Final Cut Pro X Workflows Developer Guide

Developer

Contents

Figures

Introduction

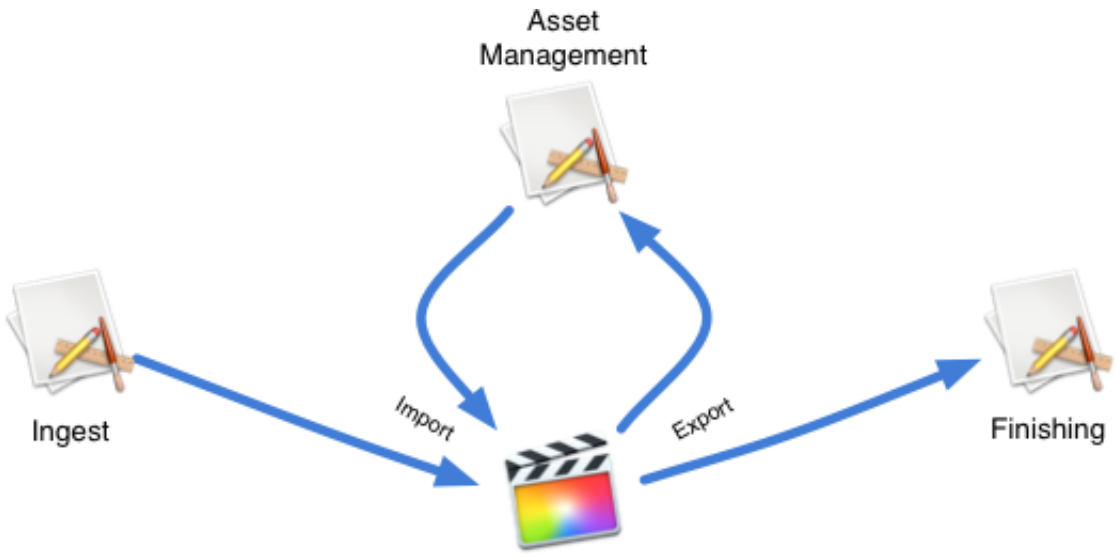
This document describes how Final Cut Pro X works with other applications as part of a workflow, and the requirements for such applications to exchange data and interact with Final Cut Pro X. The discussion applies to Final Cut Pro X v10.1 or later.

At a Glance

Final Cut Pro X works with other applications in various workflows. In some cases, another application ingests footage from an external device for editing in Final Cut Pro X. In other cases, the Final Cut Pro X editing results are passed to another application for compositing, color grading, or finishing. There are also cases when an asset management system manages the media assets used by Final Cut Pro X.

Final Cut Pro X exchanges media data and edit descriptions with other applications as shown in Figure I-1.

Figure I-1 Final Cut Pro X workflow



There are two ways in which Final Cut Pro X exchanges media data and edit descriptions with other applications:

- Incorporating media assets, along with additional information such as metadata and rough cuts, from another application. This process is called *importing*.

Relevant Chapter: Importing (page 7)

- Delivering edits to another application as rendered media, along with a description of the edits. This process is called *exporting*.

Relevant Chapter: Exporting (page 11)

In either case, the Final Cut Pro X XML Format (FCPXML) is used to represent rough cuts, edit descriptions, and metadata.

Relevant Document: *Final Cut Pro X XML Format*

Final Cut Pro X interacts with another application through Apple events to request information or trigger an action. Your application must respond to incoming Apple events.

Relevant Document: *Cocoa Scripting Guide*

In particular, for controlling the export process, Final Cut Pro X uses the events and classes defined in the ProVideo Asset Management scripting terminology suite.

Relevant Section: Scripting Definition (page 15)

Relevant Appendix: Appendix A: ProVideo Asset Management Suite (page 22)

How to Use This Document

To learn how to incorporate media from your application into Final Cut Pro X, see Importing (page 7).

