iTunes LP
Development Guide v1.0

04-14-2014

Contents

Creating iTunes LP

iTunes LP is simply a directory with an ".itlp" extension and a set of defined contents, described below. The iTunes application will recognize such directories and import them into its library. Once there, they are associated with their corresponding album, and iTunes presents a special UI in order to allow the user to initiate playback.

At its core, iTunes LP is an HTML file that is presented using an embedded version of the Safari Web Browser within iTunes. The Web engine used in iTunes 9.0 is equivalent to the Safari 4.0 release, and you can expect most content that functions correctly when loaded into Safari to do the same in iTunes, with a few restrictions.

The HTML content must be valid HTML 4.01, XHTML 1.0 or HTML 5. It can reference other assets such as images, audio, movies, CSS and JavaScript as long as they are addressed by relative URLs and are included in the iTunes LP package - referencing media external to the iTunes LP will fail. Links to external sites are allowed, but will not open within iTunes. Instead, they will launch inside the Web browser on the user's system. Also plugins such as Flash and Java are not supported - for media use the HTML 5 `audio` or `video` elements.

The following sections describe the components of an iTunes LP.

Please Note: 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 will be reviewed by the iTunes team for appropriateness of content and for technical quality.

Package Anatomy

Each iTunes LP must be a file system directory with the following components:

1. A `manifest.xml` file, which contains the information the iTunes Store requires to associate the iTunes LP with the media assets, and describes the platforms on which the iTunes LP is playable. See Appendix B for a more detailed definition of this file.

2. An `iTunesArtwork` file. This is automatically generated at purchase time, so there's no need to author this file. However, you might want to create one for testing purposes so that artwork for your iTunes LP shows up in iTunes and on Apple TV. The easiest way to create this file is to replace the artwork of an existing iTunes LP in iTunes. The format is defined by the medium, iTunes LPs follow the 1:1 aspect ratio since they are tied to music playlists. Furthermore, these images are 8-bit color RGB, non-interlaced PNGs **without** the ".png" extension. The minimum resolution is 600x600 - this can be, of course, smaller for testing purposes since that won't be the actual file the end user will see.

3. An `iTunesMetadata.plist` file, which is an XML-based property list that describes the iTunes LP. This file is also automatically generated at purchase time, so there's also no need to author this file. For testing purposes, especially for Apple TV, you will need one. A sample file will be provided. It 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 with the media in the library as well as syncing to Apple TV.

4. An `index.html` file, which is content of the iTunes LP. This is where most of your development work will be done.

Beyond that, the content developer is free to choose the structure within the LP package. However, the TuneKit framework provided by Apple expects a particular layout, and we strongly encourage you to follow the same layout even if using your own frameworks:

- A directory named `css` that contains all the included stylesheets.
- Separate directories for any media assets, such as `images`, `audio`, `video` and `fonts`.
- A directory named `controllers` which contains the JavaScript classes implementing the behavior of the iTunes LP. Within this directory there should be a file `data.js` that defines a global object `appData`. This object is intended to hold all the non-metadata that is specific to the LP, such as the names of tracks, the number of chapters in movies, the list of available features, etc.
- A directory named `views` which contains page-level HTML templates that may be loaded by the iTunes LP. In TuneKit, a controller object uses a view to show a page or screen.

- A directory named `templates` which contains sub-page-level HTML templates that may be loaded by controllers. An example could be a snippet of HTML that is inserted multiple times into the page with slight modifications.

The Manifest

The manifest is an XML file that must live in the top level folder of the 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
- call out what platforms it is compatible with, and
- identify any items in the user's iTunes library that are to be playable via the user interface of the iTunes LP.

