TELESTREAM

Users Guide 5.3

Note on License

The accompanying Software is licensed and may not be distributed without written permission.

Disclaimer

The contents of this document are subject to revision without notice due to continued progress in methodology, design, and manufacturing. Telestream shall have no liability for any error or damages of any kind resulting from the use of this document and/or software.

The Software may contain errors and is not designed or intended for use in on-line facilities, aircraft navigation or communications systems, air traffic control, direct life support machines, or weapons systems ("High Risk Activities") in which the failure of the Software would lead directly to death, personal injury or severe physical or environmental damage. You represent and warrant to Telestream that you will not use, distribute, or license the Software for High Risk Activities.

Export Regulations. Software, including technical data, is subject to Swedish export control laws, and its associated regulations, and may be subject to export or import regulations in other countries. You agree to comply strictly with all such regulations and acknowledge that you have the responsibility to obtain licenses to export, re-export, or import Software.

Copyright Statement

This document is part of the software product and, as such, is part of the license agreement governing the software. So are any other parts of the software product, such as packaging and distribution media.

The information in this document may be changed without prior notice and does not represent a commitment on the part of Telestream.

Trademarks and Patents

- Episode is a registered trademark of Telestream, Inc.
- UNIX is a registered trademark of UNIX System Laboratories, Inc.
- Apple is a trademark of Apple Computer, Inc., registered in the U.S. and other countries.
- QuickTime is a trademark of Apple Computer, Inc., registered in the U.S. and other countries.
- Windows Media is a trademark of Microsoft Inc., registered in the U.S. and other countries.
- RealNetworks, RealAudio, and RealVideo are either registered trademarks or trademarks of RealNetworks, Inc. in the United States and/or other countries.

All other trademarks are the property of their respective owners.

MPEG-4 AAC

"Supply of this Implementation of MPEG-4 AAC technology does not convey a license nor imply any right to use this Implementation in any finished end-user or ready-to-use final product. An independent license for such use is required."

MP3

This software contains code from LAME, `http://lame.sourceforge.net/`. "Supply of this product does not convey a license nor imply any right to distribute content created with this product in revenue-generating broadcast systems (terrestrial, satellite, cable and/or other networks.), streaming applications (via Internet, Intranets, and/or other networks), other content distribution systems (pay audio or audio-on-demand applications and the like) or on physical media (compact discs, digital versatile discs, semiconductor chips, hard drives, memory cards and the like). An independent license for such use is required. For details, please visit `http://mp3licensing.com/`."

OGG Vorbis

This software contains code that is ©2010, Xiph.Org Foundation. "THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE FOUNDATION OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,

DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."

PCRE

PCRE is a library of functions to support regular expressions whose syntax and semantics are as close as possible to those of the Perl 5 language.

Release 7 of PCRE is distributed under the terms of the "BSD" licence, as specified below. The documentation for PCRE, supplied in the "doc" directory, is distributed under the same terms as the software itself.

The basic library functions are written in C and are freestanding. Also included in the distribution is a set of C++ wrapper functions.

The basic library functions

Written by:	Philip Hazel
Email local part:	`ph10`
Email domain:	`cam.ac.uk`

University of Cambridge Computing Service, Cambridge, England.

The C++ wrapper functions

Contributed by: Google Inc.

The "BSD" licence

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the University of Cambridge nor the name of Google Inc. nor the names of their contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Disclaimer of Warranty on Software

You expressly acknowledge and agree that use of the Software is at your sole risk. The Software and related documentation are provided "AS IS" and without warranty of any kind and Licensor and the third party suppliers EXPRESSLY DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. NEITHER LICENSOR NOR ANY THIRD PARTY SUPPLIER WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE. FURTHERMORE, THE TERMS OF THIS DISCLAIMER AND LIMITATION OF LIABILITY BELOW DO NOT AFFECT OR PREJUDICE THE STATUTORY RIGHTS OF A CONSUMER ACQUIRING THE SOFTWARE OTHERWISE THAN IN THE COURSE OF A BUSINESS, NEITHER DO THEY LIMIT OR EXCLUDE ANY LIABILITY FOR DEATH OR PERSONAL INJURY CAUSED BY NEGLIGENCE.

Limitation of Liability

LICENSOR AND THE THIRD PARTY SUPPLIERS EXPRESSLY DISCLAIMS ALL LIABILITY FOR DAMAGES, WHATEVER THEIR CAUSE, INCLUDING DIRECT OR INDIRECT DAMAGE, SUCH AS CONSEQUENTIAL OR BUSINESS DAMAGE, AMONGST OTHERS CAUSED BY THE NON-FUNCTIONING OR MALFUNCTIONING OF THE SOFTWARE. SHOULD LICENSOR OR THE THIRD PARTY SUPPLIERS IN ANY WAY BE LIABLE FOR DAMAGES, EITHER AS PER THE TERMS OF THIS LICENSE OR OTHERWISE, THEN THIS LIABILITY WILL IN NO EVENT EXCEED THE AMOUNT PAID BY YOU FOR THE SOFTWARE. SOME JURISDICTIONS DO NOT ALLOW THE LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THIS LIMITATION MAY NOT APPLY TO YOU.

Contents

Document conventions

NOTE

Paragraphs marked like this highlight items of particular importance for the proper function of the software.

TIP

Paragraphs marked like this highlight procedures that can save time or produce particularly good results.

Paragraphs marked like this warn about features which may cause loss of data or failed execution if used incorrectly.

Document references, both internal and external, are shown in italics. Example:
See chapter 2 *Before You Install.*

Literature references are given as numbers in brackets with the full reference in the Bibliography. Example:
See [2].

Directory names, file names, code examples, and prompts, are shown in plain typewriter type. Example:
The file `printer.ppd` can be found in `/etc/cups/ppd/`.

The names of interface components are given in **bold**. Example:
Adjust the time limit with the **Time limit** slider. Select **Quit** from the **Episode Engine** drop-down menu.

Keys to be pressed on the keyboard are displayed in bold typewriter type. Example:
Press **`Return`** to select the GUI installation.

Examples of extended dialogue will include the shell `prompt>` .

Command syntax is described in Backus-Naur form.

Copy-pasting from the manual is not guaranteed to work, as the text contains formatting information which may not be accepted by the target application.

1 Using **Episode Engine**

So, you (or someone else) just finished installing **Episode Engine** and you are eager to start using it. This manual will tell you how to do it. We begin with a very quick overview to get you started, and then more fully explain day-to-day use of **Episode Engine**. The next chapters shows how to use the **Engine Admin** client to control and monitor execution, how to integrate **Episode Engine** and **Final Cut Server**, and how to integrate **Episode Engine** and **Pipeline**. Finally we will magnify all the fine print and give all the details of how things work.

1.1 What happens?

For a transcoding to take place, you need at least one source file and one *settings file*. The source file is whatever input you have, video and/or audio. The settings file is created with the companion product **Episode Encoder** and defines how the source file should be transformed into the output file.

Episode Engine can be set up to monitor any number of data sources, so that when a media file is placed in one of these sources, **Episode Engine** will automatically start the transcoding process. The resulting output file is placed in an output folder from which you can retrieve it or let a script move it to its final destination.

1.2 Once more, with details

1.2.1 Settings files

You can use any of the number of template settings supplied with **Episode Engine**, which cover most of the usual output formats, or you can use **Episode Encoder** to develop specialised settings for your particular needs. If you want, you can make very complex and fine-tuned settings, the art of which we cannot cover here, but instead refer you to the ***Episode Encoder*** *User Guide* for the details. Note that mostly settings files do not depend on the input material, so you can use the same settings file for many different input formats. Each settings file will generate a specific type of output.

If you are using watch folders (see below) you copy your settings files to the watch folders where they are to be used, but in the general case, you upload settings files to the shared settings area on the **Episode Engine** server, where they also can be used by others transcoding on the same server.

1.2.2 Input monitoring

Episode Engine uses *input monitoring* to find media source files. Typically **Episode Engine** continuously checks the contents of a folder, on local disk, shared storage, or on a remote file server, but you can also set up monitoring of a hardware device, such as a digital video camera.

The classical method of input monitoring is *watch folders*. Media files placed in watch folders that also contain settings files will be transcoded. The location of your watch folders was set up during installation, by default the watch folders are the subfolders of `/Users/Shared/Episode Engine/Input/`.

A more general method is *File Monitors* with which you can designate any arbitrary folder to be monitored for media files. When setting up the File Monitor in **Engine Admin** you define what settings files to use for transcoding the files in that folder. Note that you cannot use File Monitors on watch folders, as the methods conflict.

In a similar manner you can use the FTP and SMB Monitors, which monitor folders located on **ftp** and SMB servers, respectively.

Watch folders have some important differences in behaviour to File Monitors, these differences will be pointed out as needed.

Best practice A good way to organise things is to create separate folders for the different types of transcodings you need to perform. Consider a production environment for three different customers: One has a website with continously updated video clips in three different formats, one generates podcasts for the iPod, one burns Video CDs and DVDs. You can either set this up so that you monitor their **ftp** servers and retrieve files as they are created, or create separate accounts for the users where they can log in and deposit their material. In each case, the customers only ever see their own material and you can set the priorities of the input monitors so that customers with stricter time limits get treated before those with less urgent requirements. Event scripts (see section 4.7, *Event scripts*) can then be used to transfer the finished output files to their final destinations.

1.2.3 Transcoding

A source file will be transcoded into as many output files as there are settings files associated with the input monitor. The output files will be placed in a folder that by default will be located under `/Users/Shared/Episode Engine/`. Files generated from sources in watch folders will be placed in a folder with the same name as the input watch folder under `/Users/Shared/Episode Engine/Output/`, whereas files from input monitors will be placed in a folder of your choosing.

The name of the output file is the name of the input file concatenated with the name of the settings file and the file type extension.

By default source files in a watch folder will be deleted once they have been transcoded with all associated settings. To retain your source files, you need to set up *archiving*, in which case they are moved to an archive folder (by default in

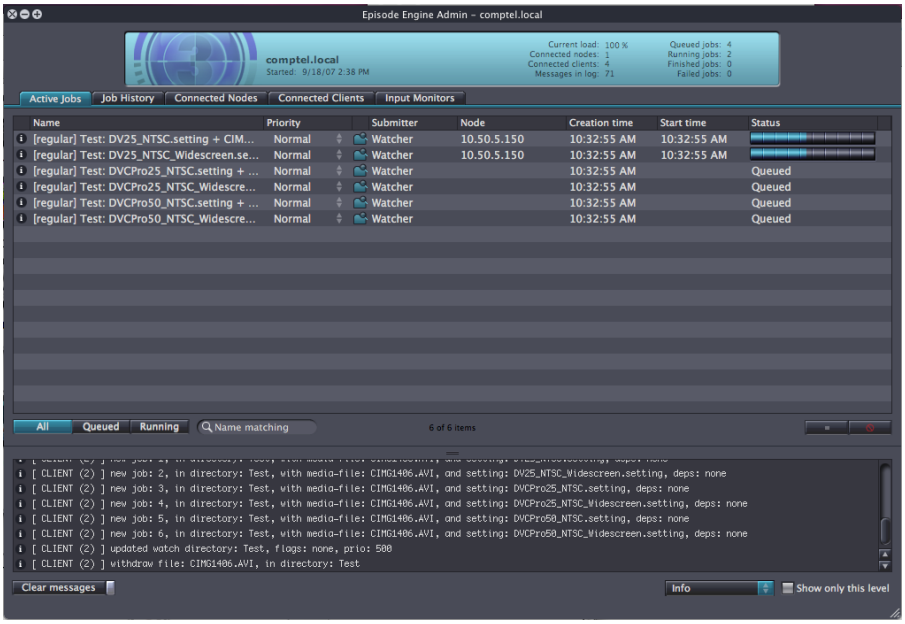
`/Users/Shared/Episode Engine/Archive/`) after they have been transcoded. (See the ***Episode Engine*** *Admin Guide* for how to set up archiving.) Other input monitors will not delete their source files.

1.3 Monitoring transcoding

In order to monitor the progress of transcoding jobs **Episode Engine** has a client called **Engine Admin**. In a default installation it is in the `Applications` folder.

On startup you must connect to a named server, since in a large installation you might have multiple instances of **Episode Engine** running. **Engine Admin** will display all servers on your local network, but you can also give the address of a server anywhere on the Internet. Of course you have to supply the password to the server you connect to.

Once you are connected to a server, you can monitor the health of jobs, clients (the processes involved in transcoding) and the computing nodes. You can also create input monitors.



1.4 Advanced features

You can get more out of your installation with the advanced features **Episode Engine** offers, by adding more information to your output files and by post-processing them once transcoding is done.

1.4.1 Watermarks

Watermarks are images added to the video to indicate origin, enforce copyright etc. A settings file can specify that a watermark be added to a video. In this

case, transcoding will not start unless the required watermark file is present in the monitored folder.

Watermark files can be designated not to be automatically deleted from a watch folder.

1.4.2 Bumpers and trailers

Bumpers and *trailers*, also called *intros* and *outros*, are short clips added respectively before and after your main material. These may be station signatures, credit rolls and similar material. You can add bumper and trailer clips to folders in the same manner as watermark files.

Bumpers and trailers can be designated not to be automatically deleted from a watch folder.

1.4.3 Metadata

Metadata are data about a media file, author, copyright date, etc. Most output formats support at least some metadata fields.

Metadata that should be applied to *all* source files should be defined in a settings file, but more often one wants to supply metadata on a per-file basis. To do this you create XML files with the same name as the source file and the extension `.inmeta`. You can use **Episode Encoder** to do this. If an output format does not support a given metadata field, it will instead be stored in an XML file with the same name as the the output file and the extension `.meta`.

To use `.inmeta` files, this must have been specified in the settings file. If this has been specified, transcoding will not start unless the required metadata file is present in the monitored folder.

`.inmeta` files are deleted from watch folders when transcoding is finished.

1.4.4 Scripts

Episode Engine can be extended by scripts in two ways.

Event scripts take actions when jobs finish or generate error messages, or when nodes fail. The scripts can be written in any language of your choice, but they are stored in a single directory, so you cannot write scripts that are specific to a given input or output folder; rather you should use the environment variables that are made available to the script to select the action appropriate to the file in question.

Event scripts are restricted to reacting to events, but the **Episode Engine** Software Development Kit lets you write scripts to submit jobs, monitor jobs, list storage depots, and manage settings files, so that you could, e g, replace the **Engine Admin** with an application of your own, precisely adapted to the needs of your organisation.

1.4.5 Split-and-stitch

Episode Engine Pro

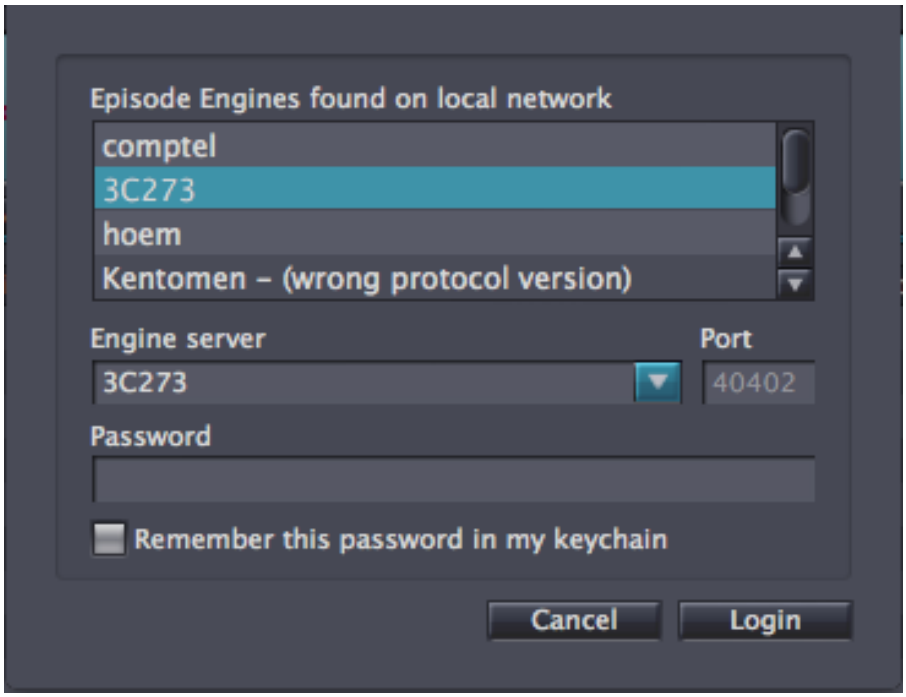
If you have the **Episode Engine Pro** version, you can run in split-and-stitch mode. This means that source files are split into several parts, each part assigned to a different cpu, transcoded in parallel and the resulting parts stitched together into a single output file.

If you use watch folders you indicate that files are to be transcoded with split-and-stitch by giving the watch folder a name that starts with `stitch` (not case sensitive). If you use input monitors, you check the **Split n' Stitch** checkbox in the input monitor window.

2 Engine Admin

2.1 Connecting

Starting **Engine Admin** brings up a window with a list of **Episode Engine** servers visible through the Bonjour service. Either select one of these or type a name or IP address of a server in the combo box **Engine server**. The combo box will remember your latest connections. After selecting a server, enter the appropriate password and press **Login** to connect.

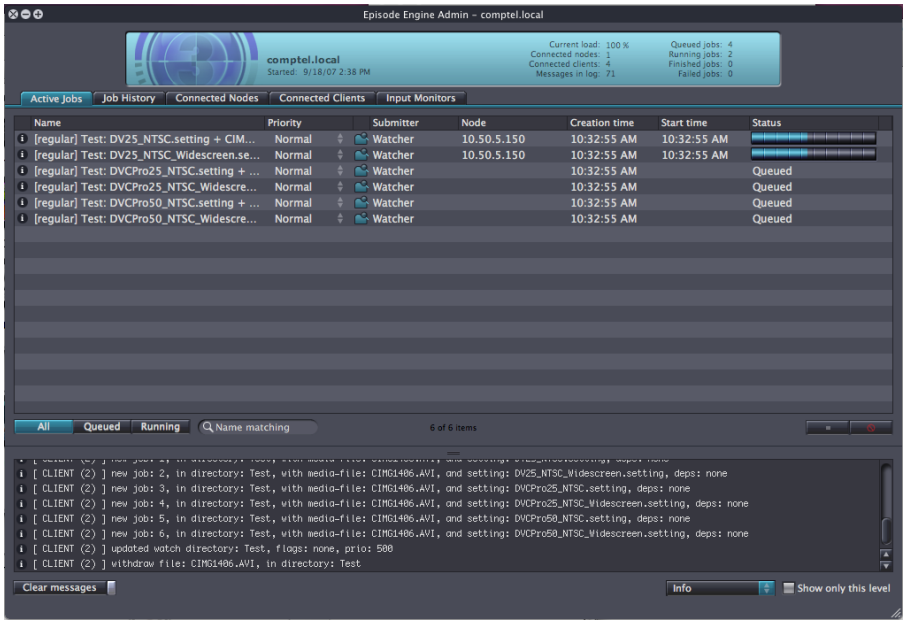


Check **Remember this password in my keychain** to let the Apple Keychain remember your password. For servers on the local network the correct port number is automatically determined, for external servers you can enter a port number if it is different from the default one.