To learn how to hand off editing results from Final Cut Pro X to your application, see Exporting (page 11).

For the requirements your application must meet to work with Final Cut Pro X in controlling the export process, refer to Application Requirements (page 14).

Prerequisites

This document assumes you have used Final Cut Pro X and have an understanding of the Final Cut Pro X XML Format (FCPXML). It also assumes a basic understanding of Apple events and Cocoa Scripting support to handle Apple events.

See Also

The following resources may be helpful while integrating your application with the Final Cut Pro X workflow:

- *Final Cut Pro X XML Format*
- Final Cut Pro X Help
- *Cocoa Scripting Guide*
- *AVFoundation Programming Guide*
- *Launch Services Programming Guide*

Importing

Overview

Final Cut Pro X can incorporate media that another application produces or manages as a collection of media files. You can use FCPXML to bundle them along with keywords, markers, and metadata, or to describe rough cuts. The application initiates the process by sending an Apple event to Final Cut Pro X.

Importing XML

FCPXML allows you to transfer the following items from your application into Final Cut Pro X:

- A collection of media assets
- A timeline sequence
- Keywords, markers, and metadata

Refer to *Final Cut Pro X XML Format* for more details.

Final Cut Pro X incorporates these items into its event and/or project when it imports an FCPXML document. Your application can initiate the process by sending an *Open Document* Apple event and indicating the file URL for the XML document to open.

Typically, you use Launch Services to send an *Open Document* Apple event to another application. Refer to the *Launch Services Programming Guide* for details on how to open documents programmatically with another application.

Working with Growing Media

Sometimes, you may want to start editing a clip while it is still being recorded or ingested. This is particularly useful if you need to deliver the result of editing before recording is finished.

Final Cut Pro X supports importing the following types of growing media files while additional media is still being recorded or ingested onto them by other third-party devices or applications:

- QuickTime Movie files with movie fragments

- MXF files

> **Note:** Normal QuickTime movies require a consolidated table of contents to allow random access of media. In contrast, movie fragments allow you to distribute the table of contents, which is suitable for accessing media while additional content is being added. Consult the *AVFoundation Programming Guide* to learn how to create a QuickTime Movie file with movie fragments.

Final Cut Pro X supports growing files by keeping track of the file modification date and updating the information of the imported asset, including duration, when it detects that the file has been modified.

> **Note:** Final Cut Pro X checks only the file modification date when it transitions *to* being the active application. No further updates appear until the transition takes place again. As a result, you see additional media in a growing media file only when this transition occurs.

Working with Custom Metadata

Final Cut Pro X can incorporate custom metadata described in an FCPXML document. (Refer to the *Final Cut Pro X XML Format* documentation for more details.) These custom metadata items do not appear in the Info Inspector until you add the respective fields to a metadata view.

Final Cut Pro X supports a mechanism for adding metadata definitions and view sets through external definition files. This mechanism allows facilities and third-party applications to install such definition files on each Final Cut Pro X station where the metadata definitions and view sets are needed.

The definition file is a plist file that Final Cut Pro X reads and uses to augment the interface with facility specific views. This plist file should be installed at either of the following locations:

- `/Library/Application Support/ProApps/Metadata Definitions/`
- `~/Library/Application Support/ProApps/Metadata Definitions/`

The plist file has the following structure:

- Root (Dictionary)

  com.apple.proapps.MetadataDefinitions (Definition Dictionary)

  com.apple.proapps.MetadataViewSets (View Set Dictionary Array)

- Definition (Dictionary)—The key is the Metadata identifier key (for example, `com.yourCompany.yourApp.yourCustomMetadata`).

displayName (String)—The Name to be displayed in the Final Cut Pro X Inspector and View Set Editor.

displayDescription (String)—The Description to be displayed in the Final Cut Pro X Metadata View Set Editor.

type (String)—The data type of this metadata, for example, string, boolean, integer, float (these are the same Metadata types as are used in FCPXML).

source (String)—The Origin of this metadata (for example, 'EXIF', 'Apple', 'BBC', '<your company name>').

editable (Boolean)—Whether the user can modify this metadata.

