

Você está aqui: [Home](#) ▸ [Dive Into HTML5](#) ▸

# Nº 5. VÍDEO NA WEB

[exibir índice analítico](#)



## MERGULHANDO



Qualquer um que tenha visitado o Youtube.com nos últimos quatro anos sabe que você pode incorporar um vídeo em uma página web. Mas antes do HTML5, não havia nenhuma maneira baseada nos padrões para fazer isso. Virtualmente todos os vídeos já assistidos “na web” eram afunilados por um plugin de terceiros — talvez QuickTime, talvez RealPlayer, talvez Flash. (YouTube usa Flash.) Esses plugins se integram com seu browser o

suficiente que você nem é avisado quando está utilizando eles. Isso é, até o momento em que você tentar assistir um vídeo em uma plataforma que não suporta esse plugin.

HTML5 define uma maneira padrão de incorporar vídeo em uma página web, usando o elemento `<video>`. O suporte para o elemento `<video>` ainda está evoluindo, o que é um modo educado de dizer que ainda não está funcionando. Pelo menos, ele não funciona em todos os lugares. Mas não se desespere! Existem alternativas, fallbacks e uma abundância de opções.

### SUPOORTE AO ELEMENTO `<VIDEO>`

IE	FIREFOX	SAFARI	CHROME	OPERA	IPHONE	ANDROID
9.0+	3.5+	3.0+	3.0+	10.5+	1.0+	2.0+

Mas o suporte para o elemento `<video>` em si é realmente uma pequena parte da história. Antes de falarmos sobre o vídeo da HTML5, você precisa entender primeiro um pouco sobre vídeo em si. (Se você já sabe sobre vídeo, você pode pular para [O Que Funciona na Web.](#))



## EMBALAGENS DE VÍDEO

Você pode pensar em arquivos de vídeo como “arquivos AVI” ou “arquivos MP4.” Na realidade, “AVI” e “MP4” são apenas os formatos das embalagens de vídeo. Assim como um arquivo ZIP pode conter qualquer tipo de arquivo dentro dele, os

formatos das embalagens de vídeo definem apenas *como* armazenar as coisas dentro dele, e não *quais* tipos de dados são armazenados. (É um pouco mais complicado que isso, porque nem todos os fluxos de vídeo são compatíveis com todos os formatos de embalagens, mas esqueça isso por enquanto.)

Um arquivo de vídeo usualmente contém múltiplas *faixas* — uma faixa de vídeo (sem áudio), além de uma ou mais faixas de áudio (sem vídeo). As faixas usualmente se interlaçam. Uma faixa de áudio contém marcações dentro dela para ajudar na sincronização entre áudio e vídeo. Faixas individuais podem ter metadados, como relações de aspectos de uma faixa de vídeo, ou a linguagem de uma faixa de áudio. Essas embalagens também podem ter metadados, como o título do próprio vídeo, uma capa para o vídeo, números de episódios (para programas de televisão), e por aí vai.

Existem *vários* formatos para embalagens de vídeo. Os mais populares incluem

- [MPEG 4](#), usualmente com a extensão `.mp4` ou `.m4v`. A embalagem MPEG 4 é baseada na antiga embalagem QuickTime da Apple (`.mov`). [Trailers de filmes no site da Apple](#) ainda utilizam a antiga embalagem QuickTime, mas filmes que você aluga no iTunes são entregues na embalagem MPEG 4.
- [Flash](#), usualmente com a extensão `.flv`. Vídeos em Flash são, sem surpresa, usados pelo Adobe Flash. Antes do Flash 9.0.60.184 (a.k.a. Flash Player 9 Update 3), esse era o único formato de embalagem que o Flash suportava. Versões mais recentes do Flash também suportam a embalagem MPEG 4.
- [Ogg](#), usualmente com a extensão `.ogv`. Ogg é um padrão aberto, de código livre, e descoberto de quaisquer patentes conhecidas. Firefox 3.5, Chrome 4, e Opera 10.5 suportam — nativamente, sem qualquer plataforma específica de plugins — o formato de embalagem Ogg, Ogg vídeo (chamado “Theora”), e Ogg áudio (chamado “Vorbis”). No desktop, Ogg é suportado fora-da-caixa por todas as principais distribuições Linux, e você pode usá-lo no Mac e Windows ao instalar os [componentes QuickTime](#) ou [filtros DirectShow](#), respectivamente. É ainda executado pelo excelente [VLC](#) em todas as plataformas.

- WebM é um novo formato de embalagem. É tecnicamente similar a outro formato, chamado Matroska. WebM foi anunciado em Maio de 2010. Foi projetado para ser usado exclusivamente com o codec de vídeo VP8 e pelo codec de áudio Vorbis. (Mais sobre isso em um minuto.) É formato nativamente, sem qualquer plataforma específica de plugins, nas últimas versões do Chromium, Google Chrome, Mozilla Firefox, e Opera. Adobe também anunciou que uma futura versão do Flash irá suportar vídeos em WebM.
- Audio Video Interleave, usualmente com a extensão .avi. O formato de embalagem AVI foi inventado pela Microsoft em um tempo mais simples, quando o fato de que computadores podiam tocar vídeos já era considerado incrível. Oficialmente não suporta funcionalidades dos mais recentes formatos de embalagens como metadados incorporados. Oficialmente não suporta até mesmo os mais modernos codecs de áudio e vídeo utilizados hoje em dia. Com o tempo, companhias tentaram extendê-lo de forma incompatível para suportar isso ou aquilo, e ainda é o formato de embalagem padrão para encoders como MEncoder.



## CODECS DE VÍDEO

Quando você fala sobre “assistir um vídeo,” você está provavelmente falando da combinação de um fluxo de vídeo e um fluxo de áudio. Mas você não tem dois arquivos diferentes; você tem apenas “um vídeo.” Talvez possa ser um arquivo AVI, ou um arquivo MP4. Esses são apenas formatos de embalagem, como um arquivo ZIP que contém múltiplos tipos de arquivos dentro dele. O formato de embalagem define como serão armazenados os fluxos de vídeo e áudio em um único arquivo.

Quando você “assiste um vídeo,” seu video player está fazendo pelo menos três coisas ao mesmo tempo:

1. Interpretando o formato de embalagem para descobrir quais faixas de vídeo e áudio estão disponíveis, e como elas são armazenadas dentro do arquivo para que possa encontrar os dados que necessitam ser decodificados depois
2. Decodificando o fluxo de vídeo e exibindo uma série de imagens na tela
3. Decodificando o fluxo de áudio e enviando o som para as caixas de som

Um *codec de vídeo* é um algoritmo que será encodificado em um fluxo de vídeo, i.e. ele especifica como fazer a seguir. (A palavra “codec” é um portmanteau, a combinação das palavras “codificar” e “decodificar.”) Seu video player *decodifica* o fluxo de vídeo de acordo com o *codec de vídeo*, depois exibe uma série de imagens, ou “frames,” na tela. A maioria dos codecs de vídeo modernos usam diversas formas para minimizar a quantidade de informação necessária para exibir um frame atrás do outro. Por exemplo, ao invés de armazenar cada frame individualmente (como um screenshot), eles irão armazenar apenas as diferenças entre os frames. A maioria dos vídeos na realidade não mudam completamente entre um frame e o outro, então isso permite alto grau de compressão, resultando em menores tamanhos de arquivo.

Existem codecs de vídeo *com perdas* e *sem perdas*. Vídeos sem perdas são muito grandes para serem usados na web, então irei me concentrar em codecs com perdas. Um *codec com perdas* significa que informação será irremediavelmente perdida durante o processo de encodificação. Como a cópia do áudio de uma fita cassete, você está perdendo informação sobre a fonte do vídeo, e degradando a qualidade, cada vez que você codifica. Ao invés do “assobio” de uma fita cassete de áudio, uma re-re-re-encodificação de vídeo pode parecer bloqueada, especialmente durante cenas com muita ação. (Na verdade, isso pode acontecer até se você codificar direto da fonte original, se você escolher um codec de vídeo pobre ou passar para ele o conjunto errado de parâmetros.) O lado bom é que codecs de vídeo com perdas podem oferecer taxas de compressão incríveis através da suavização sem bloqueios durante a reprodução, para fazer a perda menos perceptível ao olho humano.

Existe uma porção de codecs de vídeo. Os três codecs mais relevantes são H.264, Theora, e VP8.

## H.264

H.264 mais conhecido como “MPEG-4 part 10,” a.k.a. “MPEG-4 AVC,” a.k.a. “MPEG-4 Advanced Video Coding.” H.264 foi desenvolvido pelo MPEG group e padronizado em 2003. Tem por objetivo oferecer um único codec para banda-larga pobre, CPU de dispositivos pobres (celulares); alta banda-larga, alta CPU de dispositivos (computadores modernos); e qualquer coisa no meio disso. Para realizar isso, o padrão H.264 é dividido em “perfis,” onde cada um define um conjunto de funcionalidades opcionais que negocia a complexidade pelo tamanho do arquivo. Altos perfis usam mais funcionalidades opcionais, oferecem melhor qualidade visual para tamanhos de arquivo menores, levam mais tempo para codificar, e requerem mais poder da CPU para codificar em tempo real.

Para lhe dar uma ideia da variedade dos perfis, o iPhone da Apple suporta o perfil Baseline, a AppleTV suporta os perfis Baseline e Main, e Adobe Flash em um PC suporta os perfis Baseline, Main, e High. YouTube usa agora o H.264 para codificar vídeos de alta definição, tocados a partir do Adobe Flash; YouTube também provê vídeo codificado com H.264 para dispositivos móveis, incluindo iPhone e telefones rodando o sistema operacional móvel Android. Além do H.264 ser um dos codecs de vídeo mandatários pela especificação do Blu-Ray; discos Blu-Ray que usam ele geralmente utilizam no perfil High.

A maioria dos dispositivos, que não são PCs, tocam vídeo em H.264 (incluindo iPhones e reprodutores de Blu-Ray) Most non-PC devices that play H.264 video (including iPhones and standalone Blu-Ray players) na verdade realizam a decodificação em um chip dedicado, uma vez que suas CPUs principais estão longe de ter poder o suficiente para decodificar em tempo real. Atualmente, até mesmo placas gráficas de baixo nível suportam decodificação H.264 no hardware. Existem codificadores H.264 concorrentes, incluindo o open source x264. O padrão H.264 está coberto por patentes; licenciamento é

intermediado pelo [MPEG LA group](#). Vídeo H.264 pode ser incorporado nos mais populares [formatos de contêiner](#), incluindo MP4 (usado primeiramente pela [iTunes Store da Apple](#)) e MKV (usado primeiramente por entusiastas de vídeo não-comerciais).

## THEORA

[Theora](#) evoluiu do [VP3 codec](#) e tem sido subseqüentemente desenvolvido pela [Xiph.org Foundation](#). Theora é um codec livre de royalties e não é onerado por qualquer patente conhecida a não ser a patente original VP3, que foi licenciada livre de royalties. Embora o padrão tenha sido “congelado” desde 2004, o projeto Theora (que inclui um referente codificador e decodificador open source) [apenas lançou a versão 1.0 em novembro de 2008](#) e a [versão 1.1 em setembro de 2009](#).

Vídeos em Theora podem ser incorporados em qualquer formato contêiner, embora seja mais visto em Ogg. A maioria das distribuições Linux suportam Theora fora-da-caixa, e o Mozilla Firefox 3.5 [inclui suporte nativo a vídeos Theora](#) no contêiner Ogg. E por “nativo”, eu digo “disponível em qualquer plataforma sem plugins específicos daquela plataforma.” Você pode também reproduzir vídeos Theora [no Windows](#) ou [no Mac OS X](#) após instalar o software open source decodificador Xiph.org’.

## VP8

[VP8](#) é outro codec de vídeo da On2, mesma companhia que originalmente desenvolveu o VP3 (mais tarde Theora).

Técnicamente, ele produz saída em par com o perfil H.264 High, enquanto mantém uma baixa complexidade de decodificação em par com o perfil H.264 Baseline.

Em 2010, a Google adquiriu a On2 e publicou a especificação do codec de vídeo e uma amostra open source do codificador e decodificador. Como parte disso, a Google também “abriu” todas as patentes que a On2 mantinha sobre o VP8, ao licenciar livre de royalties. (Esse é o melhor que você pode esperar para patentes. Na verdade você não pode “lançar” ou anular elas uma vez emitidas. Para torná-las open source, você as licencia livre de royalties, e então qualquer um pode usar a tecnologia que a patente cobre sem pagar qualquer coisa ou negociar as licenças.) A partir de 19 de maio de 2010, **VP8 se tornou livre de royalties, um codec moderno e não onerado por qualquer patente conhecida**, diferente das patentes que a On2 (agora Google) já licenciou livre de royalties.



## CODECS DE ÁUDIO

A não ser que você ainda esteja parado nos filmes feitos antes de 1927, você irá querer uma trilha de som no seu vídeo. Como codecs de vídeo, *codecs de áudio* são algoritmos nos quais fluxos de áudio são codificados. Como codecs de vídeo, os codecs de áudio são *com perda* e *sem perda*. E como os codecs de vídeo sem perda, áudios sem perda são realmente muito grandes para colocar na web. Então irei me concentrar em codecs de áudio com perda.

Na verdade, é ainda mais estreito que isso, porque eles estão em diferentes categorias de codecs de áudio com perda. Áudio é usado em lugares onde o vídeo não é (telefonia, por exemplo) e existe toda uma categoria de codecs de áudio otimizados para codificar voz. Você não iria utilizar um desses codecs para um CD de música, porque o resultado seria como uma criança de 4 anos de idade cantando em um viva-voz. Mas você *iria* usar eles em um Asterisk PBX, porque a banda é



preciosa, e esses codecs podem comprimir a voz humana em uma fração do tamanho que codecs gerais. Entretanto, graças a falta de suporte em ambos navegadores nativos ou plugins terceiros, codecs de áudio otimizados para voz nunca chegaram realmente na web. Então irei concentrar em *codecs de áudio com propósito geral*.

Eu mencionei mais cedo, quando você “assiste um vídeo,” seu computador está fazendo ao menos três coisas ao mesmo tempo:

1. Interpretando o contêiner do formato
2. Decodificando o fluxo de vídeo
3. Decodificando o fluxo de áudio e enviando o som para os alto-falantes

O *codec de áudio* especifica como #3 — decodificar o fluxo de áudio e torná-lo em ondas digitais que depois seus alto-falantes transformam em som. Assim como os codecs de vídeo, existem todos os tipos de truques para minimizar a quantidade de informação armazenada em um fluxo de áudio. E já que estamos falando sobre codecs de áudio *com perda*, a informação está sendo perdida durante a gravação → codificação → decodificação → ciclo de vida da escuta. Diferentes codecs de áudio jogam fora diferentes coisas, mas eles tem o mesmo objetivo: enganar seus ouvidos para não notar as partes que estão faltando.

Um conceito que o áudio tem que o vídeo não tem são os *canais*. Nós estamos enviando som para os alto-falantes, certo? Bom, quantos alto-falantes você tem? Se você está sentado em um computador, você deve ter apenas dois: um na esquerda e um na direita. Meu desktop tem três: esquerda, direita e mais um no chão. Chamado “surround sound” sistemas podem ter seis ou mais alto-falantes, estrategicamente colocados pelo lugar. Cada alto-falante alimenta um *canal* em particular da gravação original. A teoria é que você pode sentar no meio dos seis alto-falantes, literalmente cercado por seis diferentes canais do som, e seu cérebro sintetiza eles e parece que você está no meio da ação. Isso funciona? Uma indústria multi-

bilionária parece acreditar que sim.

Codecs de áudio com propósito geral podem lidar com dois canais de som. Durante a gravação, o som é dividido no canal da esquerda e da direita; durante a codificação, ambos canais armazenam o mesmo fluxo de áudio; durante a decodificação, ambos canais estão decodificados e cada um é enviado para o alto-falante apropriado. Alguns codecs de áudio podem lidar com mais de dois canais, e irão controlar qual canal é qual e então seu reprodutor de áudio pode enviar para o som da direita para o alto-falante da direita.

Existem *muitos* codecs de áudio. Eu disse que haviam muitos codecs de vídeo? Esqueça isso. Existem dezenas e dezenas de codecs de áudio, mas na web, existem apenas que você deveria conhecer mais sobre: MP3, AAC, e Vorbis.

## MPEG-1 AUDIO LAYER 3

MPEG-1 Audio Layer 3 é coloquialmente conhecido como “MP3.” Se você nunca ouviu falar de MP3s, eu não sei o que fazer com você. Walmart vende reprodutores de música portáteis e os chama de “MP3 players.” *Walmart*. Em todo caso...

MP3s podem conter **até dois canais** de som. Eles podem ser codificados em diferentes *taxas de bits*: 64 kbps, 128 kbps, 192 kbps, e uma variedade de outros de 32 à 320. Altas taxas de bits significam tamanhos de arquivos maiores e melhor qualidade no áudio, embora a relação da taxa da qualidade com a taxa de bits não seja linear. (128 kbps soa duas vezes melhor que 64 kbps, mas 256 kbps não soa o dobro melhor que 128 kbps.) Além disso, o formato MP3 permite *codificação variável na taxa de bits*, o que significa que algumas partes do fluxo de codificação são mais comprimidas que outras. Por exemplo, silêncio entre as notas podem ser codificadas em uma baixa taxa de bits, então a taxa de bits pode aumentar um momento mais tarde quando múltiplos instrumentos começam a tocar um acorde complexo. MP3s também podem ser codificados em uma

taxa de bits constante, que, sem surpresas, é chamado de *codificação a taxas de bits constantes*.

O padrão MP3 não define exatamente como codificar MP3s (embora defina exatamente como devemos decodificá-los); diferentes codificadores usam diferentes modelos psicoacústicos que produzem resultados descontroladamente diferentes, mas são todos decodificados pelos mesmos players. O projeto open source LAME é o melhor codificador gratuito, e sem dúvida o melhor codificador, ponto final, para todos menos para baixas taxas de bits.

O formato MP3 (padronizado em 1991) é **coberto de patentes**, que explicam porque Linux não pode tocar arquivos MP3 fora da caixa. Praticamente todos os players de música portáteis suportam arquivos MP3s, e os fluxos de áudio MP3 podem ser embutidos em qualquer container de vídeo. Adobe Flash pode ambos, arquivos MP3 e fluxo de áudio MP3 com um container de vídeo MP4.

## CODIFICAÇÃO DE ÁUDIO AVANÇADA

Codificação de Áudio Avançada é efetivamente conhecida como "AAC" (Advanced Audio Coding). Padronizada em 1997, deu uma "levantada" proeminente quando a Apple escolheu o MP3 como formato padrão para a iTunes Store. Originalmente, todos os arquivos "AAC" "comprados" pela loja do iTunes eram criptografados com um esquema DRM proprietário da Apple, chamado FairPlay. Músicas selecionadas na loja iTunes agora estão disponíveis como arquivos desprotegidos AAC, que a Apple chama de "iTunes Plus" pois soa muito melhor do que chamar todo o resto de "iTunes Minus." O formato AAC é **coberto de patentes**; taxas de licença são disponibilizadas online.

AAC foi desenvolvido para promover uma qualidade de som melhor que MP3 na mesma *taxa de bits*, e pode codificar áudio em qualquer taxa de bits. (MP3 é limitado a um número fixo de taxa de bits, com um limite superior de 320 kbps.) AAC

pode codificar até 48 canais de som, embora na prática ninguém faça isso. O formato AAC também difere do MP3 na definição de múltiplos *perfis*, na maioria das vezes como [H.264](#), e pelas mesmas razões. O perfil de “baixa complexidade” é desenvolvido para ser reproduzido em tempo real em dispositivos com poder de CPU limitado, enquanto perfis altos oferecem melhor qualidade na mesma taxa de bits ao preço de lentidão na codificação ou decodificação.

