iTunes Extras/iTunes LP Development: Template How-To v1.1

1-26-2010

Contents

How to Use the Templates

This chapter covers step-by-step how to use the templates to create iTunes LPs and iTunes Extras. It provides an overview of iTunes LPs and iTunes Extras, explains the anatomy structure, and provides a checklist of items you'll need before you begin to use the templates. After you've created your own iTunes LP or iTunes Extras, download the materials from the Testing section of the iTunes LP and iTunes Extras site to test functionality, navigation, and asset linking.

Automatic, electronic submission of your iTunes LP or Extra is scheduled for the first quarter of 2010. Until then, the submission process is manual and limited. Please contact your label or studio rep for details and consideration. An existing iTunes contract is required. Your iTunes LP or iTunes Extras will be reviewed by the iTunes team for appropriateness of content and for technical quality.

The templates provide a starting point. The templates include basic page layouts and built-in TuneKit animations and scripts. You can use the template to create a simple, straightforward package, or you can explore TuneKit's capabilities to enhance the effects in your package. For more information on TuneKit, see *TuneKit Reference*.

Important: This is a preliminary document. Although it has been reviewed for technical accuracy, it is not final. Apple is supplying this information to help you develop iTunes LP and iTunes Extras. This information is subject to change.

About iTunes LP and iTunes Extras

iTunes LP and iTunes Extras give you a way to provide more content to your fans. With iTunes LP, you can give your customers the lyrics to songs, photos, and liner notes just like the old LP format, as well as videos, interviews, original artwork created by you, and interactive games. With iTunes Extras, you provide the experience made popular on DVDs such as deleted scenes, trailers, behind-the-scenes interviews, and more.

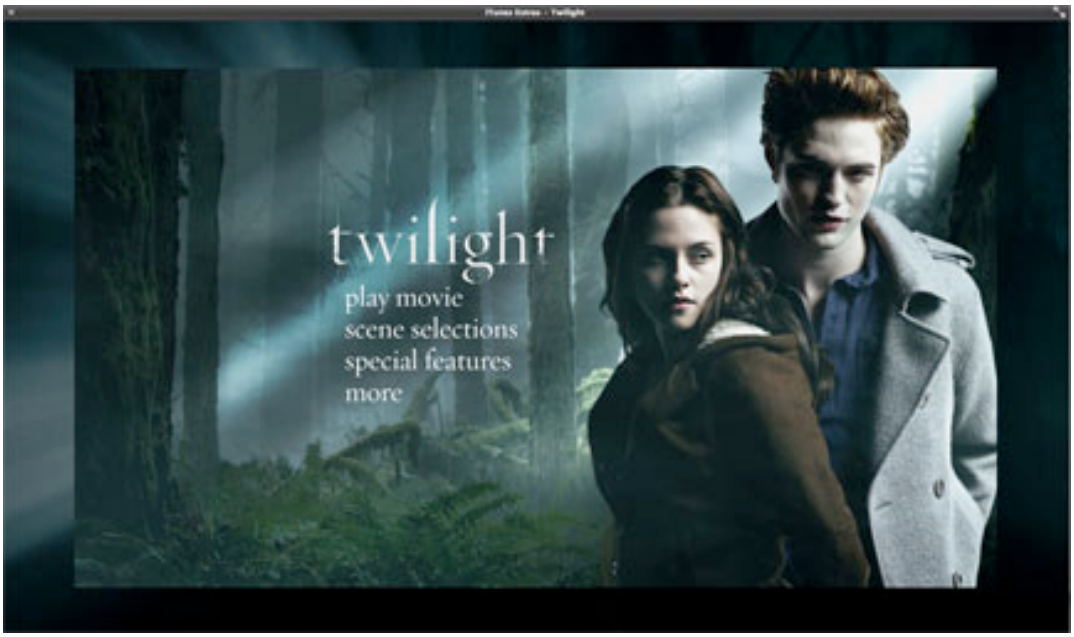
The following lists some of the extra content iTunes LP and iTunes Extras can contain, but it is not exhaustive.

- Background audio
- Videos (added content for things like deleted scenes, artist interviews, and so on)
- Lyrics
- Photos
- Artwork created by the artists
- Chapters
- Visualizers (animated screen that displays while music is playing)

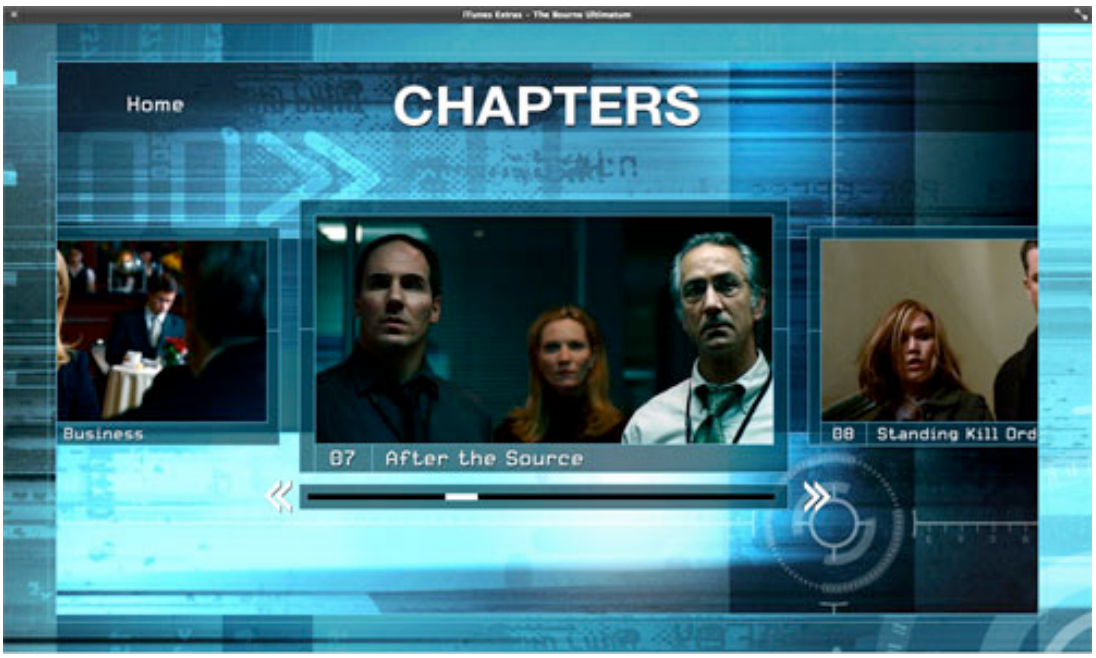
iTunes Extras Page Examples

As an example, iTunes Extras might have a Home page, a Chapters page, a Features page, and a More page. There is no limit to the number of pages iTunes Extras can have.

The *Twilight* Home page where the user can navigate to the other pages:



The *Bourne Ultimatum* Chapters (also sometimes called scene selections) page shows images and titles for the chapters in a movie, which allows the user to jump to a selected chapter:



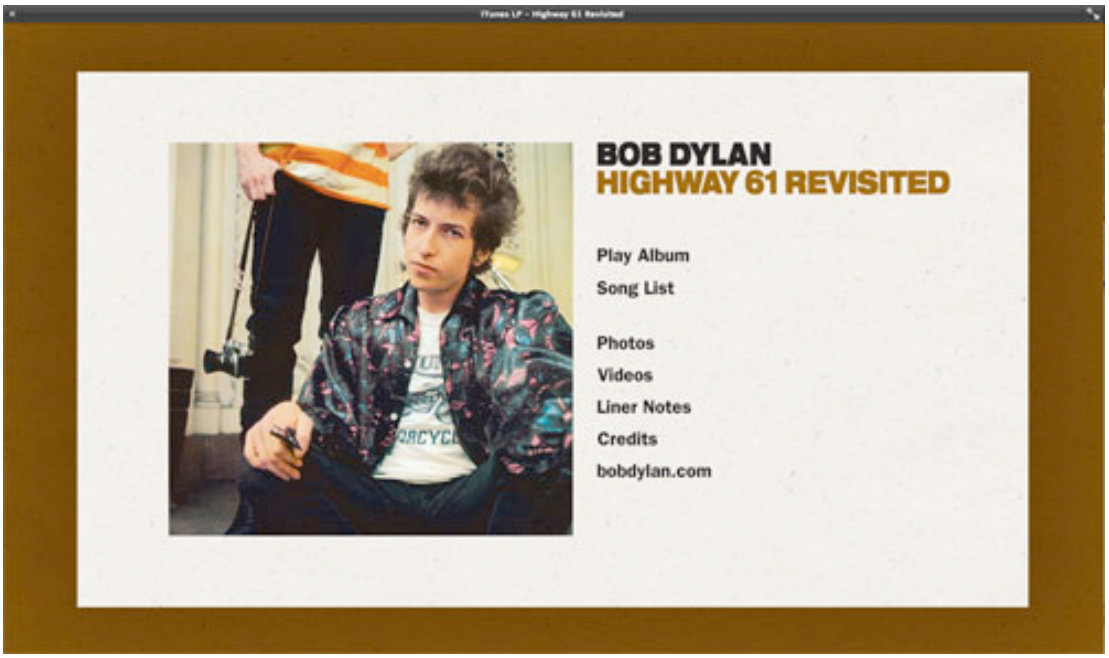
The *Bourne Ultimatum* Features page allows the user to select among several types of extras:

The *Quantum of Solace* More page provides links to external resources, such as the studio web site or iTunes albums and movies.

iTunes LP Page Examples

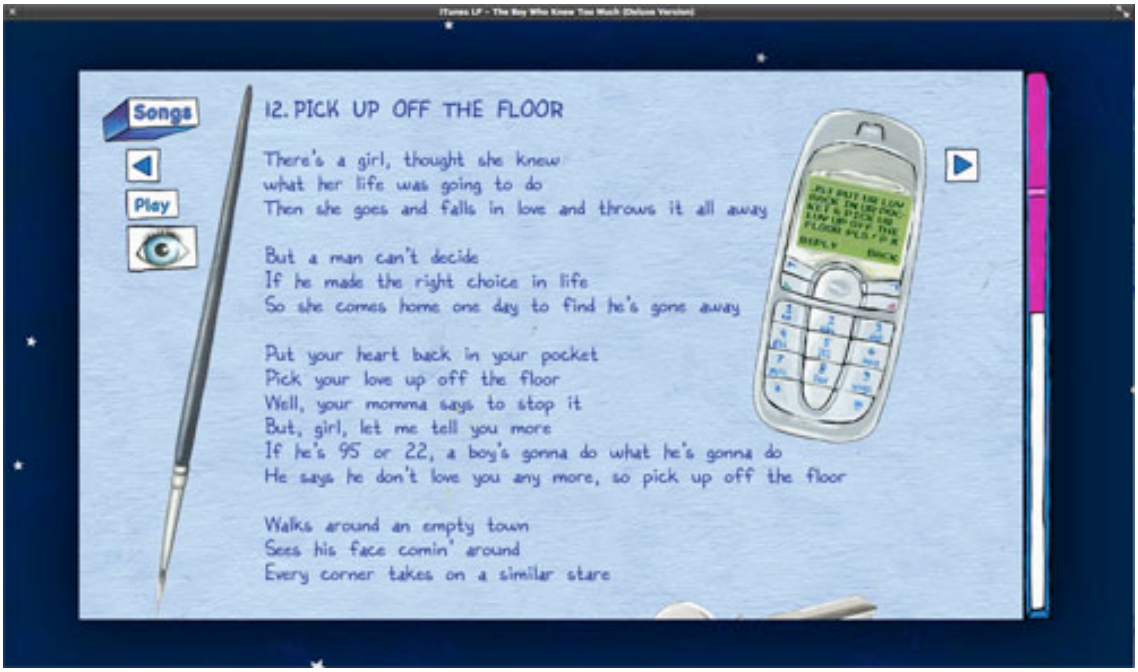
As an example, iTunes LP might have a Home page, a Track List page, a Liner Notes page, a Videos page, a Visualizer page, and a Credits page. There is no limit to the number of pages an iTunes LP can have.

The *Highway 61 Revisited* Home page where the user can navigate to the other pages:

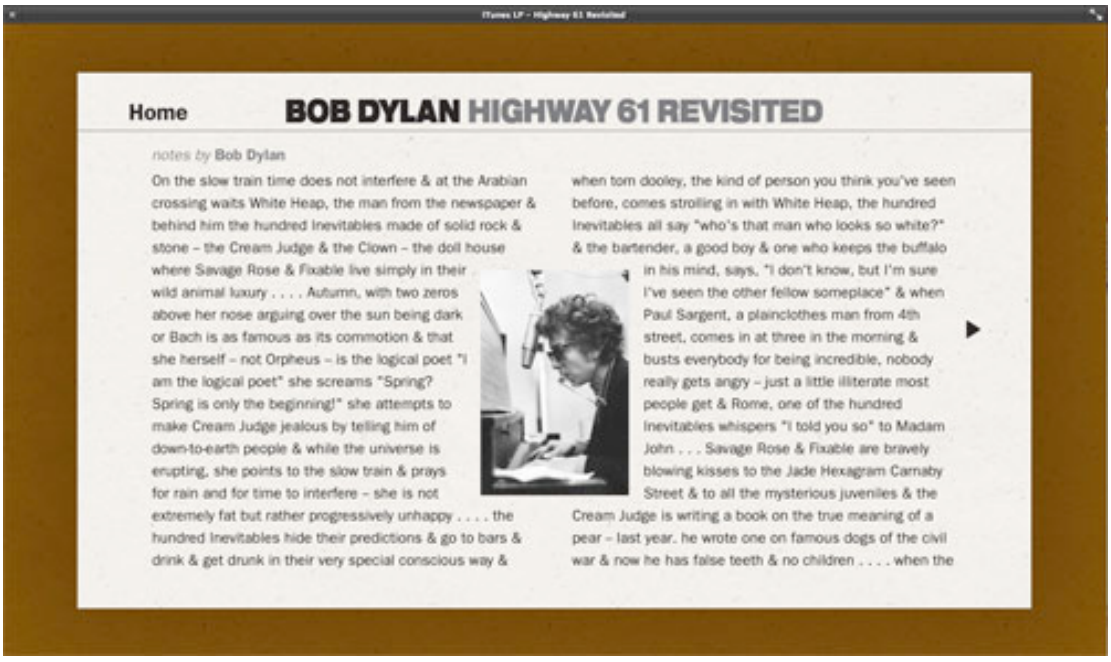


The *Boy Who Knew Too Much* Song List page shows titles for the songs, which allows the user to jump to a selected song:

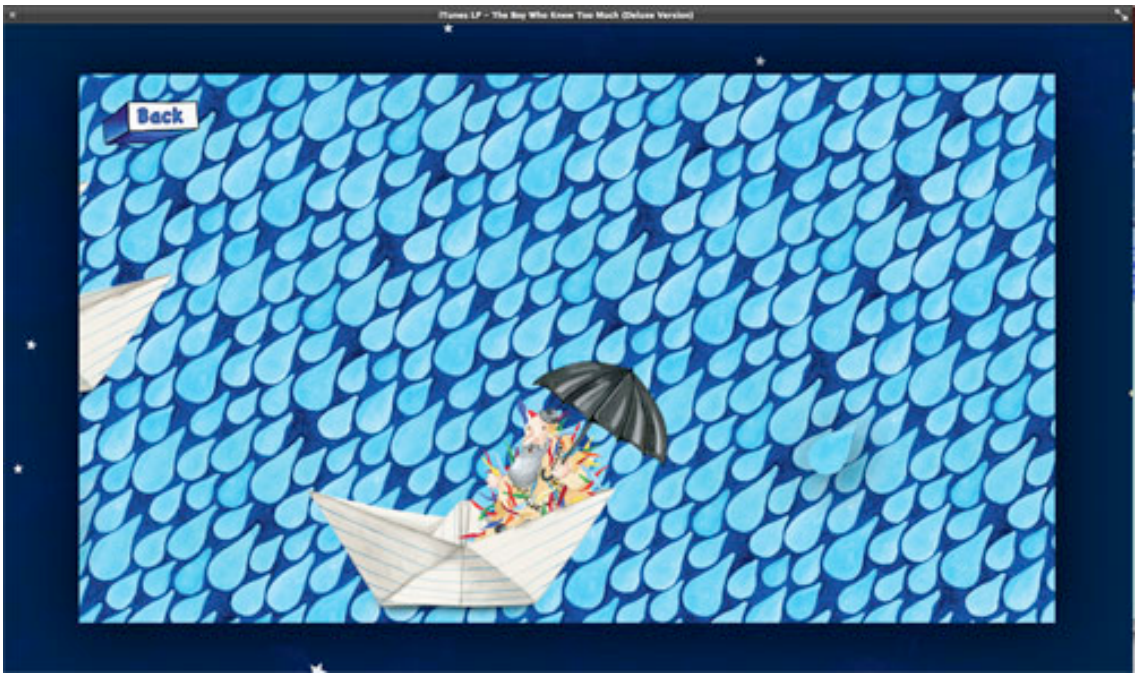
The *Boy Who Knew Too Much* Lyrics page allows the user to read the lyrics as the song plays:



The Highway 61 Revisited Liner Notes page provides notes on the making of the album.

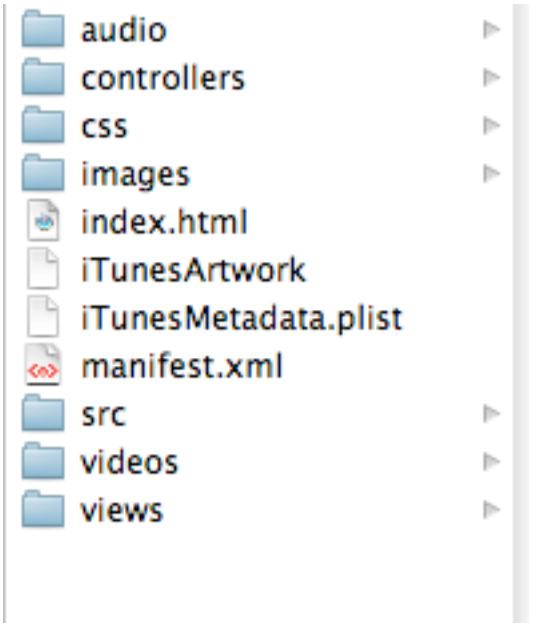


The *Boy Who Knew Too Much* Visualizer page displays an animation as a song plays.

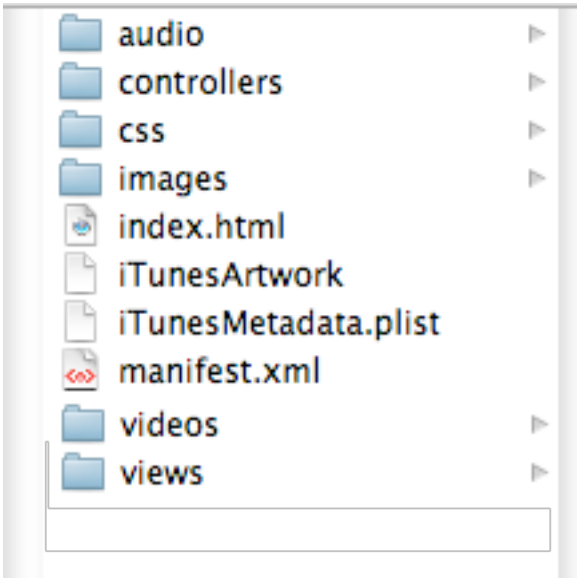


Anatomy of iTunes LP and iTunes Extras

iTunes LP and iTunes Extras consist of folders and files that contain things like assets, scripts, and page layouts. The following screenshot shows a typical structure of an iTunes Extras.



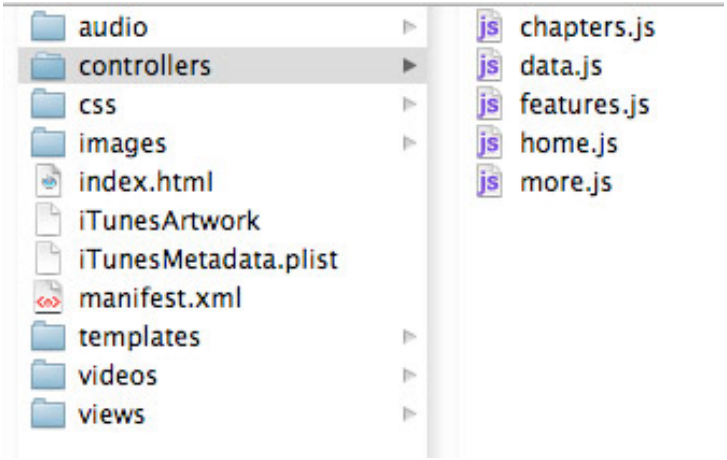
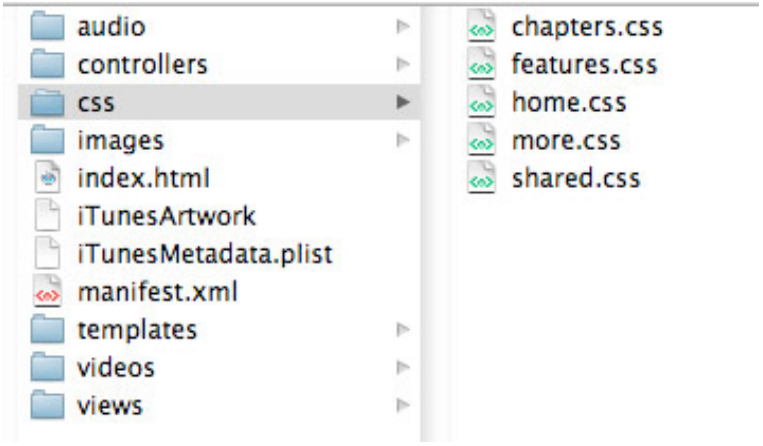
The following screenshot shows a typical structure of an iTunes LP. The structure is similar to the iTunes Extras shown above, but with iTunes LPs, you can have visualizers, which can be stored in an additional folder for those elements.

