



# **FLOSS**

MANUALS

# Table of Contents

<b>Introduction.....</b>	<b>1</b>
Patent Encumbered Video.....	1
A Better Way.....	1
<b>About This Manual.....</b>	<b>2</b>
1. Register.....	2
2. Contribute!.....	2
3. Chat.....	3
4. Mailing List.....	3
<b>What is Digital Video?.....</b>	<b>4</b>
<b>What is Theora?.....</b>	<b>5</b>
History.....	5
The Web.....	5
Patents and Copyright.....	5
Other Free Video Formats.....	6
<b>Codecs.....</b>	<b>7</b>
Quality.....	7
Bitrate and quality.....	7
Codecs and quality.....	8
Theora and quality.....	8
<b>Containers.....</b>	<b>9</b>
The difference between containers and codecs.....	9
<b>Playing Theora.....</b>	<b>10</b>
Integrating Theora.....	10
<b>VLC.....</b>	<b>11</b>
Installing.....	11
Playing Video.....	11
Basic Playback Controls.....	12
<b>Miro.....</b>	<b>13</b>
Installing.....	13
Playing A Video.....	13
<b>Introduction.....</b>	<b>15</b>
<b>HTML5 Video.....</b>	<b>16</b>
Basic Syntax.....	16
Parameters.....	16
Using your own controls / player skin.....	17
Manual Fallback options.....	17
Javascript Based Players.....	18
Support in Browsers.....	18
<b>Hosting on your own site.....</b>	<b>19</b>
Mime Types.....	19
oggz-chop.....	19
Allowing Remote Access.....	20
Serving Videos via a Script.....	20

# Table of Contents

<b>Hosting Sites.....</b>	<b>22</b>
TinyVid.tv.....	22
The Video Bay.....	22
Archive.org.....	23
Dailymotion.....	24
Wikimedia Commons.....	25
<b>Introduction.....</b>	<b>27</b>
Video Quality, Bit-Rate and File Size.....	27
Video Resolution and Frame-Rate.....	27
Clipping the Frames in the Video.....	28
Video-Audio Synchronization.....	28
Keyframe-Interval.....	28
<b>Firefogg.....</b>	<b>29</b>
Installation.....	29
Encoding your first video.....	30
Advanced Encoding Options.....	32
Custom Settings.....	33
Other Uses of Firefogg.....	34
<b>Encoding with VLC.....</b>	<b>35</b>
Using the GUI.....	35
Advanced Options.....	36
Encoding from the Command Line.....	37
Recommended Encoding Options.....	37
Why not to use the GUI.....	38
<b>ffmpeg2theora.....</b>	<b>39</b>
Basic Usage.....	39
Adding Parameters.....	39
Advanced Options.....	39
--sync.....	40
--keyint <N> .....	40
--framerate <N>.....	40
--starttime <N> --endtime <M>.....	40
Two-Pass Encoding.....	40
Why Two-Pass Encoding.....	40
Using Two-Pass Mode.....	40
<b>Thoggen.....</b>	<b>42</b>
Selecting the DVD Titles to Encode.....	42
Configuring the Encoding Process.....	43
A Note on Cropping.....	43
<b>Subtitles.....</b>	<b>45</b>
Finding Subtitles.....	45
Subtitle File Formats.....	45
Editing SRT files.....	46
Creating subtitles from scratch.....	46
<b>Distribution.....</b>	<b>47</b>

# Table of Contents

<b>Embedding Subtitles.....</b>	<b>48</b>
Example commands for subtitle embedding.....	48
Adding subtitles to an existing video.....	49
<b>Playing Subtitles.....</b>	<b>50</b>
VLC.....	50
Play subtitles on a DVD disk.....	50
Play subtitles in Matroska files.....	51
Play External Subtitles.....	52
<b>Publishing.....</b>	<b>54</b>
Hosting external subtitle files along with video on a web server.....	54
Hosting video with embedded subtitles on a web server.....	55
<b>Web Video Accessibility.....</b>	<b>56</b>
Accessibility user groups.....	56
Accessible HTML5 video controls.....	56
Providing video accessibility data.....	57
Publishing accessible video on the Web.....	57
<b>What is streaming?.....</b>	<b>59</b>
Varieties of Streaming Services.....	59
Live Streaming.....	59
Delivery.....	59
Streaming Servers.....	59
<b>Icecast.....</b>	<b>61</b>
<b>TSS.....</b>	<b>63</b>
Installing TSS.....	64
Installing on 64 bit.....	64
Using TSS.....	66
<b>Streaming with ffmpeg2theora.....</b>	<b>70</b>
Getting Ready.....	70
Set-Up.....	70
<b>Streaming with VLC.....</b>	<b>72</b>
Streaming Server Requirements.....	72
Choosing Video File to Stream.....	72
Stream Settings.....	74
<b>Saving a Theora Stream.....</b>	<b>75</b>
Extracting parts of a video.....	75
<b>Introduction.....</b>	<b>76</b>
<b>PiTiVi.....</b>	<b>77</b>
Installing.....	77
Importing Video.....	77
Adding Clips.....	78
Playback.....	79
Moving Clips.....	80
Cutting Clips.....	80

# Table of Contents

<b>PiTiVi</b>	
Deleting Clips.....	80
Export to Ogg Theora.....	81
<b>Introduction.....</b>	<b>82</b>
Installing the Ogg Video Tools.....	82
Installing from Source.....	82
<b>Create Thumbnails.....</b>	<b>84</b>
Creating a Series of Thumbnails.....	84
Creating Thumbnails with Fixed Height or Fixed Width.....	84
.....Advanced Functionality.....	85
<b>Creating a Slideshow.....</b>	<b>86</b>
Command Line Options.....	86
Adding Sound to a Slideshow.....	87
Adding a Fixed Starting Picture.....	88
A Script for Everything.....	88
<b>Creating a Video Preview.....</b>	<b>90</b>
Adding a Play Button.....	90
<b>Introduction.....</b>	<b>91</b>
Lossless Video Editing.....	91
Tools.....	91
<b>A Bit of Theory.....</b>	<b>93</b>
Anatomy of a Theora Video.....	93
Demultiplexing.....	93
Page Timestamps.....	93
Encapsulating Codec Data.....	94
Encapsulating Theora Video Data.....	94
Encapsulating Vorbis Audio Data.....	94
Ogg Skeleton.....	94
<b>Splitting a Video File.....</b>	<b>95</b>
<b>Cutting an Ogg File.....</b>	<b>96</b>
Using oggCut.....	96
Video/Audio synchronization issues with oggCut.....	96
Using oggz-chop.....	96
<b>Join Streams.....</b>	<b>98</b>
<b>Ogg File Concatenation.....</b>	<b>99</b>
The Process.....	99
Using cat.....	99
oggCat and oggz-chop.....	99
Synchronizing issues with oggCat.....	100
<b>OggResize.....</b>	<b>101</b>
Examples.....	102

# Table of Contents

<b>Analysing Ogg Files.....</b>	<b>104</b>
Information about an Ogg file.....	104
Analysing an Ogg file in detail.....	105
Tips and tricks.....	106
Getting the Duration of a Stream.....	106
Creating a sound byte from a portion of a video.....	106
Adding an embedded text subtitles stream.....	106
Getting a list of all packets in a stream in a very compressed way.....	106
<b>Other Interesting Things.....</b>	<b>108</b>
Video and SVG.....	108
Video and CSS Transforms.....	108
Video and Subtitles.....	108
Replace background with image.....	108
<b>Glossary.....</b>	<b>109</b>
<b>License.....</b>	<b>111</b>
<b>Authors.....</b>	<b>112</b>
<b>General Public License.....</b>	<b>120</b>

# Introduction

The web grew as fast as it did because of its neutral design. Open standards, free software and transparent technologies have empowered a lot of this development. Because of this open environment, web developers can easily create web pages, software developers can build upon existing projects to extend what we can do on the net, and users can browse the web, send email, and use a wide range of applications.

With online video however, almost all of the basic technologies for its creation and playback are not free or transparent. For example, if a software developer makes a new video player, they must pay royalties to the companies that hold certain patents in order to distribute it legally. These patents (like most software patents) lock down extremely basic ideas. For example: one company has patented the idea of storing pieces of an image from left to right, top to bottom! Taken together, the thousands of patents on video techniques stifle the emergence of new ways of distributing and interacting with media.

## Patent Encumbered Video

For those concerned with the dominance of private interests over public life, this is an obvious problem: how could the basic ideas necessary to develop software be controlled by a small number of corporations? It creates very concrete problems as well: the global movement for free software cannot include support for many video formats with its products. And because patents make it costly for developers to distribute their own video players, developers depend on a few products from Microsoft, Apple, and Adobe. On the web, over 98% of online video is delivered using Flash.

The problems are not just within the realm of software development - if you create or publish video online then patent-encumbered technologies affect you directly. Patent holders have the power to extract fees from content producers and MPEG LA, which represents patent holders of the popular MPEG video technologies, charges license fees from television broadcasters, DVD distributors, and others. At this moment they don't charge for online distribution, but at the end of 2009 they are expected to announce new royalty terms, and these new terms could threaten independent publishers of online video.

## A Better Way

Now, however, we have an online video technology that anyone is free to use, study, improve, and distribute without needing permission or paying fees. This technology is called Ogg Theora (or just 'Theora'). Some parts of Theora *are* patented, but the owners of those patents have granted a permanent, irrevocable, royalty-free patent license to everyone. Theora carefully avoids any patents held by traditional patent holders: to get around the ridiculous patent of image storing mentioned above, Theora stores video image information from bottom to top instead of top to bottom!

Recognizing that Theora is a crucial ingredient for the freedom of our internet, Mozilla, Opera and Google have announced support of Theora video for future or current releases of their browsers. This means that millions of users will be able to watch Theora videos using their browser, without the need for extra software. The work of the free software community, with support from Mozilla, Wikipedia and others, has brought Theora to the same level of quality as state-of-the-art video technologies.

There are other important projects with similar goals, like *Dirac*, an effort spearheaded by the BBC. But the exciting thing about Theora is that it's here now, supported by popular tools, and ready for mass adoption. By learning how to use Theora and involving it in your work, you can help make the web more exciting and more free.

# About This Manual

This manual was written during a 5 day Book Sprint in the Haus der Kulturen der Welt, Berlin (August 10-15, 2009).



Present were :

Jan Gerber

Homes Wilson

Susanne Lang

David Käthling

Jörn Seger

Plus many people helping online. Many thanks also to Leslie Hawthorn and Google for supporting this sprint, and to the Berlin Summercamp. This book is an ongoing effort...please feel free to improve it!

## 1. Register

Register at FLOSS Manuals:  
<http://en.flossmanuals.net/register>

## 2. Contribute!

Select the manual <http://en.flossmanuals.net/bin/view/TheoraCookbook/WebHome> and a chapter to work on.

If you need to ask us questions about how to contribute then join the chat room listed below and ask us! We look forward to your contribution!

For more information on using FLOSS Manuals you may also wish to read our manual:  
<http://en.flossmanuals.net/FLOSSManuals>



## 3. Chat

It's a good idea to talk with us so we can help co-ordinate all contributions. We have a chat room embedded in the FLOSS Manuals website so you can use it in the browser.

If you know how to use IRC you can connect to the following:

server: irc.freenode.net

channel: #flossmanuals

## 4. Mailing List

For discussing all things about FLOSS Manuals join our mailing list:

<http://lists.flossmanuals.net/listinfo.cgi/discuss-flossmanuals.net>

# What is Digital Video?

Video is a series of images that appears as an image in motion. The first "videos" (films) were literally a series of photographs, illuminated one-by-one at a rate fast enough to trick the human eye. Digital video can be thought of as a series of images but usually the reality is more complex.

While it is possible to store videos as a series of still images (frames), this is a somewhat wasteful approach as a lot of information must be stored. Surely there must be a better way! Well, when we look at frames in a movie we quickly see that only parts of a moving image change from one frame to the next while the rest of the image stays exactly the same. If the image is of someone walking, for example, perhaps they move while the background stays the same. So why not simply describe what changes from one frame to the next, since a little description is much less information than a whole image itself?

In fact, that's what digital video does. And in a world with limited disk space and network connections, simple ideas like this one can let you store hundreds of videos on your computer, instead of a handful or download a video in minutes, instead of hours.

In addition to this technique, digital video employs other methods to reduce the amount of data in a video. Digital video often uses tricks such as describing regions of similar color instead of describing each point one at a time -similar ways are used for describing images. Or they carefully describe certain parts of the image where lots of detail or movement is happening, for instance the part you're probably staring at, while giving less attention to the boring stuff at the edges. These ideas are not new, but they are improving rapidly with exciting consequences for producers of online video.

# What is Theora?

Theora is a video technology for creating, editing, manipulating, and playing video. This type of technology is often referred to as a video **format** or **codec** (a portmanteau of *coder-decoder*). Theora is a free video format, meaning that anyone is free to use, study, improve, and distribute it without needing permission. Some parts of Theora *are* patented, but the owners of those patents have granted a permanent, irrevocable, royalty-free patent license to everyone.

Because distribution and improvement of Theora is not limited by patents, it can be included in free software. Distributions of GNU/Linux-based operating systems, such as Ubuntu, Debian GNU/Linux, or Fedora, all include Theora "out-of-the-box". And free software web browsers like Firefox and Chrome support Theora. If we consider the six major *usage share of web browsers* statistics of July 2009, approximately 25% of Internet users across all of these statistics are using Firefox and 2.5% are using Chrome as their browser. This means that every day a huge number of people are using software capable of playing Theora video.

## History

Theora is based on an older technology called **VP3**, originally a proprietary and patented video format developed by a company called On2 Technologies. In September 2001, On2 donated VP3 to the Xiph.Org Foundation under a free software license. On2 also made an irrevocable, royalty-free license grant for any patent claims it might have over the software and any derivatives, allowing anyone to build on the VP3 technology and use it for any purpose. In 2002, On2 entered into an agreement with the Xiph.Org Foundation to make VP3 the basis of a new, free video format called Theora. On2 declared Theora to be the successor to VP3.

The Xiph.Org Foundation is a non-profit organization, that focuses on the production and mainstreaming of free multimedia formats and software. In addition to the development of Theora, they developed the free audio codec Vorbis, as well as a number of very useful tools and components that make free multimedia software easier and more comfortable to use.

After several years of beta status, Theora released its first stable (1.0) version in November 2008. Videos encoded with any version of Theora since this point will continue to be compatible with any future player. A broad community of developers with support from companies like Redhat and NGOs like the Wikimedia Foundation continue to improve Theora.

## The Web

Support for Theora video in browsers creates a special opportunity. Right now, nearly all online video requires Flash, a product owned by one company. But, now that around 25% of users can play Theora videos in their browser without having to install additional software, it is possible to challenge Flash's dominance as a web video distribution tool. Additionally, the new HTML5 standard by the W3C (World Wide Web Consortium) adds another exciting dimension — an integration of the web and video in new and exciting ways that complement Theora.

## Patents and Copyright

The world of patents is complicated, leaving plenty of room for Theora's competitors to spread fear, uncertainty, and doubt ("FUD") about its usefulness as a truly free format. In essence, Theora is free. It is free for you to use, change, redistribute, implement, sell or anything else you may like to do with it. But it's important that supporters of free formats understand the questions that arise.

One commonly spread fear is that Theora infringes on *submarine patents* — patents nobody knows about yet that the makers of Theora never had the authority to use. However it is also true that *all* modern software could infringe on submarine patents — everything from Microsoft Word to the Linux kernel. However millions of people and entire industries still use these tools. In other words, submarine patents are a problem for the software industry as a whole, however, this doesn't mean that the software industry should stop developing software.

It is also important to note that in a worst-case scenario, even if a submarine patent "emerged", Theora could probably work around it; as this sort of thing happens all the time. Large organizations like the Mozilla Foundation and Wikipedia have examined this issue and have come to this same conclusion.

The software produced by Xiph.org is also subject to copyright, and made available under free software licenses. Xiph.org provides code so that anyone can include it in any application. Xiph.org also provides a set of tools for working with Theora files. This means you can study, modify, redistribute and sell anything you make using Theora or any of the tools provided with it.

## Other Free Video Formats

It is worth noting that there is another project creating a royalty-free, advanced video compression format. It is called Dirac. Originally created by the BBC Research department, Dirac will in the future try to cover all applications from Internet streaming to Ultra-high definition TV and expand to integrate with new hardware equipment technologies. Nevertheless, Ogg Theora lends itself extremely well to online (streaming) video distribution, whereas Dirac will likely become a better choice for sharing video files that are of high-definition footage.

# Codecs

To work with digital video it often helps to know a little bit about the technology, so lets look at some of the basic concepts behind digital video.

A **codec** is a mathematical formula that reduces the file size of a video or audio file. Theora is known as a *video* codec since it works exclusively with video files.

When a codec reduces a file size of a video file, it is also said to be **compressing** the file. There are two forms of compression that are of interest here - **Lossless** and **Lossy Compression**.

1. **Lossless compression** - This is the process of compressing data information into a smaller size without removing data. To visualise this process imagine a paper bag with an object in it. When you remove the air in the bag by creating a vacuum the object in the bag is not affected even though the total size of the bag is reduced.
2. **Lossy compression** - Sometimes called 'Perceptual Encoding', this is the process of 'throwing away' data to reduce the file size. The compression algorithms used are complex and try to preserve the qualitative perceptual experience as much as possible while discarding as much data as necessary. Lossy compression is a very fine art. The algorithms that enable this take into account how the brain perceives sounds and images and then discards information from the audio or video file while maintaining an aural and visual experience resembling the original source material. To do this the process follows Psychoacoustic and Psychovisual modeling principles.

## Quality

The quality of digital video is determined by the amount of information encoded (**bitrate**) and the type of video compression (**codec**) used. While there are some codecs that can be considered to be more advanced than Theora, the difference in perceived quality is not significant.

## Bitrate and quality

Since digital video represents a moving image as information, it makes sense that the more information you have, the higher the quality of the moving image. The bitrate is literally the number of bits per second of video (and/or audio) used in encoding. For a given codec, a higher bitrate allows for higher quality. For a given duration, a higher bitrate also means a bigger file. To give some examples, DV cameras record video and audio data at 25Mbit/s (a **Mbit** is 1,000,000 bits), DVDs are encoded at 6 to 9 Mbit/s, internet video is limited by the speed of broadband connections: many people have 512kbit/s (a **kbit/s** means 1,000 bits delivered per second) or 1Mbit/s lines, with 16Mbit/s connections becoming more common recently. Right now, around 700kbit/s is commonly used for videos embedded on web pages.

There are many reasons to want a lower bitrate. The video may need to fit on a certain storage medium, like a DVD. Or you may want to deliver the video fast enough for your audience, whose average internet connection speed is limited, to be able to watch it as they receive it.

Different kinds of video may require different bitrates to achieve the same level of perceived quality. Video with lots of cuts and constantly moving camera angles requires more information to describe it than video with many still images. An action movie, for example, would require a higher bitrate than a slow moving documentary.

Most modern codecs allow for a **variable bitrate**. This means that the bitrate can change over time in response to the details required. In this case, a video codec would use more bits to encode 10 seconds of quick cuts and moving camera angles than it would use to encode 10 seconds of a relatively still image.

## Codecs and quality

Codecs reduce the necessary bitrate of a media file by describing the media in clever, more efficient ways. Video codecs describe the changes between one frame and the next, instead of describing each frame separately. Audio codecs ignore certain frequencies that the human ear doesn't notice. Just as simple techniques can dramatically reduce the bitrate and size of the file, more sophisticated techniques can reduce it even more. This is how some codecs can be considered superior to others.

When codecs use complicated mathematical techniques to encode video, you need a powerful chip to decode that video quickly enough for playback. This is a reason why sometimes it's best to use a simpler codec. Video encoded using state-of-the-art tricks may be unwatchable on an old computer, for example. Or it sometimes might be best to use an older, simpler video codec (as DVDs do) because the hardware required to play it will be cheaper.

## Theora and quality

Thanks to recent work by the Theora community, Theora achieves a similar level of quality to other modern codecs like h.264, the patent-encumbered codec used by Apple, Youtube, and others. This can be a matter of some controversy, and there are reasons to consider h.264 technically superior in quality to Theora. But the best way to decide is to see for yourself.

These sites have side-by-side comparisons between Theora and h.264:

<http://people.xiph.org/~greg/video/ytcompare/comparison.html>

<http://people.xiph.org/~maikmerten/youtube/>

# Containers

A **container** or **wrapper** is a file format that specifies how different streams of data can be stored together, or sent over a network together. It allows audio and video data to be stored in one file and played back in a synchronised manner. It also allows seeking in the data, by telling the playback software where the audio and video data is for certain points in time.

In addition to audio and video, containers may provide meta data about the data they contain, including the size of the **frames**, the **framerate**, whether the audio is in **mono** or **stereo**, the **sample rate**, and also information about the codecs used to encode the data.

When you play a digital movie that has sound, your player is reading the container, and decoding the audio and video using separate codecs. Theora video is usually stored or streamed together with **Vorbis** sound in the **Ogg** container, but it can be stored in other containers too. Matroska (.mkv) is another format people use for Theora video.

## The difference between containers and codecs.

The three letter extension at the end of the file name refers to the container, not the codec. People often get confused about this. When a file ends in .mp4 or .avi, those are containers that could contain several different combinations of audio and video streams. Certain containers don't work with certain codecs, and certain codecs work best with certain containers. But you can't tell for sure what codecs a video file requires by looking at the file extension.

# Playing Theora

To play Ogg Theora videos you need a video player that supports Ogg Theora playback. Often to playback some types of video you need to install obscure software which can be very frustrating and time consuming. Fortunately, several video players can play Ogg Theora without the need to install anything else. The two easiest players to use are notable because they work the same across all of the major Operating Systems (GNU/Linux, Mac OS X, Windows).

**VLC** is a free software video player that plays many different types of video files, including Theora. You can get it here online at <http://www.videolan.org/vlc>

**Miro** is another video player that supports Ogg Theora video (<http://getmiro.com>).

If you don't have either or don't want to install them, you can also use the **Firefox** web browser versions 3.5 and later (<http://getfirefox.com>) as a Theora viewer.

As a rule of thumb, if the video is on your desktop, use VLC or Miro. If the video is already on the web, use Firefox 3.5 or later.

## Integrating Theora

If you want to use Theora with other softwares such as Windows Media Player or QuickTime Player, you need to install components or filters that will enhance the functionality of Theora. This doesn't apply to you if you use GNU/Linux, since Theora is natively supported in most distributions and plays with Totem and other GStreamer based applications.

For Windows and Mac OS X you can add full functionality and support of Theora to all QuickTime based applications, such as the QuickTime player itself, but also iTunes or iMovie. All you need to do is install the Xiph QuickTime Components (XiphQT) that can be downloaded from the Xiph.Org Website: <http://xiph.org/quicktime/download.htm>

The other useful filter a Windows user might be interested in is the Directshow Filter. It is also offered from the Xiph.Org Foundation and adds encoding and decoding support for Ogg Vorbis, Speex, Theora and FLAC for any Directshow application, such as Windows Media Player. You can also download the filters from the Xiph.Org Website: <http://xiph.org/dshow/>



# VLC

VLC (which stands for *VideoLAN Client*) is an excellent tool for playing video and audio files. It is **free software**, it plays a wide range of formats including Theora, and it runs on a variety of platforms. If you use Windows, Mac OS X, or GNU/Linux (eg. Ubuntu) then VLC is a great option for you. VLC also works the same across each Operating System so if you know how to use it in Windows (for example) you know how to use it under Ubuntu. VLC's flexibility and reliability make it one of the most popular free software video tools.

## Installing

VLC is a desktop application that you need to download and install. Installation steps will vary depending on what platform you're using (GNU/Linux, Mac OS X, or Windows).

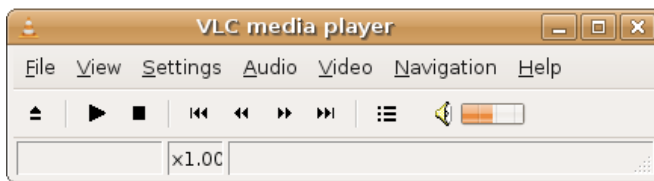
This page has links to download VLC, and instructions for various platforms:

<http://www.videolan.org/vlc/>

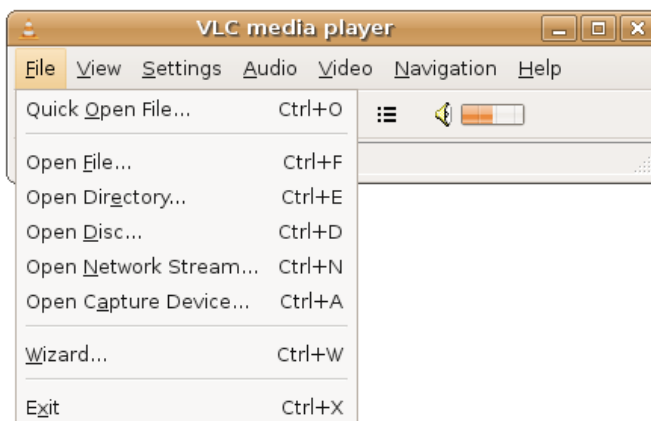
There is also a good manual about how to use and install VLC linked from the FLOSS Manuals website.

## Playing Video

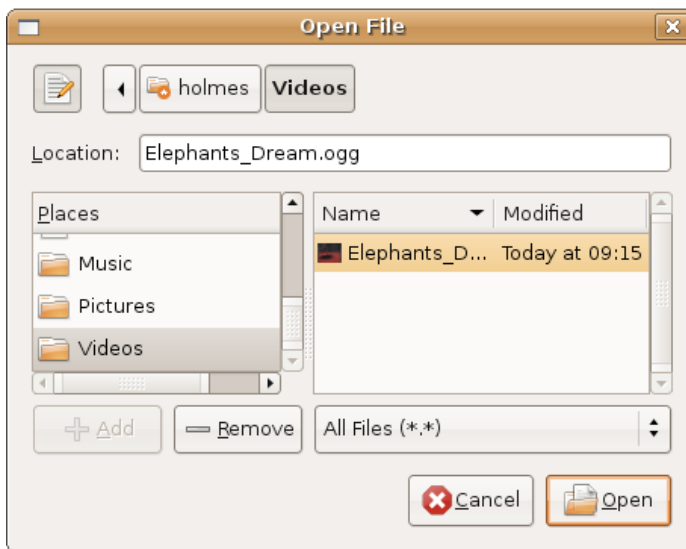
1. Run VLC. You will see a window that looks more or less like this:



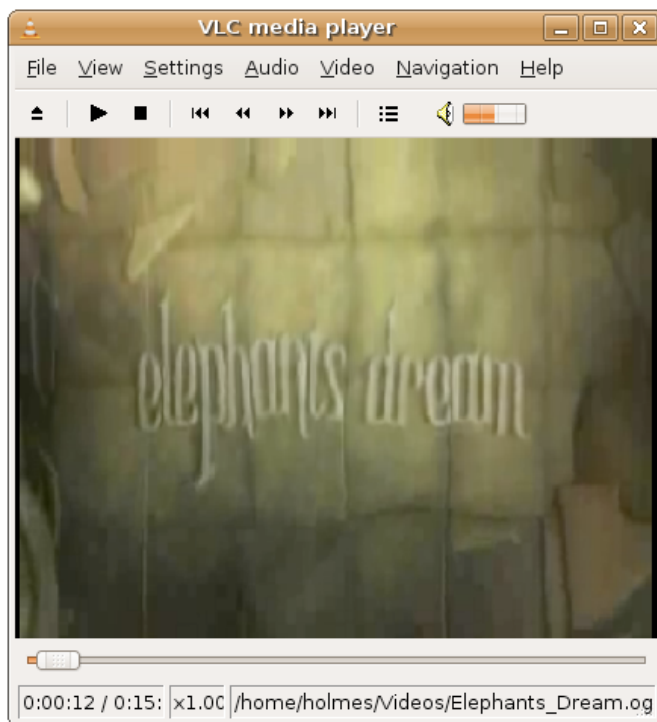
2. Go to File > Quick Open File in the VLC menu.



3. Find the Theora file you want to open. Select it, and click "Open".



4. A video will appear in your window like this, and the video should begin playing.



## Basic Playback Controls

- As in most media players, you can drag the playback position indicator to move forward or backward in the video.
- For full-screen playback, double click on the video. To exit fullscreen, double click again.

# Miro

Miro (previously called Democracy TV) is a tool for finding, downloading and watching video from a wide range of online sources. It is free software made by the nonprofit (NGO) Participatory Culture Foundation : <http://participatoryculture.org/>.

Miro plays a wide range of formats including Theora, and it runs on a variety of platforms (Operating Systems such as Windows, OSX, and GNU/Linux). In addition to video playback, Miro makes it easy to search for and download videos from specially formatted lists of videos known as podcasts or vodcasts.

You can use Miro to play video on your desktop, or to download and watch video from a URL pointing directly to the video file, or even a popular video website like YouTube. You can also use Miro to download and then watch Bittorrent files (files ending in .torrent).