The filename of the manifest must be "manifest.xml". The format of a manifest is quite simple and is enforced using an XML schema (see below). Here is an example of a manifest.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest xmlns="http://apple.com/itunes/media_archive/manifest"
        version="1.0"
        media_archive_version="1.0"
        media_archive_build_number="2189">
    <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="AcmeMusic:isrc:USS1Z0944532" name="The Trick"/>
        <library_item type="song" local_id="vol001_tr002"
            xid="AcmeMusic:isrc:USS1Z0944533" name="Knock"/>
        <library_item type="song" local_id="vol001_tr003"
            xid="AcmeMusic:isrc:USS1Z0944534" name="Defend It"/>
        <library_item type="video" local_id="vol001_tr004"
            xid="AcmeMusic:isrc:USS1Z0944535" name="Conscience Dies"/>
        <library_item type="video" local_id="vol001_tr005"
            xid="AcmeMusic:isrc:USS1Z0944536" name="Same Difference"/>
    </library_items>
</manifest>
```

The <manifest> element

The <manifest> element includes the following attributes:

- **xmlns** *(required)*: The value of this element must be "http://apple.com/itunes/media_archive/manifest" in order to identify this XML document as conforming to the XML schema for manifest files.
- **version** *(required)*: This identifies the version of the manifest schema that the document conforms to; at present, the only version is 1.0.
- **media_archive_version** *(required)*: This identifies a released version of the specific iTunes LP in which the manifest resides; it is used to support updates to the content in the future. The format of this attribute is a string consisting of one or more dot-separated strings of digits; for example, the following are valid values for this attribute: "4", "1.0", "4.2.1", "5545.12", "1.4.0.0.5"; the following are invalid values: "GX5", "3.4.2b6", "2.6GM", "1.0 (Gold)", "-3.6". When an update to an existing iTunes LP is submitted for distribution through the iTunes Store, the media_archive_version must increase relative to the previously uploaded version; for example, if a previously-available version had a media_archive_version of

"1.4", then any of the following would be acceptable as the next update: "1.4.0.0.0.0.5", "1.4.1", "1.4.332", "1.5"; but all of the following (though syntactically valid) would be unacceptable: "1.4", "1.3", "1.3.99999", "0.9".

- **media_archive_build_number** (optional): While developing an iTunes LP, there might be several builds targeting a given released version prior to its actual release. This optional attribute may be used to differentiate individual builds of the same release. It is subject to the same syntax restrictions as media_archive_build_number, but unlike media_archive_build_number, it is not subject to any restrictions regarding increasing from one update to the next.

The <requirements> element and <supported_platforms>

The <requirements> element includes a <supported_platforms> element, which, in turn, includes one or more <supported_platform> elements identifying platforms with which the iTunes LP is compatible. At present, the only valid platforms are "iTunes" and "AppleTV". The minimum version of iTunes that supports iTunes LP is 9.0; the minimum version of Apple TV that supports iTunes LP is 3.0. All iTunes LP content that is to be distributed via the iTunes Store must be compatible with iTunes 9.0 and Apple TV 3.0; hence this element must be supplied exactly as shown in the example above.

The <library_items> element

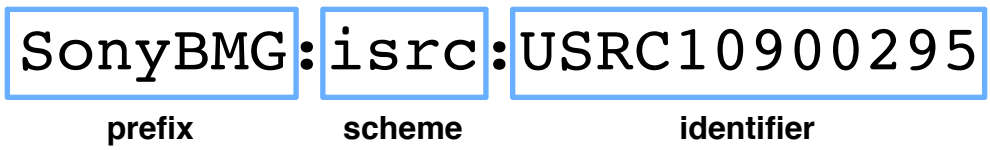
The most important element in a manifest.xml file is <library_items> which may contain zero or more <library_item> elements. Each single <library_item> element identifies an audio or video file expected to be in the user's iTunes library that is to be playable from the user interface of the iTunes LP. The following are the attributes of a <library_item> element:

- **type** (required): The value of the type attribute should be "song" for audio files and "video" for video files.
- **local_id** (optional): The local_id attribute should be populated with an identifier that is locally unique within this iTunes LP — it does not need to be globally unique across other iTunes LP content. For iTunes LP, it is conventionally populated with a string of the form "volXXX_trYYY", where XXX is the volume number (or "disc number") of the track, padded to three digits, and YYY is the track number, padded to three digits.
- **xid** (required): An XID is a globally unique identifier that can be attached to an audio or video file for use by iTunes. When iTunes LP play audio or video files from the user's iTunes library, the file to play is identified by means of its XID. In order for iTunes LP to function correctly, the XID must be set on the audio or video files themselves and declared in the xid attribute of a <library_item> tag in the manifest. XIDs must conform to the format described in the section of this document entitled The XID Format.
- **name** (optional): To aid readability of the manifest file, it is recommended that you provide a human-readable name in the name attribute of the <library_item>. Conventionally, this should be the same as the name of the item as displayed in the iTunes library.

XID Structure

An XID is a string composed of three colon-delimited segments:

1. A ***prefix*** specific to the content provider. Content providers who provide content to the iTunes Store each have a specific prefix assigned for use in this manner. All XIDs identifying content from a given content provider should use the same prefix. If you are testing iTunes LP prior to submitting it to the iTunes Store, and you have not yet finalized the identifiers for your content, you may wish to temporarily use the reserved prefix TEST, and combine it with the uuid scheme, described below. When IDs for the content are finalized, you should replace the TEST XIDs with XIDs using a true provider name as a prefix and an XID scheme based on one of the other schemes.
2. A ***scheme*** specifying the type of identifier being used. Supported schemes are described below, with examples of each.
3. An ***identifier*** conforming to the scheme.



The following are currently-supported XID schemes:

- **upc** - For content identified with a UPC code.
  Example: `buenavista:upc:760894712785`

- **isrc** - International Standard Recording Code for sound recordings and music videos.
  Example: `SonyBMG:isrc:USSM19922509`

- **isan** - International Standard Audiovisual Number for video content.
  Example: `Paramount:isan:0000-0000-6601-0000-4-0000-0000-P`

- **grid** - Global Release Identifier standard developed by the recording industry.
  Example: `Warner:grid:A10302B0000978370N`

- **uuid** - Universally Unique Identifier (RFC 4122); this is the best choice for temporary use while testing iTunes LP. UUIDs can easily be created on demand; for example, by the uuidgen command-line tool on Mac OS X.
  Example: `TEST:uuid:6F092637-666F-49F6-9F77-978627D50B5A`

- **vendor_id** - A content provider's custom identifier for a piece of content not identified by any of the standards above.
  Example: `sonypicturesentertainment:vendor_id:DA_VINCI_CODE_EC_2006`

File Naming

No files may start with the following (any combination or upper/lower case): `"itunes"` or `"apple"`. Other than that, the developer is free to choose the names for any other files in the package. However, given that iTunes LPs are cross-platform, it is strongly recommended that the following rules be adhered to:

- Use lowercase names for all files. This reduces the chance of case-sensitive file systems being unable to find a resource that has been referenced with the wrong case.
- Use standard file extensions for common file types. Since iTunes LP resources are served from the file system, there is no HTTP headers indicating what the media type of a resource is.
- Use Latin characters where possible.

Windows Support

The iTunes client sets a number of custom scrollbar properties which are inherited into iTunes content. This can sometimes cause any scrollbar customization performed within an iTunes LP to display incorrectly, especially on Windows. For this reason, if you use custom scrollbars in your content you should make sure to reset all scrollbar properties before you apply your customization.

The Apple template provides a stylesheet to do this, called "scrollbar-reset.css". This should be linked from your `index.html` file, as demonstrated by the example below.

