Authoring (Mastering) a Lib-Ray Disk

If you are a filmmaker, you now have the opportunity to release your film in the Lib-Ray format. This has a number of advantages: Relative to DVD, Lib-Ray offers high-definition video and high-quality audio options. Relative to streaming videos over the internet or offering copies of video files from download sites, Lib-Ray offers higher video quality (because it's not limited so much by bandwidth considerations for download), special features, and organized menus for handling audio and subtitle setup. Relative to Blu-Ray, Lib-Ray offers freedom from DRM and region-coding, and of course, much lower up-front costs since you are not paying license fees for the use of the format (Lib-Ray is a free and open standard).

Yes, You Can Contract This Job...

Of course, since Lib-Ray is brand new (this is only a *prototype* – version "0.2"), there are no dedicated authoring tools. Instead, you will need to create your disk using simple low-level tools, and this will require some skill with Linux and the command line. If you find this daunting, you may want to consider contracting out the mastering of your Lib-Ray disk (See `lib-ray.com` for service offerings or contact Terry Hancock directly at *digitante@gmail.com*).

This will cost you some money, but is still very economical compared to Blu-Ray production. For somewhat less money, we may also offer certification of your master disk to assure you and your customers that it truly meets the Lib-Ray standard.

We also hope to write some authoring software and templates to make do-it-yourself Lib-Ray easier, but at this point, we are still learning too!

The rest of this document will explain how to create a disk using low level tools – the same method used to create the prototype disks that we have published.

1 Authoring Tools

If you're still with me, then you've decided to master your own Lib-Ray disk using low-level tools. So I will assume you are either somewhat skilled with Linux and the command line, or you can learn what you need to know elsewhere. You will need the following tools (Most of these can be installed using the package system for your preferred GNU/Linux system. I use Debian, but any modern GNU/Linux system should do fine):

- any plain text editor (such as Vim, Emacs, or Kate)
- the Chromium web browser (for testing)

- `oggz` (a command line utility for working with Ogg multimedia streams)
- `png2theora` (a command line utility provided with the libtheora packages)
- `kateenc` (a command line utility provided with libkate)
- `flac` (a command line utility for working with FLAC lossless audio files)
- `audacity` (for converting audio formats)
- `oggenc` (a command line utility for creating Ogg Vorbis streams)
- ImageMagick (a command line based image processing tool)
- `iconv` (a useful tool for fixing encodings, used on subtitle files)
- a scriptable shell environment (examples will use `tcsh` loops for doing batch jobs, but you could use `bash` if you're more familiar with it or some other scriptable shell)

On a Debian GNU/Linux 6.0 "Squeeze" system, you can install all of these with this command:

```
    # apt-get install vim-gtk chomium-browser oggz-tools libtheora libtheora-dev
libkate1 libkate-tools flac vorbis-tools audacity imagemagick iconv tcsh
```

(Of course, here I've picked Vim GTK as the text editor, but you should pick your favorite. I actually used Kate from the KDE suite to do most of the editing on the example "Sintel" disk).

In addition, you will need a copy of the "`libray.js`" Javascript library from the lib-ray.org site (or you can copy one from an existing Lib-Ray disk). This contains the API calls used in the menus. You could theoretically write your own version, but for interface and dependency reasons, it's better to use the existing one to get the most predictable results. This library is actually very small and simple – it's just a collection of convenient scripts for storing the setup menu information and passing it along to the videoplayer.

You'll also find it useful to have a black-filled 1920x1080 image. This is easy to create in ImageMagick's interactive GUI application or in other image-processing applications, such as Gimp. We'll refer to this as "`black_1920x1080.png`" when we use it.

You will also need to collect the following materials for your film:

- a PNG stream (i.e. a collection of numbered PNG images representing each frame of your film). This is huge, obviously, but it is the simplest way to be sure you are working with a pure, un-modified version of the film without processing artifacts.
- the completely-processed soundtrack in some lossless audio format such as WAV or FLAC. This could be multiple files representing separate audio channels (as with surround sound) or it could be a single multiplexed file (such as multi-channel or stereo FLAC or WAV). Of course, you may want to provide several alternate versions or commentar tracks, so you should collect these as well.