The username is currently not modifiable.

You can connect to additional servers with **File→New Connection** (`Command-N`) and disconnect from a server with **File→Disconnect** (`Command-D`).

2.2 Active jobs

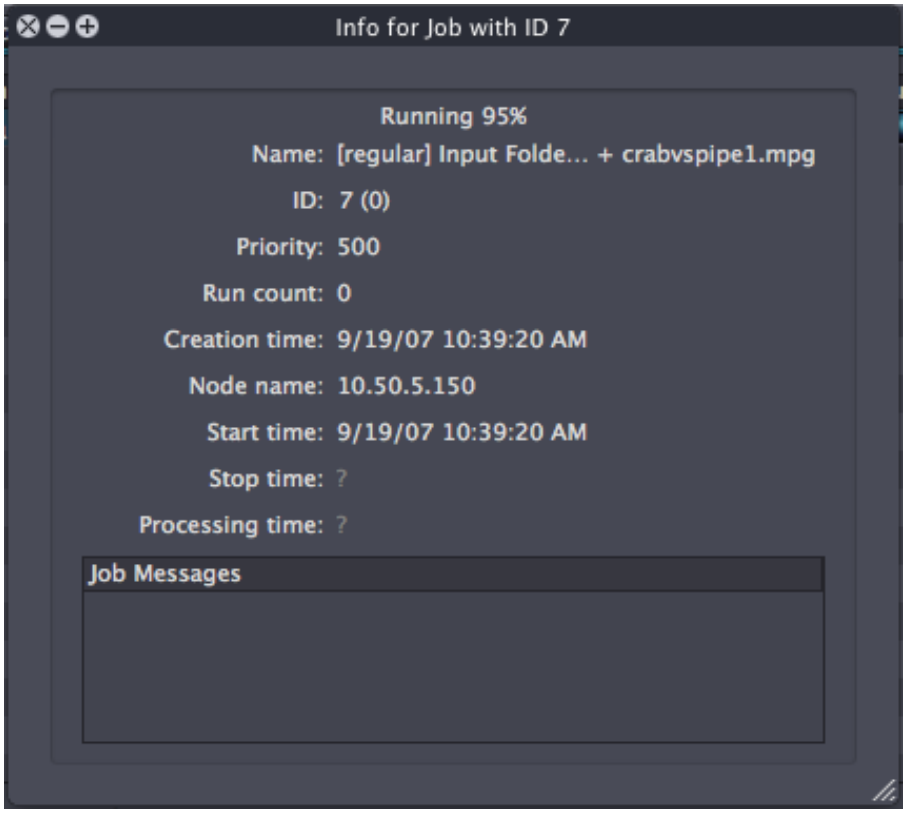


The **Active Jobs** tab shows the transcoding jobs currently underway or in queue. The three buttons **All**, **Queued** and **Running** select whether to see all at a time or just one category. Entering a string in the search field next to the buttons limits the display to jobs matching the string.

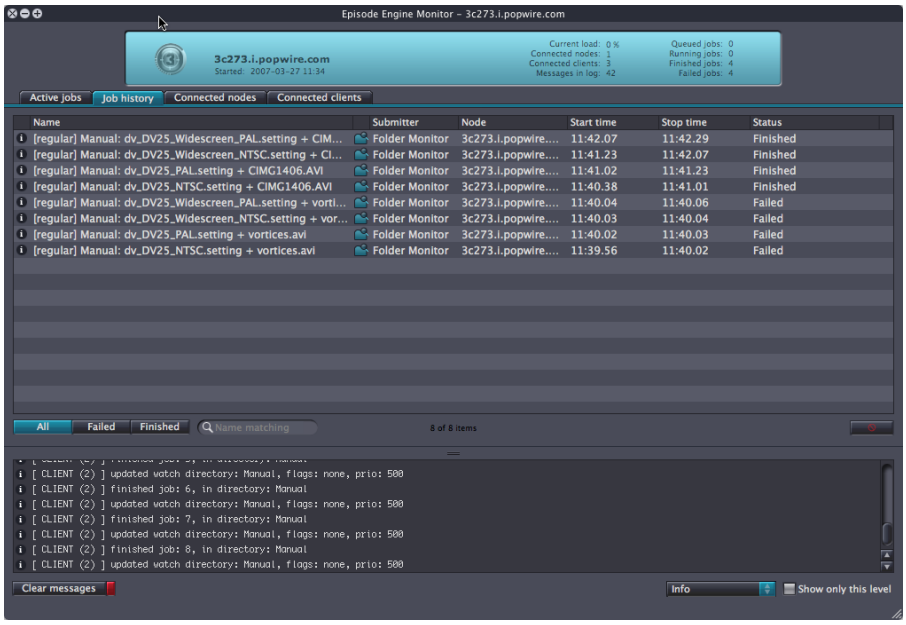
The column **Priority** shows the current priority of each job. You can adjust this priority by clicking on the priority value, which brings up a menu of symbolic priority values, corresponding to numeric values as follows: **Lowest** = 0, **Low** = 250, **Normal** = 500, **High** = 750, **Highest** = 65535. You can also select a job row and choose a priority from the **Jobs** menu.

The context menu on the job row also lets you set the priority. You can **Abort (Delete)** a job or **Stop And Requeue** it.

Each job row starts with an **Info** button ( ). Clicking on it brings up a window with detailed information on the job.



2.3 Job history

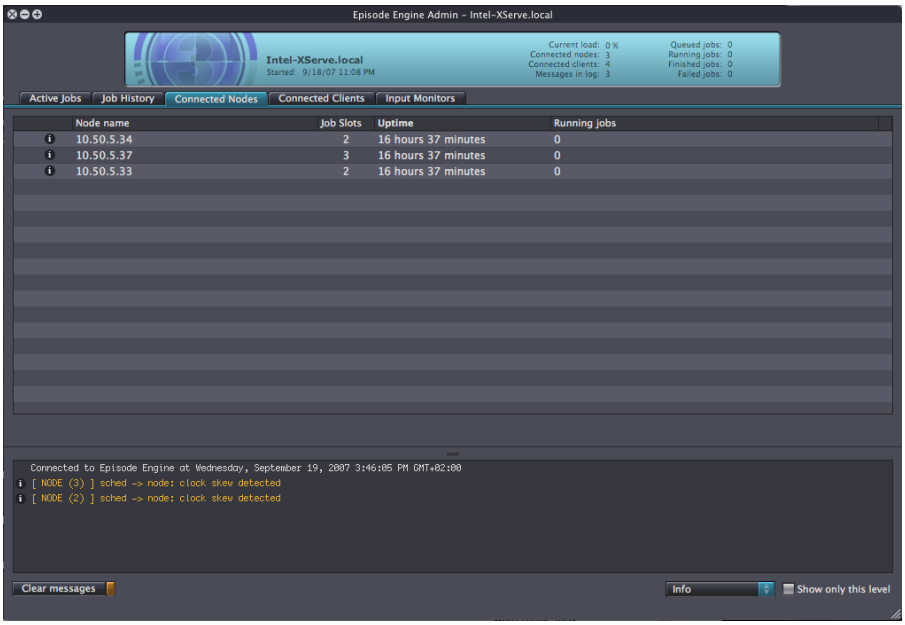


The **Job History** tab shows jobs no longer executing, either because they finished in good order or because they failed in some way. They can also be selected by category with the buttons **All**, **Failed**, and **Finished**. Entering a string in the

search field by the buttons limits the display to jobs matching the string. Select **Jobs→Clear Job History** or press **`Command-Alt-Backspace`** to remove the entries in the list. Select a job and choose **Remove** in the context menu or press the **Delete** button ( ) to remove individual jobs. The history list is automatically purged after a time period set in the `engine.conf` file, by default 24 hours.

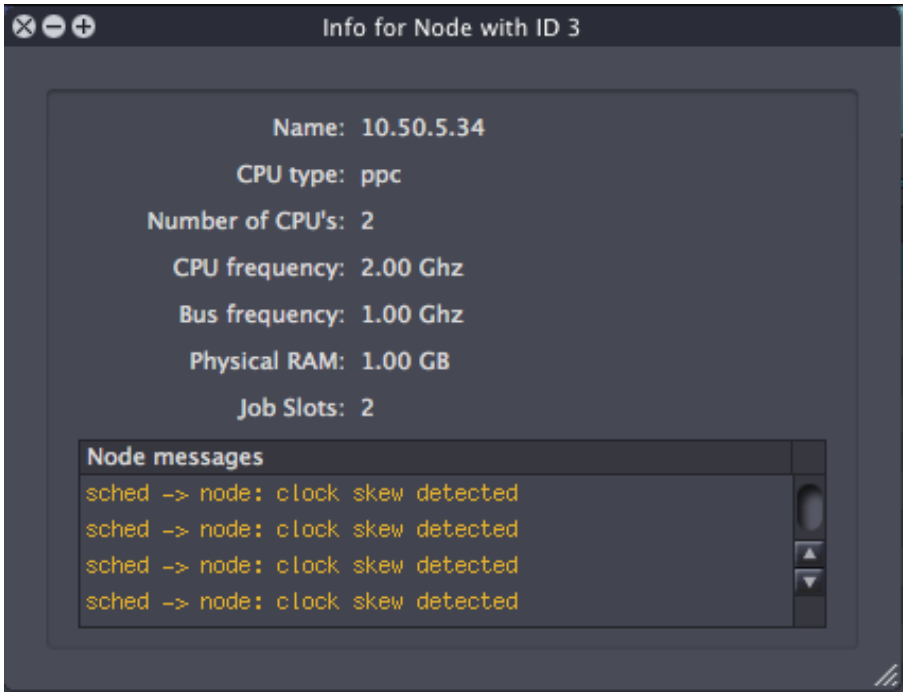
The **Info** button ( ) gives the same information as for active jobs.

2.4 Connected nodes

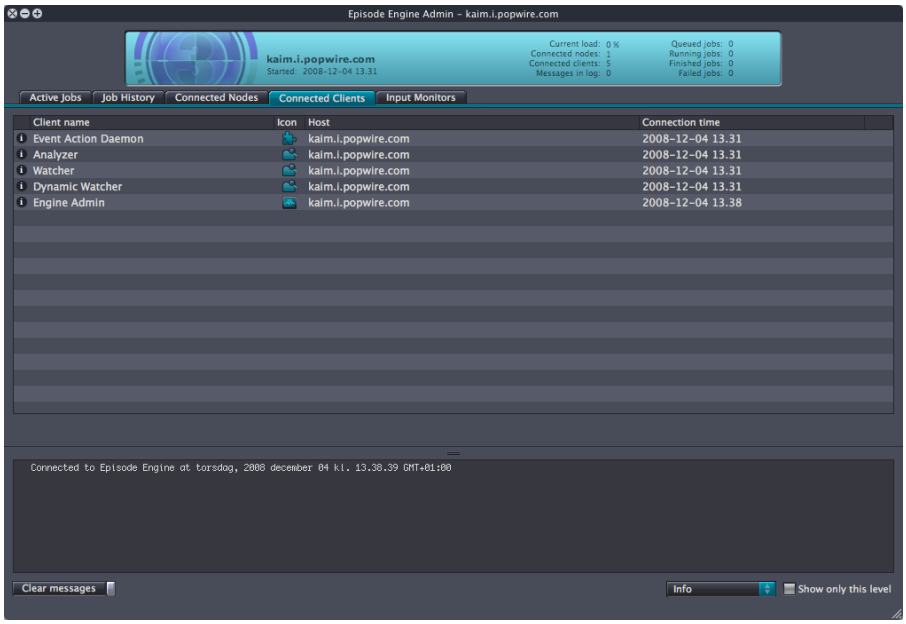


The **Connected Nodes** tab shows the transcoding nodes connected to this server process, their uptime and the number of running jobs on each.

Press the **info** icon ( ) to get a window with more information on the node and an event log for that node.

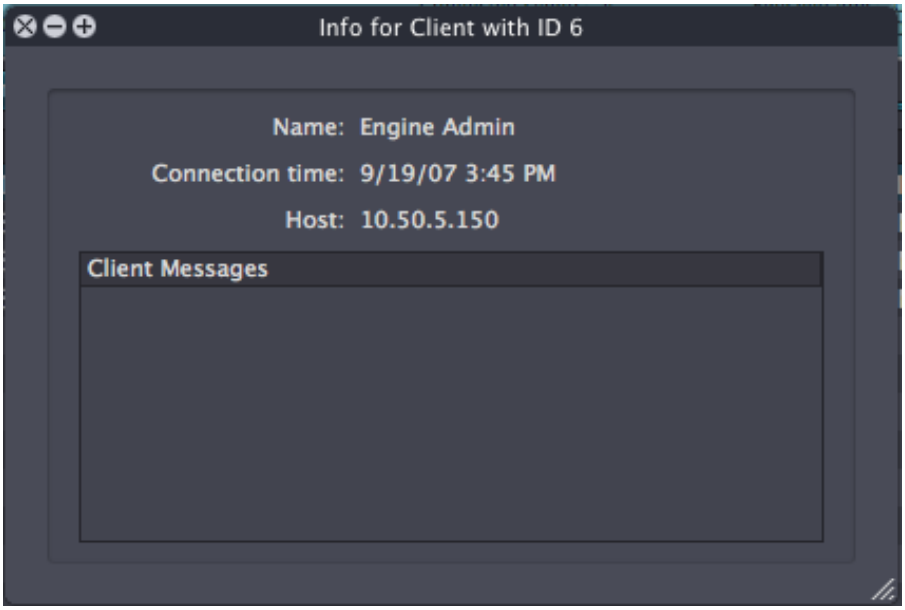


2.5 Connected clients

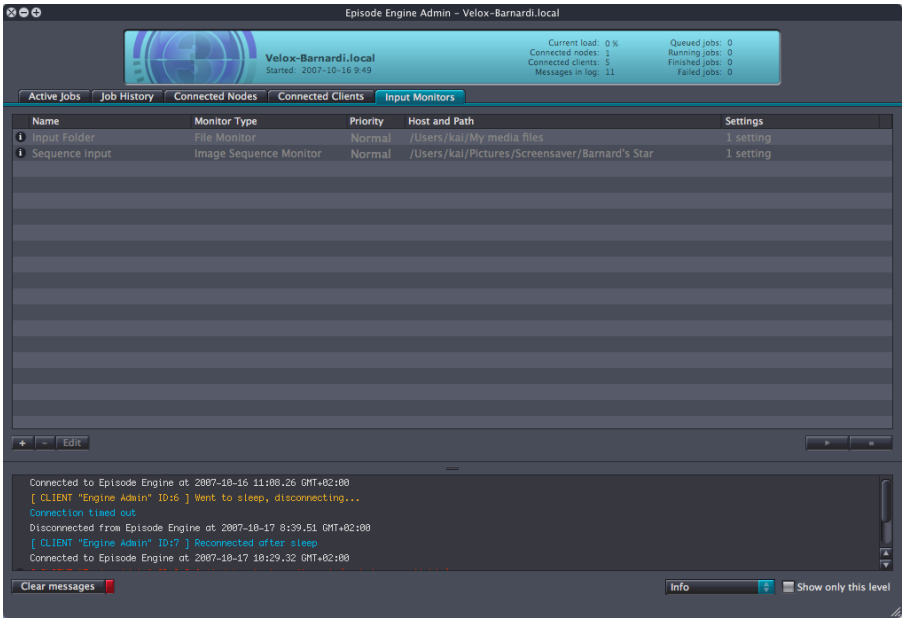


The **Connected Clients** tab shows all client processes connected to this server. One of these will always be your own **Engine Admin** process, **Event Action Daemon**, **Analyzer**, **Watcher**, and **Dynamic Watcher** will also always be visible.

Press the **info** icon ( ) to get a window with information on and from the client processes and an event log for that node.



2.6 Input Monitors

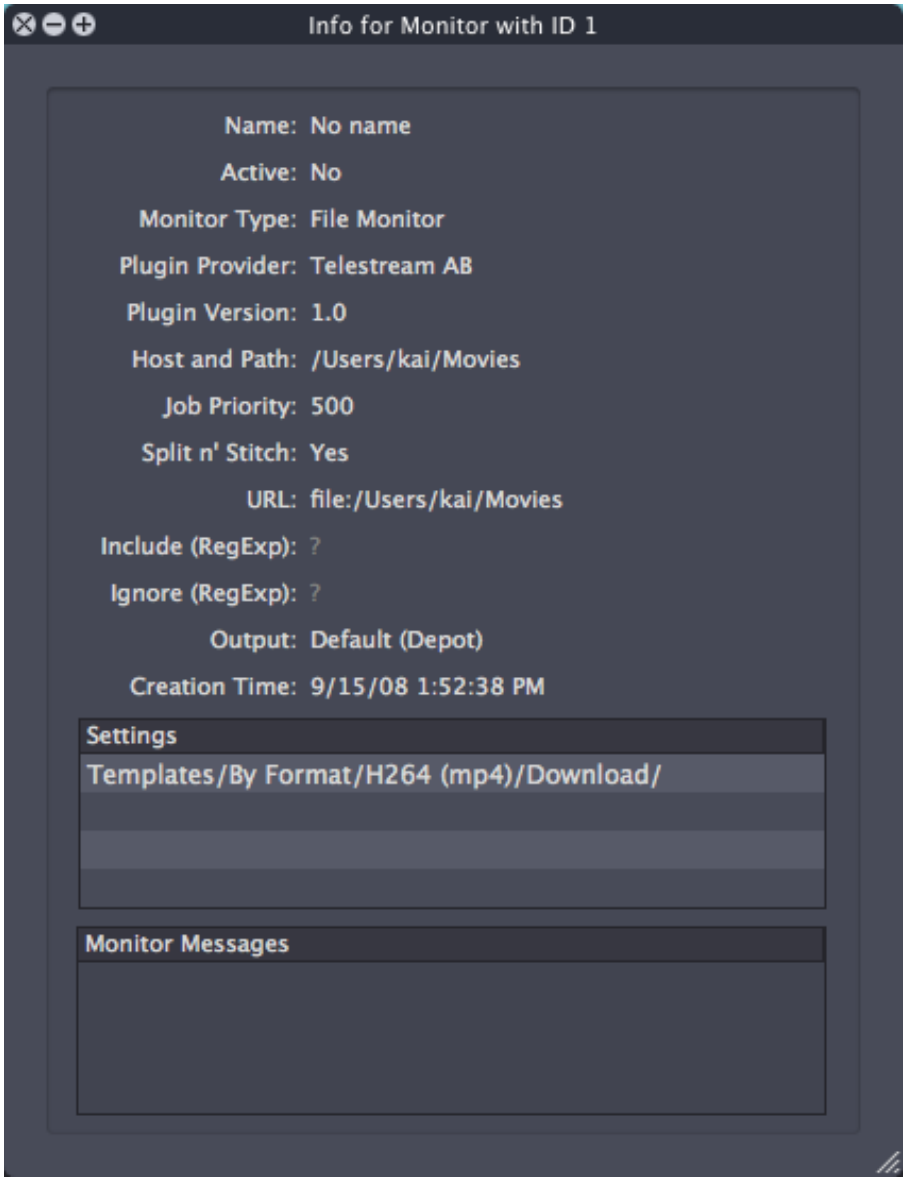


The **Input Monitors** tab shows all input monitors on this server. You can press the **+** button or select **Input Monitors→Add New Monitor** to create a new input monitor.