Todos os atuais produtos da Apple, incluindo iPods, AppleTV, e QuickTime suportam alguns perfis de AAC tanto em arquivos de áudio e em fluxos de áudio dentro um container de vídeo MP4. Adobe Flash suporta todos os perfis de AAC em MP4, como fazem os reprodutores de vídeo MPlayer e VLC. Para codificação, a biblioteca FAAC é a opção open source; o suporte é uma opção em tempo de compilação em mencoder e ffmpeg.

## VORBIS

[Vorbis](#) também chamado de “Ogg Vorbis,” embora esteja tecnicamente incorreto. (“Ogg” e apenas um [formato de container](#), e fluxos de áudio Vorbis podem ser embutido em outros containers.) Vorbis não é coberto por nenhuma patente conhecida e portanto é suportado fora da caixa pela maioria das distribuições Linux e dispositivos portáteis rodando o open source [Rockbox](#) firmware. Mozilla Firefox 3.5 suporta arquivos de áudio Vorbis em um container Ogg, ou Ogg vídeos com uma faixa de áudio Vorbis. [Android](#) celulares também podem reproduzir arquivos de audio Vorbis. streams de audio Vorbis geralmente sao embutidas em um Ogg ou em WebM contêiner, mas ele também podem ser [embutidos em um MP4](#) ou [MKV](#) container (ou, com algum hacking, [em AVI](#)). Vorbis suporta um numero arbitrário de canais de som.

Existem codificadores e decodificadores Vorbis open source, incluindo [OggConvert](#) (codificador), [ffmpeg](#) (decodificador), [aoTuV](#) (codificador), e [libvorbis](#) (decodificador). Também existem [Componentes QuickTime para Mac OS X](#) e [Filtros DirectShow para Windows](#).



# O QUE FUNCIONA NA WEB

Se os seus olhos não estão vidrados ainda, você está melhor que a maioria. Como você pode dizer, vídeo (e áudio) é um assunto complicado e esta foi a versão abreviada! Eu tenho certeza que você está se perguntando como tudo isso se relaciona com HTML5. Bem, HTML5 inclui um elemento `<video>` para embutir vídeos em uma página web. Não há restrição no codec do vídeo, codec do áudio ou no formato de container que você pode usar para seu vídeo. Um elemento `<video>` pode ter um link para múltiplos arquivos de vídeos, e o navegador escolhe qual o primeiro vídeo que irá reproduzir. Cabe a você saber qual navegador suporta qual containers e codecs.

Como está escrito, está é a paisagem do vídeo HTML5:

- Mozilla Firefox (3.5 e superior) suporta vídeo Theora e áudio Vorbis em container Ogg. Firefox 4 também suporta WebM.
- Opera (10.5 e superior) suporta vídeo Theora e áudio Vorbis em um container Ogg. Opera 10.60 também suporta WebM.
- Google Chrome (3.0 e superior) suporta vídeo Theora e áudio Vorbis em um container Ogg. Google Chrome 6.0 também suporta WebM.
- Safari no Macs e PCs Windows (3.0 e superior) irão suportar tudo que o QuickTime suporta. Em teoria, você não pode requerer que seus usuários instalem plugins QuickTime de terceiros. Na prática, alguns usuários estão fazendo isto. Então você é deixado com os formatos que o QuickTime suporta "fora da caixa." Esta é uma lista longa, mas não inclui

WebM, Theora, Vorbis, ou o container Ogg. Contudo, QuickTime *fornece* suporte para vídeo H.264 (perfil principal) e áudio AAC em um container MP4.

- Celulares como iPhone da Apple e celulares com Android da Google suportam vídeo H.264 (perfil de base) e áudio AAC (perfis de "baixa complexidade") em um container de MP4.
- Adobe Flash (9.0.60.184 e superior) suporta vídeos H.264 (todos os perfis) e áudio AAC (todos os perfis) em um container MP4.
- Internet Explorer 9 suporta todos os perfis de vídeos H.264 ou AAC ou áudio MP3 em um container MP4. Também irá reproduzir vídeo WebM se você instalar algum codec de terceiro, que por padrão não é instalado por nenhuma versão do Windows. IE9 não suporta outros codecs de terceiros (ao contrário do Safari, que vai reproduzir tudo que o QuickTime pode reproduzir).
- Internet Explorer 8 não tem suporte de vídeo HTML5 em geral, mas praticamente todos os usuários de Internet Explorer terão o plugin Adobe Flash. Mais adiante neste capítulo, Eu vou mostrar para você como usar vídeo HTML5 e gerar um fallback em Flash.

Isto deve ser mais fácil de digerir em forma de tabela.

### SUPORTE DE CODECS DE VÍDEO NOS NAVEGADORES ATUALMENTE

CODECS/CONTAINER	IE	FIREFOX	SAFARI	CHROME	OPERA	IPHONE	ANDROID
Theora+Vorbis+Ogg	.	3.5+	†	5.0+	10.5+	.	.
H.264+AAC+MP4	.	.	3.0+	5.0–?‡	.	3.0+	2.0+
WebM	.	.	†	6.0+	10.6+	.	.

† Safari irá reproduzir tudo que o QuickTime reproduz. QuickTime vem pre-instalado com suporte H.264/AAC/MP4. Existem plugins de terceiros instaláveis que provêm suporte para Theora e WebM, mas cada usuário precisa instalar esses plugins antes do Safari reconhecer os formatos dos vídeos.

‡ Google Chrome irá [perder o suporte a H.264](#) em breve. [Ler sobre o por que.](#)

Daqui há um ano, a paisagem vai parecer significativamente diferente com WebM implementado em múltiplos navegadores, estes navegadores habilitarão versões não-experimentais do WebM, e atualizações dos usuários para as novas versões.

### SUPORTE DE CODECS DE VÍDEO NOS NAVEGADORES FUTURAMENTE

CODECS/CONTAINER	IE	FIREFOX	SAFARI	CHROME	OPERA	IPHONE	ANDROID
Theora+Vorbis+Ogg	.	3.5+	†	5.0+	10.5+	.	.
H.264+AAC+MP4	9.0+	.	3.0+	.	.	3.0+	2.0+
WebM	9.0+*	4.0+	†	6.0+	10.6+	.	2.3‡

\* Internet Explorer 9 vai suportar apenas WebM "quando o usuário tiver instalado um codec VP8", o que implica que a Microsoft não enviará o codec deles mesmos.

\* Safari irá reproduzir tudo que o QuickTime pode reproduzir, mas QuickTime vem apenas com suporte H.264/AAC/MP4 pré-instalado.

\* Embora o Android 2.3 suporte WebM, ainda não existem decodificadores de hardwares, dessa forma a vida da bateria é uma preocupação.

E agora para o soco de nocaute:

### PROFESSOR MARCAÇÃO DIZ

Não existem únicas combinações de contêineres e codecs que funcionem em todos os navegadores HTML5.

E isso não vai mudar em um futuro tão próximo.

Para fazer seu vídeo assistível em todos esses dispositivos e plataformas, você precisará



codificar seu vídeo mais de uma vez.



Para máxima compatibilidade, aqui está o fluxo de trabalho com o que seu vídeo deve parecer.

1. Fazer uma versão que usa WebM (VP8 + Vorbis).
2. Fazer outra versão que use linhas de bases de vídeo H.264 e audio AAC "baixa complexidade" em um contêiner MP4.
3. Fazer outra versão que use vídeo Theora e audio Vorbis em um contêiner Ogg.
4. Linkar todos os três arquivos de vídeo apartir de um único elemento <video>, e um "fallback" para um reprodutor de vídeo flash.



# PROBLEMAS DE LICENCIAMENTO COM VÍDEO

## H.264

Antes de nós continuarmos, eu preciso salientar que há um custo em codificar seu vídeo duas vezes. Bem, existe um custo óbvio, que você tem que codificar seu vídeo duas vezes, o que consome mais computador e mais tempo do que se fosse feito



apenas uma vez. Mas há outro custo real associado com vídeo H.264: custos de licenciamento.

Se lembram quando eu expliquei [vídeo H.264](#), e eu mencionei que o codec de vídeo tem patentes-embutidas e o licenciamento foi quebrado pelo consórcio MPEG LA. Isso acaba sendo importante. Para entender porque isso é importante, eu te direciono para [O labirinto do licenciamento H.264](#):

MPEG LA Divide a licença de portfólio H.264 em duas sub-licenças: uma para manufaturas de codificadores e decodificadores e a outra para distribuidores de conteúdo. ...

A sub-licença do lado dos distribuidores fica ainda mais dividida em quatro sub-categorias chaves, duas delas (assinaturas e aquisição título-por-título ou uso remunerado) são ligados ao fato de que o usuário final paga diretamente pelo serviço de vídeos, e dois deles (televisão "gratuita" e broadcast de internet) são ligados a remuneração por outras fontes que não seja o espectador final. ...

A taxa de licenciamento para televisão "gratuita" é baseada em uma das duas opções de royalties. A primeiro é um pagamento único de \$2,500 por transmissão codificada AVC, que cobre um codificador AVC "usado por ou a favor da licença de transmissão de vídeo AVC para o usuário final" que é quem irá decodificar e visualizá-lo. Se você está se perguntando se isso é um encargo duplo, a resposta é sim: A taxa da licença já foi cobrada do codificador manufaturado, e o broadcast vai em turnos pagar uma das duas opções de royalties.

A segunda taxa de licenciamento é anual. [A] taxa anual de broadcast é dividida pelo tamanho da audiência:

- \$2,500 por ano por mercado de broadcasts de 100,000–499,999 famílias de televisão
- \$5,000 por ano por mercado de broadcasts de 500,000–999,999 famílias de televisão

- \$10,000 por ano por mercado de broadcasts de 1,000,000 ou mais famílias de televisão

Com toda a questão da televisão "livre" por que alguém envolvido em entregas não-broadcast se importaria? Como eu mencionei antes, as taxas de participação são aplicáveis para cada entrega de conteúdo. Depois de definir que televisão "livre" significa mais do que apenas "over-the-air", MPEG LA passou a definir as taxas de licenciamento para transferencia na internet via "vídeo AVC que é entregue em todo mundo para um usuário final para que o usuário final não pague pelo direito de receber ou ver". Em outras palavras, qualquer transmissão pública, seja "over-the-air", cabo, satélite, ou internet estão sujeitas a taxas de participação. ...

As taxas são potencialmente mais íngrimes para transmissões pela internet, talvez sabendo que a entrega na internet irá crescer muito mais rápido do que OTA ou televisão "livre" via cabo ou satélite. Adicionando a televisão "livre" as taxas do mercado da transmissão juntas com a taxa adicional. MPEG LA garantirá um indulto durante o tempo da primeira licença, que acaba em 31 de Dez de 2010, e note que "após a primeira licença o royalties não será mais do que equivalente econômico dos royalties pagos durante o mesmo tempo de televisão livre."

A última parte - Sobre a estrutura das taxas para transmissão pela internet - já foi alterada. A MPEG-LA recentemente anunciou que o streaming de internet não seria cobrado. Que *não* significa que H.264 é livre de royalties para todos os usuários. Em particular, codificadores ( Como o que processa o upload de vídeo do YouTube ) e codificadores (como o que esta incluso no Microsoft Internet Explorer 9) ainda são assuntos de taxas de licenciamento. Veja Livre como em uma tema de fumaça para maiores informações.



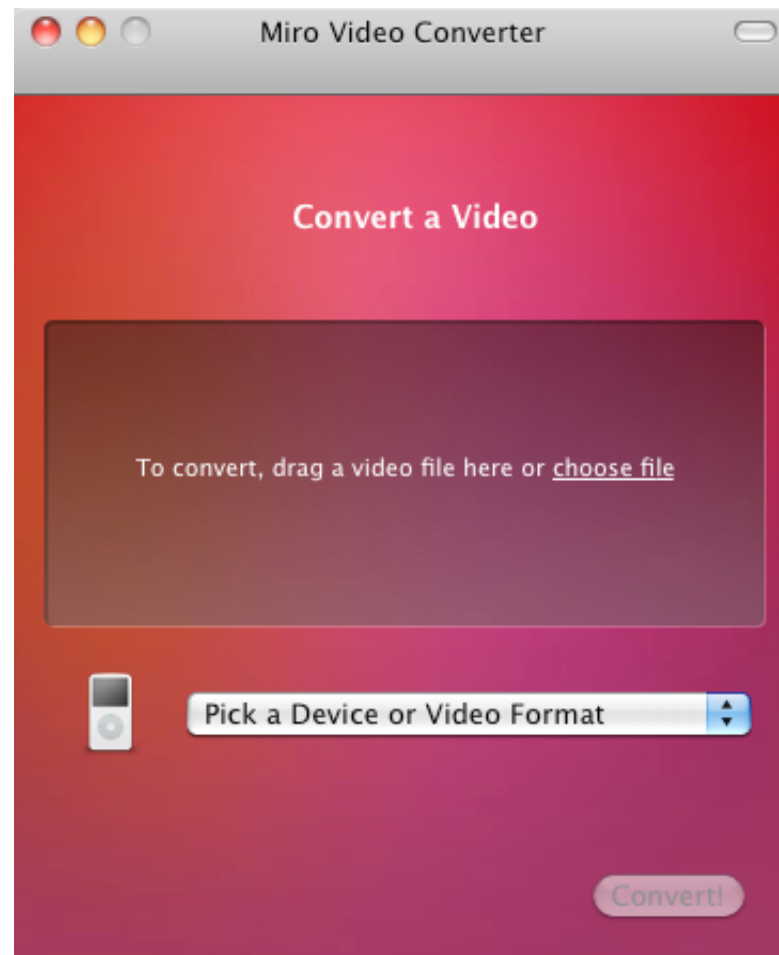
# CODIFICANDO VÍDEO COM MIRO VIDEO CONVERTER

Existem muitas ferramentas para codificar vídeo, e há muitas opções de codificação de vídeo que afetam sua qualidade. Se você não deseja tomar o tempo para entender nada sobre codificação de vídeo, esta seção é para você.

Miro Video Converter é programa de código aberto para codificação de vídeo em múltiplos formatos com licença GPL. [Faça aqui o download para Mac OS X ou Windows](#). Ele suporta todos os formatos de saída mencionados neste capítulo. Não oferece opções além de escolher um arquivo de vídeo e escolher um formato de saída. Pode receber virtualmente qualquer arquivo de vídeo como entrada, incluindo vídeo DV produzidos por filmadoras. Produz uma saída com qualidade razoável com a maioria dos vídeos. Devido a sua falta de opções, se você estiver descontente com a saída, você não tem mais recursos além de tentar outro programa.

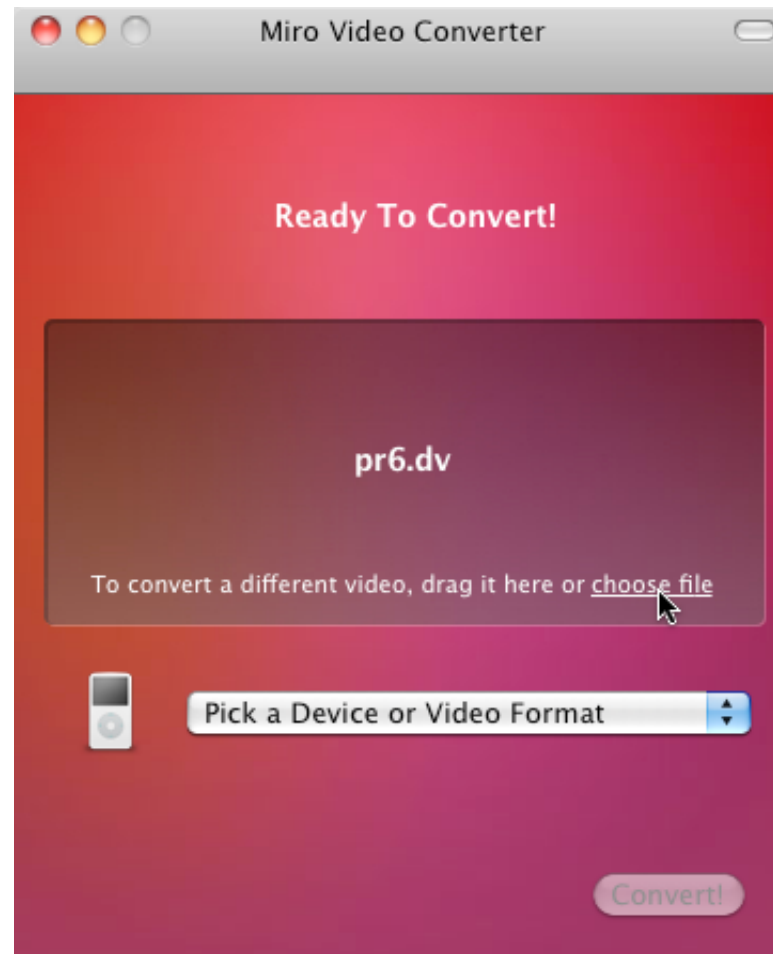
Para começar, Apenas inicie a aplicação Miro Video Converter.

*Tela principal do Miro Video Converter* ↷



Clique em "Escolher arquivo" e selecione o vídeo que você quer codificar.

*"Escolher arquivo"* ↷

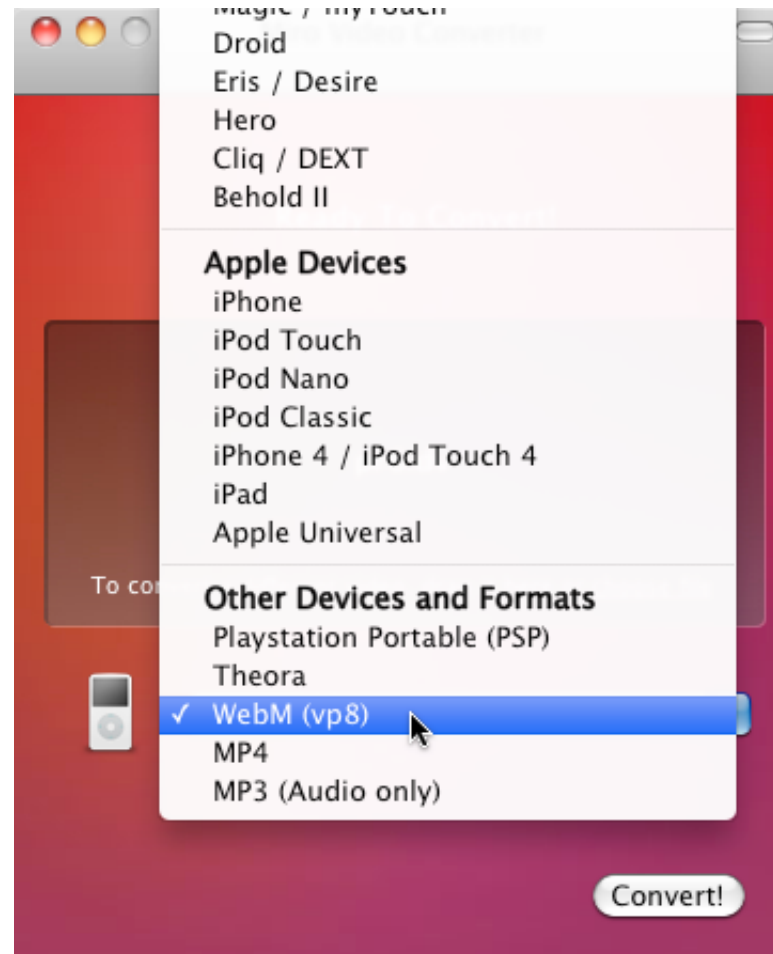


O menu dropdown "Escolha um dispositivo ou um formato de vídeo" lista uma variedade de dispositivos e formatos. Para o propósito deste capítulo, nós estamos interessados em três deles.

1. *WebM* (*vp8*) é video WebM (video VP8 e audio Vorbis em um contêiner WebM).
2. *Theora* é vídeo Theora e audio Vorbis em um contêiner Ogg.
3. *iPhone* é Perfil de Linha de base de video H.264 e AAC audio de baixa-complexidade em um contêiner MP4.