- subtitles, provided in SRT format (this is the most common format for distributing subtitles on the web, and is a very simple text format)
- a collection of images and other artwork you want to use in your Lib-Ray menus
- If you want a sound loop on the main (or other) menu pages, you should create that (this can be done very nicely in Audacity)
- You should pick a free-licensed TrueType font to use for your menus (this will be distributed on the disk and used as a "webfont" by the menu pages).
- You may also be creating "special features" pages, which are essentially just web pages. You may have videos, images, sounds, or text that you want to use with these, so you'll want to collect that as well.

2 Creating the Feature Media File

The first thing we will create is the actual video stream file. This will be a complex Ogg Theora file, with many multiplexed streams to provide video, audio (including alternate audio), and subtitles. Unfortunately, in Lib-Ray version 0.2, we are limited to only 20 streams (because this is the maximum Chromium 6.0 can handle), so we will have to be careful not to exceed that limit.

2.1 Scale, Clip, and/or Letterbox the Frames

If you are lucky enough to already have your film in 1920x1080 pixel format, then you can skip this step. However, you may have produced your film at a higher or lower resolution. And of course, you may have used a different aspect ratio – 1920x1080 is a 16:9 aspect ratio. Conventional TV is (was) 4:3. And it's not uncommon for theatrical presentations to be in wider formats such as 2.35:1 (which happens to be what "Sintel" was designed for).

In the latter case, you'll have to make a choice between clipping your frames to fit the 16:9 aspect ratio, or filling in the space with black bars – popularly called "letterboxing".

Lib-Ray calls for letterbox or clipped video (rather than leaving the video in its original form and letting the player resize it), because this avoids playback problems on some players.

However you decide to handle the resizing problem, you can do the actual processing with a `tcsh` loop and ImageMagick's `convert` and `composite` utilities. So, for example, if you had 1024x768 frames (4:3 aspect ratio), you could scale and inset the frames with a loop like this (the images are assumed to have names like `0000243.png`, with the number representing the frame count).

```
$ foreach f ( *.png )
foreach?  convert $f -geometry 1440x1080 temp.png
foreach?  composite temp.png black_1920x1080.png -gravity center lb.$f.png
```

```
foreach?  end
>
```

In this example, the 4:3 video will be scaled up slightly so that it fills the screen vertically, and then black bars will be added on the sides to fill out the 1920x1080 frames. A similar script could letterbox a wide format video to match the 16:9 aspect with letterbox bars at top and bottom. In general, the letterbox bars compress very well (they're geometrically simple and don't change), so the impact on the final compressed filesize will be small.

If you want to clip the video, you'll need to use ImageMagicks "crop" feature, which would look something like this:

```
$ foreach f ( *.png )
foreach?  convert $f -geometry 2560x1080 -crop 1920x1080+320+0 clip.$f.png
foreach?  end
>
```

The result will be a new PNG stream at exactly 1920x1080 pixels, which is the input for the next section.

2.2 Create the Theora Video Stream

Once the frame images are prepared, creating the Ogg Theora stream is relatively straightforward. For Lib-Ray, we are mostly interested in getting a high-quality image, and so you should use a quality setting of 10. For Lib-Ray v0.2, though, it is recommended to leave the chroma setting at the default 4:2:0 setting, rather than using the special chroma flags, because video players have some difficulty with the non-standard formats (i.e. 4:2:2 and 4:4:4).

Assuming the source frames are named like "`lb.0000491.png`", this might be done with this command:

```
$ theora_png2theora -v 10 -f 24 -F 1 -k 24 -o my_video.ogv lb.%07d.png
```

The options are interpreted as follows:

- "-v 10" sets the quality to 10 (no specific bitrate is set)
- "`-f 24 -F 1`" sets the frame rate to 24/1 (or just 24) per second. Other common settings would be "`-f 30 -F 1.001`" for standard NTSC color video and "`-f 25 -F 1`" for the PAL frame rate. Of course, "`-F 1`" is the default, so I could've left it out
- "`-k 24`" causes every 24th frame to be a "key-frame" in the encoding – this controls the seek resolution of the file. For Lib-Ray, the seek resolution should be the nearest integer to the frame rate so that there is approximately one key frame per second
- "`-o my_video.ogv`" provides the output filename. This should be an OGV ("Ogg Video") file.
- "`lb.%07d.png`" is a printf-style format string to match the image filenames (seven digit numbers in this example)

The command is called “theora_png2theora” in Debian, and is provided by the “libtheora” package. In other distributions, it might be called just “png2theora”.