You can edit an inactive input monitor by double-clicking the monitor row, pressing the **Edit** button, selecting **Edit Monitor** in the context menu, or selecting **Input Monitors→Edit Monitor**.

You can bring up an info window on an input monitor by clicking the **Info** button (ⓘ), pressing **`Command-I`**, selecting **Input Monitors→Get Info**, or selecting

Get Info in the context menu.



For complete information on all the different types of input monitors and how they can be configured, see section 4.2, *Input monitors*.

2.7 Message Log

At the bottom of the window is the **Message log** showing information on the processing. Only messages starting from when you connected to your current server will be shown. Next to the **Clear messages** button is a light showing the highest severity of messages in the log.

Clicking the checkbox **Show only this level** on the right, you can use the menu to filter messages according to severity level: **Debug**, **Info**, **Notice**, **Warning**, **Error**, **Critical**, **Alert** or **Emergency**.

Clear the message log as well as job info and client info logs by pressing the button **Clear messages** or selecting **Message Log→Clear Messages**.

3 Integrating **Episode Engine** and **Final Cut Server**

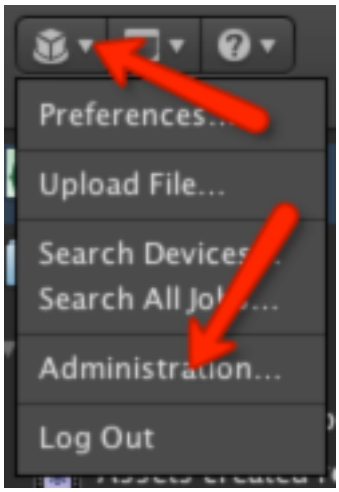
You can use **Episode Engine** as a transcoding backend to **Final Cut Server**, extending the range of output formats available to **FCS** and increasing the transcoding speed.

3.1 Setup

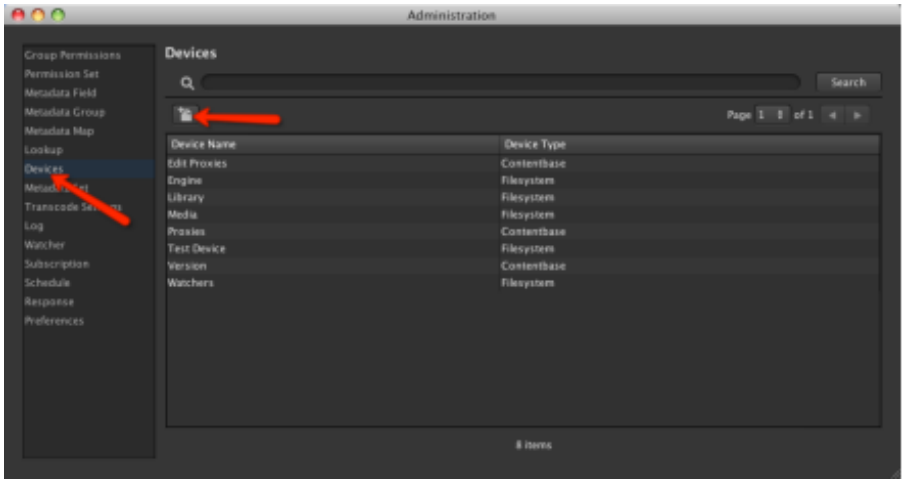
We will set up **FCS** to place files to be encoded in a **Episode Engine** watch folder.

Make sure that **FCS** can write in the **Episode Engine** watch folders.

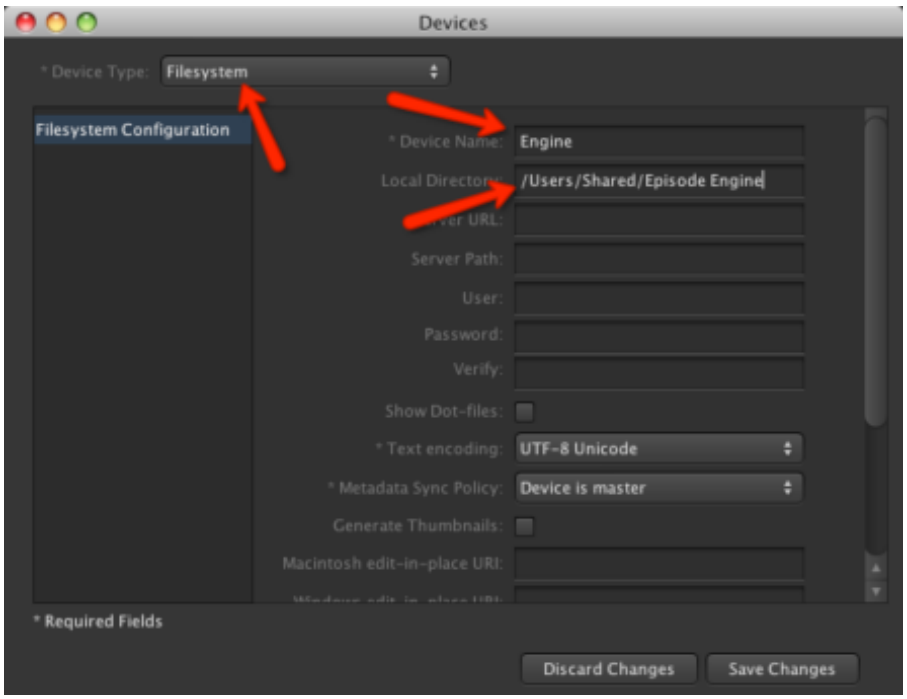
Log in to Final Cut Server as an administrator.



Set up **Episode Engine** as a rendering device by selecting **Devices** in the leftmost column of the window and then clicking the **New Device** button.

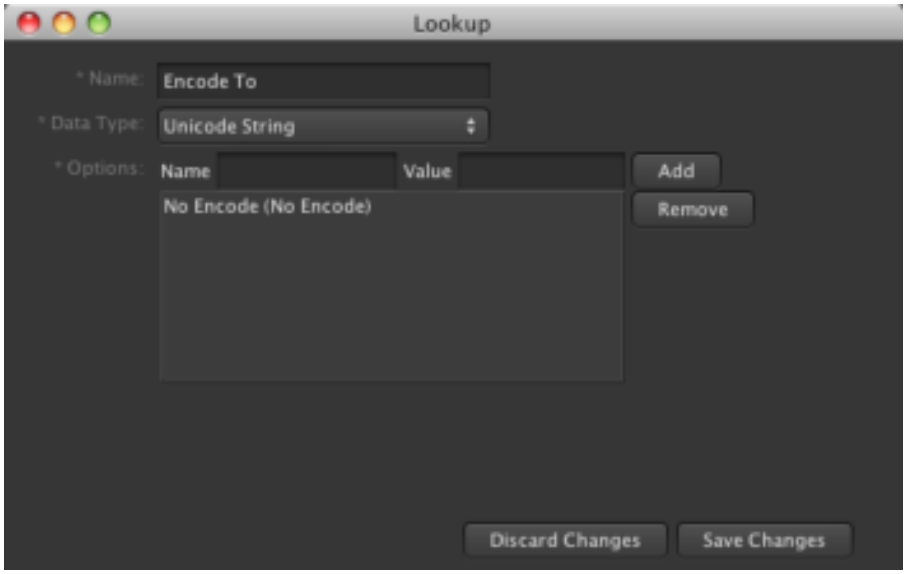


Set the **Device Type** to **Filesystem**, **Device Name** to `Engine`, and **Local Directory** to the **Episode Engine** watch folder root, by default `/Users/Shared/Episode Engine/`. Click **Save Changes** to save your device.

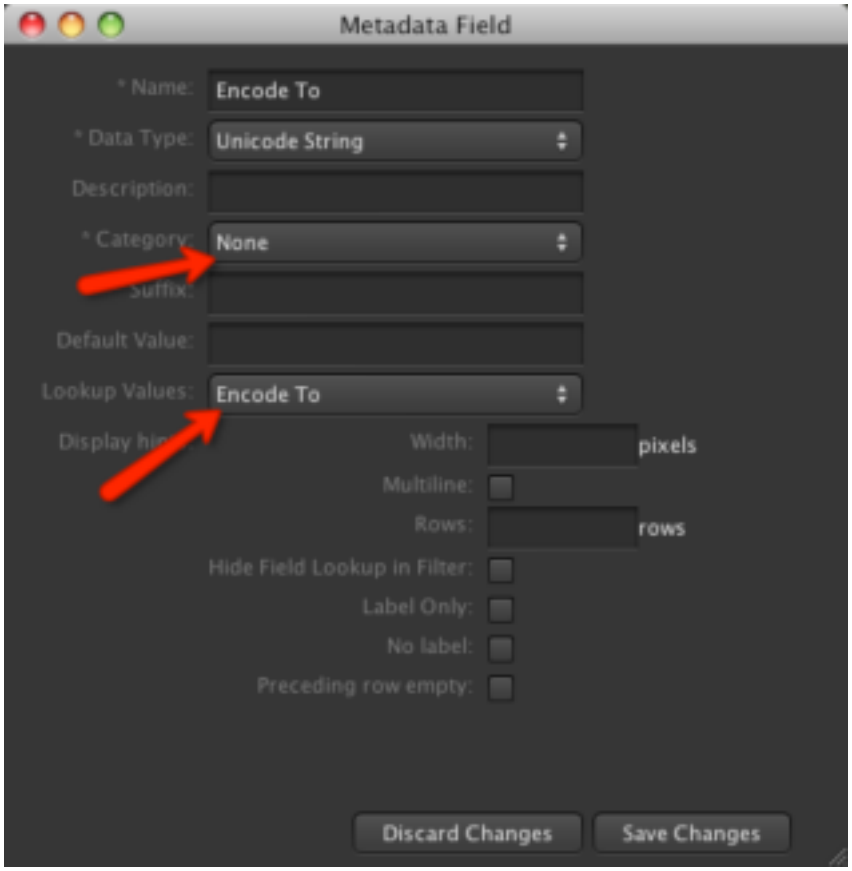


Set up references to the watch folders by clicking on **Lookup** in the leftmost column of the **Administration** window. Click on the **New lookup** button to bring up a **Lookup** window. Set the **Name** to **`Encode To`** and the **Options** to **`No Encode`**.

Click **Save Changes** to save your lookup.

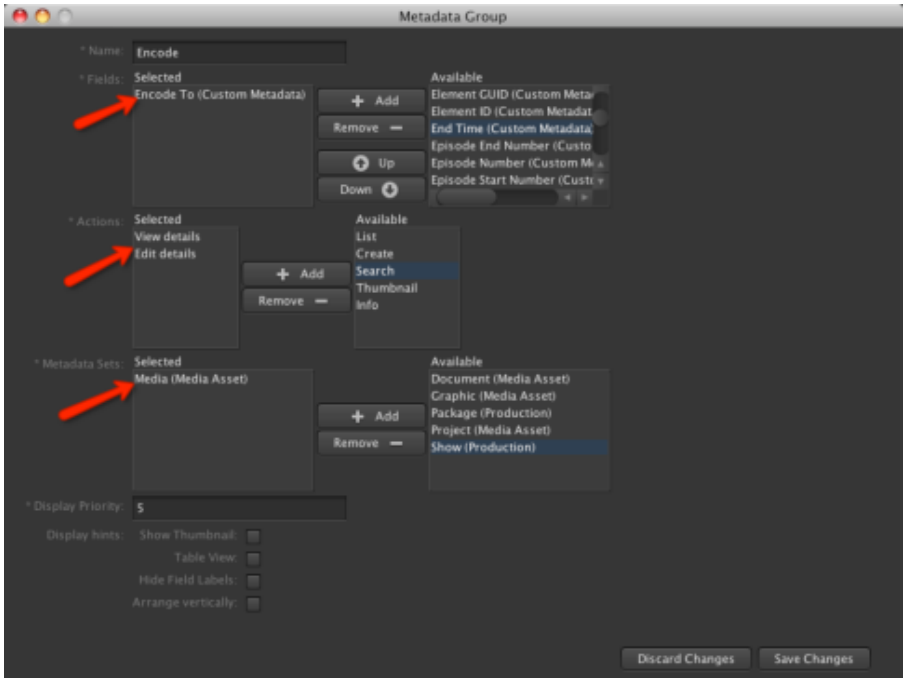


Set up encodings to use the watch folders by clicking on **Metadata Fields** in the leftmost column of the **Administration** window. Click on the **New Metadata Field** button to bring up a **Metadata Field** window. Set the **Name** to **`Encode To`** and **Category** to **None**. Set **Lookup Values** to **Encode To**, i e the lookup you just created.

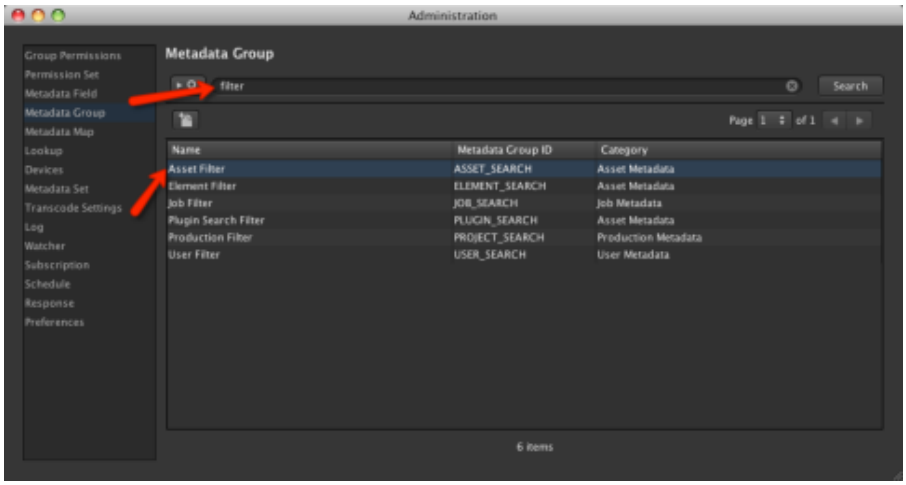


Set up a new metadata group by clicking on **Metadata Group** in the leftmost column of the **Administration** window. Click on the **New Metadata Group**

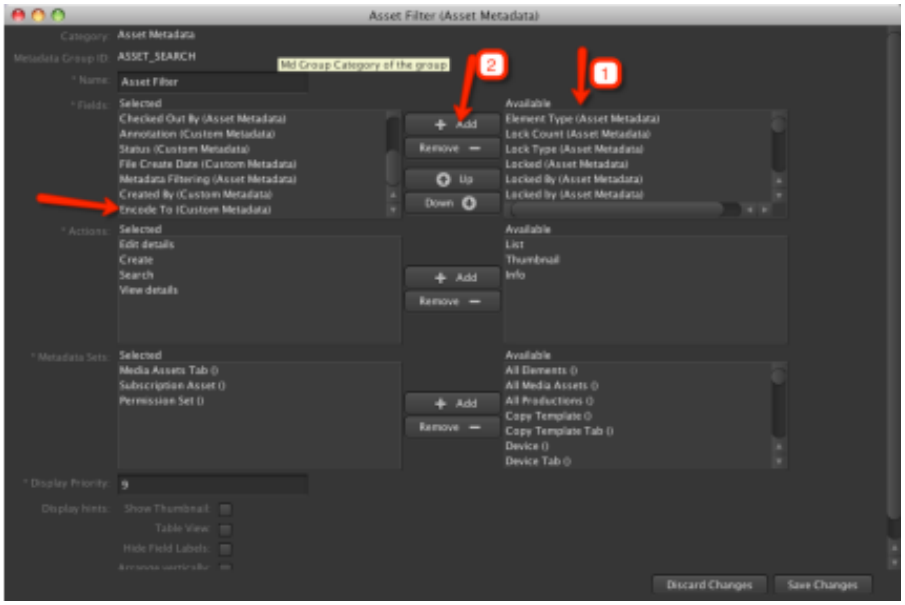
button to bring up a **Metadata Group** window. Set **Name** to **`Encode`**. Locate **Encode To** in the **Fields** and press **+ Add** to add it to the metadata fields. Similarly add **View details** and **Edit details** to **Actions**. Add **Media (Media Asset)** to **Metadata Sets**.



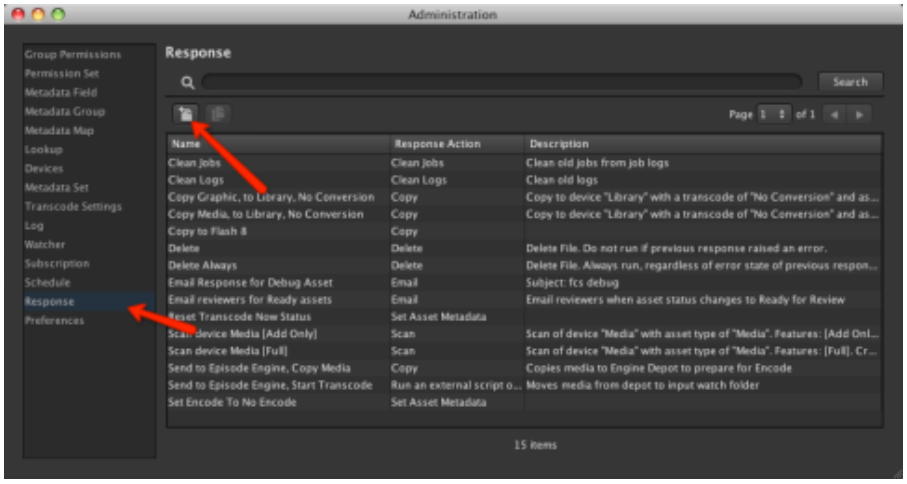
Return to the **Administration** window. Enter **`filter`** in the search box for **Metadata Group** to restrict the metadata groups shown in the window. Double-click on **Asset Filter** to open the **Asset Filter** window.