- View Set (Dictionary)

  displayName (String)

  keys (String Array)—An array of metadata identifiers (in reverse DNS style).

Example plist File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>com.apple.proapps.MetadataDefinitions</key>
    <dict>
      <key>com.yourCompany.yourApp.yourCustomMetadata</key>
      <dict>
        <key>displayDescription</key>
        <string>Description of your custom metadata</string>
        <key>displayName</key>
        <string>Your custom metadata</string>
        <key>source</key>
        <string>custom</string>
        <key>type</key>
        <string>string</string>
      </dict>
    </dict>
    <key>com.apple.proapps.MetadataViewSets</key>
```

```
    <array>
      <dict>
        <key>displayName</key>
        <string>Your Application's Set</string>
        <key>keys</key>
        <array>
          <string>com.yourCompany.yourApp.yourCustomMetadata</string>
        </array>
      </dict>
    </array>
  </dict>
</plist>
```

Exporting

Overview

Final Cut Pro X can hand off the results of editing to another application by exporting the following:

- Rendered media of the edited timeline
- Description of the edit as XML

Final Cut Pro X manages export configurations as Share Destinations. A share destination can have a target application that receives the export output when applicable. Final Cut Pro X interacts with the application to control certain aspects of the export operation.

Export Process

Final Cut Pro X exports the rendered timeline as a media file, either transcoding it into a particular format or leaving it in the rendered format (that is, Apple ProRes). The exported media file has a set of metadata, referred to as Share Metadata, that you can choose and specify values for when you start the export operation.

Final Cut Pro X can also export its events or projects as FCPXML. Final Cut Pro X can include metadata associated with a project or an asset in the exported XML. Only metadata items in the selected metadata view are included in the output. You can specify a metadata view to determine the set of metadata items that goes into the output.

Your application, when specified as the target of a share destination, can influence the export operation, as discussed in Custom Share Destination (page 11).

Custom Share Destination

Final Cut Pro X uses a share destination to manage information associated with a particular type of export or share.

You can create your own custom share destination and include the following information:

- Export settings—The choice of codec, file format, and so forth.
- Target application—The target application where the exported files are to be sent.

Final Cut Pro X queries the target application for the following export control details:

- Whether to export XML as a description of the project in addition to the rendered media. This allows the user to export both the rendered project timeline and the project description XML in a single step.
- Which set of metadata to include in the exported XML. This also allows the target application to access the set of metadata.
- The location for the export output. This allows Final Cut Pro X to proceed without having to show the Save File dialog.

To create a custom share destination, refer to the *Work with destinations* section in the *Final Cut Pro X Help*.

Final Cut Pro X supports a mechanism for distributing custom share destinations to other Final Cut Pro X users or workstations as `.fcpxdest` files.

To create and import a destination `.fcpxdest` file:

1. Drag a destination from the Destinations list in the Destinations pane of the Final Cut Pro X Preference to a location on your system.
2. Drag the `.fcpxdest` file into the list of share destinations on another Final Cut Pro X workstation.

Refer to the *Share destinations between Final Cut Pro X users* section in the *Final Cut Pro X Help*. Alternatively, you can install `.fcpxdest` files in either of the following locations:

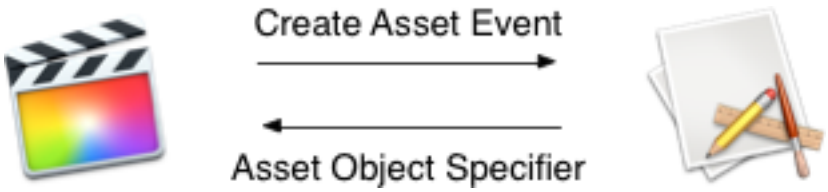
- `/Library/Application Support/ProApps/Share Destinations/`