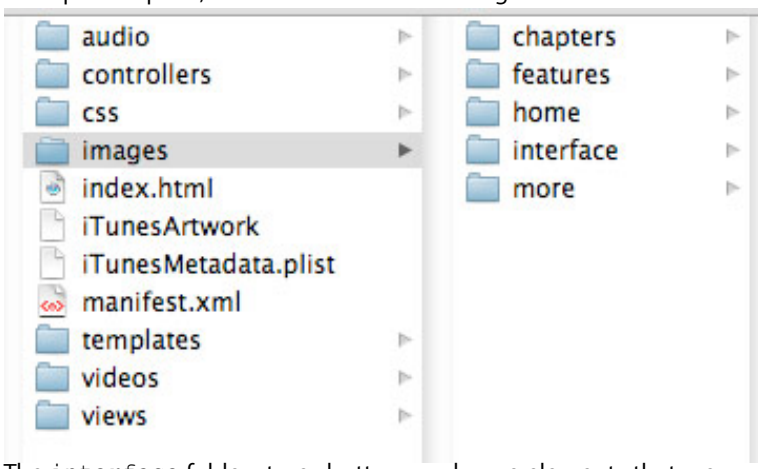


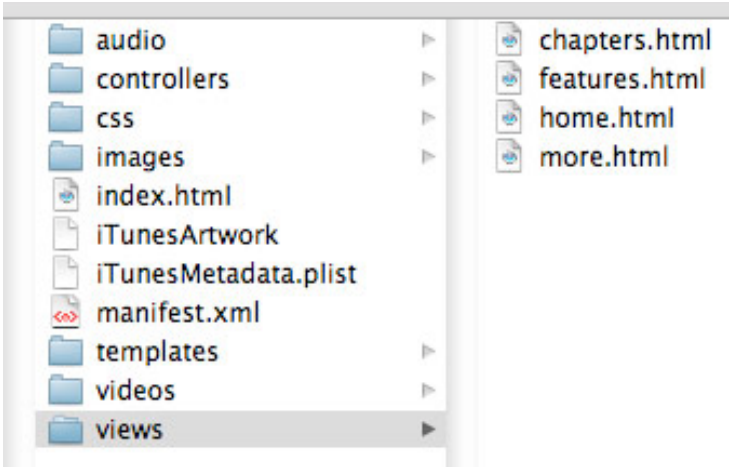
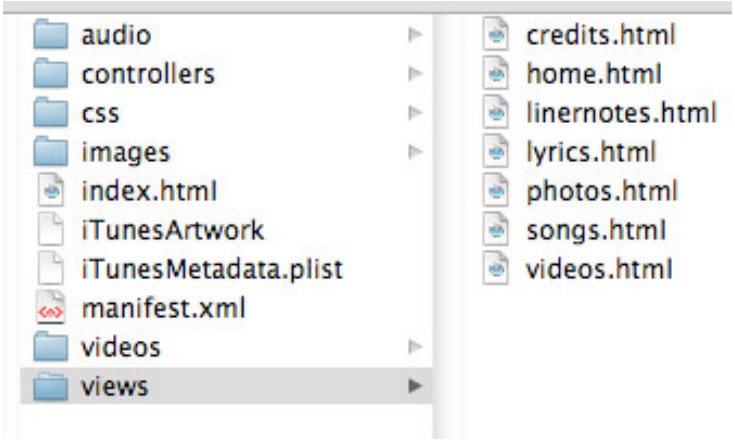
An iTunes LP has the `.itlp` extension and an iTunes Extras has the `.ite` extension. With Mac OS X, the iTunes Extras or iTunes LP is displayed as a single file. To see the contents of the file, right-click it and choose **Show Package Contents**. On Windows, the folders are not bundled into a single file; you will just see the folder and file structure.

The table below shows one example of how an iTunes Extras could be set up and the files used.

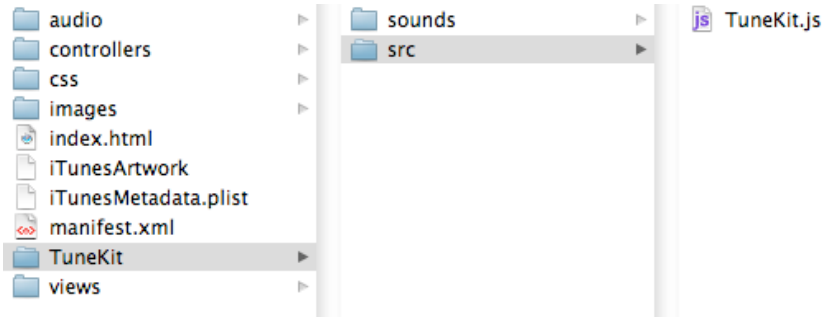
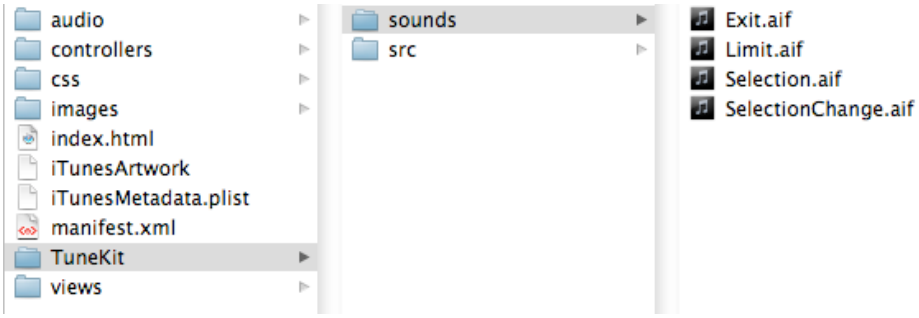
Folder/File	Purpose
audio	Stores audio files used in the iTunes Extras, such as a short music excerpt that provides background audio that loops.

controllers	Stores Javascript controller files that control elements and actions on a page and the navigation and transitions between pages. Each controller should have a corresponding HTML file in the `views` directory. The `data.js` file is where you define the background audio loop, as well as the number of chapters, number of photos, and the names of the bonus content you'll reference from the Features view in the Movie template and the Photos and Videos views in the Music template. The `data.js` file is where you declare all global media and data configuration parameters. 
css	Stores the CSS files for each View. The CSS files control the positioning of buttons, text, and images on the page. It also includes some interactive elements and animations. The shared.css file stores images, positions, elements, and animations that are shared among the views. 

images	Stores all the images used in the iTunes LP or iTunes Extras. In this example template, each view has its own images folder.  The `interface` folder stores buttons and page elements that are shared among the views, for example, the bleed, play and resume buttons, and arrows.
index.html	The page that opens when the user starts an iTunes LP or iTunes Extras. Often called the Home page.
iTunesArtwork	The icon for the iTunes LP or iTunes Extras. Also sometimes referred to as album cover art or film poster art. The format should be either PNG or JPG without the file extension. The iTunesArtwork file is automatically generated at the time of purchase, so there is no need to author this file except for testing purposes. It must be removed prior to submitting to the iTunes Store.
iTunesMetadata.plist	Describes the metadata for displaying the iTunes LP or iTunes Extras in iTunes. Metadata includes things like description, genre, copyright year, artist names, and so on. This file is automatically generated at the time of purchase, so there's also no need to author this file except for testing purposes. The `plist` file also contains name, media kind and XID mapping among other metadata. The XID mappings are used as identifiers that iTunes uses to associate the iTunes LP or iTunes Extras with the media in the library as well as syncing to Apple TV. The iTunesMetadata.plist file must be removed prior to submitting to the iTunes Store.

manifest.xml	The manifest is an XML file that must live in the top level folder of the iTunes Extras or iTunes LP; that is, it must be a sibling of the main index.html file. The purposes of the manifest are to: ■ identify the version of the iTunes LP or iTunes Extras ■ call out what platforms it is compatible with ■ identify any items in the user's iTunes Library that are to be playable via the user interface of the iTunes LP or iTunes Extras
videos	The videos folder stores bonus videos, such as deleted scenes and behind-the-scenes interviews, **as well as bonus audio tracks (with the exception of background audio which needs to live in the "audio" folder)**. The main video asset or main album asset are not stored in this folder. See Where Do the Core Assets Go?
views	Contains the HTML files for each view in the iTunes LP or iTunes Extras. A view defines the layout of the page. Each view should have a corresponding JavaScript controller in the `controllers` folder. The following shows a typical iTunes Extras:  The following shows a typical iTunes LP: 

In addition, TuneKit is included with each template. The following table explains the contents of the TuneKit folder:

src	Contains javascript libraries provided with TuneKit. 
sounds	Contains sounds for your use. These sounds can be used to provide audio feedback as the user navigates using the Apple TV remote. See the TuneKit documentation for how to use them. 

Where Do the Core Assets Go?

As mentioned above, you identify any items in the user's iTunes Library that are to be playable via the user interface of the iTunes LP or iTunes Extras in the manifest.xml file by XID. For albums, you can have both audio (for song tracks) and video (for video tracks) as part of your album tracks and you need to provide XIDs for all tracks. For movies, you need to provide an XID for the main movie. These Core Assets all live outside the iTunes LP (.itlp) and iTunes Extras (.ite). XIDs are what link the core asset files to the iTunes LP or iTunes Extras in iTunes. Keep your bonus content, such as photos, interview videos, and lyrics inside the iTunes LP or iTunes Extras (for example, in the audio or video folders). iTunes will then put the audio and video files in the customer's library. For details on which XID to use, see the *Development Guide*. Note that if you put all the asset files inside the iTunes LP or iTunes Extras, the audio and/or video files will not show up in the users' libraries and they will be unable to sync the audio and video files to their iPod or Apple TV.

Getting Started Checklist

The checklist below is a quick review of things you'll need to prepare before you can create your iTunes LP or iTunes Extras. This checklist assumes you will be starting from either the iTunes LP or iTunes Extras template, which comes with the folder/file structure already in place.

You'll need to do a few preliminary steps before getting into the templates, if you want a more customized look, read the *Design Best Practices* section, especially the section on creating images. Before using the template for either iTunes LP or iTunes Extras, you'll need to do the following:

- Decide how many views you need.
- Create an audio file to use for the background audio. This audio should be small and it must be m4a.
- Create a background image for the views. You can create one background for all views or create a different background for the views.
- Create two images for each navigation button if you don't want to use the buttons supplied with the template. One image is the button and the second image is the button as you want it to appear when hovered over.
- Create images for the titles for each view.
- Create a "bleed" graphic that flows beyond the 1280 x 720 pixel viewing area. Using a bleed graphic is optional, but it does provide a better viewing experience for your users.
- Create and size the thumbnail images for bonus scenes and chapter images. Follow the image guidelines below when sizing.
- Create the labels for bonus scenes and chapters. Follow the font guidelines below.
- Design interface elements for any navigable elements on the screens, for example, home, back, chapters, resume, play, actual play buttons, iTunes store links, and so on.
- Place your images in the appropriate folders. For example, in the template, buttons are stored in the images/interface folder. The title image used for each view is stored in the images folder for that view (for example, the title for the home view is stored in images/home.)
- Read up on TuneKit. See the next section for a very brief description of controllers. Read more in the TuneKit Reference document.

What Are Controllers?

The template comes with several built-in controllers. The controllers set the animations actions on the page, such as sliding images, setting actions for buttons, setting highlights (highlights are important for Apple TV because there is no cursor). Each view in your iTunes LP or iTunes Extras has a controller and the controllers

are stored in the controllers folder. The controller gives the view functionality; when a user interacts with the view, they are actually interacting with the controller. Although many controllers also provide animations, it is better to set the animations in the CSS because CSS animations are faster and smoother.

There are many more controllers in TuneKit than are listed here. This list just gives you some background into the basic controllers provided with the template. You can find out much more about controllers by reading the TuneKit documentation.

Controller	Description
TKController	Base class for all TuneKit controllers. It adds actions to elements and provides some effects, but no animations. You can change the parameters of the actions and effects in the CSS file.
TKPageSliderController	Provides actions that allow the user to slide between a number of images. It is used to slide images in and out of the focus position on a page. In the movie template, this is used in the Chapters view. In the music template, this is used in the Photos view. You can change the parameters of the actions and effects in the CSS file, for example, position of elements, degree of the fade, and whether or not the images resize.
TKTabController	Provides a way to add tabs to the view, where each tab includes a set of related elements within the view. Only a single tab can be displayed at a time and each tab shows a sliding set of frames that play a clip when the user clicks the play button. In the movie template, this is used in the Features view. In the music template, this is used in the Videos view. You can change the parameters of the actions and effects in the CSS file, for example, the position of elements, the degree of fade, and whether or not the images resize.

Using the iTunes Extras Template

The instructions in this section take you through creating iTunes Extras using a template. The template consists of four basic pages called views: a Home page view, a Chapters view that allows the user to begin playing the movie at a selected chapter, a Features view that provides access to extras, and a More view used for linking to external websites or to the iTunes Store. The steps will show you how you can substitute your content.

These are the general steps you can follow to edit the iTunes Extras template. Each of these steps has more detailed instructions below.

1. Edit the `iTunesMetadata.plist` file. This specifies the metadata for iTunes Extras.
2. Edit the `manifest.xml` file. This is where you declare the asset IDs.
3. Edit the `data.js` file.
4. Create your images and place them in the correct folders.
5. For each view (Home, Chapters, Features, and More):
   - Edit the `.html` file. This is where you set up the first page of the view the user will see. It is used to navigate to other parts of your iTunes Extras.
   - Edit the `.css` file. This is where you can set the location of elements and set animations, such as transitions, fades, rotations, and scaling.
   - Edit the `.js` file. This is where the actions have been set up for the view.
6. Edit the `shared.css` file.

Editing the iTunesMetadata.plist File

The `iTunesMetadata.plist` file describes the metadata for displaying iTunes LP or iTunes Extras in iTunes. Metadata includes things like description, genre, copyright year, artist names, and so on. This file is automatically generated at the time of purchase, so there's also no need to author this file except for testing purposes.

iTunesMetadata.plist

iTunesMetadata.plist

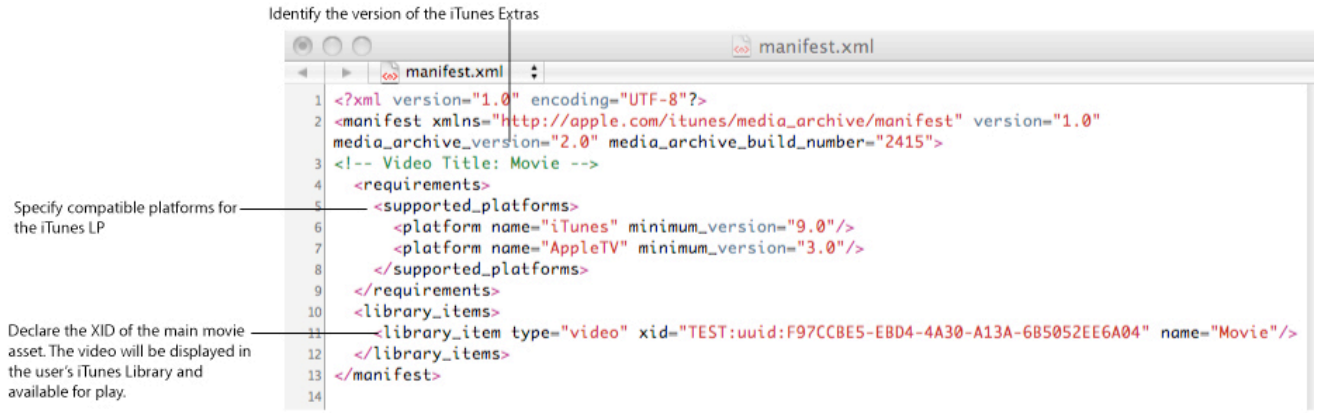
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Metadata</key>
    <dict>
        <key>artistName</key>
        <string>Artist</string>
        <key>description</key>
        <string>A short description</string>
        <key>longDescription</key>
        <string>A long description</string>
        <key>genre</key>
        <string>The genre</string>
        <key>itemName</key>
        <string>iTunes Extras - Movie</string>
        <key>kind</key>
        <string>feature-movie</string>
        <key>releaseDate</key>
        <string>2009-01-01T00:00:00Z</string>
        <key>sort-artist-status</key>
        <integer>128</integer>
        <key>sort-name</key>
        <string>Sort Name</string>
        <key>sort-name-status</key>
        <integer>1</integer>
        <key>year</key>
        <integer>2009</integer>
    </dict>
    <key>Name File</key>
    <string>iTunes Extras - Movie.ite</string>
    <key>associated-adam-ids</key>
    <array>
        <integer>123456789</integer>
    </array>
    <key>xid-asset-mapping</key>
    <dict>
        <key>studio:upc:123456789</key>
        <array>
            <integer>123456789</integer>
        </array>
    </dict>
</dict>
</plist>
```

Editing the manifest.xml File

The manifest is an XML file that must live in the top level folder of the iTunes Extras; that is, it must be a sibling of the main index.html file. The purposes of the manifest are to:

- identify the version of iTunes Extras
- call out what platforms iTunes Extras is compatible with
- identify any items in the user's iTunes library that are playable via the iTunes Extras user interface

For more detailed information on the manifest.xml format and the structure of XIDs, see the *Development Guide*.



Storing the Background Audio and Bleed image

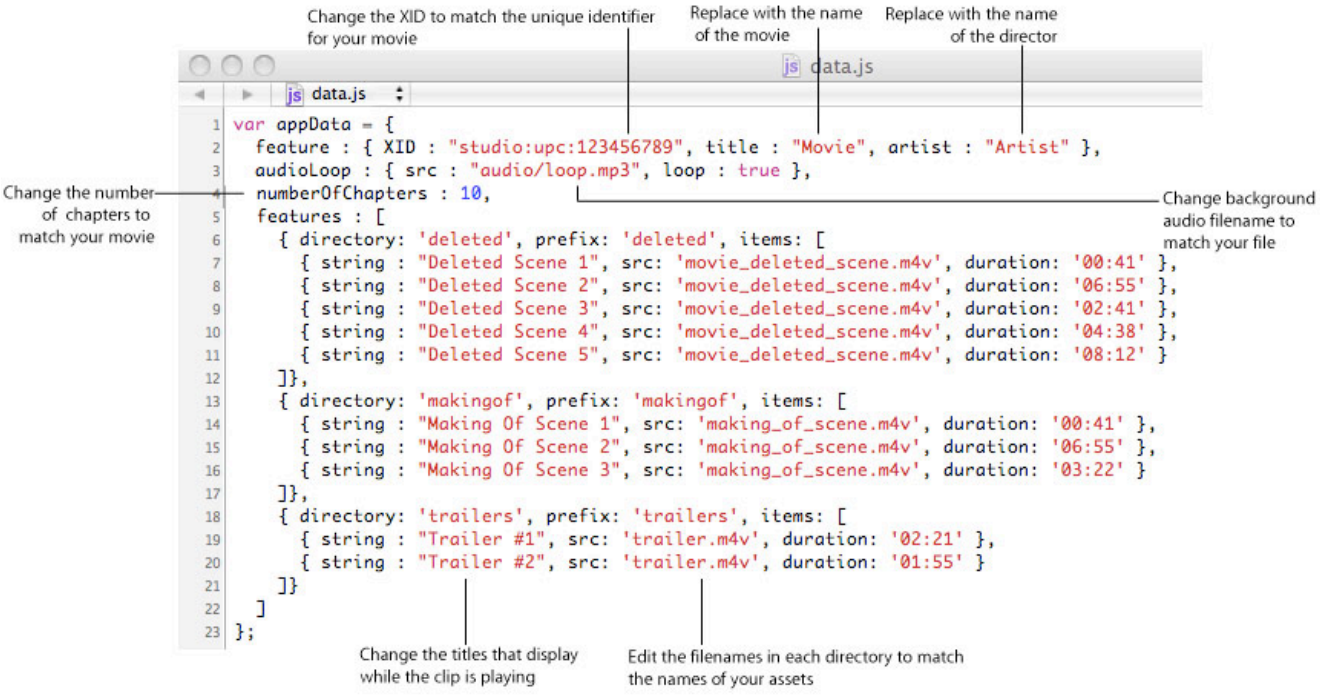
Once you have created the background audio and bleed image, place them in the appropriate folders. The background audio is an audio clip that is played when a user opens an iTunes LP and iTunes Extras. You can have the audio clip play once or repeat over and over.

- Put your background audio file in the audio folder. In `data.js`, change the filename to match the name of your audio file.
- Replace `images/interface/bleed.png` with your image. If you are changing the filename or file type (to JPG), make sure you also change the reference to the file in shared.css. To change the location of the bleed, edit the `shared.css` file.

Editing the data.js File

The `data.js` file stores things like the number of chapters, the unique identifier for the movie asset, the background audio, and titles and filenames used in the Features view. Before opening the template, edit the `data.js` file. Open it in a text editor, such as TextMate.

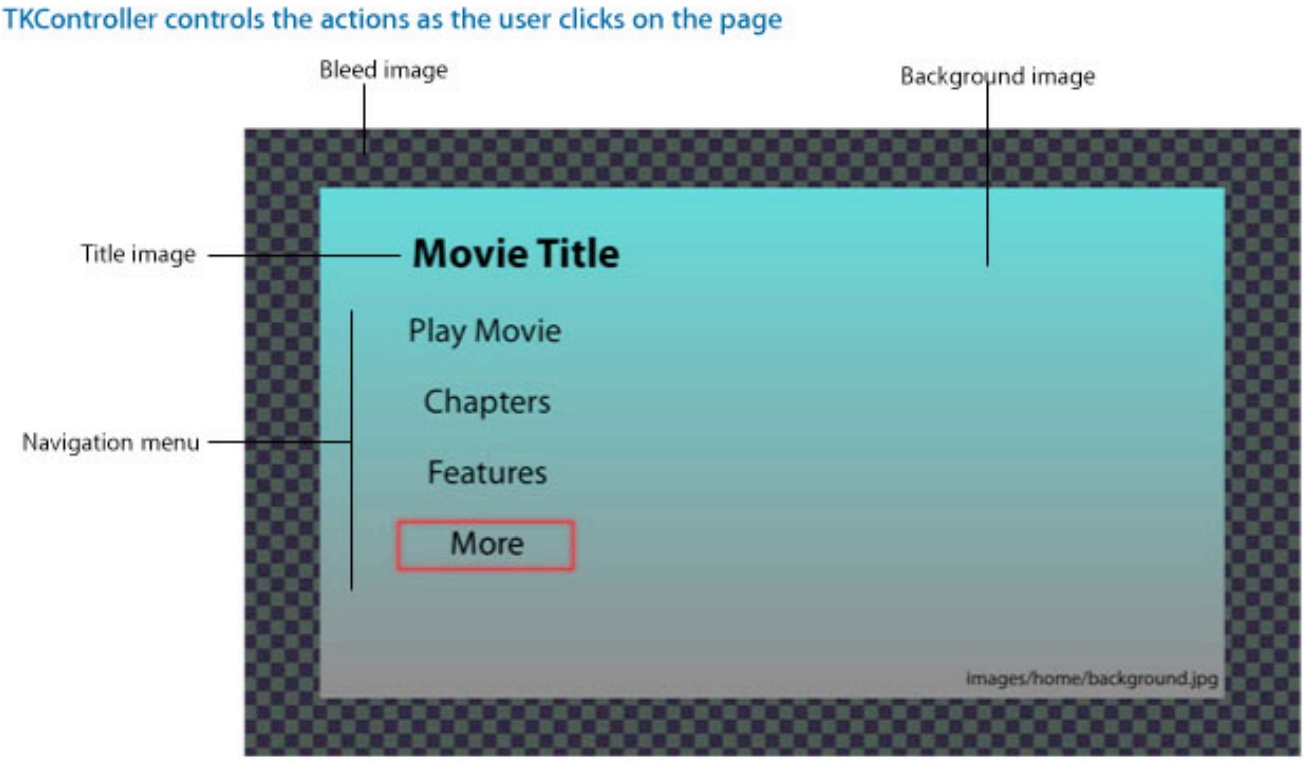
- In `data.js`, change the filename of the background audio to match the name of your audio file. This is where you also specify whether or not the audio should repeat by setting "loop" to "true" or "false."
- Identify the XID for the main movie asset.
- Change the references to your bonus content (for the template, these appear on the Features view).




