



Final Cut Server

Integration Notes

Version 1.0
May 28, 2008

Script responses

A Response Action created using "Run an external script or command" is known as a script response. These are created via the Response pane of the Administration window in the Final Cut Server client application. In order to access the Administration window you must be logged in to Final Cut Server as a user with administration privileges.

In defining a script action you specify two strings: 'Command Path' and 'Command Parameters'. Command path is the pathname of the executable file. Command Parameters is a string which will undergo substitution and simple parsing to become the argument list for the executable.

The executable file can be either a binary or some sort of script, provided the script is directly executable - i.e a shell script (or other '#!' file) whose executable bit is on. The script must be executable by the user associated with Final Cut Server. (See below.)

Environment

Environment Variables

A number of environment variables are present, but the two important ones are:

- PATH=/usr/bin:/bin:/usr/sbin:/sbin
- PWD='/Library/Logs/Final Cut Server'

The effective user ID will be the UID associated with Final Cut Server. This is the UID of the user who installed Final Cut Server.

Your script is run as a child process of the Final Cut Server store daemon process (fcsvr_stored), so your parent process ID (PPID) will be the process ID of the currently running instance of fcsvr_stored.

Command Path

The command path can either be a full pathname or may be just the name of the executable if it is located in one of the directories found in the PATH variable.

File Descriptors

There is nothing useful on standard input. Output directed to standard output or standard error will end up in the log file: '/Library/Logs/Final Cut Server/fcsvr_stored_<pid>.log', where <pid> is the process ID of the currently running store daemon.

Exit Status

The exit status of the executable is ignored.

Unicode

All the various strings are UTF-8. That is, command path, command parameters and metadata fields may each contain Unicode characters. Your script will get arguments that are UTF-8 strings.

Argument List

To create the argument list the Command Parameters string first undergoes substitution. Any substring that has the form of a metadata field name surrounded by square brackets (e.g. [Title] or [Owner]) will be substituted with the value of that metadata field from the object that triggers the response. The following caveats apply:

- The metadata field must be a field of the object that triggered the response.
- The metadata field name must match exactly. The names are case-sensitive.
- No substitution occurs if
 - the metadata field is not set
 - the metadata field is not part of the object
 - the name does not match exactly

After the substitution the string is parsed into separate arguments based on white space. White space may be quoted using either single or double quote characters. Either quote character will quote the other. There is no escape character. A quote character only begins a quote if it is:

- the first character of the string or
- immediately follows whitespace or
- immediately follows a closing quote.

A closing quote always marks the end of argument, even if immediately followed by a non-white space character.

The parser eliminates any empty string arguments from the argument list prior to the script response being invoked. It is impossible to pass an empty string as an argument to the script response.

Some examples may be helpful. I'll enclose the strings in angle brackets.

Let's assume our [Title] is <Bob's file> and the [Owner] is <> (i.e. empty string).

- If the command parameters string is <[Title] [Owner]> then we would have two arguments <Bob's> and <file>
- If the command parameters string is <"[Title]" "[Owner]"> then we would have one argument <Bob's file>
- If the command parameters string is <'[Title]' '[Owner] '> then we would have four arguments <Bob>, <s>, <file> and <>. That last argument is a single space character that comes from the space between the two quoted.

The best option is to enclose arguments in double quotes, since those should rarely occur in the input. If you want to insure you always get an argument, you will need to add a character that you remove later. For example, set the command parameters is `<"[Title]"x[Owner]>` so you get two arguments `<Bob's file>` and `<x>`. Then in your script remove the leading 'x' from the second argument.

If you want to pass a number metadata fields to the script response, or you want to insure that no untoward parsing occurs then the best option is to have a Write XML response before the script response. Pass the [Title] to script (since the name of XML file will be "[Title].xml") so the script can pick up the file.

Customizing

The environment that your script inherits comes from the Final Cut Server store daemon process (fcsvr_stored). The daemon's environment is set by launchd. The launchd plist file for the store daemon is the file

```
'/Library/LaunchDaemons/com.apple.FinalCutServer.fcsvr_stored.plist'.
```

There are some tricks we can do with this file.

Adding environment variables

The plist has a dictionary EnvironmentVariables. We can add a key-value pair to this dictionary. For example:

```
<key>MY_DIR</key>
<string>/Misc/tmp</string>
```

Would provide an environment variable named "MY_DIR" to every script response.

Customizing the PATH variable

Let's say I have a number of scripts for script responses and I don't want to put them in one of the stock directories. Again we add a key-value pair to the EnvironmentVariables dictionary.

```
<key>PATH</key>
<string>/usr/bin:/bin:/usr/sbin:/sbin:/Misc/bin</string>
```

Now I can put all my scripts into /Misc/bin and call them by name without having to specify a path.

Note that the first part of the PATH variable is "/usr/bin:/bin:/usr/sbin:/sbin" - i.e. the default value for the PATH. You should always keep this portion of the PATH.