## Installing

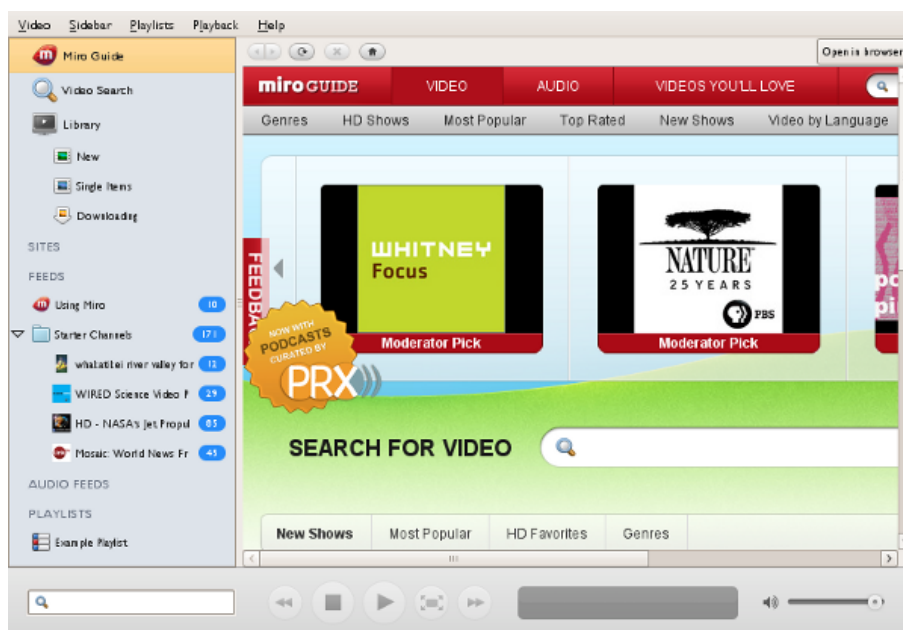
Miro is a desktop application that you need to download and install. Installation steps will vary depending on what platform you're using (GNU/Linux, Mac OS X, or Windows).

This page has links to download Miro and instructions for various platforms:

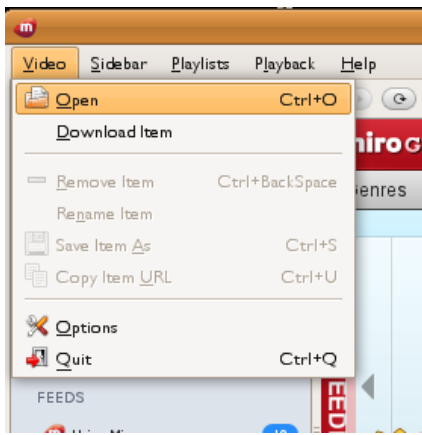
<http://getmiro.com>

## Playing A Video

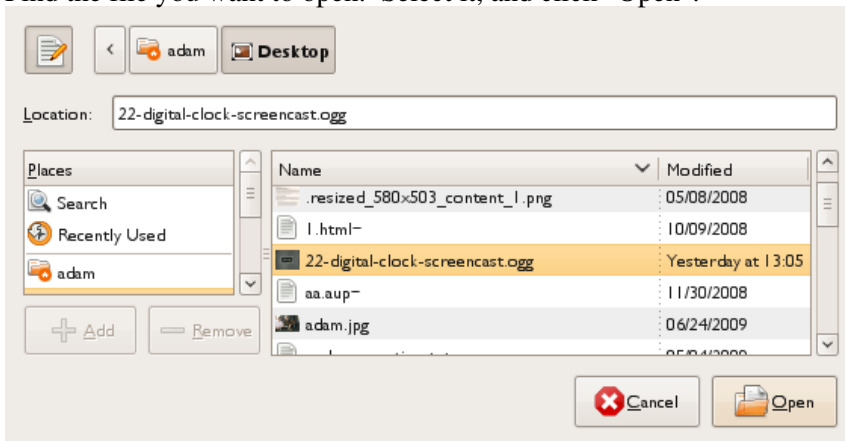
1. Run Miro. After clicking through a few messages that display the first time Miro runs, you will see a window that looks like this:



2. Go to Video > Open in the Miro menu.



3. Find the file you want to open. Select it, and click "Open".



4. The video should begin playing in the Miro Window.



# Introduction

There is a few things to consider before you put your video online.

Keep in mind that in order for viewers to watch your video on your website, they need to have a high-bandwidth connection. And the higher the video quality the higher the bandwidth needed to access the video. This means that sometimes your target audience may not even be able to access video online, and hence you may wish to consider another strategy such as distribution of the video on media like CD, DVD, or USB storage sticks.

If your audience has a fast internet connection you may wish to offer the video either for download or for playing from your website. Each strategy has its strengths depending on what it is you hope to achieve.

The advantages of offering your video for replaying on your website are mainly these :

1. you can define the context in which the video is being presented
2. it is convenient for your audience to just click and play the video
3. it is quite handy for your site visitors to just click and watch a small part of the video and not have to download a huge amount of data just to get an impression of your work
4. it is easy for your audience to show your work to others, for instance by sending around a link to the video they want to point to

The advantages of offering your video for download are these :

1. your audience can share it by file sharing networks (some see this as a weakness)
2. they can play it as many times as they wish without consuming extra bandwidth
3. they can edit it
4. they can show it offline (eg. movie screenings)
5. you can provide a higher quality video that is not possible to play in a website due to browser or bandwidth constraints

You could also opt to offer both strategies: an option to watch the video on your website and an option to download the video. You could even choose to upload two different versions of your video: one encoded in a quality more optimized for the web (something like a preview version of your film in a smaller size and poorer quality) and one video for people to download in a better quality and larger size and resolution.

If you want to share your Video on your website you have a few options. One is using the great HTML 5 *video* tag. If that is not an option for every viewer since some might use old browsers, you might want to also offer a fallback solution: the Cortado player (software you can embed in your webpage to play Theora).

If you don't have access to a server to store and deliver video, you can also use one of the many available hosting sites, that offer free hosting of Theora video (you don't want to upload your video to hosting sites such as Youtube since you would lose quality, and you would also wave some of your rights on the material).

Another strategy is to offer your video for download with bittorrent. Bittorrent is a peer-to-peer filesharing protocol that shares the internet connection of a respective number of computers that download the same file. Hence, the more people that download your video with bittorrent the faster the transfer will become for other users. Bittorrent is probably the most economic way to transfer large popular files on the internet.

# HTML5 Video

If you create video, you may wish to display the content in a webpage. The code you use to create webpages is governed by a set of rules known as HTML, and there is recently a new version of these rules called HTML 5.

HTML 5 introduces a *video* tag. A 'tag' is a few lines of HTML code that instructs the browser to display something or do something. The HTML5 video tag allows simple integration of videos in a manner very similar to placing images in a webpage.

The video can also be displayed with very nice controls for play, pause, altering the audio volume, and scrolling through the timeline of the video.



## Basic Syntax

Here is a basic example of a *video* embed tag using HTML 5 :

```
<video src="video.ogv"></video>
```

The above example embeds the video file, 'video.ogv', into a webpage. The file in this example should be located in the same directory as the HTML file because the 'src' parameter refers to a local file. To reference a video file in another directory on the same webserver you need to provide the path information just as you would for an image file.

```
<video src="/bin/edit/myvideofiles/video.ogv"></video>
```

You can also specify a file on another server:

```
<video src="http://mysite.com/video.ogv"></video>
```

## Parameters

Adding additional parameters provides more control over the video.

```
<video
  src="video.ogv"
  width="480"
  height="320"
  autoplay
  controls>
  Your Browser does not support the video tag, upgrade to Firefox 3.5+
</video>
```

In this example the *width* and *height* of the video are provided. If you don't want the image to appear distorted it is important that you set the height and width dimensions correctly. *autoplay* means that the video will be played as soon as the page loads. *controls* specifies that the controls to pause or play the video (etc) are displayed.

It is possible to include text or other HTML content inside the video tag as fallback content for browsers that do not support the video tag.

## Using your own controls / player skin

If you know a little Javascript you can control the playback quite easily. Instead of using the *controls* provided by the browser, it is possible to create your own interface and control the video element via JavaScript. There are two things you need to remember with this method :

1. Do not forget to drop the *controls* attribute
2. The video tag needs an id parameter like this :

```
<video src="video.ogv" id="myvideo"></video>
```

Some JavaScript functions:

```
video = document.getElementById("myvideo");  
//play video  
video.play();  
//pause video  
video.pause();  
//seek to second 10 in the video  
video.currentTime = 10;
```

If you have multiple video tags in a single webpage you will need to give each a unique id so that the javascript knows which video the controls refer to.

A full list of functions and events provided by the video tag can be found in the HTML5 spec at <http://www.whatwg.org/specs/web-apps/current-work/#video>

## Manual Fallback options

In the above, simple example, if the video element is not supported by the browser, it will simply fall back to displaying the text inside the video element.

Instead of falling back to the text, if the browser supports Java, it is possible to use Cortado as a fallback. Cortado is an open-source cross-browser and cross-platform Theora video player written in Java. The great thing is that the user doesn't need to download any extra Java packages as the applet uses the standard native Java in the browser. Cortado's home page can be found here :

<http://www.theora.org/cortado/>

A pre-compiled version of the applet is also available at this URL :

<http://www.theora.org/cortado.jar>

You can download the jar file, or you can refer to it directly. The following is an example to embed cortado (not all paramters are required, but listed here to provide you an idea of possible options) :

```

<applet code="com.fluendo.player.Cortado.class"
        archive="http://www.theora.org/cortado.jar"
        width="352" height="288">
  <param name="url" value="http://myserver.com/theora.ogv"/>
  <param name="framerate" value="29"/>
  <param name="keepAspect" value="true"/>
  <param name="video" value="true"/>
  <param name="audio" value="true"/>
  <param name="bufferSize" value="100"/>
  <param name="userId" value="user"/>
  <param name="password" value="test"/>
</applet>

```

If you select to download the jar file as a fallback, make sure you put it (**cortado.jar**) and the above html page in the same directory. Then change the following line to include a reference (link) to your own ogg stream (live or pre-recorded) :

```
<param name="url" value="http://myserver.com/theora.ogv"/>
```

Now if you open the webpage in a browser it should play the video.

To use Cortado as a fallback, place the Cortado tag within the HTML5 video tag -- as in the following example:

```

<video src="video.ogv" width="352" height="288">
  <applet code="com.fluendo.player.Cortado.class"
        archive="http://theora.org/cortado.jar" width="352" height="288">
    <param name="url" value="video.ogv"/>
  </applet>
</video>

```

## Javascript Based Players

Some javascript libraries exist to handle fallback selection. These libraries enable simple embedding while retaining fallback to many players and playback methods across many browsers

### Mv\_Embed

The mv\_embed library is very simple to use. A single JavaScript file include enables you to use the html5 video tag and have the attributes be rewritten to player that works across a wide range of browsers and plugins. More info on mv\_embed

```

<script type="text/javascript" src="http://metavid.org/w/mwScriptLoader.php?class=mv_embed"></script>
...
<video src="mymovie.ogv" poster="mymovie.jpeg">

```

### Itheora

ITheora is a PHP script allowing you to broadcast ogg/theora/vorbis videos (and audios) files. It's simple to install and use. Itheora includes documentation on their site on how to use their player and skins.

## Support in Browsers

Right now, latest versions of Mozilla Firefox, GNU IceCat and Epiphany browsers support Theora natively. Opera and Google Chrome have beta versions available with Theora support. Safari supports the video tag, but only supports codecs through QuickTime - that means by default it does not support Theora. With XiphQT (<http://xiph.org/quicktime>) it is possible to add Theora support to QuickTime and thus Safari.



# Hosting on your own site

Like images or HTML pages you can put your Theora videos on your own webserver.

## Mime Types

For videos to work they have to be served with the right **mime type**.

A mime type is the name given to a way of identifying different file types delivered over the internet. This information is usually delivered with the data. The extra information identifying what kind of information is being delivered is usually not readable by humans, but is interpreted by software so that the right kind of data is delivered and processed by the right kind of software. The information is sent in the **header** of the transported data.

A header is a small amount of meta data sent by one software to another which describes the kind of information being transported. Typical header information includes the length, destination, mime type etc.

Mime types have two parts - the *type* and *subtype* (although the two together is just referred to simply as a 'type'). The type is written in the form *type/subtype*. There are only four categories of type - audio, video, text, and application. There are innumerable subtypes.

The right mime type for Theora video is 'video/ogg'.

A current server should send the right information. If your server does not send the right headers, you have to change the configuration of your webserver. However, you should first consider if you know enough about configuring web services. If you are not feeling confident about this then perhaps enlist the support of a friendly techie. Assuming you are using the Apache webserver (the most popular webserver on the web), and you feel confident to change the server configuration, there are two ways you can do this :

1. you can add two lines to your apache webserver configuration
2. you might be able to provide the extra settings by placing a .htaccess file in your video folder

For the first strategy you need access to your web server configuration file (httpd.conf). In this case all files being delivered by your webserver will send the correct information. This is the best solution. However if you do not have access to your webserver configuration files (for example, if you are using a shared hosting service) then you may wish to try the second strategy. The second strategy will only effect the video being served from the same folder that you put the .htaccess file.

For both strategies you need to enter this information in the appropriate file:

```
AddType video/ogg          .ogv
AddType application/ogg     .ogg
```

For httpd.conf and .htaccess files you can place this information at the end of the file. If you do not have an .htaccess file you can just create a blank file and add this information (no other information is required).

## oggz-chop

Using Apache you can also get a lot more sophisticated. You can, for example, enable the use of URLs that reference and playback only part of any given Theora video file. If you want to include only parts of your video in your webpage or allow linking to a specific time in the video, you can use **oggz-chop** on the server.

With oggz-chop installed you can address segments of the video by providing an offset parameter in the url. To include second 23 to second 42, of your video you would use

```
<video src="http://example.com/video.ogv?t=23.0/42.0"></video>
```

Installing oggz-chop requires the action module to be enabled in apache and oggz-chop installed on the server. Describing how to enable the action module is beyond the scope of this document, so you had better consult with an Apache guru, or read some documentation on this subject before attempting it.

However...if you know you have Apache2 installed and you have administrator access to your server you could try this command to install the actions module :

```
sudo a2enmod actions
```

Try it at your own risk...

oggz-chop is part of oggz-tools, you can (probably) install that with :

```
sudo apt-get install oggz-tools
```

With those installed you have to enable oggz-chop with those two lines in your apache configuration or .htaccess file:

```
ScriptAlias /oggz-chop /usr/bin/oggz-chop
Action      video/ogg      /oggz-chop
```

## Allowing Remote Access

Videos, unlike images, can not be embedded from remote sites, if those sites do not specifically allow this. To allow other sites to include your videos on their domain add this line to your apache configuration or .htaccess file:

```
Header Set Access-Control-Allow-Origin "*"
```

With this setting, the server responds with an additional header 'Access-Control-Allow-Origin: \*' which means that the videos can be embedded in any webpage. If you want to restrict access to the videos (for example, to be only accessible from http://example.org) you have to change it to:

```
Header Set Access-Control-Allow-Origin "http://example.org"
```

Note that now, your videos can not be embedded on domains other than example.org. The Access-Control-Allow-Origin header can also contain a comma separated list of acceptable domains. [https://developer.mozilla.org/En/HTTP\\_access\\_control](https://developer.mozilla.org/En/HTTP_access_control) has a more detailed description of http access control.

## Serving Videos via a Script

As a final strategy. If you do not have control over your hosting setup but want to use videos anyway, it is possible to use a small php or cgi script to set the right headers and serve the video. Such a script could look like this:

```
<?php
$video = basename($_GET['name']);
if (file_exists($video)) {
    $fp = fopen($video, 'rb');
    header('Access-Control-Allow-Origin: *');
    header('Content-Type: video/ogg');
}
```

```
header('Content-Length: ' . filesize($video));
fpassthru($fp);
} else {
    echo "404 - video not found";
}
?>
```

If this script is placed as `index.php` in your video folder, you would use `http://example.com/videos/?name=test.ogv` instead of directly linking to your video (`http://example.com/videos/test.ogv`).

```
<video src="http://example.com/videos/?name=test.ogv"></video>
```

# Hosting Sites

There are many existing websites that will host your Theora content for free. You can then either link directly to the content from your own webpages, or refer people to the content hosted on the external site.

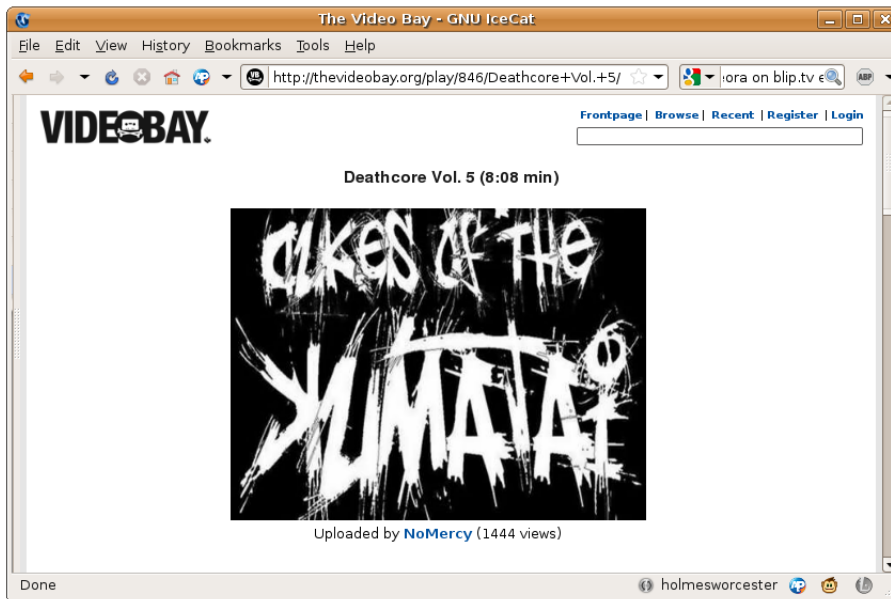
## TinyVid.tv

TinyVid (<http://tinyvid.tv/>) is a simple video sharing site set up by a member of the Theora community. You can upload a video in any format to be encoded in Theora. It also supports the Firefogg extension for Firefox (<http://firefogg.org/>) which lets you encode and upload at the same time!



## The Video Bay

The Video Bay (<http://thevideobay.org>) is a new project by the people who run The Pirate Bay (<http://thepiratebay.org/>). This site will transcode video from any format to Theora. The site's administrators have a reputation for vigorously defending peoples' rights against the claims of copyright holders.



## Archive.org

Archive.org (<http://www.archive.org>) is the website for the not-for-profit organization known as the **Internet Archive**, that focuses on the preservation of digital media. To post to the archive, you must also license the work under a **Creative Commons** (or similar) license. This is usually not a problem if you made the work yourself, but maybe an issue if you used copyrighted material (eg, music within a video) within your work, or if you are uploading something someone else made.

You can host Ogg Theora content at the Internet Archive for free, and then link to it from your own website. Archive.org will convert your video to Ogg Theora and many other formats when you upload it.

**Note:** when you visit a page on archive.org, it will notify you if your browser supports the video tag. If you see this message...



Simply reload the page and look for the message under the video icon:



[embed this](#)

Your browser supports the new <video> tag!  
Would you like to [try the new <video> tag?](#)

At this point, click the link that says "try the new <video> tag" and click the following link to always use the video tag. Then your videos will display in Theora. If you'd like a direct link to the Theora version of the video, copy the "Ogg Video" link in the left sidebar. When you have that link, you can link directly to the video and anyone with a compatible browser will see the video play:



## Dailymotion

Dailymotion (<http://dailymotion.com/>) is one of the larger video sharing sites, and recently they have been dabbling with Theora support. They have converted over 300,000 videos to Theora, and by applying for a "motion maker" account (and getting approved) you can publish videos there using Theora. Their experimental Theora portal here: <http://openvideo.dailymotion.org/>. Note: not all videos in the portal currently display in Theora, so be sure to verify that your videos are working before relying on this service.



## Wikimedia Commons

*Wikimedia Commons* (<http://commons.wikimedia.org/>) is a website managed by the *Wikimedia Foundation* (<http://wikimediafoundation.org/>), a non-for-profit organization that also manages Wikipedia. It is a database of media files available for anyone to use for any purpose. It's an open website that any can contribute to, which uses wiki software that allows for easy collaboration.

The site is managed entirely by volunteer editors, who also create the majority of its content by contributing their own work. The community is multilingual, with translators available for dozens of languages. It only collects material that is available under free content licenses or in the public domain. You can upload Ogg Theora files to Wikimedia Commons.

File:Forage harvester - forage wagon 640x480.ogv - Wikimedia Commons - GNU IceCat


File Edit View History Bookmarks Tools Help

http://commons.wikimedia.org/wiki/File:Forage\_harvest... Google

# File:Forage harvester - forage wagon 640x480.ogv

You have new messages (last change).

[File](#) [File history](#) [File links](#)



More...

Done holmesworchester

The image shows a green tractor with yellow wheels pulling a red forage harvester in a grassy field. The harvester is cutting and chopping the grass into small pieces. The background consists of a dense line of green trees. The entire scene is captured in a video format (ogv).



# Introduction

*Encoding* is the process of creating a Theora file from raw, uncompressed source video material. In case the source video material exists in some non-raw, compressed form, an intermediate decoding step is needed before creating the Theora video. This Decoding-Encoding is often referred to as Transcoding, though often Encoding is used as a synonym.

Software programs performing the encoding (resp. transcoding) are called encoders. Various Theora encoders exist, for example `ffmpeg2theora` and `vlc`, to name just a few.

Before encoding the user has to decide on at least two parameters:

- the image quality of the created Theora file
- the audio quality

Depending on the encoder used, more options might be available to control the encoding process:

- clipping a configured amount of the frames' borders during encoding
- rescaling the video resolution
- changing the frame-rate of the video
- handling the video-audio synchronisation
- setting the keyframe-interval

## Video Quality, Bit-Rate and File Size

Most Theora encoders allow the user to directly specify the subjective quality of the encoded video, usually on a scale from 0 to 10. The higher the quality, the bigger the resulting Theora files. Most encoders can alternatively be configured to encode for a given average target bitrate. While this option is useful for generating Theora video files for streaming, it sometimes yields sub-optimal quality.

Recent versions of some Theora encoders feature a two-pass encoding mode. Two-pass encoding allows the encoder to hit a configured target bit-rate with optimum video quality, and should thus be comparable to quality controlled encoding, though it comes at the cost of taking twice the encoding time. By nature live videos can not be generated with two-pass encoding.

## Video Resolution and Frame-Rate

There can be good reasons to further reduce the height and width (video resolution) of your video when encoding to Theora.

- If the Encoder produces too large files, even at low quality settings around 0, then reducing the video resolution will help reduce the file size further
- If your required maximum file size requires a very low quality setting of 0..3, leading to an unacceptable perceived quality then reducing the video resolution will mean more data can be dedicated to improving the quality
- If the playback of the encoded video should work even on low-performance computers then a lower video resolution will assist this
- If your source video material has resolution higher than standard-definition video. The Theora video codec is not designed for high-definition video and might not perform very well at it so it would be better to reduce the video resolution

If your source video material has an unusually low resolution, and you can spare the bits, increasing the video resolution during encoding might have a positive effect on overall video quality.

Adjusting the frame-rate during encode is generally a bad idea, as it often leads to jerking, reducing perceived quality by a large amount. However, if you require a very low target file size, try reducing the frame-rate to exactly half the source frame rate. This might do the job of sufficiently reducing file size without degrading quality to an unacceptable level.

## Clipping the Frames in the Video

Sometimes video source material does not make use of the full video frames, leaving black borders around the video. It is a good idea to remove black or otherwise unused parts from the video as this usually improves the quality and file size of the encoded Theora file. If possible, try to keep video width and height multiples of 16. The Theora format is capable of, but not very efficient at, storing video using other arbitrary frame sizes.

## Video-Audio Synchronization

In an ideal world, the encoder would just copy the video-audio synchronization of the source video material to the created Theora file. In practice however, this is sometimes just not possible. Theora video files must adhere to a constant frame rate throughout the full file. Also the playback speed of the audio tracks is constant in Theora. Some source video material, however might not have a 100% constant frame rate. Sometimes frames are just missing from the source video due to recording errors or as a result of using video cutting software.

In these cases, the encoding process must actively adjust audio-video synchronization. This is done either by duplicating and/or dropping frames in the video, or by changing the speed of the audio tracks.

## Keyframe-Interval

Many Theora encoders allow changing a parameter named *keyframe interval*. A larger keyframe interval reduces the target file size without sacrificing quality. Keyframes are those frames in the video, which a player can directly seek to during playback. To seek to other points in the video, all frames from the last keyframe on have to be decoded first. In a video with 24 frames/second, a keyframe interval of 240 implies that direct seeking is only possible with a granularity of 10 seconds. Also cutting and concatenation of the encoded video will be limited to the keyframe granularity.

As a rule of thumb, never set the keyframe interval to more than 10 times the target video frame rate.

# Firefogg

Firefogg is the name of an extension to the Firefox web browser that adds support for encoding your video files to Theora using a nice web interface. It also enables web-sites to provide a video-upload service that takes videos from your computer, converts them to Theora on-the-fly uploading the generated Theora file to a website.

## Installation

Firefogg requires the Firefox web browser, at least version 3.5. If your version of Firefox is older, or in case you do not have Firefox installed at all, visit [www.mozilla.com](http://www.mozilla.com) to download an up-to-date version.

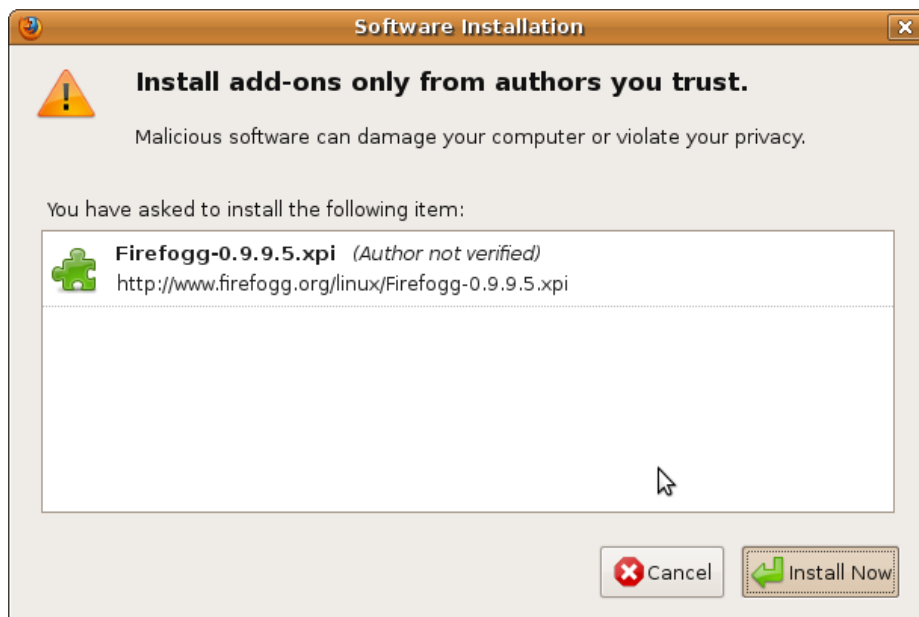
Once you have a recent Firefox version, use it to visit the Firefogg homepage at [www.firefogg.org](http://www.firefogg.org):



Now click on *Install Firefogg*. At the top of the page, Firefox now asks you to allow installation of the new software:



Click on *Allow*, to pop up the following dialog:

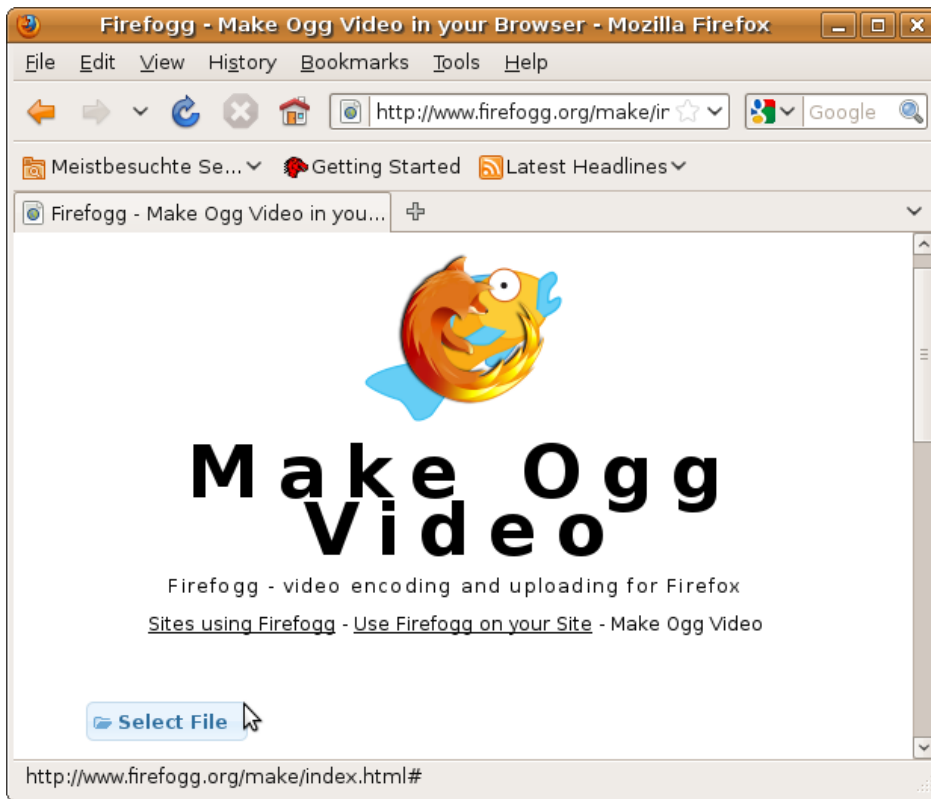


Click on *Install Now*. After installation, you are asked to restart Firefox. Click on *Restart Firefox* to proceed.

