both a lua data file for each sheet, plus the sprite sheet image file. In the demo, these lua files are saved in the root directory (same place as the main.lua file) and named PageN@2x.lua (where N is the page number) for the hires version. If you also want lores versions for an iPhone 3 or other low-memory devices, just save them as PageN.lua.

Background Image

We save the background images as 50% compression jpegs when they're at full resolution, and 70% jpegs when they're saved as 50% size (768x1024 or smaller). Again, if we want to zoom in on a page, we want to store the page at a higher resolution so it remains sharp when zoom in.

Coordinates Data

To extract the coordinate information from the layers in all of these folders, we resize the entire Photoshop file to ½ size (Image:Image Size), 768x1024, remove the bottom background color layer and save it as a new file. Then use the CoronaComicsCoordCreator Applescript droplet by Andy Hullinger. It looks at a Photoshop file, and saves the coordinates for each layer into the clipboard (this works fine in Photoshop CS4, but may only work one time in CS5 before you have to restart Photoshop). This script only looks at the layers one level deep within the groups, so it will ignore the background image since it's nested 2 levels in. You can either open the droplet and it will ask you for a Photoshop psd file, or drop the psd file on the droplet. Wait a bit, and it will finish with an alert telling you to paste the clipboard contents into a file. Here's what the droplet will pull from the sample Photoshop file (after you've reduced the image size to 768x1024 and removed the bottom layer):

```
local data =
{

    [7] = {
        { 49,87,674,280},
        { 46,384,218,285},
        { 269,374,233,305},
        { 514,384,211,281},
        { 45,685,225,302},
        { 287,685,209,305},
        { 509,684,215,306},
    },
    [7] = {
        { 101,94,454,268},
        { 396,94,325,269},
        { 50,391,212,274},
        { 184,374,318,305},
        { 516,391,204,273},
        { 50,693,214,292},
        { 291,689,199,297},
        { 514,690,204,297},
    },
    [7] = {
        { 224,73,143,113},
        { 614,170,129,79},
        { 50,373,200,132},
        { 129,597,102,61},
        { 490,375,211,154},
        { 51,693,211,320},
        { 391,815,92,90},
        { 487,687,244,142},
    },

}
```

Next, split these coordinate sets into the three separate files, frames.lua, coordinates.lua and balloons.lua files, and make changes as needed.

frames.lua

The values within each element:

```
x_position, y_position, frame_width, frame_height [, frame_number ]|[,
link_flag, link_URL]
```

The first set goes in the frames.lua file. As we mentioned above, since we have more camera positions than panels, we're indicating where each double-tap should take you. This is relatively unusual, though, with most of the pages not needing this extra value.

```
[7] = {
    { 49,87,674,280,1},     -- zoom into frame 1
    { 46,384,218,285,3},    -- zoom into frame 3
    { 269,374,233,305,4},   -- zoom into frame 4
    { 514,384,211,281,5},   -- etc.
    { 45,685,225,302,6},
    { 287,685,209,305,7},
    { 509,684,215,306,8},
},
```

The frames.lua file does double duty by providing hotspot links that can be used to open websites (either in Safari or a web popup), take you to iTunes, open an email link in the Mail app. (It may also work to take you to the App Store using a URL like `itms-apps://ax.itunes.apple.com/WebObjects/MZStore.woa/wa/viewSoftware?id=428588931` but this is untested.)

The link_flag meanings are:

```
-1= outside web URL, open in Safari
-2= open web URL in popup
-3= open email in the Mail app
-4= iTunes link
```

Examples:

```
-- open in Safari
{ 254,558,224,49, -1, "http://ElectricEggplant.com"},

-- open in pop-up window
{ 213,131,171,49, -2, "http://AnnieFox.com"},

-- launch iTunes app, go to this page
{ 643,369,62,44, -4, "://itunes.apple.com/us/podcast/family-confidential-
secrets/id313148780"},

-- email link
{ 437,974,240,42, -3,
"mailto:help4kids@freespirit.com?subject=Be%20Confident%20App%20Email"},
```

coordinates.lua

The values within each element:

```
x_position, y_position, camera_view_width, camera_view_height
```

The next set goes into coordinates.lua. No changes need to be made here:

```
[7] = {
    { 101,94,454,268},
    { 396,94,325,269},
    { 50,391,212,274},
    { 184,374,318,305},
    { 516,391,204,273},
    { 50,693,214,292},
    { 291,689,199,297},
    { 514,690,204,297},
},
```

If a page does not have any frames, you still need to indicate that with an empty array:

```
[9] = {           -- no more frames from here to the end
  },
[10] = {
  },
```

balloons.lua

The values within each element:
`balloon_number, x_position, y_position, balloon_width, balloon_height [, preScaled_flag]`