```
var appData = {
  feature : { XID : "studio:upc:123456789", title : "Movie", artist : "Artist" },
  audioLoop : { src : "audio/loop.mp3", loop : true },
  numberOfChapters : 10,
  features : [
    { directory: 'deleted', prefix: 'deleted', items: [
      { string : "Deleted Scene 1", src: 'movie_deleted_scene.m4v', duration: '00:41' },
      { string : "Deleted Scene 2", src: 'movie_deleted_scene.m4v', duration: '06:55' },
      { string : "Deleted Scene 3", src: 'movie_deleted_scene.m4v', duration: '02:41' },
      { string : "Deleted Scene 4", src: 'movie_deleted_scene.m4v', duration: '04:38' },
      { string : "Deleted Scene 5", src: 'movie_deleted_scene.m4v', duration: '08:12' }
    ]},
    { directory: 'makingof', prefix: 'makingof', items: [
      { string : "Making Of Scene 1", src: 'making_of_scene.m4v', duration: '00:41' },
      { string : "Making Of Scene 2", src: 'making_of_scene.m4v', duration: '06:55' },
      { string : "Making Of Scene 3", src: 'making_of_scene.m4v', duration: '03:22' }
    ]},
    { directory: 'trailers', prefix: 'trailers', items: [
      { string : "Trailer #1", src: 'trailer.m4v', duration: '02:21' },
      { string : "Trailer #2", src: 'trailer.m4v', duration: '01:55' }
    ]}
  ]
};
```

Changing the Home Page

On the sample Home page template, the movie title and navigation text are actually images. You can easily substitute your own images. You could also replace the images with text (using an SVG font). The Home view uses the basic TKController.

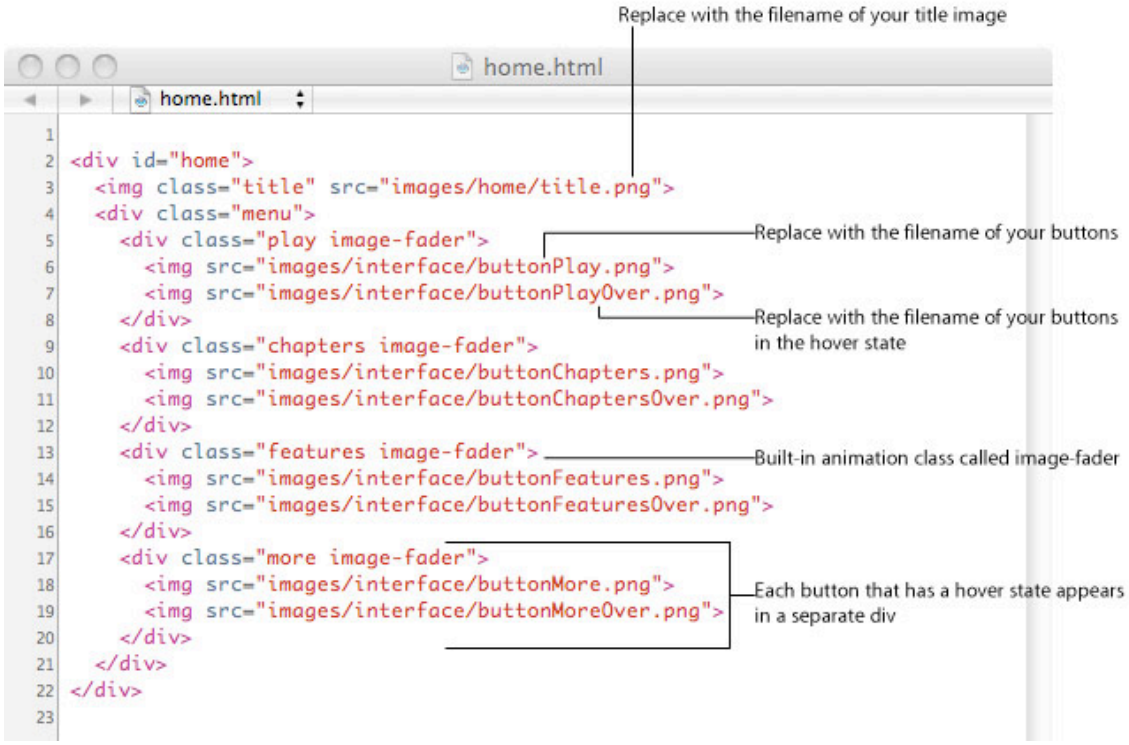


To set up your Home page view:

1. In the Movies template folder, open the views folder, and open `home.html` with a text editor, such as TextMate.

2. Change the filename for the title image to match the filename of your image.

3. Replace the filenames of the buttons for the non-hover states with the names of the non-hover state button images you created.



4. Replace the filenames of the buttons for the hover states (ending with Over.png) with the names of the hover state button images you created.

5. In the Movies template folder, open the css folder, and open `home.css` with a text editor, such as TextMate.

home.css

```
#home {
  background-image: url("../images/home/background.jpg");
}

/* Title */

#home > .title {
  position: absolute;
  left: 100px;
  top: 55px;
}

/* Menu */

.menu {
  position: absolute;
  left: 100px;
}

.menu > .play {
  top: 160px;
}

.menu > .chapters {
  top: 260px;
}

.menu > .features {
  top: 360px;
}

.menu > .more {
  top: 460px;
}
```

Replace this filename with the name of the image you are using for the background for this view

Change the position of the title and/or buttons by editing the values

6. To see how the navigation and animations work for the Home view, open `home.js`.

The controller used on this page

home.js

```
var homeController = new TKController({
  id: 'home',
  actions : [
    { selector: '.menu > .play', action: bookletController.playFeature }
  ],
  navigatesTo : [
    { selector: '.menu > .chapters', controller: 'chapters' },
    { selector: '.menu > .features', controller: 'features' },
    { selector: '.menu > .more', controller: 'more' }
  ],
  // make the PLAY button be default highlight
  highlightedElement : '.menu > .play'
});
```

Indicates the actions that occur when users interact with elements on the page

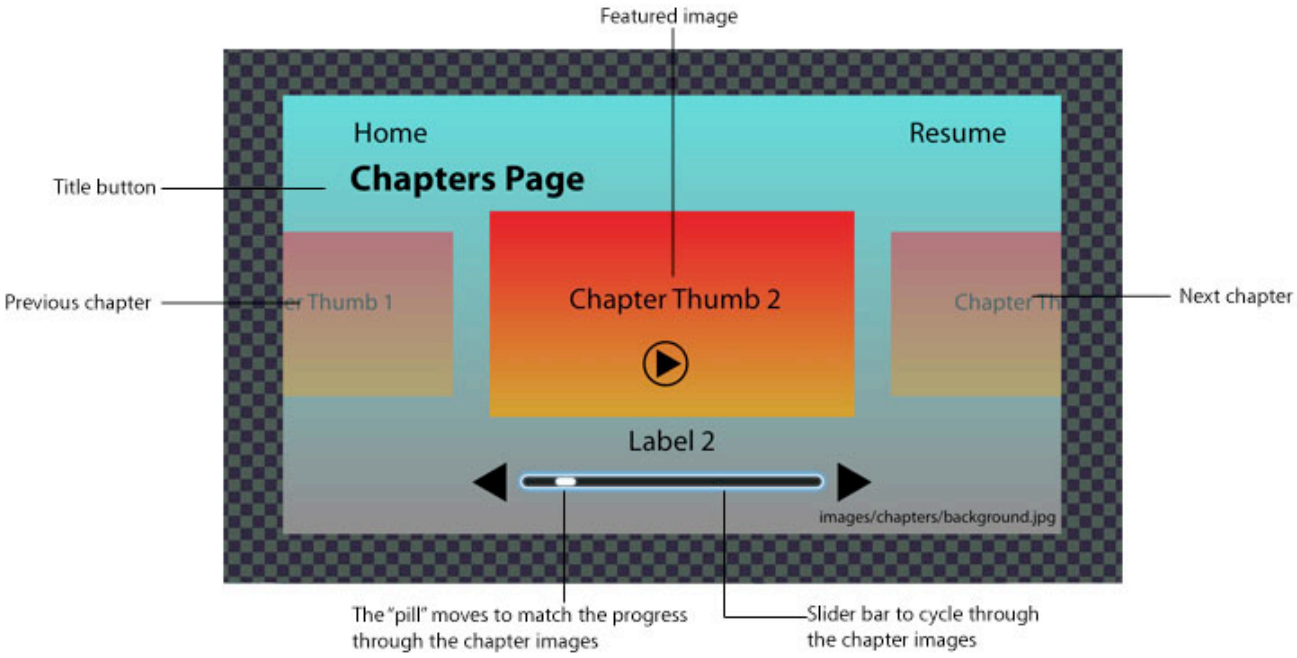
This is where you set up navigation or any specialized action within the iTunes Extras

Choose which element you want highlighted when the user opens the view (Apple TV or Emulator only)

Setting Up your Chapters View

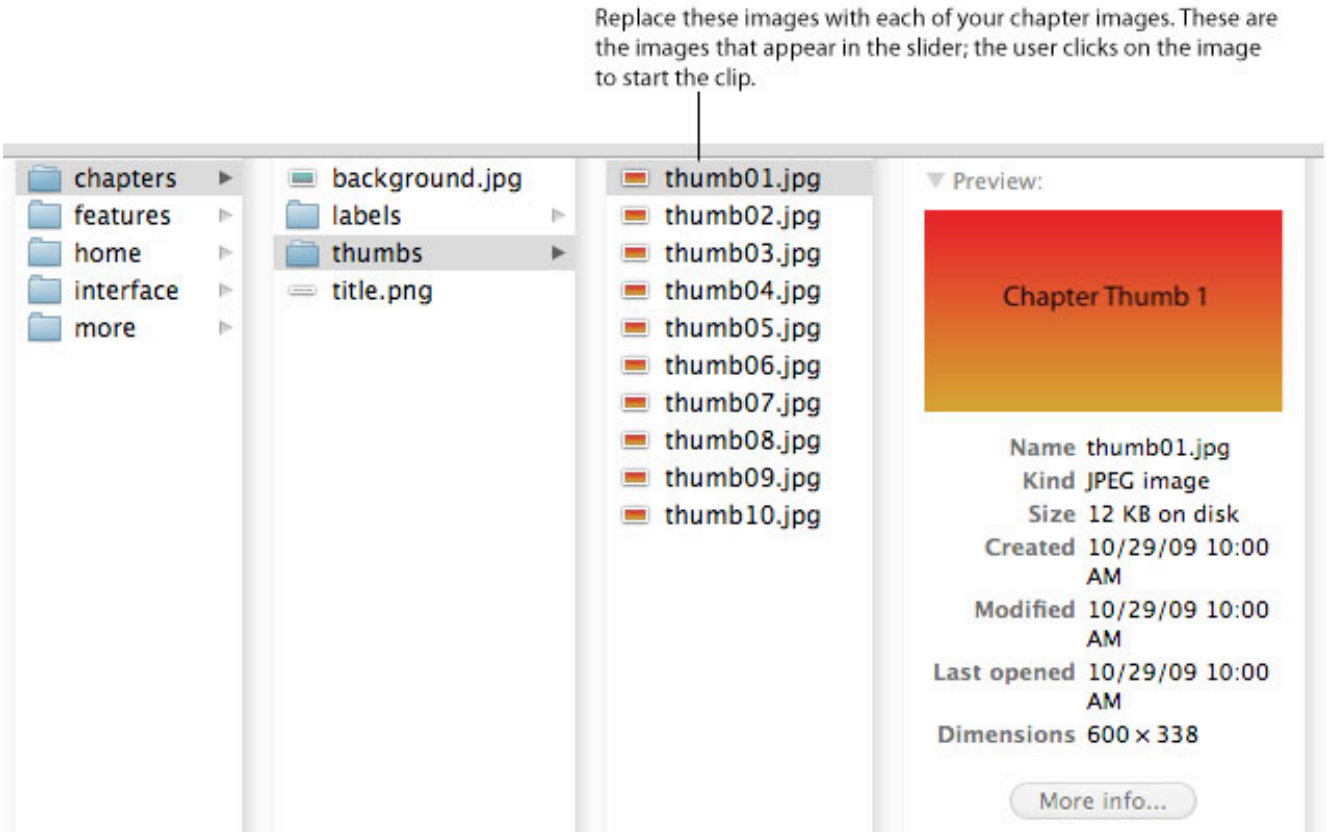
The template provides a Chapters screen that shows images and titles for the chapters in a movie, which allows the user to begin playback at a selected chapter. The Chapters view uses the TKSliderController.

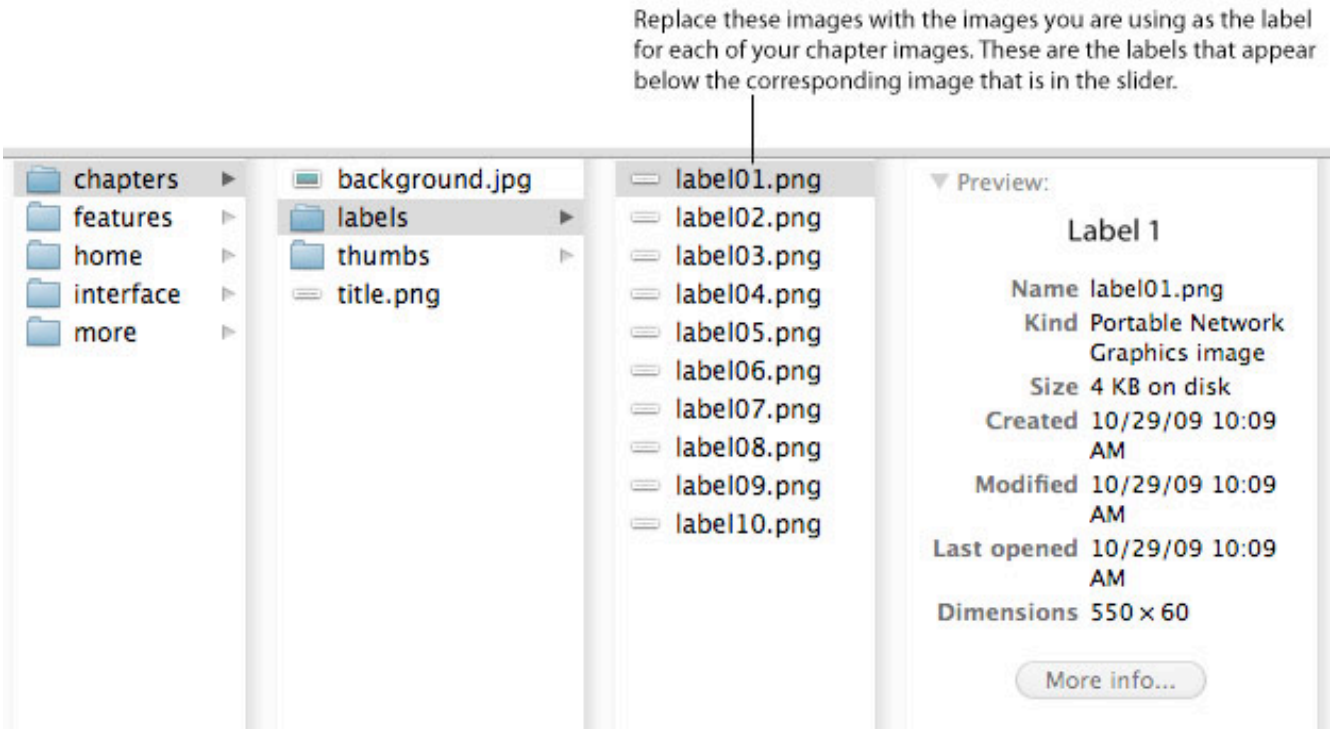
TKSliderController controls the animations as the user cycles through the chapter images



Note: The images that are not in the focus position will focus themselves when hit, so make sure that the feature image is on top of them via z-index.

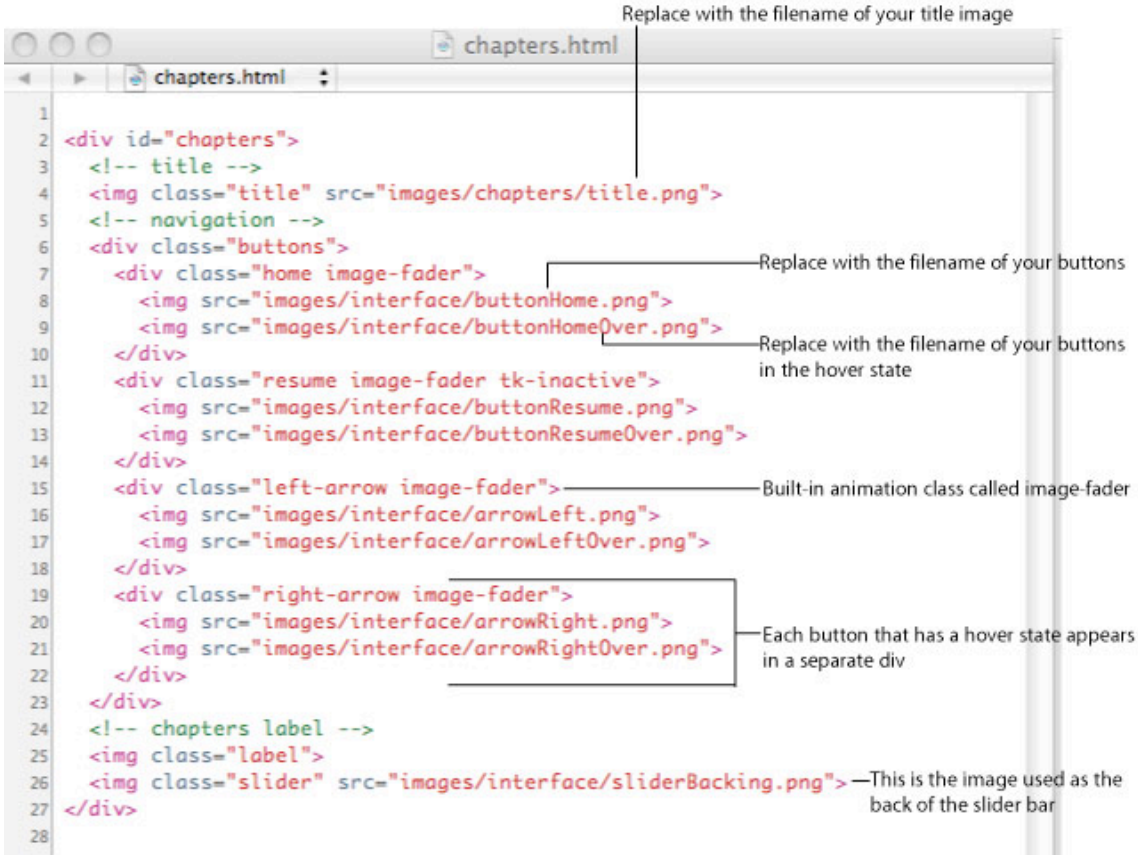
Before setting up your Chapters view, put your chapter images and labels in the `images/chapters/thumbs` and `images/chapters/labels` folders:





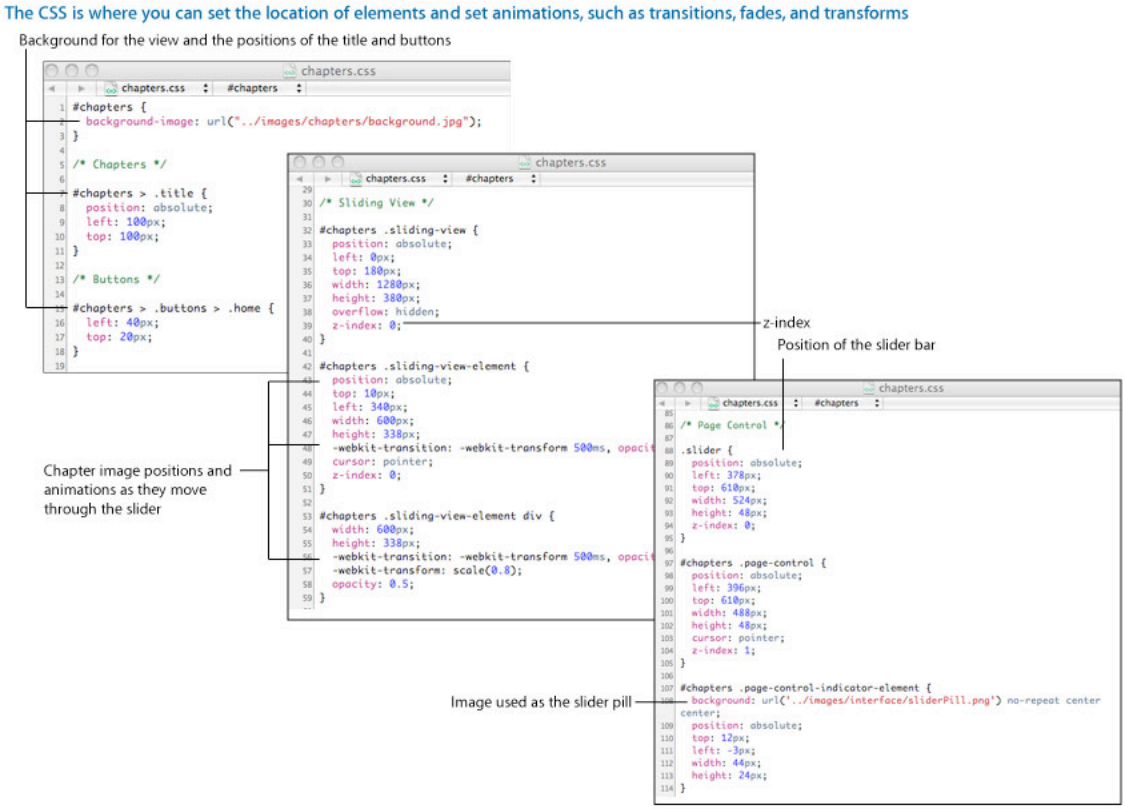
To set up your Chapters view:

1. In the Movies template folder, open the views folder, and open chapter.html with a text editor, such as TextMate.
2. Change the filename for the title image to match the filename of your image.
3. Replace the filenames of the buttons for the non-hover states with the names of the non-hover state button images you created.



4. Replace the filenames of the buttons for the hover states (ending with Over.png) with the names of the hover state button images you created.

5. In the Movies template folder, open the css folder, and open chapters.css with a text editor, such as TextMate.



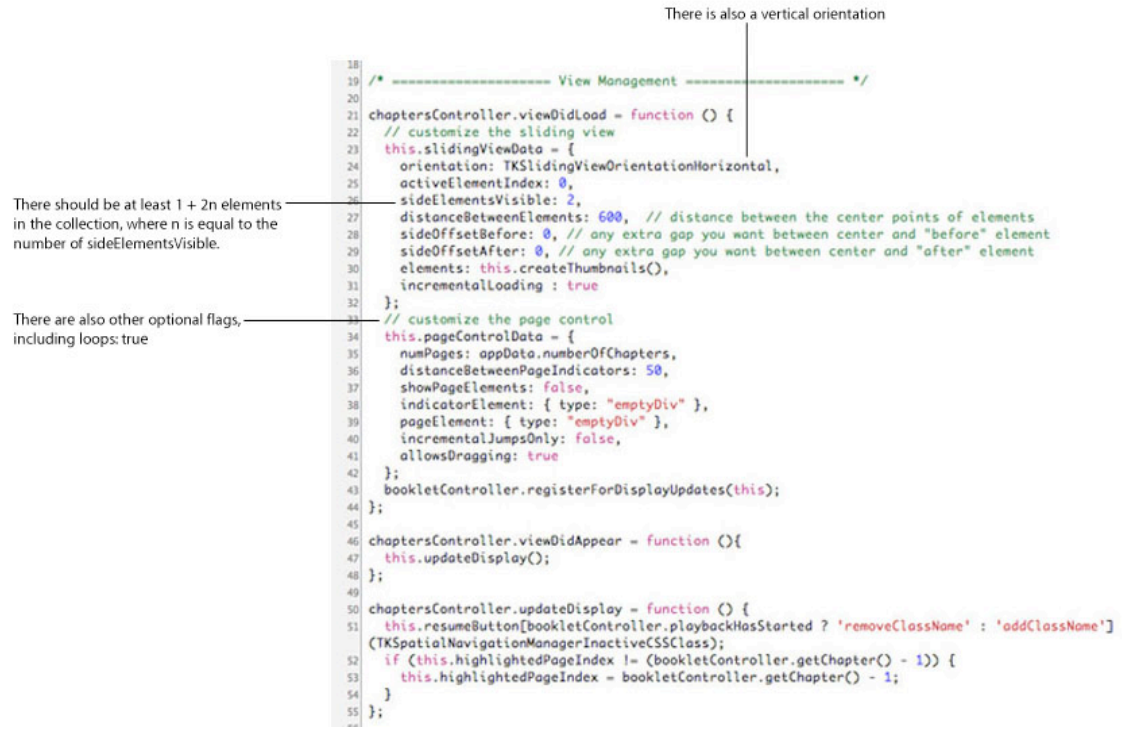
6. In the CSS file, replace the background image file with the one you are using for the Chapters view.

7. If you want to re-position any elements, change the number of pixels from the left and/or top as needed.

8. To modify the animated transitions that occur as the chapter images slide in and out of the focus view, change the values for `-webkit-transform`, `opacity`, and `scale`. The classes to use are `sliding-view-element-focused`, `sliding-view-element-before`, and `sliding-view-element-after`; you can change the opacity to create a fade, change the scale to shrink the image when not in focus, and use other webkit animations available.

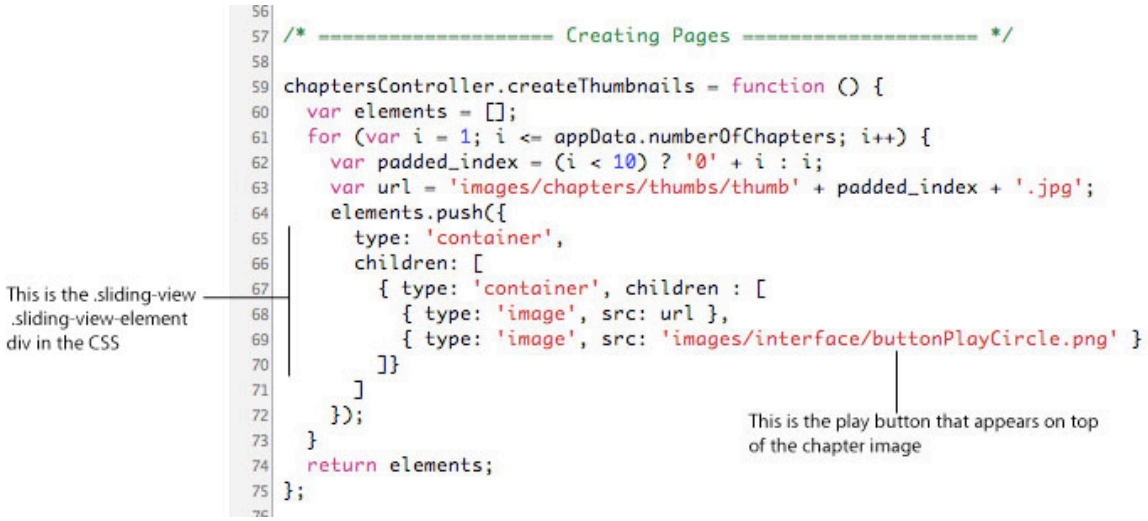
9. To see how the navigation and animations have been set up for the Chapters view, open `chapters.js`.

10. In the View Management section, you can change the distance between the center points of the chapter images and the gap between them.