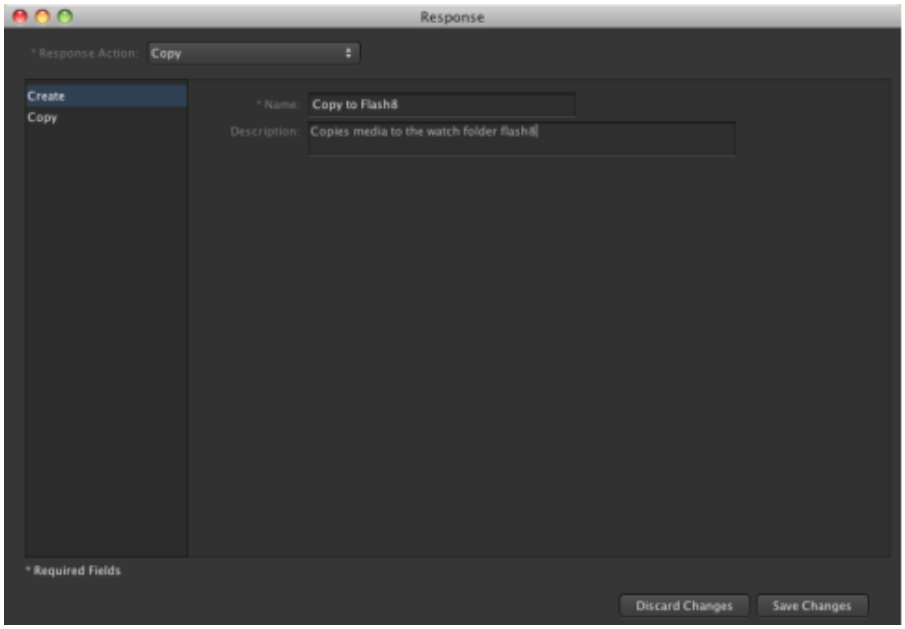
Locate **Encode To** in the **Available** list. Press **+ Add** to add it to the **Fields** list.



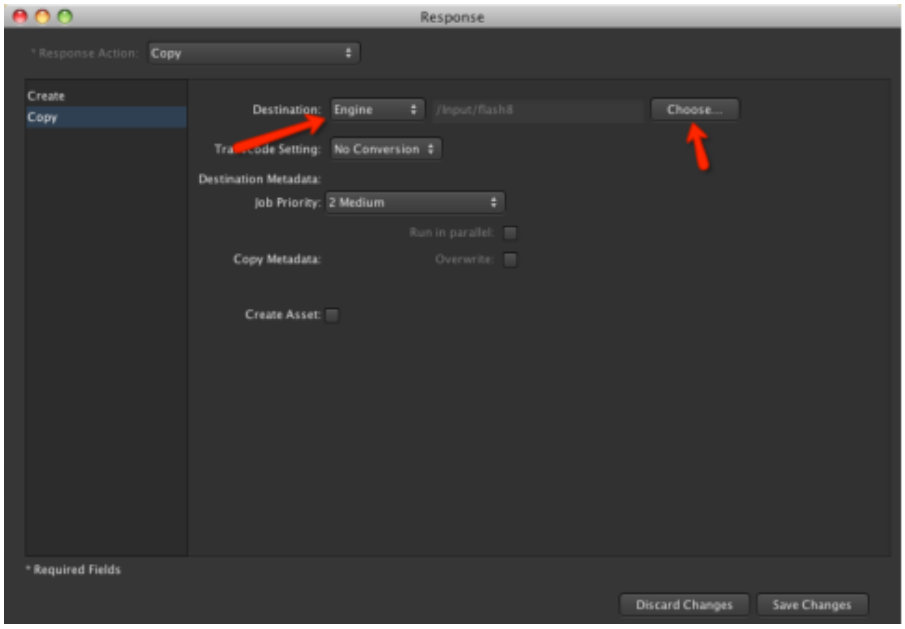
Set up job submission by clicking on **Response** in the leftmost column of the **Administration** window. Click on the **New Response** button to bring up a **Response** window.



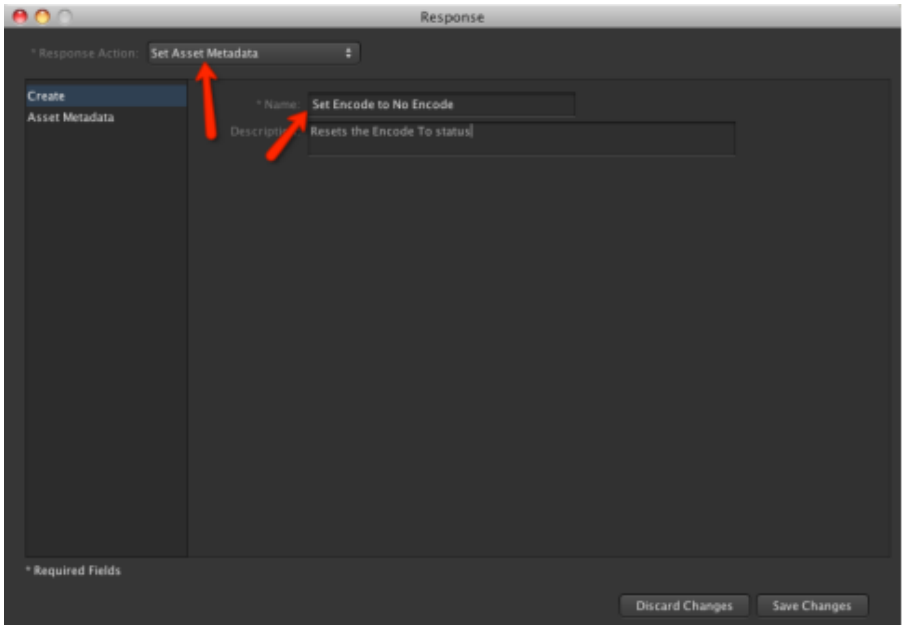
Select **Copy** in the **Response Action** menu. Select **Create** in the leftmost column of the window. Set the **Name** to something descriptive, in this example `Copy to Flash8`. Fill in the **Description** field with a longer explanation of the action.



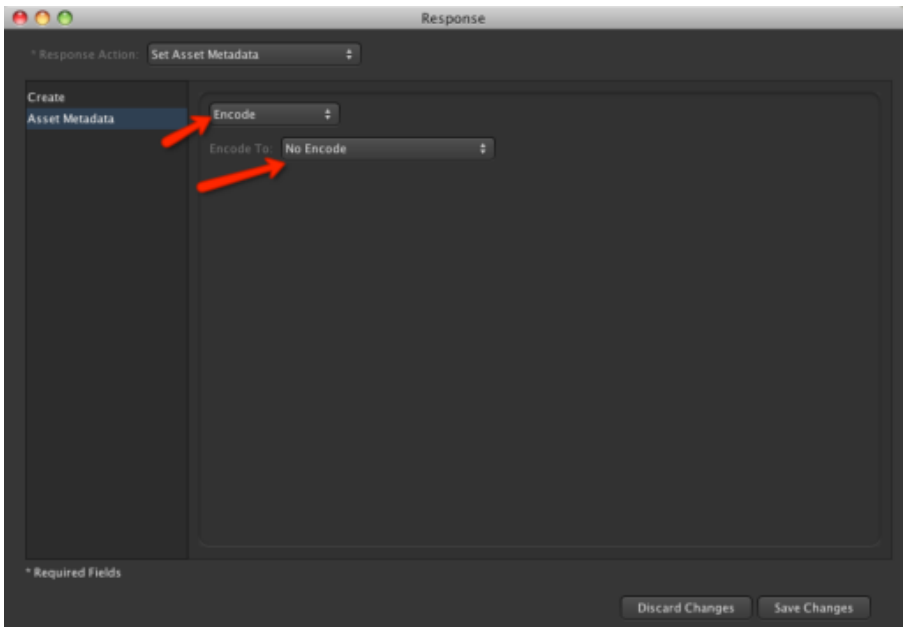
Select **Copy** in the leftmost column of the window. Select **Engine** (the device you created in the first step) in the **Destination** menu. Click **Choose...** to browse to the desired watch folder.



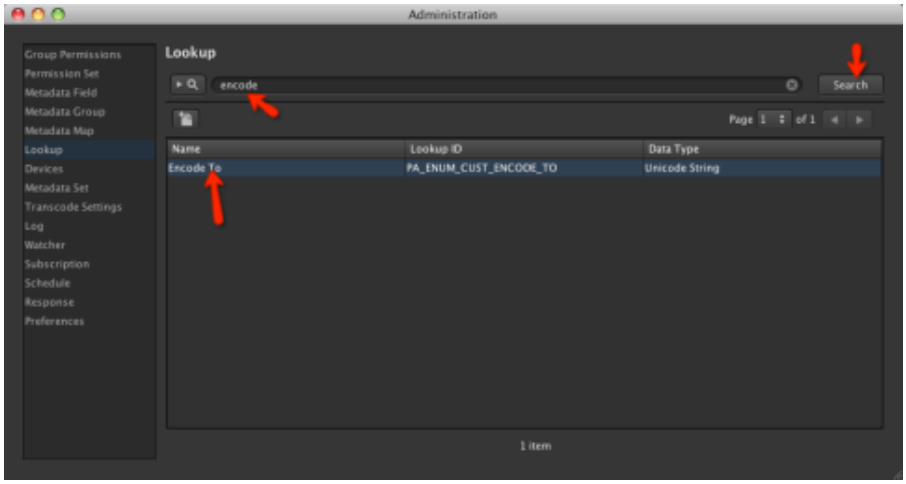
Select **Set Asset Metadata** in the **Response Action** menu. Select **Create** in the leftmost column of the window. Set **Name** to **`Set Encode to No Encode`**.



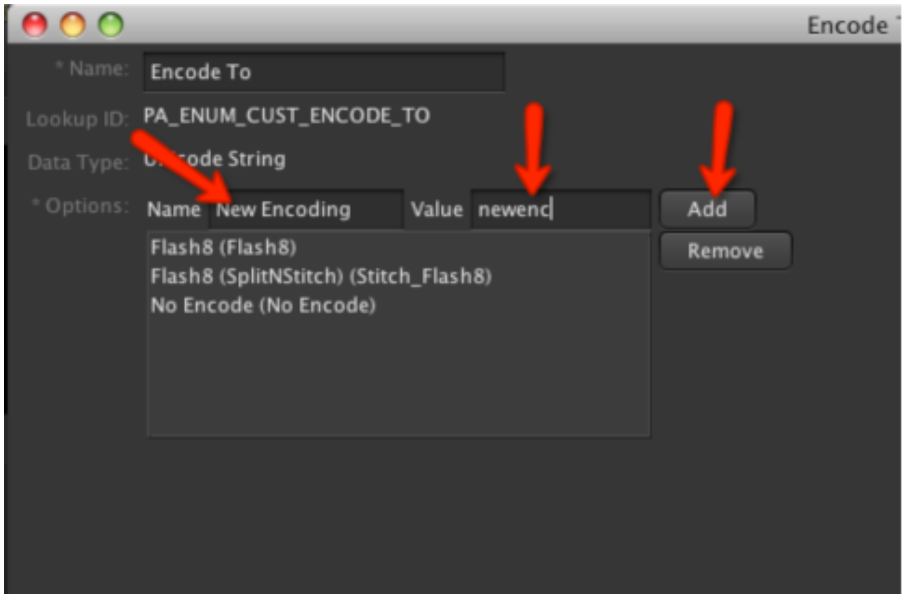
Select **Asset Metadata** in the leftmost column of the window. Select **Encode** in the top menu on the right. Set **Encode To** to **No Encode**.



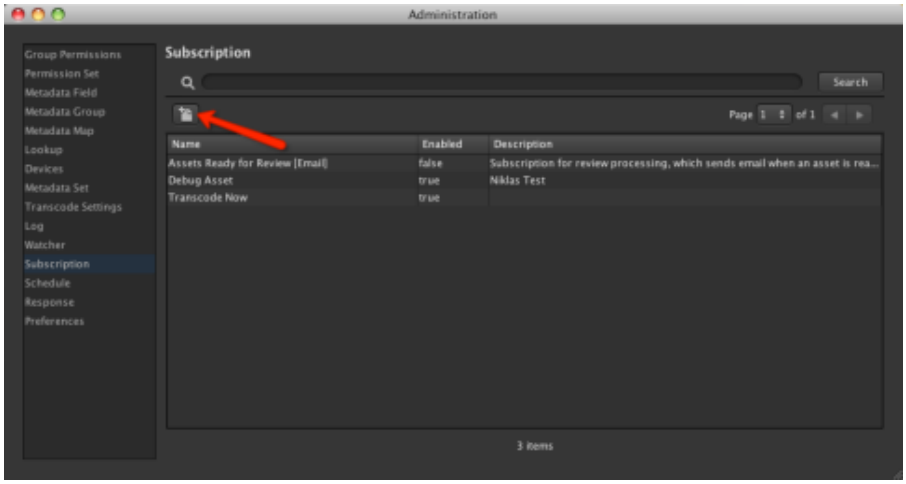
Select **Lookup** in the leftmost column of the **Administration** window. Locate your `Encode To` lookup and double-click on it to open its window.



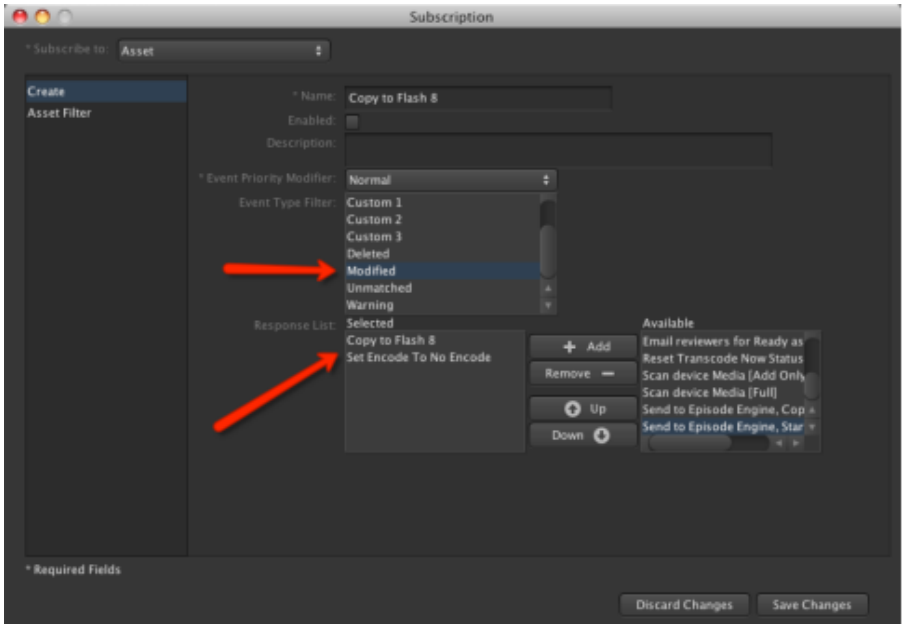
Add **Name** and **Value** pairs to the **Options**. They will correspond to watch folders.



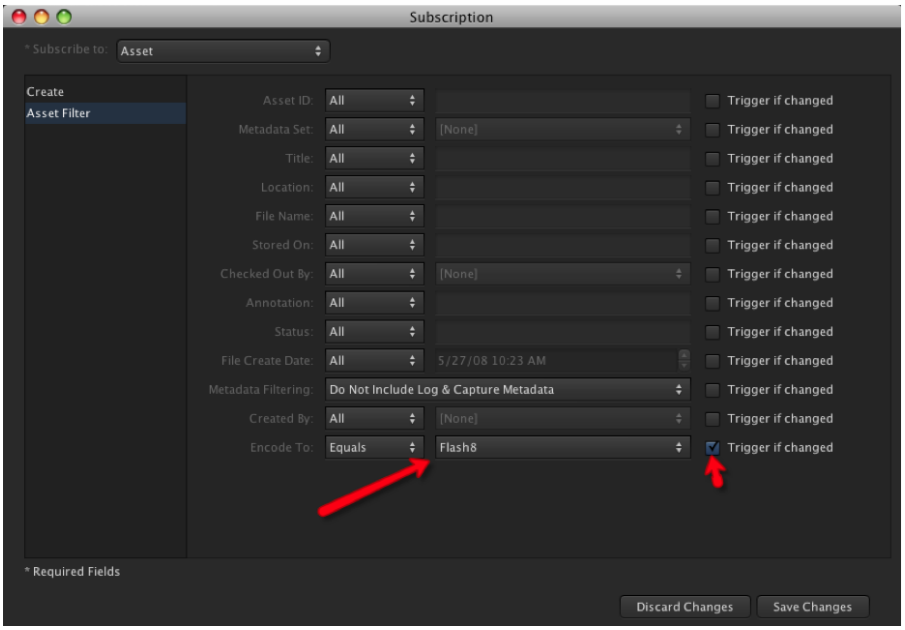
Select **Subscription** in the leftmost column of the **Administration** window. Click the **New Subscription** button to open the **Subscription** window.



Select **Create** in the leftmost column of the **Subscription** window. Set **Name** to a descriptive value. To trigger an encoding when the **Encode To** changes, select **Modified** in the **Event Type Filter**. Add the **Encode To** value you set in the **Asset Filter** above as well as the **Set Encode to No Encode** response.



Select **Asset Filter** in the leftmost column of the **Subscription** window. Select the value in the **Encode To** menu that shall trigger the subscription and check the **Trigger if changed** checkbox.



Repeat this for each **Name/Value** you set up in your lookup above. Be careful to add **Set Encode to No Encode** to each subscription response list, otherwise **FCS** will keep sending the asset to **Episode Engine** for transcoding.

4 Reference section

Here we will go through all the fine print and details of using **Episode Engine**. This is where to look if you have problems or want to figure out how to perform non-standard functions.

4.1 Watch folders

Watch folder root The default root for input watch folders is `/Users/Shared/Episode Engine/Input/`. The location is set during installation, but can be changed in the **System Preferences** for **Episode Engine**, or by editing the `services.conf` file, as explained in the Administrator's Guide.

Note that only the first level of folders under the root folder work as watch folders. Consequently, files placed in `/Users/Shared/Episode Engine/Input/Test` will be transcoded, but files placed in `/Users/Shared/Episode Engine/Input/` or `/Users/Shared/Episode Engine/Input/Test/Extras` will not be seen.

Episode Engine must have write access to watch folders.

Output folder creation Output folders with the same names as the input watch folders are created under the output folder root. The location of the output folder root is set during installation but can be changed in the **System Preferences** for **Episode Engine**, or by editing the `services.conf` file, as explained in the Administrator's Guide.

File discovery delay Input watch folders are *polled* for new files. When a new file is detected the **watcher** will check every few seconds to see if the size of the file has changed. When the size is stable, writing is assumed to have finished and the file can be safely sent for transcoding. The checking interval can be adjusted in the `services.conf` file, as explained in the Administrator's Guide.

If the process which writes the file allocates the total size from the start and then "fills in" the contents of the file, **Episode Engine** is not able to detect this and will prematurely delete the source file and start transcoding, which will cause errors.

Linking to source files Moving or copying source files to a watch folder consumes both time and storage space. It is considerably more efficient to instead *link* to the file. Hard links can only refer to files on the same volume, so a link on local storage cannot refer to a file on shared storage. Symbolic links instead contain the name of the referenced file, so they can be used cross-device.