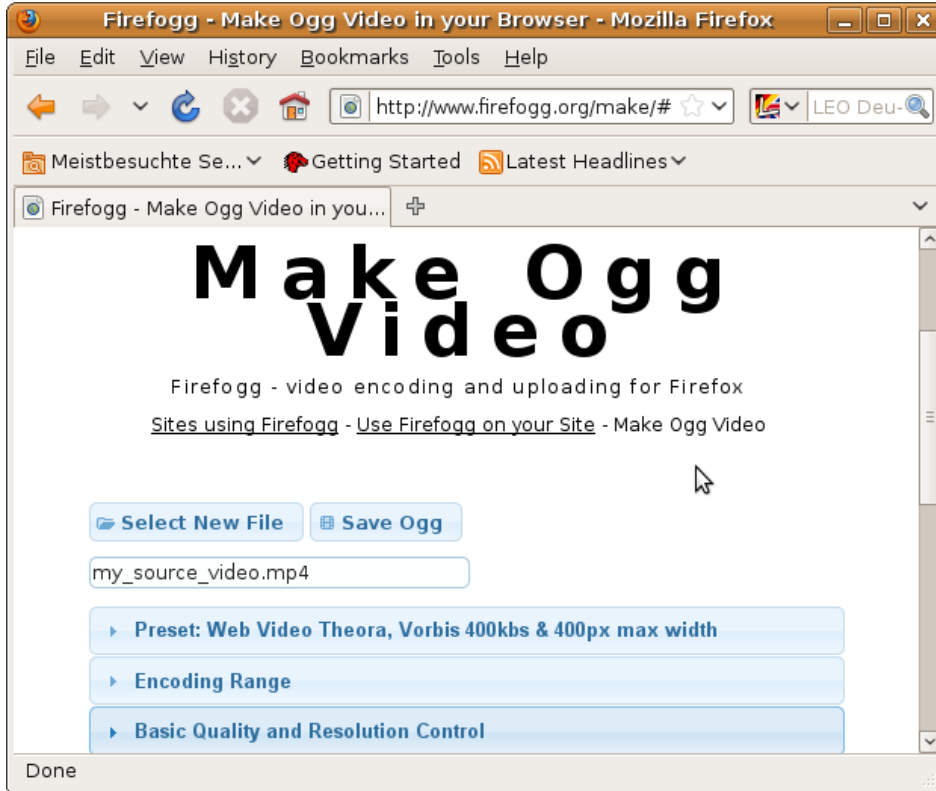
## Encoding your first video

To encode videos with Firefogg, you need an internet connection. Parts of the encoding software reside on the internet and will not be installed to your computer.

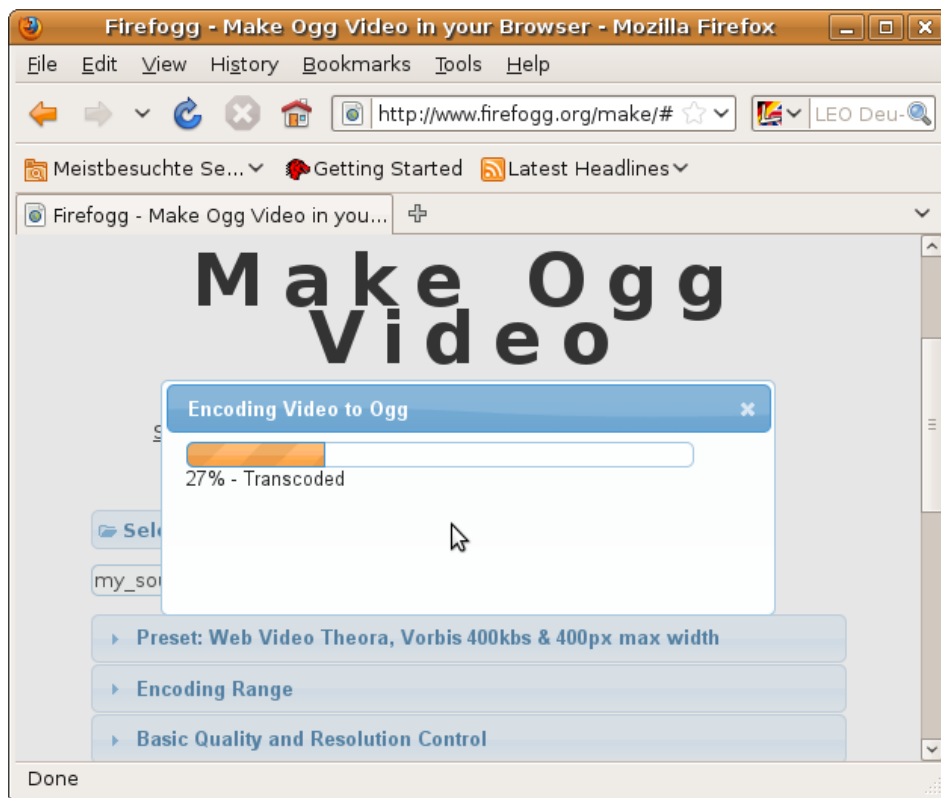
The encoding software is started by visiting <http://firefogg.org/make>. The following site shows up:



Click on *Select File*, which pops up a file dialog allowing you to select the source video file to encode. In this example we are using the video *my\_source\_video.mp4*. After selecting the file, you are brought to a dialog that asks you for tuning encoding parameters:



For now, just select *Save Ogg*. You are asked to select the name of the Theora file to create. Remember. We select the name *my\_theora\_video.ogv* (remember, the correct file name extension for Theora videos is *.ogv*). Now encoding starts, using the default set of parameters:

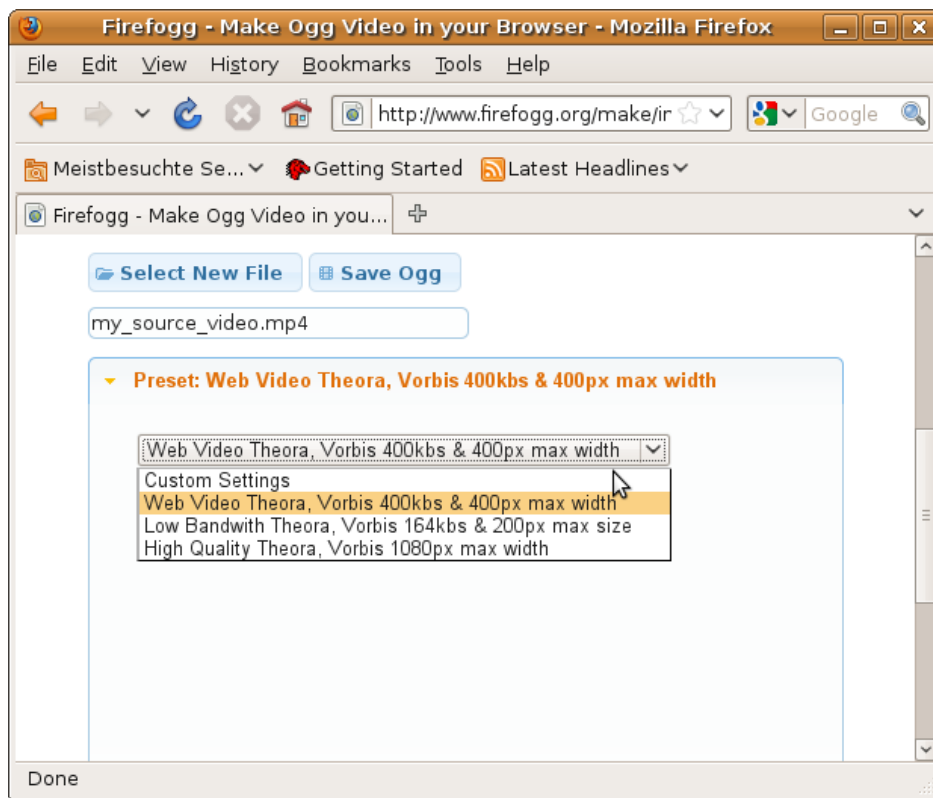


Now wait for the encoding to progress to 100% and you're done.

## Advanced Encoding Options

Using the default set of parameters for encoding video yields very small Theora files optimized for web streaming. Perceived video quality will actually be quite low for most tastes. But wait, Firefogg is as advanced as most other Theora encoders. After a little tuning, very high quality videos can be easily created. Tuning options are available on the web page below the *Save Ogg* button.

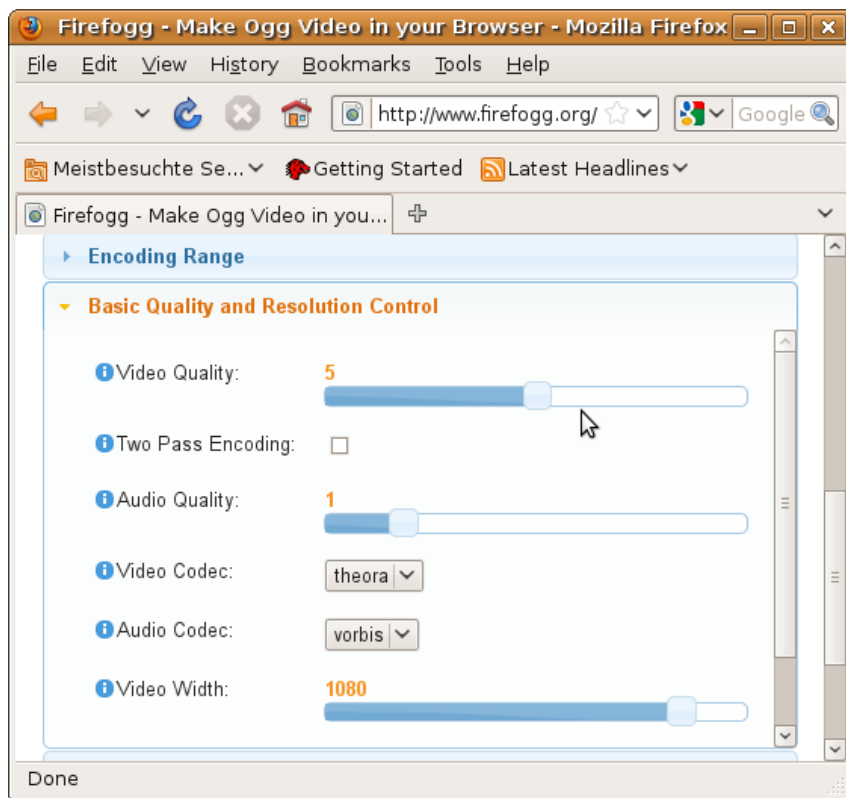
The easiest way to configure encoding is by selecting one of three presets available from the *Preset* menu:



The first two presets called *Web Video Theora* and *Low Bandwidth Theora* are both optimized for streaming video over the internet. If you intend to play back the created Theora file solely from CDs, USB sticks or your computer's hard disk, you should try to *not* use them. Go for the *High Quality Theora* preset instead.

## Custom Settings

More details of the encoding process can be configured by selecting *Custom Settings* and manually adjusting lower-level options in the other available menus. Lets have a look at the most important menu, *Basic Quality and Resolution Control*:



Here you can control the quality of the created Theora file, and also choose to change the frame size of the encoded video. Encoding a video for a target quality (instead of a target bit rate) is the preferred encoding mode for Theora, so in most cases you won't want to try the encoding options in the other menus.

## Other Uses of Firefogg

Once installed, Firefogg can be used on websites that support it. There is a list of sites that support Firefogg at <http://firefogg.org/sites>.

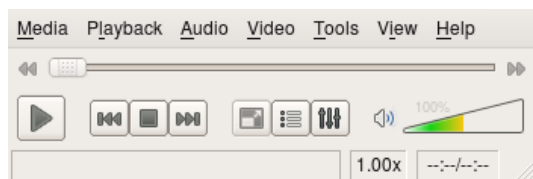


# Encoding with VLC

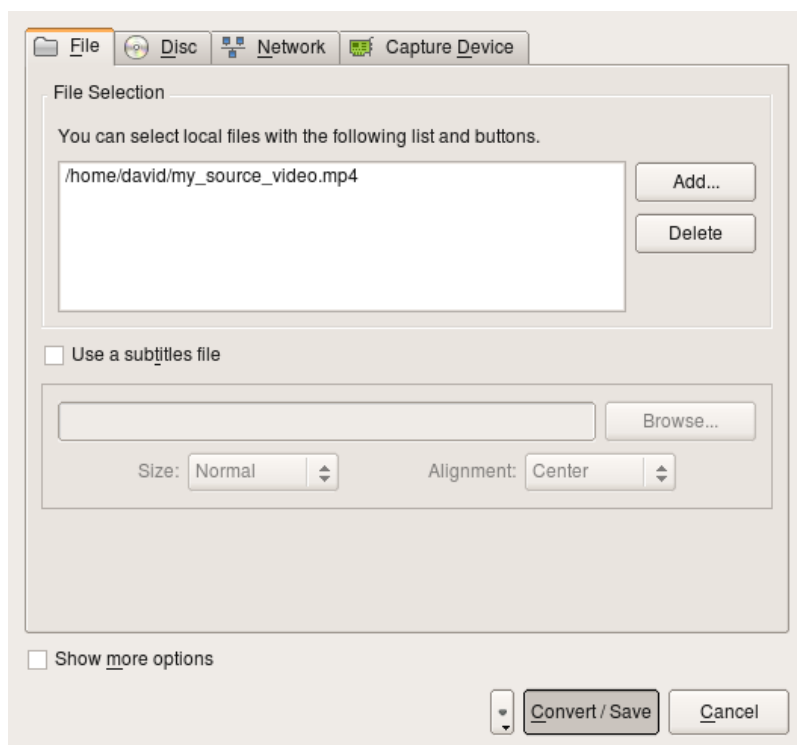
The VLC media player ([www.videolan.org](http://www.videolan.org)) allows easy encoding of video files to Theora. Encoding can either be performed via the graphical user interface (GUI), or from the command line. The following instructions have been written for use with VLC version 1.0.1 running on Ubuntu. Other versions may differ in details, though the overall process will be the same.

## Using the GUI

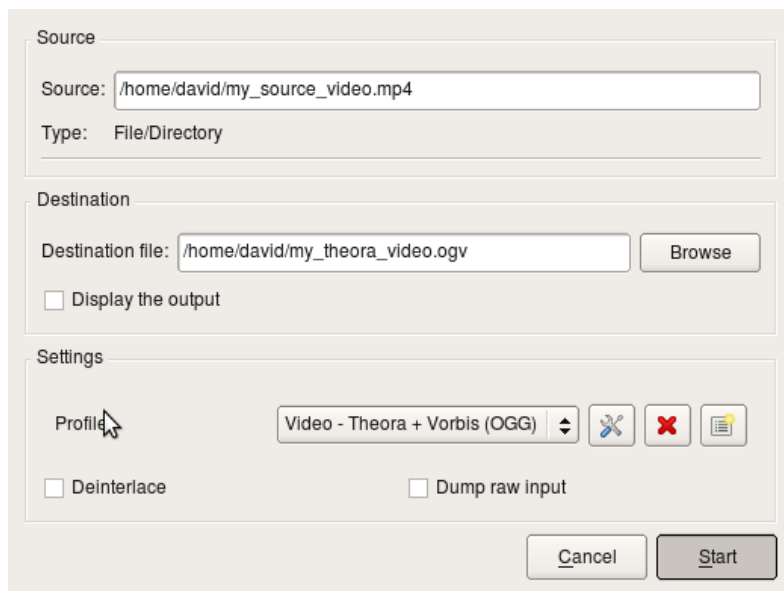
When you start VLC, you are immediately greeted by its main window:



In the *Media Menu* select *Convert/Save*. This brings you to the following dialog:

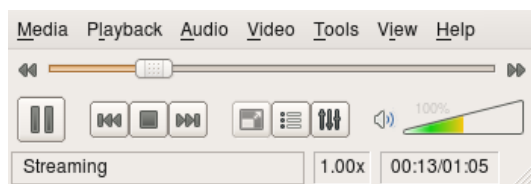


Under *File Selection* click *Add* and select the source video file for encoding. Then click on *Convert/Save* at the bottom. This leads you to the encoding dialog:



Under *Destination* click *Browse* and select the location and name of the Theora file that you want to create. Remember that the correct file name extension for a Theora file is **.ogv**. Under *Settings* set the Profile to "Video - Theora + Vorbis (OGG)".

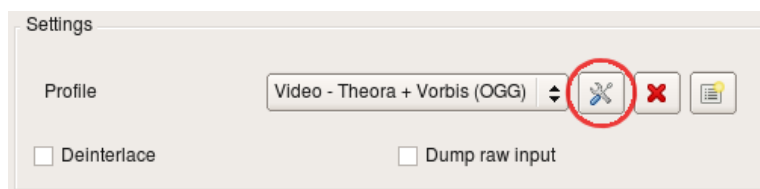
Then press the *Start* button. This starts the encoding process and brings you back to VLC's main dialog:



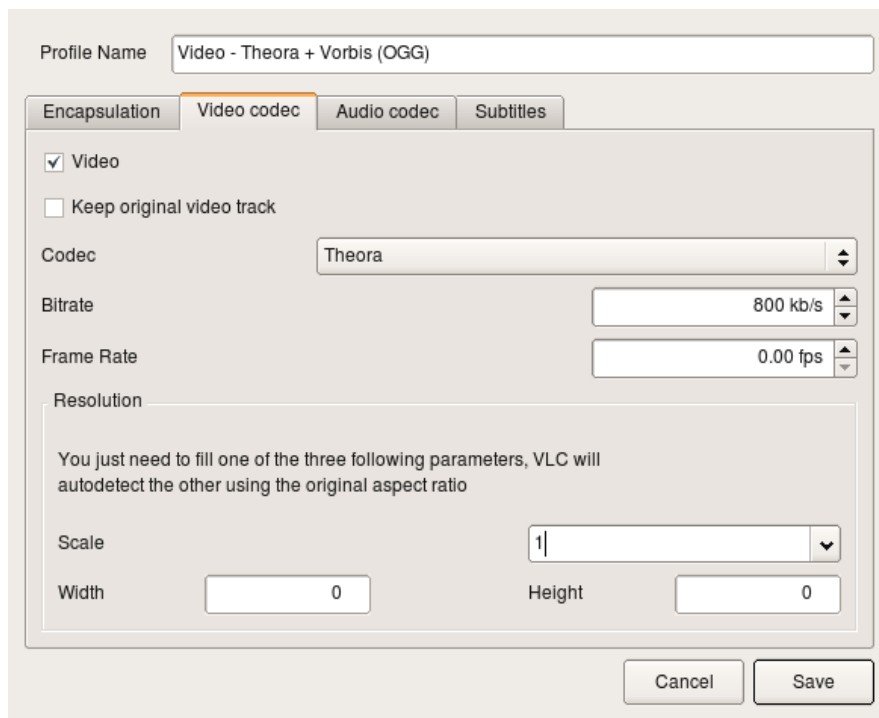
At the window's bottom, the text "Streaming" is now displayed. This indicates that it is busy encoding your file. The slider will slowly move to the right as encoding progresses. Once encoding is done, the slider jumps back to the left and the "Streaming" display disappears.

## Advanced Options

If you are not satisfied with the encoding result, try adjusting some of the more advance encoding parameters. In the previous encoding dialog, before hitting *Start*, press the button with the "tool" icon, just to the left of the profile selection:



This opens up a new dialog, with the 4 tabs, labeled; "Encapsulation", "Video codec", "Audio codec" and "Subtitles". Make sure you do not change any parameters under "Encapsulation". The video encoding options you need are under the "Video codec" tab, shown below:



If your encoding result's video quality was too low, try increasing *Bitrate*. If your video source material has a very high resolution, try setting *Scale* to 0.5 to encode at half the original resolution. As of this writing, changing video resolution fails to work properly for some source material, when using VLC 1.0.1.

## Encoding from the Command Line

If you are encoding a number of files to Theora, clicking through the VLC dialog windows can become tedious and error-prone. Here VLC's command line interface comes to the rescue. While not as intuitive as the GUI, it allows you to exactly repeat an encoding process with constant parameters.

Use the following command to encode your source video, (in the example below the files are called "my\_source\_video.mp4" to "my\_theora\_video.ogv" you can of course, use whatever name you want) with the same parameters as used in the GUI example above:

```
vlc my_source_video.mp4 \
  --sout="#transcode{vcodec=theo,vb=800,scale=1,
  deinterlace=0,acodec=vorb,ab=128,channels=2,\
  samplerate=44100}:standard{access=file,mux=ogg,\
  dst='my_theora_video.ogv'}"
```

All the parameters that had previously been specified in the advanced encoding options dialog window, are now given in text form. The only options whose meaning might not be immediately obvious are *vb* which means video bitrate and *ab*, which refers to the bitrate of the encoded audio.

## Recommended Encoding Options

The command line shown above merely utilizes the parameter set provided by the GUI interface, which is in no way optimized for Theora encoding. We can do much better by using options that are exclusively available on the command line only. A better option might be to use the following command line as a basis for encoding. Tweak it to suit your needs:

```
vlc my_source_video.mp4 \
  --sout-theora-quality=5 \
  --sout-vorbis-quality=1 \
```

```
--sout="#transcode{venc=theora,vcodec=theo,\
scale=0.1,deinterlace=0,croptop=0,\
cropbottom=0,cropleft=0,cropright=0,\
acodec=vorb,channels=2,samplerate=44100}\
:standard{access=file,mux=ogg,\
dst='my_theora_video.ogv'}"
```

In this example video and audio quality are specified as numbers in the range 0 (low quality) up to 10 (highest quality). In case you want to remove black or noisy borders around the video, adjust the options *croptop* through *cropright*.

These examples require that your VLC installation comes with the VLC Theora plugin. Verify the plugin's presence by typing:

```
vlc -p theora
```

Even if this prints out *"No matching module found"* it may still be possible to encode to Theora, by using the ffmpeg plugin that supports Theora as well. However, the advanced *"--sout-theora-quality"* option is not available with ffmpeg.

## Why not to use the GUI

The Theora codec works best when encoding for a specified video quality. Encoding for a given target bitrate will always give inferior results for a file of the same size. Unfortunately there is no way to specify target video quality via the GUI, which is why you just shouldn't use it for any kind of professional encoding work. Also, note that you cannot crop borders of the video when encoding from the VLC GUI.

Keep in mind that this chapter was written using the 1.0.1 version of VLC, which was current at the time of this writing. Newer versions may outgrow these limitations.

â

# ffmpeg2theora

ffmpeg2theora is a very advanced Theora encoding application. The advanced functionality comes at the price of having to use a command line, as no graphical user interface is provided. For GNU/Linux, Mac OS X and Windows download ffmpeg2theora from <http://v2v.cc/~j/ffmpeg2theora/download.html>. If you are running a recent version of GNU/Linux, chances are good that your distribution already comes with software packages for ffmpeg2theora, that can be installed with the distribution's software packet manager.

## Basic Usage

Open a command prompt and enter:

```
ffmpeg2theora my_source_video.mp4 -o my_theora_video.ogv
```

This encodes the source video file "my\_source\_video.mp4", creating a new Theora video file named "my\_theora\_video.ogv".

## Adding Parameters

When you are unhappy with the result of encoding, it's time to start tuning encoding parameters. We'll start with setting the video quality of the encoded video. Quality is given as a number in range 0 (lowest quality, smallest file) up to 10 (highest quality, largest file). Try this to encode at a high video quality of 9, and a very high audio quality of 6:

```
ffmpeg2theora my_source_video.mp4 -o my_theora_video.ogv \  
  --videoquality 9 --audioquality 6
```

The following example exposes the basic encoding parameters. Just copy-paste and adjust it to your needs:

```
ffmpeg2theora my_source_video.mp4 -o my_theora_video.ogv \  
  --videoquality 9 --audioquality 6 \  
  --croptop 0 --cropbottom 0 --cropleft 0 --cropright 0 \  
  --width 720 --height 576 \  
  --title "Video Title" --artist "Artist Name" --date "1997-12-31"
```

If you do not wish to scale the video's frame size, drop the `--width` and `--height` options. There is no way to specify a scale factor, so check the input video's size and computing the target frame size as required. In most cases it is better to only specify one of the `--width` or `--height` options, the missing option is then automatically adjusted to a correct value.

## Advanced Options

Ffmpeg2theora supports a multitude of other parameters for advanced use, which cannot all be described in detail here. To get an overview of all available options and short descriptions, type:

```
ffmpeg2theora --help
```

Depending on the operating system you are using, you *might* be able to open up the ffmpeg2theora manual by typing

```
man ffmpeg2theora
```

The following options often prove useful:

### **--sync**

Copy any audio-video synchronization of the source video file to the destination Theora video. Depending on the source video used, this may fix problems of audio-video delay drift introduced by the encoding process.

### **--keyint <N>**

Set the keyframe interval, i.e. the number of frames between keyframes, of the generated file. Large values of <N> lead to a reduced file size, however seeking and cutting does not work well with Theora files that have a large keyframe interval.

### **--framerate <N>**

Set the frame rate of the generated video file. In case you are attempting to create Theora videos with extremely small file size, try specifying half the input video's framerate.

### **--starttime <N> --endtime <M>**

These two options allow you to copy only a part of the source video when encoding. Specify <N> and <M> as the number of seconds from the start of the video.

## **Two-Pass Encoding**

The upcoming version 0.25 of ffmpeg2theora is going to support a two-pass encoding mode, which is described in this section. Once version 0.25 is released, just download it from <http://v2v.cc/~j/ffmpeg2theora/download.html>. The examples given below are not going to work with older versions.

## **Why Two-Pass Encoding**

A lot of hype is surrounding two-pass encoding. Many people assume that you need to encode in two passes to achieve a constant subjective quality throughout a video. This is how it used to be for many non-free video codecs such as DivX. However, as we have seen, ffmpeg2theora is well capable of encoding for a constant target quality in a single pass using option *--videoquality*.

The only real advantage of using two-pass mode over using *--videoquality*, is the ability to create a Theora video of a given file size. Imagine you want to encode a video, which must fit onto a single CD with 700 MB of available storage. You want a constant video quality, but in advance you can't possibly guess which *--videoquality* will exactly hit 700 MB. Using two-pass mode exactly achieves that.

### **Using Two-Pass Mode**

So you want to encode "my\_source\_video.mp4" into a Theora video, with a file size of exactly 700 MB. ffmpeg2theora does not allow you to directly specify the size of the encoded file. Instead you specify the average video bitrate for the video. Note that the audio is also going to require some data, which has to be taken into account.

To decide on an average video bitrate for the file we first need to find out the duration of the source video "my\_source\_video.mp4". Ffmpeg2theora can help us with that. Type:

```
ffmpeg2theora --info "my_source_video.mp4"
```

Which prints, among other information:

```
{
```

```
--sync
```

```
"duration": 2365.165100,  
"bitrate": 6437.331055,  
[...]  
}
```

The *duration* shown is in seconds. If we divide the available 700 MB of space by the 2365 seconds to encode, we come to an average *byte* rate of 296 kByte/s. Multiplied by 8 we get the average *bit* rate of 2368 kBit/s.

We cannot use the full 2368 kBit/s for our video only. We also have an audio track, that is going to take 128 kBit/s. The average video bit rate we can use is thus 2240 kBit/s. Of this we subtract another 1% to account for any overhead in the encapsulation that is used to contain the video and audio tracks. This leaves us with 2218 kBit/s for the video and 128 kBit/s for the audio.

The following command is going to perform the two-pass encoding, creating the Theora video file "my\_theora\_video.ogv":

```
ffmpeg2theora my_source_video.mp4 -o my_theora_video.ogv \  
  --two-pass --videobitrate 2218 --audiobitrate 128
```

Note that unlike other two-pass encoders, only *one* invocation of the `ffmpeg2theora` command is required. If you require more control, performing cropping, scaling etc., feel free to copy-paste other options from the examples given in previous sections.

# Thoggen

Thoggen (<http://www.thoggen.net/>) is a simple, easy to use DVD extraction program for GNU/Linux, to create Theora videos. Note that only the video part of DVDs can be encoded; any menus present on the DVD are going to be stripped out.