2.3 Prepare the Audio Streams

The audio is somewhat simpler, and you are probably already familiar with some of these tools. Converting separate mono FLAC files for 5.1 surround sound into a single 5.1 multi-channel FLAC is a little tricky, but it can be done by importing the tracks into Audacity and then exporting the project to FLAC with the “Use Custom Mix” option turned on. This will let you map separate tracks to separate channels in a multi-channel FLAC file.

You will also need to create Vorbis audio, which can be done with the command line tool oggenc, or you can simply export the soundtrack from Audacity into Vorbis format (which makes sense if you’ve already got Audacity open).

At minimum, for Lib-Ray v2.0, you should have one Vorbis stereo track (for Chromium compatibility) and one FLAC stereo track (for high-fidelity playback). Additional tracks are optional.

2.4 Prepare the Subtitles

For subtitles, you’ll need to process the SRT subtitles. You may need to use other utilities to ensure the encoding is correct (I used Mozilla to view and test the SRT subtitles for the “Sintel” project. Many were in Windows-specific encodings). The command line utility “iconv” is a handy tool which can convert many encodings into UTF-8, which is what we’ll want.

Once the SRT files are clean, converting them to Ogg Kate is simple:

```
> foreach lang ( af ar bg bn cz da de en eo es fi )
foreach?  echo $lang
foreach?  kateenc -t srt -c SUB -l $lang -o ../../OGG/sintel_$lang.ogg sintel_$lang.srt
foreach?  end
>
```

You’ll note that in this case, we don’t loop through the file names, but rather through the languge codes, which we then use to set the filenames. The languages are given with two-character ISO-639 codes. This becomes part of the filename, but is also embedded within the stream to identify it to the player.

You can create as many subtitles streams as you want, but the ones converted to Ogg Kate for inclusion in the video file will be subject to the 20 stream limit: so assuming that you provide only the video and the vorbis and flac soundtracks, you will be limited to 17 subtitle tracks embedded in the file.

You should save the SRT files, because they will be used directly as soon as Chromium is patched to handle subtitle playback correctly. The Ogg Kate subtitles we are creating here will be combined with the media file and will be readable by standalone players like VLC.

2.5 Merge the Streams

Once all of the streams are available, it's a simple matter to combine them into one multiplexed stream:

```
    $ oggz merge my_video.ogv my_vorbis.ogg my_flac.ogg sub_en.ogg sub_fr.ogg sub_es.ogg
-o my_muxed_video.ogv
```

And that's pretty much it. You should now have a video file with multiplexed audio and subtitles.

3 Create the Menus

Next you'll create some HTML5 menus. This is just HTML with some special design considerations and syntax requiremets set by the Lib-Ray standard. Lib-Ray requires the following pages:

- `index.html` (in the top-level directory of the disk) - This is the "main menu" page. This is the only page whose name cannot be changed.
- `Menu/feature.html` - This is the page which contains the video itself, and provides control over it from the HTML5 menus. The actual file name can be different, if it makes more sense, though "feature.html" is the recommended choice for a single feature based Lib-Ray disk.

The following pages are strongly recommended, and the tools for providing them are part of the `libray.js` Javascript library:

- `Menu/scenes.html` - The scene selection menu. Calls to the Lib-Ray "`play()`" function implement the links
- `Menu/audio.html` - Control over the audio track selection. Calls to the Lib-Ray "`setAudio()`" function implement the links (not yet supported by viewers)
- `Menu/subtitle.html` - Control over subtitle tracks. Calls to "`setSubtitle()`" implement this (also not yet supported by viewers)

Variations are possible: for example, there could be more than one "feature" page, such as: "`episode01.html`", "`episode02.html`", etc. Or the "audio" and "subtitle" pages could be merged into a single "language" or "setup" page (as is common with DVDs).

3.1 Design Considerations for Lib-Ray Menu Pages

Although Lib-Ray menus are technologically implemented as HTML pages, they are not "web pages", and there are a number of good reasons why you should not design them the way you would a web page. Consider:

- No download penalty – bandwidth is effectively unlimited. Feel free to include full-screen background images

- Composition and rendering time is just as bad as ever. Tricks which speed up webpages by drawing more things locally will probably render more slowly than just using an image
- The screen size is fixed. Lib-Ray assumes a 1920x1080 pixel display (larger sizes may be included in later versions). It is assumed that if any scaling is to be done, the viewer should handle it. This means you don't have to design for a flexible display size
- The assumption of a "10-foot user interface" means you must use large fonts and few words. You will also have to provide alternative input methods (such as tabs and hotkeys)
- Although background music or animated backgrounds are considered tacky and impolite on the web, they are a normal part of movie menus, and Lib-Ray disks can certainly use them, via the `audio` or `video` tags, or by animating features using Javascript and CSS.