Primeiro selecione "WebM".

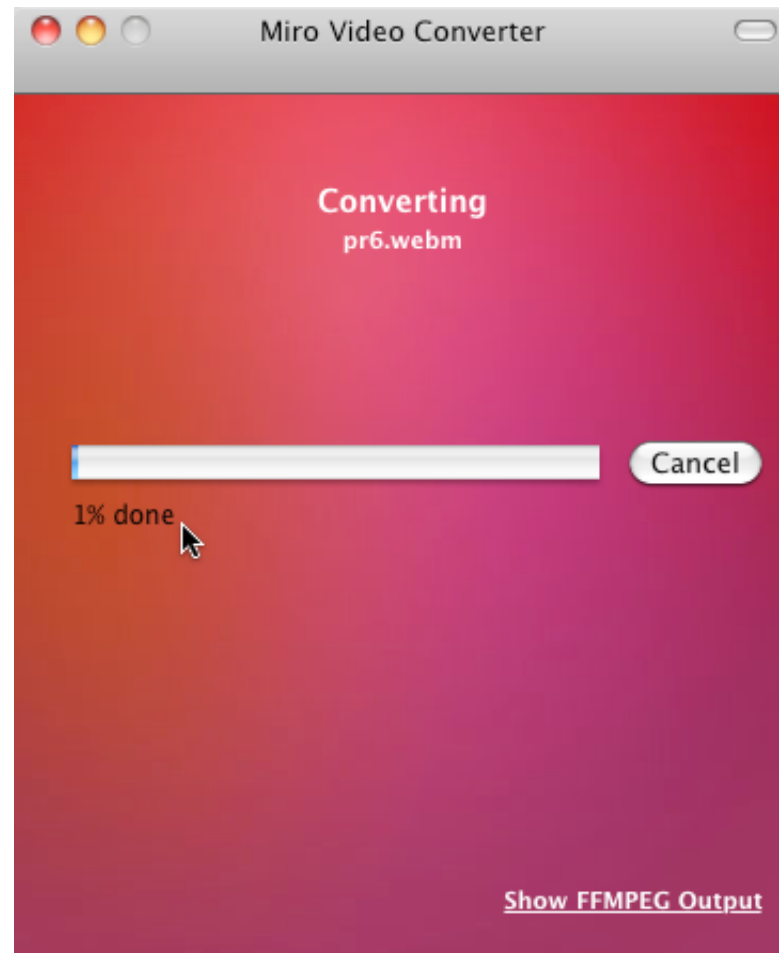
*Escolha a saída WebM~*



Click no botão "Converter" e o conversor Miro Video vai imediatamente começar a codificar seu vídeo. O arquivo de saída

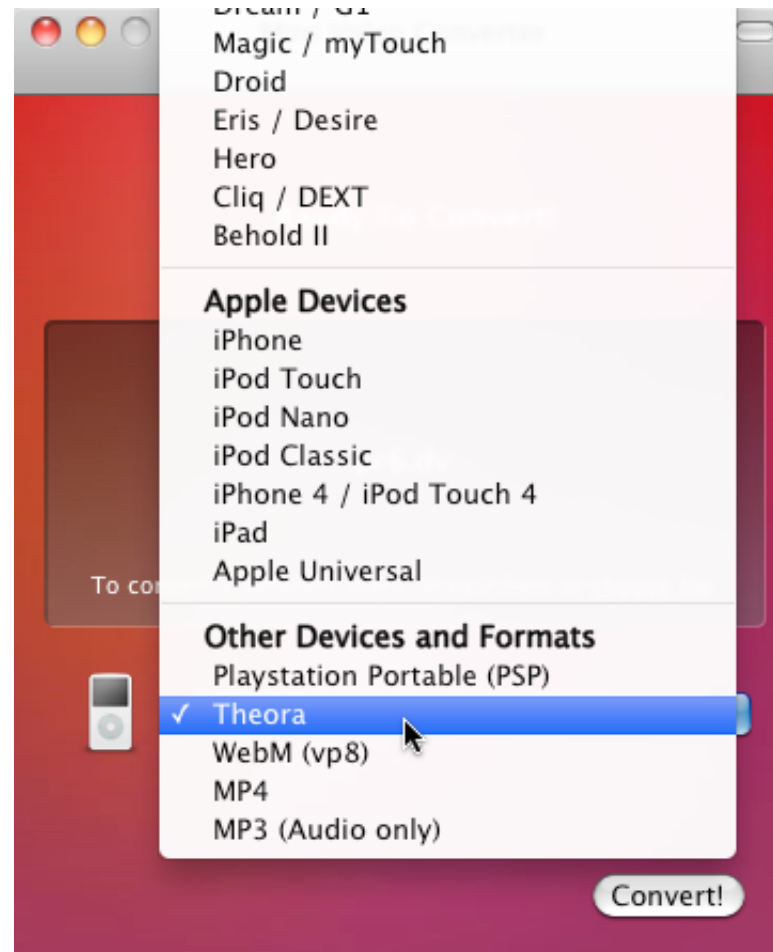
será nomeado ARQUIVOFONTE.webm e vai ser salvo no mesmo diretório do vídeo de origem.

*Você ficará olhando para esta tela  
por um bom tempo ↷*



Uma vez que a codificação for completada, você será mandado de volta para a tela principal. Desta vez, selecione "Theora" na lista de dispositivos e formatos.

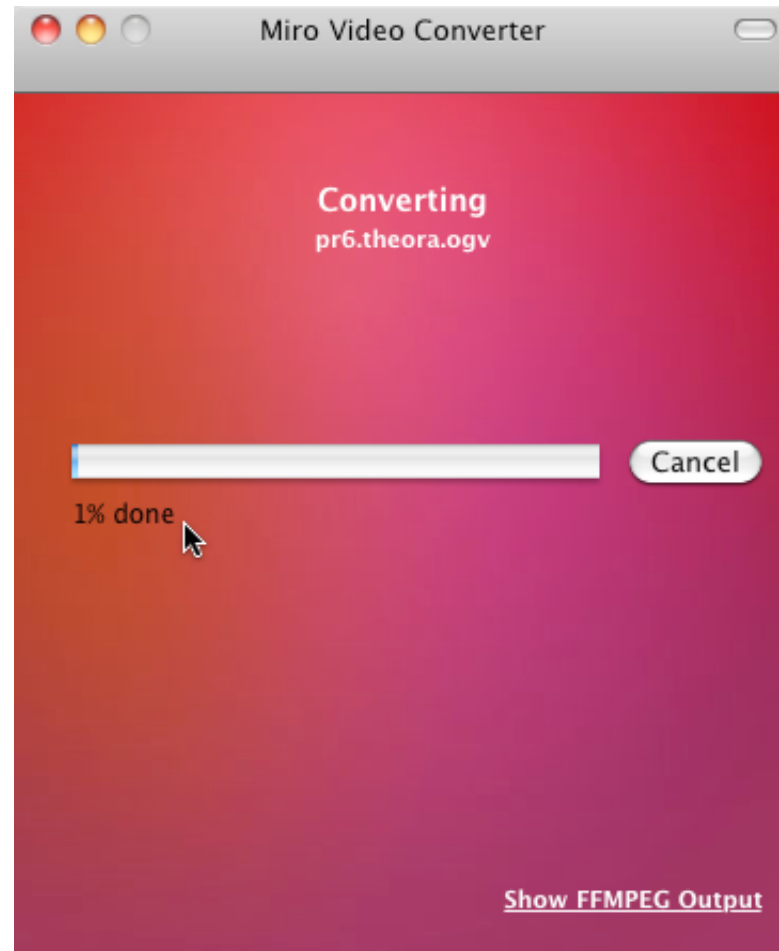
*Hora do Theora* ~





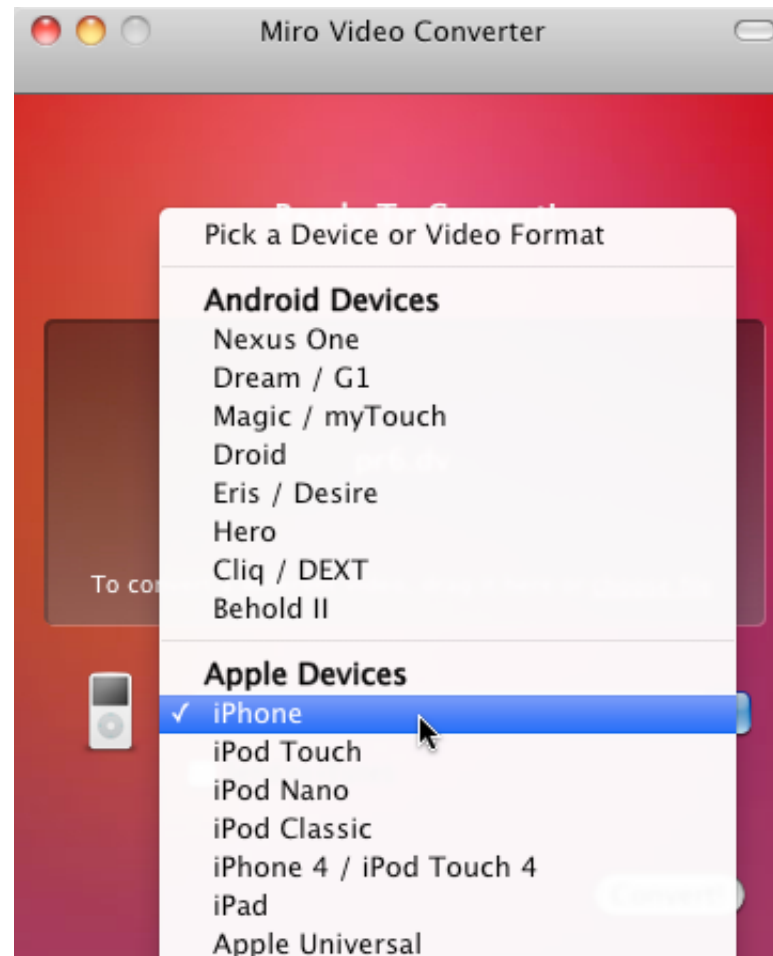
É isto; aperte o botão de "Converter" novamente para codificar seu vídeo Theora. O vídeo vai ser nomeado ARQUIVOFONTE.theora.ogv e será salvo no mesmo diretório do arquivo de origem.

*Hora de tomar um cafézinho ☺*



Finalmente, codifique seu vídeo H.264 compatível com iPhone selecionando "iPhone" na lista de dispositivos e formatos.

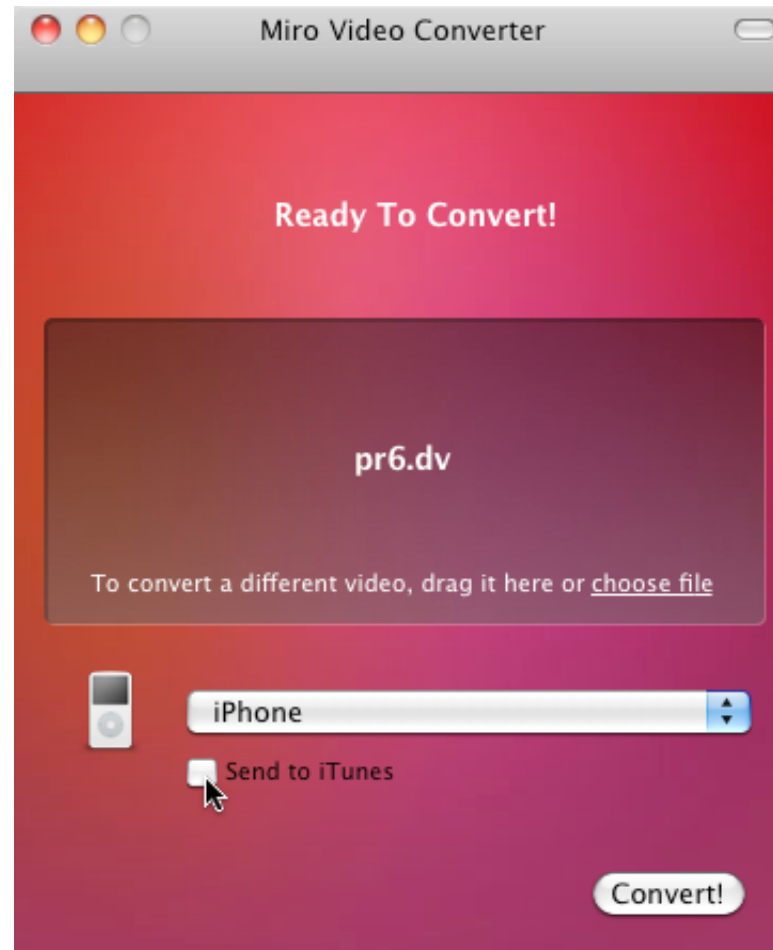
*iPhone, não iPhone 4* ↷



Para vídeo compatível com iPhone, o Conversor Miro Video dará a você uma opção de enviar o arquivo codificado para sua

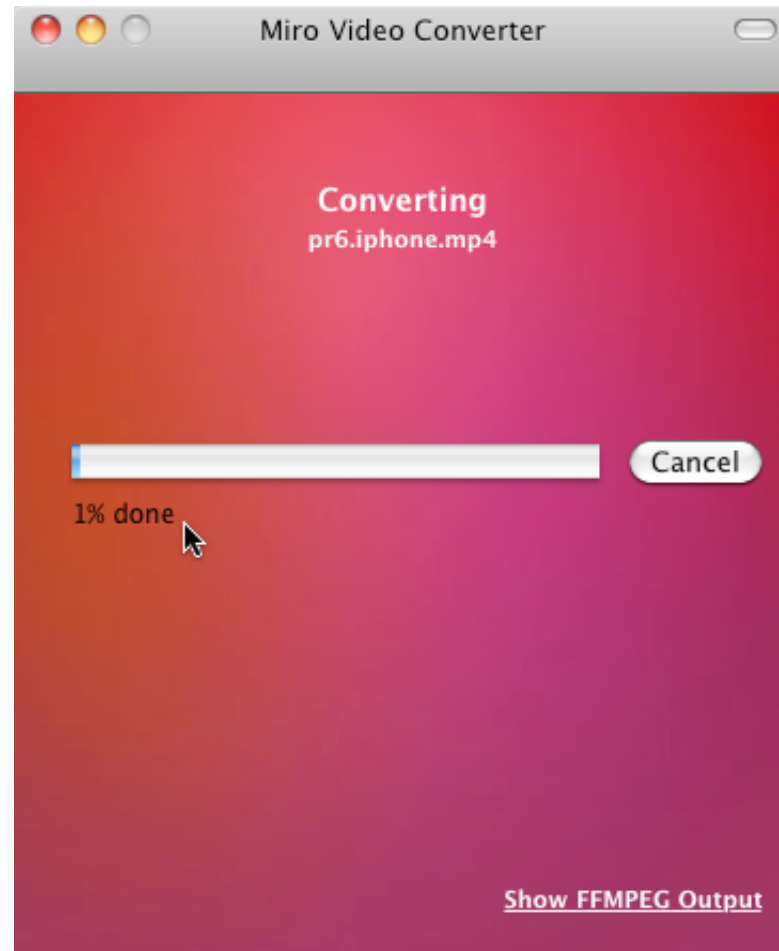
biblioteca iTunes. Eu não tenho uma opinião formada se você gostaria de fazer isto, mas isto não é necessário para publicar vídeo na web.

*Não envie para o iTunes~*



Aperte o botão mágico "Converter" e aguarde. O arquivo codificado será nomeado ARQUIVOFONTE.iphone.mp4 e vai ser salvo no mesmo diretório do arquivo de origem.

*Faça yoga ou algo parecido ↷*



Você deverá ter três arquivos de vídeo ao lado do seu arquivo de vídeo fonte. Se você estiver satisfeito com a qualidade do vídeo, avance para [Enfim, a Marcação](#) para ver como montar eles em um único elemento <vídeo> que funcione cross-browsers. Se você gostaria de aprender mais sobre outras ferramentas ou codificação de vídeo, leia.



## CODIFICANDO VÍDEO OGG COM FIREFOGG

(Nesta seção, eu vou usar "Video Ogg" como abreviação para "Vídeo Theora e Áudio Vorbis em um contêiner Ogg". Esta é a combinação de codecs+contêiners que funcionam nativamente no Mozilla Firefox e Google Chrome.)

Firefogg é uma extensão do Firefox open source, com licença GPL para codificar vídeo Ogg. Para usar isto, você vai precisar instalar [Mozilla Firefox](#) 3.5 ou mais recente, e visitar [firefogg.org](#).

*Página inicial do Firefogg* ↪



Clique "Instalar Firefogg." Firefox irá perguntar se você realmente quer autorizar o site a instalar uma extensão. Clique

"Permitir" para continuar.

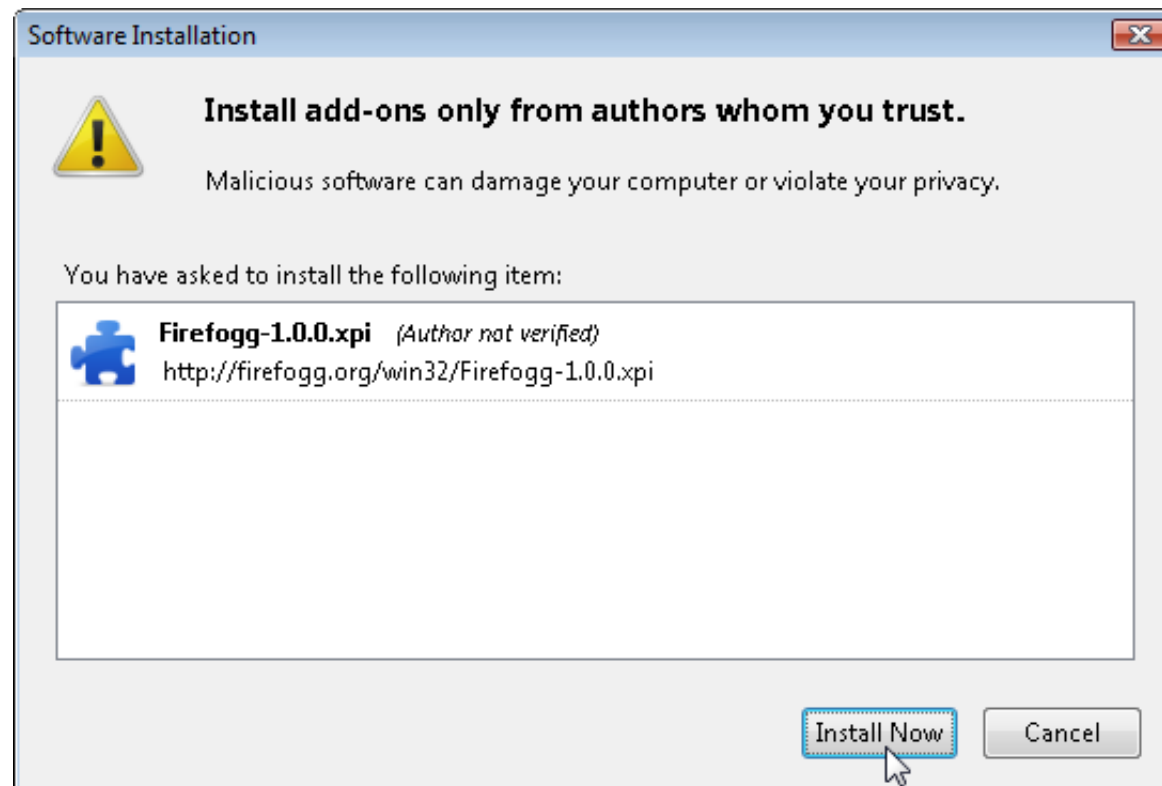
↪ *Permitir instalação do Firefogg*



Firefox vai apresentar a janela de instalação do software padrão. Clique "Instalar" para continuar.