Encoding a DVD using Thoggen involves two steps:

1. Selecting the DVD titles to extract and encode
2. Configuring parameters of the encoding process

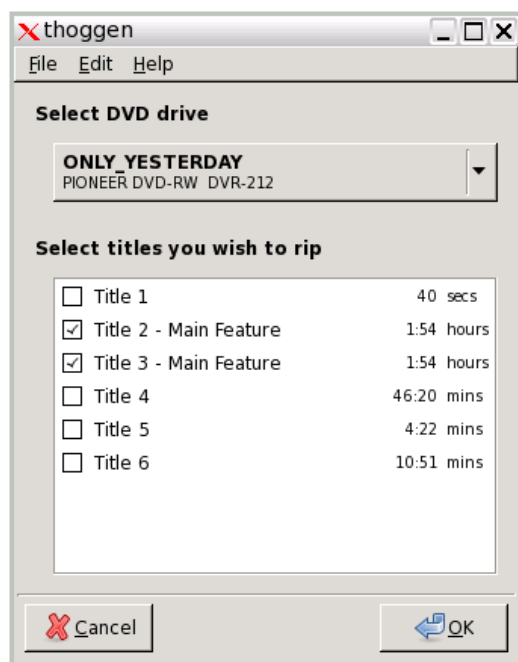
If you do not have Thoggen installed you will have to first do this of course. If you are running Ubuntu simply type this in a terminal window :

```
sudo apt-get install thoggen
```

You will be asked for your password, and then Thoggen will be installed.

## Selecting the DVD Titles to Encode

Upon starting, Thoggen will automatically detect any DVD media present in any DVD drive, presenting a list of titles it found on the DVD:



You are asked to select the titles that are going to be extracted and converted to Theora. The longest title is selected by default, as this is usually the main video. If this is not what you actually want, change it to suit your needs.

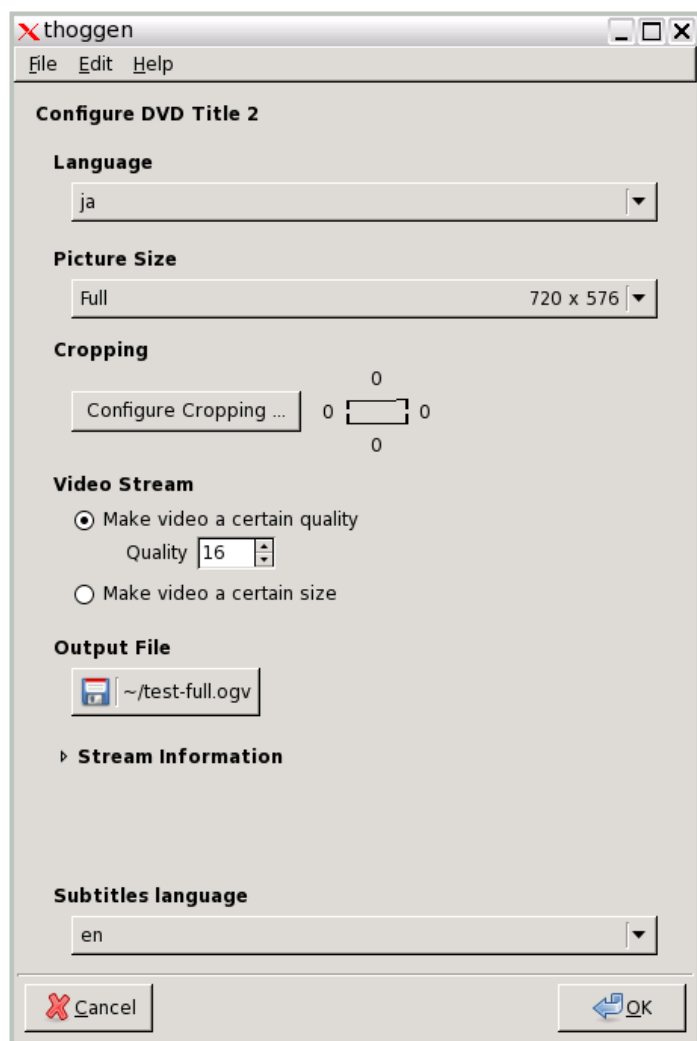
Note that you may also convert from a DVD image on your hard drive, instead of directly from a DVD. To do this select the image location from the 'File' menu. The same list of titles will then be displayed, as if it was taken directly from the physical DVD.

When done selecting, press *OK* and you will be taken to the next dialog.



## Configuring the Encoding Process

The following dialog now allows you to specify parameters of the encoding process:



If you selected more than one video in the previous step, this window will be presented once for each of them. For multi-language DVDs use this dialog to select which languages to include. Select the quality and video size, and where to save the video. If unsure which option to choose, just go with the defaults. Note that video quality is specified as a number in range 1 (lowest quality, smallest output file) to 63 (highest quality, largest file)

When you are happy with the settings, press *OK*, to start the encoding. You are then presented with a slideshow preview of the video along with a progress bar, while waiting for the encoding process to finish.

You can now use VLC, or any other player supporting Theora, to view your copy of the DVD, without having to actually search for and insert the DVD into the drive whenever you want to watch your movie.

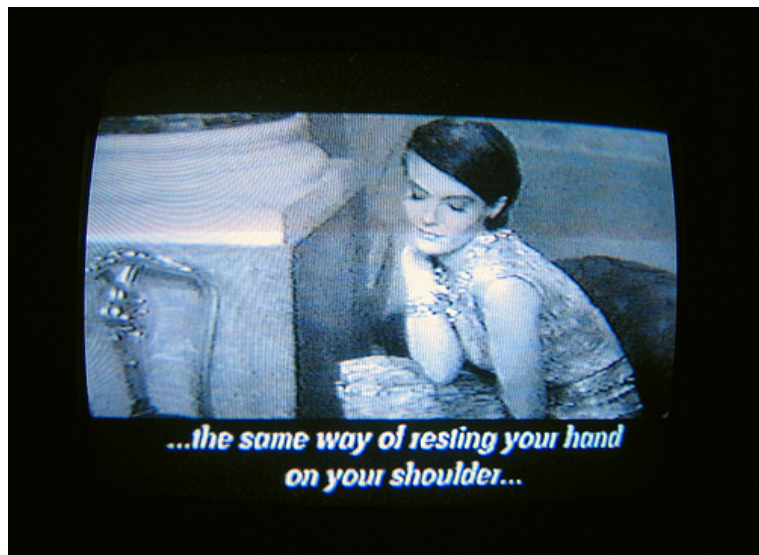
## A Note on Cropping

Thoggen can remove borders at the left, right, top and bottom when encoding the DVD, available from the *Configure Cropping* button. It is often actually a good idea to remove about 5% the movie's width and height. This is due to the fact that DVDs are manufactured with ancient analog televisions in mind, hence there are usually borders around the movie. Older DVDs sometimes contain noise and artifacts in the movie borders, only visible when playing the DVD on a PC.

For a typical DVD resolution of 720x576 pixels, try cropping 8 pixels on the left, right, top and bottom, leaving a total frame size of 704x560.

# Subtitles

Most videos include people speaking in a certain language; to make these video accessible and understandable to a global audience the video must be subtitled or dubbed. Subtitles are by far the easiest to produce: audio dubbing requires time and software expertise, but you can create subtitles with just a video player and a text editor.



## Finding Subtitles

Before starting a subtitling translation project, it's worth searching for existing subtitles, particularly if the video is a well-known or commercial work. For example, if you are including a scene from an American documentary in a video, there are resources to search for subtitles for this material. However, outside of well-known video and films, pre-created subtitles are rare, and subtitles available under an open license are even harder to find.

There are a few issues that come up when searching for subtitles. For cinematic films, for example, there are almost invariably many different versions of the film. One can imagine that any extra scene, extended title sequence, or formatting change can alter the timing of subtitles onscreen which many times renders subtitles useless. Therefore, it is important to find subtitles that are accurate for the audio of the particular film version. There are free software tools like Sub Downloader (<http://www.subdownloader.net/>) that help with this problem by matching subtitle sets to specific film versions. Another issue that comes up is the file format of the subtitle file itself. There are different formats for different types of video as well as different types of physical media (HD, DVD, Blu Ray etc.) which affect the selection of subtitles for a given piece of film.

The following are resources for finding subtitles :

- OpenSubtitles.org: <http://www.opensubtitles.org/en>
- TinyTM: <http://tinytm.sourceforge.net/>
- DivX Subtitles: <http://www.divxsubtitles.net/>
- AllSubs.org <http://www.allsubs.org>

## Subtitle File Formats

A subtitle file format specifies the format of a file (text or image) containing the subtitle and timing information. Some text-based formats also allow for specifying styling information, such as colors or location of the subtitle.

Some subtitle file formats are:

1. Micro DVD (.sub) - a text-based format, with video frame timing, and no text styling
2. Sub Rip (.srt) - a text-based format, with video duration timing, and no text styling
3. VOB Sub (.sub, .idx) - an image-based format, generally used in DVDs
4. Sub Station Alpha / Advanced Sub Station (.ssa, .ass) - a text-based format with video duration timing, and text styling and metadata information attributes.
5. Sub Viewer (.sub) - a text-based format with video duration timing, text styling and metadata information attributes.

We will only focus on the subtitle format Sub Rip (.srt), which is supported by most software video players and subtitle creation programs. SRT files can also be created and edited by text editors, or more specialised software like Jubler, GnomeSubtitle, Gaupol and SubtitleEditor.

## Editing SRT files

An SRT subtitle file is just a text file that is formatted in a simple way so the player can read it and co-relate subtitles to the time they should be played in the video. SRT is a very simple, widely used subtitle format. If you find an existing SRT file for the video you need to subtitle, it's easy to create subtitles for other languages when you know how an SRT file works.

A SRT file is made of a list of lines looking like this:

```
1
00:03:05,260 --> 00:03:07,920
Hello, world.
```

The first line is the number of the subtitle, incrementing from 1 to as many as needed. The second line is the time at which the subtitle appears and disappears in the video, and is recorded in hours:minutes:seconds,milliseconds. The third line, and any lines after it up to the first blank line, are the subtitle text. One blank line is required to mark the end of the subtitle text. You can add as many such triplets as you need for the remaining subtitles.

For the above example, it just means that the first subtitle shows up at 3 minutes 5.26 seconds into the video, disappears at 3 minutes and 7.92 seconds, and reads "Hello, world". That's it.

## Creating subtitles from scratch

To create a subtitle file from scratch, you may want a more advanced tool that makes it easier to assign subtitles to specific points in a video. Jubler, GnomeSubtitle, Gaupol and SubtitleEditor are all free software tools, and worth checking out.

FLOSS Manuals has created a complete guide to Jubler (which is both free software and cross-platform). You can find the guide here: <http://en.flossmanuals.net/jubler>

# Distribution

Using a subtitle format like .SRT means that you can distribute subtitles for many different languages without distributing a different version of your video for each language. You just need to make separate .SRT subtitle files for every required language, and make those files available on the web.

This strategy is very common in the world of subtitles. Including the subtitles as a separate files allows that file to be accessed, changed, or removed without affecting the video file itself. The disadvantage of this technique is that the subtitle file format becomes an issue. Players must accept the format in order to properly display the subtitles. And users must know a little bit about how subtitles work in order to play the subtitle files correctly. If you're distributing .SRT files with a downloadable video, be sure to include some instructions on how to retrieve and play the subtitles. If you're distributing video on the web, you can use HTML5 and javascript to offer different subtitle tracks on your webpage.

Its also possible to explore embedding multiple .srt files within the video file itself. This provides the user with the option to choose from among the translations you make available (or to display no subtitles at all) without the need for additional subtitle files. Patent-unencumbered video container formats that support this include Matroska Multimedia Container (MKV) and the Ogg container format.

# Embedding Subtitles

If you want your video file to contain a subtitle file, so you don't have to distribute the .srt file separately, you need to embed the subtitle file into the video. The video encoding tool **ffmpeg2theora** has a few command line options to include subtitles in your video.

ffmpeg2theora is available for most operating systems, including Windows, Mac OS X, and GNU/Linux.

Related to subtitles three commands are important:

- `--subtitles` pointing to a subtitle file in SRT format,
- `--subtitles-language` to define the language of the subtitles
- `--subtitles-encoding` to specify the character set of the subtitles file used.

Let's have a look at some of the required options for using ffmpeg2theora for embedding srt files in a Theora video file.

**subtitles-language** - This option sets the specified language. Every language has a standard code, which helps people describe a language, whatever their own language. For example, in English the language spoken in Germany is called German, but in Germany, it's called Deutsch. To prevent confusion, there is an international standard (ISO 639-1) that represents each language with a two letter code. In our example, the code for German is 'de'.

**subtitles-encoding** - This option specifies the encoding standard for text, a complexity necessary given varying strategies for representing the wide range of characters used by all languages on earth. For a long time, computers used 7-bit character sets of 127 characters to represent the alphabet and other writing symbols. For example, US-ASCII has 94 printing characters and 33 control codes. Numerous 8-bit character sets, with 256 codes, have appeared since then for alphabets and syllabaries, and several encoding systems using 16-bits for writing systems based on Chinese characters. However, 7 or even 8 bits is not enough space for all the typographical symbols in even one alphabet, much less for the dozens of writing systems in use today. People created the Unicode Character Set to support all languages at once. The UTF-8 encoding of Unicode is specified for use "on the wire", that is, in all external communications between systems.

However, a lot of people still use old encodings. The bad thing about these is that they overlap, using the same set of codes for completely different characters. The usual result of rendering a text according to an incorrect encoding is gibberish. So, by default, subtitles are expected to be in Unicode UTF-8 encoding. If they are not, you need to tell ffmpeg2theora. If you're writing in English, chances are you'll be writing in ASCII, ISO-8859-1 (Latin-1), or possibly Windows code page 1252. By design, US-ASCII is a subset of UTF-8, so you'll be OK there, but you will get into trouble if you use any extension of ASCII in a Unicode context.

## Example commands for subtitle embedding

Here are a few examples that take an existing mp4 video file (input.mp4) and output a ogg video file (output.ogg) with embedded subtitles :

If you have a subtitles file in English (the language code for English is 'en'):

```
ffmpeg2theora input.mp4 --subtitles english-subtitles.srt --subtitles-language en -o output.ogg
```

If you have a subtitles file in Spanish, encoded in latin1 :

```
ffmpeg2theora input.mp4 --subtitles spanish.srt --subtitles-language es --subtitles-encoding lat
```

There are other subtitles options for ffmpeg2theora, but these are the main ones.

## Adding subtitles to an existing video

If you have a Theora video with no embedded subtitles, it's easy to add some too, without the need to encode the video again. Since each subtitles language is stored in the Ogg file separately, they can be manipulated separately.

Internally, subtitles embedded in an Ogg file are encoded as **Kate streams**. Such streams are created by ffmpeg2theora, but can also be created 'raw' from a SRT file. The kateenc tool does this. On Ubuntu kateenc is part of the kate-tools package. To install do this:

```
sudo apt-get install libkate-tools
```

For instance, the following creates a new English subtitles stream from a SRT file. Remember, the code for English is 'en':

```
kateenc -t srt -o english-subtitles.ogg english.srt -c SUB -l en
```

Now you've got a single subtitles stream, which you can add to your Theora video:

```
oggz-merge -o video-with-subtitles.ogv original-video.ogv english-subtitles.og
```

On Ubuntu oggz-merge is part of the oggz tools package, to install, do this:

```
sudo apt-get install oggz-tools
```

In fact, the oggz tools allow more more powerful manipulation of all the different tracks in the video, so you can add more audio languages too, etc.

# Playing Subtitles

There are several easy ways to play back subtitles with an Ogg Theora video.

## VLC

First, install VLC from the website (<http://videolan.org/vlc>) if you haven't already. These instructions assume you have a file or DVD with subtitles which you want to display while you are playing the Video.

There are three ways you may want to use VLC to display subtitles.

1. From a DVD
2. From a Multilingual file (ie Matroska)
3. From a separate subtitle file which is distributed with the Video file.

## Play subtitles on a DVD disk

To do this put the DVD disk into your DVD drive. Open up VLC player and select **File > Open Disk**.

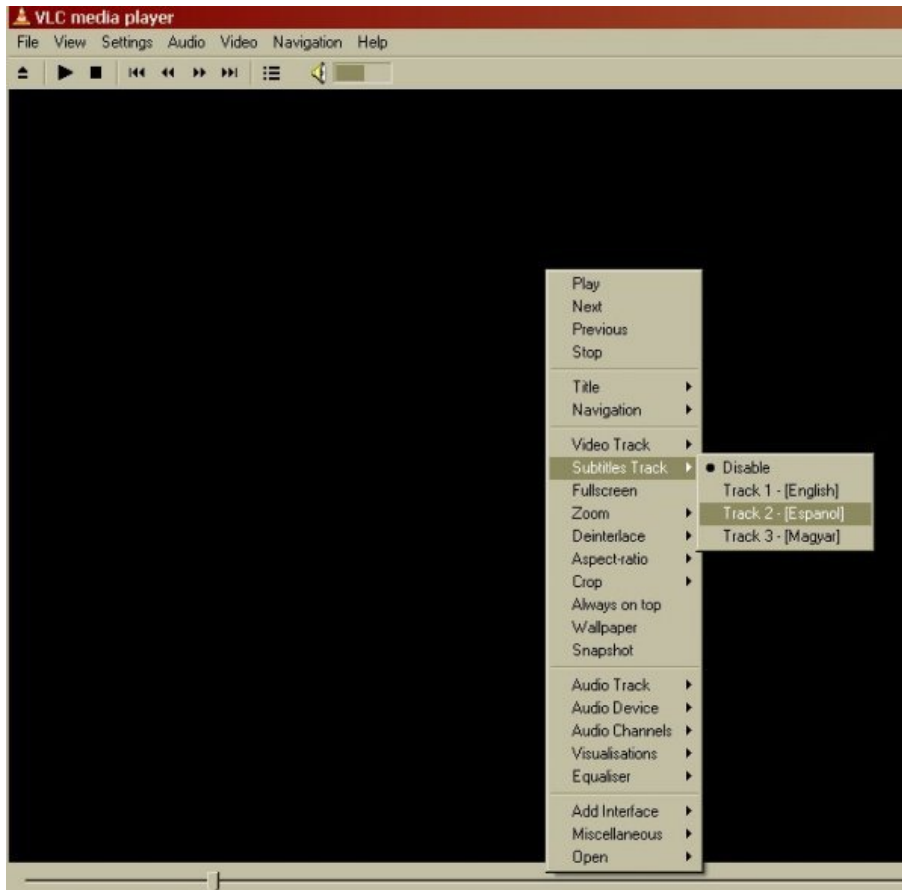


Enter the DVD Drive letter. It may appear automatically. On Windows this may be drive D:, and on GNU/Linux something like /media/dvd.



Then click OK. The menu page of your DVD should appear. Click on the video you want to watch. Then when the video starts quickly right hand click the mouse on the Video image. Select the Subtitle track you wish to view.

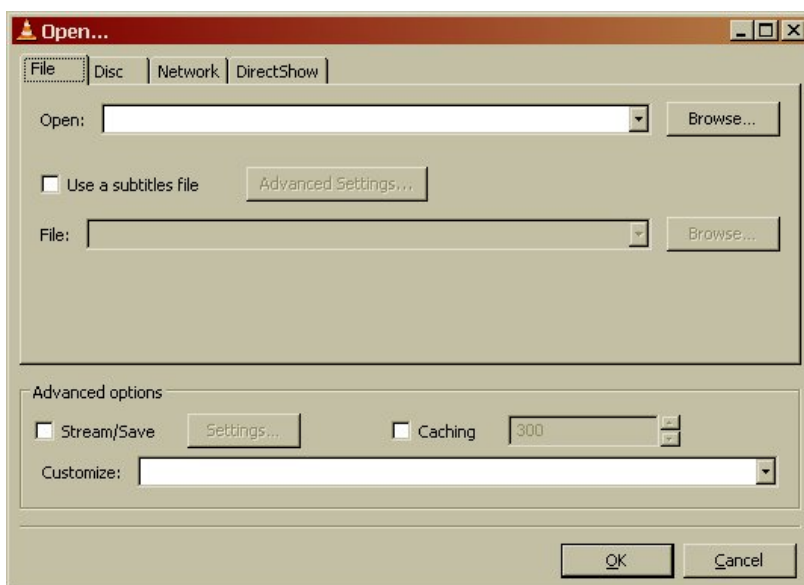




The subtitles should then appear on screen.

## Play subtitles in Matroska files

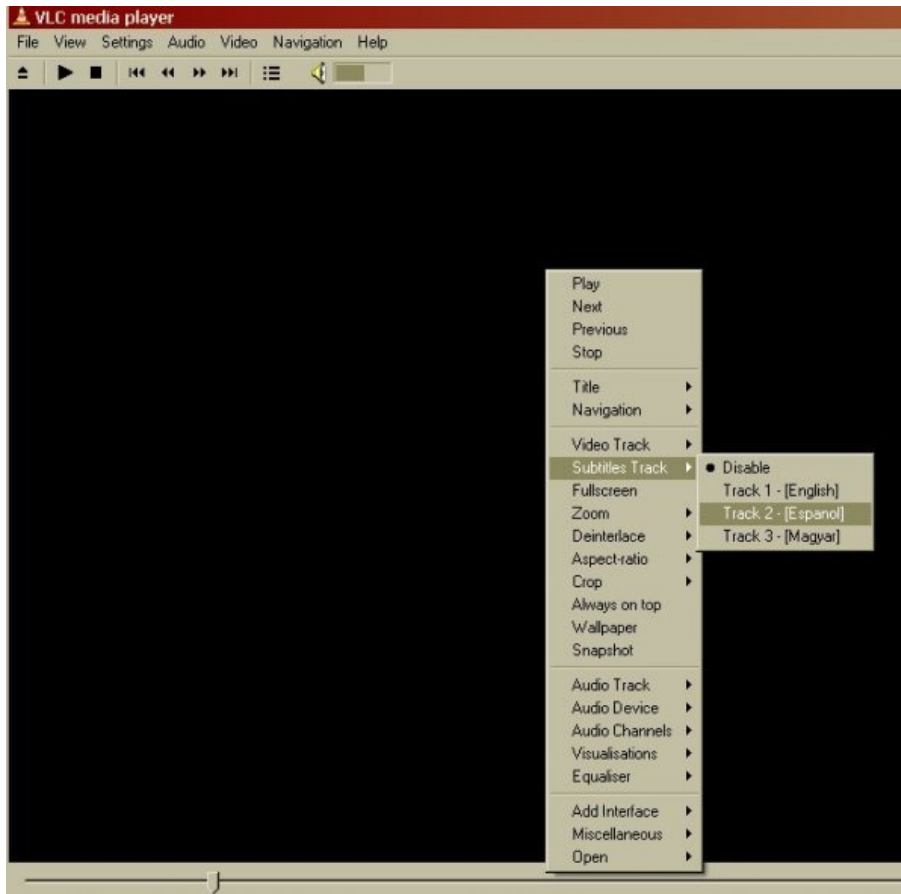
The process for this is exactly the same as above except when starting the process you select **File > Open File**. You then see this screen.



You should then click on the **Browse** button to select the video file you want to play. If this file is a matroska file with an \*.mkv extension then you can click OK after browsing for the file as the file already has the

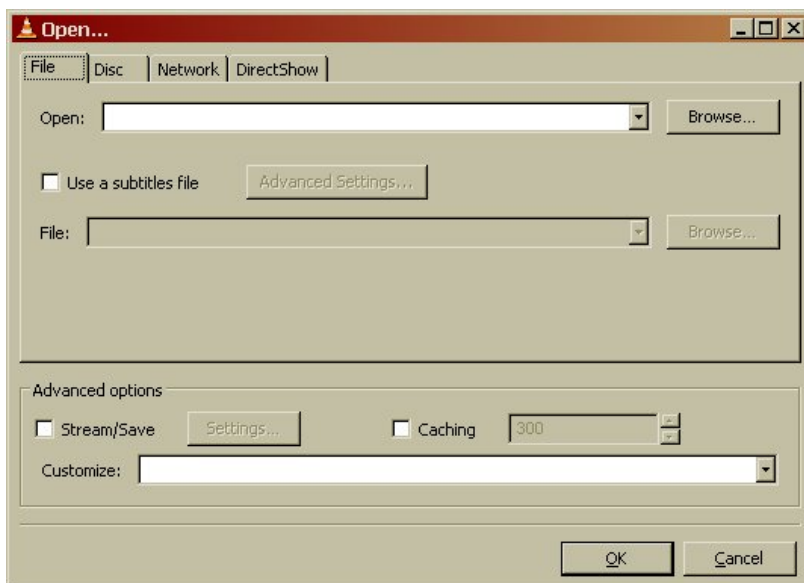
subtitle information.

Then Select the subtitle language stream by right clicking the video screen and selecting **Subtitle Track >** and choose the language



## Play External Subtitles

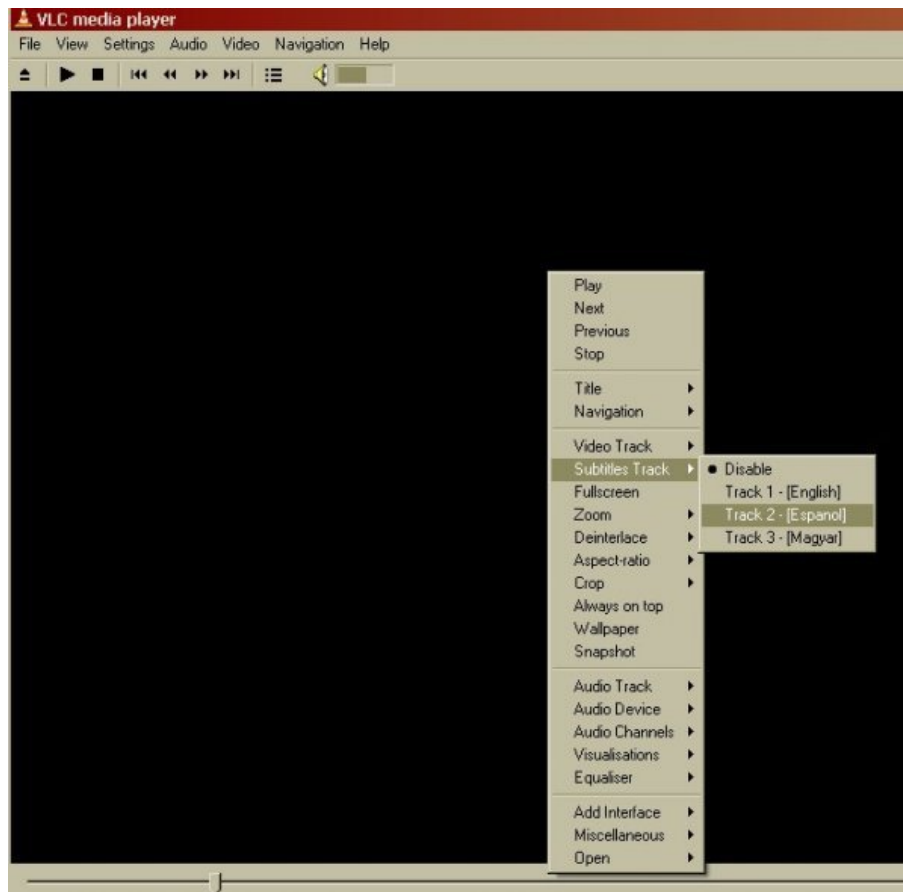
If you want to play an external subtitle file for example a srt file. Select **File > Open File**



In the **Open** box click the **Browse** button and choose your video file.

Play subtitles in Matroska files

Then put a tick in the box **Use a subtitle file**, and click Browse to locate your external subtitle file.



Then Select the subtitle language stream by right clicking the video screen and selecting **Subtitle Track >** and select the track of subtitles (for an external file like an srt file there will normally only be one track).

# Publishing

Depending on how you created the subtitles - embedded in the video or as a separate .srt file - you have different options to publish it online with your video.

## Hosting external subtitle files along with video on a web server

We will assume you have one or more subtitles (in SRT format) and the video itself.

### HTML5 *video* tag and Javascript

You can offer a web preview of Theora video alongside a given .srt with the help of jquery.srt.

Firstly, notice below that we will integrate JQuery, a popular GPL Javascript library (<http://jquery.com/>), and an example Javascript implementation of displaying subtitles in a webpage from a SRT file available at: <http://v2v.cc/~j/jquery.srt/jquery.srt.js>

A simple HTML document excerpt is shown below, which includes the modification to include the Javascript files and to reference to your subtitle file. Only one subtitle file can be referenced at a time, unless you start developing further with Javascript.

It is a small script that can load an srt file and display it inside a *div* on your page, under the video or as an overlay.

```
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript" src="jquery.srt.js"></script>

<video src="http://example.com/video.ogv" id="video" controls>
</video>
<div class="srt"
    data-video="video"
    data-srt="http://example.com/video.srt" />
```

this example could look like this on your page.



make sure there are no illegal books being printed.

For another example using subtitles in several languages, you can have a look at this demo from Mozilla <http://people.mozilla.com/~prouget/demos/srt/index2.xhtml>

Another example using multiple subtitles and providing an interface to select them can be found at <http://www.annodex.net/~silvia/itext/>.