3.2 Specifics for the index.html page

The main menu page needs to have a "Play" button and access to other menus via hyperlinks. This "Play" button is simply a hyperlink to the "`Menu/feature.html`" page (playback is actually initiated automatically once that page is loaded). The "Play" button (to be compliant with Lib-Ray 0.2), must have an attribute "tabindex" and it must be set to "1". It must also have an "accesskey" attribute, with the value "p". This is necessary to make the "Play" button on the remote work as an alternate way to activate the "Play" control on a conforming Lib-Ray HTPC or player. Here's the HTML code for a typical Lib-Ray compliant "Play" button:

```
<a href="Menu/feature.html" tabindex="1" accesskey="p">Play</a>
```

Typically, of course, there will be additional markup to control the appearance of the button. For example, there may be a "style" attribute. Or there may be a "class" attribute used to assign CSS stylesheet values to the button. It may also be included in a div element or other structure used to provide the desired page layout. But the attributes above are sufficient to meet the Lib-Ray standard.

Additional links may lead to other menus: the "audio" and "subtitle" (or combined "setup") menus, for example, or to extras. These are simply hyperlinks. No special associations are required by Lib-Ray 0.2 (additional accesskey codes may be required by later revisions of the standard, so as to provide additional macros for the remote control).

3.3 The Scenes menu pages

One of the most important roles of the menu system is to give viewers a convenient way to jump directly to the scene they want to find in the movie. This is the role of the "scenes" menu (or menus). Unlike the DVD standard, the Lib-Ray standard (v0.2) does not use "chapter" positions in the video file. Instead, the scene hyperlink simply changes the playback position to the nearest minute and second and then links to "Menu/feature.html" so as to play from that position.

This is done in the menu via a Javascript link which actually calls the "play()" function from the libray.js library. This function takes two arguments: the first is the desired position in minutes, while the second is the seconds. Thus "0, 0" is the beginning of the film at 0:00, and "12:32" is "twelve minutes and thirty-two seconds" into the film.

3.4 The Audio and Subtitle pages

The "audio" and "subtitle" pages work in almost exactly the same way, so I'm going to cover them together. In both cases, the role of the menu is to set a value that will be remembered and used by the player to play back the correct track while the video is playing. Also in both cases, this is actually implemented with a Javascript link calling a function in the `libray.js` Javascript library.

The functions used are "`setAudio()`" and "`setSubtitle()`". These functions actually make no direct changes. Instead, they set values into the browser's "Session Storage" representing the user's choice. These values can be used by CSS code to indicate which link has been activated, thus enabling a feedback display.

When the feature is played, using the "`Menu/feature.html`" page, it reads these values from Session Storage and calls Javascript methods on the feature video element to honor the choice. Regrettably, these methods are not supported by any browser at the time of this writing (this is hardly surprising considering the drafts were written only a couple of weeks before now!), so the Audio and Subtitle features so provided won't actually work, and users who want to see them will have to wait for upgrades or use a workaround (such as viewing the "`feature.ogv`" video file using a standalone video player).

3.5 The feature.html page

The "`Menu/feature.html`" page acts as a frame page to control the video player embedded in the browser. The use of the frame page allows for the video player attributes to be exposed to the Javascript Document Object Model (DOM), so that the `libray.js` Javascript library can control its behavior.

This is not immediately obvious when playing in a fully-compliant browser, because the video is made to fill the entire screen, and the CSS used eliminates the scrollbars that would normally be rendered. When the browser window is also rendered fullscreen without window borders, the effect is much the same as playing a native video player in fullscreen mode.

For an ordinary single-feature video disk, there is really very little reason to make any custom changes to feature.html (except perhaps for the title), and the page is very small, so here is the entire text of a prototype (with no external tracks):

```
   <!DOCTYPE html>
   <html>
   <head>
   <title>Lib-Ray Feature Video</title>
   <meta charset="utf-8"/>
   <!-- Basic Lib-Ray library communicates menu-based control settings
to video player -->
   <script src="libray.js"></script>
   </head>
   <body onLoad="honor_videosettings();" style="overflow:  hidden; overflow-style:
move; margin:  0; padding:  0; border:  0;" >
   <video id="feature" preload="auto" autoplay="autoplay" style="margin:
0; padding:  0; border:  0;" width="1920" height="1080">
   <source src="../Media/feature.ogv" type='video/ogg; codecs="theora,
vorbis"'/>
   Video appears not to be loading.  You may want to open the <a href="../Media/feature.ogv">video
file</a> with an external player (i.e.  right click and select what player
to open the file with)
   </video>
   <a href="../index.html" id="quit" accesskey="q"></a>
   </body>
   </html>
```