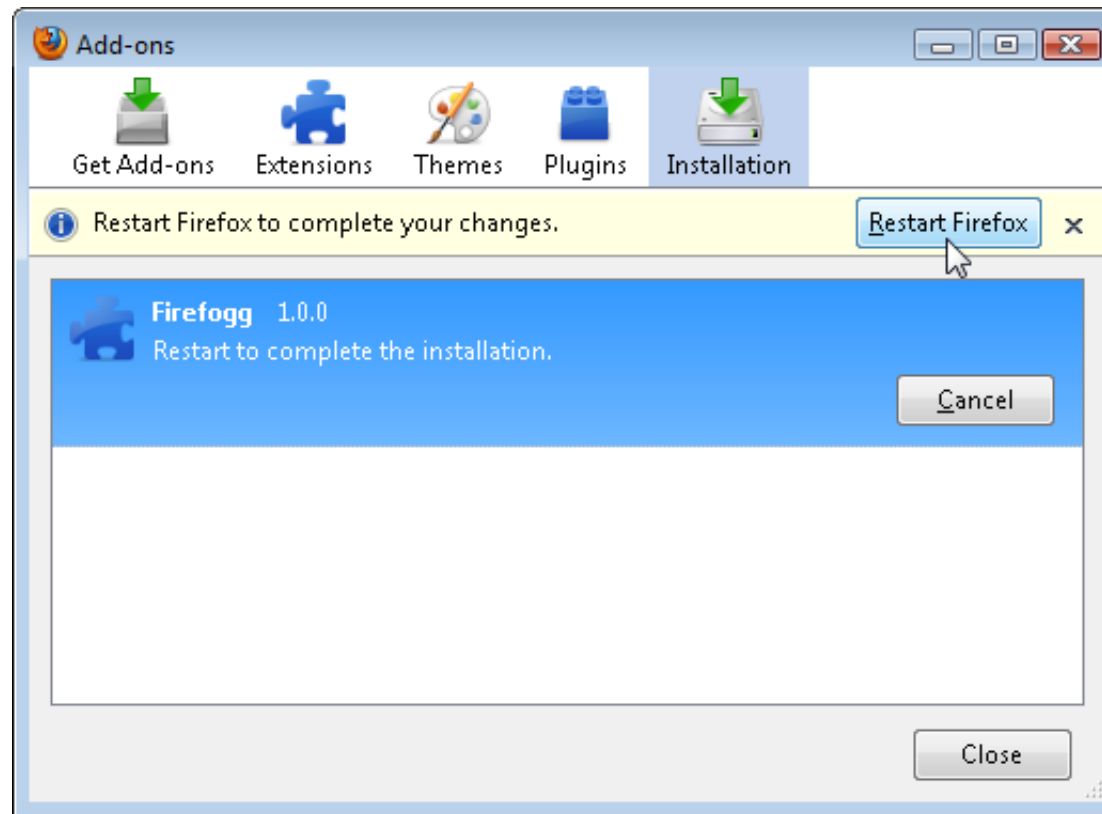


## *Instalar Firefogg ↷*



Clique "Reiniciar Firefox" para completar a instalação.

## *↷ Reiniciar Firefox*



Depois de reiniciar o Firefox, firefogg.org vai confirmar que o Firefogg foi instalado com sucesso.

*Instalação completada com sucesso* ↪



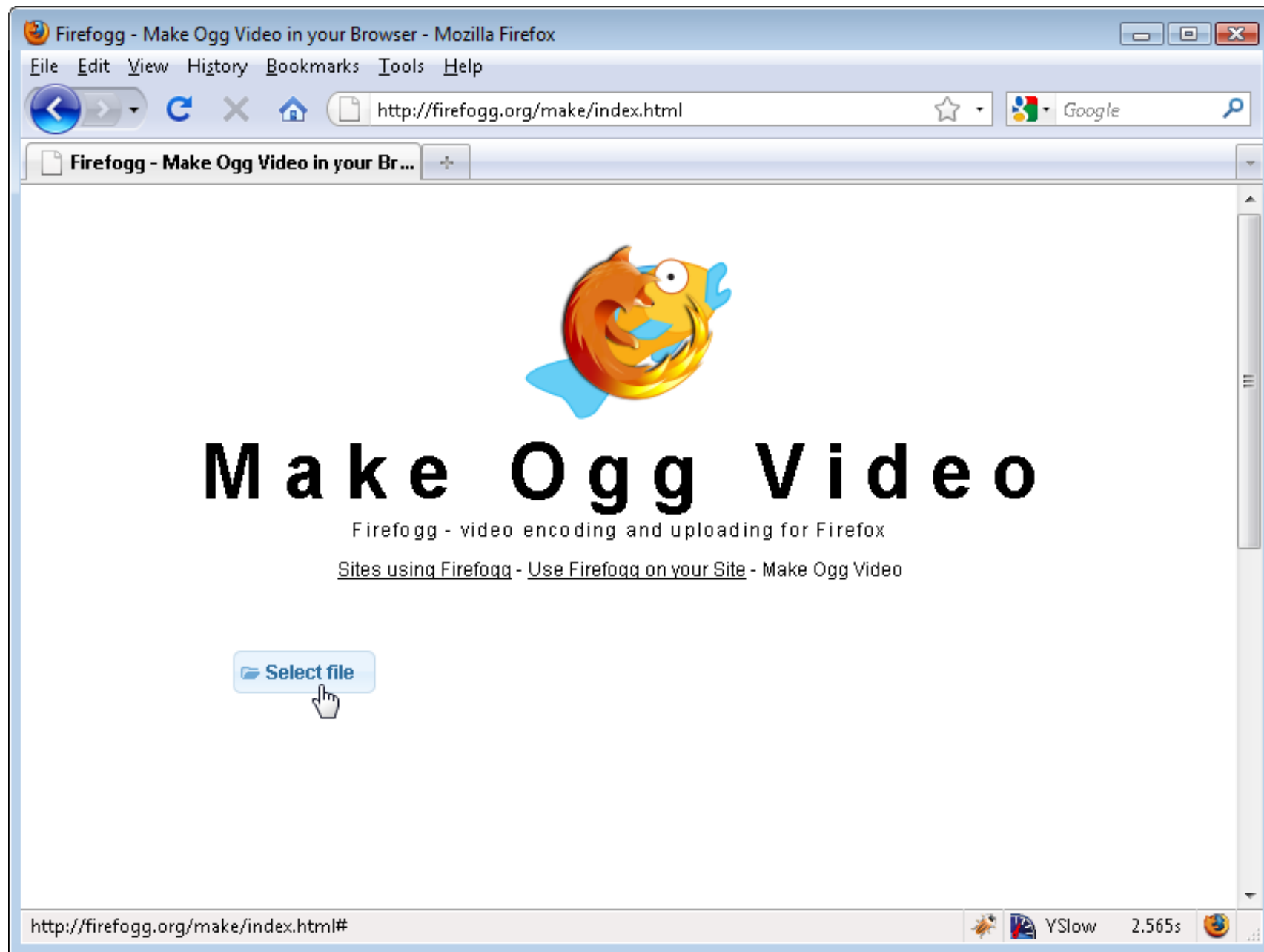
Clique "Criar vídeo Ogg" para iniciar o processo de codificação.

↪ *Vamos criar um vídeo!*



Clique "Selecione o arquivo" para selecionar seu vídeo de origem.

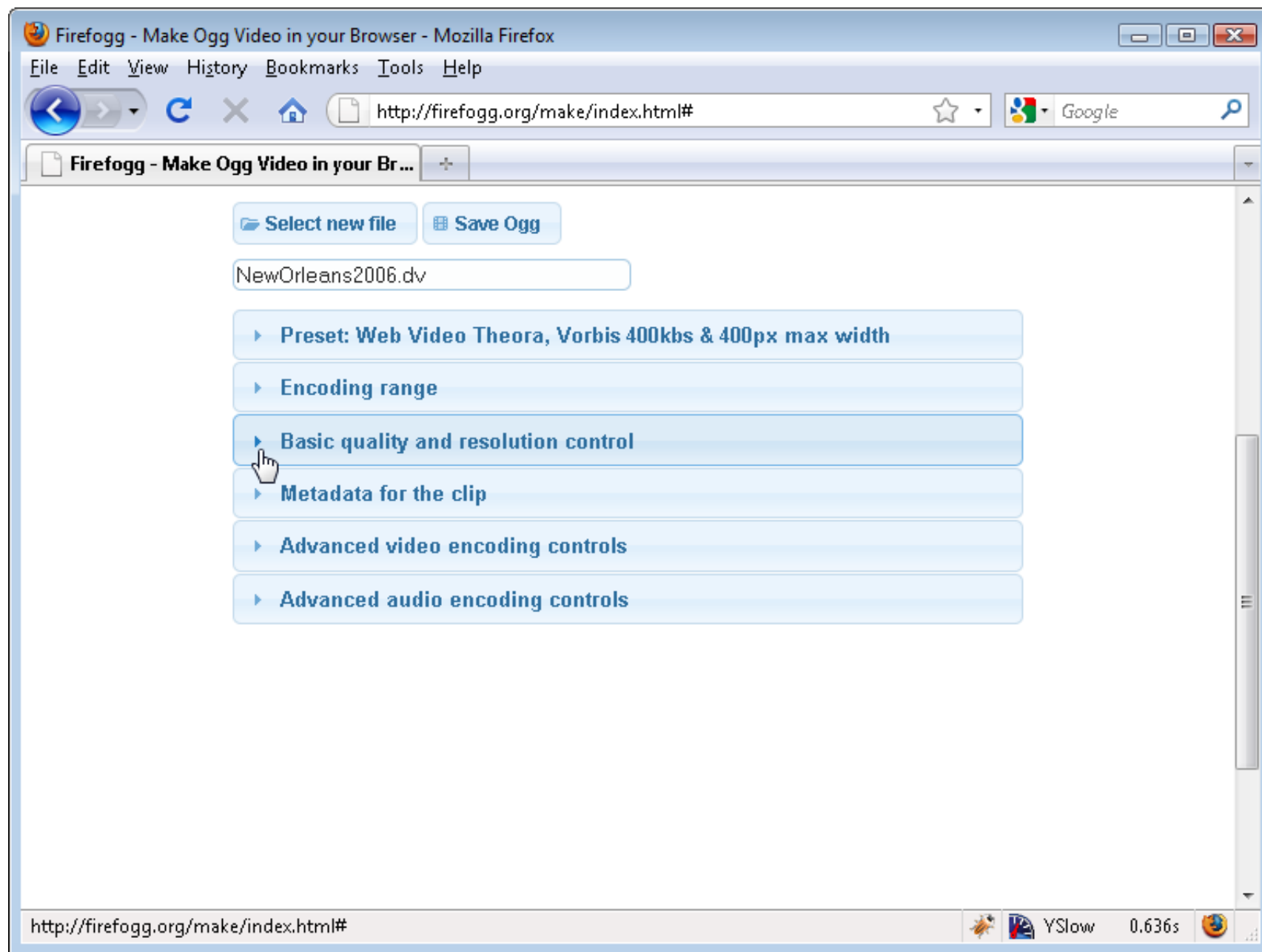
*Selecione seu arquivo de vídeo ↷*



Firefogg tem seis "abas":

1. Predefinições. A predefinição padrão é "vídeo web," o que é bom para nosso propósito.
2. Limite de Codificação. Codificar vídeo pode levar um bom tempo. Quando você estiver começando, talvez você queira codificar apenas partes do seu vídeo (como, os primeiros 30 segundos) até que você encontre uma combinação de configurações que você goste.
3. Qualidade básica e controle de resolução. Esta é onde estão a maioria das opções importantes.
4. Metadata. Eu não vou cobrir isto aqui, mas você pode adicionar metadata ao seu codificador de vídeo como título e autor. Provavelmente você tem adicionado metadata a sua coleção de música com iTunes ou algum outro gerenciador de musicas. É a mesma idéia.
5. Controle avançado de codificação de vídeo. Não mecha com este a menos que saiba o que está fazendo. (Firefogg oferece ajuda interativa na maioria destas opções. Clique no simbolo "i" próximo a cada opção para aprender mais a respeito.)
6. Controle avançado de codificação de áudio. Novamente, Não mecha com este a menos que saiba o que está fazendo.

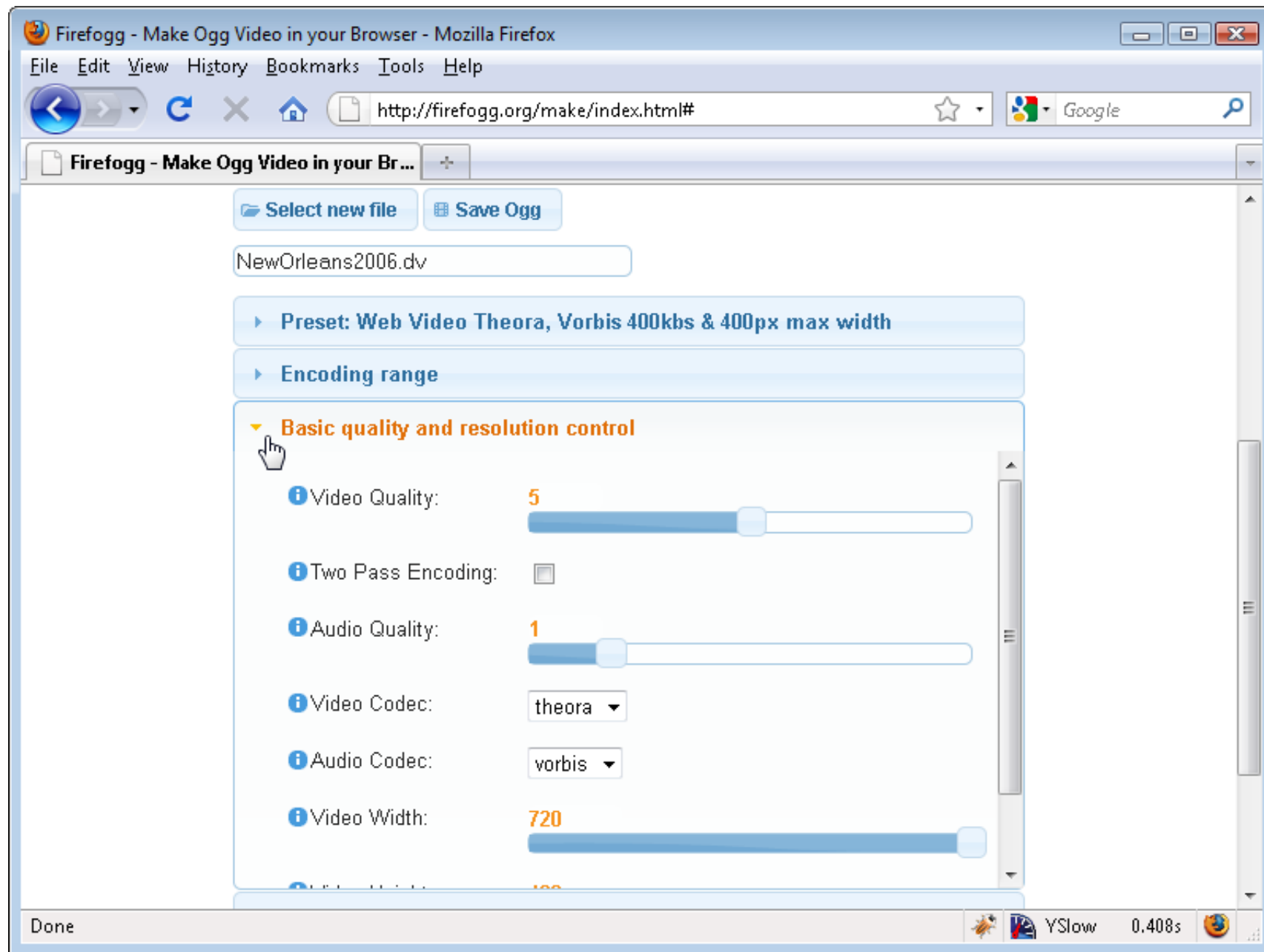




A única opção que eu vou cobrir estão na tabela "Qualidade básica e controle de resolução". Ela contém todas as opções

importantes:

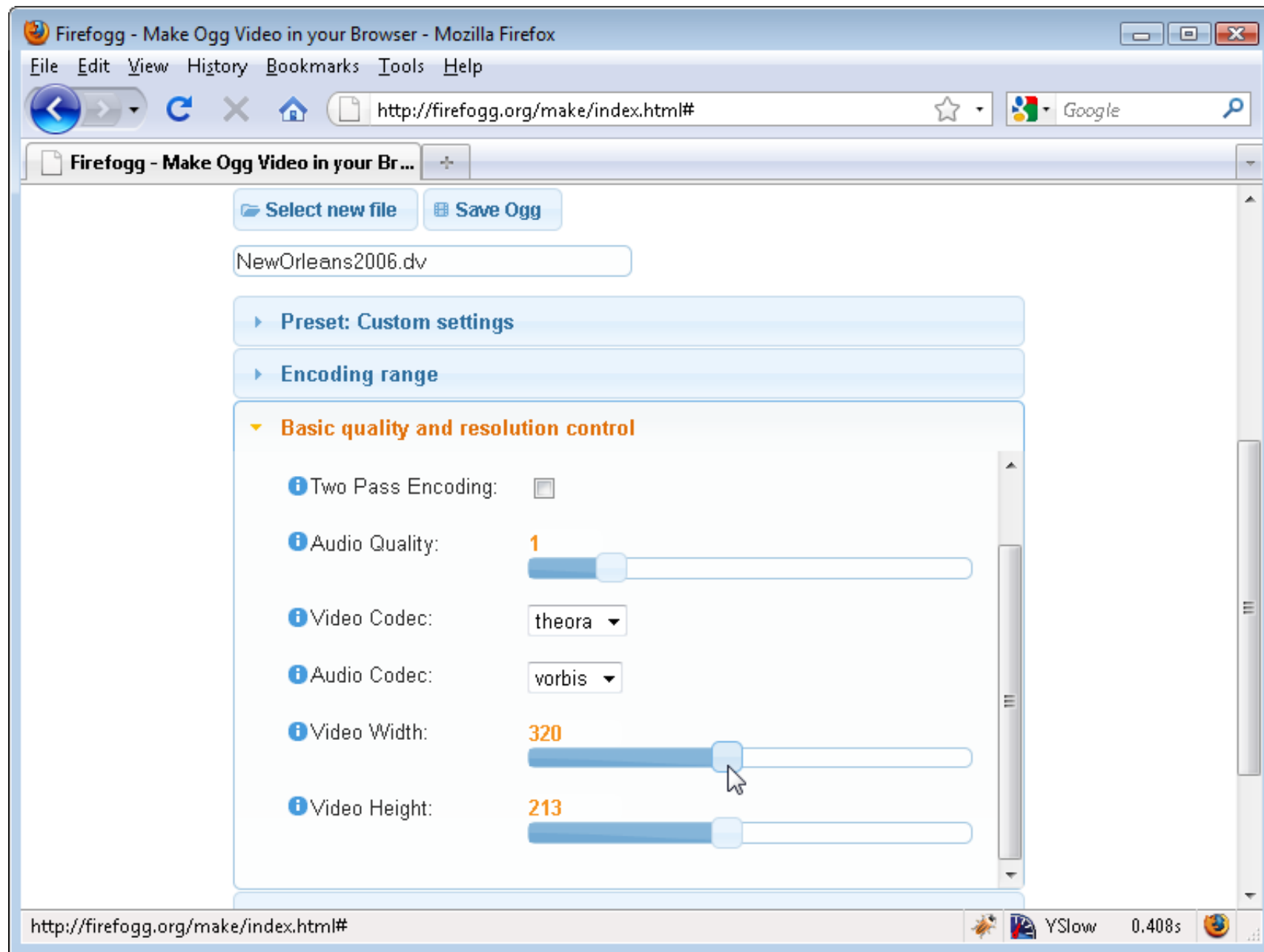
- Qualidade de Vídeo. É mensurada em uma escala de 0 (menor qualidade) a 10 (maior qualidade). Números altos significam tamanhos de arquivos maiores, então você terá de experimentar para determinar qual a melhor relação tamanho/qualidade para sua necessidade.
- Qualidade de Áudio. É mensurada em uma escala de -1 (menor qualidade) a 10 (maior qualidade). Números altos significam tamanhos de arquivos maiores, assim como as configurações de qualidade de vídeo.
- Vídeo Codec. Sempre deve ser "Theora"
- Áudio Codec. Sempre deve ser "Vorbis"
- Largura e comprimento de vídeo. Este é o padrão atual de largura e comprimento do seu vídeo fonte. Se você deseja redimensionar o vídeo durante a codificação, você pode trocar a largura (ou comprimento) aqui. Firefogg vai automaticamente ajustar as outras dimensões para manter as proporções originais (então seu vídeo não terminará esticado ou amassado).