## Hosting video with embedded subtitles on a web server

If you have Ogg files with embedded subtitles and want to display those in the browser, you can use the *video* tag right now. However, not all browsers support the video tag. Cortado, a Java plugin that can play Theora videos, also has support for embedded subtitles, and it works in all browsers that have the Java installed. You can get the latest version of Cortado from <http://www.theora.org/cortado/>

```
<applet code="com.fluendo.player.Cortado.class" archive="cortado.jar"
  width="512" height="288">
  <param name="url" value="video.ogv"/>
  <param name="kateLanguage" value="en">
</applet>
```

To use subtitles with Cortado, you pass it the url to the video with embedded subtitles and in the same way you pass a parameter for the subtitles track to use. The easiest way is to select the language you want via the `kateLanguage` parameter:

```
<param name="kateLanguage" value="en">
```

It is also possible to change or disable the subtitle via Javascript. If you want to switch to French subtitles, you could set the `kateLanguage` option from your script by calling:

```
document.applets[0].setParam("kateLanguage", "fr");
```

# Web Video Accessibility

When we talk about Web video here, we explicitly refer to video published in Ogg Theora/Vorbis format inside a Web browser that supports the HTML5 video element.

Accessibility of video refers to several different aspects of usability of video, depending on what user group we are looking at. So, before diving into the different aspects and how they can be supported, we list the user groups and their specific requirements.

## Accessibility user groups

1. **Non-native speakers:** when watching a video in a foreign language, it is impossible to follow. For this purpose, subtitles have been invented. *Subtitles* are time-aligned transcriptions of the spoken words in a video, which have been translated into different non-native languages. Alternatively, an audio track in the native language can also be created, which then replaces the original audio track. This is called *dubbing*.
2. **Deaf or hard-of-hearing (HoH):** when a HoH person is watching a video, it is impossible to follow because none of the sounds and spoken words are perceived. *Captions* are time-aligned transcriptions of the spoken words and the noises, sound effects, music and other sounds in a video.
3. **Blind or vision-impaired (VI):** when a VI person is trying to "watch" a video, without special help it is not possible to interact with the video player controls in the first place, and secondly it is impossible to follow the video because none of the visual displays are translated into signals that VI person can perceive. Firstly then it is important to make the *video controls accessible*. Secondly it is important to provide a time-aligned description of the visual channel. There are two senses that can be used to replace the visual channel: hearing and touch. In order to provide an aural representation of the visual content, we can either create a *spoken audio description (AD)* through an additional audio track for the video, or we can create a *textual audio description (TAD)* that a screen reader will read out in a time-aligned manner. Similarly, the TAD can also be output to a braille device, such that a VI person can perceive the visual channel through touch.

## Accessible HTML5 video controls

A key accessibility challenge for browser vendors with the HTML5 video element is to make the default controls accessible through the keyboard. The HTML5 video element provides an attribute called *controls* which requests the browser to create default controls on top of the video.

Here is what the current specification says:

â This user interface should include features to begin playback, pause playback, seek to an arbitrary position in the content (if the content supports arbitrary seeking), change the volume, and show the media content in manners more suitable to the user (e.g. full-screen video or in an independent resizable window).â

In Firefox 3.5, the controls attribute currently creates the following controls:

- play/pause button (toggles between the two)
- slider for current playback position and seeking (also displays how much of the video has currently been downloaded)
- duration display
- roll-over button for volume on/off and to display slider for volume
- FAK fullscreen is not currently implemented

Further, the HTML5 specification prescribes that if the *controls* attribute is not available, user agents may provide controls to affect playback of the media resource (e.g. play, pause, seeking, and volume controls), but such features should not interfere with the page's normal rendering. For example, such features could be exposed in the media element's context menu.

In Firefox 3.5, this has been implemented with a right-click context menu, which contains:

- play/pause toggle
- mute/unmute toggle
- show/hide controls toggle

When the controls are being displayed, there are keyboard shortcuts to control them:

- *space bar* toggles between play and pause
- *left/right arrow* winds video forward/back by 5 sec
- *CTRL+left/right arrow* winds video forward/back by 60sec
- *HOME+left/right* jumps to beginning/end of video
- when focused on the volume button, *up/down arrow* increases/decreases volume.

To make these controls accessible to VI users, Firefox exposes them to screen readers using MSAA or AT-SPI. It implies having to use focus mode for now. Exposure through iSimpleDOM interfaces on Windows (<http://www.marcozehe.de/2009/06/11/exposure-of-audio-and-video-elements-to-assistive-technologies/>) are still in development. Once in focus mode, the keyboard shortcuts listed above make the video controls accessible.

## Providing video accessibility data

As described above, accessibility for a particular video is provided through creating additional data that accompanies the original video. A fully accessible video may consist of all of the following:

- original video track
- original audio track
- audio tracks that contain dubs in foreign languages
- captions in all languages (which also covers the need for subtitles)
- audio tracks that contain spoken audio descriptions in all languages
- textual audio descriptions in all languages

All of the mentioned data that provides accessibility to video is time-aligned with the original video. It can be provided in two different ways:

- **out-of-band (external files)**: as a separate text or audio file that relates to the original video. A typical example of out-of-band accessibility data are subtitles in srt files (see section on subtitles).
- **in-line (embedded)**: multiplexed together with the original video inside a single binary file. A typical example of in-line accessibility data are srt subtitles encoded in Kate tracks in Ogg (see section on embedding subtitles).

## Publishing accessible video on the Web

The current HTML5 specification does not contain explicit means to publish, style, and position accessibility data for audio and video. The suggestion is to use in-line accessibility data and have the video decoder deal with it. Also, the suggestion is to use javascript where there is a necessity for out-of-band accessibility data. There is work in progress on improving this situation. The idea is to expose accessibility data to the Web browser in the same manner independent of whether the data originates resides in-line or out-of-band.

Several demos have been made with **out-of-band** subtitles, captions, and audio descriptions and the HTML5 video tag:

- Raul Rouget's subtitle demo using srt (<http://blog.mozbox.org/post/2009/03/10/video-tag-and-subtitles>)
- Jan Gerber's subtitle demo using srt (<http://v2v.cc/~j/jquery.srt/>)
- Philippe Le Hegaret's caption demo using DFXP (select HTML5) (<http://www.w3.org/2008/12/dfxp-testsuite/web-framework/START.html>)
- Silvia Pfeiffer's caption & audio description demo using srt (<http://blog.gingertech.net/2009/07/29/first-experiments-with-itext/>)

These are all implemented using javascript, so you can learn from them. There is also a more detailed introduction to Jan Gerber's javascript library (see section on Publishing) for subtitle support in this Cookbook.

Silvia Pfeiffer's demo includes a proposal for how to associate out-of-band accessibility data through a new HTML5 tag with videos. This specification is continuing to evolve and is expected to eventually lead to native browser support of time-aligned accessibility data. A similar proposal has been made by Greg Millam from Google (<http://lists.whatwg.org/pipermail/whatwg-whatwg.org/2009-February/018600.html>). Such a specification may look as follows:

```
<video src="elephant.ogv" poster="elephant.png" controls>
  <itext lang="fr" type="text/srt" src="elephant.fr.srt" category="SUB"></itext>
  <itext lang="en" type="text/srt" src="elephant.en.srt" category="CC"></itext>
  <itext lang="en" type="text/srt" src="audiodesc.srt" category="TAD"></itext>
</video>
```

Note: this is an example proposal only, which is not currently supported natively by any browsers.



# What is streaming?

Streaming Media is the term used to describe the *real-time delivery* of audio and video over the internet. Streaming Media allows for *live transmission* of audio or video over the internet, transforming the internet into a broadcasting medium.

The main difference between a streaming service and an online delivery (or archive) service is that streamed data do not specify a start nor an end. A stream is a continuous flow of video or audio data to which a viewer or listener can just connect. So a stream will continue even if no one is connected to that stream.

## Varieties of Streaming Services

There are two types of delivery of audio and video data over the internet: Playlist Streaming and Live Streaming.

### Playlist Streaming

The delivery of a stream of pre-recorded media files without any interaction over the internet can be called playlist streaming. This type of stream is built upon a list of files that should be streamed. Many players are able to stream files or list of files (e.g. VLC). Even if the files are acquired from an archive, this is a kind of streaming, as the audience does not read the complete file, but joins a stream. This delivery usually occurs using the http protocol.

### Live Streaming

The delivery of live audio and/or video over the internet. This allows the user to experience an event as it occurs in realtime. There are many examples of this, such as online radio or viewing live performances. Only True Streaming supports live streaming.

This manual will mainly deal with True Streaming of both live and archived content.

## Delivery

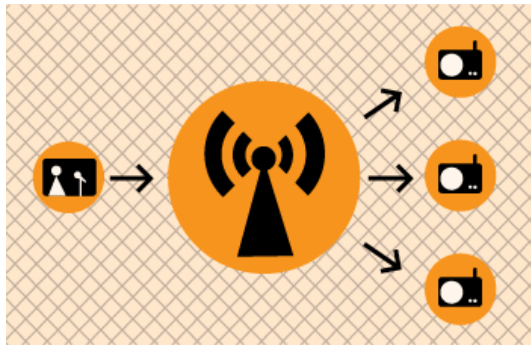
Providing video files (especially theora videos) is nearly as easy as providing pictures on a webpage. All that is needed for such a scenario is a web server. This is often the cheapest way to deliver content on a small scale. This method cannot be used for live streaming and does not enable advanced features such as multiple bitrate encoding. However, the streamed delivery of archived videos (such as archive.org or youtube) generates most webtraffic nowadays.

To enable live streaming, and to gain full functionality and efficiency, a streaming media server is required. This server is usually standard server hardware but with the necessary streaming server software installed. It is quite normal to install a streaming server on the same machine as an existing web server.

## Streaming Servers

Perhaps a good way to understand how streaming works is to imagine a radio station.

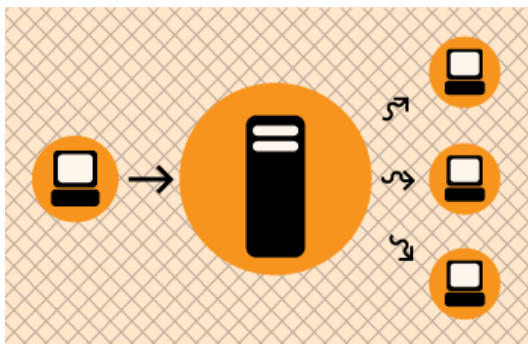
A radio station is comprised of three components - a *studio*, a *transmitter*, and your audience's *receivers*.



The basic diagram above shows how a transmitting radio station works. The radio studio is the source of the audio. In this space there are usually mixing desks, cd-players, minidisc players, turntables etc. From the studio an audio signal is sent to the transmitter. This can be sent from the studio to the transmitter by either a cable (sometimes called a "landline") or by a microwave link. Then the transmitter sends the audio via FM so that radio receivers (tuners) can pick it up and play it.

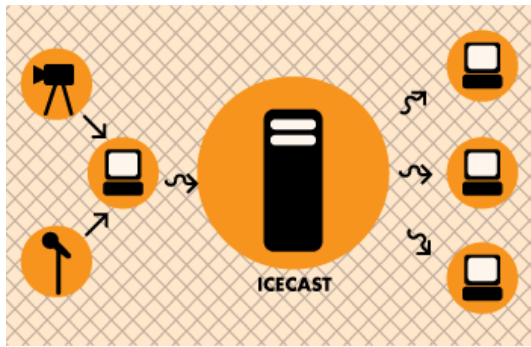
Radio works this way because it is trying to distribute the studio audio to as many people as possible. If you imagine the radio studio without the transmitter then the station would have a somewhat reduced audience! Essentially only those people who could get there (and fit into the studio!) would be able to listen. So the transmitter works as a distributor, allowing more people to connect via their radio receivers and hence enlarge the potential audience.

This is a close analogy to why streaming exists and how it works. If you are just playing audio on a computer in your room then the audience isn't going to be so big... so, we utilise streaming to distribute this audio to more people.



The analogy is obvious: the computer replaces the radio studio, the streaming server replaces the transmitter, and your listeners connect to the server using computers, just like radio receivers to the broadcast signal. The analogy can be taken quite a long way. Having a bigger radio transmitter is like having more bandwidth available at the streaming server - both allow more people to connect.

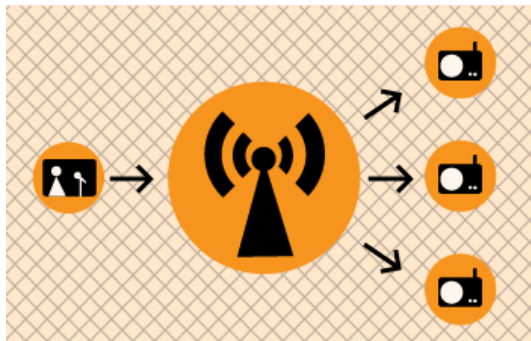
# Icecast



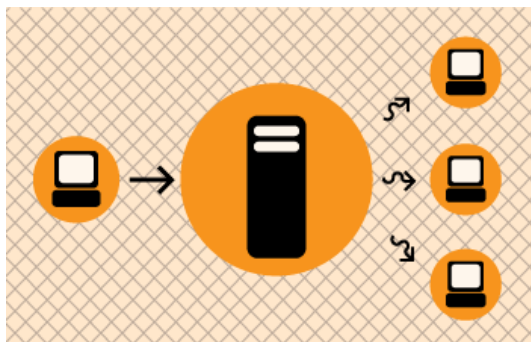
In the world of online audio and video there is a group of technologies and techniques referred to as **streaming**. Streaming, at its simplest, is the delivery of audio and video over the internet in **realtime**. Traditionally streaming has been used for delivering live internet radio, or for broadcasting events over the internet so you can watch them at home.

Streaming is often talked about using terms from the broadcasting industry as live internet audio and video are often seen as being analogous to broadcast television and radio. Hence you might talk about 'internet radio' to refer to live online audio. Sometimes people refer to sending live audio or video using streaming as 'broadcasting on the net' or as a 'internet broadcast'. These terms are helpful for helping us understand the intended purpose of streaming but don't take them too literally. However it is useful to refer to these models when explaining the role of Icecast.

With broadcast television or radio there is transmitter that distributes the signal sent from the studio, to your television or radio.



With internet broadcasting (streaming) you replace the transmitter with a **streaming server**, which is really a kind of software.



**Icecast** is this kind of software. It enables you to distribute live audio and video across the internet in realtime. Note that this is only part of the equation. You also need a **stream encoder** that can send the original audio or

video stream to Icecast. Icecast distributes the stream, it does not create the stream - that's the job of the encoder.

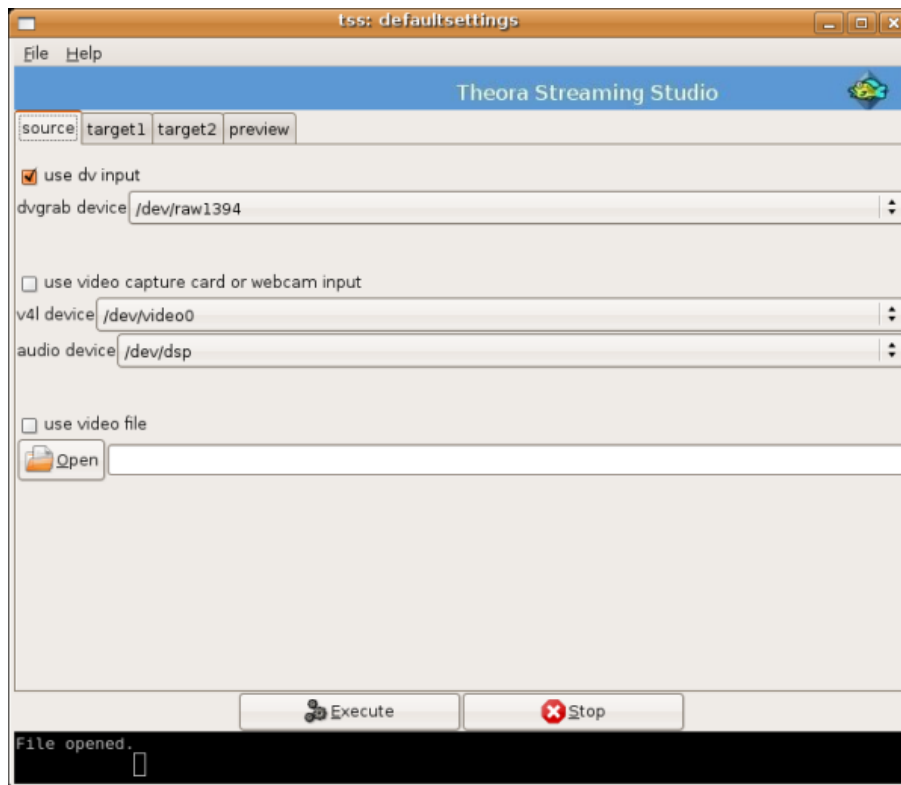
Icecast is a very mature technology and very robust. It supports the delivery of many different formats of streaming audio (including MP3 and Ogg Vorbis) and it supports the delivery of Ogg Theora for streaming video. Icecast runs on **Linux** and **Windows**. If you wish to install it on Linux you will need to have some experience with Linux; specifically you will need to know a bit about how to install software from the command line, and be comfortable editing text based configuration files. The Windows installation process is a little bit more straight forward but you still need to be comfortable editing text based configuration files.

Installing and running Icecast is not recommended for inexperienced users.

# TSS

**TSS (Theora Streaming Studio)** is an excellent encoder for sending **Ogg Theora** streams to an **Icecast** server. There is a home page for TSS that is worth reading for an overview of what it does and there are also installers available :

<http://gollum.artefacte.org/tss/>



TSS is a **GUI** (graphic user interface) encoder, which means you don't have to do anything tricky on the **command line**. TSS can send out one or two streams with different settings, from the same video source. The video source can be digital video (DV) or analog video via a video capture device such as a webcam, USB video-in or video card with video-in. Bear in mind that a fast computer with at least 2gig ram will be required to stream to 2 targets without frame drops or loss of audio synchronisation.

TSS can also display a preview of the outgoing stream, and can simultaneously archive the streams to disk (but that depends a little on how good your computer is).

TSS actually manages the command line softwares :

1. **dvgrab** - grabs the video from a live video source such as a camera
2. **ffmpeg2theora** - converts video files to Ogg Theora
3. **oggfwd** - sends the video to a streaming server

TSS simply takes the configuration you set in the GUI and creates the appropriate command line using these three softwares so you don't need to enter in complex commands manually.

One can always copy the command line being used by TSS, it will be printed out on the status window, the small black space we can see above under the **Execute** and **Stop** buttons. Simply scroll your mouse pointer and select the text. You could then run this piped commands directly from command line, saving some resources by avoiding launch of the graphical system.

TSS main developer is Lluís Gàmiz i Bigordà, the current release (February 2009) is 0.2. The project status is active (always good to know!).

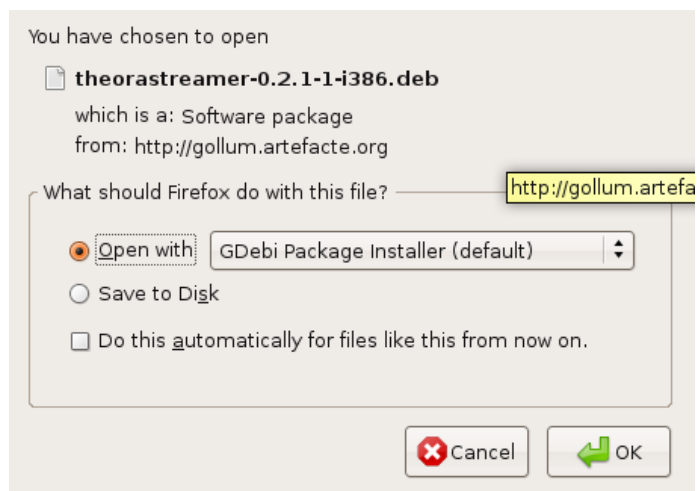
## Installing TSS

TSS can only be installed on GNU/Linux. We will look at installing TSS on Ubuntu.

First visit the TSS download page :

<http://gollum.artefacte.org/tss/#download>

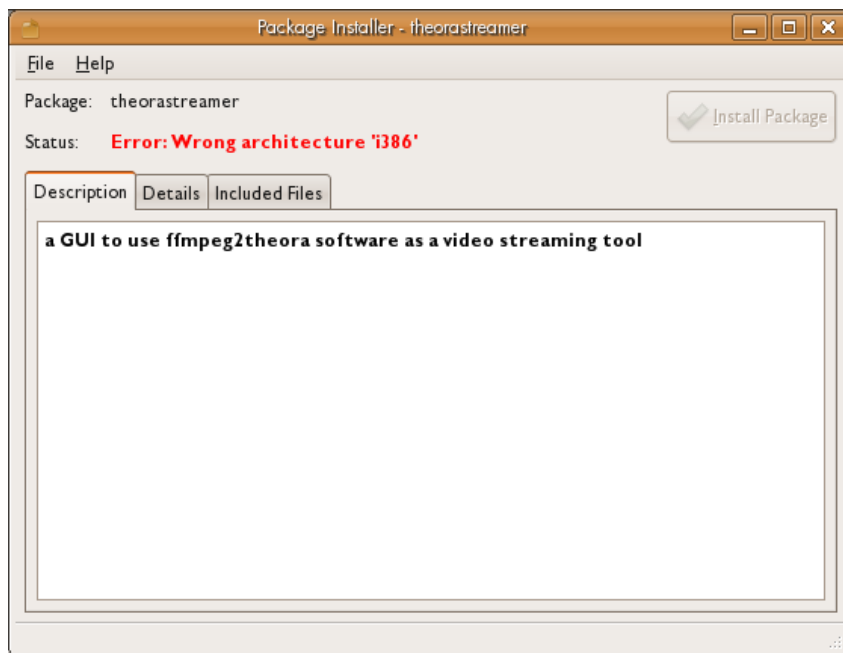
Here you will see a section with two links. One link is marked 'Debian / Ubuntu package'. We wish to install this version. Click on this link using FireFox and you will see a window appear like this :



The option selected by default is what we want to use. If we open the installer with **GDebi Package Installer**, then the installer will download and install automatically...sounds easy! So lets do this. Simply click 'OK' and it will install.

## Installing on 64 bit

When installing TSS you might see an error like this :



This means you are trying to install the software on a computer with a different 'architecture' than allowed by the software. Actually you can bypass this but you have to do it on the command line. First, instead of using GDebi you need to download the sources (.tar.gz file). Then from the same directory as where the installer is located you need to type this on the command line :

```
sudo apt-get install dvgrab ffmpeg ffmpeg2theora libtheora-bin oggfwf libvte-dev
```

The above command might take a while as there is a lot to download and install. When the process is complete follow this with these commands :

```
tar zxvf tss-0.2.tar.gz
cd tss-0.2
./autogen.sh
```

Note : the name of the file to be installed will probably be different from that listed above.

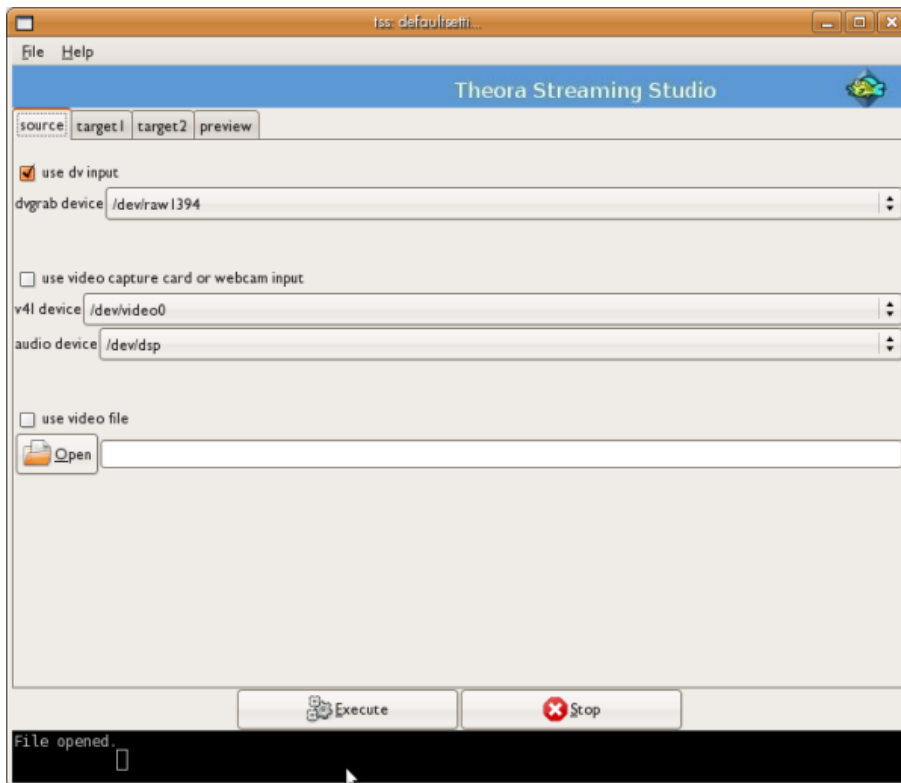
Next execute the following commands :

```
make
sudo make install
```

Now try typing 'TSS' on the command line, if it doesn't work then try this :

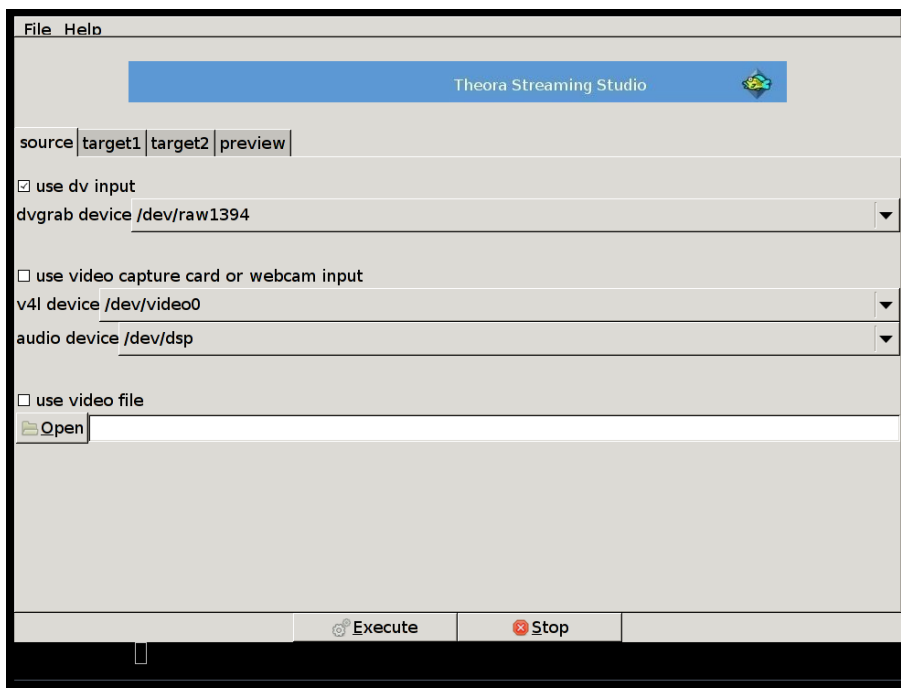
```
sudo cp src/tss /usr/bin/
```

The try starting TSS again from the command line :



## Using TSS

There are two main panels in TSS that you have to verify have the right settings in order to send a stream : the **source** and **target** tabs :

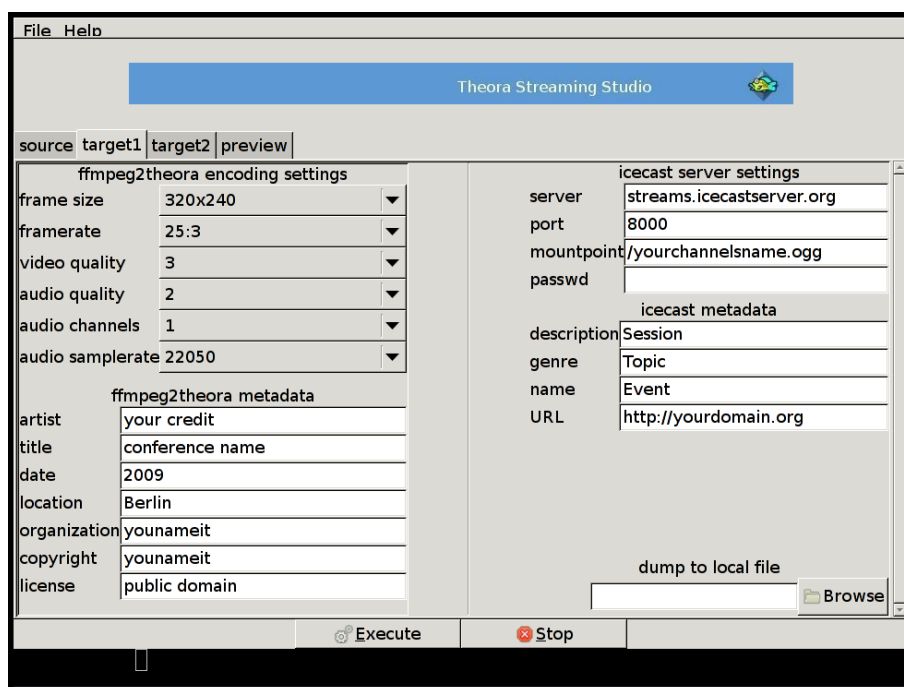


On the **source** tab one can choose the type of video input. When operating on a minimal setup there is a option to use a webcam via video for GNU/Linux (v4l). You can also choose a DV input - this is useful for streaming live video from a hand held video camera.