Let's go over some of the elements of this file. The DOCTYPE, html, and head lines are all part of standard HTML5. The title is supposedly required by the HTML standard, so I've included a simple generic title – you are free to pick something more descriptive, such as the title of the video, if you like (in normal playback, this title won't be visible – though it may appear on the window bar if you open the document up in the browser's windowed mode).

All of the Lib-Ray disk pages use UTF-8 encoding, as the meta tag indicates.

The libray.js Javascript library is provided for use in all Lib-Ray disks and provides the basic API for menu design.

The settings on the body tag provide the "fullscreen" appearance, especially the style attribute with its "overflow" options.

The video tag uses the correct settings to automatically begin playback from the "Media/feature.ogv" file.

A little bit of warning text appears if the browser doesn't recognize the video tag for some reason. It suggests trying to view the video file with an external viewer, which is probably the best option if the browser is not compliant.

Finally, outside the video tag there is an additional anchor with no content, but an accesskey of "q", which links back to the main menu. Thus (on Chromium) a key combination of "Alt+Q" will cause playback to stop and control to jump to the main menu – which is the behavior we want for the "Stop" button on the remote, which is what "Alt+Q" should be mapped to.

The HTML5 standard allows for additional subtitle tracks to be added from external files.

3.6 Assemble Menus and Video

All of the menus and the multimedia video feature need to be assembled onto the disk image. Typically, you'll create a directory to represent the disk. Inside it you'll need to create three folders: "`Menu`", "`Media`", and "`Extra`". All of the "disk menu" pages will go into the `Menu` folder (except for the main disk menu, which will be "`index.html`" in the disk's root directory).

Supporting files for the menus, including CSS stylesheets, image buttons, and the "`libray.js`" Javascript library should also be kept in the "`Menu`" folder. Menu sound loops are a gray area, but they should normally go in the "`Media`" folder (you can refer to them with a relative link, like this: "`../Media/audio_menu_soundloop.flac`").

All of the video features (and featurettes) will go into "`Media`", with appropriately descriptive names ("`feature.ogv`" is the recommended name for the main feature on the disk, if there is only one. Other names such as "`episode01.ogv`" may be used to distinguish multiple features, if you prefer).

If you have external SubRip SRT subtitle files to include on the disk (e.g. to escape the 20-stream limit in the current version of Chromium), then these should be stored in the `Media` directory, too. The SRT files should use the same filestem as the feature they apply to with an underscore and then an ISO language code, followed by the "`.srt`" extension. So, for example, for the main feature's Spanish subtitle track, the filename should be: "`feature_es.srt`". The reason for this naming convention is that some video viewers (like VLC) will automatically detect and load such subtitle tracks when loading the video file.

It's a matter of author discretion whether "Disk Extras" should be included in the "`Menu`" directory or the "`Extra`" directory, but you should understand what the choice will mean to the viewer in terms of "Lib-Ray Compliant" players.

To be considered compliant, a player must completely support the disk menu system under "Menu". However, you must not include any links to external resources in the disk menus – in other words, they must be self-contained on the disk (of course, you can have a links into the other directories on disk, including the "`Extra`" directory). A compliant player (such as a Home Theater PC) can, however, elect to completely ignore all content in the "`Extra`" directory and/or refuse to load external links. This is essentially a data security consideration for the viewer, as well as a matter of convenience for offline use.

On the other hand, general purpose desktop computers will be able to access the "`Extra`" folder's contents in the normal way, and the content there is unconstrained by the above rules: you can provide links to external web resources, as well as use non-HTML content (such as PDFs – just like this one on the prototype disk – or even software packages).

4 Write Metadata

Now that we have a watchable disk. First, load the cover art in an appropriate image editor and export an image at least 750x1000 pixels in size (750 wide and 1000 high). This image can be a JPEG ("`cover.jpg`") or a PNG ("`cover.png`").