Neste exemplo, vou redimensionar o vídeo para metade do tamanho original de largura. Note como o Firefogg

automaticamente ajusta o comprimento.

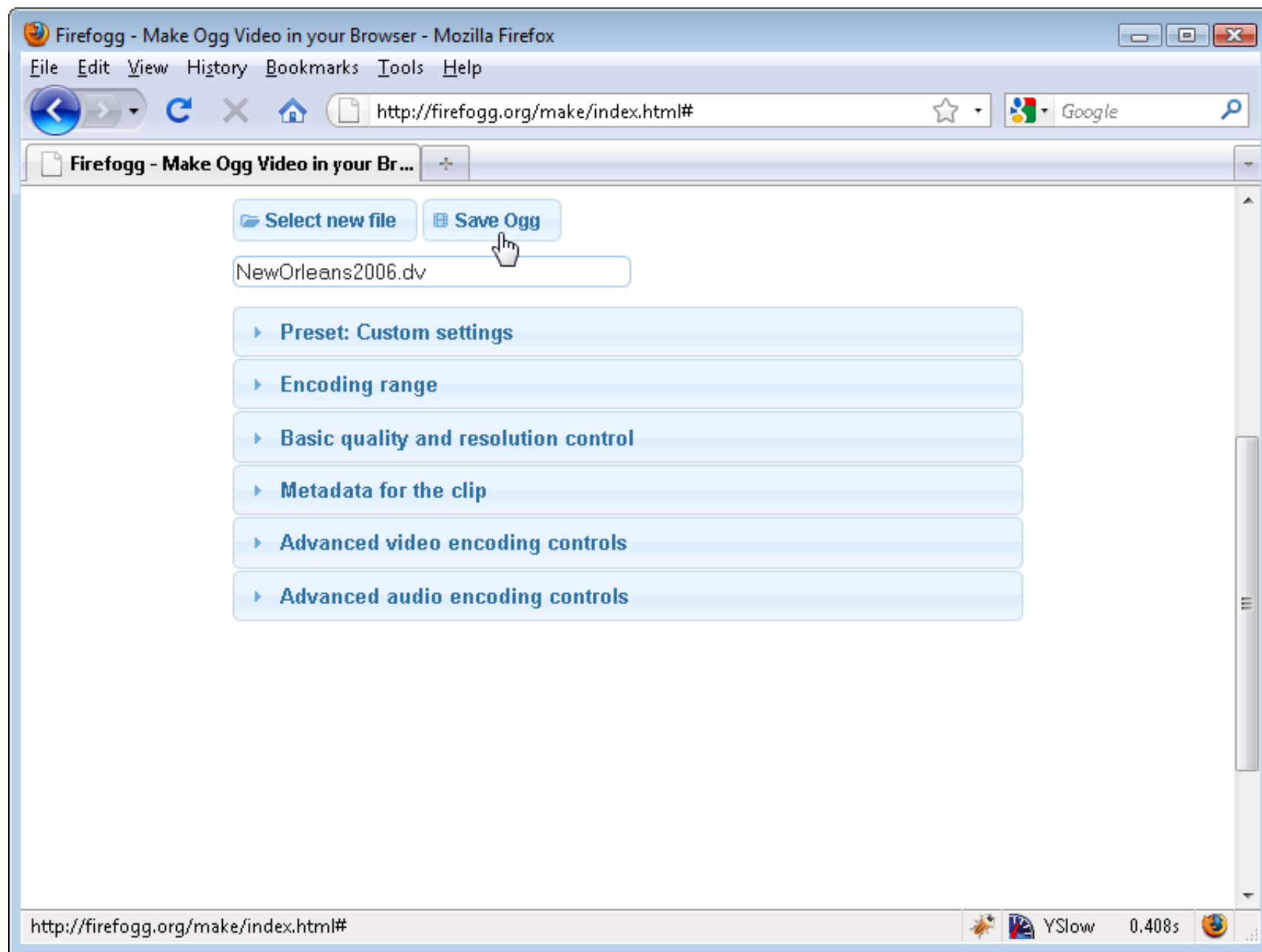
*Ajustando largura e comprimento de vídeo ↷*



Uma vez expandido com todos os botões, clique em "Save Ogg" para iniciar o processo de codificação atual. Firefogg vai

perguntar para você um nome de arquivo para o vídeo codificado.

*"Salvar Ogg" ↩*

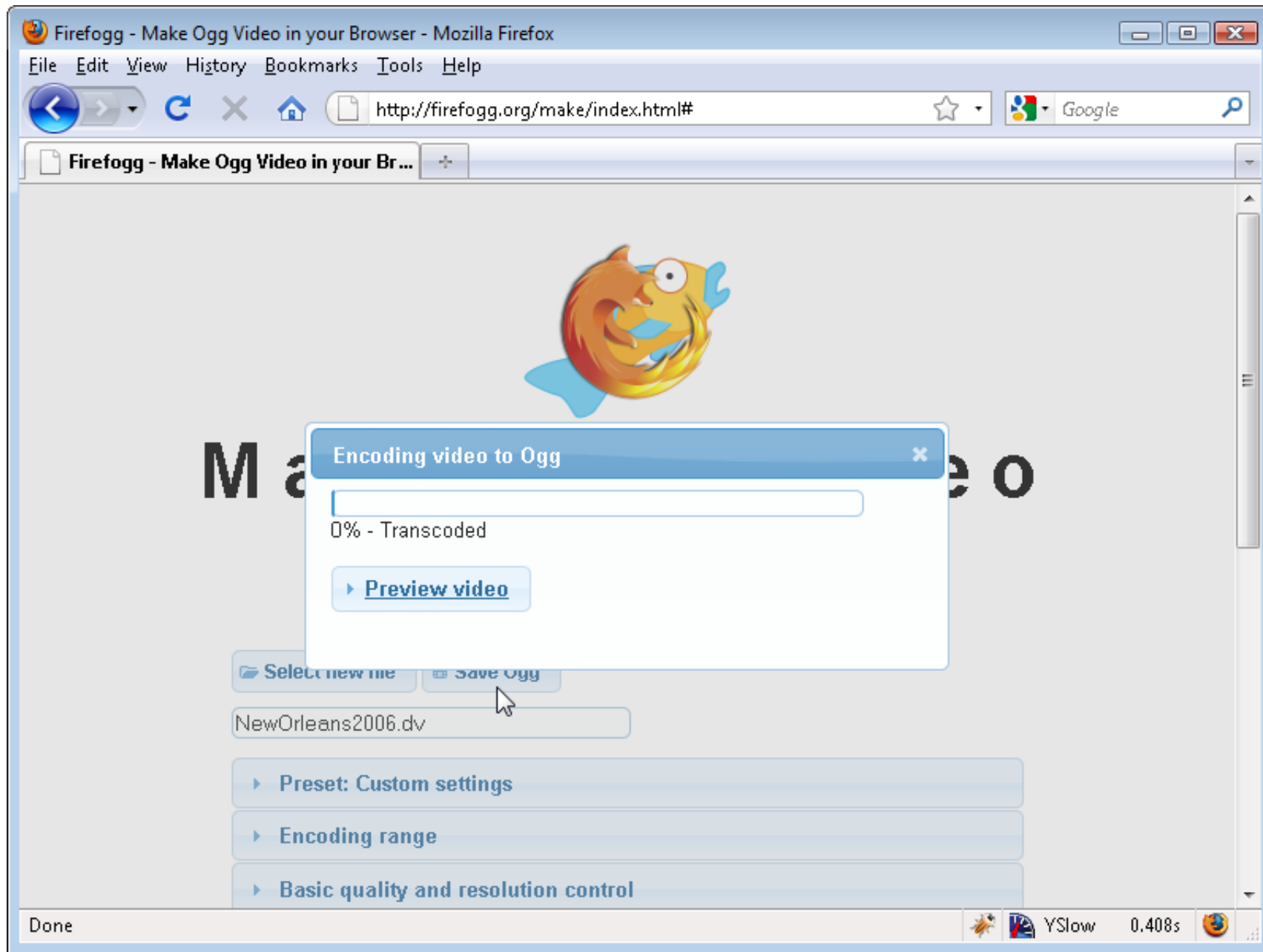


Firefogg vai mostrar uma bela barra de progresso enquanto ele codifica seu vídeo. Tudo que você tem que fazer é aguardar (

e aguardar, e aguardar)!

↪ *Processo de codificação*







# CODIFICAÇÃO EM LOTES DE VÍDEOS OGG COM FFMPEG2THEORA

(Como na seção anterior, nesta eu vou usar "Video Ogg", como encurtamento de "Theora vídeo e áudio Vorbis em um contêiner Ogg". Esta é a combinação de codecs+contêiners que funciona nativamente no Mozilla Firefox e Google Chrome.)

Se você está procurando por uma codificação em lotes de diversos arquivos de vídeo Ogg e você quer automatizar o processo, você definitivamente deveria dar uma olhada no [ffmpeg2theora](#).

O ffmpeg2theora é uma aplicação para codificação de vídeos Ogg de código livre e licença GPL. Binários pré-compilados estão disponíveis [para Mac OS X, Windows, e distribuições Linux modernas](#). Ele pode pegar virtualmente qualquer tipo de arquivo como entrada, incluindo vídeos DV produzidos por câmeras simples, não profissionais.

Para usar o ffmpeg2theora, você precisará chamá-lo pela linha de comando. (No Mac OS X, abra Aplicações → Utilitários → Terminal. No Windows, abra o seu menu Iniciar → Programas → Acessórios → Prompt de Comando.)

O ffmpeg2theora tem diversos parâmetros de linha de comando. (Escreva `ffmpeg2theora --help` para ler sobre todos eles.) Eu irei focar em apenas três deles.

- `--video-quality Q`, onde “Q” é um número entre 0–10.
- `--audio-quality Q`, onde “Q” é um número entre -2–10.
- `--max_size=WxH`, onde “W” e “H” são a largura e altura máximos que você deseja para o vídeo (o “x” entre eles é realmente apenas a letra “x”). O `ffmpeg2theora` irá redimensionar o vídeo proporcionalmente para ajustar a estas dimensões, então o vídeo codificado deverá ser menor que `WxH`. Por exemplo, realizar a codificação de um vídeo 720×480 com tamanho máximo `--max_size 320x240` irá produzir um arquivo que será 230×213.

Assim, aqui está como você deve codificar um vídeo com as mesmas configurações que nós usamos na seção anterior ([codificação com Firefogg](#)).

```
you@localhost$ ffmpeg2theora --videoquality 5
                        --audioquality 1
                        --max_size 320x240
                        pr6.dv
```

O vídeo codificado será salvo no mesmo diretório do arquivo de vídeo original, com a extensão `.ogv` adicionada. Você pode especificar um lugar diferente e/ou um nome diferente passando os seguintes parâmetros `--output=/path/to/encoded/video` para o `ffmpeg2theora`.



# CODIFICANDO VÍDEOS H.264 COM O HANDBRAKE

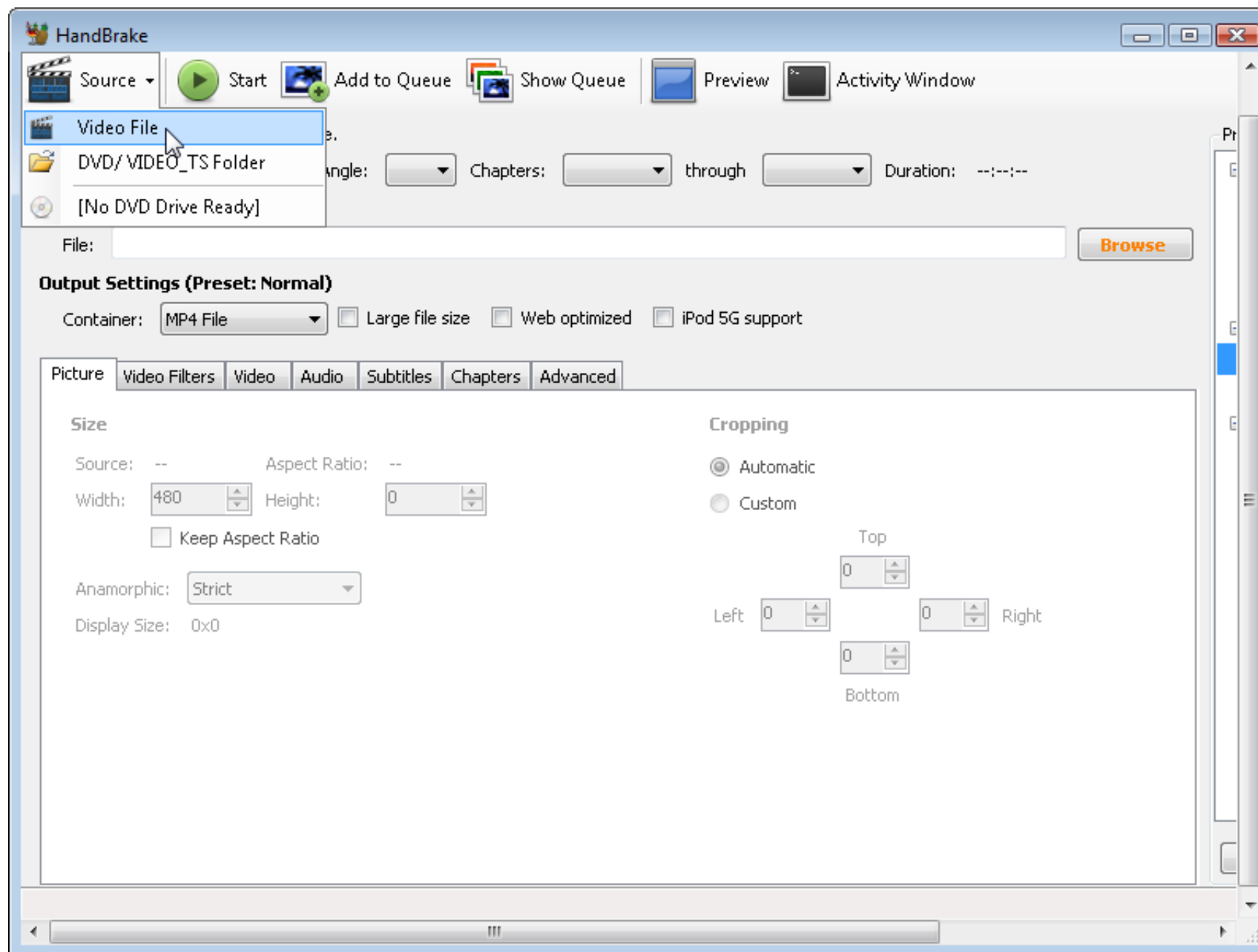
(Nesta seção, eu irei usar "vídeo H.264" como uma abreviação para "Conjunto de especificações e perfis de vídeo H.264 e Perfil de áudio de baixa complexidade AAC em um container MPEG-4". Esta é a combinação de codificadores+containers que funcionam nativamente no Safari, no Adobe Flash, no iPhone e nos aparelhos que utilizam Google Android.)

Tirando as questões de licenciamento, o jeito mais fácil de codificar um vídeo H.264 é usar o HandBrake. O HandBrake é uma aplicação para codificação de vídeos H.264 open source, com licença GPL (as versões anteriores realizavam codificações para outros formatos de vídeo também, mas na última versão os desenvolvedores resolveram retirar o suporte para a maioria dos outros formatos e estão focando todos seus esforços nos vídeos H.264). Os binários pré-compilados estão disponíveis para Windows, Mac OS X e distribuições Linux modernas.

O HandBrake possui duas versões: gráfica e linha de comando. Eu irei fazer um passo-a-passo da interface gráfica primeiro, então veremos quais são as configurações recomendadas traduzidas para a versão de linha de comando.

Depois de abrir a aplicação HandBrake a primeira coisa a fazer é selecionar o seu vídeo de origem. Clique em "Origem" no menu dropdown e escolha "Arquivo de Vídeo" para selecionar um arquivo. O HandBrake teoricamente pode pegar qualquer arquivo de vídeo como origem, incluindo vídeos DV produzidos por câmeras simples.

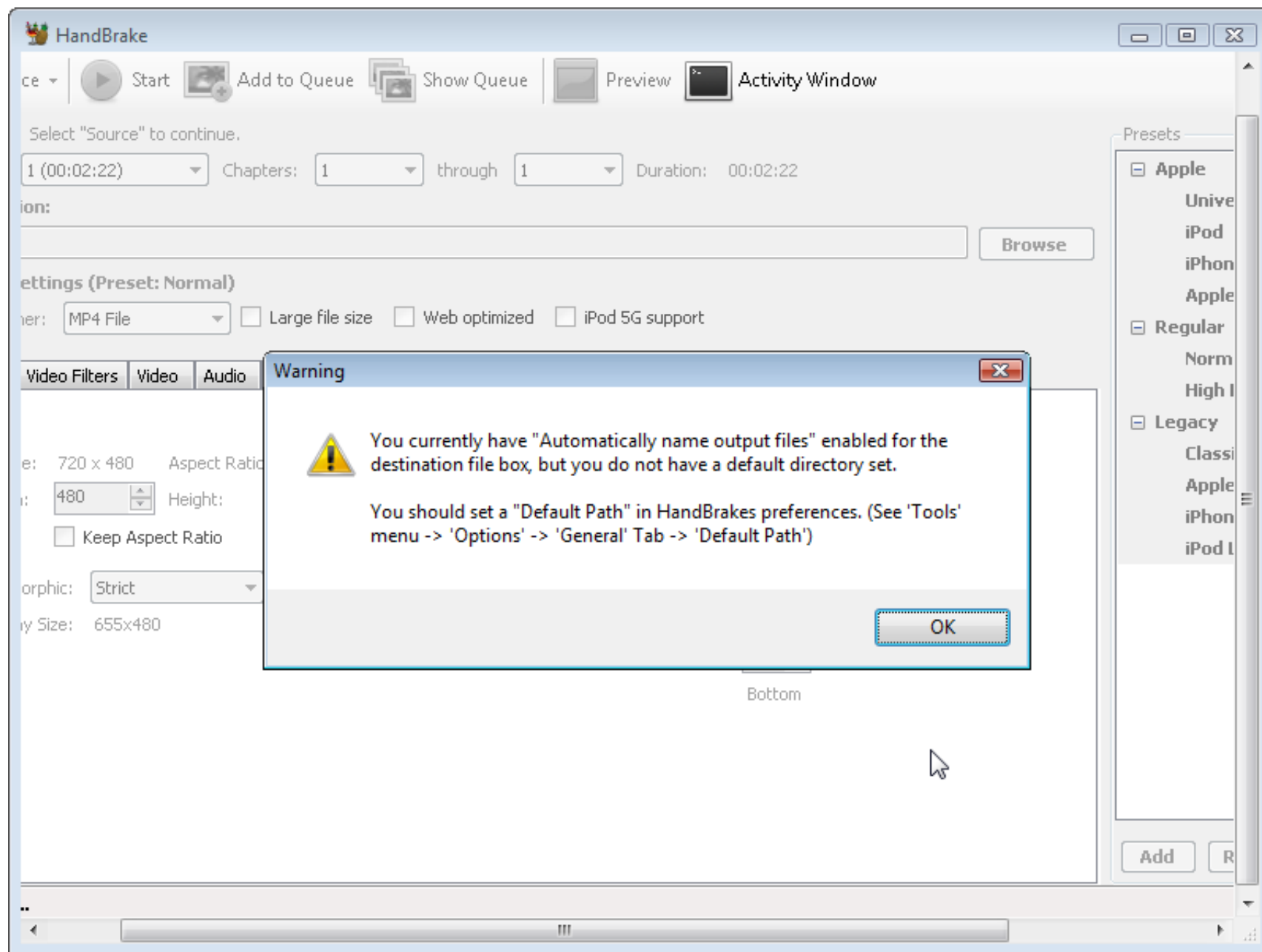
*Selecione seu vídeo de Origem ↷*



O HandBrake irá reclamar que você não configurou um diretório padrão para salvar os arquivos codificados. Você pode

ignorar esse alerta com segurança ou, então, abrir a janela de opções (dentro do menu "Tools") e configurar o diretório padrão de saída.