Caution

You may notice that the plist has a key called "WorkingDirectory". This does determine the working directory that script responses inherit, but you should not change the value, as the store daemon depends on it.

A note about GUI applications

As previously mentioned any scripting language you have installed may be used: shell, ruby, perl, python, AppleScript, etc.

You may be tempted to create an AppleScript that starts up a desktop application. For example, iChat. Don't do this. Because script responses are started from a daemon they can not reliably invoke GUI applications.

For the details see "Technical Note TN2083: Daemons and Agents"

Archive Device

There is one other type of customization script in Final Cut Server. When you configure an Archive device you can set a "Post-archive command" and a "Pre-restore command". Both of these values are simply the pathname of the command, no arguments can be specified. The executable will have exactly the same sort of environment as a script response. A single argument will be passed. The argument is the path of the asset being archived or restored. In both cases, this path is to the archive device.

Write XML & Read XML

Exporting - Write XML

Exporting of metadata for an entity is implemented using the "Write XML" response type. A response of this type has only one parameter, a destination directory on some device, where the files will be written. It can be triggered by any subscription rule - the response will write out the metadata of the entity which triggered the subscription. The name of the XML file will be based on the Title of the entity.

An example of what the XML for metadata output looks like is the following:

```
<?xmlversion="1.0"?>
<FinalCutServer>
  <getMdReply>
    <entityType="asset"entityId="/asset/2858">
      <metadata>
        <mdValuefieldName="Size"dataType="int64">42531</mdValue>
        <mdValuefieldName="Location"dataType="string">/indo</mdValue>
        <mdValuefieldName="CreateDate"dataType="dateTime">2001-09-2601:06:51+0</mdValue>
        <mdValuefieldName="StoredOn"dataType="string">pikkies2</mdValue>
        <mdValuefieldName="PromoVersion"dataType="string">NextWeek</mdValue>
        <mdValuefieldName="AssetNumber"dataType="integer">2858</mdValue>
        <mdValuefieldName="LastModified"dataType="dateTime">2005-07-0806:34:03+0</mdValue>
        <mdValuefieldName="Title"dataType="string">samarinda</mdValue>
      </metadata>
    </entityType>
  </getMdReply>
</FinalCutServer>
```

```

    </entity>
  </getMdReply>
</FinalCutServer>

```

Importing - Read XML

Metadata can be modified using the "Read XML" response type. The Read XML response has no parameters at all. Everything comes from the XML file which triggered the response. It is usually driven by a watcher. The name of the file is not significant.

The XML file should contain one or more setMd requests. The entity is identified by the entityId, the field(s) to be modified are identified by name. The entityId should be one you obtained earlier from a Write XML. Only fields that can exist in the entity are used. Extra fields are ignored.

An example of what the XML looks like is:

```

<?xml version="1.0" encoding="UTF-8"?>
<FinalCutServer>
  <request reqId="setMd" entityId="/asset/11447">
    <params>
      <mdValue fieldName="Title" dataType="string">Ibu at Darling Harbour</mdValue>
    </params>
  </request>
</FinalCutServer>

```

The <FinalCutServer> element wraps the entire set of requests - if there are to be multiple requests in a file then they should each be in their own <request> element within the one <FinalCutServer> element.

A note about creation

The Read XML response can not create an asset, only to make modifications to existing assets.

Schema

Below is a minimal XML Schema for the Read XML & Write XML. The schema does not specify values for the dataType attribute. The following values can appear in the current version: 'bool', 'coords', 'dateTime', 'float', 'int64', 'integer', 'string', 'time-code'.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="MetadataValueType">
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="fieldName" type="xsd:string"/>
        <xsd:attribute name="dataType" type="xsd:string"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
  <xsd:complexType name="MetadataListType">

```

```

    <xsd:sequence>
      <xsd:element name="mdValue" type="MetadataValueType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="EntityType">
    <xsd:sequence>
      <xsd:element name="metadata" type="MetadataListType"/>
    </xsd:sequence>
    <xsd:attribute name="entityId" type="xsd:string"/>
    <xsd:attribute name="entityType" type="xsd:string"/>
  </xsd:complexType>
  <xsd:complexType name="RequestType">
    <xsd:sequence>
      <xsd:element name="params" type="MetadataListType"/>
    </xsd:sequence>
    <xsd:attribute name="reqId" type="xsd:string"/>
    <xsd:attribute name="entityId" type="xsd:string"/>
  </xsd:complexType>
  <xsd:complexType name="GetMdReplyType">
    <xsd:sequence>
      <xsd:element name="entity" type="EntityType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SearchReplyType">
    <xsd:sequence>
      <xsd:element name="entity" type="EntityType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="fcsvrDocument">
    <xsd:choice>
      <xsd:element name="request" type="RequestType"/>
      <xsd:element name="getMdReply" type="GetMdReplyType"/>
      <xsd:element name="searchReply" type="SearchReplyType"/>
    </xsd:choice>
  </xsd:complexType>
  <xsd:element name="FinalCutServer" type="fcsvrDocument"/>
</xsd:schema>

```