```
/* ==================== View Management ==================== */

chaptersController.viewDidLoad = function () {
  // customize the sliding view
  this.slidingViewData = {
    orientation: TKSlidingViewOrientationHorizontal,
    activeElementIndex: 0,
    sideElementsVisible: 2,
    distanceBetweenElements: 600,  // distance between the center points of elements
    sideOffsetBefore: 0, // any extra gap you want between center and "before" element
    sideOffsetAfter: 0, // any extra gap you want between center and "after" element
    elements: this.createThumbnails(),
    incrementalLoading : true
  };
  // customize the page control
  this.pageControlData = {
    numPages: appData.numberOfChapters,
    distanceBetweenPageIndicators: 50,
    showPageElements: false,
    indicatorElement: { type: "emptyDiv" },
    pageElement: { type: "emptyDiv" },
    incrementalJumpsOnly: false,
    allowsDragging: true
  };
  bookletController.registerForDisplayUpdates(this);
};

chaptersController.viewDidAppear = function (){
  this.updateDisplay();
};

chaptersController.updateDisplay = function () {
  this.resumeButton[bookletController.playbackHasStarted ? 'removeClassName' : 'addClassName']
(TKSpatialNavigationManagerInactiveCSSClass);
  if (this.highlightedPageIndex != (bookletController.getChapter() - 1)) {
    this.highlightedPageIndex = bookletController.getChapter() - 1;
  }
};
```

11. The Creating Pages section includes the script to grab the chapter image thumbnails and push them in and out of the focus view. It also determines the play button that is on top of the focus chapter image.

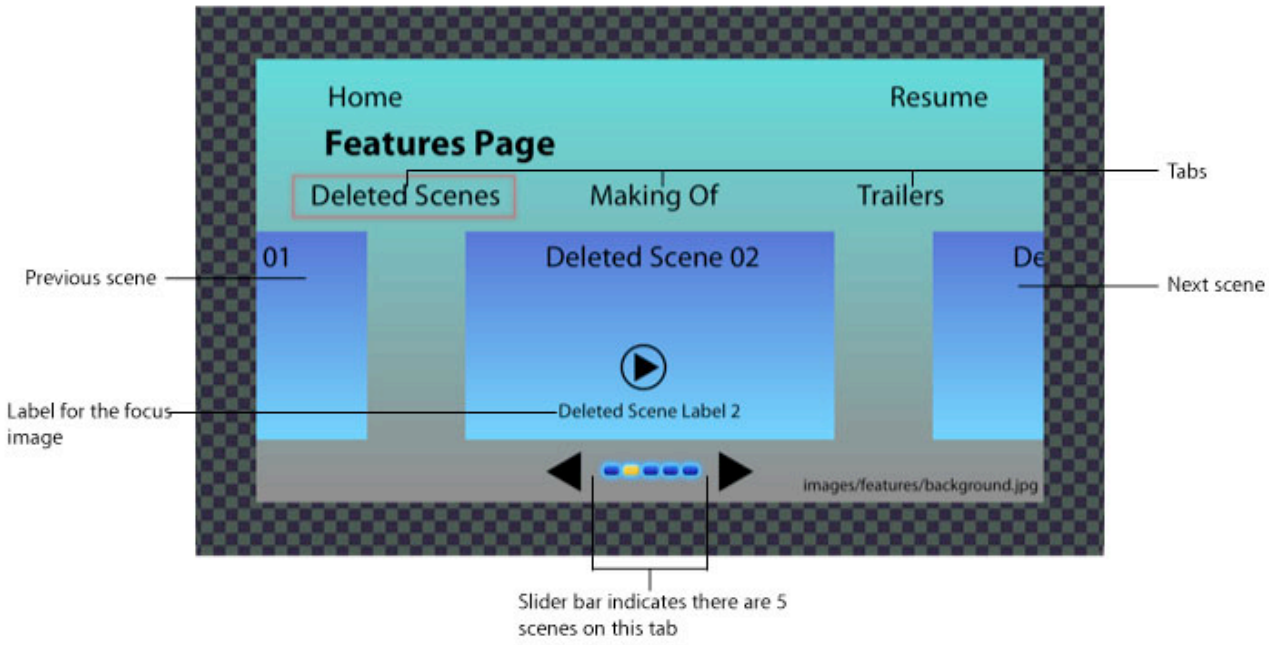



```
/* ==================== Creating Pages ==================== */

chaptersController.createThumbnails = function () {
  var elements = [];
  for (var i = 1; i <= appData.numberOfChapters; i++) {
    var padded_index = (i < 10) ? '0' + i : i;
    var url = 'images/chapters/thumbs/thumb' + padded_index + '.jpg';
    elements.push({
      type: 'container',
      children: [
        { type: 'container', children : [
          { type: 'image', src: url },
          { type: 'image', src: 'images/interface/buttonPlayCircle.png' }
        ]}
      ]
    });
  }
  return elements;
};
```

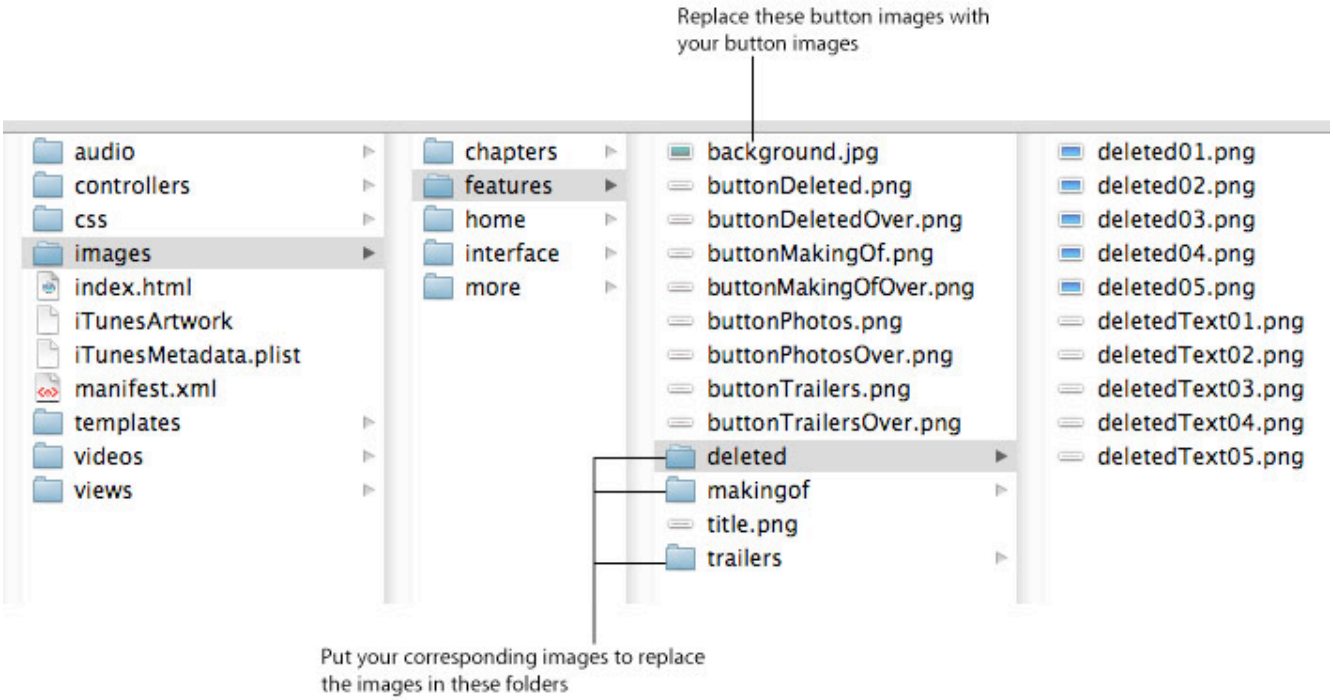
Setting Up the Features View

The template provides a Features screen that has tabs to allow the user to select among three types of bonus content: deleted scenes, making of videos, and trailers. The Features view uses the TKTabController.

TKTabController controls the tabs and what happens when the user clicks a tab

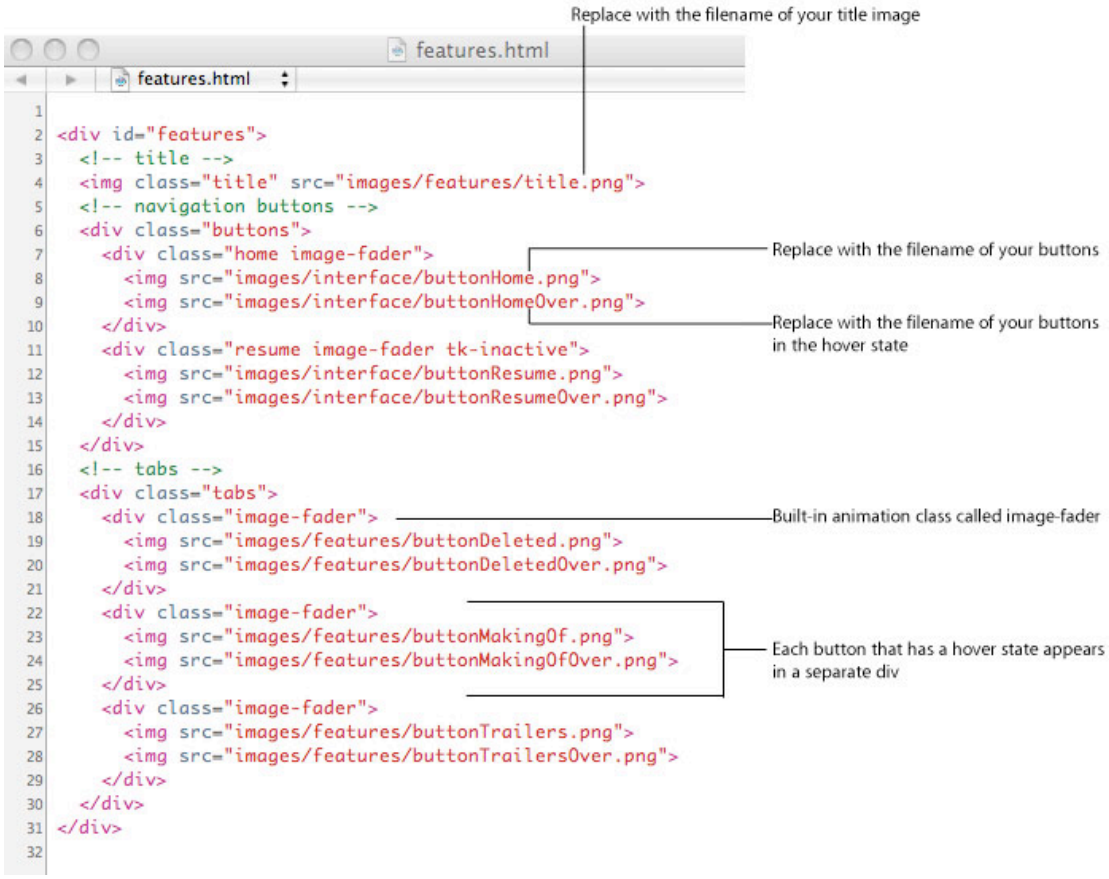


Before setting up your Features view, put your images and buttons in the appropriate `images/features` folders:



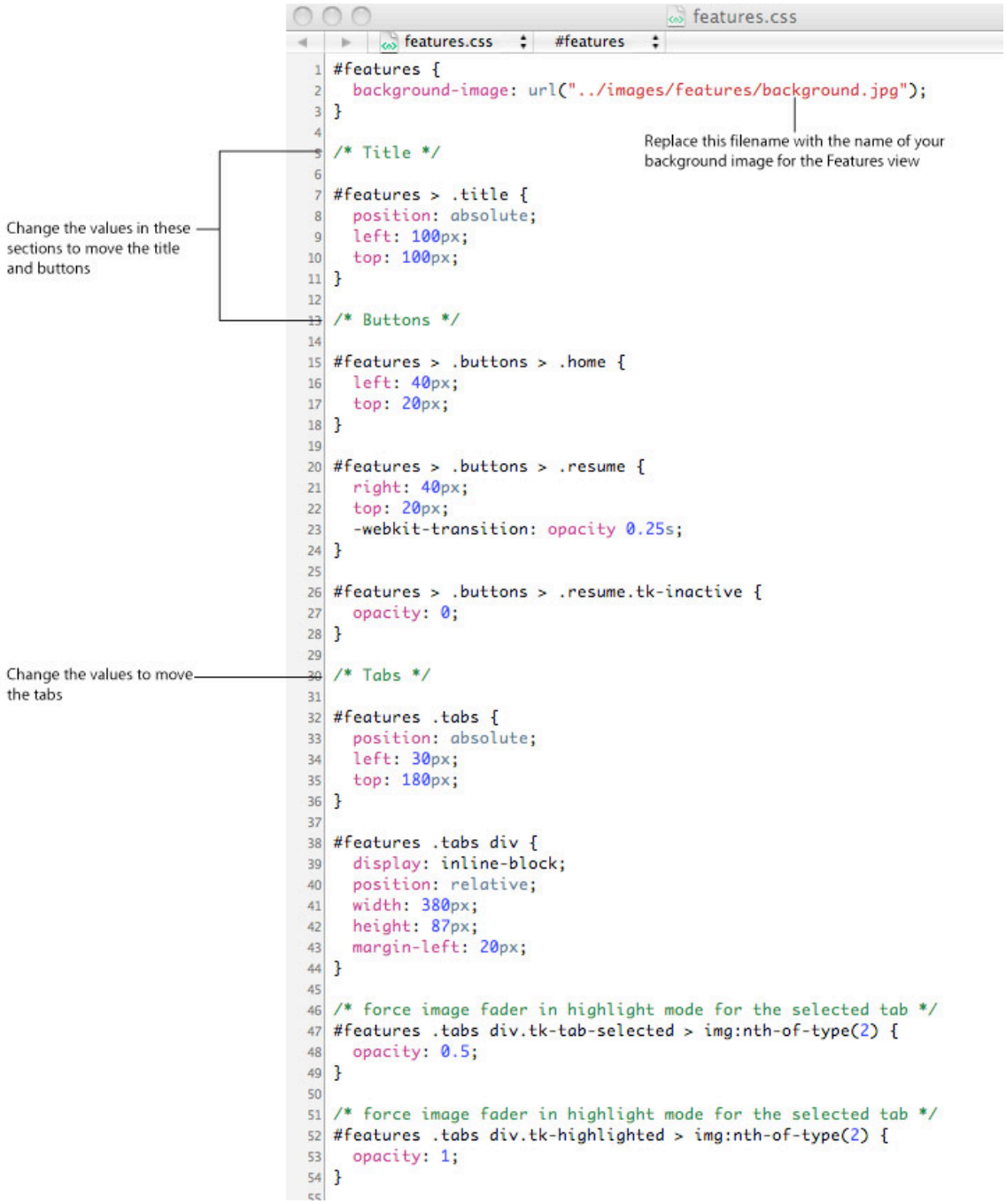
To set up your Features view:

1. In the Movies template folder, open the views folder, and open `features.html` with a text editor, such as TextMate.
2. Change the filename for the title image to match the filename of your image.
3. Replace the filenames of the buttons for the non-hover states with the names of the non-hover state button images you created.



4. Replace the filenames of the buttons for the hover states (ending with Over.png) with the names of the hover state button images you created.

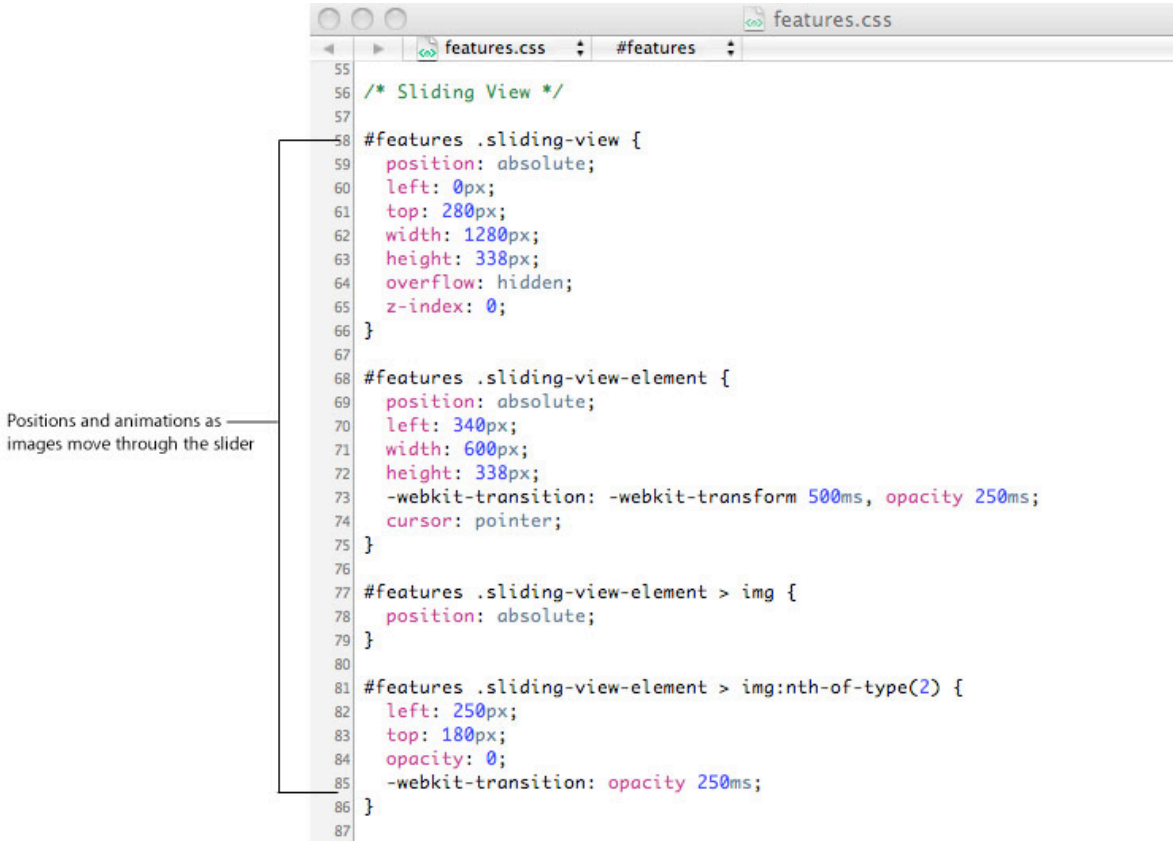
5. In the Movies template folder, open the css folder, and open `features.css` with a text editor, such as TextMate.