```
<link rel="stylesheet" href="css/scrollbar-reset.css" type="text/css"
media="screen" charset="utf-8">
```

Audio and Video

- **Bonus Content (Audio and Video) Metadata for Display**

When a user plays any bonus content in iTunes, the title of the bonus content and the main asset metadata (such as movie title and director for movies and album title and artist for song tracks) displays in the LCD area at the top of the iTunes window.

For iTunes LPs, open the `data.js` file. Change "Music" to the title of the album and change "Artist" to the performing artist.

- **Linking to Audio and Video in iTunes**

iTunes LP packages include the core assets (for example, the album song and video tracks) and bonus content assets. Bonus content assets can include bonus songs, interview videos, deleted scenes and so on. The process for hooking them up in iTunes differs depending on if the assets are core assets or if they are bonus content assets.

Linking to Core Assets
Core Assets are defined as the main album or movie displayed and downloaded within iTunes.
When a user plays core audio or video content from the iTunes LP, the file iTunes plays is identified by its XID. An XID is a globally unique identifier that can be attached to an audio or video file for use by iTunes. The XID must be set on the audio or video files themselves and declared in the xid attribute of `a <library_item> tag` in the manifest file.
To play a song track or video file by its XID, use:

- `bookletController.play(trackObj)`, where `trackObj` looks like `{ xid : "XID" }`. All other fields are optional.

The `bookletController.play` really does two things:

- calls `.getTrack`, which will return null if iTunes can't find that XID
- calls `.play` on the trackObj, which causes said track to play

Alternatively, you can use a playlist if you want continuous playback of multiple items.

Linking to Bonus Content
Bonus Content is defined as ancillary video or audio files that are delivered outside of the core asset and are not visible on a track listing or elsewhere from iTunes itself; bonus content is only accessed from within the iTunes LP itself.
Bonus content is accessed by file path within the iTunes LP. For example, bonus videos are stored in the `videos` folder.

Playback is by `bookletController.playNonLibraryContent(trackObj)` where `trackObj` has the required fields `{ src : "filename", string : "string to display in LCD"}`. This function also requires `appData.feature.artist` and `appData.feature.title` to be set in order to construct the metadata. `src` should be the name of the file; `bookletController` appends `"videos/"` to the front of this name, and looks in that directory. This is what the iTunes Store import script expects, so it is important not to deviate from this.
Note that since bonus content is played by path, they cannot be resumed; they always play starting from the beginning.