TSS also offers the possibility to stream an existing (archived) video file from your computer to an Icecast server.

The second tab is called **target1**, here we will define the stream settings for our main stream. Below is an example :



All fields in the **ffmpeg2theroa encoding settings** section must be completed.

It is important we keep in mind that any information we enter here will have a direct influence on the final output **bitrate** of our stream. You cannot simply nominate an outgoing bitrate, rather the outgoing bitrate is based upon the choices you make about frame size, audio quality etc.

Also, remember that the higher this bitrate is - the more bandwidth will be required to send it to the Icecast server, *and* the higher the bitrate the higher the bandwidth requirements of the audience connecting to your stream. So if some of our viewers are in Colombia where the average network connection is still around the 128kbps (**kb/s** or **kbit/s** is a unit of data transfer rate equal to 1,000 bits per second). This will mean that even though we have broadband connection sufficient to allow us to send a stream at a high resolution and high quality compression rate, we need to be careful to send a stream at a bitrate less than 128kbps, otherwise few people in Colombia would be able to see it.

The following categories relate to the options in the TSS encoding setting for both **target1** and **target2** :

**Frame size:** here we define the dimensions in pixels of the video stream. 320 by 240 has become a standard. For low bandwidth settings one might opt for 240 by 180.

**Frame rate:** This setting defines the number of frames per second we want to transmit. Different events will require different frame rates. In the case of a conference, where the subjects don't move or change position too much, a 12fps (fps = frames per second) setting could be appropriate. In the example above we decided to sacrifice on the frame rate in favor for a bigger frame size. We selected 25:3 that will be around 8fps.

**Video Quality:** This setting takes values in the range of 0 to 10. The default encoding quality for video of ffmpeg2theora is 5, one should use higher values for better quality.

**Audio Quality:** This setting takes values in the range of -2 to 10. The encoding quality for audio defaults to: 1. Use higher values for better quality.

**Audio Channels:** Sets the number of output audio channels. The choices are 0 (no sound), 1 (mono) or 2 (stereo).

**Audio samplerate:** Expressed in Hz (Hertz) this box sets the output sample rate of the audio.

Below the boxes dealing with the technical specifications of the outgoing stream is where you enter information that you want the audience to know :

ffmpeg2theora metadata	
artist	<input type="text" value="put your name here"/>
title	<input type="text" value="the title of the show"/>
date	<input type="text" value="5-5-2007"/>
location	<input type="text" value="Marrackesh"/>
organization	<input type="text" value="artefacte"/>
copyright	<input type="text" value="artefacte"/>
license	<input type="text" value="creative commons share alike"/>

The information in the **ffmpeg2theora metadata** boxes is potentially available to your audience. Whether they can see the information depends on how they are viewing the stream (what player/browser) and if they know how to access this information. You can enter any information here, it doesn't matter - ignore the categories (location, date etc) if you want to.

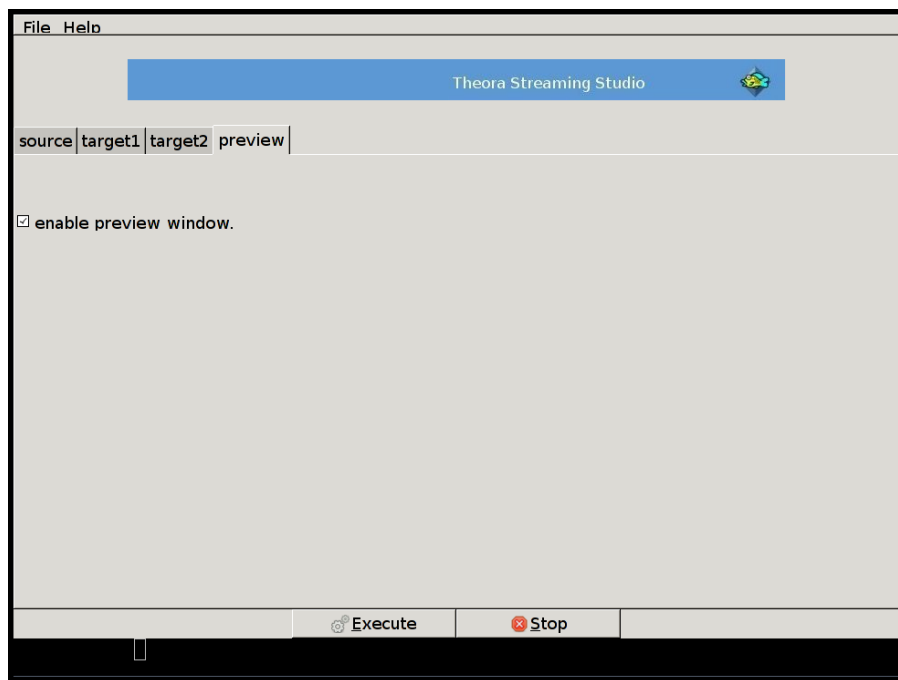
The **icecast server settings** and **icecast metadata** entry boxes are next.

icecast server settings	
server	<input type="text" value="gollum.artefacte.org"/>
port	<input type="text" value="8000"/>
mountpoint	<input type="text" value="/yourmountpointhere.ogg"/>
passwd	<input type="text"/>
icecast metadata	
description	<input type="text" value="between love and coding"/>
genre	<input type="text" value="underground burlesque"/>
name	<input type="text" value="the title of the show"/>
URL	<input type="text" value="http://gollum.artefacte.org/tss"/>

In the **icecast server settings** we have to define the address (**URL**) of the streaming server, the **port** this server is using (usually 8000), the name of our **mountpoint (channel)**, and the **password**. If you lack access to an icecast server there are few free projects out there dedicated to assisting people to stream using free software, check on [estudiolivre.org](http://estudiolivre.org) or [GISS.tv](http://GISS.tv) to name a few.

The **icecast metadata** settings is simply where you put information about your stream that you want the audience to know. This information will be displayed on the servers webpage (each Icecast server installation has its own webpage). Although they have specific titles you can put any information in here, it doesn't matter.

The last tab has the title **preview**.



If you check **enable preview window** you will see a preview of the live stream as you encode it. This video is not the same as the video coming from the Icecast Server, so it *always* pays to monitor *both* the preview and the stream coming from the server (monitor the incoming stream on another computer).

If you wish to send *one* stream you need only pay attention to the settings in the Source and target1 tabs. If you wish to send *two* streams you have to pay attention to the settings in target2 as well.

Once you have specified all the required information for each of the outgoing streams *and* have connected the correct video source (or you have chosen a working video file to stream) then you just press 'Execute' and the streaming should begin!

# Streaming with ffmpeg2theora

This method focuses on streaming video from a **DV** or **Firewire** camera using GNU/Linux (e.g. Ubuntu). The video stream will be **Ogg Theora** which means your users will be able to watch using Theora enabled player, FireFox 3.5, or in a java applet such as Cortado. To follow this you need to know how to use the **command line** and you need **sudo** access. If you have no idea what these things are, then you should consider reading a good book introducing you to the command line.

## Getting Ready

Please note : A **Firewire** camera is not the same as a **DV** camera. However for our purposes here they work in the same way. **Firewire** is Apple's name for the interface standard known as **IEEE 1394**, other manufacturers use other names such as i.Link (Sony) or Lynx (Texas Instruments) - but it doesn't matter, they are all IEEE 1394.

You will need the following:

- DV cam
- GNU/Linux machine with firewire inputs
- firewire cable
- internet connection
- access to a 'theora-enabled' Icecast2 server (you need the **IP address** of the server, the **port** you should use, and the **password**)

## Set-Up

Lets start setting up the software you need. We will need to install the following applications:

- dvgrab
- ffmpeg2theora
- oggfw

With Ubuntu you can run this command line:

```
sudo apt-get install dvgrab ffmpeg2theora oggfw
```

The above command should all appear on one line. You will be asked for your password, type this in and the installation process will begin. Now you have everything you need to get started. So first we need to plug the **dv** (or **firewire**) camera in to the computer. You need to attach the **firewire** camera to the **firewire** socket on the camera, and the other end of course goes into the **firewire** socket of your laptop or whatever computer you are using.

Now, turn on the camera.

Next you need to enter the following command line, which provides the streaming server details you have for the **Icecast2** (theora-enabled) server. The command is:

```
sudo dvgrab --format raw - | ffmpeg2theora -a 0 -v 5 -f dv -x 320 -y 240 -o /dev/stdout - | oggfw
```

Remember the command will have to be all on one line (the above example is not). Also replace the details below with the information you have about your Icecast server:

- *icecastserver*

- *8000*
- *pwd*

**icecastserver** should be replaced with the **hostname** or **IP address** of the streaming server. **8000** is the port number and is probably the same. **pwd** should be replaced by the **password** of your server. Lastly, you can replace **/theora.ogv** with your mount point, this depends on the configuration of your icecast server, but can be anything as long as it starts with a forward slash (/) and ends in **.ogg** or **.ogv**.

Now, you should be streaming! To check the connection use **VLC** or **Firefox 3.5**

# Streaming with VLC

VLC can stream theora files to an **Icecast2** streaming server. This works with VLC 0.9.4 and above on **Ubuntu**, and also on earlier versions of VLC on Windows (although the process is different for earlier releases of VLC). It is possible to stream video files from your computer, a live webcam or hand held camera. If you have a laptop with a camera built-in then you can also send live streams using this.

If you are using Ubuntu it is better to upgrade your version of Ubuntu to Intrepid (8.10) or Jaunty (9.04) as they both come with a version of VLC which is easy to stream with. If you don't wish to upgrade the Operating System then you need to install the latest version of VLC from source files - be warned, upgrading VLC from sources is complex.

## Streaming Server Requirements

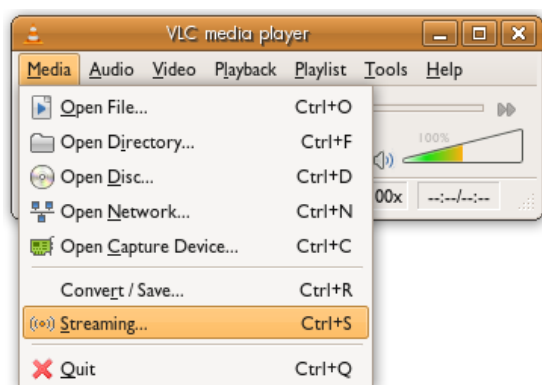
We will examine streaming to an **Icecast Server**. You need access either to an Icecast Server or to someone who can create an account for you and provide the access details. You need to know the following information about the Icecast Server:

- **Hostname** or **IP Number** of the **Icecast Server**
- **Port** (Default is **8000**)
- **Password**
- **Username** (This may not be necessary depending on how **Icecast** is configured.)
- **Mountpoint** (This may not be necessary depending on how **Icecast** is configured.)

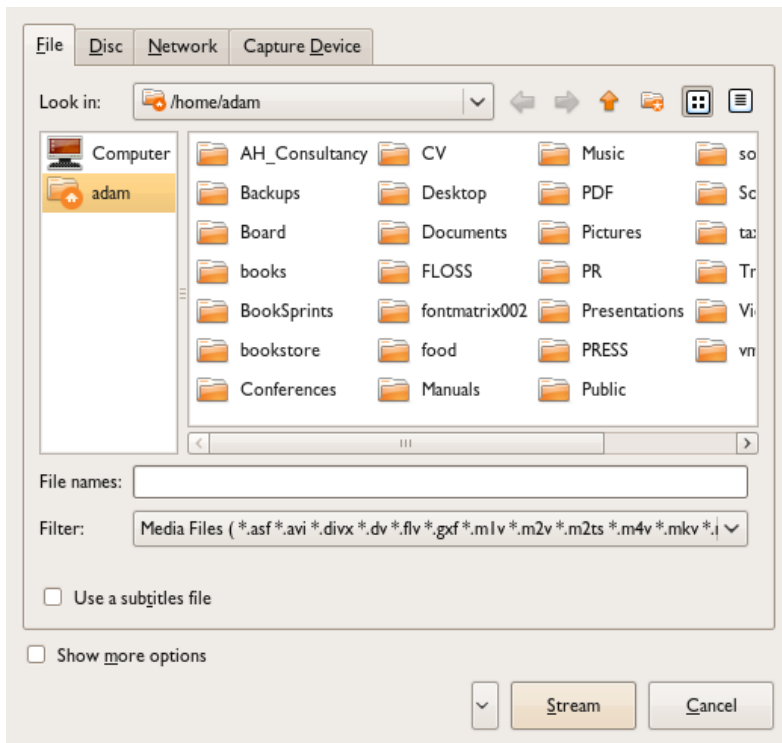
## Choosing Video File to Stream

Lets look at streaming using the camera built into a laptop. The process is very similar for streaming from an external camera or archived video files.

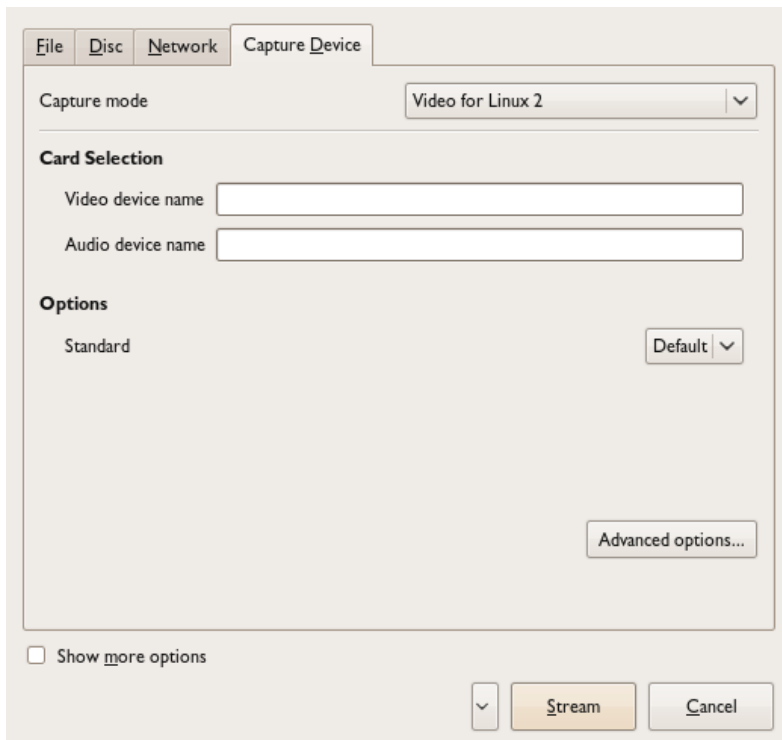
The first thing you need to do is open **VLC**. Now from the **Media** menu choose **Streaming...** :



This will open a panel where you can choose the source for the video stream.



We will choose the last tab - **Capture Device**, however here you could also choose any of the other tabs to stream video from a disc (a DVD for example), archived files, or relay an incoming video stream. The capture device window looks like this :



If all works well you might not need to change any settings in this section at all. Instead you just click on the button titled **Stream**.

# Stream Settings

Under the Outputs section of the stream settings there is a section for configuring Icecast.



The screenshot shows a configuration form for Icecast. It has a checkbox labeled 'IceCast' which is currently unchecked. To the right of the checkbox are four input fields: 'Address' (empty), 'Port' (set to 1024), 'Mount Point' (empty), and 'Login:pass:' (empty).

You just need to check the box here and then fill in the details. There are a few things to remember here :

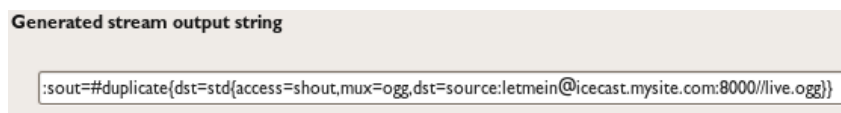
1. The Mountpoint should always start with a '/'.  
2. The address does not start with 'http://'.  
3. Most often you do not get a username for Icecast. In this case use the username 'source'.  
4. The port is almost always 8000.  
5. The format for the **Login:pass** field is exactly as the title suggests - the login followed by a ':' and then the pass. All with no spaces. For example, a stream to a default install of Icecast2 (the default password is always 'letmein') would be :

```
source:letmein
```



The screenshot shows the same configuration form as above, but now the 'IceCast' checkbox is checked. The 'Address' field contains 'icecast.mysite.com', the 'Port' field is set to 8000, the 'Mount Point' field contains '/live.ogg', and the 'Login:pass:' field contains 'source:letmein'.

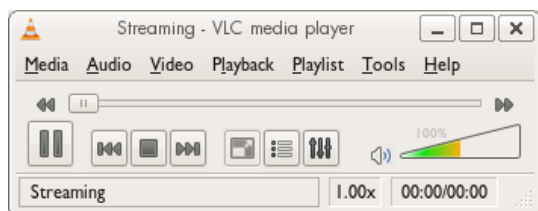
As you enter the information you will see the text change in the 'Generated stream output field' at the bottom of the window.



The screenshot shows a text box titled 'Generated stream output string'. The text inside the box is: `:sout=#duplicate{dst=std{access=shout,mux=ogg,dst=source:letmein@icecast.mysite.com:8000/live.ogg}}`

If you were experienced you might be able to directly type this information in without using the fields provided, but it's far easier to just to fill in the boxes as required.

Now all you have to do is press 'Stream' and the live stream will be sent to the Icecast2 server. However, VLC does not provide good error feedback, so it might sit there looking like it's happily streaming whereas in fact nothing is happening. VLC might even display 'Streaming' in the status bar, but this means nothing!



The only way to know it is actually working is to look at the status page on your Icecast server, or use a video player to playback the stream live from the server.



# Saving a Theora Stream

If you are running GNU/Linux it is easy to rip an Ogg Theora stream. This is because Theora uses HTTP to stream, hence you can use the **wget** software, which is usually included in linux distributions. You direct **wget** to capture the stream like so:

```
wget http://streamingserver:8000/streamname
```

For example, if the server was **icecast.streaminguitcase.com** and the stream name was **theora.ogg**, you would use the following command for archiving:

```
wget http://icecast.streaminguitcase.com:8000/theora.ogg
```

This is good but if **wget** buffers, it will quit the archiving. However you can try this:

```
while true; do wget http://icecast.streaminguitcase.com:8000/theora.ogg;done
```

This can be run as a command on one line of a shell, or you can make it into a handy little script.

Another bonus from using **wget** to archive is that it will increment the file names. So if you archive the stream in the example above the file would be saved as **theora.ogg** if you stopped **wget** and started it again the next file would be called **theora.ogg.1** etc. This means you don't overwrite the existing archive file.

## Extracting parts of a video

For non live videos, it is sometimes possible to extract parts of the video as well. This only works if the hosting provider has **oggz-chop** installed to allow server-side seeking. If it is installed you can specify an 'in' and 'out' point. So if you just want to download the video from second 23 to second 42, you add **?t=23/42** at the end of the URL:

```
wget http://example.org/theora.ogv?t=23.0/42.0
```

With Firefox 3.5 you can also open the URL in the browser and save it with right click Save Video As...

# Introduction

**Ogg Theora** is not only an excellent distribution format, but it can also be edited. If you have a GNU/Linux machine with enough memory and computational power, you can explore editing software like **LiVES** (<http://lives.sourceforge.net>), **PiTiVi** (<http://www.pitivi.org/>), **Kdenlive** (<http://www.kdenlive.org/>), and **Cinelerra** (<http://cinelerra.org/>).

At this point in time, for simple editing of Theora files, PiTivi and LiVES are the best options. Kdenlive often crashes when exporting to Theora (and the output file is incorrectly formatted), and Cinelerra is difficult both to install and to use.

If you use Ubuntu then PiTiVi is an even better choice since installation is very simple. In addition PiTiVi is built on the **GStreamer** multimedia framework (used by Ubuntu), so you can import and edit pretty much any media file supported by GStreamer (which is a lot), and export to Theora.

**LiVES** and **PiTiVi** work like most editing software: they decompress the video files in the process of importing the video data into the editing software, so you can view and browse through all the images and sounds, rearrange them and do all kinds of cool stuff. When you finish, you re-compress the new edit by exporting the resulting arrangement. This process of de- and re-compressing comes at a cost, as you lose a little quality with each de- or recompressing process that you run. That's why you call this editing process **lossy editing**.

# PiTiVi

**PiTiVi** is a video editor for GNU/Linux that supports importing and exporting Ogg Theora videos. Right now it supports basic editing, cropping clips, arranging on a timeline and adjusting of audio levels.

PiTiVi has its own documentation that you might also wish to read located online at <http://www.pitivi.org/wiki/Manual#Proposed>

There is also a very good manual available in PDF format that you can find online at <http://jeff.ecchi.ca/blog/?p=897>

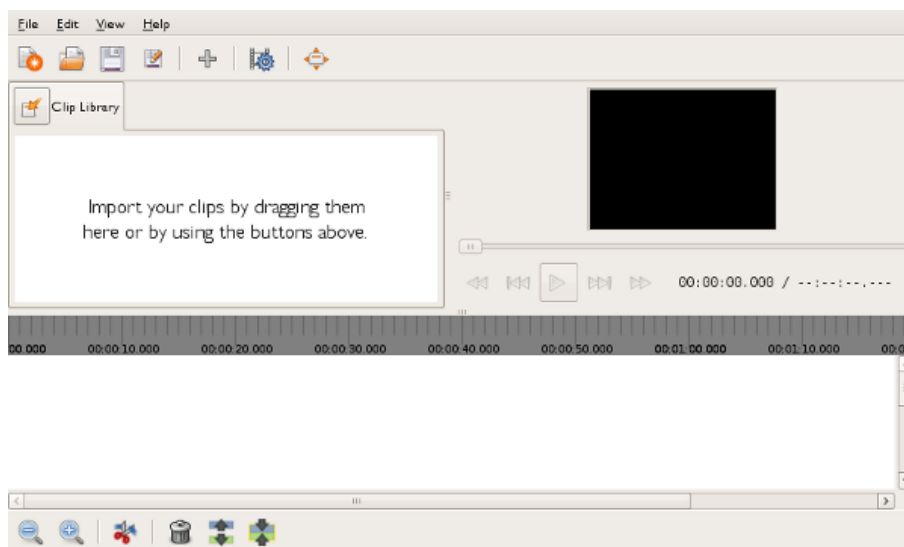
## Installing

PiTiVi only works with GNU/Linux. If you have Ubuntu Jaunty (9.04) you can install a very useful version of PiTiVi from the command line (terminal) with the command :

```
sudo apt-get install pitivi
```

You will be asked for your password, when you enter it the installation will proceed automatically.

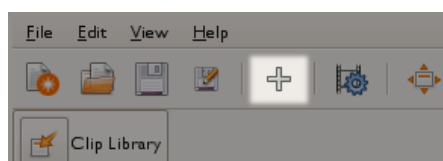
When the installation is complete you can open PiTiVi from the Applications menu (listed under **Sound & Video**).



You can also check <http://pitivi.org/wiki/Downloads> for instructions to install newer versions. PiTiVi is actively developed right now and there are new releases coming out regularly.

## Importing Video

Once you opened PiTiVi you can press the plus (+) button in the interface to Import clips.



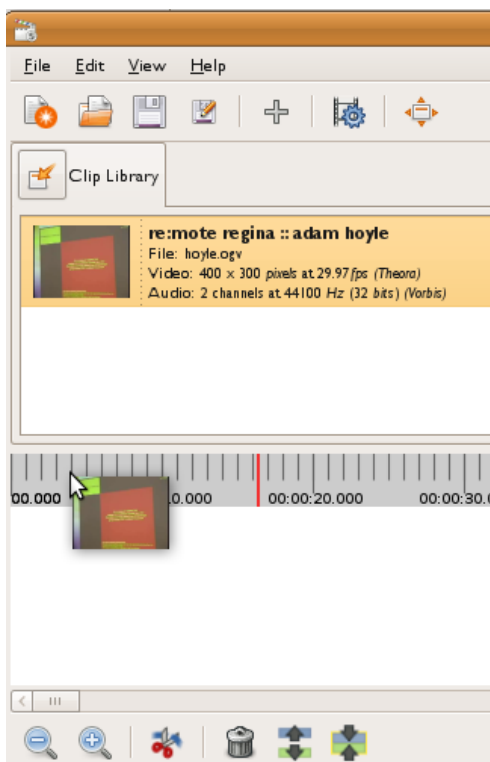
Imported clips are added to your **Clip Library**. You can use all formats that **GStreamer** accepts. If you don't know what this means then the easiest way to know what files work with PiTiVi is to play them in Totem (found in the Sound and Video Applications menu and titled 'Movie Player'). If a file can play in Movie Player then it can also be used as a clip in PiTiVi.

You can continue adding as many clips as you like to the Clip Library (the video of the Digital Clock is created by Riccardo Iaconelli).

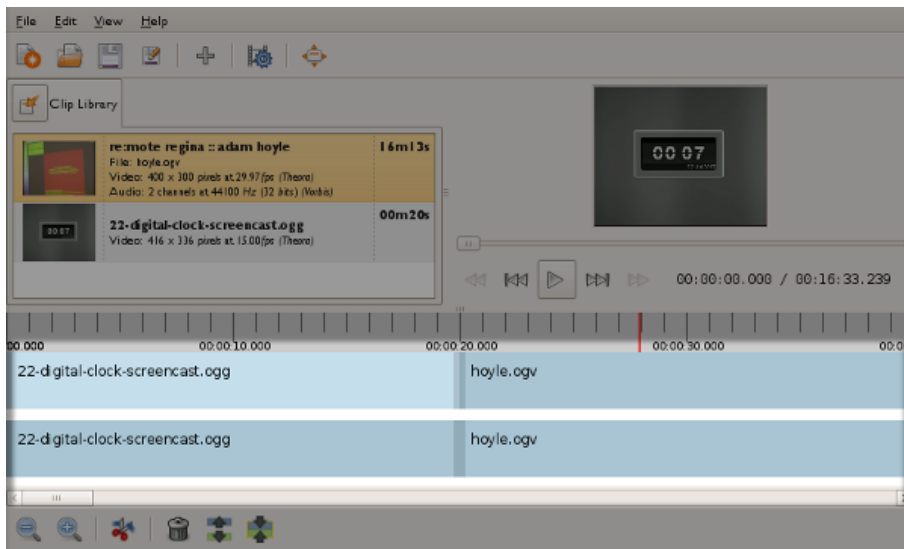


## Adding Clips

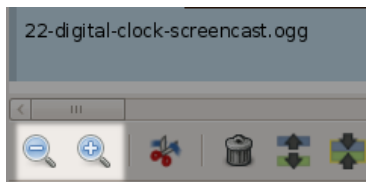
Once you imported the clips you want to use, you can drag them from the Clip Library on the timeline below.



If you add multiple clips to the time line they will appear as slightly different colors and with the filenames displayed :

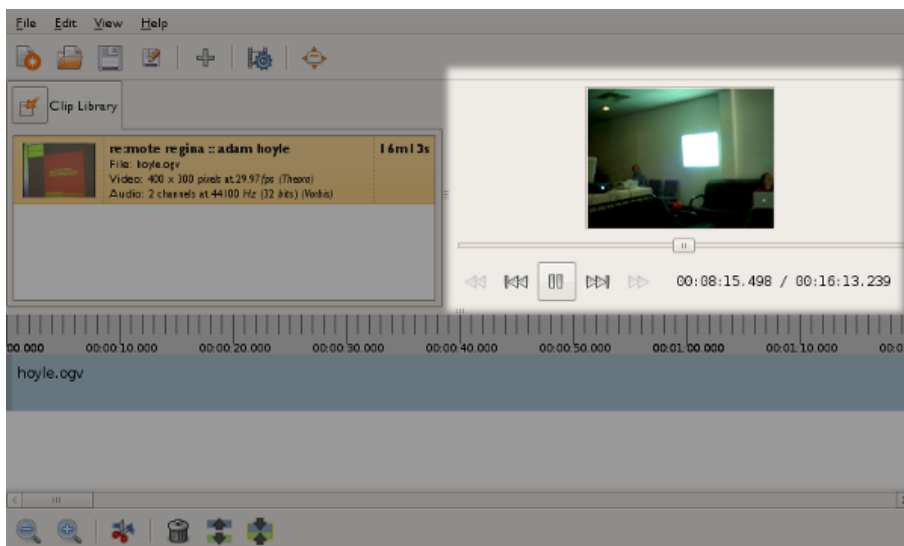


If you can't see this kind of detail then use the zoom tools to zoom in and out of the time line :

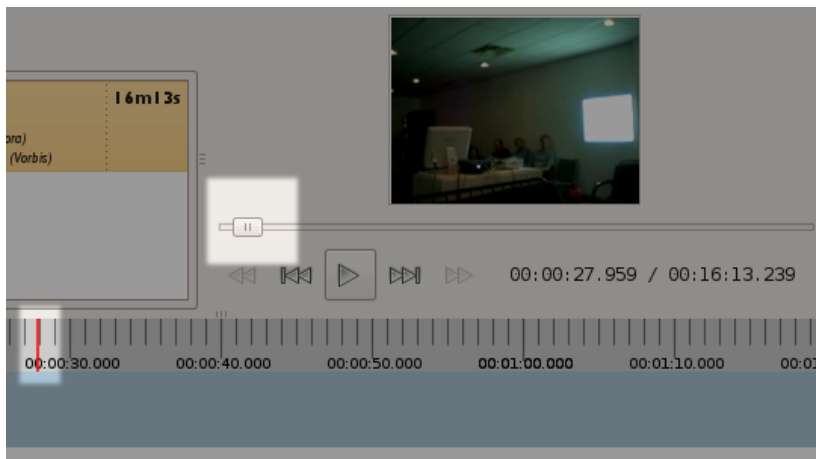


## Playback

When the clip is on the timeline you do not see the individual frames, but you can drag the slider backwards and forwards in the player area to see the video.