```
#features {
  background-image: url("../images/features/background.jpg");
}

/* Title */

#features > .title {
  position: absolute;
  left: 100px;
  top: 100px;
}

/* Buttons */

#features > .buttons > .home {
  left: 40px;
  top: 20px;
}

#features > .buttons > .resume {
  right: 40px;
  top: 20px;
  -webkit-transition: opacity 0.25s;
}

#features > .buttons > .resume.tk-inactive {
  opacity: 0;
}

/* Tabs */

#features .tabs {
  position: absolute;
  left: 30px;
  top: 180px;
}

#features .tabs div {
  display: inline-block;
  position: relative;
  width: 380px;
  height: 87px;
  margin-left: 20px;
}

/* force image fader in highlight mode for the selected tab */
#features .tabs div.tk-tab-selected > img:nth-of-type(2) {
  opacity: 0.5;
}

/* force image fader in highlight mode for the selected tab */
#features .tabs div.tk-highlighted > img:nth-of-type(2) {
  opacity: 1;
}
```

6. In the CSS file, replace the background image file with the one you are using for the Features view.

7. If you want to re-position any elements, change the number of pixels from the left and/or top as needed. Also, change the width and height to reflect your content.



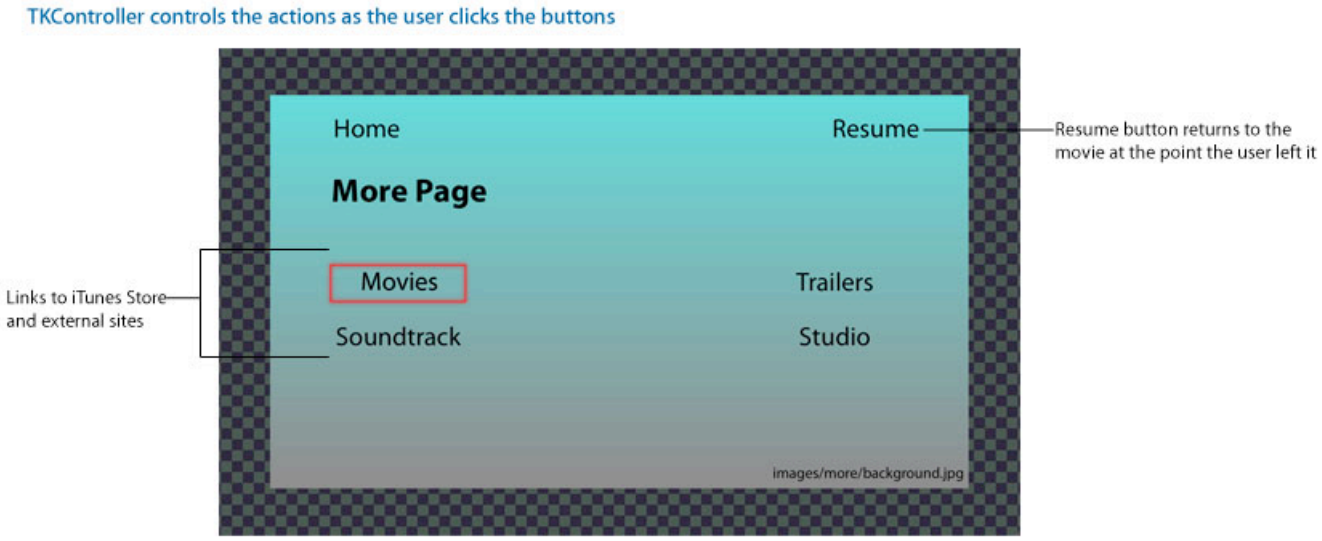

```
/* Sliding View */

#features .sliding-view {
  position: absolute;
  left: 0px;
  top: 280px;
  width: 1280px;
  height: 338px;
  overflow: hidden;
  z-index: 0;
}

#features .sliding-view-element {
  position: absolute;
  left: 340px;
  width: 600px;
  height: 338px;
  -webkit-transition: -webkit-transform 500ms, opacity 250ms;
  cursor: pointer;
}

#features .sliding-view-element > img {
  position: absolute;
}

#features .sliding-view-element > img:nth-of-type(2) {
  left: 250px;
  top: 180px;
  opacity: 0;
  -webkit-transition: opacity 250ms;
}
```

8. To modify the animated transitions that occur as the bonus video images slide in and out of the focus view, change the values for `-webkit-transform` and `opacity`. `-webkit-transform` sets the animation timing function on the element. You can specify any CSS rule to animate, such as `opacity`, but `translations` can be used in some places to give a sliding effect.

9. To see how the navigation and animations work for the Features view, open `features.js`.

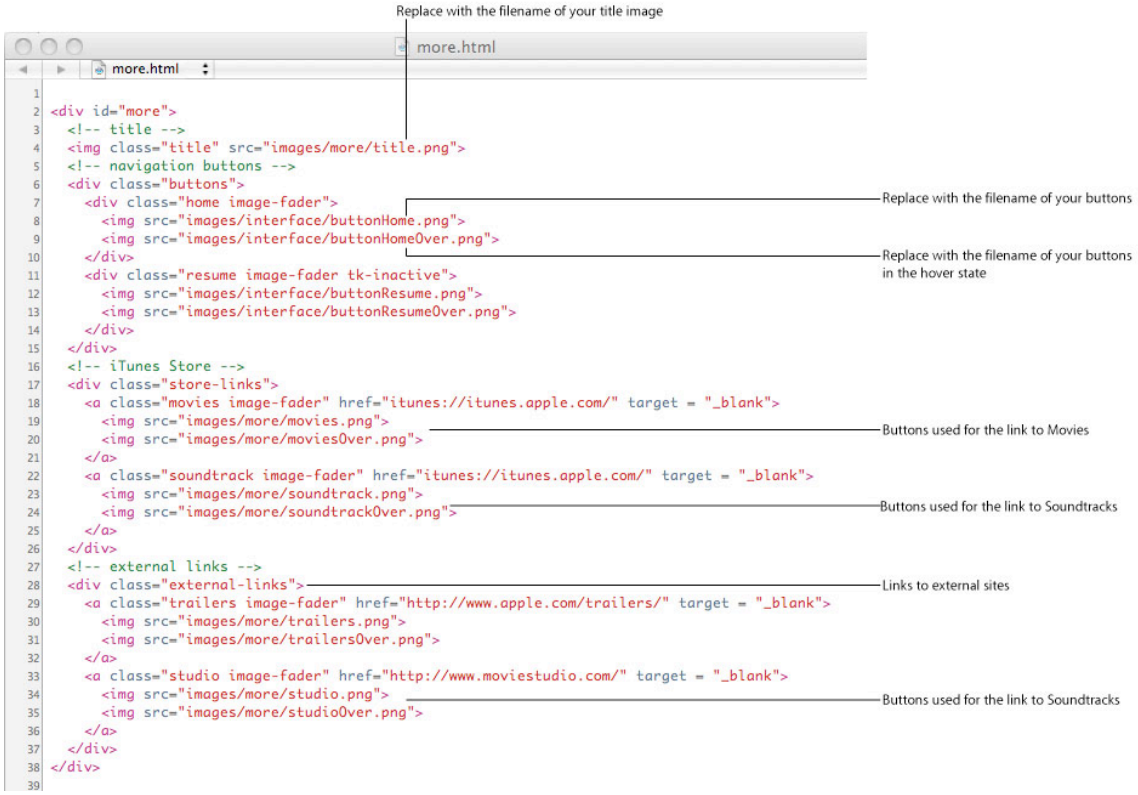
Setting Up the More View

The template provides a More screen that provides links to external resources, such as the studio homepage or iTunes albums and movies. The More view uses the basic TKController.



To set up your More page view:

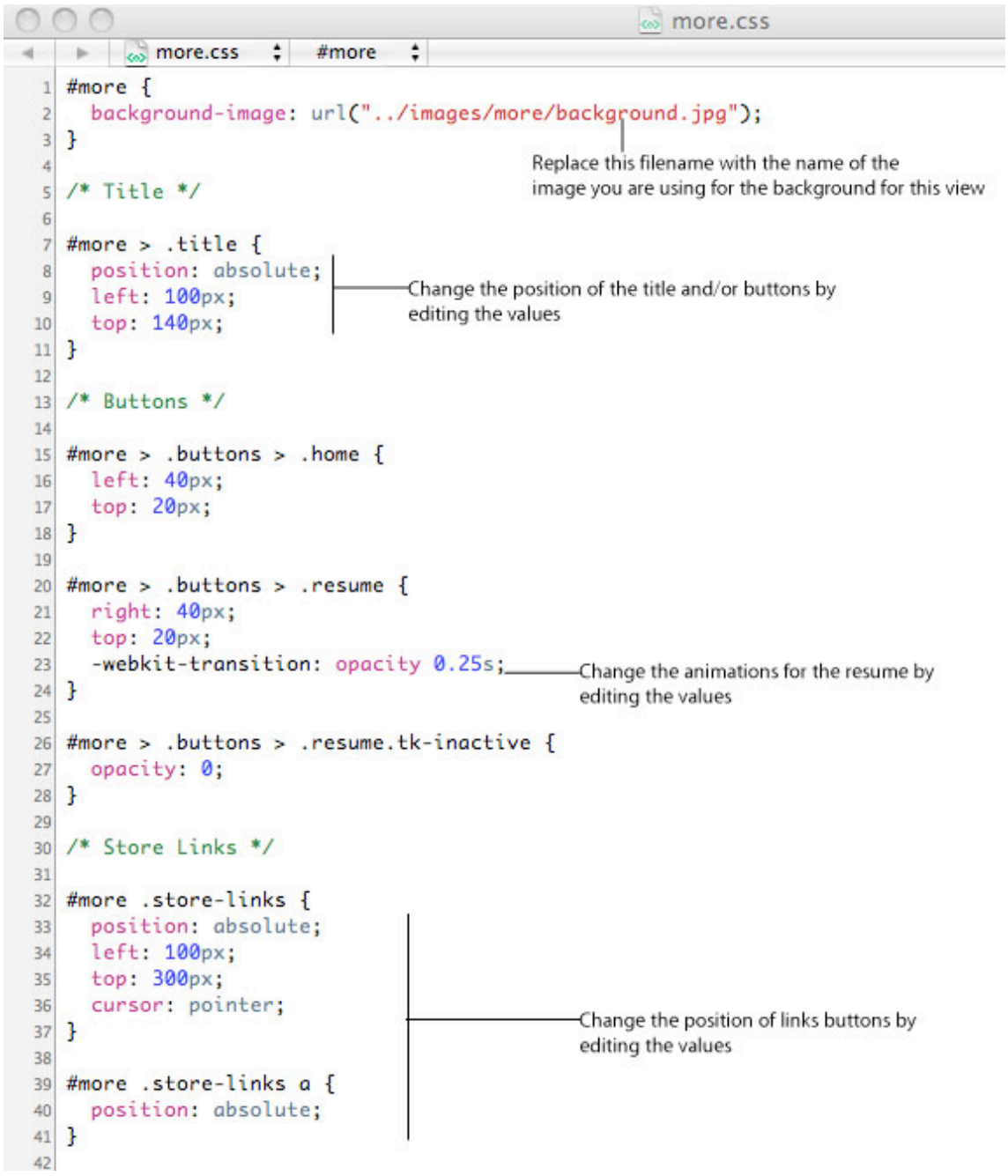
1. In the Movies template folder, open the `views` folder, and open `more.html` with a text editor, such as TextMate.
2. Change the filename for the title image to match the filename of your image.
3. Replace the filenames of the buttons for the non-hover states with the names of the non-hover state button images you created.




```
<div id="more">
  <!-- title -->
  <img class="title" src="images/more/title.png">
  <!-- navigation buttons -->
  <div class="buttons">
    <div class="home image-fader">
      <img src="images/interface/buttonHome.png">
      <img src="images/interface/buttonHomeOver.png">
    </div>
    <div class="resume image-fader tk-inactive">
      <img src="images/interface/buttonResume.png">
      <img src="images/interface/buttonResumeOver.png">
    </div>
  </div>
  <!-- iTunes Store -->
  <div class="store-links">
    <a class="movies image-fader" href="itunes://itunes.apple.com/" target = "_blank">
      <img src="images/more/movies.png">
      <img src="images/more/moviesOver.png">
    </a>
    <a class="soundtrack image-fader" href="itunes://itunes.apple.com/" target = "_blank">
      <img src="images/more/soundtrack.png">
      <img src="images/more/soundtrackOver.png">
    </a>
  </div>
  <!-- external links -->
  <div class="external-links">
    <a class="trailers image-fader" href="http://www.apple.com/trailers/" target = "_blank">
      <img src="images/more/trailers.png">
      <img src="images/more/trailersOver.png">
    </a>
    <a class="studio image-fader" href="http://www.moviestudio.com/" target = "_blank">
      <img src="images/more/studio.png">
      <img src="images/more/studioOver.png">
    </a>
  </div>
</div>
```

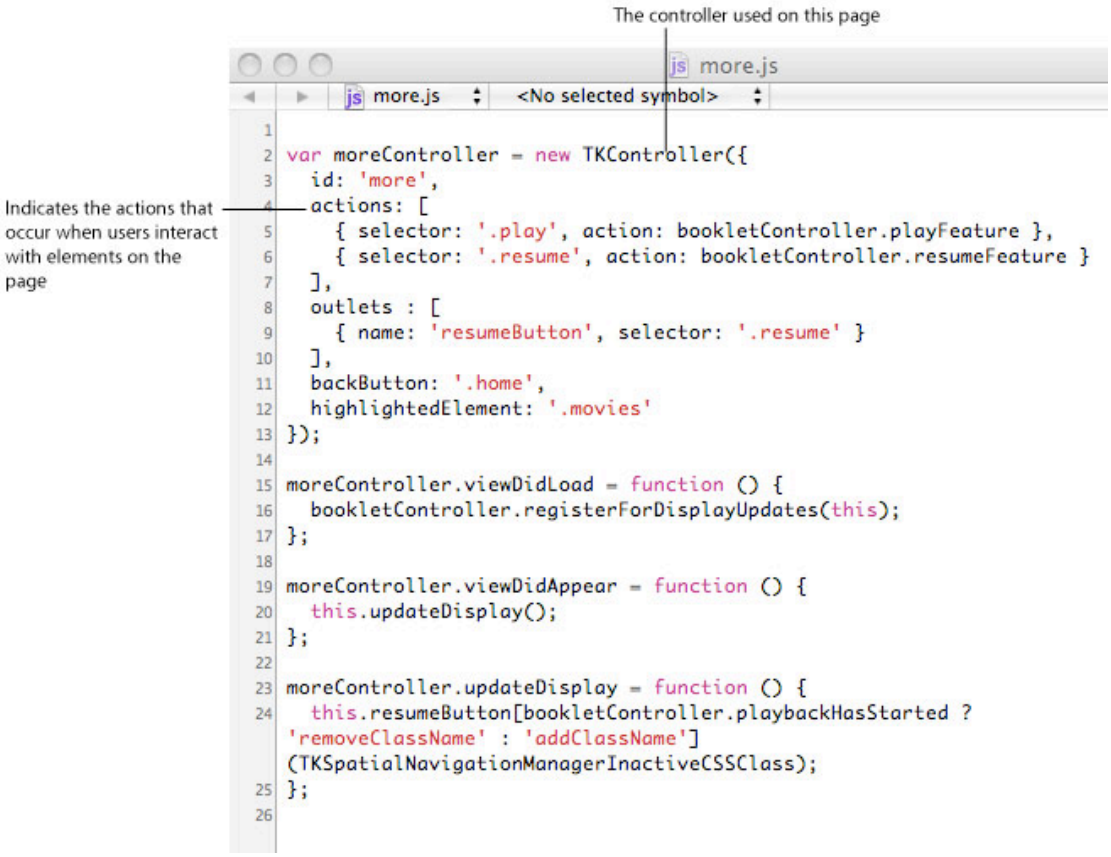
4. Replace the filenames of the buttons for the hover states (ending with Over.png) with the names of the hover state button images you created.

5. In the Movies template folder, open the `css` folder, and open `more.css` with a text editor, such as TextMate.




```
#more {
  background-image: url("../images/more/background.jpg");
}

/* Title */

#more > .title {
  position: absolute;
  left: 100px;
  top: 140px;
}

/* Buttons */

#more > .buttons > .home {
  left: 40px;
  top: 20px;
}

#more > .buttons > .resume {
  right: 40px;
  top: 20px;
  -webkit-transition: opacity 0.25s;
}

#more > .buttons > .resume.tk-inactive {
  opacity: 0;
}

/* Store Links */

#more .store-links {
  position: absolute;
  left: 100px;
  top: 300px;
  cursor: pointer;
}

#more .store-links a {
  position: absolute;
}
```

6. To see how the navigation and animations work for the More view, open `more.js`.



Setting Up the shared.css File

Take a look at the `shared.css` file to determine if there is anything you need to change. For example, the location of the bleed and scrollbar settings.

Using the iTunes LP Template

The instructions in this section take you through creating an iTunes LP using a template. The template consists of seven basic pages called views: a Home page view, a Songs view that allows the user to begin playing song tracks, a Lyrics view where the users can view the lyrics to songs, a Liner Notes view that provides background information about the album, a Videos view where users can watch bonus videos, a Photos view where the user can looks at bonus photos, and a Credit view for album credits. The steps will show you how you can substitute your content.

These are the general steps you can follow to edit the iTunes LP template. Each of these steps has more detailed instructions below.

1. Edit the `iTunesMetadata.plist` file. This specifies the metadata for iTunes LP.
2. Edit the `manifest.xml` file. This is where you specify the asset IDs. This is required for iTunes to accept drag-n-drop packages.
3. Edit the `data.js` file.
4. Create your images and place them in the correct folders. See Design Best Practices.
5. For each view (Home, Songs, Lyrics, Liner Notes, Videos, Photos, and Credits):
   - Edit the `.html` file. This is where you set up the first page of the view the user will see. It is used to navigate to other parts of your iTunes LP.
   - Edit the `.css` file. This is where you can set the location of elements and set animations, such as transitions, fades, rotations, and scaling.
   - Edit the `.js` file. This is where the basic actions and animations have been set up for the view.
6. Edit the `shared.css` file.

Editing the iTunesMetadata.plist File

The `iTunesMetadata.plist` file describes the metadata for displaying the iTunes LP in iTunes. Metadata includes things like description, genre, copyright year, artist names, and so on. This file is automatically generated at purchase time, so there's also no need to author this file except for testing purposes.

- Open the iTunesMetadata.plist file in an editor.
- Replace the placeholder metadata with the metadata for the iTunes LP.

iTunesMetadata.plist

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
   PropertyList-1.0.dtd">
3  <plist version="1.0">
4  <dict>
5          <key>Metadata</key>
6          <dict>
7                  <key>artistName</key>
8                  <string>Artist Name</string>
9                  <key>genre</key>
10                 <string>Hard Rock</string>
11                 <key>itemName</key>
12                 <string>iTunes LP - Album Name</string>
13                 <key>playlistName</key>
14                 <string>Album Name</string>
15                 <key>sort-album</key>
16                 <string>Album Name</string>
17                 <key>sort-album-status</key>
18                 <integer>128</integer>
19                 <key>sort-artist</key>
20                 <string>Artist Name</string>
21                 <key>sort-artist-status</key>
22                 <integer>128</integer>
23                 <key>sort-name</key>
24                 <string>iTunes LP - Album Name</string>
25                 <key>sort-name-status</key>
26                 <integer>128</integer>
27                 <key>year</key>
28                 <integer>2009</integer>
29         </dict>
30         <key>Name File</key>
31         <string>iTunes LP - Album.itlp</string>
32 </dict>
33 </plist>
```

Change the metadata strings to match the metadata for your iTunes LP

This will pair the album with its music. This becomes important when testing on Apple TV

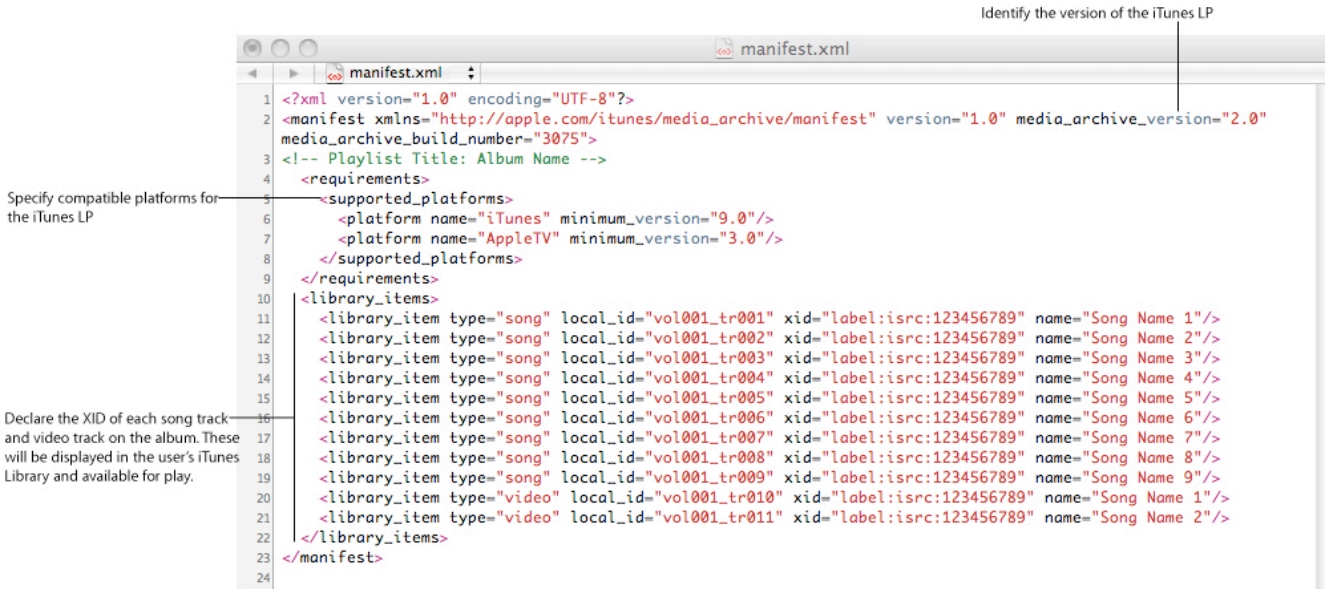
This determines the name on import

Editing the manifest.xml File

The manifest is an XML file that must live in the top level folder of the iTunes LP (the same level as the main index.html file). The purposes of the manifest are to:

- identify the version of iTunes LP
- call out what platforms iTunes LP is compatible with
- identify any items in the user's iTunes library that are playable via the iTunes LP user interface

For more detailed information on the manifest.xml format and the structure of XIDs, see the *Development Guide*.




```
<?xml version="1.0" encoding="UTF-8"?>
<manifest xmlns="http://apple.com/itunes/media_archive/manifest" version="1.0" media_archive_version="2.0"
media_archive_build_number="3075">
<!-- Playlist Title: Album Name -->
  <requirements>
    <supported_platforms>
      <platform name="iTunes" minimum_version="9.0"/>
      <platform name="AppleTV" minimum_version="3.0"/>
    </supported_platforms>
  </requirements>
  <library_items>
    <library_item type="song" local_id="vol001_tr001" xid="label:isrc:123456789" name="Song Name 1"/>
    <library_item type="song" local_id="vol001_tr002" xid="label:isrc:123456789" name="Song Name 2"/>
    <library_item type="song" local_id="vol001_tr003" xid="label:isrc:123456789" name="Song Name 3"/>
    <library_item type="song" local_id="vol001_tr004" xid="label:isrc:123456789" name="Song Name 4"/>
    <library_item type="song" local_id="vol001_tr005" xid="label:isrc:123456789" name="Song Name 5"/>
    <library_item type="song" local_id="vol001_tr006" xid="label:isrc:123456789" name="Song Name 6"/>
    <library_item type="song" local_id="vol001_tr007" xid="label:isrc:123456789" name="Song Name 7"/>
    <library_item type="song" local_id="vol001_tr008" xid="label:isrc:123456789" name="Song Name 8"/>
    <library_item type="song" local_id="vol001_tr009" xid="label:isrc:123456789" name="Song Name 9"/>
    <library_item type="video" local_id="vol001_tr010" xid="label:isrc:123456789" name="Song Name 1"/>
    <library_item type="video" local_id="vol001_tr011" xid="label:isrc:123456789" name="Song Name 2"/>
  </library_items>
</manifest>
```

Storing the Background Audio and Bleed image

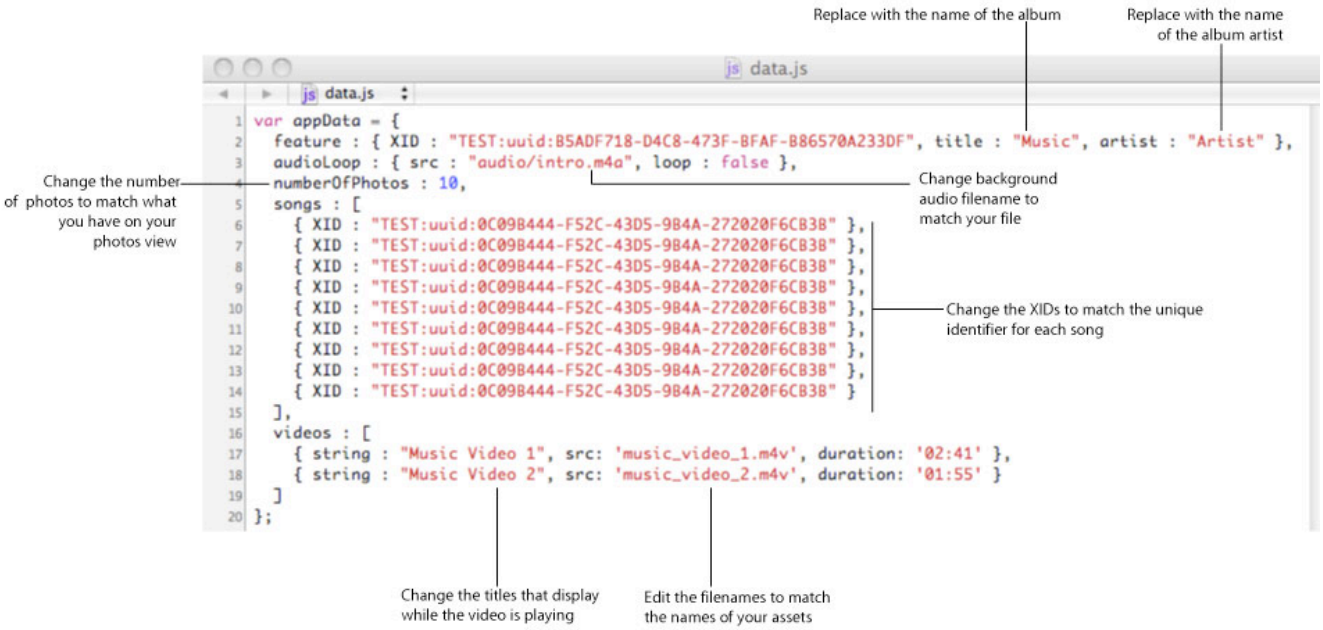
Once you have created the background audio and bleed image, place them in the appropriate folders. The background audio is an audio clip that is played when a user opens an iTunes LP and iTunes Extras. You can have the audio clip play once or repeat over and over.

- Put your background audio file in the audio folder. In `data.js`, change the filename to match the name of your audio file.
- Replace `images/interface/bleed.png` with your image. If you are changing the filename or file type (to JPG), make sure you also change the reference to the file in `shared.css`. To change the location of the bleed, edit the `shared.css` file.

Editing the data.js File

The `data.js` file sets constant variables, such as the number of photos, the background audio, locations of the bonus content, such as artist interviews. It also provides the unique identifiers for each song track and video track on the album. Before opening the template, edit the `data.js` file. Open it in a text editor, such as TextMate.

- In `data.js`, change the filename of the background audio to match the name of your audio file. This is where you also specify whether or not the audio should repeat by setting "loop" to "true" or "false."
- Identify the XIDs for the entire album as well as each song and video track.

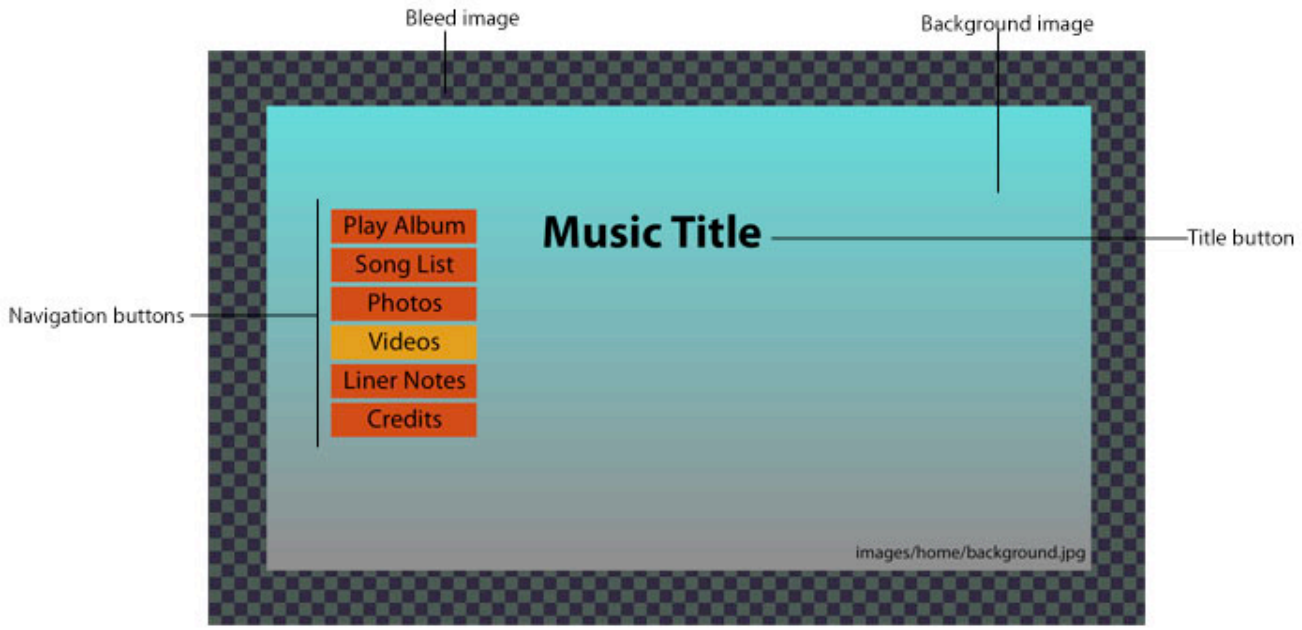



```
var appData = {
  feature : { XID : "TEST:uuid:B5ADF718-D4C8-473F-BFAF-B86570A233DF", title : "Music", artist : "Artist" },
  audioLoop : { src : "audio/intro.m4a", loop : false },
  numberOfPhotos : 10,
  songs : [
    { XID : "TEST:uuid:0C09B444-F52C-43D5-9B4A-272020F6CB3B" },
    { XID : "TEST:uuid:0C09B444-F52C-43D5-9B4A-272020F6CB3B" },
    { XID : "TEST:uuid:0C09B444-F52C-43D5-9B4A-272020F6CB3B" },
    { XID : "TEST:uuid:0C09B444-F52C-43D5-9B4A-272020F6CB3B" },
    { XID : "TEST:uuid:0C09B444-F52C-43D5-9B4A-272020F6CB3B" },
    { XID : "TEST:uuid:0C09B444-F52C-43D5-9B4A-272020F6CB3B" },
    { XID : "TEST:uuid:0C09B444-F52C-43D5-9B4A-272020F6CB3B" },
    { XID : "TEST:uuid:0C09B444-F52C-43D5-9B4A-272020F6CB3B" },
    { XID : "TEST:uuid:0C09B444-F52C-43D5-9B4A-272020F6CB3B" }
  ],
  videos : [
    { string : "Music Video 1", src: 'music_video_1.m4v', duration: '02:41' },
    { string : "Music Video 2", src: 'music_video_2.m4v', duration: '01:55' }
  ]
};
```

Changing the Home Page

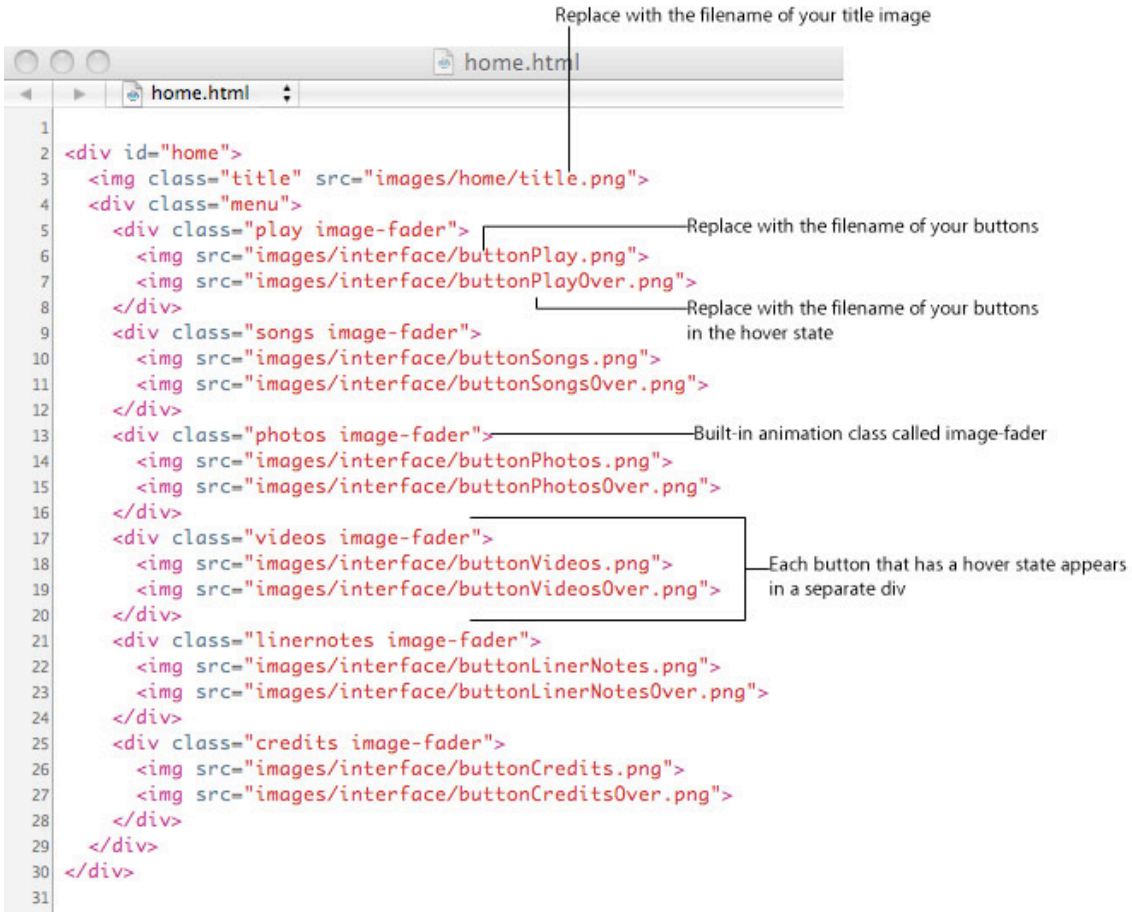
On the sample Home page template, the album title and navigation buttons are actually images. You can use the images supplied with the template, but you can easily substitute your own images. The Home view uses the basic TKController.

TKController controls the actions as the user clicks the buttons



To set up your Home page view:

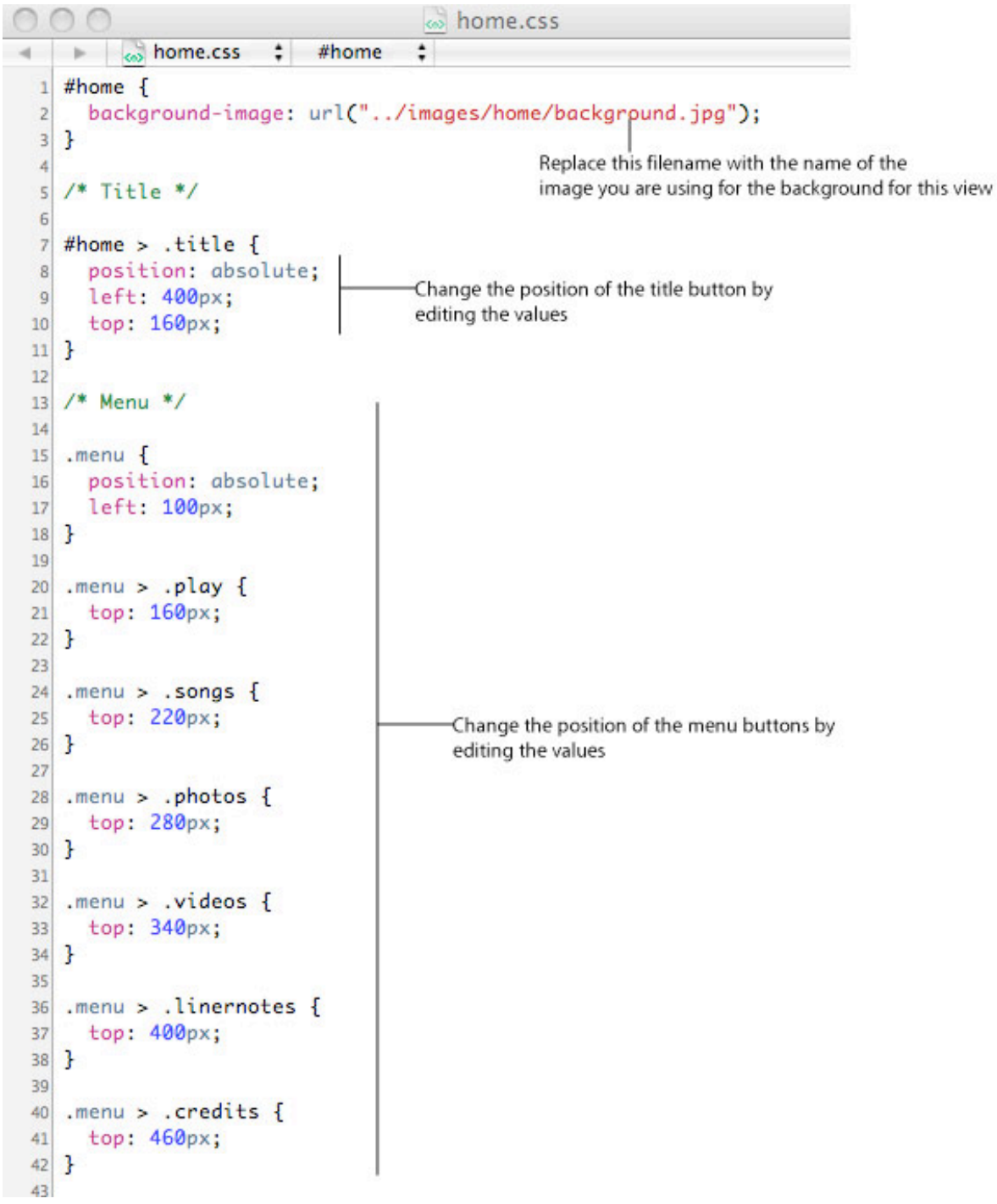
1. In the Music template folder, open the `views` folder, and open `home.html` with a text editor, such as TextMate.
2. Change the filename for the title image to match the filename of your image.
3. Replace the filenames of the buttons for the non-hover states with the names of the non-hover state button images you created.




```
1
2  <div id="home">
3    <img class="title" src="images/home/title.png">
4    <div class="menu">
5      <div class="play image-fader">
6        <img src="images/interface/buttonPlay.png">
7        <img src="images/interface/buttonPlayOver.png">
8      </div>
9      <div class="songs image-fader">
10       <img src="images/interface/buttonSongs.png">
11       <img src="images/interface/buttonSongsOver.png">
12     </div>
13     <div class="photos image-fader">
14       <img src="images/interface/buttonPhotos.png">
15       <img src="images/interface/buttonPhotosOver.png">
16     </div>
17     <div class="videos image-fader">
18       <img src="images/interface/buttonVideos.png">
19       <img src="images/interface/buttonVideosOver.png">
20     </div>
21     <div class="linernotes image-fader">
22       <img src="images/interface/buttonLinerNotes.png">
23       <img src="images/interface/buttonLinerNotesOver.png">
24     </div>
25     <div class="credits image-fader">
26       <img src="images/interface/buttonCredits.png">
27       <img src="images/interface/buttonCreditsOver.png">
28     </div>
29   </div>
30 </div>
31
```

4. Replace the filenames of the buttons for the hover states (ending with Over.png) with the names of the hover state button images you created.

5. In the Music template folder, open the `css` folder, and open `home.css` with a text editor, such as TextMate.



6. To see how the navigation and animations work for the Home view, open `home.js`.


```
var albumHelper = {};

albumHelper.playAlbum = function() {
  var playlist = bookletController.buildPlaylist(appData.songs);
  playlist.play();
};

var homeController = new TKController({
  id: 'home',
  actions : [
    { selector: '.menu > .play', action: albumHelper.playAlbum }
  ],
  navigatesTo : [
    { selector: '.menu > .songs', controller: 'songs' },
    { selector: '.menu > .photos', controller: 'photos' },
    { selector: '.menu > .videos', controller: 'videos' },
    { selector: '.menu > .linernotes', controller: 'linernotes' },
    { selector: '.menu > .credits', controller: 'credits' }
  ],
  // make the PLAY button be default highlight
  highlightedElement : '.menu > .play'
});
```

Setting Up your Song List View

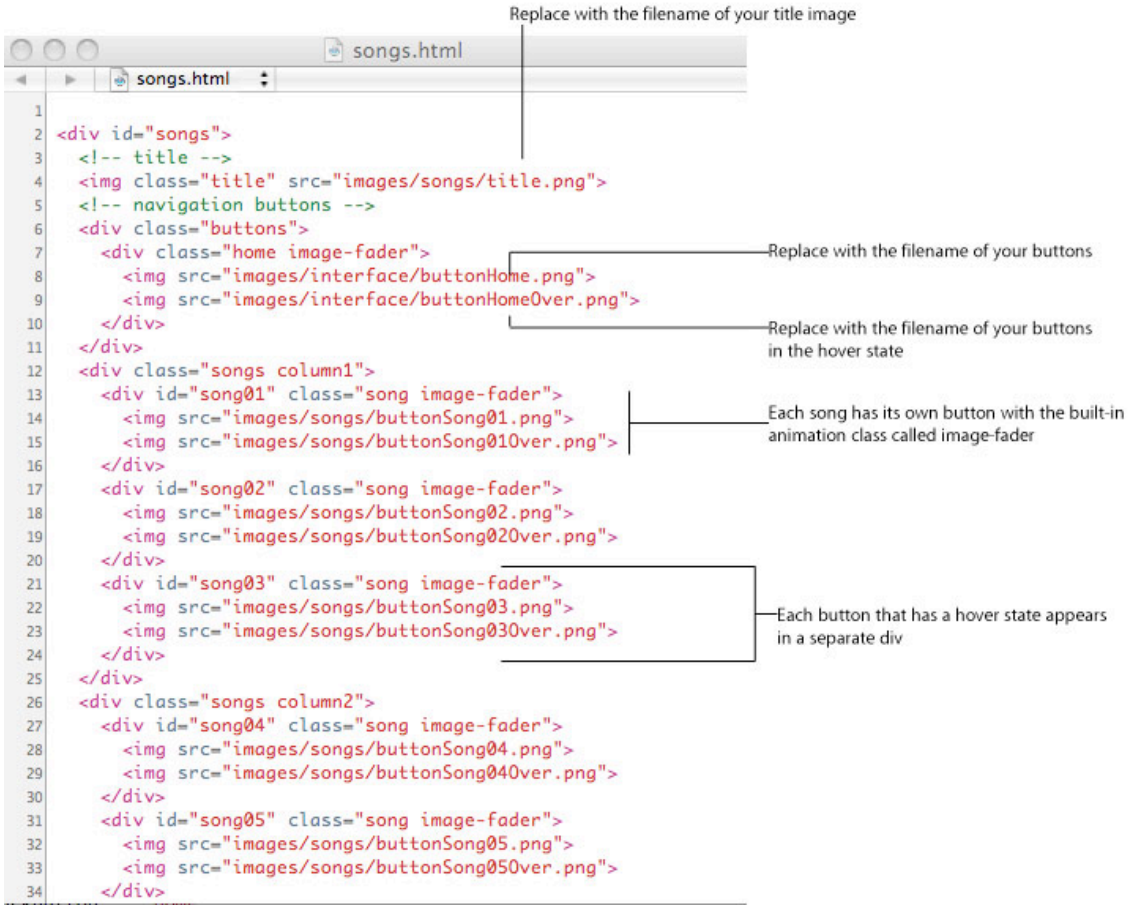
The template provides a Song List screen that shows a list of song titles, that allows the user to play an individual song. The Song List view uses the TKController.