The next set goes into balloons.lua. Each row tells us what we need to do with a specific frame, *so there's a 1-1 correspondence between the number of frames and the number of rows in this file*. Editing the output from CoronaComicsCoordCreator, we need to insert the balloon_number as the first value of each row:

```
[7] = {
    {1,224,73,143,113},
    {2,614,170,129,79},
    {3,50,373,200,132},
    {4,129,597,102,61},
    {5,490,375,211,154},
    {6,51,693,211,320},
    {7,391,815,92,90},
    {8,487,687,244,142},
},
```

Since we're including the balloon number, it's possible to re-use a balloon on multiple frames, as on page 4:

```
    {1,20,67,191,240},  -- Same balloon is displayed in frames 1 and 2
    {1,20,67,191,240},
```

If a frame has no balloon, it still needs to be represented, but with an empty row, as on page 2:

```
[2] = {          -- 4 frames, with balloons only on frames 2 and 4
    {},
    {1,600,81,85,98},
    {},
    {2,207,724,154,90},
    {},
},
```

As with the coordinates.lua file, if a page does not have any frames, you still need to indicate that with an empty array, but with different syntax than the coordinates.lua file:

```
[9] = {
    {},
},
[10] = {
    {},
},
```

The code is currently set up with the assumption that any page that has no frames (i.e., you only view it in full-page mode) does not have a word balloon layer.

For pages that do have more than 1 frame but no word balloons (as is the case with our Page 1), you'll still need to include dummy sprite files. Just take our Page1.lua and Page1@2x.lua files and rename them to the page numbers you need. And in the images/sprites folder, duplicate our Page1.png and Page1@2x.png and rename to the page numbers you need. A good future enhancement might be to have a table that indicates which pages have no word balloons to avoid having to do this.

Sound

Sound is all table driven and stored in the sound.lua file. (Note that this is true for all pages but not the opening cover animation.) There are two levels of sound control available: sounds for the entire page (e.g., when you're in zoomed-out mode), and sounds for each frame (zoomed-in mode). If there is no sound defined for a specific frame, then the sound defined for the entire page is used instead.

It's important to remember that you're managing not only when to turn on a sound, but when to turn it off as well. If frame 2 has music playing but frame 3 doesn't, then you need to make sure you turn that sound off in frame 3.

Likewise, if you want the same sound to play in frames 4-6, you have to make sure you turn it on in each one. That's because there are several ways to zoom in to a frame without visiting the frames on either side: double-tap, returning from the Info page, and via the bookmarking feature when the app starts up again.

This also applies when you're moving from page to page (either forwards or backwards). For example, if you're moving from page 4 to page 5 or (page 5 to page 4) in zoomed out mode, make sure the previous page's sounds are turned off. Same thing when moving from the last panel of page 4 to the first panel of page 5, or backwards, the first panel of page 5 to the last panel of page 4.

Managing Sound Channels

For our sound code to work properly, we're manually managing which sound channel most of the sounds will be played on. This may no longer be necessary within the Corona SDK, but at the time we wrote the code, that was the only way to know if a channel was playing and whether to turn it on/off.

In main.lua, we reserve the number of channels we plan to use:

```
audio.reserveChannels( 14 ) -- reserve 14 audio channels for manual allocation
```

You can change this value as needed for your app.

At the top of the sound.lua file, we're reserving various channels for music, ambient sound, and looping sounds. We have to assign them to channels because we need to be able to query whether the sound's playing, and at what volume.

This complicates matters a bit since we need to make sure we're not allocating the same channel in adjacent frames (so we can have clean cross-fades).

Take a look at the sound.lua file from the sample code. In the comment at the top you'll see how we're using these 14 channels:

```
-- channel 1-2      -- music channels
-- channel 3-6      -- ambient (e.g., outdoor background sound)
-- channel 7-14     -- looping sounds
-- channel 15-32    -- open channels for UI, repeating sounds, or quick sounds
```

Next in the file, we set up named values to make it easier to follow which channel is being used for what:

```
local musicCh1 = 1
local musicCh2 = 2
local ambCh1 = 3
local ambCh2 = 4
local ambCh3 = 5
local ambCh4 = 6
local loopCh1 = 7
local loopCh2 = 8
local loopCh3 = 9
local loopCh4 = 10
local loopCh5 = 11
local loopCh6 = 12
local loopCh7 = 13
local loopCh8 = 14
local loopCh9 = 15
```

Then we actually load all the sounds we're using. This could become a problem for very long books, or if there are many more sounds used than the 75 we use in our actual graphic novel app… that's because keeping them all in memory, even if we're using loadStream for most, locks up memory. A better solution might be to load needed sounds on a chapter-by-chapter basis (unloading those no longer needed).

Note that in this sample code we're using audio.loadSound, but in our full program, we're mostly using audio.loadStream, especially for the longer sounds (ambient, music). Just remember that a sound loaded with loadStream cannot be shared simultaneously across multiple channels.