When you play back video like this you can see the position (time) of the playback displayed by a red tracker on the time line :

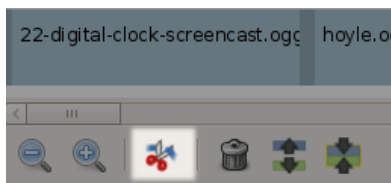


## Moving Clips

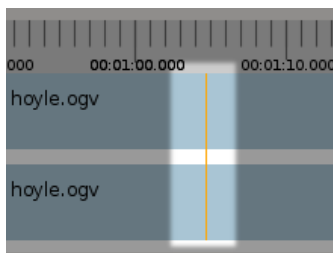
You can move the clips around by dragging them around the time line.

## Cutting Clips

To cut a clip in two you need to click on the scissors icon :



Then move the cursor around the time line. You will see an orange line appear showing the potential position of the cut :



When you have the orange line at the right place click on the timeline and the selected video file will be cut in two at that point.

## Deleting Clips

To delete a clip, click on it with the cursor so that it is selected and then press the Trash Can icon :

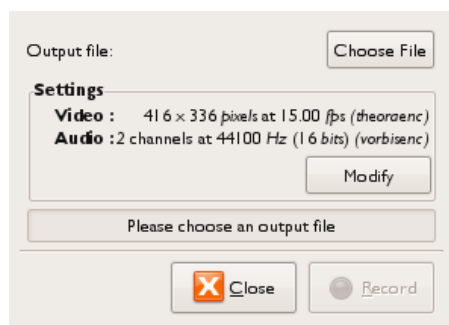


## Export to Ogg Theora

Once your project is ready to be exported, you can click the **Render Project** button :



You will now see the Render settings window :



Click **Choose File** to and choose where to save the exported file to. You could also adjust the encoding settings pressing **Modify**, once the settings fit your needs, press Render and get a coffee.

# Introduction

If you want to play around with your Ogg files you do not need to launch a full-blown editing software. A number of useful tools exist that help you perform simple tasks with Ogg files on the command-line. The **Ogg Video Tools** is a toolbox for Theora video files, that includes a number of handy command line tools for creating an manipulating Theora files. Use them for cutting and splicing or to create thumbnails, preview or slideshow videos without having to engage into a complex editing process.

## Installing the Ogg Video Tools

Many GNU/Linux distributions provide the Ogg Video Tools within their repositories (e.g. Fedora, openSuSE). If the tools are not available for your preferred distribution, or you want to upgrade to the most recent version, continue reading the *Installing from Source* section below. This also applies to BSD and Mac OS users.

Windows users can download a ZIP file, containing the executable command line tools from <http://sourceforge.net/projects/oggvideotools/files/>.

## Installing from Source

Visit <http://sourceforge.net/projects/oggvideotools> and click on *Download Now* to get a copy of the source code.

Before continuing installation, make sure that you have all required development packages installed. The Ogg Video Tools depend on the following other packages:

- theora and theora-dev
- vorbis and vorbis-dev
- ogg and ogg-dev
- GD-lib with its development parts
- SDL and SDL-dev (in case you want to compile oggScroll)

The Ogg Video Tools will successfully compile even with all of these libraries missing, but in that case only a subset (the editing tools) are going to be compiled. The other tools will not be available at all after installation.

After making sure that the required packages are present, open a terminal and enter the following command (without the initial dollar sign):

```
$ tar xzvf oggvideotools*.tar.gz
$ cd oggvideotools*
$ ./configure
$ make
$ su
<type root password>
$ make install
```

Where the *./configure* command on the third line should output something similar to:

```
[ ... ]
checking for bzero... yes
checking for memset... yes
configure: creating ./config.status
```



Creating the following tools:

oggJoin  
oggSplit  
oggCut  
oggCat  
oggDump

oggSlideshow  
oggThumb  
oggResize  
oggSilence

oggScroll

# Create Thumbnails

The Ogg Video Tools (<http://sourceforge.net/projects/oggvideotools>) come with a tool called `oggThumb` that extracts thumbnails from Theora video files.

Example command:

```
$ oggThumb -t1,2,3,4 -s320x240 my_theora_video.ogv
```

Here we create four thumbnails from a video file called "my\_theora\_video.ogv", taken at 1, 2, 3 and 4 seconds into the video. In this example the thumbnail images are stored with the following file names:

```
my_theora_video_0.jpg
my_theora_video_1.jpg
my_theora_video_2.jpg
my_theora_video_3.jpg
```

## Creating a Series of Thumbnails

If you need to extract thumbnails at fixed intervals throughout the full video, without knowing the length of the video in advance, use the following shell script to do the job:

```
#!/bin/sh
length=$(oggLength "$1")
timestring="0"
div=$((($2*1000))
for((i=1; i<$2; i++)) do
    value=$(echo "scale=3; $i*$length/$div" | bc -q 2>/dev/null)
    timestring="$timestring,$value"
done
oggThumb "$3" -t$timestring "$1"
```

The script is then called as:

```
$ ./mkThumb my_theora_video.ogv 15
```

This script invocation extracts 15 pictures from start to end of the video file, at equidistant intervals. The pictures are output as files, using the same naming convention as previously described. Replace *my\_theora\_video.ogv* with the name of the video you want to work on. The number of thumbnails to generate is given by the last argument. Change it to suit your needs.

## Creating Thumbnails with Fixed Height or Fixed Width

You can request thumbnails of a certain size by adding option `-s` to the command line:

```
$ oggThumb -t1,2,3,4 -s320x240 my_theora_video.ogv
```

Here we generate thumbnails sized 320x240. You can omit one either width or height, by setting it to 0:

```
$ oggThumb -t1,2,3,4 -s120x0 my_theora_video.ogv
```

Here the height of the video is set to match a width of 120. `oggThumb` computes the height so as to preserve the aspect ratio of the video frames.

## Advanced Functionality

Use option *-f* instead of *-t* to select the thumbnail time positions by frame number instead of a number of seconds. Another option worth mentioning is *-o png* which generates thumbnails as PNG images, which have a higher quality than the JPG files generated by default.

The full documentation of the oggThumb command line utility can be accessed by typing:

```
$ man oggThumb
```

on the command line. Note that the windows version comes with a set of HTML files instead, containing a manual that can be read in your web browser.

# Creating a Slideshow

The Ogg Video Tools provide a command line tool called **oggSlideshow** to create slideshows from a series of pictures in JPEG or PNG format with different effects for the changeover and presentation.

Synopsis: **oggSlideshow** [ Options ] **picture1 picture2 picture3**

Each picture is read in from the file system and resized as necessary to match the video frame size, which can be given by the **-s** option. If a picture's aspect ratio does not match the video frame, black borders are added.

The following effects for the picture changeover are supported:

**plain:** Pictures are presented one after another.

**crossfade:** The first image fades out while the next one gradually appears. The crossfade duration is 1 second.

**blurring:** The picture is blurred before it disappears. The next picture appears blurred, then comes into focus.

Another special effect is the **Ken Burns** effect, named after the famous documentary film maker. This effect slides and zooms through the picture during presentation. The changeover in this effect is set to crossfade. â â

There exist options to set bit-rate, the length of the per-picture presentation time, output file name and more, summarized below:

## Command Line Options

**-s**

Set the size of the video frame. The size is given as `<width>x<height>`. The default size is 480x320.

*Example: -s 320x240*

**-f**

Set the frame rate of the video, given as the number of frames per second. The default frame rate is 24 frames per second.

*Example: -f 16*

**-o**

Set the output file name of the created video. The default name is `slideshow.ogv`.

*Example: -o myShow.ogv*

**-l**

Set the presentation time (length) per picture. If you have 10 pictures and specify a length of 10 seconds, the overall video length will be 1:40 (i.e. 100 seconds).

*Example: -l 10*

**-d**

Set the datarate measured used for the video, measured as bit per seconds. This is intended more as an upper threshold. The resulting Theora file may be smaller than expected.

*Example: -d 1024000*

**-t**

Set the picture changeover effect type, as described above. Supported types are:

- ◇ kb: Ken Burns effect (default)
- ◇ cf: picture crossfade for changeover
- ◇ p: plain picture presentation
- ◇ b: blurring as changeover

*Example: -t p*

**-e**

Enable reframing. In case a picture does not match the aspect ratio of the video frame, it can be "reframed" by adding black borders. This option is only useful with the Ken Burns effect (option *-t kb*). With the other changeover effects, pictures are automatically reframed even without option *-e*.

*Example: -t kb -e*

## Adding Sound to a Slideshow

Using oggSlideshow you can create silent video files only. But adding audio afterwards is merely a matter of using a few more tools.

Let us assume you created a slideshow using all images from your directory *somePics*, as shown in the following example command line:

```
$ oggSlideshow -s 320x240 -f 24 -o slideshow.ogv -d 512000 somePics/*.jpg
```

Before adding audio, we first need to create an audio track that has the same length as the slide show. The slideshow duration is printed at the end of the slideshow creation process. Alternatively it can be read out using the following command:

```
$ oggLength slideshow.ogv
```

This returns the length of the video in milliseconds. Using that information we can cut an Ogg/Vorbis audio stream to the same length. 5.6 seconds in this example::

```
$ oggCut -l 5600 myAudioTrack.ogg myAudioTrackAdjusted.ogg
```

To multiplex both streams to one media file (*newSlideshow.ogm*), we use oggJoin:

```
$ oggJoin newSlideshow.ogv myAudioTrackAdjusted.ogg slideshow.ogv
```

Now you are done!

## Adding a Fixed Starting Picture

The first picture in a slideshow often needs to be a title picture giving some details about the slideshow. Create a short and plain slideshow to use as title sequence, using only one picture:

```
$ oggSlideshow -s 320x240 -f 24 -tp -o startPicture.ogv -l 10 -d 512000 \ startPicture.jpg
```

Here the slideshow type is set to *plain* (*-tp*), making the picture just appear as it is. The duration is set to 10 seconds (in contrast to the default setting of 8 seconds).

We prepend the title picture to our *slideshow.ogv* using *oggCat*:

```
$ oggCat overallSlideshow.ogv startPicture.ogv slideshow.ogv
```

Of course we can also add an audio track as described above:

```
oggJoin newSlideshow.ogv myAudioTrackAdjusted.ogg overallSlideshow.ogv
```

## A Script for Everything

The following shell script creates combines everything described in this chapter, creating a slideshow with title sequence and audio.

```
#!/bin/sh
#
# usage: ./mkSlideshow ~/mypicDir/ audiofile.oga outputFile.ogm
#
# Variables to be changed
#
# video frame size
SIZE="640x360"
#
# data rate of the outgoing slideshow stream in bit/s
DATARATE="1024000"
#
# presentation time of one picture in seconds
PR_TIME="10"
#
# frame rate in pictures/s
FRAMERATE="24"
#
# reframe picture
# This adds black borders to picture to meet the aspect ratio
# of the video frame size specified earlier.
# With the Ken Burns effect, this is not strictly necessary,
# but the sliding may be smoother
#REFRAME="-e"
REFRAME=""
#
# resample
# This option says, how the picture should be loaded (by gdlib)
# As the resize mechanism of gdlib is really good, it is used to
# bring it do a value "near" the video frame size (usually a bit
# bigger). You usually do not see a big difference, if you change
# this value :-), so keep it as it is (default = 1.2)
RESAMPLE="1.2"
```

```

#
# slideshow type
# kb - Ken Burns Effect (sliding and zooming)
# p - plain (picture display only, no crossfade between pictures)
# cf - crossfade (picture display, crossfading between pictures)
TYPE="kb"
#
#
# Temporal file name
TMP_VIDEOFILE="slideshow_tmp.ogv"
TMP_AUDIOFILE="audio_tmp.oga"

# creating the slideshow
oggSlideshow -s $SIZE -d $DATARATE -l $PR_TIME -f $FRAMERATE \
  $REFRAME -r $RESAMPLE -t $TYPE -o $TMP_VIDEOFILE $1/*.jpg

# what is the length of this
LENGTHVIDEO=`oggLength $TMP_VIDEOFILE`

#
# cut the audio file
LENGTHAUDIO=`oggLength $2`

#
# is the audio file to short?
if [ $LENGTHVIDEO -gt $LENGTHAUDIO ]
then
  echo "warning slideshow ($LENGTHVIDEO) is longer than your audio file ($LENGTHAUDIO)"
  exit -1
fi

# cutting the audiofile
oggCut -l$LENGTHVIDEO -i$2 -o$TMP_AUDIOFILE

#
# Join audio and video file
oggJoin $3 $TMP_VIDEOFILE $TMP_AUDIOFILE

#
# remove old files
rm -f $TMP_VIDEOFILE $TMP_AUDIOFILE

```

# Creating a Video Preview

Hosting online video, it is often preferable to embed a short preview clip of a film into the main webpage, which is linked to the original video. Use the command line tool **oggResize** from the Ogg Video Tools (<http://sourceforge.net/projects/oggvideotools>) to create such a preview.

A typical command line calling `oggResize` looks as follows:

```
$ oggResize -s256x144 -d64000 -f1 -p24 bigBuckBunny.ogv bigbuckbunnyPreview.ogv
```

Here we create the Theora video file *bigbuckbunnyPreview.ogv* from the original file *bigBuckBunny.ogv* scaled to a small frame size of 256x144 pixels.

The `-p` option makes `oggResize` include only every 24th picture from the original, reducing duration to 1/24 of the original length, usually creating a fast-forward version of the video. Since it does not make much sense to fast-forward the audio stream of the video, audio is removed.

But here we also added option `-f1` thereby adjusting the frame rate of the created video to one frame per second. This prevents the video from being fast-forwarded, and is much easier to watch, looking similar to an animated GIF image.

## Adding a Play Button

First we need a PNG picture of the button, with a transparency mask that allows us to overlay it over our video content. The transparency mask is called an alpha channel, nowadays most image editing programs allow you to create one. Try for example the free software image editor *GIMP*.

With the `-A` and `-a` options you overlay a picture over the output video before and after the video is rescaled respectively, to the size given by `-s`.

```
$ oggResize -s256x144 -d64000 -f1 -p24 -A Play-256x144.png bigBuckBunny.ogv \ bigbuckbunnyPreview
```

So that's it! Now you got a nice preview of the *bigBuckBunny.ogv* video, with a play button on top inviting users to click on.



# Introduction

The command line is a text-based method of calling utility programs and others in a shell command interpreter session in a terminal program. Most GNU/Linux systems have a program called Terminal on the main application menu. In most cases, it offers the user the Bourne-Again Shell, 'bash', a successor to the original Bourne shell, 'sh', but there are several others in use.

Every user should know how to use simple text commands such as 'df -h', which says to get a 'disk fullness' report in 'human'-readable units (MB rather than disk sectors). The basic features and many of the intermediate concepts of the command line are documented in the Floss Manuals book *Introduction to the Gnu/Linux Command Line*, available at no charge at <http://en.flossmanuals.net/gnulinux>. Readers of that book who try out the examples given will be well prepared for anything described in this chapter. That is, they will have passed the beginner stage, and be ready to enter the threshold of the advanced user stage. Advanced command line use in GNU/Linux basically means two things: using the full range of options available in a command (as listed by the man command--try 'man man'), and writing scripts of more than a few lines using more than just simple commands.

Many video editing tools come in both a command line version and a full-screen text or GUI version that accepts a set of selections from the user, and composes a command line to execute. For example, installing packages from the command line in Debian, Ubuntu, and related GNU/Linux distributions uses the apt-get utility. (In Red Hat, use yum.) The full-screen text package installer in these systems is aptitude, and the GUI installer is synaptic. A well-written GUI tool can make it much easier to create, save, and reuse command options than selecting commands in text, or can help in navigating large file systems or databases. On the other hand, many GUI tools do not give control of all command options, so that more advanced use requires the command line, or alternatively saving complex commands in text files, and making them into executable script files with the chmod command.

## Lossless Video Editing

Video editing is a very expensive venture in terms of your computer's computational power and memory consumption. This is because complex video editing tools must enable you to work with every frame of a video and every sample of an audio stream.

However, handling every single frame is not always necessary. In some cases, people only want to extract the video or the audio stream from a file, or merge ('multiplex') a file from video and audio streams, or they don't care about frame-accurate video cutting.

In these cases the video and audio streams don't need to be re-encoded and this makes the handling of video less costly in terms of your computer's resources. It also means the editing tools are very fast.

This editing process is called **lossless**, as there is no re-encoding involved. Re-encoding video always causes a loss in quality and editing video without re-encoding is therefore lossless. The encoded packets (chunks of data), either video or audio, are not touched. They are kept as they are, so they are decoded with the same quality they had before the editing process.

## Tools

There are two main tool boxes for lossless video editing: The **Ogg Video Tools** and the **Oggz Tools**. They are both collections of command line tools. Ogg Video Tools and Oggz Tools works on GNU/Linux, OSX, and Windows.

Oggz can be found here: <http://www.xiph.org/oggz/>

Ogg Tools can be found here : <http://sourceforge.net/projects/oggvideotools>

In some cases the two tool sets overlap in their functionality. Oggz Tools is a bit more focused on developers, while the Ogg Video Tools is more focused on end users. Ogg Video Tools provides a lot more tools for creating and handling ogg video files, while Oggz Tools provides a number of other tools, especially for analysing ogg files.

Ogg Video Tools can handle video streams created by the Theora encoder and audio streams created by the Vorbis encoder. Oggz can handle additional stream formats, like FLAC, Speex, CELT etc.

**Ogg Video Tools** command line tools include:

- **oggSplit**: demultiplexes a multiplexed ogg file into several files with one stream in each.
- **oggJoin**: multiplexes Ogg audio (vorbis) and Ogg video (theora) files into a single Ogg file.
- **oggCut**: creates a new Ogg file as a subpart of an original input Ogg file.
- **oggCat**: concatenates two or more Ogg files.
- **oggResize**: changes Theora and/or Vorbis streams in multiple ways.
- **oggDump**: prints a dump of packets or pages within a given Ogg file.
- **oggLength**: returns the length of a given Ogg file.
- **oggSilence**: creates a vorbis audio file of a given length and filled with silence.

**Oggz Tools** command line tools include:

- **oggz-chop**: extracts part of an Ogg file between given start and/or end times.
- **oggz-merge**: merges ogg files together, interleaving pages in order of presentation time.
- **oggz-sort**: sorts the pages of an Ogg file in order of presentation time.
- **oggz-rip**: extracts one or more logical bitstreams from an Ogg file.
- **oggz-info**: displays info on the codecs used by an Ogg file.
- **oggz-comment**: lists or edits comments within the Ogg stream headers.
- **oggz-validate**: checks an Ogg file for common problems.
- **oggz-dump**: prints a dump of the packet streams within a given Ogg file.

# A Bit of Theory

Editing Theora videos using command line tools like Oggz Tools and Ogg Video Tools does not require knowledge about the intricate details of how Theora works. However, as these command line tools process Theora files at a very low level, some side effects are going to show up that can not be explained without going into some of the details of the Theora format.

If you think you can live with minor inaccuracies caused by editing, feel free to skip reading this chapter.

## Anatomy of a Theora Video

A video file normally consists of a video stream and an audio stream. To store both streams into one file, a so-called container format is used. The container used with Theora video is the Ogg container, holding a Theora video stream and one or more Vorbis audio streams. Audio and video streams are stored interleaved. That means that every stream is segmented into several blocks of data with nearly equal size. These blocks are called *pages*.

Every page has a time stamp that gives information about where the page is placed within the stream.

Video and the audio streams are interleaved by concatenating the pages of both streams in ascending order of their timestamps.

## Demultiplexing

Technically, splitting a video file into its video and audio streams is easy. This is due to the fact that the two streams can be separated by collecting the video and audio pages into different files. The process of splitting a video file into its streams is called *demultiplexing*.

As all necessary information required for playback is contained within the streams, without the Ogg container adding any further information, each of the split files itself constitutes a standard compliant Ogg contained stream and can be read by any Theora/Vorbis aware video or audio player.

## Page Timestamps

As mentioned earlier, streams are stored segmented into several pages. A page has a *header that holds so-called meta data* describing this part of the video/audio stream. The header includes timing information, a unique stream identification number, a page number and some other information.

According to the Ogg standard, timing information for every page is given by the *granule position*, a 64 bit value contained in the header. How granule position relates to an actual time position like the number of milliseconds from the start of the video is entirely defined by the stream, and not covered by the Ogg container specification. For that reason a software handling Ogg files needs a granule position interpreter to correctly handle the file. This interpreter needs to be aware of the codec and stream specific information to produce timing information that can be compared across different streams.

While not defining how to interpret granule positions, the Ogg Standard specifies that all pages within an Ogg file must be stored in ascending order of the corresponding time position. So any tool that cuts or concatenates streams needs working granule position interpreters for every contained stream in order to correctly interleave the pages.

## Encapsulating Codec Data

Pages are of nearly equal size by default (around 4096 Bytes). However, audio and video packets created by a specific codec, usually do not fit exactly into a page. Audio packets are usually much smaller. Video packets can be of a very different sizes, smaller or larger, depending on various factors.

Data that is produced by the video and audio codecs are firstly encapsulated into an *Ogg packet*. These packets are then placed into the *Ogg pages*. A packet can either be split over multiple pages, or combined with other packets to fill up an otherwise underfull page, as required.

## Encapsulating Theora Video Data

Theora video data created by the encoder consists of two types of *Ogg packets*: so called key frames (often also called I-Frames), which are full pictures, and P-Frames, which carry only the differences between the last and the current picture.

In order to display a given frame at a given time, the decoder must know the previous key frame and must decode every frame (including the key frame) up to the actual time position of the given frame.

To be able to decode a video at all, the decoder needs information about the stream itself, such as the video frame size. This information is placed into the header packets at the beginning of the stream.

## Encapsulating Vorbis Audio Data

Vorbis audio data created by the encoder carries a certain number of audio *samples*. A sample is a unit of audio data. The number of samples per Ogg packet is fixed and can only vary between two possible sizes, defined in the stream header.

Similar to video data, audio data packets depend on each other. To decode one audio packet, the previous packet is needed .

The audio decoder, depends on the stream parameters such as sample rate, bitrate etc, which are stored in stream header packets at the stream's beginning.

## Ogg Skeleton

As was stated before, a video stream can not start at arbitrary time position due to its nature of having key frames (I-Frames) and delta frames (P-Frames). In addition audio is stored in packets, which have a given timing granularity.

To ensure synchronization and a correct starting point, *Ogg Skeleton* pages carry information about the starting position of a each stream. A decoder, that reads the Ogg Skeleton information, can then seek to the correct audio and video position and start playing from this position.

# Splitting a Video File

Splitting a video file into its constituent streams is often called *demultiplexing*. The command line tool `oggSplit` from the Ogg Video Tools performs demultiplexing.

Synopsis: **`oggSplit originalFile.ogv`**

This command extracts all streams in *originalFile.ogv* into separate files. It uses the following naming scheme for the created files:

**`<codecName>_<ID>.<extension>`**

The *codecName* is set to *theora*, *vorbis* or *unknown*, depending on the codec type of the stream in question.

Every stream has a specific *Identification Number (ID)*, set by the encoder. This ID is a 32 Bit number that is unique for every stream within a single Ogg file. As there can be more than one audio or video stream within a file (e.g. different audio streams for different languages), this ID is needed to distinguish among them.

The *extension* is set to match the contents of the stream. *ogv* is used for video files, *oga* for audio files and unknown streams get the extension *ogg*.

*Example:*

```
# oggSplit myFile.ogv
# ls
theora_a1bb23c1.ogv      vorbis_a316522.oga
```

# Cutting an Ogg File

Cutting an Ogg file consisting of video and audio streams into sections by time, while preserving synchronization between streams in the output files, is a bit tricky. Because there are several ways to do so, this section gives you an idea of the possibilities and their advantages and drawbacks.

Tools to edit video files can be divided between lossless and lossy tools. Lossless tools only handle the ogg stream (ogg pages and ogg packets). So the quality of the data is preserved. Lossy tools must first decode the data to allow editing and then re-encode, thus losing quality. This type of editing provides greater flexibility (e.g. it allows overlaying of images) at the expense of quality. Beyond that, the decoding and encoding process requires much more time than just handling Ogg stream packets.

This chapter only deals with tools that losslessly cut video files.

## Using oggCut

The command line tool oggCut is a tool that can cut Ogg files on a keyframe basis. Use it like:

```
oggCut -s <StartTime> -e <EndTime> originalFile.ogv createdFile.ogv
```

where *<StartTime>* and *<EndTime>* are given in milliseconds. Alternatively you can cut using a start time and length using option *-s*:

```
oggCut -s <StartTime> -l <Length> originalFile.ogv createdFile.ogv
```

Here, too *<Length>* is measured in milliseconds.

When oggCut is executed, it searches for the first Ogg Theora packet with a timestamp greater than or equal to the time given by the *-s* option. If this packet is found, the program seeks to the next keyframe and creates the new file with that keyframe as the first frame.

When the program finds a packet with a timestamp greater than the end time, it closes the newly created file and stops.

As the start time is likely not the position of a keyframe, the created file may be smaller than expected.

## Video/Audio synchronization issues with oggCut

Because oggCut only operates only on packet boundaries, and because Vorbis and Theora stream packets are not synchronous, the audio and video packets usually start at different time positions.

Therefore when oggCut creates a new file that starts at time *t*, the next audio packet will likely start at a position *t+x*, where *x* is the time offset between the keyframe video and the next audio packet.

When these two streams are written to the new file, this time offset information is lost. Therefore the player must assume that both files start at time 0. For that reason, the audio stream always plays a bit too early. However, as an audio block is quite short, the difference is usually only a few milliseconds and is hardly noticeable.

## Using oggz-chop

The program oggz-chop from oggz-tools cuts Theora videos at any position requested. If necessary it cuts at non-keyframes or even in the middle of a frame's presentation time.

Technically this kind of fine-granularity cutting is not possible with Theora video technology. It is achieved by a trick: instead of really cutting the video at the requested position, it is cut at the closest point, making the resulting video file longer than requested. Then a so-called Ogg Skeleton header is added to the video that instructs the player to only play the actually requested range of the video.

The superfluous video frames that could not be cut from the video are still present, they will just be hidden on playback.

The disadvantages of this approach are obvious: on the one hand, disk space is wasted by storing unnecessary amounts of data, on the other hand, the newly cut file will only play back correctly in video players that understand the Ogg Skeleton information. Another not so obvious disadvantage is that files that were created this way can not be concatenated without the hidden part reappearing. This is due to the fact that the Ogg Skeleton can only hide frames at the start and end of a video. Concatenating two videos of this kind, will leave hidden frames in the middle -- something Ogg Skeleton does not support.

Now that you understand the implications of using oggz-chop, let's look at how it is used:

```
oggz-chop -s <StartTime> -e <EndTime> originalFile.ogv createdFile.ogv
```

Here *<StartTime>* and *<EndTime>* are specified as *<Hour>:<Minute>:<Second>*. *<Second>* can contain a fractional part so you are free to specify time at any resolution.

We should mention another option: you can run oggz-chop with *-k* to force it to omit any Ogg Skeleton information. This way granularity of the cutting process is limited to keyframes and you can be sure that the generated file will play back correctly even in older players that do not support the Skeleton.

# Join Streams

Ogg Video Tools provides the command line tool **oggJoin** to interleave two or more streams into one Theora video file. It is able to interpret the page granule position and interleave the streams in case they contain Theora video or Vorbis audio.

This process is also called multiplexing.

Synopsis: **oggJoin newFile.ogv stream1.ogv stream2.oga [ stream3.oga [...] ]**

**Note:** The video stream (.ogv) *must* come first.

Although oggJoin allows arranging the streams in any possible order, the Theora standard demands that the video stream be the first stream. Correctly stated: the Theora header must be the first header within a multiplexed file, in order to allow players to differentiate between audio only and video files. This is achieved by placing the video stream first stream in the list, as shown in the synopsis.



# Ogg File Concatenation

To concatenate two or more Ogg files with audio and video streams into a single video file without reencoding is easy, but can lead to the same synchronization problems as using oggCut.

The command line tool oggCat creates a new Ogg file from a number of other Ogg files.

Synopsis: **oggCat outputFile.ogv inputfile1.ogv inputfile2.ogv [ inputfile3.ogv [ .. ] ]**

To create an Ogg file, the parameters for all streams must match. If one or more parameters in an input file do not match, that file will be ignored, but the process of creating the output file will continue. The output file is a valid ogg file. It is just not the ogg file desired.

## The Process

oggCat reads in the first file mostly as it is. However, it does not handle any unknown streams and does not integrate them into the output file.

To find a corresponding stream within the succeeding Ogg files, oggCat creates a parameter table with all necessary information about the streams of the first file (inputfile1.ogv), so that the first file serves as a "reference".