TKController controls the actions as the user clicks the buttons

To set up your Songs List view:

1. In the Music template folder, open the `views` folder, and open `songs.html` with a text editor, such as TextMate.
2. Change the filename for the title image to match the filename of your image.

3. Replace the filenames of the buttons for the non-hover states with the names of the non-hover state button images you created.




```
<div id="songs">
  <!-- title -->
  <img class="title" src="images/songs/title.png">
  <!-- navigation buttons -->
  <div class="buttons">
    <div class="home image-fader">
      <img src="images/interface/buttonHome.png">
      <img src="images/interface/buttonHomeOver.png">
    </div>
  </div>
  <div class="songs column1">
    <div id="song01" class="song image-fader">
      <img src="images/songs/buttonSong01.png">
      <img src="images/songs/buttonSong01Over.png">
    </div>
    <div id="song02" class="song image-fader">
      <img src="images/songs/buttonSong02.png">
      <img src="images/songs/buttonSong02Over.png">
    </div>
    <div id="song03" class="song image-fader">
      <img src="images/songs/buttonSong03.png">
      <img src="images/songs/buttonSong03Over.png">
    </div>
  </div>
  <div class="songs column2">
    <div id="song04" class="song image-fader">
      <img src="images/songs/buttonSong04.png">
      <img src="images/songs/buttonSong04Over.png">
    </div>
    <div id="song05" class="song image-fader">
      <img src="images/songs/buttonSong05.png">
      <img src="images/songs/buttonSong05Over.png">
    </div>
```

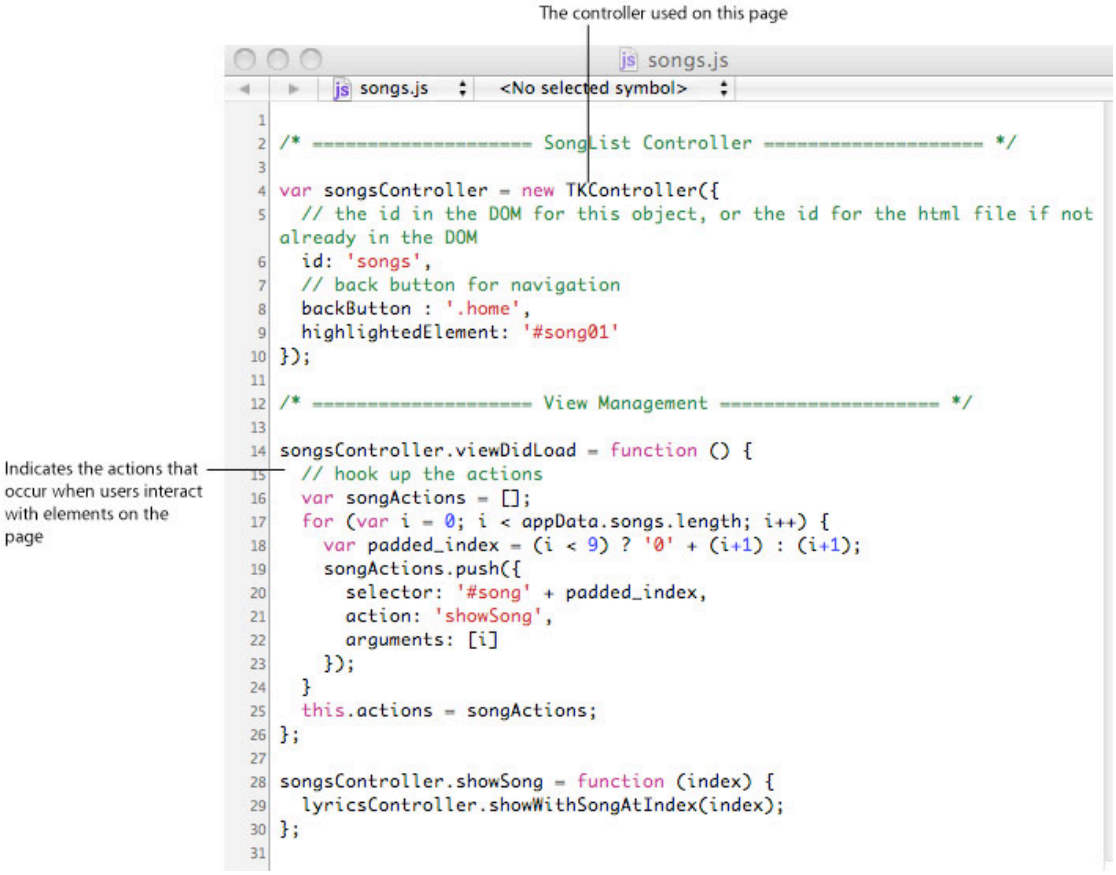
4. Replace the filenames of the buttons for the hover states (ending with Over.png) with the names of the hover state button images you created.

5. In the Music template folder, open the `css` folder, and open `songs.css` with a text editor, such as TextMate.

6. In the CSS file, replace the background image file with the one you are using for the Songs List view.

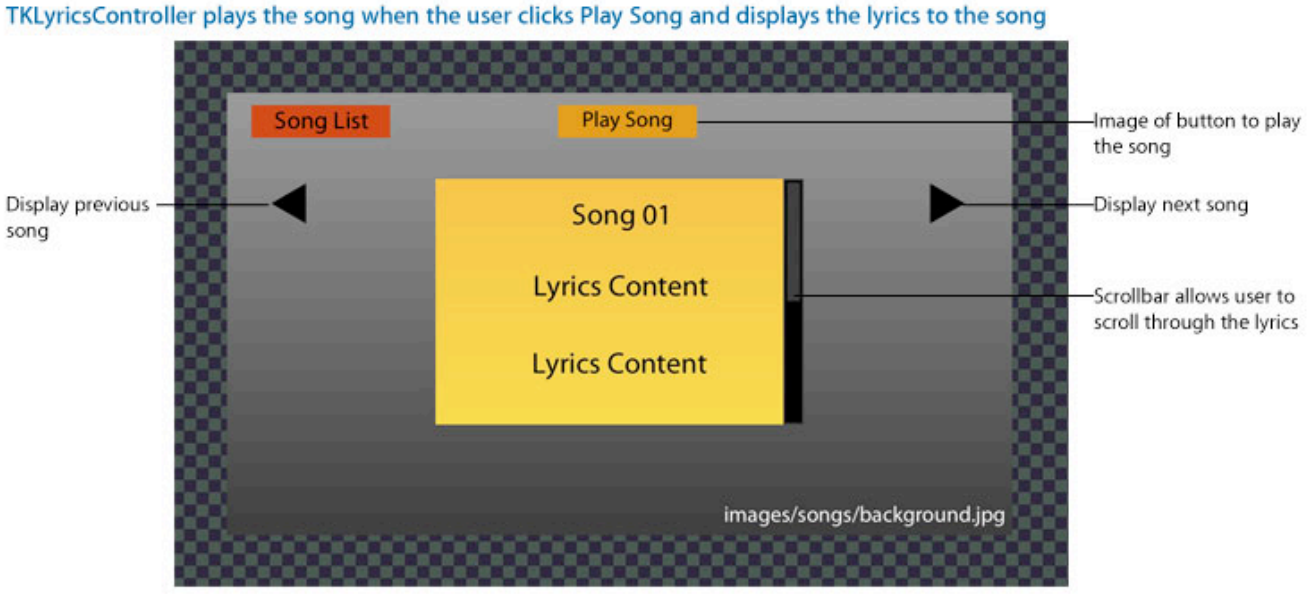
7. If you want to re-position any elements, change the number of pixels from the left and/or top as needed.

8. To see how the navigation and animations have been set up for the Songs List view, open `songs.js`.

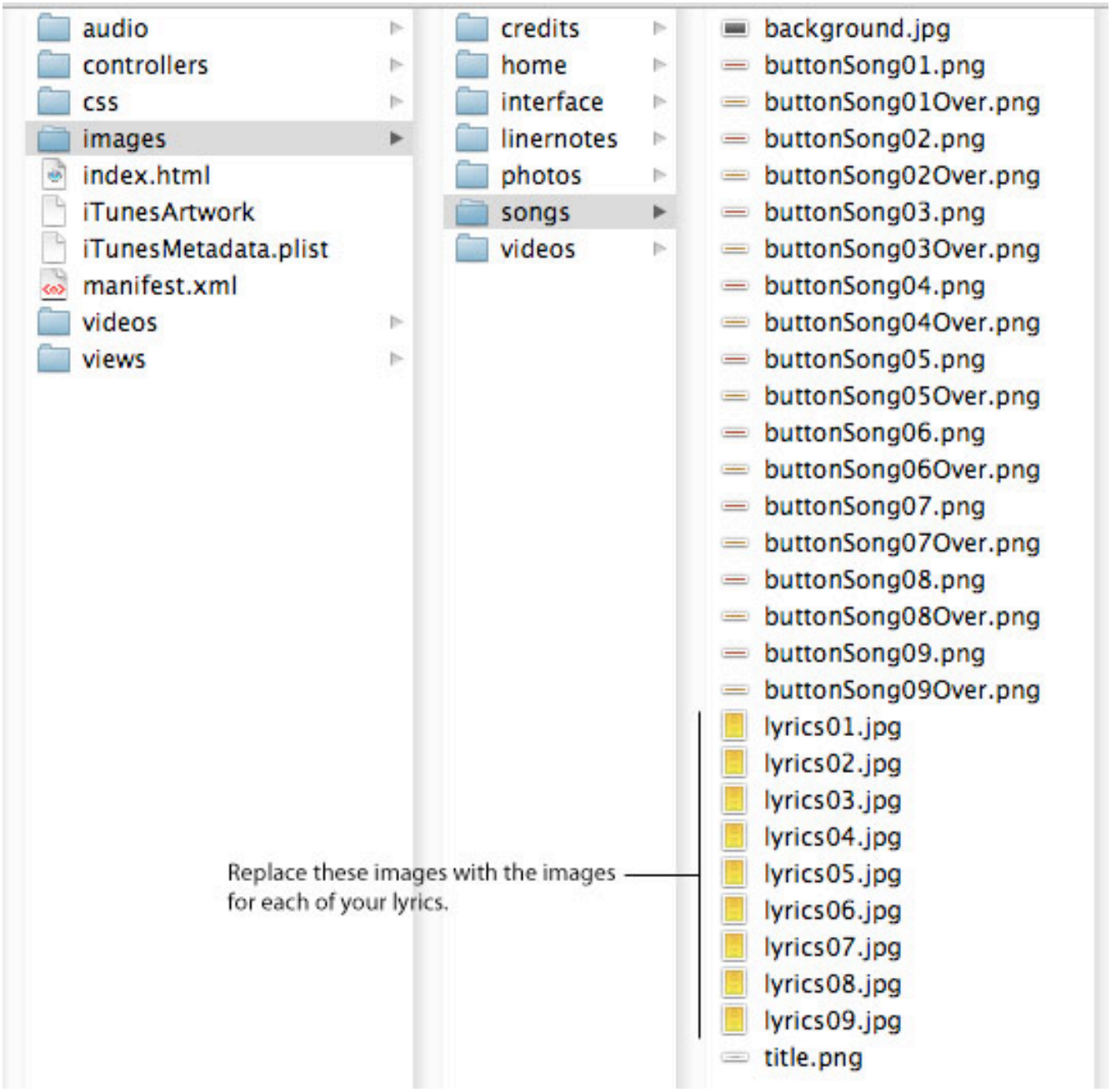


Setting Up your Lyrics View

The template provides a Lyrics screen that displays the lyrics to the currently selected song and a button to play the song. For each song, create an image of the lyrics. See *Design Best Practices*. The Lyrics view uses the TKLyricsController.

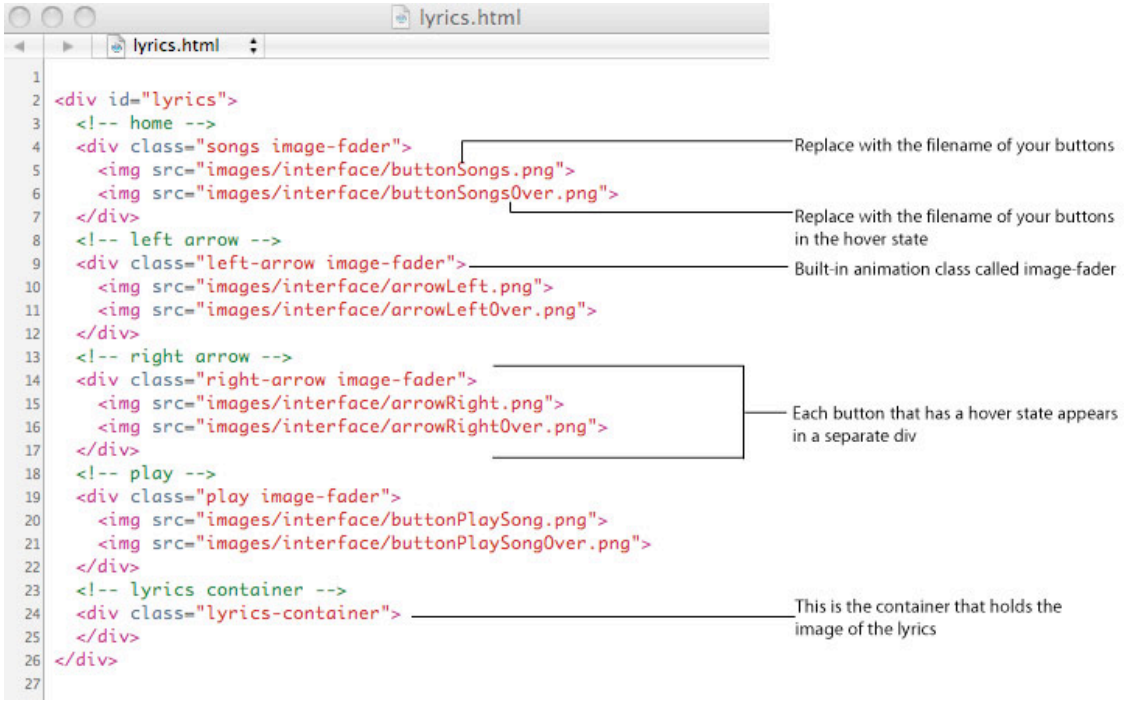


Before setting up your Lyrics view, put your lyrics images and labels in the `images/songs` folder:



To set up your Lyrics view:

1. In the Music template folder, open the `views` folder, and open `lyrics.html` with a text editor, such as TextMate.
2. Change the filename for the title image to match the filename of your image.
3. Replace the filenames of the buttons for the non-hover states with the names of the non-hover state button images you created.



4. Replace the filenames of the buttons for the hover states (ending with Over.png) with the names of the hover state button images you created.

5. In the Music template folder, open the `css` folder, and open `lyrics.css` with a text editor, such as TextMate.

The CSS is where you can set the location of elements

Background for the view and the positions of the title and buttons

lyrics.css

lyrics.css <No selected symbol>

```
1  #lyrics {
2    background-image: url("../images/songs/background.jpg");
3  }
4
5  #lyrics .songs {
6    left: 40px;
7    top: 20px;
8  }
9
10 /* arrows */
11
12 #lyrics .left-arrow {
13   left: 61px;
14   top: 140px;
15   cursor: pointer;
16 }
17
18 #lyrics .right-arrow {
19   left: 1134px;
20   top: 140px;
21   cursor: pointer;
22 }
23
24 /* play */
25
26 #lyrics .play {
27   left: 540px;
28   top: 20px;
29 }
30
31 #lyrics .lyrics-container {
32   position: absolute;
33   top: 140px;
34   left: 340px;
35   width: 600px;
36   height: 400px;
37   overflow-x: hidden;
38   overflow-y: scroll;
39 }
```

Position of the lyrics container

6. In the CSS file, replace the background image file with the one you are using for the Lyrics view.

7. If you want to re-position any elements, change the number of pixels from the left and/or top as needed.

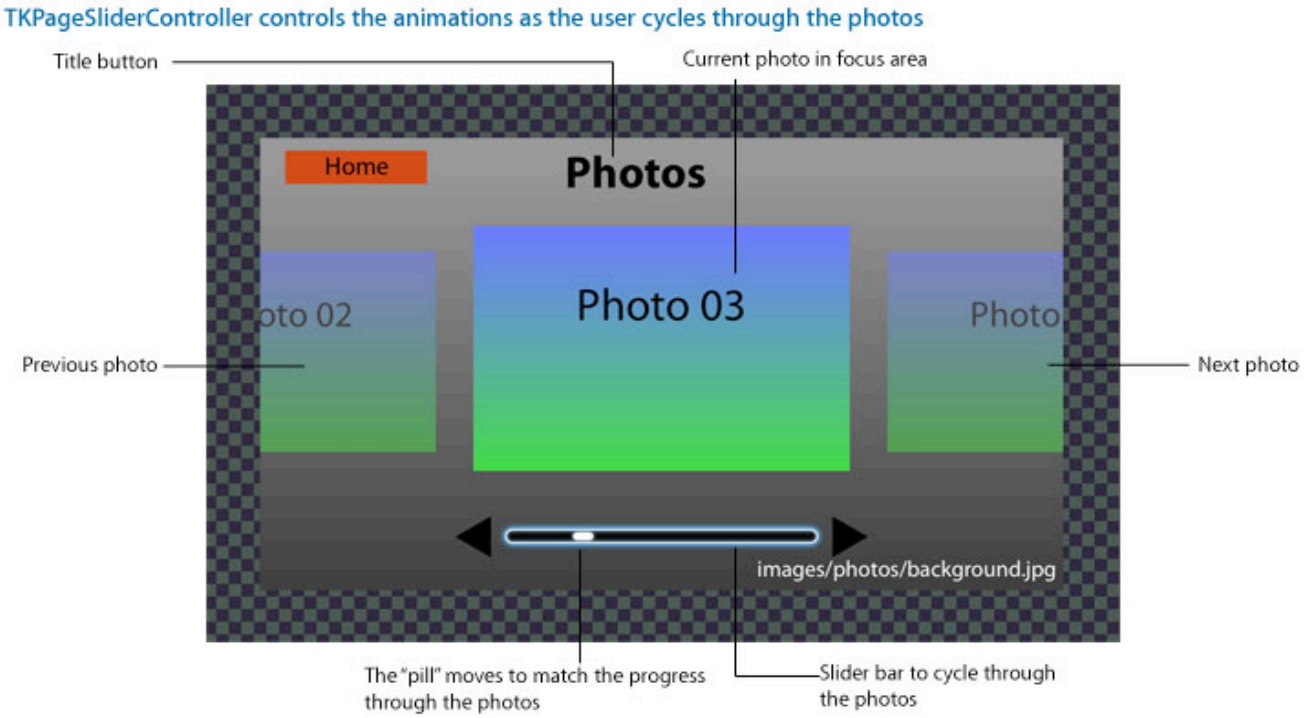
8. To see how the navigation and animations have been set up for the Lyrics view, open `lyrics.js`.

lyrics.js

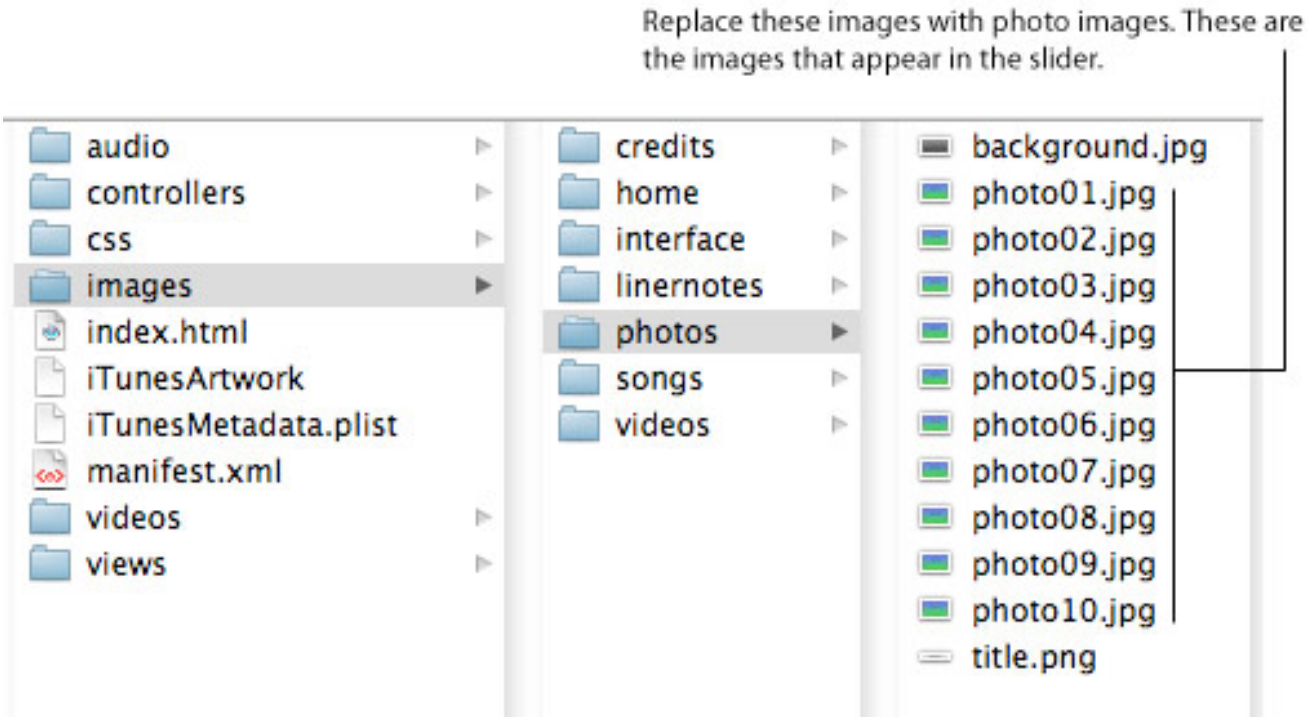
```
/* ==================== Lyrics Controller ==================== */

var lyricsController = new TKLyricsController({
  id: 'lyrics',
  actions : [
    { selector: '.play', action: 'playCurrentSong' }
  ],
  outlets : [
    { name: 'container', selector: '.lyrics-container' }
  ],
  backButton: '.songs',
  highlightedElement: '.play',
  scrollableElement : '.lyrics-container',
  numberOfSongs: appData.songs.length,
  previousSongButton: '.left-arrow',
  nextSongButton: '.right-arrow'
});

/* ==================== Creating Pages ==================== */

lyricsController.songDidChange = function (songIndex) {
  // clean up the container and reset scroll offset
  this.container.textContent = '';
  this.container.scrollTop = 0;
  var padded_index = (songIndex < 9) ? '0' + (songIndex+1) : (songIndex+1);
  this.container.appendChild(document.createElement('img')).src = 'images/songs/
lyrics' + padded_index + '.jpg';
};

/* ==================== Jumping to a song ==================== */

lyricsController.showWithSongAtIndex = function (index) {
  bookletController.navigation.pushController(this);
  this.currentSong = index;
};

/* ==================== Actions ==================== */

lyricsController.playCurrentSong = function () {
    bookletController.play(appData.songs[this.currentSong]);
};

lyricsController.preferredElementToHighlightInDirection = function (currentElement,
direction) {
  return (currentElement.hasClassName('right-arrow') && direction == KEYBOARD_LEFT) ?
this.view.querySelector('.left-arrow') : undefined;
};
```

Setting Up the Photos View

The template provides a Photos screen that allows the user to cycle through photo bonus content. The Photos view uses the TKPageSliderController.