↪ *Ignore isso*

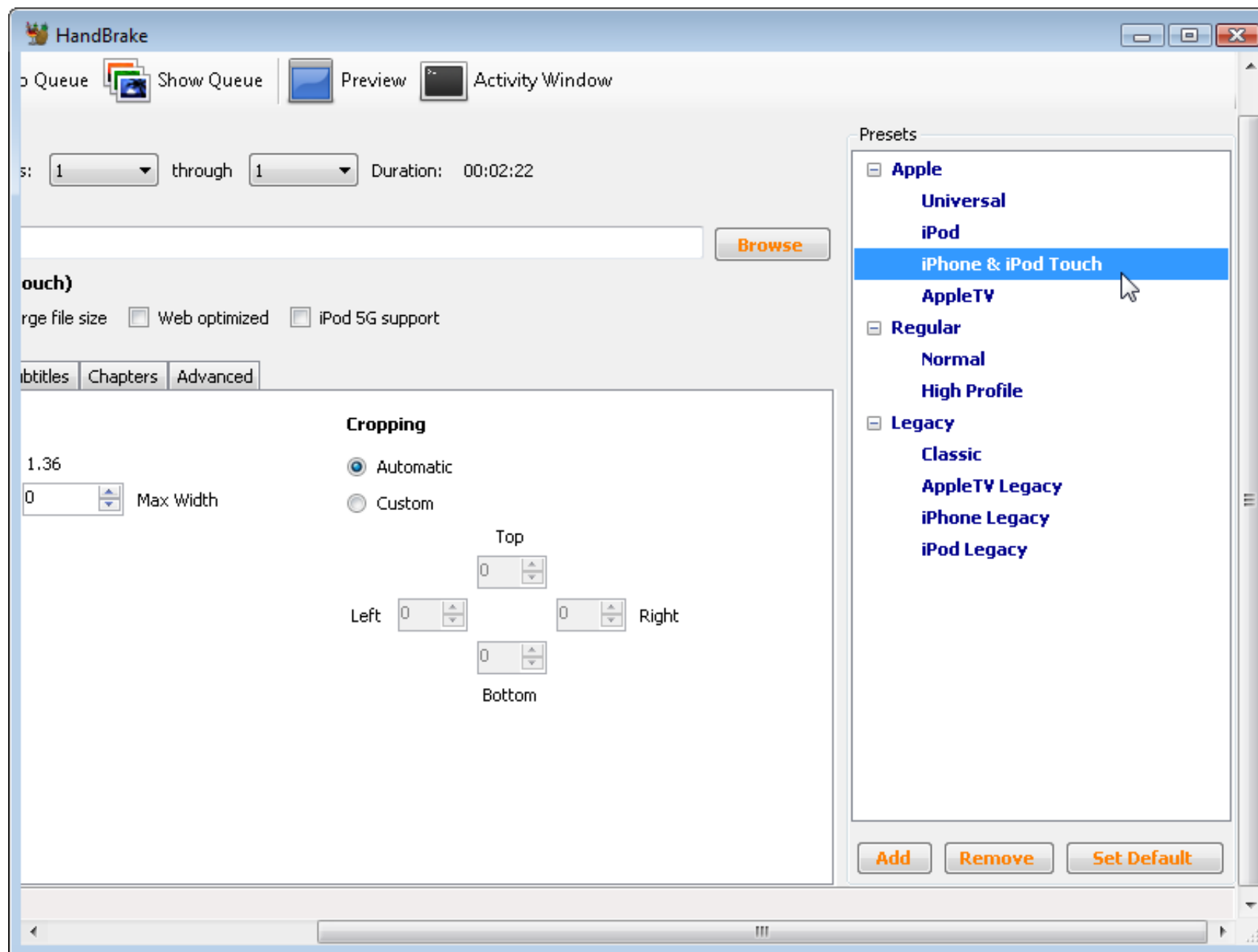


No lado direito há uma lista de predefinições. Selecionando a opção "iPhone & iPod Touch" irá configurar a maioria das



opções que você precisa.

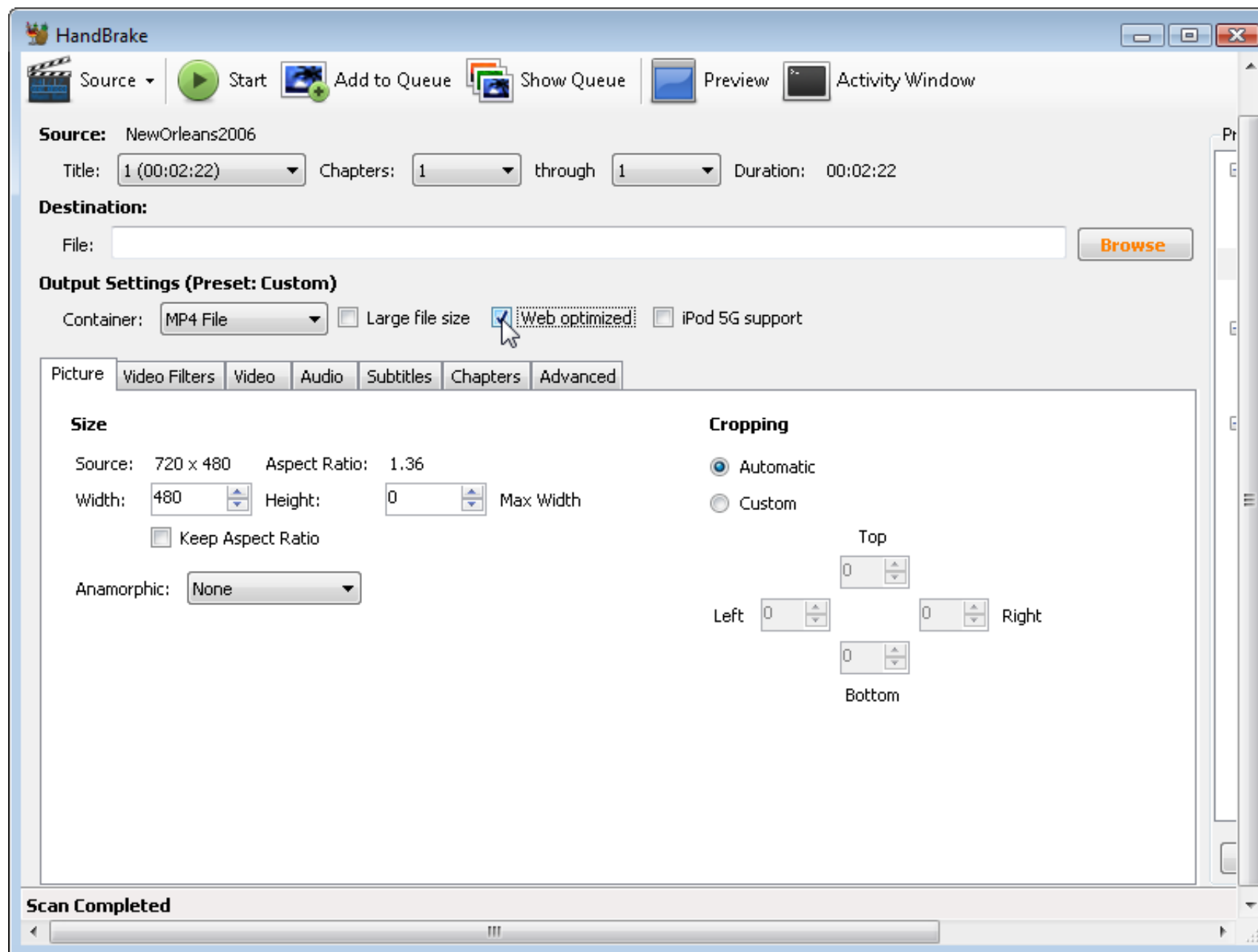
*Selecione a predefinição iPhone ↷*



Uma importante opção, e que está desativada por padrão, é a de "Otimizar para Web". Selecionando esta opção os metadados

do vídeo codificado serão reordenados, para que você possa assistir o início do vídeo enquanto o resto está sendo baixado em segundo plano. Eu recomendo fortemente sempre conferir esta opção. Ela não afeta a qualidade ou tamanho do arquivo do vídeo codificado, portanto, não há razão para não utilizá-la.

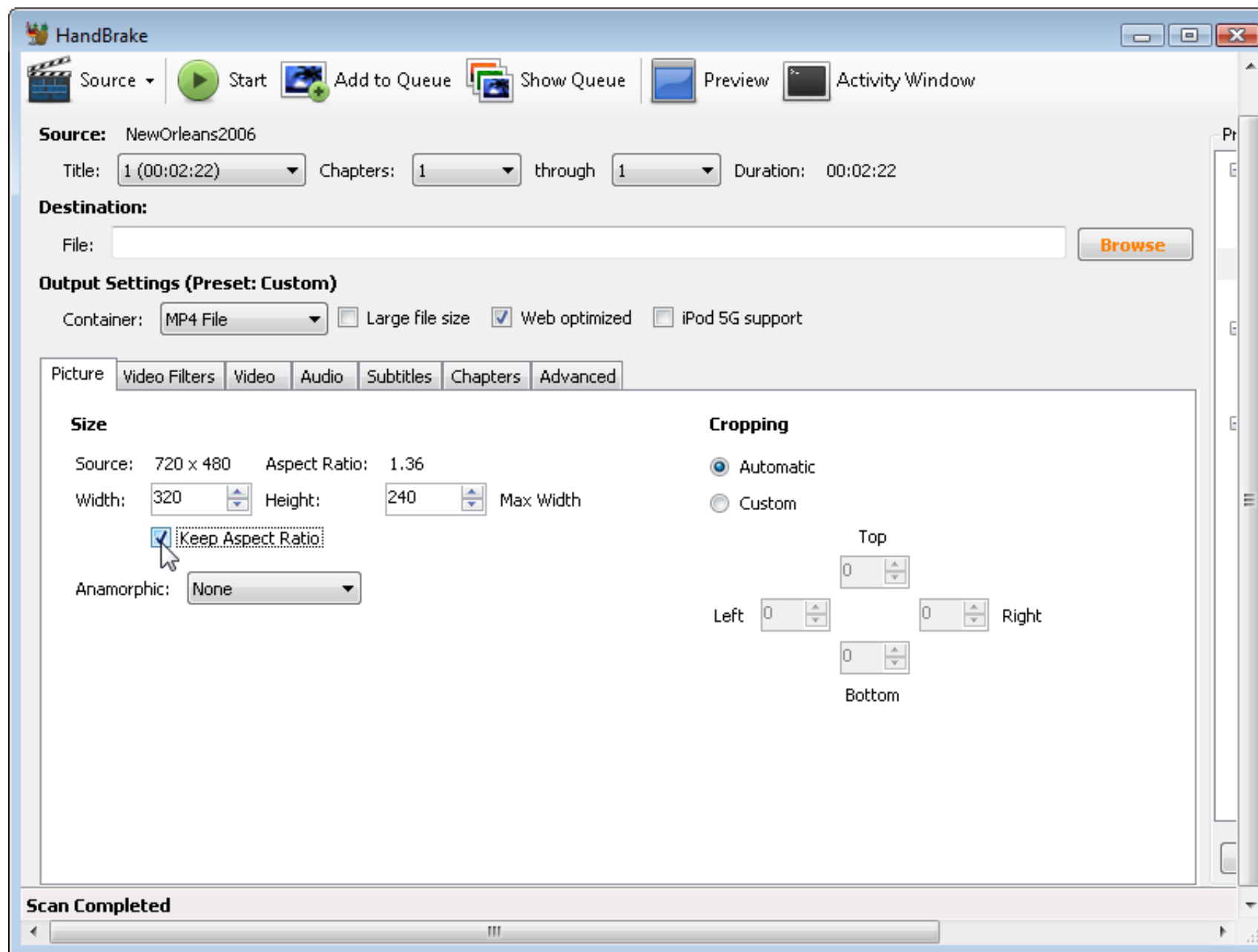
↪ *Sempre otimizar para web*



Na aba de "Imagem" você pode configurar a altura e largura máxima do vídeo codificado. Você também deve selecionar a

opção "Manter a proporção" para assegurar que o HandBrake não irá alargar ou esticar o seu vídeo enquanto o redimensiona.

*Configure a largura e altura ↷*



Na aba "VÍdeo" você pode configurar quatro opções importantes.

- Codec de Vídeo. Tenha certeza que é "H.264 (x264)"
- Codificação em dois passos. Se esta opção estiver selecionada, o HandBrake irá rodar duas vezes a codificação de vídeo. Na primeira vez, ela apenas analisará o vídeo, procurando por coisas como composição de cores, movimento e paradas de cena. Na segunda vez é que realmente será codificado o vídeo utilizando a informação obtida durante a primeira vez. Como você pode esperar, isto toma duas vezes o tempo do que a codificação em um passo, mas o resultado é um vídeo melhor sem aumentar o tamanho do arquivo. Eu sempre habilito a codificação em dois passos para vídeos H.264. A menos que você esteja fazendo o próximo YouTube e codificando vídeos 24 horas por dia, você provavelmente deveria utilizar a codificação em dois passos também.
- Primeira passagem turbo. Uma vez que você tenha habilitado a codificação em dois passos, você pode recuperar algum tempo habilitando a "primeira passagem turbo." Isto reduzirá a quantidade de trabalho feita durante a primeira passagem (análise do vídeo), enquanto degrada levemente a qualidade. Eu usualmente habilito esta opção, mas se a qualidade é a coisa mais importante para você, você deveria deixar isto desabilitado.
- Qualidade. Existem diferentes modos de especificar a "qualidade" dos seus vídeos codificados. Você pode configurar o tamanho final do arquivo e o HandBrake irá fazer o melhor para que o vídeo codificado não seja maior do que o definido. Você pode configurar a média da "taxa de bits" o que é literalmente o número de bits requeridos para armazenar um segundo valor do vídeo codificado (isto é chamado de "média" da taxa de bits porque alguns segundos irão precisar de mais bits que outros). Ou você pode especificar uma qualidade constante numa escada de 0 à 100%. Números maiores irão resultar em uma melhor qualidade, mas em arquivos maiores. Não há apenas uma resposta certa para qual qualidade que você deve usar.

**PERGUNTE AO PROFESSOR MARCAÇÃO**

P: Posso usar codecs em dois passos na codificação de vídeo Ogg?



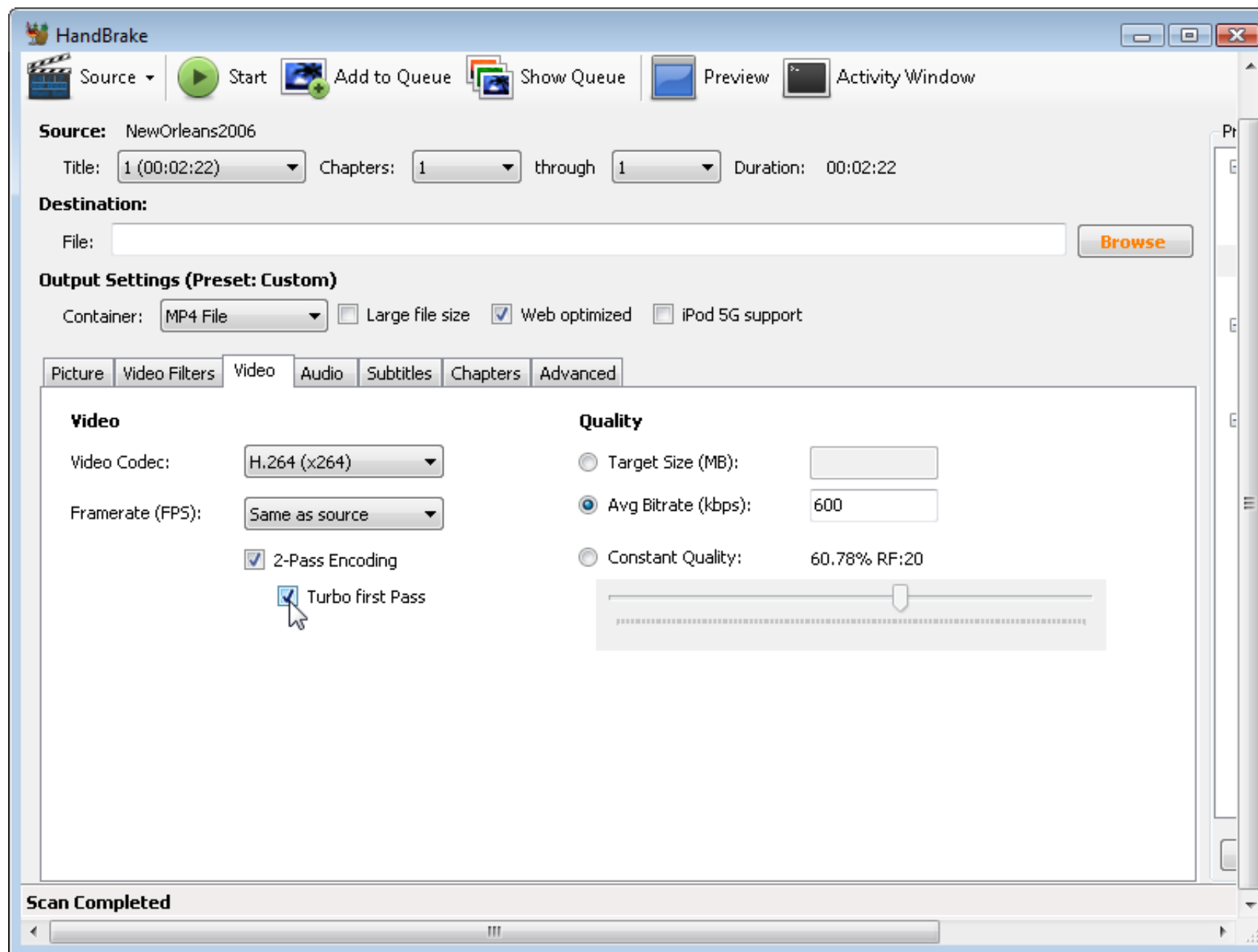
R: Sim, sim mas devido as diferenças fundamentais em como o codificador funciona, você provavelmente não precisa fazê-lo. Codecs em dois passos H.264 quase sempre resultam em vídeos de qualidade mais altas. Codecs Ogg em dois passos de vídeos Ogg só é útil se você está tentando fazer com que seu vídeo codificado tenha um tamanho específico. (Talvez isto é algo em que você esteja interessado, mas não é o que é mostrado nestes exemplos, e provavelmente não vale o tempo extra, codificando o vídeo.) Para a melhor qualidade de vídeo Ogg, use as configurações de qualidade de vídeo, e não se preocupe com codecs em dois passos.