Next we'll need to create the metadata file. This can be very simple (here are the contents of the "meta.cnf":

```
[Lib-Ray]
# Mandatory fields
```

```
LibRayVersion:  0.2
LibRayID: 1 # ID for the producer (sign up for this with lib-ray.org)
DiskID: 2 # ID for this disk (you assign this number)
Cover:  cover.jpg
Title:  Sintel

[Copyright]
Date:  2010
RightsOwner:  Blender Foundation
RightsURL: www.blender.org
License:  Creative Commons Attribution Only, Version 3.0, Unported
LicenseURL: http://creativecommons.org/licenses/by/3.0
```

This file follows the common "INI" or "conf" format. It is divided into labeled sections, and each section contains a series of key-value pairs. I hope to have a more well-defined set of keywords in the final Lib-Ray standard, but for the prototype, we have just these few.

The mandatory fields include "LibRayVersion" which for this prototype should read "0.2". This is the version of the Lib-Ray standard that the disk is meant to conform to – players will be able to use this field to provide backward compatibility in the future. The field "LibRayID" is meant to contain a unique publisher ID registered with `lib-ray.org`.

No formal process for assigning these IDs has been established yet, but you can email me (Terry Hancock), and I'll give you a number and keep a record to avoid collisions (these numbers will be above 1000). Otherwise you can pick a number in the 100-1000 range (reserved for public experimental disks. 0-100 are reserved for use by `lib-ray.org`). Once you are assigned a number (or pick one), you should use that on all of your disks.

The next field, "`DiskID`" is a unique alphanumeric identifier that you assign to the disk. You can use numbers, letters, dashes, or underscores. There should be no spaces or other characters. The length is your decision, but can't exceed 256 characters (compliant players can ignore IDs longer than this). The idea of course is that you can use whatever ID system works for you. Here, I've simply used "2" to indicate that this is the second prototype disk. Combined with your assigned `LibRayID`, this forms a globally-unique identifier for the disk.

The "`Cover`" field, of course, refers to the cover image. This currently must be either "`cover.jpg`" or "`cover.png`".

The "`[Copyright]`" section is not mandatory for Lib-Ray compliance, but will probably be used a lot. The "`Date`" field provides the copyright year. The "`RightsOwner`" is whoever owns the copyright for the work (not necessarily the creator, so there can be a separate "`Creator`" field). The "`RightsURL`" optionally provides a URL for the rights-holder. The "`License`" field identifies the license under which the work is released. This can use common license names, or it can say "`All Rights Reserved`" for conventional proprietary publications. Another field, "`LicenseURL`" can provide a link to a complete copy of the license.

If you choose, you can provide more complex metadata in XML-RDF format, using conventional controlled vocabularies such as Dublin Core in another top-level file, "`meta.rdf`", and referred to by an extra keyword "`MetadataRDF`" in the

"`[Lib-Ray]`" section, like this:

```
        MetadataRDF: meta.rdf
```

However, the "`meta.cnf`" file is still required to be Lib-Ray compliant. Players are not required to read this data, but may do so for enhanced functionality.

5 Burn the Disk

Finally, of course, you'll want to put the data on a disk. This can be done with a variety of burner tools. For "Sintel", I put the data on a single-layer DVD-R.

For full-length movies, it's unlikely that even a double-layer DVD-R will suffice. For those movies, I recommend using a Blu-Ray "BD-R" disk. These are dye-based optical disks (like the DVD-R and CD-R) and are not subject to the same kinds of DRM problems. You can burn these on your own system or have the disks duplicated commercially (there are companies that can do this for you).

Either way, of course, you'll need to make an ISO. I used "K3B" to do that.

First, I started a "New Data Project". Then I moved the three top level files and three top level folders into it. I gave the volume a descriptive name ("SINTEL_LibRay_v0.2"), and then clicked on the "Burn" button. From there, it's necessary to select "Only create image" unless you just want to burn it immediately to a disk. You can determine where the ISO will be stored by using the "Image" tab. It's very important to go to the "Filesystem" tab and select "UDF" – this is the high volume format preferred for data DVDs, and it allows files to be over 2 GB (the "`feature.ogv`" file is very likely to be larger than this!).

Then click "Burn" and wait for it to finish – now you have Lib-Ray ISO. You can of course also opt to burn the data straight to a DVD-R disk (larger movies may require larger media, such as BD-R disks).

To make a "Lib-Ray Flash" movie instead, you can simply copy the same file structure onto a Flash card or USB stick of sufficient volume (the Sintel ISO is about 2.1 GB).

"That's all!"