- `~/Library/Application Support/ProApps/Share Destinations/`

Interaction with Target Application

When a project timeline is exported through a custom share destination with a target application specified, Final Cut Pro X first determines whether the application is capable of responding to inquires from Final Cut Pro X. It does this by inspecting the bundle plist of the application. The information that should be in the bundle plist is discussed in Signaling Application Capabilities (page 15).

Assuming Final Cut Pro X has determined that the application is capable of responding to its queries, it sends a *Create Asset* Apple event requesting a new placeholder asset to which the export output should be associated. This request also includes the name of the asset, the set of share metadata, and a list of metadata views. The

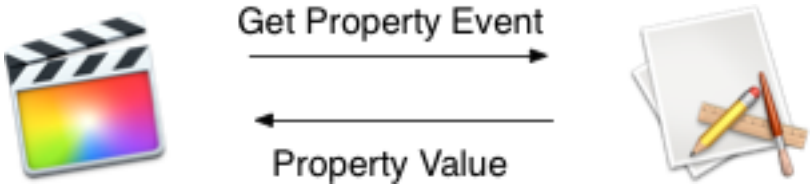
reason for providing metadata views here is to let the application determine the set of metadata that goes into the project XML. The application is expected to return an object specifier identifying the newly created placeholder asset object.



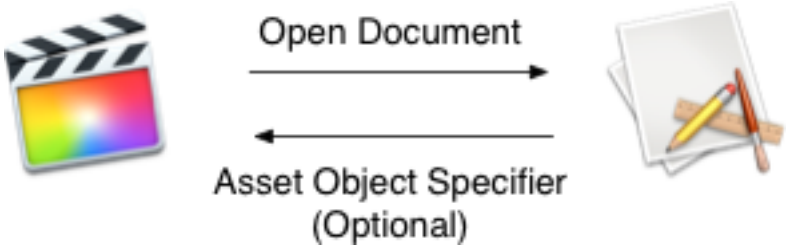
After the new asset object is successfully created, Final Cut Pro X asks the application for the following:

- The location for the export output files.
- Whether the application wants the project XML in addition to the rendered/transcoded media.
- An updated set of share metadata.
- The metadata view to include in the project XML.

Final Cut Pro X does this by sending a series of *Get Property* Apple events. The application is expected to return the appropriate property values.



With these pieces of information, Final Cut Pro X places the rendered/transcoded media file (and the XML file, if applicable) at the location returned from the application. When the export is complete, Final Cut Pro X notifies the application that the export output files are available by sending an *Open Document* Apple event. This event has the file URLs of the export output files.



Application Requirements

In order to control the aspects of the export operation, the application must do the following:

- Indicate in its bundle plist that it can respond to inquires from Final Cut Pro X.
- Respond to Apple events sent by Final Cut Pro X.

The most important requirement is being able to respond to Apple events sent from Final Cut Pro X. Consult the *Cocoa Scripting Guide* for details of how an application implements such functionality in general. In particular, the following are required:

- Define the scripting terminology including the events, the object classes, and the record types.
- Implement command-handler classes that are referenced from the scripting terminology definitions.
- Implement the object classes and associated properties according to key-value coding (KVC) and other required conventions documented in the *Cocoa Scripting Guide*.

> **Note:** Typically, you do not specifically need to handle *Get Property* Apple events in your code by virtue of the Cocoa scripting support classes and KVC. And because Cocoa handles the *Open Document* Apple event, the application does not need a command-handler class specifically for that event. Some customization may be needed to ensure the files handed through the *Open Document* Apple event are properly associated to an asset object. See Notes on Implementation (page 20) for more details.

Refer to Appendix A: ProVideo Asset Management Suite (page 22) for more information on the scripting terminology definitions.

Implementation Details

Application implementation involves the following:

- Indicating its capabilities—See Signaling Application Capabilities (page 15).
- Defining its scripting terminology—See Scripting Definition (page 15).
- Interacting with Final Cut Pro X using Apple events—See Interaction Through Apple events (page 17).

Signaling Application Capabilities

Final Cut Pro X needs to know up front whether the application understands the protocol. To indicate this capability, the application must have the following entry in its bundle plist:

```
<key>com.apple.proapps.MediaAssetProtocol</key>
<dict>
</dict>
```

Note: The value is an empty dictionary. The content is reserved for future use.

Scripting Definition

The application needs to have a scripting terminology definition that defines the events, object classes, and record types described in this section. Refer to Preparing a Scripting Definition File in the *Cocoa Scripting Guide*. Also, refer to Appendix A: ProVideo Asset Management Suite (page 22) for the scripting definition Final Cut Pro X uses.

Record Type: asset location

Describes where files associated with an asset are located.

Property	Type	Description
`folder`	file	The containing folder path.
`base name`	text	The base name for the export output files. This name must match the asset name.
`has description`	boolean	Set to `true` if the application expects XML as the exported output.
`has media`	boolean	Set to `true` if the application expects media as the exported output.

Note: Final Cut Pro X currently exports the media even if the `has media` property is *false*. In a future version, Final Cut Pro X is expected to be able to export XML only. It is considered an error if both the `has description` and the `has media` properties are *false*.

Object Class Type: asset

Represents an asset object corresponding to the export output in the application.

Property	Type	Description
`id`	text	The unique asset identifier.
`name`	text	The asset name.
`location info`	asset location	The location of files associated with the asset.
`metadata`	user defined record	The shared metadata set associated with the asset.
`data options`	user defined record	The options set for the asset data.

Event Type: make

Creates a new object, specifically an asset object.

Parameter	Type	Description
`new`	type	The new object's class.
`at`	location specifier	The location in which to insert the new object.
`with data`	any	The initial contents of the object.
`with properties`	record	A list of object properties used to initialize the new object, including the following: • `name` (text)—The asset name. • `metadata` (record)—The metadata set associated with the asset object. • `data options` (record)—The option set used to create the asset data.

> **Note:** Final Cut Pro X does not use the `at` and `with data` parameters; they are inherited from the event definition in the standard suite. They are included here in case the application needs to support these parameters for other clients.

To enable access to the objects and to the event from a sandboxed application, the application must specify the `com.apple.assetmanagement.import` access group for the asset object class and for the respective element in its container, and must also specify the same access group for the *Create Asset* event. See Appendix A: ProVideo Asset Management Suite (page 22) for more details.

Interaction Through Apple events

Once Final Cut Pro X confirms that the application is capable of participating in the interaction, it sends a series of Apple events in the order described below. The application is expected to handle these events and return the appropriate information.

> **Note:** The synopsis for each Apple event is represented in AppleScript syntax.

Refer to Appendix B: AppleScript Example (page 28) for a script example and a sample log of events and responses that illustrate the interaction.

1. Create Asset event

```
make new asset with properties { name: <asset name>, metadata: <metadata
 record>, data options: <options record> }
```

Final Cut Pro X sends a *Create Asset* event when the user initiates the share operation. The application is expected to create a placeholder asset object that represents the output of the share operation and to return an object specifier that Final Cut Pro X uses to reference the asset object in subsequent Apple events.

While handling this event the application has an opportunity to bring up its UI, before possibly a time-consuming transcode operation starts. This UI can be used to let the user specify information relevant to the operation the application is about to perform on the export output. This information may include custom metadata to be stored along with the media or data that determines the location of the output.

> **Note:** Generally, user interaction is not encouraged while an application is handling incoming Apple events, especially when this is happening as part of an automated workflow. If the application anticipates the same event is used in an automated workflow, you can check the user interaction level of the Apple event sent. See Notes on Implementation (page 20) for more information.

The event parameters include the following asset object properties:

Property	Description
`name`	The asset name. Final Cut Pro X derives this from the project name.
`metadata`	A record representing the set of share metadata, where the keys are the reverse DNS style metadata keys and the values are the associated metadata values.

Property	Description
`data options`	A record containing the export output options. This record holds the information about the available metadata views from which the application can choose. **Key:** availableMetadataSets **Type:** list **Description:** A list of available metadata views in Final Cut Pro X. The application should indicate its choice in response to another Apple event, sent later, that asks for the value of the data options property.

The return value for this Apple event is an object specifier for the new asset object created.

2. Get Location Info Property event