A symbolic link is created like this:

```
prompt> ln -s /Volumes/RAID/sourcefile.mov \
          /Users/Shared/Episode\ Engine/Input/WatchFolder/
```

File deletion and archiving Source files and any optional (settings, watermark, metadata, audio, bumper and trailer) files specific to that source file are deleted when all settings files have been used to generate output; global files will never be automatically deleted (see section 4.6, *Optional files*). If a settings file specifies a watermark, audio, bumper/trailer and/or a metadata file to be used, transcoding will only start when the required files are present in the watch folder. If you have enabled archiving in the **System Preferences**, the files will instead be moved to an archive folder, by default in `/Users/Shared/Episode Engine/Archive/`.

File adding order Files should be added to a watch folder in the following order:

1. Settings files.
2. Metadata, watermark, bumper, trailer, audio sources (see section 4.6, *Optional files*).
3. Source files.

Priority Different folders can have different priorities for transcoding, so that you for example can have a special folder for urgent jobs, or give higher priorities to certain customers. Other jobs will not be placed on hold if a source file is placed into a high-priority folder, but it will be the first one selected when a node becomes free. The priority is an integer in the range 1–1000, higher numbers corresponding to higher priority. You assign a folder a priority value p by adding the suffix `^`p to the folder name. Folders without such a suffix will have a priority of 500.

Split-and-stitch You indicate that files are to be transcoded with split-and-stitch by giving the watch folder a name that starts with `stitch` (not case sensitive).

Episode Engine Pro

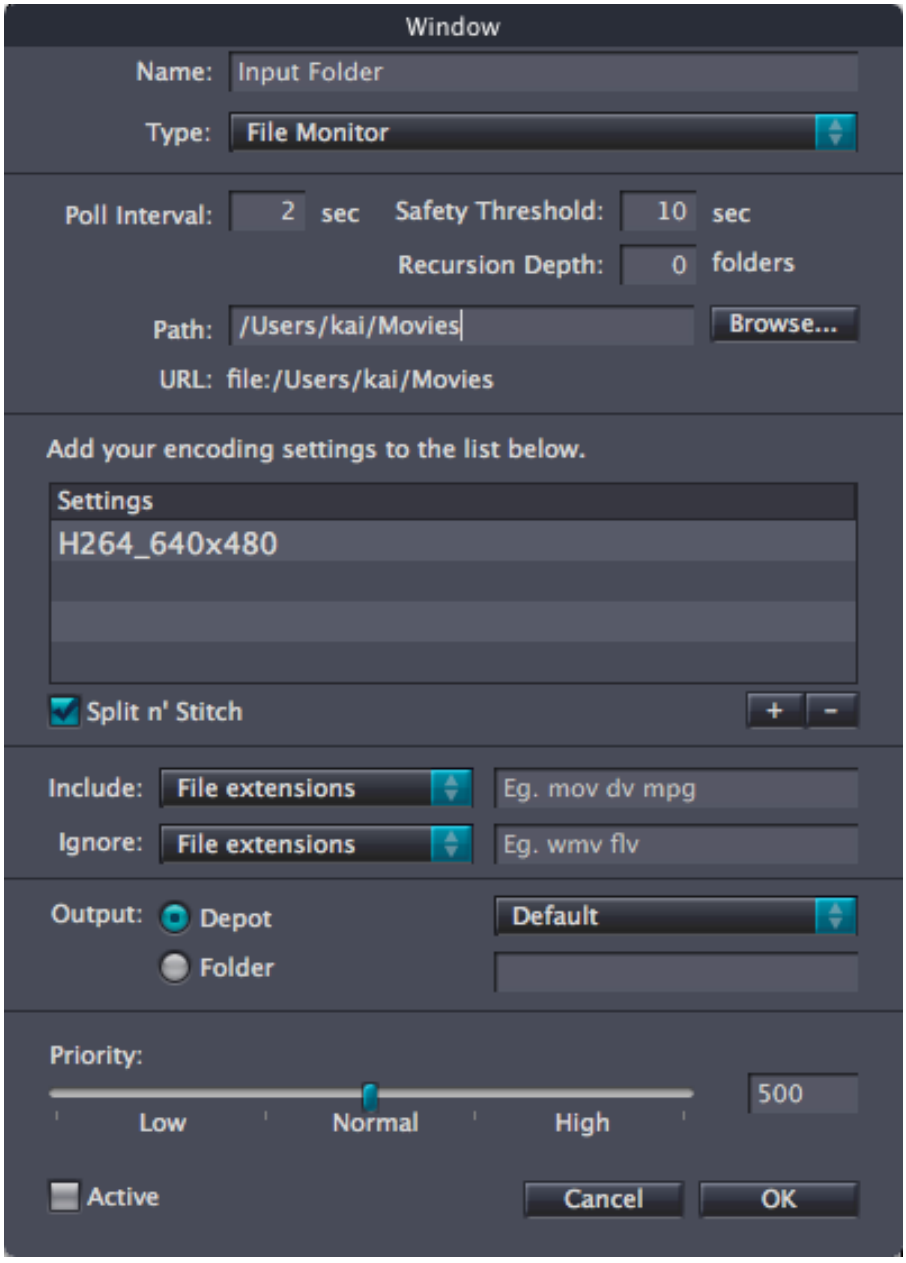
Due to the operation of the split-and-stitch functions, there will be some amount of bitrate overhead, you should therefore not use split-and-stitching for files with very low bitrates—these will in general not get much of a transcoding speedup anyway.

Split-and-stitch operation is not supported for output of hinted streaming files, multi-bit-rate (MBR) files, and RealMedia files.

4.2 Input monitors

We will describe the File Monitor in detail and for the other input monitors only describe the aspects in which they differ from the File Monitor.

4.2.1 File Monitor



All input monitors have a name. This name is shown in the **Input Monitors** list and also defines the name of the folder in which the output files are stored.

A File Monitor can monitor a folder anywhere on an Apple File System, but you cannot monitor a watch folder, as source files are deleted from watch folders.

The parameters you can set for the file monitor are the following:

Poll Interval The folder will be checked for new files every *t* seconds. Note that only *new* files in the folder are detected—any files already in the folder when the monitor is activated will be ignored.

Safety Threshold When a new file is found, an external process may still be writing to it. Thus the watcher will check again after the safety threshold. If the file has not changed during that time, it is assumed that it is safe to start reading from it.

Recursion Depth By default the file monitor will only watch for files in the folder pointed to by **Path**, but e g the RED camera places files in directory structures. Setting **Recursion Depth** to a value greater than `0` will search for input files that many directory levels down from the given input folder. Example: A file monitor on `/Users/jrn/video` with **Recursion Depth** set to `2` watches for files in `/Users/jrn/video`, `/Users/jrn/video/sub/`, and `/Users/jrn/video/sub/sub`.

Path The path to the monitored folder. The path is shown as a URL in the **URL** field below.

Settings All settings that will be used for transcoding files found in the monitored folder. Press the **+** button to bring up a browser of the Shared settings storage from which you can select settings to use. Select a settings row and press **–** to delete it from the file monitor.

Note that if you edit a setting in **Episode Encoder**, it is not automatically updated in the input monitors. You have to upload it from **Episode Encoder** to **Episode Engine** and then delete it from and re-add it to the settings in the monitor in order for the new version of the setting to be used in that monitor.

Note also that settings in the `Templates` folder are not guaranteed to remain in the same location or keep the same name between different versions of **Episode Engine**. You should therefore not use settings in the `Templates` folder, but instead upload a copy to a private folder which is retained between installations.

Episode Engine Pro

Split n' Stitch If checked, files in the monitored folder will be transcoded with the split-and-stitch function. Note that split-and-stitching is not possible for streamable output, multi-bitrate output, and RealMedia output—in these cases, regular transcoding will be performed instead.

Include Only files which match the inclusion criteria will be transcoded. Inclusion criteria can be given as:

File extensions A space-separated list of file extensions, i e the file types that will be encoded. E g, **`mov mp4`** means that only files with the extensions **`mov`** or **`mp4`** will be transcoded.

File name contains A space-separated list of strings that have to be present in the filenames. E g, **`NTSC PAL`** means that only files with **`NTSC`** or **`PAL`** in their names will be transcoded.

Regular expression A Perl compatible regular expression to match the filenames. E g, **`(.*NTSC.*)|(.*PAL.*)`**) means that only files with **`NTSC`** or **`PAL`** in their names will be transcoded.

Ignore Files which match the exclusion criteria will not be transcoded. The criteria are specified as for inclusion. **Include** and **Ignore** can be combined, inclusion being performed first.

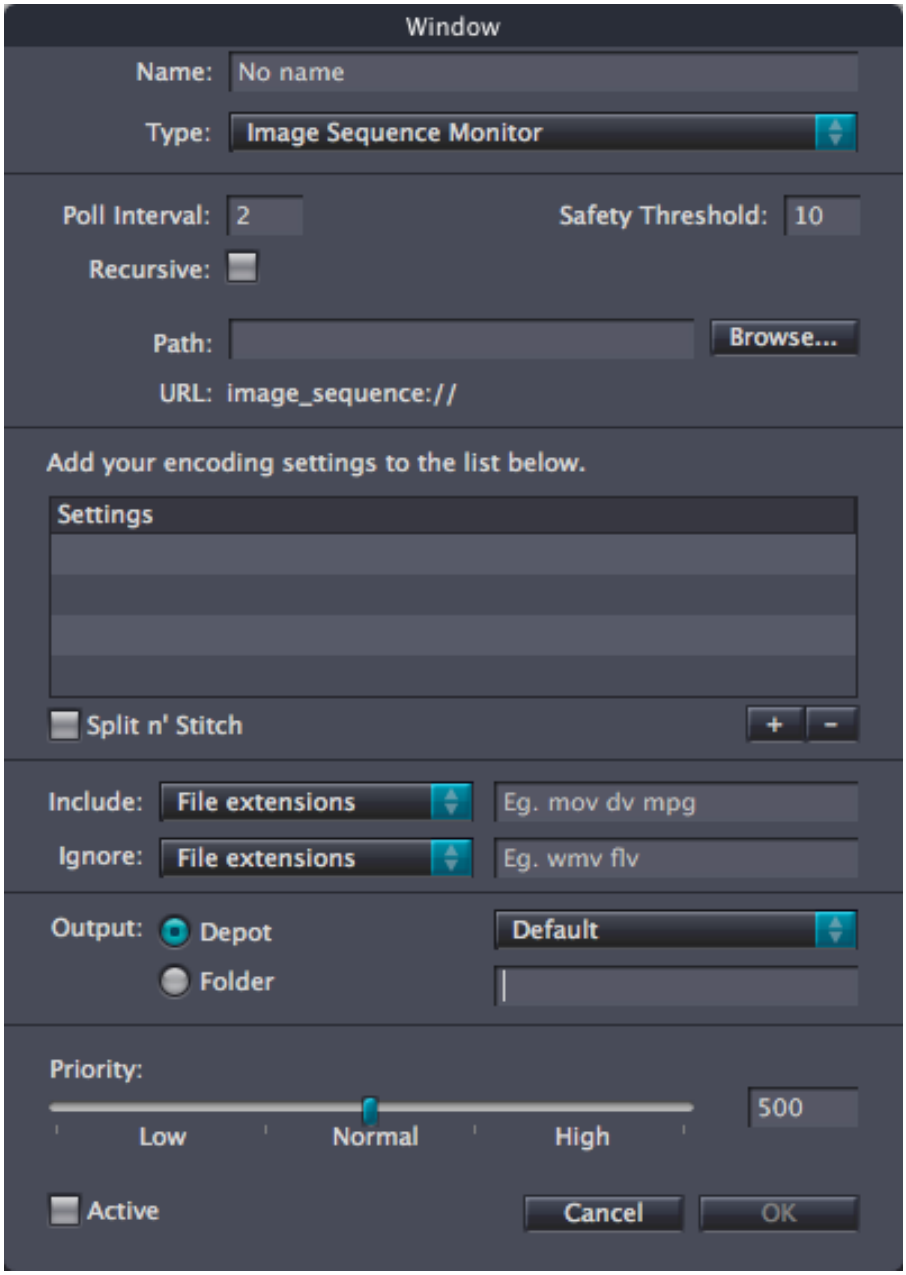
Output The default behaviour of **Episode Engine** is to store output files in the folder *`depot`*`/Monitors/`*`monitorname`*, where *`depot`* is the output depot first listed in `/usr/local/pwce/etc/depots.conf`. You can select an alternative depot from those defined in the configuration file.

Checking **Folder** lets you enter a path to a folder where the output files will be stored.

Priority The priority for jobs originating with this input monitor. Higher-priority jobs will be transcoded before lower-priority jobs.

Active You have to explicitly activate the input monitor for it to actually start monitoring. Inactive input monitors are still shown in the **Input Monitors** list. Select an input monitor and select **Input Monitors→Start Monitor** or **Start Monitor** from the context menu to activate it. Likewise select **Input Monitor→Stop Monitor** or **Stop Monitor** from the context menu to inactivate an input monitor.

4.2.2 Image Sequence Monitor



The Image Sequence Monitor works lets you import a series of still images from a local folder. The supported formats are BMP, DPX, GIF, JPEG, Targa and TIFF. Only DPX files contain frame rate information, all other formats will be assumed to be at 25 fps. In other words, if your source material is, for example, a TIFF sequence, and you set the output frame rate with the **Frame Rate** or **Advanced Frame Rate** filter, the frame rate conversion is done based on an input frame rate of 25 fps.

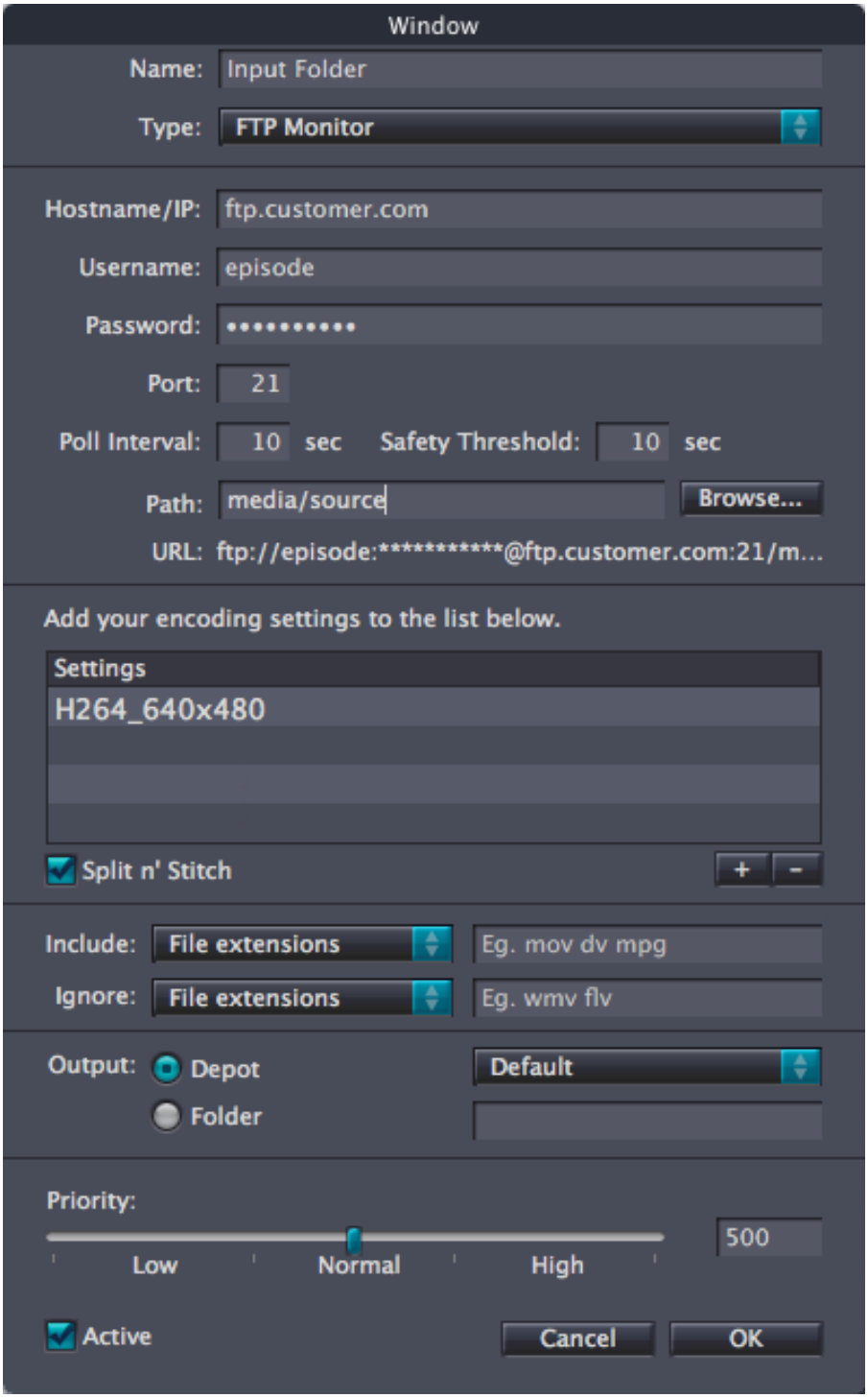
Like the File Monitor, the Image Sequence Monitor will only detect files placed in the monitored folder *after* the monitor has been activated. When no new files have arrived in a period equal to three times the **Safety Threshold**, the sequence

is considered to be finished and it is sent for transcoding.

The image files should be named as ⟨name⟩ ⟨sequencenumber⟩ . ⟨extension⟩. Files *must* be added to the monitored folder in sequence and with an increment of 1. If the sequence is broken, this will interpreted as the start of a new image sequence and send the previous sequence for transcoding.

The **Recursive** button specifies that any subfolders of the monitored folder will also be monitored, generating separate source clips for each subfolder.