Finally we have our table for each page and frame indicating what sounds should be played, for how long, etc. Here are the parameters that can be included:

- channel — which sound channel to use. If none is specified, then use one of the open channels between 15-32 is used.

- sound — handle the handle of the loaded sound we want to use, e.g., sndAmbOutdoors.
- delay — how long to wait until starting the sound. The code automatically makes sure we're still on the correct frame when it actually is time to trigger the sound.
- start_volume — initial volume we want the sound to start at (or, if no fade, then also to remain at), values of 0-1
- targ_volume — volume we want the sound to end up at after a fade completes values of 0-1.
- fade_duration — how long should it take for the sound to reach its targ_volume (in milliseconds)
- loop — how many times to loop. We use -1 if it's to loop forever.

The structure of the data is that for each page, the first row of values represents what to do for that page when it's in full-page mode.

The second row is for frame 1, third row for frame 2, etc. If we want the frame to play exactly the same thing the page is playing, we can give it an empty value, as in page 7, frames 1, 2, 6, 7:

```
   { }, -- frame 1
   { }, -- frame 2
```

Note that we can assign multiple sound commands for each frame. Take a look at frame 3 of page 6:

```
{ {channel=ambCh1, sound=sndAmbPoolFlashback, targ_volume=.3,
fade_duration=1000, loop=-1},                          -- frame 3
   {delay=500, sound=sndIceInGlass, start_volume=1},    -- no channel given
   {delay=1000, sound=sndIceInGlass, start_volume=1},   -- so open channels
   {delay=2500, sound=sndIceInGlass, start_volume=1},   -- 15-32 are used
   {delay=4500, sound=sndIceInGlass, start_volume=1},
   {delay=7500, sound=sndIceInGlass, start_volume=1},
   {delay=8200, sound=sndIceInGlass, start_volume=1},
   {delay=8900, sound=sndIceInGlass, start_volume=1},
   {channel=loopCh2, targ_volume=0, fade_duration=1000}, -- sndAbbyCryFlashback
},
```

Here we're saying we'll be using ambCh1 as the sound channel to play the sndAmbPoolFlashback sound, change the volume from whatever it is to .3 over 1 second (1000 ms), and keep looping this sound forever.

In the next line, we're triggering the sound of ice clinking in a glass multiple times, each one delayed a different amount of time (having a way to trigger the same sound over and over again with random delays would be a good enhancement). We're not assigning a channel because it's a short sound and it's ok if multiple instances of it are running at the same time. And we are giving it a starting volume of 1 (full on).

Finally, we're fading out the sndAbbyCryFlashback sound which gets triggered in the next frame (frame 4 of page 6). That way if the reader moves from frame 4 back to frame 3, we'll be sure the sndAbbyCryFlashback sound fades out.

The code that handles this table is now in Reader.lua (lines 1002-1107). It would probably make sense to move these routines into their own file...

Cover Animation

Our cover animation uses X-Pressive.com's Particle Candy package for a dust effect when the "Graphic Novel Edition Lite" stamp hits page. If you own this software, include your lib_particle_candy.lua file in the root folder and uncomment line 127 in main.lua:

```
Particles = require("lib_particle_candy")
```

There's also a flag you can set to false if you don't want to use any cover animation. It's on line 145 of main.lua:

```
_G.coverAnimation = true   -- set to false if you want to start on
                           -- Page0 without the cover animation
```

Hardwired Code

You'll probably want to comment out or remove code that we've added that is specific for out own app. This includes:

main.lua

- Setting up the URL into the App Store to rate the app, line 167 (replace with your own URL).
- Parameters for different page ranges, lines 169-188.
- URL used for our "newsfeed", lines 177 and 187 (for a lite version).
- Setting up the Info page buttons, lines 442-582 (with the chapter buttons set up on 556-582).
- Default domain to test if you're online if none are passed as a parameter, line 634 (please use your own domain).
- Buttons to buy the app and see the review, shown on the last page for the Lite version (i.e., this demo version), lines 751-835.

Reader.lua

- The code to add the two help messages on page 1, lines 385-540 for the doubleTapAnimation() and infoTapAnimation() functions, and the code in showFrame() to call these routines on page 1, lines 1168-1175.
- Also in showFrame(), lines 1155-1168 to add touchable areas over the "Continue…" text on some of the About pages plus . Probably would be better to add another class of touchable links in the frames.lua file to handle jumping to a specific page, and the corresponding code in Reader:touch() to act on this touch, lines 1445-1446.

- Naming our bookmarks file “msc1prefs.txt” in line 1592.

promote.lua

- “Rate this app” alert message text, line 10.
- Domain to use as a test to see whether we have Internet access, line 158 (please change this to your own domain).

There are probably many other areas where we coded this for our own purposes rather than creating a flexible general case tool. Feel free to make your modifications, and if you have fixes or changes that add more abilities, feel free to publish those changes.