```
get location info of <asset object specifier>
```

Once a new asset object is created in the application, Final Cut Pro X sends a *Get Location Info Property* event asking for the location of the export output files. The application is expected to return an **asset location** record as described in Scripting Definition (page 15). In particular, the `folder` property of the record specifies the folder location where the export output files are placed. The `base name` property is used as the base name of the export output files. The media file has the `.mov` extension, and the XML file has the `.fcpxml` extension. If the application expects the XML file and the media file, be sure to set the `has description` property to *true* in the asset location record.

3. Get Metadata Property event

```
get metadata of <asset object specifier>
```

Final Cut Pro X sends a *Get Metadata Property* event asking for an updated set of metadata. Final Cut Pro X then updates the project and includes the updated metadata into the export output. Final Cut Pro X previously supplied the original set of metadata through the *Create New Asset* event. The application is expected to return a user record that has metadata keys and their values.

> **Note:** Final Cut Pro X incorporates updates only on Share Metadata—see Appendix C: Share Metadata (page 30) for the list of Share Metadata keys.

In particular, the metadata key `com.apple.proapps.share.id` represents the unique ID of the media asset. The application can generate a unique ID for its own purposes and include that ID in the updated metadata set. Final Cut Pro X includes this ID in the export output so that the application can use it to track the files associated to a particular asset.

4. Get Data Options Property event

```
get data options of <asset object specifier>
```

Final Cut Pro X sends a *Get Data Options Property* event requesting the data options to be used in generating the export output. The application is expected to return a user record with the following key and its associated value:

Key	Type	Description
`metadataSet`	text	The name of the metadata view containing the metadata items to be included in the project XML.

5. Open Documents event

```
open <list of export output files>
```

Once the export operation has completed, Final Cut Pro X sends the path to the exported output files through the *Open Document* event. The expected return value is the object specifier for the asset object. The application is expected to associate the files to the asset it has created, using the folder and the base name of the files to match against the location information of the asset.

> **Note:** In certain cases, Final Cut Pro X creates multiple media files, for example, when the user requests to export each role as a separate file. In these cases, the export file name starts with the asset name returned as the base name property of the asset location record, followed by " - ", as a delimiter, and a suffix that indicates the role.

Notes on Implementation

Asset Object

A media asset may correspond to a document in some applications, while in others it may just be an element of a larger entity (such as a media asset database) which itself may be a document. The protocol between Final Cut Pro X and the application through Apple events is intended to be agnostic to the object containment hierarchy in the application, as far as the application returns an object specifier with which Final Cut Pro X can access a specific asset object.

The scripting definition (see Appendix A: ProVideo Asset Management Suite (page 22)) defines an asset as a document element, but it does not have to be that way.

When an asset object is not a document, handling an *Open Document* Apple event to associate files to an asset might be a bit awkward. Since Cocoa handles the *Open Document* Apple event, you would need a custom document controller to override the behavior of Cocoa when the application opens a file, and associate a file with either an existing asset object or a new asset object if necessary. The association should be done based on the URL of the asset.

Scripting support class extensions

Applications using Cocoa Scripting Support to handle Apple events sent by Final Cut Pro X must support conversions between the Apple event descriptor types used by Final Cut Pro X and the respective Foundation classes as Objective-C categories (as listed below), since they are not currently implemented by Cocoa.

- Apple event record descriptor (*typeAERecord*) and *NSDictionary*