4.2.3 FTP Monitor



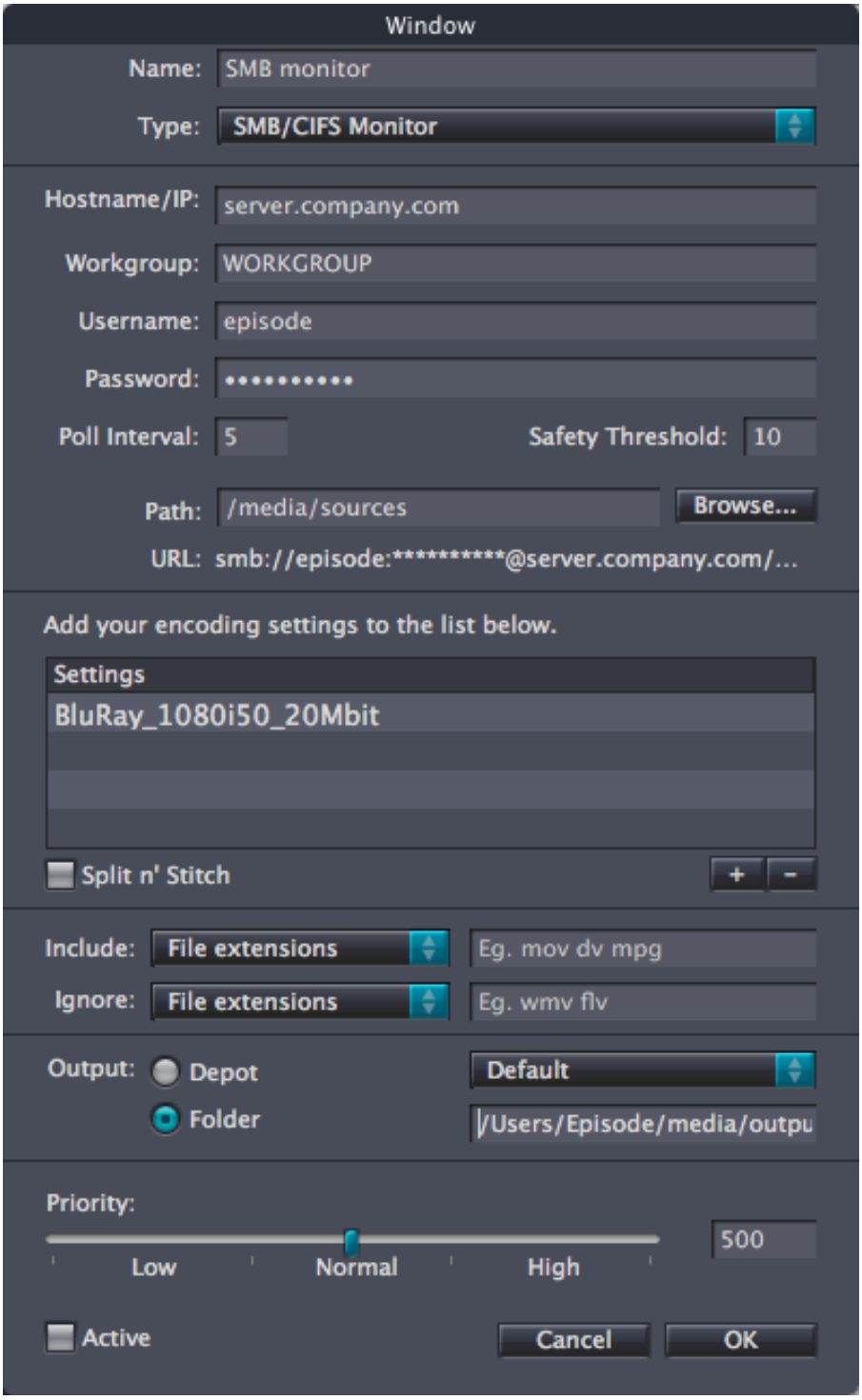
While one may open an FTP connection in the **Finder** and access it with a File Monitor, an FTP Monitor is more efficient and will reconnect if the connection is lost. The FTP Monitor will only detect files placed in the monitored folder *after* the monitor has been activated. In addition to the fields that File Monitor has, FTP Monitor has additional fields for connecting to and logging in on an FTP server.

Since an FTP connection typically has relatively low bandwidth it is usually not meaningful to use split-and-stitch for files retrieved from an FTP server, as the file cannot be split until it has been retrieved in its entirety.

NOTE

Different FTP servers behave slightly differently and it may be that the FTP Monitor does not communicate well with your particular server. Testing is necessary.

4.2.4 SMB/CIFS Monitor

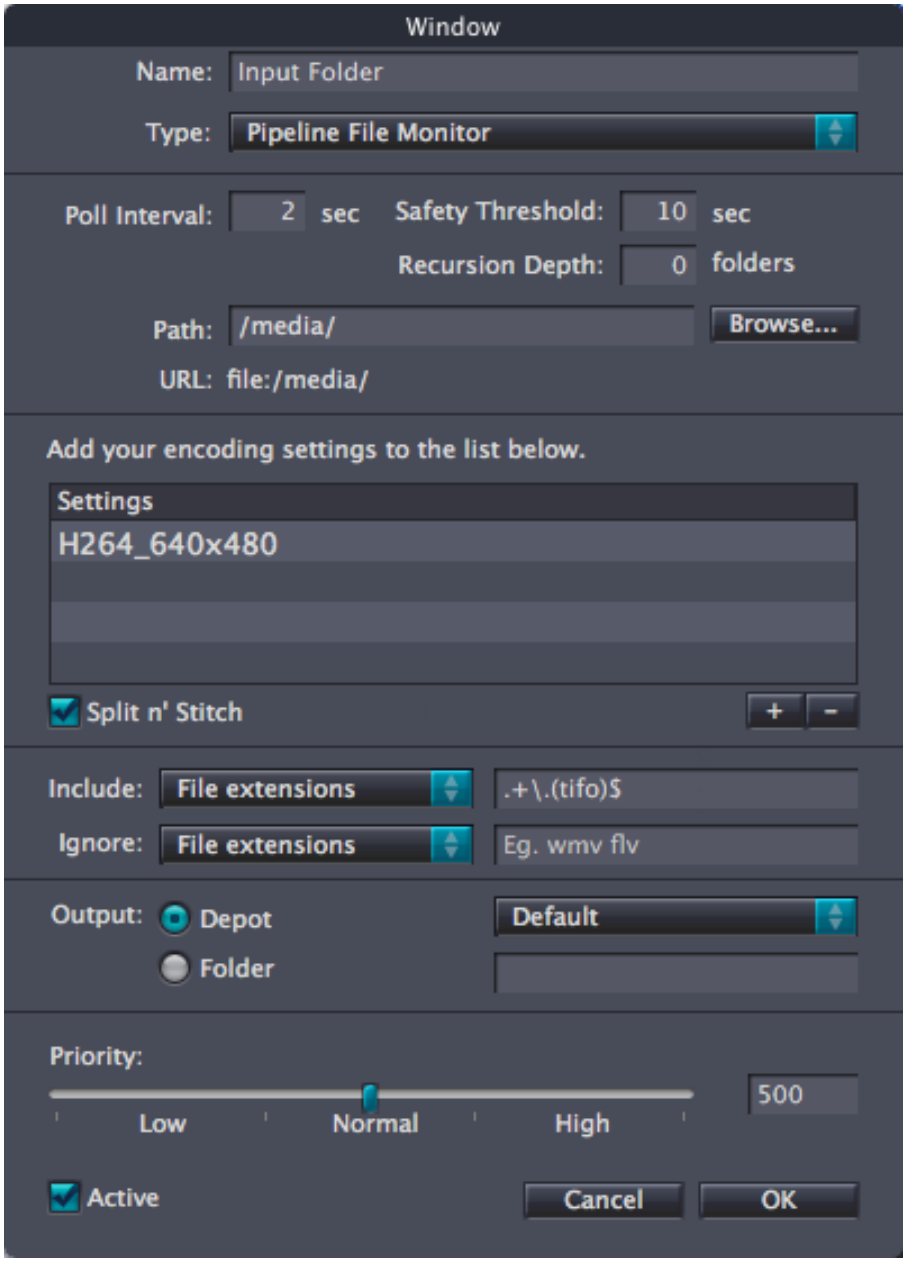


The SMB/CIFS Monitor lets you connect to a Microsoft Windows Network file server. The SMB/CIFS Monitor will only detect files placed in the monitored folder *after* the monitor has been activated. In addition to the fields that File Monitor has, SMB/CIFS Monitor has additional fields for connecting to and logging in on a Windows Network file server.

NOTE

If a file is *copied* into a monitored folder, Windows does not update the size field while copying, causing transcoding to start prematurely and therefore failing. Move files instead of copying, or set the **Safety Threshold** high enough that you can be sure that the file has finished copying before it is retrieved by the SMB/CIFS Monitor.

4.2.5 Pipeline File Monitor



The Pipeline File Monitor communicates with the Pipeline video capture device. It has the same fields as the standard File Monitor.

4.3 Storage depots

Storage depots are volumes accessible to both **Episode Engine** and any media-producing clients. The storage depots are defined in the XML file `/usr/local/pwce/etc/depots.conf`.

4.4 Shared settings

In the default configuration input monitors will find their settings files in `/Users/Shared/Episode Engine/Settings`. These settings are common for all users of the same **Episode Engine** server. You can upload settings to shared settings directly from **Episode Encoder**.

4.5 Hardware acceleration

The Matrox H.264 hardware transcoder can only be used by a single process at a time. If you are using such hardware you must therefore set the **System Preferences** for **Episode Engine** to allow only a single parallel job on each node.

4.6 Optional files

As explained in section 1.4, *Advanced features*, the basic media file can be extended with additional information, such as metadata, watermarks and bumpers/trailers. These "optional files" are placed in the same folders as the source files to be transcoded, so to distinguish them from the source files, they must have special file extensions indicating their role:

`.setting`, `.mbrsetting`	Settings file
`.watermark`	Watermark file
`.bumper`, `.intro`	Bumper clip
`.trailer`, `.outro`	Trailer clip
`.inmeta`	Input metadata
`.audio`	Audio source

In **Episode Encoder** you indicate in the settings if a file is to be extended with optional files and you also explicitly name the files to be used, but when exporting the settings file for use in **Episode Engine**, the names are dropped and only the information that optional files will be needed is carried along, so that you are not restricted to just specifically named file(s).

If the use of any optional file has been requested, transcoding will not begin before all requisite files are present in the folder. Conversely, if a setting does *not* require the presence of a given file, the presence of such a file in the folder does not matter. **Engine Admin** will indicate what optional files a settings file requires.

Optional files are either global or specific to a source file. A global file is the default file, whereas a source-file specific file will be matched with one particular

source file and used for that instead of any global file. Optional files are independent of each other, so a source file can be transcoded with, e g, a global watermark file and a specific metadata file.

A global file must have a filename that starts with `!` followed by anything and end with the appropriate extension from the list above, e g `!Anyfilename.trailer`. A source-file specific file must have the same name as the source file, including its extension, plus the appropriate extension from the list above, e g `MyFile.mpg.trailer` will match the source file `MyFile.mpg`. Note that this naming convention obscures the file type of watermarks and bumper/trailer clips, but **Episode Engine** is not dependent on the file extension to determine the file format.

Global files are never automatically removed from a watch folder, per-source files are removed along with their matching source file. Settings files are always considered global, regardless of their names, so they are never removed from watch folders. Other input monitors do not remove any files.

4.6.1 Example

Consider a monitored folder containing a settings file `Example.setting`, a GIF file `!general.watermark`, and a JPEG file `special.mov.watermark`.

When the three source files `general.mov`, `special.avi` and `special.mov` are added to the folder, they will be transcoded with the settings in `Example.setting`. These settings specify that a watermark is to be used.

Since its name matches the name of the watermark, `special.mov` will be watermarked with `special.mov.watermark`, while `general.mov` and `special.avi` will be watermarked with the global watermark `!general.watermark`.

4.6.2 Watermarks

Positioning The size and position of the watermark in the frame is set in **Episode Encoder**.

Supported formats

Format	Comments
Bitmap	24 bit RGB
GIF	
JPEG	EXIF metadata also supported
QuickTime	
Targa	24 bit RGB, 32 bit RGB
TIFF	24 bit RGB, 32 bit RGB

4.6.3 Bumpers and trailers

Clip contents Bumpers and trailers are transcoded in the same format, size and frame rate as the output file but will not be otherwise transformed, in particular they will not be deinterlaced if the output is to be progressive and vice versa. Bumpers and trailers must have video and audio tracks corresponding to the video and audio tracks of the output file, i e output with both video and audio requires bumpers and trailers with both video and audio, but video-only output does not require audio tracks in the bumper and trailer.

4.6.4 Metadata

Metadata sources Metadata for an output file can come from three different sources:

1. The **Metadata** tab in **Episode Encoder**.
2. The **Engine** tab in **Episode Encoder**.
3. An `.inmeta` file with metadata in XML format.

Metadata from settings The **Metadata** tab in **Episode Encoder** contains exactly the tags that are supported by the given output format, the metadata are therefore written directly to the output file.

Extra metadata fields Metadata from the **Engine** tab and/or an `.inmeta` file are written to the output file for those tags that are supported by the output data format. They override metadata with the same tags given in the **Metadata** tab. Tags which are not supported by the output format are written to a `.meta` file that is placed in the output folder together with the output media file.

The use of an `.inmeta` file requires checking the **Use .inmeta File** button in the **Engine** tab in **Episode Encoder**. If this option is used there *has* to be an `.inmeta` file in the monitored folder before transcoding can begin.

Naming scheme The `.inmeta` file must have the same name as the source file plus the extension `.inmeta`, e g, a source file `sample.mov` requires an inmeta file called `sample.mov.inmeta`. The `.meta` file will have the same name as the output media file plus the extension `.meta`, e g, an output file called `mp4_qcif_128_meta_sample.mp4` will have a metadata output file called `mp4_qcif_128_meta_sample.mp4.meta`.

Example `.inmeta` file The precise syntax, the Document Type Definition, for `.inmeta` files is in `/usr/local/pwce/etc/inmeta.dtd`.

This is the `sample.mov.inmeta` file from the `Verify` package in your distribution. Please use that file instead of copy-pasting the formatted text below.

<!**DOCTYPE** meta−data **SYSTEM** "inmeta.dtd">

```
<meta−data>

  <!−− Metadata can be written to the output file, the .meta file or both. The "type"
       attribute of the <meta−group> element specifies the destination. You can specify
       up to eight metadata fields to be written to the output file: "title", "author",
       "artist", "producer", "description", "copyright", "creation date", and "software".
       The names of these will be converted to whatever the corresponding field name
       is in the particular output format. If a field is not supported by the output format
       it will be ignored. Metadata written to the .meta file are not interpreted, but just
       written as they are given. −−>

  <!−− Values written to both output file and .meta file: −−>
  <meta−group type="movie meta">
    <meta name="title" value="Example Movie"/>
    <meta name="copyright" value="Telestream, Inc"/>
    <meta name="software" value="Episode Engine"/>
  </meta−group>

  <!−− Values written only to the output file: −−>
  <meta−group type="movie">
    <meta name="author" value="Ingmar Bergman"/>
    <meta name="artist" value="Sven Nyqvist"/>
    <meta name="producer" value="Harry Schein"/>
    <meta name="description" value="A story of love and life lost"/>
    <meta name="creation date" value="1984−04−01"/>
  </meta−group>

  <!−− Values written only to the .meta file: −−>
  <meta−group type="meta">
    <meta name="database−id" value="123"/>
    <meta name="Team" value="The Pops"/>
    <meta name="result" value="2−0"/>
  </meta−group>

  <!−− The data in the <meta−movie−info> element are written as a log to the .meta
       file after transcoding. The keys given in the "tokens" attributes determine what
       data will be recorded. The different elements specify what will be recorded for
       the entire movie, for each track of the movie, and specifically for video, audio
       and streaming hint tracks. Note also that the "tokens" list can be split over
       multiple elements (as in the two <meta−movie> elements) if that makes the file
       more legible. −−>
  <meta−movie−info>
    <meta−movie tokens="duration bitrate size"/>
    <meta−movie tokens="format duration bitrate size tracks"/>
    <meta−track tokens="type format start duration bitrate size"/>
    <meta−video−track tokens="width height framerate"/>
    <meta−audio−track tokens="channels bitspersample samplerate"/>
    <meta−hint−track tokens="payload fmtp"/>
  </meta−movie−info>

</meta−data>
```

Example `.meta` file This is the `.meta` file generated from the `.inmeta` file above, with comments added:

```
<?xml version="1.0" encoding="UTF−8"?>
```

```
<!DOCTYPE meta−data SYSTEM "meta.dtd">
<meta−data version="1.0">

  <!−− The metadata specified in the .inmeta file, output in alphabetical order. −−>

  <meta name="meta−source" value="inmeta">
    <meta name="Team" value="The Pops"/>
    <meta name="copyright" value="Telestream, Inc"/>
    <meta name="database−id" value="123"/>
    <meta name="result" value="2−0"/>
    <meta name="software" value="Episode Engine"/>
    <meta name="title" value="Example Movie"/>
  </meta>

  <!−− The path to the output file. −−>
  <meta name="movie"
        value="/Users/Shared/Episode%20Engine/Output/Test/Input−stream.3gp">

    <!−− These elements are based on the tokens lists in the <meta−movie−info>
         element of the .inmeta file. The values form a log of the transcoding. −−>
    <meta name="format" value=".3gp"/>
    <meta name="duration" value="10.54"/>
    <meta name="bitrate" value="117.26"/>
    <meta name="size" value="158223"/>

    <!−− The <meta−track tokens> in the .inmeta file generate a <meta
         name="track"> for each track with the requested values. In addition,
         <meta−video−track tokens> generate values that are specific for the video
         track. −−>
    <meta name="track" value="0">
      <meta name="type" value="vide"/>
      <meta name="format" value="mp4v"/>
      <meta name="start" value="0.00"/>
      <meta name="duration" value="10.60"/>
      <meta name="bitrate" value="36.00"/>
      <meta name="size" value="49483"/>
      <meta name="width" value="176"/>
      <meta name="height" value="144"/>
      <meta name="framerate" value="7.36"/>
    </meta>

    <!−− The <meta−audio−track tokens> in the .inmeta file generate the audio track
         specific values. −−>
    <meta name="track" value="1">
      <meta name="type" value="audi"/>
      <meta name="format" value="mp4a"/>
      <meta name="start" value="0.00"/>
      <meta name="duration" value="10.49"/>
      <meta name="bitrate" value="15.00"/>
      <meta name="size" value="21307"/>
      <meta name="channels" value="1"/>
      <meta name="bitspersample" value="16"/>
      <meta name="samplerate" value="22050.00"/>
    </meta>

    <!−− This is the streaming hint track for the video. The "type" to "size" elements
         come from the <meta−track> element, "payload" and "fmt" from the
```

```
        <meta−hint−track> in the .inmeta file. −−>
    <meta name="track" value="2">
      <meta name="type" value="hint"/>
      <meta name="format" value="mp4v"/>
      <meta name="start" value="0.00"/>
      <meta name="duration" value="10.60"/>
      <meta name="bitrate" value="36.00"/>
      <meta name="size" value="52532"/>
      <meta name="payload" value="96"/>
      <meta name="attribute" value="a=fmtp:96
          profile−level−id=8;config=000001B008000001B50EA040C0CF000001000000012000845D4C2
    </meta>

    <!−− The same <meta−hint−track> is used for the audio hint track. −−>
    <meta name="track" value="3">
      <meta name="type" value="hint"/>
      <meta name="format" value="mp4a"/>
      <meta name="start" value="0.00"/>
      <meta name="duration" value="10.59"/>
      <meta name="bitrate" value="15.00"/>
      <meta name="size" value="28831"/>
      <meta name="payload" value="97"/>
      <meta name="attribute" value="a=fmtp:97
          profile−level−id=15;object=2;cpresent=0;config=400027103FC0"/>
    </meta>

  <!−− This </meta> ends the earlier <meta name="movie">. −−>
  </meta>

<!−− This ends the metadata document and file. −−>
</meta−data>
```

4.6.5 Separate audio source files

Combining a single video file and single audio file Sometimes one wishes to combine video and audio from different sources, but the **Episode Encoder** user interface does not let you create such a setting. However, you can manually edit the settings file to achieve this effect.