```
var t = bookletController.buildPlaylist(appData.songs);
 t.play();        ///starts at 1
t.play(index)    ///starts at index
```

Linking to iTunes Store and External Web Sites

iTunes LP packages usually have a "More" page that includes links to iTunes Store for additional purchases, such as the soundtrack for the movie or outside links to the studio. The screenshot below shows the More view html file for the Movie template:

The links are just normal HTML anchor tags. The only thing to consider is that they must target "_blank".

For iTunes Store links, note that the `href` is targeting `itunes://` not `http:`

```
<a href="itunes://itunes.apple.com/us/album/albumname/albumid"
target="_blank"></a>
```

For web links, use standard http:

```
<a href="http://www.apple.com/trailers/" target="_blank"></a>
```

Background Audio

It is common for an iTunes LP to play some ambient audio while active. This audio should be a file included within the iTunes LP package and accessed via the `audio` element of HTML.

Even though the playback is controlled by the LP, the iTunes application interface will allow the user to control volume.

The provided Apple template has an example of background audio, and the TuneKit framework has a simple interface to support the feature. Details are in the Appendix B at the end of this document.

Best Practices for Creating Background Audio:

iTunes LP:
We recommend that your background audio file for an iTunes LP be no longer than 30 seconds and does not loop. We further recommend that this file have a lower volume than the audio files of your album (reduce by 5-10 decibels).

Encoding Background Audio Files:

You need to use iTunes to create your final background audio file in iTunes Plus .m4a format (256 kbps stereo, 44.100 kHz, AAC).

To do this, once you have your master background audio file (.wav file format) copy it into iTunes.

1. From within iTunes, select File Menu > Preferences... > General > Import Settings.
2. Select AAC Encoder and use the iTunes Plus setting.
3. Then go to your library, select the master audio .wav file and select the Advanced Menu > Create AAC Version.
4. This will create the final iTunes Plus .m4a file you can use in your iTunes LP.
5. Finally, place this iTunes Plus .m4a file in the "audio" folder within your .ite or .itlp package for it to work appropriately.

Code to Implement Background Audio and Audio Looping Functionality:

```
audioLoop : { src : "path/to/file.m4a", loop : true},
audioLoop : { src : "path/to/file.m4a", loop : false},
```

Animations

iTunes LPs are intended to run on a range of systems, both desktop PCs with varying capabilities, and Apple TV, which is a resource-limited platform. Since the iTunes LP may be launched on a low-power machine, the content developer should be aware that complex animations may create a sub-optimal user experience.

Often, an acceptable technique for animations is to use CSS. CSS Animations have the advantage that they are executed by the Safari engine and can be accelerated when possible. For this reason they often have exceptionally better performance than animations performed in JavaScript. An example of CSS Animations is given in the section below on "Image Faders".

Animations in CSS also have the benefit that they can be conditionally excluded from Apple TV where necessary. See the section below on Apple TV compatibility. Also, CSS animations are usually easier to pause or remove than JavaScript. Content developers should always try to reduce the amount of CPU load, especially when media is playing.

Controller Animations

This listing shows how to fire complex animations on the first load of the home controller. The idea is to find an element and remove a top level class that will remove some CSS animation rule. This example stops the initial animation from occurring every time the controller becomes active by changing a class on the top level wrapper of the animating elements. This causes different CSS rules to be in effect.

```
homeController.viewDidLoad = function () {
  this.firstTime = true;
}

homeController.viewDidAppear = function (){
  if (this.firstTime) {
    var _this = this;
    setTimeout(function() {

document.querySelector(animationTopLevelDIV).removeClassName(animatingClass);
    }, 0);
```

Using movies in the User Interface

There are two types of movies used in an iTunes LP. The most common type is a video that can be considered part of the extra features provided by the package, such as an interview or a behind the scenes look at a movie scene. These are described above, in the section on "Audio and Video".

The second type of movie is part of the user interface. For example, a small clip from the movie that plays in the background of the home screen.

If possible, these background ambient effects should be implemented using animations (preferably CSS Animations). However, in the case where the effect really is a movie clip, the HTML `video` element is appropriate. Note, however, that this is not currently supported on Apple TV, so the design must accommodate an alternate effect, such as a static image or crossfading between a reduced number of static frames.

Furthermore, it is essential that any inline videos be paused if the main media asset begins playback or the video is obscured by other UI elements.

Lastly, due to current performance restrictions, developers should keep the dimensions, framerate and bitrate of inline movies to an absolute minimum. In many cases, videos larger than about 25% of the screen size will have unacceptable results.

Controller Transitions

Transitions between controllers can be specified in the controller constructor using the `becomesActiveTransition` and `becomesInactiveTransition` properties. Default is to a 100% opacity change, although any CSS rule can be specified.

The following shows how to set up transitions in the controller constructor (items in angle brackets are CSS values. Note that you can have multiple comma-delimited CSS rules).