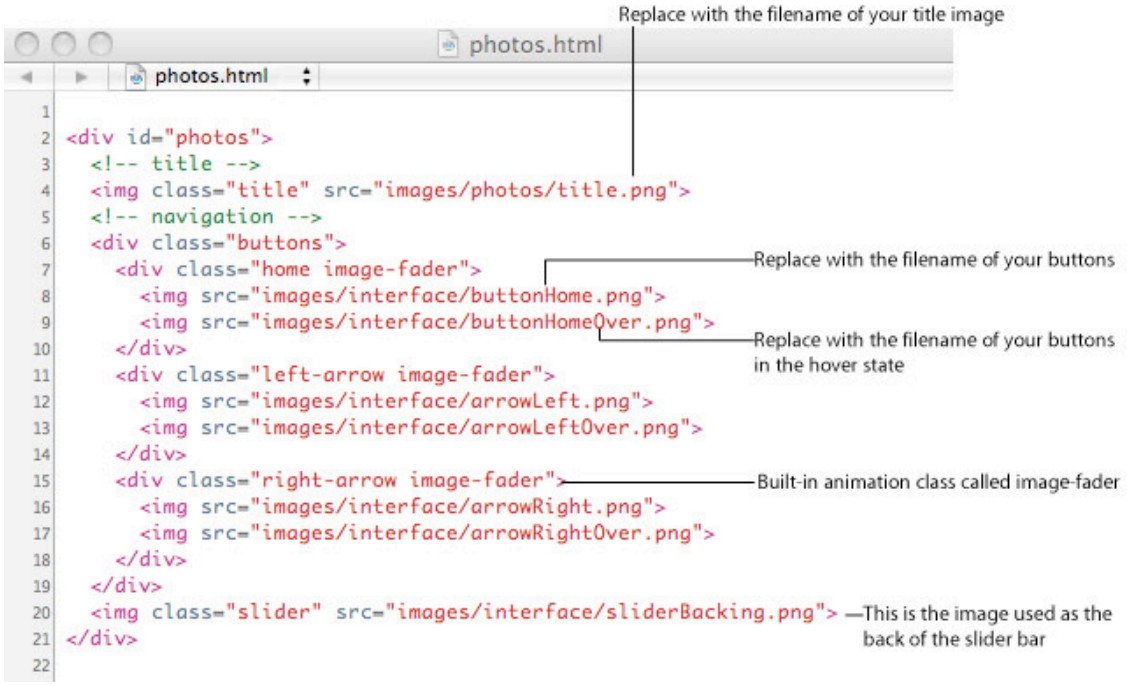


Before setting up your Photos view, put your photos in the `images/photos` folders:



To set up your Photos view:

1. In the Music template folder, open the `views` folder, and open `photo.html` with a text editor, such as TextMate.
2. Change the filename for the title image to match the filename of your image.
3. Replace the filenames of the buttons for the non-hover states with the names of the non-hover state button images you created.




```
1
2  <div id="photos">
3    <!-- title -->
4    <img class="title" src="images/photos/title.png">
5    <!-- navigation -->
6    <div class="buttons">
7      <div class="home image-fader">
8        <img src="images/interface/buttonHome.png">
9        <img src="images/interface/buttonHomeOver.png">
10     </div>
11     <div class="left-arrow image-fader">
12       <img src="images/interface/arrowLeft.png">
13       <img src="images/interface/arrowLeftOver.png">
14     </div>
15     <div class="right-arrow image-fader">
16       <img src="images/interface/arrowRight.png">
17       <img src="images/interface/arrowRightOver.png">
18     </div>
19   </div>
20   <img class="slider" src="images/interface/sliderBacking.png">
21 </div>
22
```

4. Replace the filenames of the buttons for the hover states (ending with Over.png) with the names of the hover state button images you created.

5. In the Music template folder, open the `css` folder, and open `photos.css` with a text editor, such as TextMate. The screenshot below shows only the Sliding View portion of the css file.

6. In the CSS file, replace the background image file with the one you are using for the Photos view.
7. If you want to re-position any elements, change the number of pixels from the left and/or top as needed.
8. To modify the animated transitions that occur as the photos slide in and out of the focus view, change the values for `-webkit-transform` and `opacity`.

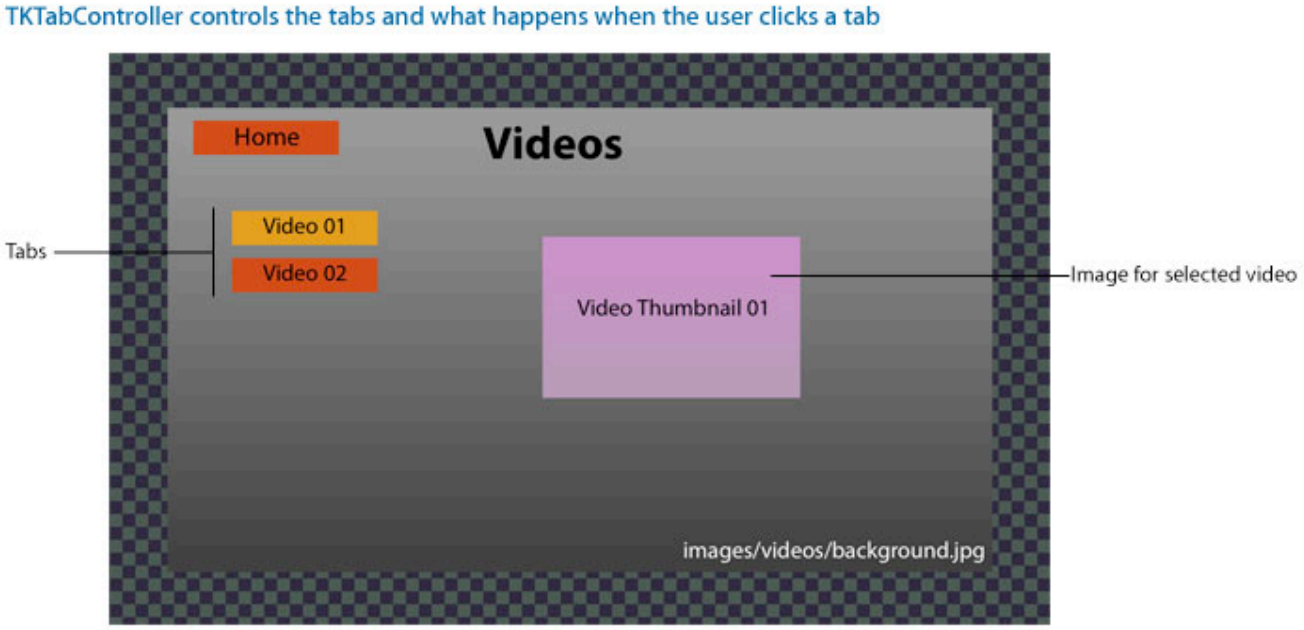
9. To see how the navigation and animations work for the Photos view, open `photos.js`.

photos.js

```
/* ==================== Photos Controller ==================== */

var photosController = new TKPageSliderController({
  id: 'photos',
  previousPageButton : '.left-arrow',
  nextPageButton : '.right-arrow',
  backButton: '.home'
});

/* ==================== View Management ==================== */

photosController.viewDidLoad = function () {
  // customize the sliding view
  this.slidingViewData = {
    orientation: TKSlidingViewOrientationHorizontal,
    activeElementIndex: 0,
    sideElementsVisible: 2,
    distanceBetweenElements: 600,  // distance between the center points of elements
    sideOffsetBefore: 0, // any extra gap you want between center and "before" element
    sideOffsetAfter: 0, // any extra gap you want between center and "after" element
    elements: this.createPhotos(),
    incrementalLoading : true
  };
  // customize the page control
  this.pageControlData = {
    numPages: appData.numberOfPhotos,
    distanceBetweenPageIndicators: 50,
    showPageElements: false,
    indicatorElement: { type: "emptyDiv" },
    pageElement: { type: "emptyDiv" },
    incrementalJumpsOnly: false,
    allowsDragging: true
  };
};

photosController.createPhotos = function () {
  var elements = [];
  for (var i = 1; i <= appData.numberOfPhotos; i++) {
    var padded_index = (i < 10) ? '0' + i : i;
    var url = 'images/photos/photo' + padded_index + '.jpg';
    elements.push({
      type: 'container',
      children: [ {type: 'image', src: url } ]
    });
  }
  return elements;
};
```

Setting Up the Videos View

The template provides a Videos view that has tabs to allow the user to select among bonus videos. The Videos view uses the TKTabController.

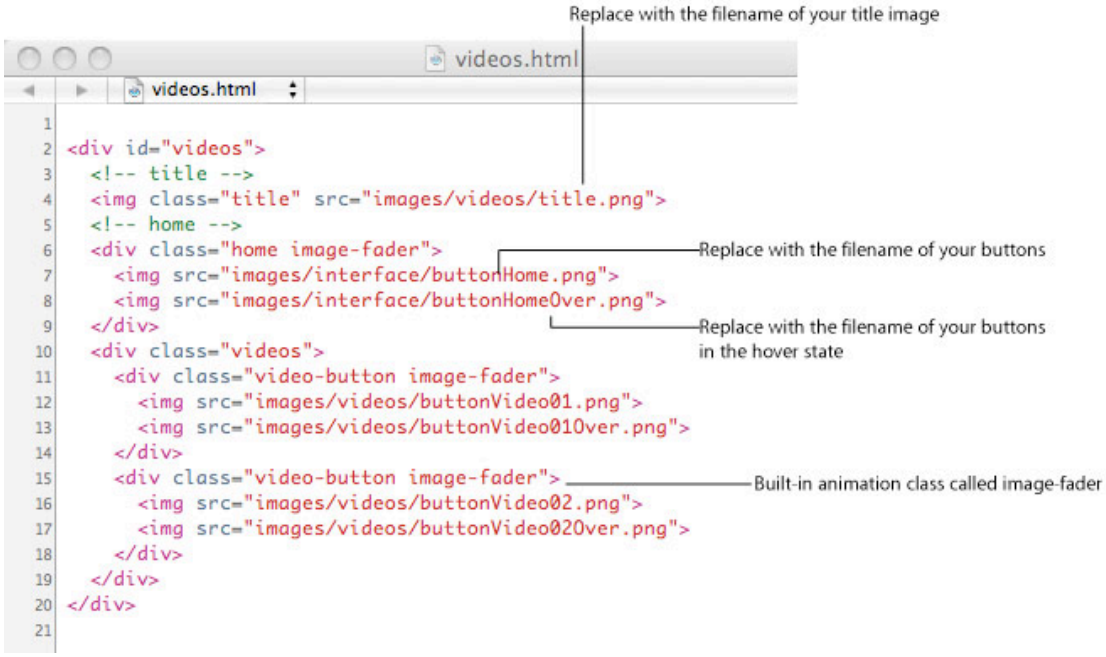


Before setting up your Videos view, put your video images in the `images/videos` and put the video clips in the `music/videos` folders.

To set up your Videos view:

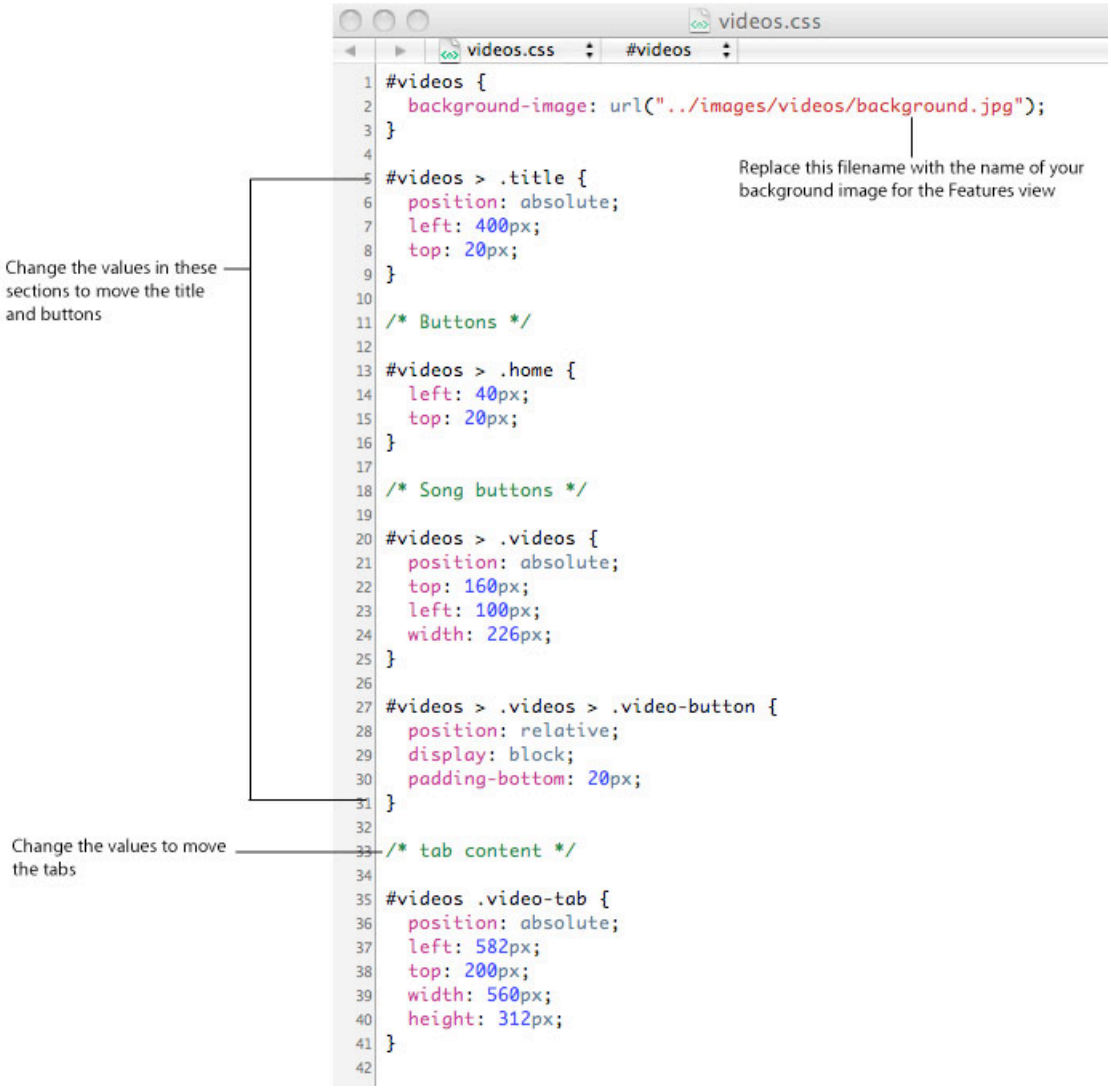
1. In the Music template folder, open the `views` folder, and open `videos.html` with a text editor, such as TextMate.
2. Change the filename for the title image to match the filename of your image.

3. Replace the filenames of the buttons for the non-hover states with the names of the non-hover state button images you created.



4. Replace the filenames of the buttons for the hover states (ending with Over.png) with the names of the hover state button images you created.

5. In the Music template folder, open the `css` folder, and open `videos.css` with a text editor, such as TextMate.

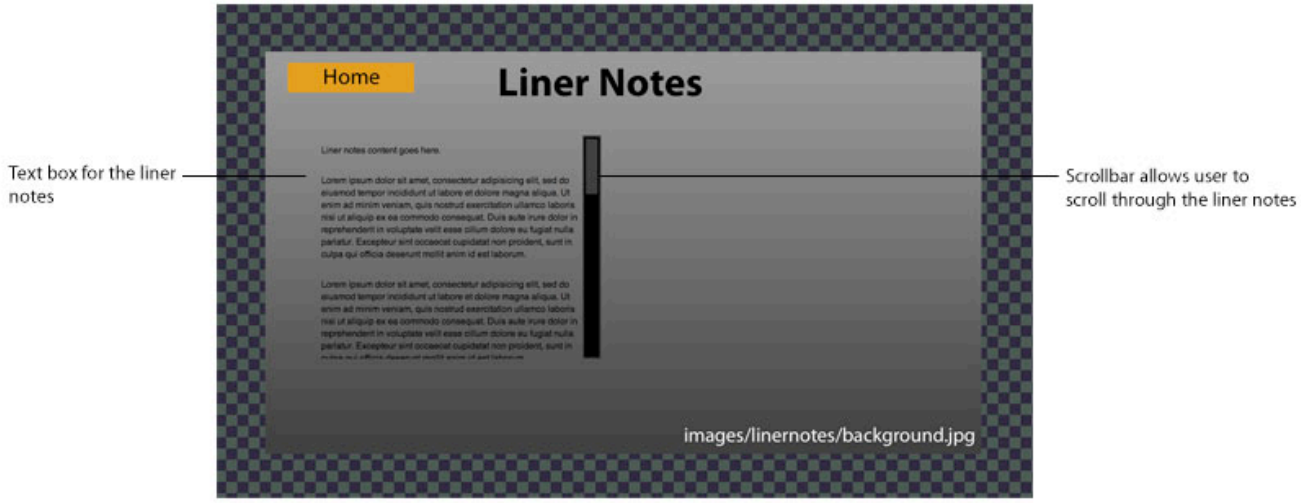


6. In the CSS file, replace the background image file with the one you are using for the Videos view.
7. If you want to re-position any elements, change the number of pixels from the left and/or top as needed.
8. To see how the navigation and animations work for the Videos view, open `videos.js`.

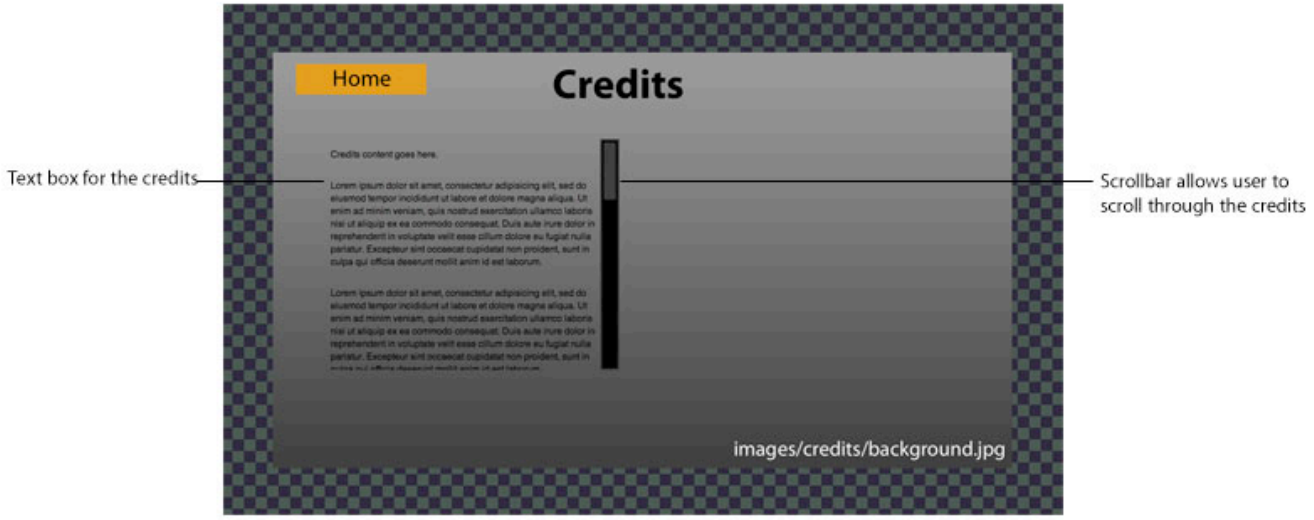
Setting Up the Liner Notes and Credits Views

The template provides Liner Notes and Credits views so your user can read them. Both of these views use the basic TKController and are very similar to set up.

TKController displays the text and scrollbar that scrolls through the liner notes.



TKController displays the text and scrollbar that scrolls through the credits.



To set up your Liner Notes (and Credits) views:

1. In the Music template folder, open the `views` folder, and open `linernotes.html` with a text editor, such as TextMate.
2. Change the filename for the title image to match the filename of your image.
3. Replace the filenames of the buttons for the non-hover states with the names of the non-hover state button images you created.
4. Replace the filenames of the buttons for the hover states (ending with Over.png) with the names of the hover state button images you created.
5. In the Music template folder, open the `css` folder, and open `linernotes.css` with a text editor, such as TextMate.

6. To see how the navigation and animations work for the Liner Notes view, open `linernotes.js`.
7. Repeat the steps above for the Credits view, substituting `credits.html`, `credits.css`, and `credits.js`.