Neste exemplo, Eu escolhi uma taxa de bit médio de 600 kbps, que é bastante alta para um vídeo codificado de 320x240. (Mais tarde neste capítulo, eu vou mostrar um exemplo de vídeo codificado a 200 kbps.) Eu escolhi também codecs em dois passos com um primeiro passo "Turbo".

↪ *Opções de qualidade de vídeo*

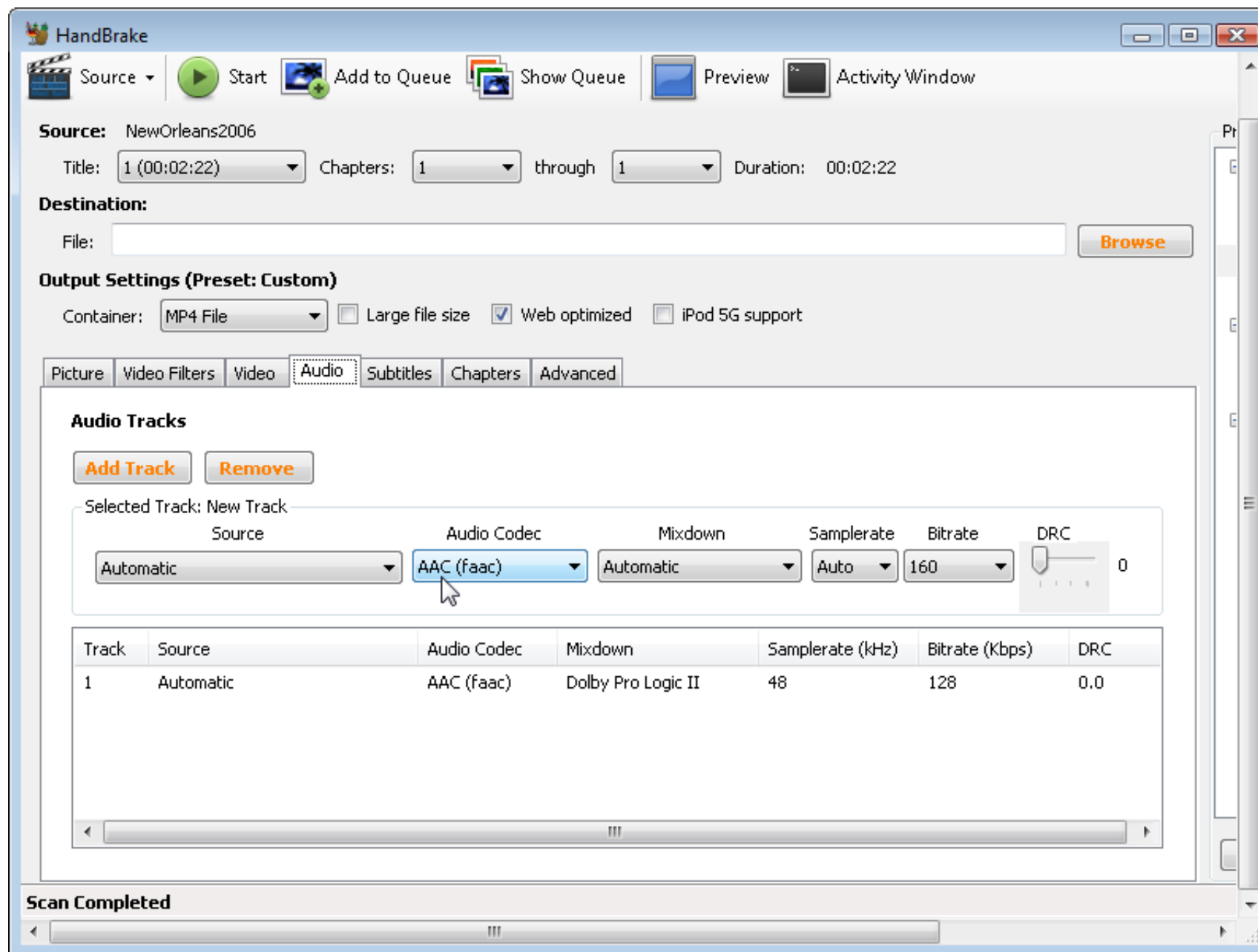




Na aba "Audio", você provavelmente não precisará mudar nada. Se o seu vídeo fonte tiver várias faixas de áudio, você pode

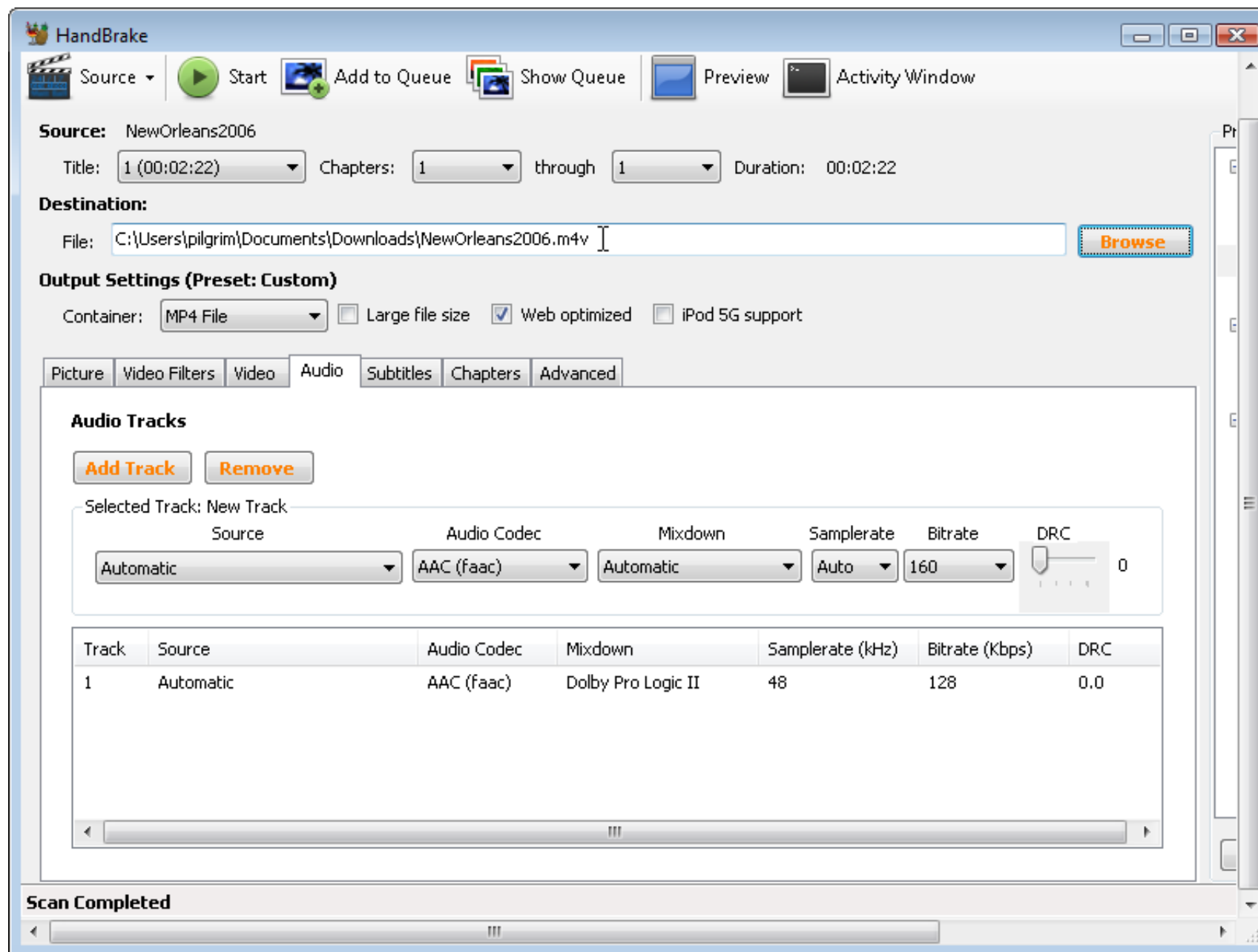
precisar selecionar qual delas você deseja no vídeo. Se seu vídeo é na maior parte um discurso pessoal (ao contrário de músicas ou sons ambientes genéricos), você provavelmente pode reduzir a taxa de bit de áudio para 96 kbps or algo parecido. Por outro lado, o padrão predefinido que você herda do "iPhone" deve ser bom.

*Opções na qualidade de Áudio ↷*



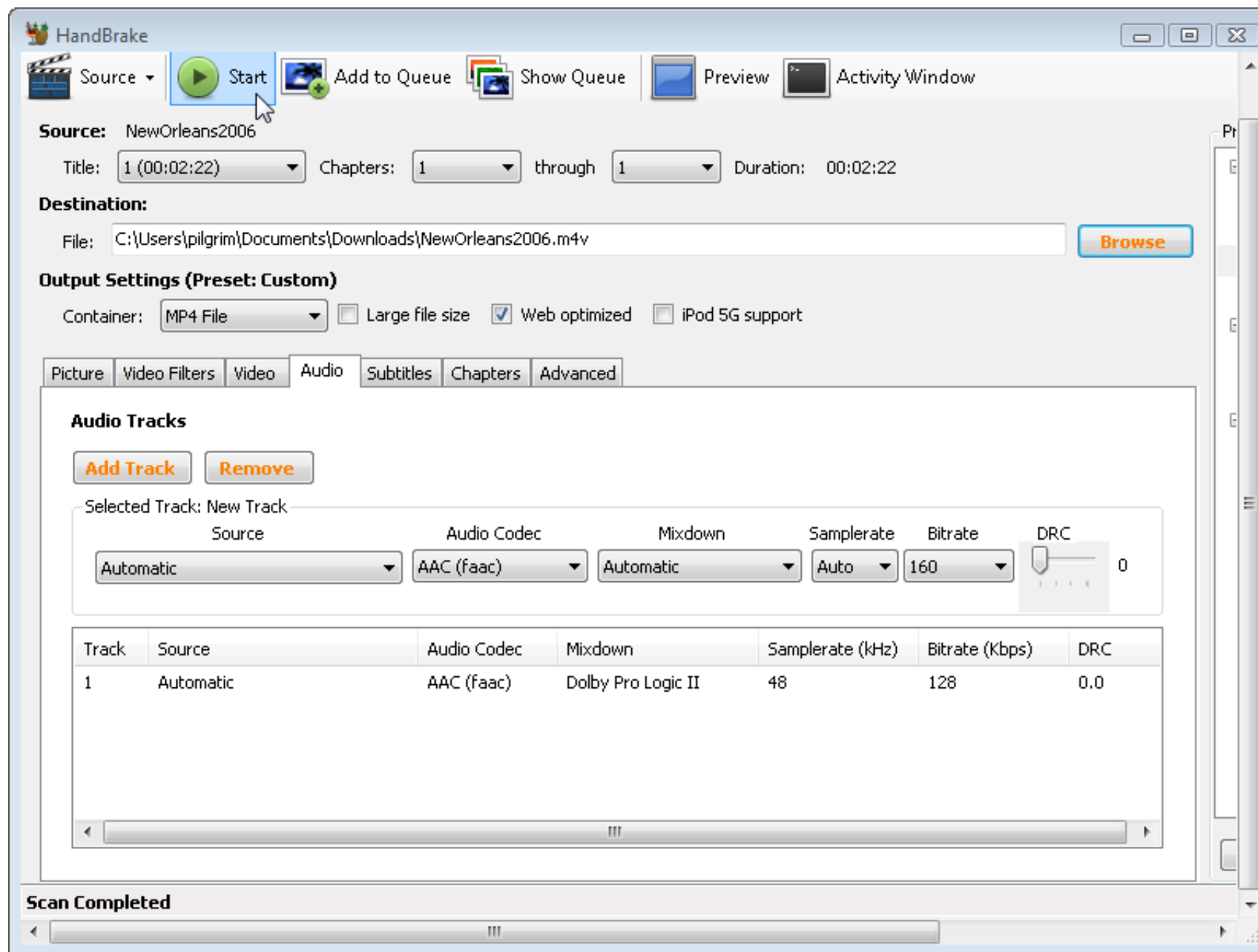
Próximo, clique no botão do "Browse" a escolha o diretório e o nome do arquivo para salvar seu vídeo codificado.

↪ *Defina o destino e o nome do arquivo*



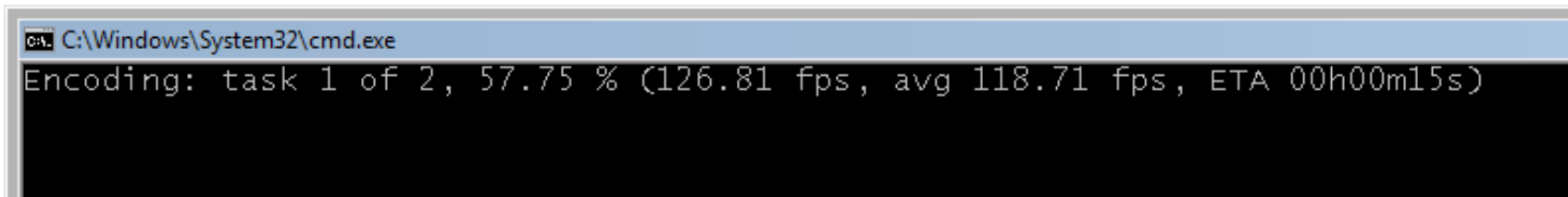
Finalmente, clique "Start" para começar a codificar.

*Vamos fazer alguns vídeos! ↷*



HandBrake irá mostrar a estatística do progresso enquanto codifica seu vídeo.

↪ *Paciência, Gafanhoto*

A screenshot of a Windows command prompt window. The title bar shows 'C:\Windows\System32\cmd.exe'. The command prompt displays the text: 'Encoding: task 1 of 2, 57.75 % (126.81 fps, avg 118.71 fps, ETA 00h00m15s)'.

```
C:\Windows\System32\cmd.exe
Encoding: task 1 of 2, 57.75 % (126.81 fps, avg 118.71 fps, ETA 00h00m15s)
```



## CODIFICAÇÃO BATCH DE VÍDEOS H.264 COM HANDBRAKE

(Assim como na seção anterior, nesta seção eu vou usar "Vídeo H.264" como abreviação para "Perfil de Linha de base de vídeo H.264 e AAC áudio de baixa-complexidade em um container MP4". Esta é a combinação de codecs + container que funcionam nativamente no Safari, Adobe Flash, iPhone, e nos dispositivos Android.)

HandBrake também vem em uma edição na linha de comandos. Tal como ffmpeg2theora, a edição da linha de comando do HandBrake oferece uma vertiginoso matriz de opções. (Digite HandBrakeCLI --help para ler a respeito.) Eu irei focar em



apenas alguns:

- `--preset "X"`, onde "X"; é o nome do padrão HandBrake. A predefinição que você quer para vídeo web H.264 é chamada "iPhone" & "iPhone Touch", e é importante colocar o nome inteiro entre aspas.
- `--width W`, onde "W" é a largura do seu vídeo codificado. HandBrake irá automaticamente ajustar a altura para manter a proporção original do vídeo.
- `--vb Q`, onde "Q" é a taxa de bit média (mensurada em kilobits por segundo).
- `--two-pass`, que habilita a codificação em dois passos.
- `--turbo`, que habilita primeiro passo turbo durante a codificação em dois passos.
- `--input F`, onde "F" é o nome do seu vídeo fonte.
- `--output E`, onde "E" é o destino para seu vídeo codificado.

Aqui tem um exemplo da chamada do HandBrake na linha de comando, com marcas na linha de comando que acertam as configurações que escolhemos [Com a versão gráfica do HandBrake.](#)

```
you@localhost$ HandBrakeCLI --preset "iPhone & iPod Touch"
--width 320
--vb 600
--two-pass
--turbo
--input pr6.dv
--output pr6.mp4
```

De cima para baixo, estes comandos rodam HandBrake com predefinições "iPhone & iPhone Touch", redimensiona o vídeo

para 320x240, define a taxa de bit média para 600 kbps, habilita a codificação em dois passos com primeiro passo turbo, lê o arquivo pr6.dv, e codifica como pr6.mp4. Ufa!



# CODIFICANDO VÍDEOS WEBM COM FFMPEG

WebM é completamente suportado por ffmpeg 0.6+. Na linha de comando, rode ffmpeg sem parâmetros e verifique que foi compilado com suporte VP8:

```
you@localhost$ ffmpeg
FFmpeg version SVN-r23197, Copyright (c) 2000-2010 the FFmpeg developers
  built on May 19 2010 22:32:20 with gcc 4.4.3
  configuration: --enable-gpl --enable-version3 --enable-nonfree --enable-postproc --enable-pthreads
--enable-libfaac --enable-libfaad --enable-libmp3lame --enable-libopencore-amrnb --enable-
libopencore-amrwb --enable-libtheora --enable-libx264 --enable-libxvid --enable-x11grab
--enable-libvorbis --enable-libvpx
```

Se você não ver as palavras mágicas “--enable-libvorbis” e “--enable-libvpx,” você não tem a versão certa do ffmpeg. (Se você mesmo compilou ffmpeg, verifique se não há duas versões instaladas. Isso é tranquilo, eles não irão conflitar entre si. Você precisará apenas usar o caminho completo da versão VP8 do ffmpeg.)

Eu vou fazer uma codificação de dois passos. Passo 1 apenas para verificar o arquivo de vídeo inserido (-i pr6.dv) e escrever algumas estatísticas no arquivo de log (que será auto-nomeado de pr6.dv-0.log). Eu especifico o codec de vídeo com o parâmetro -vcodec:

```
you@localhost$ ffmpeg -pass 1 -passlogfile pr6.dv -threads 16 -keyint_min 0 -g 250 -skip_threshold 0 -qmin 1 -qmax 51 -i pr6.dv -vcodec libvpx -b 614400 -s 320x240 -aspect 4:3 -an -y NUL
```

A maioria da linha de comando do ffmpeg tem nada a ver com VP8 ou WebM. libvpx suporta um número específico de opções VP8 que você pode passar para o ffmpeg, mas eu não sei ainda como trabalhar com elas. Assim que encontrar uma boa explicação sobre elas, irei colocar o link aqui e incorporá-la na narrativa se valer a pena.

Para o segundo passo, ffmpeg irá ler as estatísticas que foram escritas durante o primeiro passo e realmente codificar o vídeo e áudio. Irá escrever um arquivo .webm.

```
you@localhost$ ffmpeg -pass 2 -passlogfile pr6.dv -threads 16 -keyint_min 0 -g 250 -skip_threshold 0 -qmin 1 -qmax 51 -i pr6.dv -vcodec libvpx -b 614400 -s 320x240 -aspect 4:3 -acodec libvorbis -y pr6.webm
```

Existem cinco parâmetros importantes aqui:

- -vcodec libvpx especifica que nós estamos codificando com o codec de vídeo VP8. WebM sempre usa vídeo VP8.
- -b 614400 especifica a taxa de bits. Diferente de outros formatos, libvpx espera que a taxa de bits seja mesmo em bits, e não kilobits. Se você quer um vídeo de 600 kbps, multiplique 600 por 1024 e tenha 614400.

- `-s 320x240` especifica o tamanho desejado, largura por altura.
- `-aspect 4:3` especifica a proporção do vídeo. A definição padrão de vídeo normalmente é 4:3, mas a maioria dos vídeos de alta definição são 16:9 ou 16:10. Nos meus testes, encontrei que preciso especificar de forma explícita isso na linha de comando, ao invés de recorrer ao `ffmpeg` para auto-detectar isso.
- `-acodec libvorbis` especifica que nós estamos codificando com codec de áudio Vorbis. WebM sempre usa áudio Vorbis.



## ENFIM, A MARCAÇÃO

Eu tenho certeza que esse era pra ser um livro sobre HTML. Então cadê a marcação?