```
becomesActiveTransition : {
   properties : ['<css rule>'],
   from : ['<initial value>'],
   to : ['<ending value>']
 },
 becomesInactiveTransition : {
   properties : ['<css rule>'],
   from : ['<initial value>'],
   to : [<ending value>']
 },
```

In the following example, when this controller becomes active, (meaning it is selected and shows on screen), it animates its X translation from 100% to 0%. This causes the view to slide in from the side. When this controller becomes inactive, (meaning another controller is selected and shows on the screen), it animates its X translation from 0% to 100%. This causes the view to slide off of the screen.

```
     becomesActiveTransition : {
         properties : ['-webkit-transform'],
         from : ['translateX(100%)'],
         to : ['translateX(0)']
       },
       becomesInactiveTransition : {
         properties : ['-webkit-transform'],
         to : ['translateX(100%)']
       },
```

By combining transitions with CSS transforms, you can create many interesting effects. Using CSS transforms, you can resize, fade, move, and rotate the elements on the page. You can specify as many properties as you would like, with the indices matching the from/to arrays. See CSS3 documentation for more information.

Visualizers

For information about developing Visualizers, see the Design Best Practices document.

To exclude the visualizer from Apple TV, create variable `t` that is an iTunes temporary playlist of all the songs in an iTunes LP. Play the temporary playlist `t`. If the platform is not Apple TV, show the visualizer by pushing its controller onto the navigation stack. `albumHelper.play()` is bound to an action. The following is an example:

```
albumHelper.play = function (){
    var t = bookletController.buildPlaylist(appData.songs);
     t.play();
     if (window.iTunes.platform == "Windows" || window.iTunes.platform ==
"Mac" || window.iTunes.platform == "Emulator") {

TKNavigationController.sharedNavigation.pushController(visualizerController);
     }
}
```

Apple TV Compatibility

The same iTunes LP content used on the desktop is used when running on Apple TV. Therefore, it is up to the content developer to code a single package that will work on both platforms. In the majority of cases, the functionality available is identical; it is the user experience that differs (both in performance and interface).

When launched on Apple TV, the iTunes LP will be displayed within a viewport that is sized at 1280px wide and 720px high, even though the display resolution may be different.

Content developers should ensure that all navigable content fits within the title and action-safe areas of a television display. Action-safe calls for a 5% border on each side, while title-safe calls for a 10% border on each side. As an example, for a 1280- x 720 display, action-safe is 1152 x 648 and title-safe is 1024 x 576, both centered in the middle of the screen.

Also, Apple TV does not have a pointer device, only a six-button directional remote. Content developers should ensure that the design of the LP makes sense under these interaction restrictions. It is not acceptable to have areas of the UI that cannot be reached. Rather, these should be hidden from the user when running on Apple TV.

User interaction on the Apple TV is not automatic. If you are using the TuneKit framework, it will handle a large amount of the interaction for you. Otherwise, the iTunes LP must handle all the interaction itself. This is a fairly complicated process, hence we strongly recommend developers use TuneKit or at least examine the code in order to replicate the behavior.

In order to indicate that the iTunes LP can be displayed on Apple TV, the `index.html` file should include the following two `meta` tags in its header section.

```
<meta name="hdtv-fullscreen" content="true"/>
<meta name="hdtv-cursor-off" content="true"/>
```

Lastly, there are a number of platform restrictions on Apple TV:

- iTunes LP cannot use inline video. All `video` elements will display full-screen within the native Apple TV playback interface.
- iTunes LP can use animated GIF images, but they are a significant performance drain. Their use should be avoided if possible.
- The iTunes LP must ensure that they allow the user to navigate back to the native menu system. Do not attempt to override the system behavior.
- Be conscious of the risk of screen burn-in, especially if using visualizers. To avoid burn-in on televisions, it is recommended that a visualizer cause every pixel to change frequently.

Custom style rules

Apple TV often requires a significantly different design than that created for a computer. It is possible in most cases to achieve this using the same HTML content via conditional CSS rules. Apple TV has a special media feature (`-webkit-apple-tv`) that can be used in a CSS media rule to apply styles that are only visible on the TV. The most common case for such a rule is to disable the page bleed, since it will never be displayed on the TV:

```
/* do not show bleed image on Apple TV */
@media (-webkit-apple-tv) {
  body {
    background-image: none;
  }
}
```

Unfortunately, there currently is no equivalent inverse approach, that will apply rules only to the desktop and not to Apple TV. You have to either author it in reverse (rules for desktop in the shared space, override them in Apple TV only rules) or use more specific but potentially fragile and non-future compatible media queries (such as screensize).

Using JavaScript, there are two ways to indicate that the code that follows is for Apple TV:

```
if (window.iTunes.platform == "Windows" || window.iTunes.platform == "Mac" ||
window.iTunes.platform == "Emulator") {
     //code here
 }
```

Or,

```
if (IS_APPLE_TV){
    //code here
}
```

Using CSS, there is one way to indicate that the code that follows is for Apple TV:

```
@media (-webkit-apple-tv) {
    //atv specific rules here
}
```

The following JavaScript example from Batman excludes loading of a background video if the platform is Apple TV:

```
homeController.viewDidLoad = function (){
 this.backgroundMovie = null;
 if (window.iTunes.platform == "Windows" || window.iTunes.platform == "Mac"
|| window.iTunes.platform == "Emulator") {
   this.backgroundMovie = document.createElement("video");
   this.backgroundMovie.src = "images/home/batCavePJPG12.mov";
   this.backgroundMovie.loop = true;
   this.backgroundMovie.autoplay = true;
   this._view.appendChild(this.backgroundMovie);
   this.backgroundMovie.load();
 }
 bookletController.registerForDisplayUpdates(this);
};
```

The following CSS example excludes the bleed image from displaying on Apple TV:

```
/* remove the bleed on Apple TV */
@media (-webkit-apple-tv) {
 #bleed {
   background-image: none;
 }
}
```

To exclude the visualizer from Apple TV, create variable `t` that is an iTunes temporary playlist of all the songs in an iTunes LP. Play the temporary playlist `t`. If the platform is not Apple TV, show the visualizer by pushing its controller onto the navigation stack. `albumHelper.play()` is bound to an action. The following is an example:

Appendix A : The Manifest XSD

In order to be considered structurally valid, the manifest.xml file must conform to the schema below. You can validate your manifest.xml file against this schema using any number of tools supporting the standard XSD schema format, including xmllint, which is included on Mac OS X. For example, the following command uses xmllint to validate a manifest:

xmllint --noout --schema manifest.xsd manifest.xml

Below are the contents of manifest.xsd for use in validation:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://apple.com/itunes/media_archive/manifest"
            xmlns:ima="http://apple.com/itunes/media_archive/manifest"
            elementFormDefault="qualified">
    <xsd:element name="manifest" type="ima:Manifest"/>
    <xsd:complexType name="Manifest">
        <xsd:sequence>
            <xsd:element name="requirements" type="ima:RequirementList"/>
            <xsd:element name="library_items" type="ima:LibraryList"/>
        </xsd:sequence>
        <xsd:attribute name="version" type="ima:VersionString" use="required"/>
        <xsd:attribute name="media_archive_version" type="ima:VersionString"
use="required"/>
        <xsd:attribute name="media_archive_build_number" type="ima:VersionString"
use="optional"/>
    </xsd:complexType>
    <xsd:complexType name="RequirementList">
        <xsd:sequence>
            <xsd:element name="supported_platforms"
type="ima:SupportedPlatformList" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="SupportedPlatformList">
        <xsd:sequence>
            <xsd:element name="platform" type="ima:SupportedPlatformItem"
minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="SupportedPlatformItem">
        <xsd:attribute name="name" type="xsd:string" use="required"/>
        <xsd:attribute name="minimum_version" type="xsd:string" use="required"/>
    </xsd:complexType>
    <xsd:complexType name="LibraryList">
        <xsd:sequence>
            <xsd:element name="library_item" type="ima:LibraryItem" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="LibraryItem">
        <xsd:attribute name="type" type="ima:LibraryItemTypeString" use="required"/
>
        <xsd:attribute name="xid" type="ima:XIDString" use="required"/>
        <xsd:attribute name="local_id" type="xsd:string" use="optional"/>
        <xsd:attribute name="name" type="xsd:string" use="optional"/>
    </xsd:complexType>
    <xsd:simpleType name="XIDString">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="[^:]+:(upc|isrc|isan|grid|uuid|vendor_id):[^:]+"/>
```

```
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="LibraryItemTypeString">
        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="song"/>
            <xsd:enumeration value="video"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="VersionString">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="[0-9]+(\.[0-9]+)*"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
```

Appendix B: TuneKit Programming

The TuneKit controller classes are designed to streamline the creation of Cocktail booklets that follow basic design patterns. Using the TuneKit controllers, authors can use JSON to instantiate most of the controllers' properties and rely on an automated spatial keyboard and Apple Remote focus navigation across interactive portions of the booklet.

This section gives an introduction to the TuneKit framework. For more comprehensive details, refer to the TuneKit API documentation.

TKController

The TKController class is the base class for all controllers, as such all of its properties apply to other controller types described below. A basic controller is instantiated as follows:

```
var controller = new TKController({
  id: 'home',
  actions : [
    { selector: '.play', action: bookletController.playFeature }
  ],
  navigatesTo : [
    { selector: '.chapters', controller: chaptersController },
    { selector: '.extras', controller: extrasController }
  ]
});
```

In this example, we create a controller which will automatically load the element with id "home" and wires up actions and navigation anchors to its content.