A settings file is an XML file, so it can be edited in **Textedit**, **Emacs** or whatever word processor you want, as long as you remember to save in pure text format. Note that you cannot edit a manually edited settings file in **Episode Encoder**, as the file would become corrupted.

In particular and relevant for our current task, a settings file contains clauses specifying the source file(s). Consider a settings file that contains the following clause, specifying that a single file is the source for both video and audio:

```
<importer media="movie" in="file://!SRCFILE!">
  <audio−out id="audio_4f70140_importer"/>
  <video−out id="video_16e7a10_importer"/>
</importer>
```

The !SRCFILE! is a tag that represents the source file that is to be transcoded. A matching audio source file is represented in the settings file by the template

!AUDIOFILE!, so to use separate video and audio sources we would rewrite the clause as:

```
<importer media="movie" in="file://!SRCFILE!">
  <video−out id="video_16e7a10_importer"/>
</importer>
<importer media="movie" in="file://!AUDIOFILE!">
  <audio−out id="audio_4f70140_importer"/>
</importer>
```

On encoding you would add the audio file along with the video source file to the watch folder or monitored folder. The audio file is to be named with the full file name of the video file + the extension `.audio`, ie, a video file `sample.mov` requires the audio file to be named `sample.mov.audio`.

In this example we have retained the id values of the video and data streams as they are used by the filters "downstream" in the transcoding process. The id values will be different for different settings files, but they can be renamed to anything as long as the names are consistently used in the file.

Creating multiple audio tracks In general, **Episode Engine** can only output a single audio track, but MPEG-2 Transport Streams can be output with up to eight audio tracks.

In this example we will assume that we have two audio sources, one in French, one in English. As in the previous example, we edit an existing settings file and take out the audio import into separate clauses:

```
<importer media="movie" in="!AUDIOFILE_01!">
  <audio−out id="audio_01"/>
</importer>
<importer media="movie" in="!AUDIOFILE_02!">
  <audio−out id="audio_02"/>
</importer>
```

You have to create copies of any audio filters in the setting, so that both audio tracks are correctly transcoded. Keep careful track of the id values connecting the filters, so that the data are progressed through all filters.

MPEG TS output is indicated in the settings file with a clause like this:

```
<exporter media="movie" out="!DSTPATH!/!NAME!−!SETTING!.mpg"
      type="ts__">
  <in id="video_14fb69b0_mp2v_encoder"/>
  <in id="audio_14fbd080_mp2a_encoder"/> <!−− We will change this −−>
  <option name="audio_pid" value="69"/> <!−− We will change this −−>
  <option name="video_pid" value="68"/>
  <option name="pcr_pid" value="67"/>
  <option name="pmt_pid" value="66"/>
  <option name="transport_rate_mode" value="0"/>
  <option name="transport_rate" value="10000"/>
  <option name="transport_rate_unit" value="1"/>
  <option name="pes_mode" value="0"/>
  <option name="pes_max_size" value="16348"/>
  <option name="program_number" value="1"/>
  <option name="language_code" value="0"/> <!−− We will change this −−>
</exporter>
```

We modify the above to specify two audio tracks in the output:

```
<exporter media="movie" out="!DSTPATH!/!NAME!−!SETTING!.mpg"
      type="ts__">
  <in id="video_14fb69b0_mp2v_encoder"/>

  <!−− The id from the last filter for the first audio track −−>
  <in id="audio_01_mp2a_encoder"/>
  <!−− The id from the last filter for the second audio track −−>
  <in id="audio_02_mp2a_encoder"/>

  <option name="video_pid" value="68"/>
  <option name="pcr_pid" value="67"/>
  <option name="pmt_pid" value="66"/>
  <option name="transport_rate_mode" value="0"/>
  <option name="transport_rate" value="10000"/>
  <option name="transport_rate_unit" value="1"/>
  <option name="pes_mode" value="0"/>
  <option name="pes_max_size" value="16348"/>
  <option name="program_number" value="1"/>

  <!−− Specify the number of audio tracks −−>
  <option name="num_audio_tracks" value="2"/>
  <option name="audio_01_pid" value="1001"/>
  <option name="audio_01_language_code" value="13"/> <!−− French −−>
  <option name="audio_02_pid" value="1002"/>
  <option name="audio_02_language_code" value="11"/> <!−− English −−>
</exporter>
```

In our previous example we had a single audio source file, which we could drop into the watch folder together with the video file, but there is no general mechanism for supplying two or more audio source files. We therefore have to supply the names of the audio files in the settings file. We could do this by explicitly entering the file names in the importer clauses and then rename the actual files for each transcoding, thus:

```
<importer media="movie"
 in="/Users/Shared/Episode Engine/Input/TS/Show_french.audio">
  <audio−out id="audio_01"/>
</importer>
<importer media="movie"
 in="/Users/Shared/Episode Engine/Input/TS/Show_english.audio">
  <audio−out id="audio_02"/>
</importer>
```

However, a more flexible approach may be to use the **engine** application described in appendix B, ***engine***. This requires you to start each transcoding with a command, this may or may not fit with your workflow. In this case we modify the placeholders in the settings file, here called `ts_dual_audio.setting`, for each transcoding.

```
engine add media --filename /Users/jrn/Movies/Show.mov \
                 --setting_id ts_dual_audio.setting \
                 --replace AUDIOFILE_01 /Users/jrn/Movies/Show_french.aiff \
                           AUDIOFILE_02 /Users/jrn/Movies/Show_english.aiff
```

The replacement function is general, any tag within ! can be replaced. You can consider **`--filename`** a shorthand for **`--replace SRCFILE`**. You could of

course use the same mechanism for the previous example with a single audio source file, should it be more convenient for you.

4.7 Event scripts

Scripts placed in `/usr/local/pwce/evt/` (default location) will be executed by **Episode Engine** when given events occur.

The **Episode Engine** log file will show the return value of your script if not equal to 0. In order to get any printouts and error messages from the script, create a file `/usr/local/pwce/svc-ctrl/pwevent/log` that contains the path to a file where the script output will be written, e g `/tmp/eventlog`, and then restart **Episode Engine**.

Form-based scripting You can write scripts in any language of your choice, but if you feel uncomfortable with programming, **Episode Encoder** will let you fill in a form to handle the most common issues of moving output files to their final destinations. See the chapter *Engine tab* in the ***Episode Encoder*** *User Guide* for the details.

Script names Scripts must be named on the form ⟨digit⟩ ⟨digit⟩_⟨type⟩ [⟨optionaltext⟩]. The prefix number defines the order in which the scripts will be executed, with lower numbers executed first. ⟨type⟩ is `job`, `node`, or `msg`, corresponding to the type of event that triggered the execution of the script as detailed below. ⟨optionaltext⟩ can be used to indicate what the script does.

Job scripts A job script is executed when a job finishes. Note that the script is triggered even if the job finished due to an error. The script is executed in an environment where the following variables are set:

`JOB_CREATION_TIME` The time when the job was created.

`JOB_ID` A unique integer that identifies this job.

`JOB_IN_URLS` The URLs of all input files for this job, separated by commas.

`JOB_NAME` Identical to job_media.

`JOB_NODE_ID` An integer identifying the node that ran the job.

`JOB_OUT_FOLDER` Identical to job_folder.

`JOB_PRIO` The priority of the job.

`JOB_REAL_TIME` The wall-clock time in seconds used for the job.

`JOB_REASON` One of no−start, bad−com, bad−job, fail, crash, lost, cancel or finish, depending on how the job finished. If the job finished successfully (finish), two additional variables will be set:

`JOB_OUT_PATH` The path to the output file.

JOB_OUT_URL The URL to the output file.

JOB_RUN_COUNT The number of times the job was processed. This should normally be "1", but may be up to one higher than the value of max−job−retry in `engine.conf`.

JOB_SETTING_NAME Identical to job_setting.

JOB_SOURCE_FILE Identical to job_media.

JOB_START_TIME The time when the job was started.

JOB_STOP_TIME The time when the job was finished.

JOB_OUT_URLS The URLs of all output media files of this job, separated by commas.

client_host The name of the node that submitted the job.

client_name The name of the client process that submitted the job.

job_folder The name (not full path) of the folder containing the output media file. This variable is set only if client_name is Watcher, i e if the job originates in a watch folder. For jobs created by input monitors, this variable is empty.

job_kind A string indicating the type of job, can be regular, split, or stitch.

job_media The name of the input media file. If the job is submitted by the **watcher** client, i e if the job originates in a watch folder, only the filename is given, otherwise the full URL of the file is given.

job_name The name of the job as set by the client.

job_setting The name of the settings file.

monitor_depot The name of the storage depot where the output files are placed, if this was specified in the input monitor.

monitor_name The name of the input monitor that submitted the job.

monitor_outfolder The name of the folder where the output files are placed, if this was selected in the input monitor.

owner The username of the user that submitted the job.

Node scripts A "node" script is executed when contact is lost with a node (possibly due to an orderly shutdown at that node). The script is executed in an environment where the following variables are set:

NODE_ID The integer identifying the node.

NODE_NAME A string identifying the node, normally its host name.

NODE_STOPTIME The time when contact was lost.

Msg scripts A "msg" script is triggered when a log message of severity up to 3 (= ERROR) has been generated. The script is executed in an environment where the following variables are set:

`MSG_ENTITY` One of engine, client or node, indicating what type of entity caused the log message. For client and node the following variables are also set:

`MSG_ENTITY_ID` The integer identifying the client or node.

`MSG_ENTITY_NAME` A string identifying the entity, normally the host name of the node.

`MSG_SEVERITY` One of ERROR, CRITICAL, ALERT, or EMERGENCY.

`MSG_STRING` The actual log message.

`MSG_TIME` The time when the message was generated.

NOTE

Depending on exactly what processes have processed a job, some of the variable values may be empty strings.

TIP

Scripts are located centrally, but sometimes it may simplify matters to have scripts specific to each watch folder/input monitor. You can solve this by placing script files in the respective output folders and then have a dispatcher script in `/usr/local/pwce/evt/` that locates the specific script and executes it.

Alternatively you can supply parameters in metadata, which can then be read by the script.

Example A set of example scripts are located in `/usr/local/pwce/evt/examples/`.

Below is the example script `00_job`. It attempts to move the transcoding output to an **ftp** server. If that fails it constructs and sends an email message describing the problem to an administrator.

The script is written in **bash**, but any language can be used.

```
#!/bin/bash

USER="anonymous"
PASSWORD="anonymous"
HOST="example.com"

ADDRESS="ftp://${USER}:${PASSWORD}@${HOST}/"

RETURN_CODE=0
```

```
# Check if the job completed successfully
if [ "$JOB_REASON" == "finish" ]; then
    # Create temp file for FTP server directory listing (to see if upload was successful)
    TEMPFILE=`/usr/bin/mktemp /tmp/ftp_XXXXXX`
    # Extract file name to use for naming file on FTP server
    FILENAME=`/usr/bin/basename "$JOB_OUT_PATH"`
    # Connect, set binary mode, upload file, list directory, close connection.
    /usr/bin/ftp -iv "${ADDRESS}" << EOF
binary
put "${JOB_OUT_PATH}" "${FILENAME}"
ls . "$TEMPFILE"
quit
EOF
    # Check if the file exists in the FTP directory listing
    if /bin/cat "$TEMPFILE" | /usr/bin/grep "$FILENAME" >/dev/null 2>&1; then
        # FTP upload ok
        RETURN_CODE=0
    else
        # FTP upload failed
        RETURN_CODE=1
    fi
    # Remove temporary file again
    /bin/rm "$TEMPFILE" >/dev/null 2>&1
else
    # Send mail to admin about the failed job
    RECIPIENT="admin@example.com"
    MESSAGE="Job '$JOB_NAME' with id $JOB_ID dropped with reason:
        $JOB_REASON"
    /usr/sbin/sendmail $RECIPIENT << EOF
From: "Episode Engine" <noreply@example.com>
Subject: Job '$JOB_NAME' failed

$MESSAGE
EOF
fi

exit $RETURN_CODE
```

Appendix A Supported formats

The following media formats and codecs are supported by **Episode Engine**:

3GPP (`.3gp`)

The 3GPP (3rd Generation Partnership Project) video format is based on the ISO/IEC 14496 (MPEG-4) media file format and intended for mobile phones. It is defined in 3GPP TS 26.244: *Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; Transparent end-to-end packet switched streaming service (PSS); 3GPP file format (3GP)*, see `http://www.3gpp.org/` for additional information.

Supported codecs: AAC, AMR NB, H.263, H.264, MPEG-4.

Restrictions: H.264 High Profile is input only. Multi-Bit Rate is output only.

Pro adds: H.264 High Profile support for output. HE-AAC, but for output only.

3GPP2 (`.3g2`)

The 3GPP2 (3rd Generation Partnership Project 2) video format is based on the ISO/IEC 14496 (MPEG-4) media file format and intended for mobile phones. It is defined in 3GPP2 C.S0050-B: *3GPP2 File Formats for Multimedia Services*, see `http://www.3gpp2.org/` for additional information.

Supported codecs: AAC, AMR NB, EVRC, H.263, H.264, MPEG-4, QCELP.

Restrictions: EVRC, H.264 High Profile, QCELP are input only.

Pro adds: EVRC, H.264 High Profile, QCELP support for output. HE-AAC, but for output only.

3GPP2 (EZMovie) (`.3g2`)

The 3GPP2 format can be extended with the EZMovie features developed by KDDI Corporation. Among other things, EZMovie lets a distributor limit how many times a file is played. See `http://www.au.kddi.com/ezfactory/tec/spec/ezmovie01.html` for additional information. (In Japanese.)

Supported codecs: AAC, AMR NB, EVRC, H.263, H.264, MPEG-4, QCELP.

Restrictions: EZMovie is only available in Pro and is output only.

ADTS (.aac)

Audio Data Transport Stream is a wrapper format for AAC-encoded audio files. It is defined in ISO/IEC 13818: *Information technology – Generic coding of moving pictures and associated audio information – Part 7: Advanced Audio Coding (AAC)*. See http://www.iso.org/ for additional information.

Supported codecs: AAC.

Restrictions: HE-AAC not supported for input.

AIFF (.aif)

The Audio Interchange File Format was developed by Apple. It is described in *Inside Macintosh: Sound*. See http://developer.apple.com/ for additional information.

Supported codecs: PCM.

AMC (.amc)

AMC is based on the MPEG-4 standard and has been developed by KDDI Corporation. It supports the EZMovie features. Among other things, EZMovie lets a distributor limit how many times a file is played. See http://www.au.kddi.com/ezfactory/tec/spec/ezmovie01.html for additional information. (In Japanese.)

Supported codecs: MPEG-4, QCELP

Restrictions: EZMovie files with distribution restrictions are only supported for output.

Pro adds: EZMovie only available in Pro.

AMR (.amr)

AMR (Adaptive Multi-Rate) is a mandatory audio codec in 3GPP. The file format is defined in IETF RFC 4867: *RTP Payload Format and File Storage Format for the Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (AMR-WB) Audio Codecs*. See http://www.3gpp.org/ for additional information.

Supported codecs: AMR Narrowband.

Restrictions: Input only.

Pro adds: AMR support for output.

ATSC A/52 (.a52)

ATSC A/52 is an audio format developed by Advanced Television Systems Committee, Inc. It is defined in ATSC A/52B: *Digital Audio Compression Standard*

(AC-3, E-AC-3) Revision B. See `http://www.atsc.org/` for additional information.

Supported codecs: ATSC A/52.

AVI (`.avi`)

AVI (Audio Video Interleave) is a multimedia container format developed by Microsoft. It is described in *AVI RIFF File Reference*. See `http://msdn.microsoft.com/` for additional information.

Supported codecs: DV25, DVCPRO25, DVCPRO50, MJPEG, MP3, MPEG-4 (*DivX*, *XViD*, FMP4), PCM, RGB16 (555), RGB16 (556), RGB24, RGB32, UYVY, Windows RGB, YCbCr 4:2:0, Y8, YUY2, YV16, YVU16, YVU9, YV12.

Restrictions: DivX, DVCPRO25, DVCPRO50, FMP4, MJPEG, Windows RGB, XviD, YCbCr only supported for input. **Episode Engine** uses QuickTime to read AVI files, you can thus extend the number of codecs available by installing additional QuickTime components (popular such codecs are indicated by *italics* in the list above). However, we do not guarantee full functionality of, nor offer helpline support for any such third party components.

Pro adds: DVCPRO25, DVCPRO50, Windows RGB, YCbCr supported for output.

DPX (`.dpx`)

DPX (Digital Picture eXchange) is derived from the earlier Kodak Cineon format. DPX is defined in SMPTE 268M: *File Format for Digital Moving-Picture Exchange (DPX), Version 2.0*. See `http://www.smpte.org/` for additional information.

Supported codecs: RGB

Restrictions: Input only.

DV (`.dv`)

DV (Digital Video) has been developed by several producers of video cameras. DV is defined in IEC 61834: *Recording - Helical-scan digital video cassette recording system using 6,35 mm magnetic tape for consumer use (525-60, 625-50, 1125-60 and 1250-50 systems)*, DVCPRO and DVCPRO50 are defined in SMPTE 314M: *Television—Data Structure for DV-Based Audio, Data and Compressed Video—25 and 50 Mb/s*. See `http://www.iec.ch/` and `http://www.smpte.org/` for additional information.

Supported codecs: DV25, DVCPRO25, DVCPRO50.

Restrictions: DVCPRO25 and DVCPRO50 only supported for input.

Pro adds: DVCPRO25 and DVCPRO50 supported for output.

Flash (`.flv`)

The Adobe Flash video format is defined in *Video File Format Specification, Version 10*. See `http://www.adobe.com/devnet/flv/` for addition information.

Supported codecs: H.263, MP3, VP6.

Flash (`.swf`)

The Adobe Small Web Format format is a multimedia wrapper format. It is defined in *SWF File Format Specification, Version 10*. See `http://www.adobe.com/devnet/swf/` for additional information.

Supported codecs: H.263, MP3, VP6.

Restrictions: Only audio and video data are supported.

GXF (`.gxf`)

GXF (General eXchange Format) is an interchange format for storage and data transfer originally developed by Grass Valley Group. It is defined in SMPTE 360M: *General Exchange Format (GXF)*. See `http://www.smpte.org/` for additional information.

Supported codecs: MPEG-2, PCM.

Restrictions: Input only.

Pro adds: GXF supported for output.

MP3 (`.mp3`)

Properly MPEG-1 Audio Layer III. It is defined in ISO/IEC 11172-3: *Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 3: Audio*. See `http://www.iso.ch/` for additional information.

Supported codecs: Lame MP3

MPEG Elementary Stream (`.m1a, .m1v, .m2v, .mpg`)

An MPEG Elementary stream contains a single medium, audio or video, and can in turn be contained in a Program Stream. MPEG-1 elementary streams are defined in ISO/IEC 11172-1: *Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 1: Systems*, MPEG-2 elementary streams in ISO/IEC 13818-1: *Information technology – Generic coding of moving pictures and associated audio information: Systems*. See `http://www.iso.ch/` for additional information.