HTML5 lhe dá duas formas de incluir vídeos na sua página web. Ambas envolvem o elemento `<video>`. Se você tem apenas um arquivo de vídeo, você pode simplesmente criar link para ele com o atributo `src`. Isso é muito similar a incluir um vídeo com uma tag ``.

*Um arquivo de vídeo* ↪

```
<video src="pr6.webm"></video>
```

Tecnicamente, isso é tudo que você precisa. Mas assim como uma tag <img>, você sempre pode incluir atributos `width` e `height` nas suas tags de <video>. Os atributos `width` e `height` podem ser os mesmos que os atributos de largura e altura máxima que você especificou durante o processo de codificação. Não se preocupe se uma dimensão do vídeo é um pouco menor que isso. Seu navegador irá centralizar o vídeo dentro da caixa definida pela tag <video>. Nunca será esticado ou achatado fora da proporção.

```
<video src="pr6.webm" width="320" height="240"></video>
```

Por padrão, o elemento <video> não irá expor quaisquer controles do player. Você pode criar seus próprios controles com os bons e velhos HTML, CSS, e JavaScript. O elemento <video> possui métodos como `play()` e `pause()` e propriedades de leitura/escrita chamados `currentTime`. Também há propriedades de leitura/escrita como `volume` e `muted`. Então você realmente tem tudo o que precisa para criar sua própria interface.

Se você não quer construir sua própria interface, você pode dizer ao navegador para exibir os controles pré-definidos. Para isso, apenas inclua o atributo `controls` na sua tag <video>.

```
<video src="pr6.webm" width="320" height="240" controls></video>
```

Há outros dois atributos opcionais que eu quero mencionar antes de ir além: `preload` e `autoplay`. Não dispare o mensageiro; deixe-me explicar porque eles são úteis. O atributo `preload` diz ao navegador que você gostaria de começar o download do arquivo de vídeo assim que a página carregar. Isso faz sentido se o único objetivo da página é visualizar um vídeo. Por outro lado, se é apenas um material suplementar que apenas alguns visitantes irão assistir, então você pode definir o `preload` como `none` para dizer ao navegador que minimize o tráfego de rede.

Aqui está um exemplo de um vídeo que irá começar a fazer o download (mas não tocar) assim que a página carregar:

```
<video src="pr6.webm" width="320" height="240" preload></video>
```

E aqui está um exemplo de um vídeo que *não* irá começar o download assim que a página carregar:

```
<video src="pr6.webm" width="320" height="240" preload="none"></video>
```

O atributo `autoplay` faz exatamente o que parece: diz ao navegador que você gostaria de começar o download do vídeo assim que a página carregar, *e* que você gostaria de começar a tocar o vídeo assim que possível. Algumas pessoas amam isso; outras odeiam. Mas deixe-me explicar por que é importante ter um atributo como esse na HTML5. Algumas pessoas irão querer que seus vídeos toquem automaticamente, mesmo que isso irrite seus visitantes. Se a HTML5 *não* definisse como padrão um modo de tocar automaticamente um vídeo, as pessoas iriam recorrer ao JavaScript com hacks para fazer isso de qualquer forma. (Por exemplo, chamando o método `video's play()` durante o evento da janela de `load`.) Isso poderia ser muito difícil para os usuários contra-atacarem. Por outro lado, basta adicionar uma extensão no seu navegador (ou escrever uma, se necessário) para dizer “ignore o atributo `autoplay`, não quero nunca ver tocar vídeos automaticamente.”

Aqui está um exemplo de vídeo que irá começar o download e tocar assim que for possível, depois de carregar a página:

```
<video src="pr6.webm" width="320" height="240" autoplay></video>
```

E aqui está um script [Greasemonkey](http://greasemonkey.org/) que você pode instalar localmente no seu Firefox para prever que o vídeo da HTML5 toque automaticamente. Ele usa o atributo `autoplay` do DOM definido na HTML5, que é equivalente ao atributo `autoplay`

da sua marcação HTML. [[disable\\_video\\_autoplay.user.js](#)]

```
// ==UserScript==
// @name          Disable video autoplay
// @namespace      http://diveintomark.org/projects/greasemonkey/
// @description    Ensures that HTML5 video elements do not autoplay
// @include        *
// ==/UserScript==

var arVideos = document.getElementsByTagName('video');
for (var i = arVideos.length - 1; i >= 0; i--) {
    var elmVideo = arVideos[i];
    elmVideo.autoplay = false;
}
```

Mas espere um segundo... Você tem seguido por todo esse capítulo, você não tem apenas um arquivo de vídeo; você tem três. Um é um arquivo .ogv criado com [Firefogg](#) ou [ffmpeg2theora](#). O segundo é um arquivo .mp4 criado com [HandBrake](#). O terceiro é um arquivo .webm que você criou com [ffmpeg](#). HTML5 provê um modo de criar links para três deles: o elemento <source>. Cada elemento <video> pode conter mais que um elemento <source>. Seu navegador irá atrás de uma lista de vídeos, em ordem, e tocar apenas o primeiro que estiver habilitado para isso.

Isso levanta outra questão: como o navegador sabe qual vídeo tocar? Bom, no pior cenário, ele carrega cada um dos vídeos e tenta tocar eles. Porém, isso é uma tremenda perda de banda. Você irá economizar muito tráfego de rede se disser primeiro

ao navegador sobre cada vídeo. Você diz isso com o atributo `type` no elemento `<source>`.

Aqui está a coisa toda:

*Três (!) arquivos de vídeo ~*

```
<video width="320" height="240" controls>
  <source src="pr6.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
  <source src="pr6.webm" type='video/webm; codecs="vp8, vorbis"'>
  <source src="pr6.ogv" type='video/ogg; codecs="theora, vorbis"'>
</video>
```

Vamos quebrar isso. O elemento `<video>` especifica a largura e altura do vídeo, mas isso não linka para um arquivo de vídeo. Dentro do elemento `<video>` existem três elementos `<source>`. Cada elemento `<source>` linka para um único arquivo de vídeo (com o atributo `src`), e ainda dá informações sobre o formato de vídeo (no atributo `type`).

O atributo `type` parece complicado — droga, e é mesmo complicado. É uma combinação de três pedaços de informações: o formato de container, o codec de vídeo, e o codec de áudio. Vamos começar por baixo. Para o arquivo de vídeo `.ogv`, o formato de container format é Ogg, representado por `video/ogg`. (Tecnicamente falando, isso é o MIME type para arquivos de vídeo Ogg.) O codec de vídeo é Theora, e o codec de áudio é Vorbis. Isso é simples o suficiente, exceto pelo formato do valor do atributo parecer meio estranho. O valor por si precisa incluir marcações por aspas, que significa que você precisa usar um tipo diferente de marcação por aspas para todo valor.



```
<source src="pr6.ogv" type='video/ogg; codecs="theora, vorbis"'>
```

WebM é praticamente a mesma coisa, mas com um diferente MIME type (video/webm ao invés de video/ogg) e um diferente codec de vídeo (vp8 ao invés de theora) listados no parâmetro de codecs.

```
<source src="pr6.webm" type='video/webm; codecs="vp8, vorbis"'>
```

O vídeo H.264 é ainda mais complicado. Lembra quando eu disse que ambos vídeo H.264 e áudio AAC podem vir com diferentes “perfis”? Nós codificamos com o perfil “baseline” H.264 e com o perfil “low-complexity” AAC, quando envolvidos em um container MPEG-4. Toda sua informação está inclusa no atributo type.

```
<source src="pr6.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
```

O benefício de passar por todos esses problemas é que o navegador irá checar o atributo type primeiro para poder tocar um arquivo de vídeo em particular. Se um navegador decide que não pode tocar um vídeo em particular, *ele não irá realizar o download do arquivo*. Nem mesmo uma parte dele. Você irá economizar banda, e seus visitantes irão ver o vídeo que eles vieram pra ver, mais rápido.

Se você seguir as instruções nesse capítulo para codificar seus vídeos, você pode apenas copiar e colar os valores dos atributos type desse exemplo. Ou então, você terá que lidar com parâmetros type você mesmo.

PROFESSOR MARCAÇÃO DIZ

iPads rodando iOS 3.x tem um bug que previnam eles de notificar qualquer coisa menos a primeira origem do vídeo listado. iOS 4 (um upgrade gratuito para todos iPads) corrige esse bug. Se você quer entregar vídeo para usuários de iPad que ainda não tenham atualizado para iOS 4, você terá que listar seu arquivo MP4 primeiro, seguido dos formatos de vídeo gratuitos. *Sigh.*



## MIME TYPES ENCONTRAM SEU MONSTRO

Existem tantas peças no quebra-cabeça do vídeo, eu até mesmo hesitado em trazer isso à tona. Mas é importante, porque uma configuração mal feita no web server pode levar a incontáveis frustrações quando você tenta debugar o porquê do seu vídeo tocar localmente no seu computador mas falhar ao tentar tocar quando você realiza o deploy para o site em produção. Se você se deparar com esse problema, a principal causa é provavelmente MIME types.

Eu mencionei MIME types [na história desse capítulo](#), mas você provavelmente passou por aquilo sem dar a importância

devida. Então aqui está em:

## PROFESSOR MARCAÇÃO GRITA

ARQUIVOS DE VÍDEO DEVEM SER SERVIDOS COM O MIME TYPE CORRETO!



Qual é o MIME type correto? Você já viu ele; é uma parte do valor do atributo `type` no elemento `<source>`. Mas definir o atributo `type` no seu HTML não é o suficiente. Você também precisa garantir que seu web server inclui o MIME type correto no Content-Type do cabeçalho HTTP.

Se você está usando um servidor web Apache ou qualquer derivado Apache, você pode usar uma diretiva AddType no `httpd.conf` do seu site ou em um arquivo `.htaccess` no diretório onde você armazenou seus arquivos de vídeo. (Se você usa outro servidor web, consulte a documentação dele sobre como definir Content-Type no cabeçalho HTTP para tipos específicos de arquivo.)

```
AddType video/ogg .ogv  
AddType video/mp4 .mp4  
AddType video/webm .webm
```

A primeira linha é para vídeos com container Ogg. A segunda linha é para vídeos com container MPEG-4. A terceira é para WebM. Defina uma vez e esqueça isso. Se você esquecer de definir isso, seus vídeos *irão* falhar ao tocar em alguns navegadores, mesmo que você inclua o MIME no atributo `type` do seu HTML.

Para ainda mais detalhes sobre como configurar seu servidor web, eu direciono sua atenção para esse excelente arquivo na Mozilla Developer Center: [Configurando servidores para mídia Ogg](#). (O conselho desse artigo se aplica para vídeo MP4 e WebM também.)



## E QUANTO AO IE?

Internet Explorer 9 [suporta o elemento <video> da HTML5](#), mas a [Microsoft prometeu publicamente](#) que a versão final do IE 9 irá suportar vídeo H.264 video e áudio AACem um container MPEG-4, assim como Safari e iPhone.

Mas e quanto as versões antigas do Internet Explorer? Tipo, você sabe, todas as versões antigas incluindo o IE 8? A maioria das pessoas que usam Internet Explorer também possuem um plugin Adobe Flash plugin instalado. Versões modernas do

Adobe Flash (começando com 9.0.60.184) suportam vídeo H.264 e áudio AAC em um container MPEG-4, assim como Safari e iPhone. Uma vez [codificado com vídeo H.264](#) para Safari, você pode tocar isso em um video player baseado em Flash se você detectar que um dos seus visitantes não possuem navegadores com suporte ao HTML5.

[FlowPlayer](#) é de código livre, licenciado com GPL, com video player baseado em Flash. ([Licenças comerciais também estão disponíveis.](#)) FlowPlayer não sabe nada sobre seu elemento <video>. Não irá magicamente transformar uma tag <video> em um objeto Flash. Mas um HTML5 bem feito sabe como lidar com isso, porque você pode colocar um elemento <object> dentro do <video>. Navegadores que não suportam vídeo HTML5 irão ignorar o elemento <video> e simplesmente renderizar o elemento <object>, no qual irá invocar o plugin Flash e tocar o vídeo usando FlowPlayer. Navegadores que suportam vídeo HTML5 irão encontrar a origem do vídeo que eles podem tocar e tocá-los, *e ignorar o elemento <object> junto com isso.*

Esse último pedaço é a chave para todo quebra-cabeça: HTML5 especifica que todos os elementos (além dos elementos <source>) são filhos de um elemento <video> devem ser ignorados. Isso permite que você possa usar vídeos HTML5 em navegadores mais novos e prover um fall back em Flash para navegadores mais antigos, sem precisar de qualquer hack JavaScript. Você pode ler mais sobre essa técnica em: [Vídeos Para Todos.](#)



## PROBLEMAS EM IPHONES E IPADS

iOS é o sistema operacional da Apple's que equipa iPhones, iPod Touches, e iPads. iOS 3.2 tem um número grande de problemas com vídeo HTML5.

1. iOS não irá reconhecer o vídeo se você incluir um atributo `poster`. O atributo `poster` do elemento `<video>` permite exibir uma imagem personalizada enquanto o vídeo é carregado, ou até que o usuário pressione o “play.” Esse bug é consertado no iOS 4.0, mas irá levar tempo para usuários atualizarem-se.
2. Se você tem múltiplos elementos `<source>`, iOS irá reconhecer qualquer coisa menos o primeiro. Já que dispositivos iOS suportam apenas H.264+AAC+MP4, isso é efetivamente significa que você precisa listar seu MP4 primeiro. Esse bug também é corrigido no iOS 4.0.



## PROBLEMAS EM DISPOSITIVOS ANDROID

Android é o sistema operacional da Google's que equipa diferentes telefones e dispositivos de mão. Versões do Android antes da 2.3 tinham uma série de problemas com vídeo HTML5.

1. O atributo `type` nos elementos `<source>` confundem muito o Android. A única forma para reconhecer uma origem de vídeo é, ironicamente, omitindo o atributo `type` e garantir que arquivos de vídeo H.264+AAC+MP4 terminem com o formato `.mp4`. Você pode ainda incluir o atributo `type` nas outras origens de vídeo, já que H.264 é o único formato de vídeo que Android 2.2 suportam. (Esse bug é corrigido no Android 2.3.)

2. O atributo `controls` não foi suportado. Não há efeitos problemáticos nisso, mas o Android não irá mostrar qualquer controle de interface para um vídeo. Você irá precisar do seu próprio controle na interface. No mínimo, você deveria prover um script que comece tocando o vídeo quando o usuário clicar no vídeo. Esse bug também é corrigido no Android 2.3.



## UM COMPLETO E VIVO EXEMPLO

Aqui está um exemplo vivo de um vídeo que usa essas técnicas. Eu extendi o código do “Vídeo para Todos” para incluir suporte a vídeo WebM. Codifiquei a mesma origem do vídeo em três formatos, com esses comandos:

```
## Theora/Vorbis/Ogg
```

```
you@localhost$ ffmpeg2theora --videobitrate 200 --max_size 320x240 --output pr6.ogv pr6.dv
```

```
## H.264/AAC/MP4
```

```
you@localhost$ HandBrakeCLI --preset "iPhone & iPod Touch" --vb 200 --width 320 --two-pass --turbo -  
-optimize --input pr6.dv --output pr6.mp4
```

```
## VP8/Vorbis/WebM
```

```
you@localhost$ ffmpeg -pass 1 -passlogfile pr6.dv -threads 16 -keyint_min 0 -g 250 -skip_threshold
```

```
0 -qmin 1 -qmax 51 -i pr6.dv -vcodec libvpx -b 204800 -s 320x240 -aspect 4:3 -an -f webm -y NUL
you@localhost$ ffmpeg -pass 2 -passlogfile pr6.dv -threads 16 -keyint_min 0 -g 250 -skip_threshold
0 -qmin 1 -qmax 51 -i pr6.dv -vcodec libvpx -b 204800 -s 320x240 -aspect 4:3 -acodec libvorbis -ac 2
-y pr6.webm
```

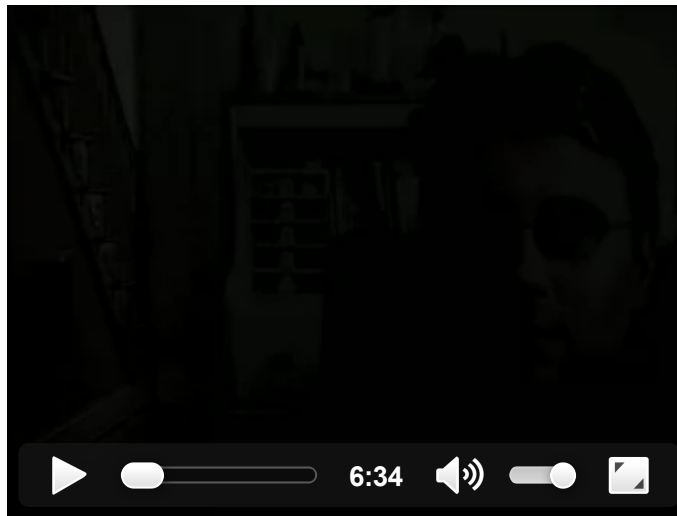
A marcação final usa o elemento <video> para HTML5, um elemento <object> dentro para fallback em Flash, e um pequeno pedaço de script para beneficiar dispositivos Android:

```
<video id="movie" width="320" height="240" preload controls>
  <source src="pr6.webm" type='video/webm; codecs="vp8, vorbis"' />
  <source src="pr6.ogv" type='video/ogg; codecs="theora, vorbis"' />
  <source src="pr6.mp4" />
  <object width="320" height="240" type="application/x-shockwave-flash"
    data="flowplayer-3.2.1.swf">
    <param name="movie" value="flowplayer-3.2.1.swf" />
    <param name="allowfullscreen" value="true" />
    <param name="flashvars" value='config={"clip": {"url": "http://wearehugh.com/dih5/pr6.mp4",
"autoPlay":false, "autoBuffering":true}}' />
    <p>Download video as <a href="pr6.mp4">MP4</a>, <a href="pr6.webm">WebM</a>, or <a
href="pr6.ogv">Ogg</a>.</p>
  </object>
</video>
<script>
  var v = document.getElementById("movie");
```



```
v.onclick = function() {  
  if (v.paused) {  
    v.play();  
  } else {  
    v.pause();  
  }  
};  
</script>
```

Com essa combinação de HTML5 e Flash, você deve conseguir assistir esse vídeo em quase todos os navegadores e dispositivos:



# LEITURA COMPLEMENTAR

- [HTML5: The <video> element](#)
- [Video for Everybody](#)
- [A gentle introduction to video encoding](#)
- [Theora 1.1 is released — what you need to know](#)
- [Configuring servers for Ogg media](#)
- [Encoding with the x264 codec](#)
- [Video type parameters](#)
- [Everything you need to know about HTML5 audio and video](#)
- [Making HTML5 video work on Android phones. \*Le sigh.\*](#)
- [Internet Explorer 9 Guide for Developers: HTML5 video and audio elements](#)

Controles customizados para vídeo HTML5:

- [VideoJS](#)
- [MediaElement.js](#)
- [Kaltura HTML5 Video & Media JavaScript Library](#)



Este foi o “Vídeo na Web”. Consulte o [Sumário](#), caso queira continuar com a leitura.

## VOCE SABIA?

Em associação com o Google Press, O'Reilly está distribuindo este livro em uma variedade de formatos, incluindo papel, ePub, Mobi, e DRM-de graça PDF. A edição paga é chamada de “HTML5: Up & Running,” e está disponível agora. Este capítulo está incluído na edição paga.

Se você gostou deste capítulo e quer demonstrar sua apreciação, você pode [comprar “HTML5: Up & Running” com este link afiliado](#) ou [comprar a edição eletrônica diretamente da O'Reilly](#). Você terá um livro, e eu um dinheirinho. Eu não estou aceitando doações diretas.



Copyright MMIX–MMXI [Mark Pilgrim](#)

powered by Google™

Search