View

Controllers manage an HTML fragment referred to as the view. The view is loaded dynamically by first looking for an element in the existing DOM tree with the given id property, and if it cannot be found, loading the content of the HTML page in the booklet's views/ directory with the filename of the id. If this also fails, an empty element with that id is created.

Outlets

For any element that you would like to get a reference to, you can define an outlet. Using the outlets property, simply define an array of objects each specifying a name and a selector to target the element. Then, you'll simply have a reference of the name provided on your controller, without having to write any code to get a pointer to it. So if you define an outlet with name : myOutlet, you'll be able to just use the .myOutlet property on your controller object to get to it.

Actions

Actions simply tie an HTML element specified by a CSS selector to a JavaScript callback. Use the actions property to assign an array of such actions, where each action is an object with a selector property specifying the CSS selector to use to target the element, and an action property for the JS function to call. The value of action can be a reference to a function defined elsewhere or a direct inline function definition. It's important to use controller actions instead of wiring click events directly in order to abstract the actual user interaction required depending on the platform. Then TuneKit will handle the user interaction for you. This is particularly important on Apple TV, where the lack of pointer means the target for events like click are unclear.

Navigation

Instead of manually interacting with the TKNavigationController, you can simply identify HTML elements that move to a new controller with the navigatesTo property. Simply pass an array of objects defining both a selector and a controller, which should be another TKController instance, or one of its subclasses. TuneKit will then automatically set up user interaction that navigates between controllers when the user activates the element given in the selector.
Also, you can define an element that acts as a back button, thus popping the controller from the navigation stack. Simply use the backButton property and set it to whatever CSS selector matches the element you wish to use as a trigger to go back.

Custom Transitions

As the user navigates from controller to controller, the TKNavigationController will use a fade transition by default, but you can provide a custom transition. Use the becomesActiveTransition and becomesInactiveTransition properties. Both these properties expect an object supporting the following:

- properties: an array of CSS properties that are transitioned during the transition. REQUIRED.
- base: an associative array of basic properties to apply before the transition starts. Even indexes are CSS properties and odd indexes are values. OPTIONAL.
- from: an array of CSS values for the initial state of the transition, each index mapping the property at the same index in properties. OPTIONAL.
- to: an array of CSS values for the end state of the transition, each index mapping the property at the same index in properties. REQUIRED.
- duration: a float duration for the transition in seconds. OPTIONAL, defaults to 0.5.
- delay: a float duration for the delay to apply before the transition starts in seconds. OPTIONAL, defaults to 0.

Callbacks

Controllers have a variety of functions that you can override and subclass to monitor their life cycle and various interactions:

- viewDidLoad: the controller's view was loaded for the first time, although it may not have been appended to the DOM tree just yet.
- viewWillAppear: the controller's view will appear on screen, for instance as it's pushed as the topController of the navigation stack.
- viewWillDisappear: the controller's view will disappear from screen, for instance when the user navigates away from it.
- viewDidAppear: the controller's view has appeared on screen, for instance as it's pushed as the topController of the navigation stack and the transition has completed.
- viewDidDisappear: the controller's view has disappeared from screen, for instance when the user navigates away from it and the transition has completed.

TKNavigationController

Traditionally, you will use a single TKNavigationController instance to manage the navigation between the various controllers. On top of the TKController properties defined above, TKNavigationController offers two extra properties. The rootController property lets you define the controller to use as the first controller displayed on screen, while the delegate property lets you define an optional supporting object that will get delegate method calls to track as controllers are pushed onto the stack. There are two optional delegate methods for a TKNavigationController:

- navigationControllerWillShowController(navigationController, controller): notifies that a controller will appear, either because it's being pushed, or it was the back controller when the top controller got popped off the stack. It can be interesting to implement this delegate method to customize the transitions used on the controller going off-screen and the one coming on-screen.
- navigationControllerDidShowController(navigationController, controller): notifies that a controllers did appear, after a push or pop transition has completed.

Note that you can access the top controller at all times using the topController property.

TKTabController

A TKTabController allows to bind a series of elements to a series of controllers that will be shown one at a time as the user selects tabs. The two key properties for a tab controller are:

- tabs: a CSS selector that matches the elements used as tabs
- controllers: an array of TKController instances

There should be as many tabs as there are controllers or the behavior is undefined. When a tab is selected, it receives a DOM focus event and uses the transitions defined on the target controller to show it, and remove the previous controller. You can find out what the current controller is using the selectedController and selectedIndex properties. As tabs are selected, the following delegate methods will be triggered on the object passed as the delegate property:

- tabControllerWillShowController(tabController, controller): notifies that a controller will appear.
- tabControllerDidShowController(tabController, controller): notifies that a controllers did appear.