Supported codecs: AES, MPEG Audio, MPEG-1, MPEG-2

Restrictions: AES is input only.

MPEG Program Stream (`.mpg`)

An MPEG program stream contains elementary streams. Program streams are intended for reliable media such as DVD or SVCD. MPEG-1 program streams are defined in ISO/IEC 11172-1: *Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 1: Systems*, MPEG-2 program streams in ISO/IEC 13818-1: *Information technology – Generic coding of moving pictures and associated audio information: Systems*. See `http://www.iso.ch/` for additional information.

Supported codecs: AAC, AES, ATSC A/52, H.264, MPEG Audio, MPEG-1, MPEG-2, MPEG-4, PCM

Restrictions: AAC is output only. AES, H.264 High Profile is input only.

Pro adds: H.264 High Profile supported for output.

MPEG Transport Stream (`.m2t, .ts`)

An MPEG Transport Stream is intended for broadcast media where packets may be lost and viewers have to be able to enter a transmission in mid-stream. Elementary streams are interleaved (muxed) on the Transport Stream. MPEG-2 program streams are defined in ISO/IEC 13818-1: *Information technology – Generic coding of moving pictures and associated audio information: Systems*. See `http://www.iso.ch/` for additional information.

Supported codecs: AES, ATSC A/52, H.264, HDV, MPEG Audio, MPEG-1, MPEG-2, PCM.

Restrictions: Only supported for input.

Pro adds: Support for input and output. AAC, MP3, and MPEG-4 support for output only.

MPEG-4 (`.m4a, .m4b, .m4v, .mp4`)

MPEG-4 is intended to improve on the earlier MPEG standards. The `.m4a`, `.m4b`, and `.m4v` versions are adapted for iPods as audio, audiobook and video specialisations, respectively. PlayStation Portable can play MPEG-4 files, but requires that they be named `M4Vxxxxx.mp4`, where `xxxxx` is five decimal digits, and stored in the directory `E:\MP _ROOT\100MNV01` on the PSP. MPEG-4 is defined in ISO/IEC 14496: *Information technology – Coding of audio-visual objects*. See `http://www.iso.ch/` for additional information.

Supported codecs: AAC, H.264, MPEG-4.

Restrictions: H.264 High Profile is input only.

Pro adds: H.264 High Profile and HE-AAC supported for output.

MXF (`.mxf`)

The Material eXchange Format is a wrapper format. There are currently major interoperability problems with different implementations of MXF, so interoperability has to be tested for each case. The MXF file format is defined in SMPTE 377M: *Television - Material Exchange Format (MXF) – File Format Specification*. See `http://www.mxf.info/` for additional information.

Supported codecs: AES, BWF, D-10/IMX, DV25, DVCPRO25, DVCPRO50, *DVCPROHD*, JPEG2000, MPEG-2, MPEG-4, XDCam HD. Codecs in *italics* require third party plugins. While any installed codec plugins will be used, we do not guarantee full functionality of nor offer helpline support for any such third party components.

Restrictions: All formats are input only.

Pro adds: AES, BWF, D-10/IMX, *DNxHD*, DV25, DVCPRO25, DVCPRO50, *DVCPROHD*, MPEG-2, XDCam HD supported for output.

OGG (`.ogg`)

Ogg is an open media wrapper format designed for efficient streaming and manipulation. It is defined in RFC 3533: *The Ogg Encapsulation Format Version 0*. See `http://www.Xiph.Org/` for additional information.

Supported codecs: Vorbis.

Restrictions: Output only.

QuickTime (`.mov`)

QuickTime is a multimedia framework developed by Apple. It is defined in *QuickTime File Format Specification*. See `http://developer.apple.com/documentation/QuickTime/` for additional information.

Supported codecs: AAC, AMR NB, Apple Prores, Apple Video, *Avid*, *Avid DNxHD*, *Avid DV*, *Avid Meridien*,

Restrictions: B

Pro adds: lackmagic, Cinepak, D-10/IMX, DV25, DVCPRO25, DVCPRO50, *DVCPRO100*, H.261, H.263, H.264, HDV, IMA, Mace 3:1, Mace 6:1, *Media 100*, MJPEG, MP3, MPEG-4, PCM, RGB, RGB16 (555), RGB16 (556), RGB24, RGB32, Sheer Video, Sorenson Video 1, XDCAM HD, YCbCr (YUV), UYVY, Y8, YUY2, YV16, YVU16, YVU9, YV12.D-10/IMX, HDV, MJPEG, Targa Cine YUV are input only. QuickTime reference files are input only. Timecodes are not supported by the native QuickTime importer—this mainly affects reference files. You can extend the number of codecs available by installing additional QuickTime components (popular such codecs are indicated by *italics* in the list above). However, we do not guarantee full functionality of, nor offer helpline support for, any such third party components.D-10/IMX, HDV, HE-AAC, MJPEG supported for output.

Wave (.wav)

The Waveform audio format was developed by Microsoft and IBM. It is described in *Multiple Channel Audio Data and WAVE Files*. See http://www.microsoft.com/ for additional information.

Supported codecs: PCM.

Windows Media (.wma, .wmv)

Windows Media is a proprietary multimedia framework developed by Microsoft. See http://www.microsoft.com/ for additional information.

Supported codecs: Intellistream, VC-1, Windows Media, Windows Media MBR, WMA Pro, WMA Standard.

Restrictions: Only the largest stream is read from Intellistream multi-bit rate files. ASF files can only be read if they have WMV9 content.

Appendix B **engine**

engine is a command line tool for communicating with **Episode Engine**. You can use it interactively or in shell scripts. In the default installation it is located in `/usr/local/pwce/bin/`.

engine takes a set of options common to all commands:

--host ***host*** Optional. The host on which the **Episode Engine** controller is running. *host* can be a host name or an IP address. The default is **`localhost`**.

--port ***port*** Optional. The port number for communication with **Episode Engine**. The default is **`40402`**.

--password ***password*** Optional. The client password for **Episode Engine**. The default is **`anonymous`**.

--timeout ***time*** Optional. The number of seconds **engine** will wait for a response from **Episode Engine**. The default is **`2.0`**.

The commands to **engine** are in the form of keywords. Commands optionally take arguments.

help List all available commands to **engine** and their arguments.

list depot List the storage depots available to **Episode Engine**. Arguments:

-n ***name*** **| --name** ***name*** Optional. The name of a depot. If no name is given, all depots are listed.

-s ***style*** **| --style** ***style*** Optional. Output format. Either of

xml XML as in the `depots.conf` file.

key-value | kv ***`key=value`*** lines.

default An indented list in braces.

list setting Display the contents of one or several settings files. Arguments:

-p ***setting*****, . . . | --settingid** ***setting*****, . . .** Mandatory. The path of a settings file. Several files can be listed, separated by "`,`", their contents will be shown immediately following each other.

list settings List settings or settings groups. Arguments:

-g ***groupid*****, . . . | --groupid** ***groupid*****, . . .** Optional. The path of a settings group. Several groups can be listed, separated by "`,`". Any arguments not preceded by **–** or **––** switches are interpreted as group names.

-s *style* | --style *style* Optional. Output format. Either of

flat One entry per line.

list Entries on a line, separated by "`,`". This can be used as input to other commands that require comma-separated arguments.

tree Default. Entries are shown in a tree format. Implies the **`--recursive`** option. Ignores the **`--no-groups`** and **`--no-settings`** options.

-r | --recursive Optional. List the contents of groups within groups.

-n | --names Optional. List basenames instead of full path names.

--no-groups Optional. Do not list setting groups.

--no-settings Optional. Do not list settings.

list job Output information on one or several jobs. The default output format is a single line with the selected properties for each job. The job ID and current state are always printed. The possible job states are:

Created The job has been created by **Episode Engine**.

Queued The job has been inserted in the job queue.

Running The job has started and is running.

Stopped The job has been stopped by the operator.

Failed The job has failed and will not be rerun.

Finished The job has finished successfully.

Arguments:

-i *jobid* ... | --id *jobid* ... Optional. The id of the job to be listed. If no job id is given, all jobs will be listed.

-n | --name Optional. Include the job name in the output.

-h | --history Optional. Each job will display all earlier states in addition to the current state.

-p | --progress Optional. Indicate the progress value in the output. This is only useful if the job is running, all other job states will have a progress value of `0`.

-r | --reason Optional. Display additional information about the job state. The job states `Stopped`, `Failed`, and `Finished` can have the reasons `NotStarted`, `BadCommunication`, `BadJob`, `Failed`, `Lost`, `Cancelled`, `Aborted`, `Finished`. All other states will display the reason `Unspecified`.

-v | --vertical Optional. List the job information on separate lines.

-a | --archive Optional. In addition to the currently executing jobs, list finished jobs, as far back as set in `engine.conf`.

add setting Upload a setting to **Episode Engine**. The setting is then shared by all users. Arguments:

-f *filename* | --filename *filename* Mandatory. The filename of the setting. Any arguments not preceded by **-** or **--** switches are interpreted as filenames.

-n *name* | --name *name* Optional. The name given to the setting. If no name is given, *filename* is used.

-g *groupid* | --groupid *groupid* Optional. The path of the group to place the setting in. If no group path is given, the setting will placed in the root group.

add settinggroup Create a new settings group. Arguments:

-n *name* | --name *name* Mandatory. The name of the new group. Any arguments not preceded by **-** or **--** switches are interpreted as group names, in which case only the last one will be used.

-g *groupid* | --groupid *groupid* Optional. The path of an existing group to place the new group in. If no path is given, the new group will be placed in the root group.

add file | add media Submit a file and a set of settings for execution. Arguments:

-f *filename* | --filename *filename* | -n *filename* | --input *filename* Mandatory. The source file to be transcoded. Any arguments not preceded by **-** or **--** switches are interpreted as file names for execution.

-o *filename* | --output *filename* Optional. Set output path. Default output directory for local source files is the same as the input directory, for files retrieved with ftp or smb monitors it is the default depot.

-s *filename* | --dstpath *filename* Optional. Set the output directory. Ignored if a **-o**/**--output** argument is given.

-p *settingid* ... | --settingid *settingid* ... | -x *settingid* ... | --encoder *settingid* ... Conditionally mandatory. Each setting supplied will create a job. Note that multiple settings should not be used in conjunction with the **-o**/**--output** option, as the output of all settings will be written to that file, the last job to finish overwriting the previous results.

-g *groupid* ... | --groupid *groupid* ... Conditionally mandatory. All settings in each listed group will be used for creating jobs.

One of **--settingid** and **--groupid** must be given.

-y *filename* | --intro *filename* Optional. Set the intro/bumper file. Ignored if no setting specifies an intro.

-z *filename* | --outro *filename* Optional. Set the outro/trailer file. Ignored if no setting specifies an outro.

-k *filename* | --watermark *filename* Optional. Set the watermark file. Ignored if no setting specifies a watermark.

-m *filename* | --metafile *filename* Optional. Set the input metadata file. Ignored if no setting specifies an input metadata file.

-a *filename* | --audiofile *filename* Optional. Set an audio file to be added to the video input. Ignored if no audio file is specified in any setting.

--replace *tag string* ... Optional. Replaces the tag *tag* (!tag!) in the setting with *string*. Tags are case insensitive.

--meta *key value* ... Optional. When the job has finished executing, any event script run will receive the value in *value* in the environment variable *key*. Any number of pairs of *key* and *value* can be given. See chapter 4, *Reference section* for more information on event scripts.

--split Optional. Run the job in split-and-stitch mode.

--min ***time*** Optional. Set the minimum duration in seconds of a split segment. Ignored unless **`--split`** is specified.

--min ***number*** Optional. Set the maximum number of split segments. Ignored unless **`--split`** is specified.

--noreg Optional. Fail if the job cannot be split. Ignored unless **`--split`** is specified.

--prio ***prio*** Optional. Set the job priority in the range 0–65535. A higher number corresponds to higher priority.

--jobname ***jobname*** Optional. Set a name for the job. You can modify the job name with the following special sequences:

#I Incremental count.

#P Setting ID.

#N Setting name.

This will not change the output file name, which will always be the source file name concatenated with the settings name. Any earlier files with the same name will be overwritten.

--id-out | --id-out-vertical Optional. Output the job IDs. **`--id-out-vertical`** will output the job IDs separated by newlines.

-w | --wait [-v | --verbose] Optional. Wait for the jobs to finish before exiting **engine**. **`--verbose`** will show the progress of the jobs.

add directory | add folder Submit files within a directory and a set of settings for execution. Files can be selected or excluded based on their filenames. Arguments:

-d ***directory*** **| --directory** ***directory*** Mandatory. The directory containing the files to be submitted. Any arguments not preceded by **-** or **--** switches are interpreted as directory names.

-r [***maxdepth*** **] | --recursive [** ***maxdepth*** **]** Optional. Recursively search all subdirectories. *maxdepth* limits the number of levels searched.

-i ***regexp*** **| --include** ***regexp*** Optional. Only submit files matching the regular expression *regexp*.

-e ***regexp*** **| --exclude** ***regexp*** Optional. Do not submit files matching the regular expression *regexp*.

-t | --test Optional. Print matching files only, do not submit any jobs.

-s ***filename*** **| --dstpath** ***filename*** Optional. Set the output directory.

-p ***settingid ...*** **| --settingid** ***settingid ...*** **| -x** ***settingid ...*** **| --encoder** ***settingid ...*** Conditionally mandatory. Each setting supplied will create a job.

-g ***groupid ...*** **| --groupid** ***groupid ...*** Conditionally mandatory. All settings in each listed group will be used for creating jobs.

One of **`--settingid`** and **`--groupid`** must be given.

-y ***filename*** **| --intro** ***filename*** Optional. Set the intro/bumper file. Ignored if no setting specifies an intro.

-z ***filename*** **| --outro** ***filename*** Optional. Set the outro/trailer file. Ignored if no setting specifies an outro.

-k *filename* | --watermark *filename* Optional. Set the watermark file. Ignored if no setting specifies a watermark.

-m *filename* | --metafile *filename* Optional. Set the input metadata file. Ignored if no setting specifies an input metadata file.

-a *filename* | --audiofile *filename* Optional. Set an audio file to be added to the video input. Ignored if no audio file is specified in any setting.

--replace *tag string ...* Optional. Replaces the tag *tag* (!tag!) in the setting with *string*. Tags are case insensitive.

--meta *key value ...* Optional. When the jobs have finished executing, any event script run will receive the value in *value* in the environment variable *key*. Any number of pairs of *key* and *value* can be given. See chapter 4, *Reference section* for more information on event scripts.

--prio *prio* Optional. Set the job priority in the range 0–65535. A higher number corresponds to higher priority.

--jobname *jobname* Optional. Set a name for the job. You can modify the job name with the following special sequences:

#I Incremental count.

#P Setting ID.

#N Setting name.

This will not change the output file name, which will always be the source file name concatenated with the settings name. Any earlier files with the same name will be overwritten.

--id-out | --id-out-vertical Optional. Output the job IDs. **`--id-out-vertical`** will output the job IDs separated by newlines.

-w | --wait [-v | --verbose] Optional. Wait for the jobs to finish before exiting **engine**. **`--verbose`** will show the progress of the jobs.

remove setting Delete a shared setting. Arguments:

-p *settingid ...* | --settingid *settingid ...* Mandatory. The path to the setting to be deleted. Any arguments not preceded by **-** or **--** switches are interpreted as setting paths.

remove settinggroup Delete a shared setting group and all settings it contains. Arguments:

-g *groupid ...* | --groupid *groupid ...* Mandatory. The path to the settings group to be deleted. Any arguments not preceded by **-** or **--** switches are interpreted as group paths.

remove job Abort a job. Arguments:

-i *jobid ...* | --id *jobid ...* Conditionally mandatory. Any arguments not preceded by **-** or **--** switches are interpreted as job IDs.

--all Conditionally mandatory. Abort all jobs in **Episode Engine**.

One of **`--id`** and **`--all`** must be given.

-c | --cancel Stop and requeue a job instead of aborting.

analyze [-f *filename* | --filename *filename*] Output the following information on the media file *filename*: Duration and number of media tracks; then for each media track: Track type, media type and duration; for video tracks: Width, height, aspect ratio and frame rate; for audio tracks: Number of channels, bits/sample, bytes/sample and sample rate.

B.1 Examples

```
prompt> engine list setting 'Templates/By Format/Audio Only/AIFF/24bit_96kHz.setting'
<?xml version="1.0"?>
<!DOCTYPE job SYSTEM "job.dtd">
<job version="1.1">
<!--5.0-->
  <description/>
  <meta-data meta-file="no">
    <meta-group type="movie"/>
    ...
```

Note the use of quotes to protect the spaces in the setting path.

```
prompt> engine list settings 'Templates/By Format/Audio Only/AIFF' --style list --name
24bit_96kHz,24bit_48kHz,16bit_48kHz,16bit_44kHz

prompt> engine list settings --style tree --groupid 'Templates/By Format/Audio Only'
.
|== Templates/By Format/Audio Only/AAC m4a
    |-- Templates/By Format/Audio Only/AAC m4a/160kbit_44kHz_stereo.setting
    |-- Templates/By Format/Audio Only/AAC m4a/192kbit_44kHz_stereo.setting
    |-- Templates/By Format/Audio Only/AAC m4a/128kbit_44kHz_stereo.setting
    |-- Templates/By Format/Audio Only/AAC m4a/64kbit_44kHz_mono.setting
    |-- Templates/By Format/Audio Only/AAC m4a/256kbit_48kHz_stereo.setting
    `-- Templates/By Format/Audio Only/AAC m4a/32kbit_32kHz_mono.setting
|== Templates/By Format/Audio Only/AIFF
    |-- Templates/By Format/Audio Only/AIFF/24bit_96kHz.setting
    |-- Templates/By Format/Audio Only/AIFF/24bit_48kHz.setting
    |-- Templates/By Format/Audio Only/AIFF/16bit_48kHz.setting
    `-- Templates/By Format/Audio Only/AIFF/16bit_44kHz.setting
|== Templates/By Format/Audio Only/Surround
    |-- Templates/By Format/Audio Only/Surround/WMA_Lossless_96kHz_surround.setting
...

prompt> engine add file /Users/Shared/Demo/CIMG1406.AVI --groupid Test --id-out-vertic
29
30
31
prompt> engine  list job --vertical
29
Queued
30
Queued
31
Queued

prompt> engine --password fnord remove job --all
prompt>
```

```
prompt> engine analyze -f ~/Movies/CIMG1406.AVI
duration            : 10.531322
number of tracks    : 2

track type          : video
media type          : jpeg
duration            : 10.531322
width               : 320
height              : 240
aspect ratio        : 0/0 0.000000
frame rate          : 14.718000

track type          : audio
media type          : pc8U
duration            : 10.531250
channels            : 1
bits/sample         : 8
bytes/sample        : 1
sample rate         : 8000.000000
```

Index