The streams in the following files (inputfile2.ogv inputfile3.ogv ...) are tested against the parameters of the reference table, so they do not have to be in the same order as the reference file.

### Example:

The first Ogg file consists of one theora and one vorbis stream. The theora stream video has a fr

The next file consists of three streams: two theora streams and one vorbis stream. oggCat can han

If there are no matching parameters in any of the streams, oggCat prints a line suggesting how the file might be reencoded:

```
vorbis parameter compare: data rate not matching 32000 != 64000
You may try to reencode with the datarate of the other file
```

```
Please try to resample with the following command
oggResize -D 32000 <file see below>
```

## Using cat

Another way to concatenate two videos is by adding one file after another with the command line tool **cat**. This method triggers the player to setup a new decoding session for the next stream.

The Ogg Standard allows opening a stream after another stream has ended, but most players do not implement this method of re-opening a decoder session. Therefore this method should always be tested with the players that should be used after this session.

## oggCat and oggz-chop

When a file has been created with oggz-chop, concatenation is not possible because of the way oggCat works.

The reason is that files with a skeleton and different time offsets defined in this skeleton for every stream (as is the case for most files created with oggz-chop) can not be concatenated to one continuous stream. This behavior is caused by the fact that two ogg streams would overlap. The presented frames would overlap the ones that are not presented, as they are only needed for the decoding of the first video frame. This is not possible in one Ogg stream.

## **Synchronizing issues with oggCat**

Different streams in one Ogg file may not have the (exact) same length. During the concatenation process the streams of the next Ogg file in the concatenation list are placed directly after the corresponding streams of the starting file without any time gaps. Therefore synchronization of the second file will suffer from an inaccurate stream ending of the first file.

# OggResize

The command line tool **oggResize** can resize the height and width of an ogg file (ogg, oga or ogv) in various ways. However, it does only handle Ogg Theora and Vorbis streams. The advantage of oggResize over other tools is that it converts only the streams that need to be changed. Streams that need no conversion will not be effected.

oggResize can change the **video frame size**, video or audio **datarate** and the video **frame rate** or audio **sample rate**.

It can also add **comments** to any stream and can include **PNG images** with an alpha channel, which are rendered into the video at any time period before or after the resizing process.

Synopsis: **oggResize [options] inputfile.ogv outputfile.ogv**

The available options are:

**-s** sets the size of the video frame. The size is given as `<width>x<height>`. The default size is 480x320. By default the aspect ratio of the original video frame is kept as it is, so if the size of the new video frame does not match in aspect ratio, oggResize adds black borders at the top and bottom or right and left, to fit the new aspect ratio.

If you want to stretch the video frames to the new size, use the **-t** option.

*Example: -s 320x240*

**-d** sets the datarate in **bits per second** for the video encoder (theora). This is meant to be an upper threshold. So the file may be smaller than assumed. If not set, the datarate of the original stream is used.

*Example: -d 1024000*

**-D** sets the datarate in **bits per second** for the audio encoder (vorbis). If not set, the datarate of the original stream is used.

*Example: -D 64000*

**-f** sets the frame rate of the video with numerator and denominator and is the frames per second. If only one number is given, the denominator is set to 1. If not set, the frame rate of the original video is used.

*Example: -f 25:2*

**-F** sets the sample frequency (sample rate) of the audio data in Hertz. If the sample frequency does not match the original file, resampling is invoked.

*Example: -F 32000*

**-t** if this option is set, the picture will be stretched to the output size. In this case, the aspect ratio of the output video may not match the original video frame aspect ratio. This option help omitting the black borders, that appear if the aspect ratio of the original and the new video do not match.

*Example: -t*

**-c** adds comments to the video (theora) stream. Comments are in the form 'type=value'. More than one comment can be concatenated with a semicolon. It is recommended that apostrophes (single quote) be used, as the command line may use the semicolon as a separator.

*Example: -c 'AUTHOR=yorn;DATE=03.07.09'*

**-C** adds comments to the audio (vorbis) stream. Comments are in the form 'type=value'. More than one comment can be concatenated with a semicolon. It is recommended that apostrophes (single quote) be used, as the command line may use the semicolon as a separator.

*Example: -C 'AUTHOR=yorn;DATE=03.07.09'*

**-q** specifies the quality for the resizing process. Values can be chosen between 1 (best quality, with slight blurring) and 6 (worst quality). The default value is 2.

*Example: -q1*

**-p** creates a preview film. The number given with this option specifies the interval between the frames selected, e.g. *-p24* means that every 24th frame is selected/shown. Thus the newly created video plays 24 times faster. This option can be combined with the *-f* option to control the frame rate. With both options nice video previews can be created. If *-p* is used, the audio stream is ignored.

*Example: -p 24*

**-a** adds a picture to the video frame before it is resized. The picture does not have to fit into the video frame. It is always placed in the upper left corner of the video frame.

*Expression: <picture1.png>[,<startTime>[,<endTime>[,s]]]*

*The default start time is 0. The default end time is -1. By default s is not set. In this case the picture does not fade in. If s is set then the picture does fade in.*

*It is possible to superimpose more than one picture on a video frame. To concatenate the expressions use the colon. If the timelines overlap, the pictures are superimposed on one another, so the last picture is the foreground layer.*

*Example: -a etwas.png,2,7,s:etwasneues.png,5,10*

**-A** adds a picture to the video frame after it is resized.

*The syntax is the same expression as with -a option*

## Examples

*Changing the video frame size and the datarate:*

```
oggResize -s320x240 -d512000 orig.ogv new.ogv
```

*This command converts the video file orig.ogv to the new video file new.ogv. The frame size of the newly created video is set to 320x240 pixels and the data rate is set to 512 kByte/s. If the parameters of the original file fit the new parameters, the video stream is just copied, without changing the data. If the video frame size is different, it is changed to the new size by some appropriate algorithms.*

*If there was an audio stream within the orig.ogv file, it is copied into the new file.*

*Changing the audio data rate, sample rate and numbers of channels:*

```
oggResize -D64000 -F16000 -N1 orig.ogv new.ogv
```

*This command line converts only the audio stream of file orig.ogv to a sample rate of 16kHz, a datarate of 64 kBytes/s and a mono channel. As with video, the audio stream is only reencoded, if the given parameters do not match that ones in the vorbis audio stream. The video stream is copied as is.*

*This is a very fast method of reencoding only the audio stream, without any other procedures such as splitting and rejoining the streams of an ogg file.*

*Changing the video frame size, the audio data rate, the video data rate and adding two pictures on top of the video for a period of time:*

```
oggResize -s300x200 -D32000 -d1024000 -A etwas.png,2,7,s:etwasneues.png,5,10 \ orig.ogv new.ogv
```

*This command line converts the audio and video stream as described in the before mentioned examples.*

*Additionally it adds a picture in PNG format and an alpha channel **etwas.png** to the video from second 2 to second 7 with a smooth fade in and fade out and the picture **etwasneues.png** is placed on top of the video frame from second 5 to second 10 without any fading.*

# Analysing Ogg Files

There are a number of tools available to analyse ogg files from the packages **Vorbis tools**, **oggz tools** and **Ogg Video Tools**. As with editing, the tools may overlapping in functionality, however here we will focus on the usual usage of the available tools and the fields where they could be used.

The tools that are discussed here are :

From **Vorbis tools**:

- ogginfo

From **oggz tools**:

- oggz-info
- oggz-comment
- oggz-validate
- oggz-sort
- oggz-dump
- oggz-sort

From **Ogg Video Tools**:

- oggDump
- oggLength

## Information about an Ogg file

If you want to get some information about what streams are available within a file and what nature these streams have, you use **ogginfo** or **oggz-info**.

```
$ oggz-info big_buck_bunny_480p_stereo-small.ogv Content-Duration: 00:09:56.384 Theora: serialno 0000020573 9546 packets in 6797 pages, 1.4 packets/page, 1.037% Ogg overhead Video-Framerate: 16.000 fps Video-Width: 320 Video-Height: 240 Vorbis: serialno 0000003594 18641 packets in 576 pages, 32.4 packets/page, 1.412% Ogg overhead Audio-Samplerate: 16000 Hz Audio-Channels: 2
```

The printed output here is about which streams are available within the Ogg file and the parameters the streams have (e.g. for theora the video frame size and the frame rate and for vorbis the sample rate and the channel number).

**ogginfo** even prints out more detailed information, e.g. version information and all information available from the video and audio header packets, e.g. aspect ratio or colour spaces.

**oggz-info** can tell you more about an Ogg file. In particular, the **-a** option will tell you even more detailed information on your file. However, these information are mostly from the statistical area.

To read or edit the comments fields, that are delivered with every stream within the Ogg file. To do so, the oggz tools provide the **oggz-comment** command line tool.

Example for printing the comments.

```
$ oggz-comment -l big_buck_bunny_480p_stereo-small.ogg
Theora: serialno 0000020573
        Vendor: Xiph.Org libTheora I 20081020 3 2 1
```

```

ENCODER: oggvideotools-0.8-win32
Vorbis: serialno 0000003594
Vendor: Xiph.Org libVorbis I 20090514
ENCODER: oggvideotools-0.8

```

To add a comment, you add a "Tag=Value" pair to the end of your command line.

```
$ oggz-comment big_buck_bunny_480p_stereo-small.ogg -o bigbuck_comment.oggv \ "LICENCE=Creative Co
```

```

$ oggz-comment -l bigbuck_comment.oggv
Theora: serialno 0000020573
Vendor: Xiph.Org libTheora I 20081020 3 2 1
ENCODER: oggvideotools-0.8-win32
LICENCE: Creative Commons CC-BY-SA
Vorbis: serialno 0000003594
Vendor: Xiph.Org libVorbis I 20090514
ENCODER: oggVideoTools 0.8
LICENCE: Creative Commons CC-BY-SA

```

You can also add a comment only to one of the available streams or delete comments. Please refer to the help page for these more advanced options.

## Analysing an Ogg file in detail

Mainly for developers, it is often useful to have a look into the stream itself and to analyse if an Ogg file is compliant with the Ogg Standard.

For validating a file you use the command line tool **oggz-validate**:

```

$ oggz-validate input.oggv
input.oggv: Error:
serialno 1101839243: Terminal header page has non-zero granulepos
serialno 1101839243: Terminal header page contains non-header segment

```

A rather common problem that **oggz-validate** might report, is badly sorted Ogg files. These will usually play, but may cause issues such as intermittent stuttering, or increased memory usage. The **oggz-sort** tool may be used to correct these sorting issues:

```
$ oggz-sort -o output.oggv input.oggv
```

A more detailed view can be given by the tools **oggz-dump** and **oggDump**. Both of these tools write detailed output data to the console.

**oggz-dump** can output the packets of all or some specific streams within a given Ogg file.

```

$ oggz-dump myfile.oggv
[ ... ]
0b40: 6f13 abc9 f3ac 9dc2 cec5 9c62 0e70 1fab o.....b.p..
0b50: f891 01a8 0633 430e 308a 6f8c 86c3 131a .....3C.0.o.....
0b60: 3ab6 840d edab e79f fe24 a4 :.. .....$.

00:00:32.104: serialno 0938763527, calc. gpos 513664, packetno 1022: 269 bytes
0000: 3eb7 e92b 50b4 0028 a93c 8bc4 0039 0948 >..+P..(<...9 H
0010: 1d58 0000 0080 5f03 db01 ee92 2482 48dd .X...._.....$.H.
0020: d73f 2562 feeb de81 94fe 75fb b7ff decf .?%b.....u.....
0030: 18c0 b830 619f b6c5 cf0d 00f3 b5f5 bbbd ...0a.... .....
[ ... ]

```

With **oggDump** you can decide if you like to see the pages or the packet information of an Ogg file by setting the **-p** (packet) or **-g** (page) option. Further more you can specify the detail level of the stream information

output by using `-l` (detail). The detail can be set from 1 (not detailed) to 5 (most details):

```
$ oggDump -l3 -g myVideo.ogv
[ ... ]
Ogg Page: header length = 42 and body length = 3600
Header Information:
    Ogg Version      : 0
    Serial No       : 0x37f46507
    Packet Type     : fresh packet
    Page Type      : normal page
    Last Page      : normal page
    Granule Position : 0(0x0)
    Page Number    : 1
    Checksum       : 0x5ced317c
    Table Segments  : 15

Segments:
67 ff ff ff ff ff ff ff ff ff ff ff ff ff b6

Header Hex dump:
4f 67 67 53 00 00 00 00 00 00 00 00 00 07 65
f4 37 01 00 00 00 7c 31

[ ... ]
```

## Tips and tricks

This subsection lists some useful ways to use several of the tools described above.

### Getting the Duration of a Stream

To figure out the duration of a video file you can use **oggz-info** or **ogginfo**, but here you need to extract the information via script from the output. **oggLength** gives you another way to receive the length directly, so you can use it more easily.

#### Create an Ogg Vorbis file filled with silence that fits exactly to a video

```
$ oggSilence -l`oggLength videoFile.ogv` -o audioSilence.oga
```

### Creating a sound byte from a portion of a video

These two commands save a short 5 second audio clip from 40 seconds into a video:

```
$ oggz-rip -c vorbis video.ogv -o temp.ogg
$ oggz-chop -s 40 -e 45 -o soundbyte.ogg temp.og
```

### Adding an embedded text subtitles stream

You've written German subtitles for an English language video you found on the web ?

```
$ kateenc -t srt -l de -c SUB -o subtitles.ogg subtitles.srt
$ oggz-merge -o video-with-german-subtitles.ogv original-video.ogv subtitles.ogg
```

### Getting a list of all packets in a stream in a very compressed way

```
$ oggz-dump video.ogv | grep packetno | less
```



00:00:00.437: serialno 0000020573, calc. gpos 0|7, packetno 10: 414 bytes 00:00:00.500: serialno 0000020573, calc. gpos 0|8, packetno 11: 692 bytes 00:00:00.562: serialno 0000020573, calc. gpos 0|9, packetno 12: 600 bytes 00:00:00.625: serialno 0000020573, granulepos 0|10, packetno 13: 859 bytes 00:00:00.687: serialno 0000020573, calc. gpos 0|11, packetno 14: 700 bytes 00:00:00.750: serialno 0000020573, calc. gpos 0|12, packetno 15: 1.154 kB 00:00:00.812: serialno 0000020573, calc. gpos 0|13, packetno 16: 878 bytes 00:00:00.875: serialno 0000020573, granulepos 0|14, packetno 17: 1.342 kB 00:00:00.000: serialno 0000003594, calc. gpos 0, packetno 3: 118 bytes 00:00:00.032: serialno 0000003594, calc. gpos 512, packetno 4: 115 bytes 00:00:00.064: serialno 0000003594, calc. gpos 1024, packetno 5: 127 bytes

## Other Interesting Things

Here are some additional projects you might find interesting. You will need Firefox 3.5 to view most of these.

### Video and SVG

[http://www.double.co.nz/video\\_test/video.svg](http://www.double.co.nz/video_test/video.svg)

### Video and CSS Transforms

<http://www.zachstronaut.com/lab/isocube.html>

<http://hacks.mozilla.org/2009/06/tristan-washing-machine/>

<http://hacks.mozilla.org/2009/07/video-more-than-just-a-tag/>

### Video and Subtitles

<http://people.mozilla.com/~prouget/demos/srt/index2.xhtml>

<http://people.mozilla.com/~prouget/demos/DynamicContentInjection/play.xhtml>

<http://www.annodex.net/~silvia/itext/>

[http://www.annodex.net/~silvia/itext/elephant\\_no\\_skin.html](http://www.annodex.net/~silvia/itext/elephant_no_skin.html)

### Replace background with image

<http://people.mozilla.com/~prouget/demos/green/green.xhtml>

# Glossary

**absolute URL** Link specifying the complete path to a file. Contrast with *relative URL*.

**codec** Short for '*coder-decoder*'; a device or computer program capable of encoding and/or decoding a digital data stream or analog signal.

**compression** A method for generating smaller files for transmission or storage, with the capability of regenerating the original data exactly (*lossless* compression) or approximately (*lossy* compression).

**container** A file format that specifies how different streams of data or files (such as encoded audio and video) can be stored or sent over a network together. Also called a wrapper. A particular container format can include video data in many encodings. The .ogg format combines Ogg Vorbis audio and Ogg Theora video, plus *metadata*. Matroska (.mkv), .mp4, and .avi containers can also contain Ogg Theora video.

**decoder** A device or computer program for converting a compressed or otherwise encoded file or data stream back to the original format and either the exact original data (*lossless* compression), or some approximation of it (*lossy* compression).

**demultiplex** Convert a file or *stream* containing multiple data streams to separate files or streams containing one stream each.

**distribution** In GNU/Linux and BSD, a set of software selected, maintained, and distributed under Free license by some organization or individual, including the operating system, essential utilities and data, and applications. For example, Ubuntu Linux or FreeBSD.

**encoder** A device or computer program for converting a source file or data *stream* to another format, usually compressed.

**Firefogg** Firefox add-on for encoding video in Ogg Theora while it is uploading to a Web site. Requires specific software in the Web server.

**GUI** Graphical User Interface

**H.264** A patent-encumbered *codec* used by Apple, Youtube, and others.

**HTML5** Version 5 of Hyper Text Markup Language, which includes new functions for managing video.

**jar file** Java archive file containing software written in Java.

**key frame** In video *compression*, a frame that is described in full, usually at the start of a scene or cut. Succeeding frames can be described by their differences from a preceding frame.

**lossy** In data compression and encoding, an irreversible algorithm that enables recovery of an approximate copy of the original data. Some lossy algorithms permit the user to set the degree of accuracy in the encoding step.

**lossless** In data compression and encoding, a reversible algorithm that enables recovery of a bit-perfect copy of the original data.

**metadata** Information about the contents of a file, such as title, author, and so on.

**NGO** Non-Governmental Organization, usually but not always a non-profit corporation.

**patent, software** A licensed monopoly on an idea or algorithm. The legality and desirability of software patents are hotly debated, and a good deal of ingenuity is put into circumventing them.

**relative URL** Link to a file starting from the current location in the file system. Contrast with *absolute URL*.

**split** *Demultiplex*.

**stream** A continuous data transmission without a predefined size, as opposed to a file of some specific size.

**streaming** Real-time delivery of multimedia content, in contrast with downloading followed by playing.

**torrent** 1) Distributed download software such as BitTorrent, and its method of operation, in which pieces of a file are assembled from multiple sites. Users who have downloaded a file, and have more upload bandwidth than they need, offer to let others get part of the file from them. 2) Download software using a torrent client. 3) The tiny file that specifies how to torrent a much larger file.

**transcode** Change content from one encoding to another. When transcoding, best results are attained when starting from the originals, as successive lossy encodings lose quality with each step.

**wrapper** *A container*.

# License

All chapters copyright of the authors (see below). Unless otherwise stated all chapters in this manual licensed with **GNU General Public License version 2**

This documentation is free documentation; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This documentation is distributed in the hope that it will be useful, but **WITHOUT ANY WARRANTY**; without even the implied warranty of **MERCHANTABILITY** or **FITNESS FOR A PARTICULAR PURPOSE**. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this documentation; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

# Authors

## *ABOUT THIS MANUAL*

© adam hyde 2009

Modifications:

David Köhling 2009

William Merriam 2009

---

## *ACCESSIBILITY*

© TWikiGuest 2009

Modifications:

adam hyde 2009

Silvia Pfeiffer 2009

---

## *ANALYSING OGG*

© Mark Hancock 2009

Modifications:

adam hyde 2009

Jan Gerber 2009

ogg k 2009

Jörn Seger 2009

---

## *STREAM RIPPING*

© adam hyde 2007, 2009

Modifications:

Alan Toner 2009

Edward Cherlin 2009

Holmes Wilson 2009

Jan Gerber 2009

ogg k 2009

Thomas Middleton 2008

Jörn Seger 2009

---

## *CAT FILES*

© adam hyde 2009

Modifications:

Robert Valliant 2009

Jörn Seger 2009

---

## *CODECS*

© TWikiGuest 2008

Modifications:

adam hyde 2009

Holmes Wilson 2009

Jaap Hoetmer 2009

Jan Gerber 2009

ogg k 2009

Robert Valliant 2009

susanne lang 2009

Jörn Seger 2009

---

## *CONTAINERS*

© TWikiGuest 2008

Modifications:  
adam hyde 2009  
Holmes Wilson 2009  
Jan Gerber 2009  
ogg k 2009  
susanne lang 2009  
JÃ¶rn Seger 2009

---

#### *DISTRIBUTION*

© adam hyde 2007, 2009  
Modifications:  
Holmes Wilson 2009  
ogg k 2009  
susanne lang 2009  
Thomas Middleton 2008

---

#### *CREATE PREVIEW*

© JÃ¶rn Seger 2009  
Modifications:  
adam hyde 2009  
Alan Toner 2009  
David KÃ¼hling 2009

---

#### *CREATE SLIDESHOWS*

© JÃ¶rn Seger 2009  
Modifications:  
adam hyde 2009  
Alan Toner 2009  
David KÃ¼hling 2009  
Edward Cherlin 2009

---

#### *CREATE THUMBNAILS*

© JÃ¶rn Seger 2009  
Modifications:  
adam hyde 2009  
Alan Toner 2009  
David KÃ¼hling 2009  
Mark Hancock 2009

---

#### *CREDITS*

© adam hyde 2006, 2007, 2009

---

#### *CUT A FILE*

© adam hyde 2009  
Modifications:  
David KÃ¼hling 2009  
Edward Cherlin 2009  
Robert Valliant 2009

---

#### *INTRODUCTION*

© adam hyde 2007, 2009  
Modifications:  
Alan Toner 2009

Edward Cherlin 2009  
Holmes Wilson 2009  
Jan Gerber 2009  
salsa man 2009  
susanne lang 2009  
Thomas Middleton 2008  
JÃ¶rn Seger 2009

---

#### *EMBEDDING SUBTITLES*

© adam hyde 2009  
Modifications:  
Edward Cherlin 2009  
Holmes Wilson 2009  
Jan Gerber 2009  
susanne lang 2009

---

#### *INTRODUCTION*

© TWikiGuest 2008  
Modifications:  
adam hyde 2009  
David Kä¼hling 2009  
Jan Gerber 2009  
jay maechtlen 2009  
JÃ¶rn Seger 2009

---

#### *FFMPEG2THEORA*

© TWikiGuest 2008  
Modifications:  
adam hyde 2009  
David Kä¼hling 2009  
Edward Cherlin 2009  
Holmes Wilson 2009  
Jan Gerber 2009  
ogg k 2009  
JÃ¶rn Seger 2009

---

#### *FFMPEG2THEORA*

© adam hyde 2007, 2009  
Modifications:  
Alan Toner 2009  
Holmes Wilson 2009  
Jan Gerber 2009  
Thomas Middleton 2008  
JÃ¶rn Seger 2009

---

#### *FIREFOGG*

© maxigas maxigas 2007  
Modifications:  
adam hyde 2009  
David Kä¼hling 2009  
Jan Gerber 2009  
Peter W 2009  
Thomas Middleton 2008  
JÃ¶rn Seger 2009



---

## *GLOSSARY*

© adam hyde 2006, 2009

Modifications:

Edward Cherlin 2009

Holmes Wilson 2009

Jan Gerber 2009

ogg k 2009

Thomas Middleton 2008

---

## *HTML5*

© Jörn Seger 2009

Modifications:

adam hyde 2009

Ali Gunduz 2009

Andrew Nicholson 2009

Jan Gerber 2009

Michael Dale 2009

ogg k 2009

Robert Valliant 2009

Silvia Pfeiffer 2009

---

## *HOSTING SITES*

© TWikiGuest 2008

Modifications:

adam hyde 2009

Andrew Nicholson 2009

Holmes Wilson 2009

Jan Gerber 2009

---

## *ICECAST*

© adam hyde 2006, 2007, 2008

Modifications:

Zita Joyce 2008

---

## *INTERESTING EXTRAS*

© TWikiGuest 2009

Modifications:

adam hyde 2009

Jan Gerber 2009

Silvia Pfeiffer 2009

---

## *INTRODUCTION*

© adam hyde 2006, 2007, 2009

Modifications:

Brylie Oxley 2009

David Köhling 2009

Edward Cherlin 2009

Holmes Wilson 2009

Jan Gerber 2009

Maik Merten 2009

ogg k 2009

susanne lang 2009

Thomas Middleton 2008

---

### *JOIN FILES*

© adam hyde 2009

Modifications:

David Kä¼hling 2009

Robert Valliant 2009

---

### *INTRODUCTION*

© TWikiGuest 2008

Modifications:

adam hyde 2009

David Kä¼hling 2009

Edward Cherlin 2009

Holmes Wilson 2009

ogg k 2009

Robert Valliant 2009

JÄ¼rn Seger 2009

---

### *MIRO*

© JÄ¼rn Seger 2008

Modifications:

adam hyde 2009

Holmes Wilson 2009

TWikiGuest 2008

Thom Hastings 2009

---

### *HOSTING THEORA YOURSELF*

© TWikiGuest 2008

Modifications:

adam hyde 2009

Chris Double 2009

Holmes Wilson 2009

Jan Gerber 2009

susanne lang 2009

---

### *RESIZING*

© Jan Gerber 2009

Modifications:

adam hyde 2009

Robert Valliant 2009

JÄ¼rn Seger 2009

---

### *INTRODUCTION*

© TWikiGuest 2008

Modifications:

adam hyde 2009

Jan Gerber 2009

ogg k 2009

susanne lang 2009

JÄ¼rn Seger 2009

---

### *PITIVI*

© adam hyde 2006, 2009

Modifications:  
Edward Cherlin 2009  
Holmes Wilson 2009  
Jan Gerber 2009  
Thomas Middleton 2008

---

### *HOW TO PLAY THEORA*

© TWikiGuest 2008  
Modifications:  
adam hyde 2009  
Gergely Mt© 2009  
Holmes Wilson 2009  
Jan Gerber 2009  
ogg k 2009  
Robert Valliant 2009  
Stjepan Rajko 2009  
susanne lang 2009

---

### *PLAYING SUBTITLES*

© adam hyde 2009  
Modifications:  
Edward Cherlin 2009  
Holmes Wilson 2009  
susanne lang 2009

---

### *PUBLISHING*

© adam hyde 2007, 2009  
Modifications:  
another sam 2009  
Edward Cherlin 2009  
Holmes Wilson 2009  
Jan Gerber 2009  
Mark Grandi 2009  
susanne lang 2009  
Thomas Middleton 2008

---

### *SPLIT A FILE*

© adam hyde 2009  
Modifications:  
David Khling 2009  
ogg k 2009  
Robert Valliant 2009  
Jrn Seger 2009

---

### *INTRODUCTION*

© ogg k 2009  
Modifications:  
adam hyde 2009  
Edward Cherlin 2009  
Holmes Wilson 2009  
susanne lang 2009

---

### *TSS*

© TWikiGuest 2008

Modifications:  
adam hyde 2009  
Alan Toner 2009  
alejo duque 2009  
Holmes Wilson 2009

---

*THEORA THEORY*

© JÃ¶rn Seger 2008, 2009

Modifications:  
adam hyde 2008, 2009  
Alan Toner 2009  
David KÃ¼hling 2009  
Robert Valliant 2009

---

*THOGEN*

© adam hyde 2007, 2009

Modifications:  
David KÃ¼hling 2009  
Edward Cherlin 2009  
Jan Gerber 2009  
ogg k 2009  
JÃ¶rn Seger 2009

---

*VLC*

© TWikiGuest 2008

Modifications:  
adam hyde 2009  
David KÃ¼hling 2009  
Dennis J 2009  
Mark Hancock 2009  
JÃ¶rn Seger 2009

---

*VLC*

© TWikiGuest 2008

Modifications:  
adam hyde 2009  
Holmes Wilson 2009  
Jan Gerber 2009

---

*VLC*

© adam hyde 2007, 2009

Modifications:  
Alan Toner 2009  
Edward Cherlin 2009  
jay maechtlen 2009  
Thomas Middleton 2008  
JÃ¶rn Seger 2009

---

*WHAT IS STREAMING?*

© TWikiGuest 2008

Modifications:  
adam hyde 2009  
Alan Toner 2009  
JÃ¶rn Seger 2009

---

---

*WHAT IS THEORA?*

© TWikiGuest 2008

Modifications:

adam hyde 2009

Edward Cherlin 2009

Holmes Wilson 2009

jay maechtlen 2009

Joshua Gay 2009

Lachlan Musicman 2009

susanne lang 2009

---

*WHAT IS VIDEO?*

© TWikiGuest 2008

Modifications:

adam hyde 2009

Brylie Oxley 2009

Edward Cherlin 2009

Holmes Wilson 2009

Lachlan Musicman 2009

susanne lang 2009

---

*INTRODUCTION*

© Mark Hancock 2009

Modifications:

adam hyde 2009

Alan Toner 2009

David Köhling 2009

susanne lang 2009

Jörn Seger 2009

---



Free manuals for free software

# General Public License

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.  
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

## Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

## TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

**0.** This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without

limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

**1.** You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

**2.** You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

**a)** You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

**b)** You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

**c)** If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

**3.** You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

- 4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
- 5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.
- 6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
- 7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.



It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

**8.** If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

**9.** The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

**10.** If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

## **NO WARRANTY**

**11.** BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

**12.** IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

**END OF TERMS AND CONDITIONS**