TKPageSliderController

A TKPageSliderController combines and controls a TKSlidingView and a TKPageControl, always keeping the two in sync. There are two properties to provide the data to either of those views:

- slidingViewData: the data for the managed TKSlidingView.
- pageControlData: the data for the managed TKPageControl.

The TKPageSliderController will set itself as the delegate to both views and you should attempt to monitor delegate method calls on either objects. Instead, you'll be notified of a change in focus or selection using the methods pageWasFocused(index) and pageWasSelected(index), both providing the index of the element of interest.

You can optionally provide a pointer to elements that should be used as triggers to move one page back or one page further using the previousPageButton and nextPageButton properties. Both expect a CSS selector and the correct scripting logic will be automatically provided to connect with the sliding view.

Image faders (hover buttons)

TuneKit provides a very simple way to have buttons that react to hover (as the pointer moves over the element) and to any specific focus navigation that is applied by the system. The technique also has the benefit of being flicker-free (no flashing on first use) since it does not require the images to be preloaded. While it is not mandatory, we encourage you to use this technique.

An image fader is a container element with two children. The container is marked with the CSS class "image-fader". The children of the container should be the "off" and "on" states for the hover button respectively. Here is an example that provides a "Home" button within a view.

```
<div class="home image-fader">
  <img src="images/home.png">
  <img src="images/homeOver.png">
</div>
```

When the image fader is not active (when it doesn't have focus and the pointer is not over the element) the first child will be visible. As the element gets focus or is hovered, the first child will fade out and the second child will fade in.

The following is the CSS that provides the functionality:

```
.image-fader {
  position: absolute;
  font-size: 0;
 cursor:pointer;
}

.image-fader > img {
  -webkit-transition: opacity 0.25s;
}

.image-fader > img:nth-of-type(2) {
  position: absolute;
  top: 0;
  left: 0;
  opacity: 0;
}

.image-fader:hover,
.image-fader.tk-highlighted > img:nth-of-type(2) {
  opacity: 1;
}
```

Preloading image assets

Please note that preloading images assets on Apple TV can negatively affect performance, especially when iTunes LP is first loaded.

While TuneKit provides a technique for "hover" buttons that avoids the need for preloading images, in some cases it is unavoidable (for example, when the image content is dynamically loaded from script). Every TuneKit controller has an optional `preloads` attribute that takes a list of URLs to load as the view is processed for the first time. While this does not guarantee that the image will be available when you need to display it (especially on non-desktop platforms) it is a good compromise between using performance and appearance.

```
var extrasController = new TKController({
  ...
  preloads: [
    'images/extras/backgroundA.jpg',
    'images/extras/backgroundB.jpg'
  ];
  ...
});
```

Background Audio

When TuneKit initializes it looks in the `appData` object for an attribute called `audioLoop` which specifies if any background music should be played.

The `audioLoop` attribute is an object with two parameters:

- `src`: a URL that points to an audio file. The supported formats for audio are AAC and MP3. REQUIRED
- `loop`: a boolean value that indicates whether or not the audio should repeat infinitely. This also controls whether the music restarts after the user has played media (such as the movie or listening to music). The default value is false. OPTIONAL.

An example of playing background audio is shown below.

```
audioLoop : { src : "audio/background.m4a", loop : false }
```

Navigation on the Apple TV

In general, any interactive element you register with any of the properties from TKController or any of its subclass automatically registers as a keyboard-navigable element. On top of such elements, any HTML `<a>` elements will be registered as well. Any time an element is reached using the automated keyboard navigation, it's given the tk-highlighted CSS class, on top of any other CSS classes it may already have. It also receives a highlight DOM event. When an element loses highlight, the custom CSS class is removed and the unhighlight DOM event is triggered.

In order to find the nearest element to navigate to in a given direction, the built-in navigation mechanism queries the CSS metrics of all the navigable elements in real-time. As such, it is important that all elements that can be given highlight have available CSS metrics and those metrics are accurate (that is, the element is not sized larger than the area it requires to display). Note that container elements that have only absolutely-positioned children will not get a size, so make sure to provide an explicit size on those elements using CSS. It is easy using the Web Inspector to check that your navigable elements have a size, you can simply hover over the HTML element in the Elements tab and see the bounding box painted as an overlay on the webpage.

Note that there are some simple ways to customize the metrics for a controller. You can override the customMetricsForElement(element) method on your controller to provide custom metrics for a given element instead of those that would be naturally queried from CSS.

You can also explicitly exclude elements from the list of navigable elements by applying the tk-inactive CSS class to them.

By default, the top-most element in the controller is highlighted. You can use the highlightedElement property to specify a different one with a CSS selector.

Finally, you might want to completely override the navigation system to provide your own custom navigation. This can be done easily by overriding the preferredElementToHighlightInDirection(currentElement, direction) method on your controller.