```
@interface NSDictionary (UserDefinedRecord)

+(NSDictionary*)scriptingUserDefinedRecordWithDescriptor:(NSAppleEventDescriptor*)desc;
  -(NSAppleEventDescriptor*)scriptingUserDefinedRecordDescriptor;
@end
```

- Apple event list descriptor (*typeAEList*) and *NSArray*

```
@interface NSArray (UserList)
  +(NSArray*)scriptingUserListWithDescriptor:(NSAppleEventDescriptor*)desc;
  -(NSAppleEventDescriptor*)scriptingUserListDescriptor;
@end
```

- Apple event generic descriptor (list, record, etc.) and *id*

```
@interface NSAppleEventDescriptor (GenericObject)
  +(NSAppleEventDescriptor*)descriptorWithObject:(id)object;
  -(id)objectValue;
@end
```

- Apple event file URL descriptor (*typeFileURL*) and *NSURL* (optional)

```
@interface NSAppleEventDescriptor (URLValue)
  +(NSAppleEventDescriptor*)descriptorWithURL:(NSURL*)url;
  -(NSURL*)urlValue;
@end
```

Refer to Appendix D: Scripting Support Categories (page 32) for sample implementations of these categories.

> **Note:** The `NSAppleEventDescriptor (URLValue)` category is not strictly necessary; however, it is convenient and ensures proper packaging of file URLs, and is therefore highly recommended.

Apple event interaction level

To avoid initiating user interactions while handling Apple events sent by an automated workflow, make sure your application handling the events checks the user interaction level of events. You can do this within your scripting command handler by getting the current Apple event and its `keyInteractLevelAttr` attribute.

The value of the `keyInteractLevelAttr` attribute can be one of the following:

```
kAENeverInteract    = 0x00000010, /* server should not interact with user */
kAECanInteract      = 0x00000020, /* server may try to interact with user */
kAEAlwaysInteract   = 0x00000030, /* server should always interact with user where
 appropriate */
kAECanSwitchLayer   = 0x00000040, /* interaction may switch layer */
```

Consult *NSAppleEventManager Class Reference* for details on getting the current Apple event.

Appendix A: ProVideo Asset Management Suite

An application that interacts with Final Cut Pro X to control the export process must support the events and object classes described in Scripting Definition (page 15). A complete scripting definition is presented below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dictionary SYSTEM "file://localhost/System/Library/DTDs/sdef.dtd">

<!--

   File: AssetManagementSuite.sdef
 Abstract: Asset Management suite scripting terminology definitions
     for the SimpleAssetManager sample code.
  Version: 1.4

 Disclaimer: IMPORTANT:  This Apple software is supplied to you by Apple
 Inc. ("Apple") in consideration of your agreement to the following
 terms, and your use, installation, modification or redistribution of
 this Apple software constitutes acceptance of these terms.  If you do
 not agree with these terms, please do not use, install, modify or
 redistribute this Apple software.

 In consideration of your agreement to abide by the following terms, and
 subject to these terms, Apple grants you a personal, non-exclusive
 license, under Apple's copyrights in this original Apple software (the
 "Apple Software"), to use, reproduce, modify and redistribute the Apple
 Software, with or without modifications, in source and/or binary forms;
 provided that if you redistribute the Apple Software in its entirety and
 without modifications, you must retain this notice and the following
 text and disclaimers in all such redistributions of the Apple Software.
 Neither the name, trademarks, service marks or logos of Apple Inc. may
 be used to endorse or promote products derived from the Apple Software
```

```
 without specific prior written permission from Apple.  Except as
 expressly stated in this notice, no other rights or licenses, express or
 implied, are granted by Apple herein, including but not limited to any
 patent rights that may be infringed by your derivative works or by other
 works in which the Apple Software may be incorporated.

 The Apple Software is provided by Apple on an "AS IS" basis.  APPLE
 MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION
 THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY AND FITNESS
 FOR A PARTICULAR PURPOSE, REGARDING THE APPLE SOFTWARE OR ITS USE AND
 OPERATION ALONE OR IN COMBINATION WITH YOUR PRODUCTS.

 IN NO EVENT SHALL APPLE BE LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL
 OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 INTERRUPTION) ARISING IN ANY WAY OUT OF THE USE, REPRODUCTION,
 MODIFICATION AND/OR DISTRIBUTION OF THE APPLE SOFTWARE, HOWEVER CAUSED
 AND WHETHER UNDER THEORY OF CONTRACT, TORT (INCLUDING NEGLIGENCE),
 STRICT LIABILITY OR OTHERWISE, EVEN IF APPLE HAS BEEN ADVISED OF THE
 POSSIBILITY OF SUCH DAMAGE.

 Copyright (C) 2011-2013 Apple Inc. All Rights Reserved.


-->



<!-- declare the namespace for using XInclude so we can include the standard suite
 -->
<dictionary xmlns:xi="http://www.w3.org/2003/XInclude">


  <!-- use XInclude to include the standard suite -->
 <xi:include href="file:///System/Library/ScriptingDefinitions/CocoaStandard.sdef"
 xpointer="xpointer(/dictionary/suite)"/>


  <!-- specific suite(s) for the application follow... -->
```

```
  <suite name="ProVideo Asset Management" code="pvam" description="Scripting
terminology for Pro Video Asset Management applications.">


    <record-type name="asset location" code="aslc">
      <access-group identifier="com.apple.proapps.assetmanagement.import"
access="rw"/>
      <property name="folder" code="asfd" type="file" description="Directory in
which the asset files exist.">
        <cocoa key="folder"/>
      </property>
      <property name="base name" code="asbn" type="text" description="Base file
name of the asset files.">
        <cocoa key="basename"/>
      </property>
      <property name="has media" code="ashm" type="boolean" description="Whether
the asset has rendered media">
        <cocoa key="hasMedia"/>
      </property>
      <property name="has description" code="ashd" type="boolean"
description="Whether the asset has an XML sescription">
        <cocoa key="hasDescription"/>
      </property>
    </record-type>


    <value-type name="user defined record" code="usrf">
      <cocoa class="NSDictionary"/>
    </value-type>


    <command name="open" code="aevtodoc" description="Open a document.">
      <access-group identifier="com.apple.proapps.assetmanagement.import"
access="rw"/>
      <direct-parameter description="The file(s) to be opened.">
        <type type="file"/>
        <type type="file" list="yes"/>
      </direct-parameter>
      <result description="The opened document(s).">
        <type type="document"/>
```

```
        <type type="document" list="yes"/>
      </result>
    </command>


    <command name="make" code="corecrel" description="Create a new object.">
      <access-group identifier="com.apple.proapps.assetmanagement.import"
access="rw"/>
      <cocoa class="SAMMakeCommand"/>
      <parameter name="new" code="kocl" type="type" description="The class of the
 new object.">
        <cocoa key="ObjectClass"/>
      </parameter>
      <parameter name="at" code="insh" type="location specifier" optional="yes"
description="The location at which to insert the object.">
        <cocoa key="Location"/>
      </parameter>
      <parameter name="with data" code="data" type="any" optional="yes"
description="The initial contents of the object.">
        <cocoa key="ObjectData"/>
      </parameter>
      <parameter name="with properties" code="prdt" type="record" optional="yes"
description="The initial values for properties of the object.">
        <cocoa key="KeyDictionary"/>
      </parameter>
      <result type="specifier" description="The new object."/>
    </command>


    <class-extension name="application" extends="application">
      <access-group identifier="com.apple.proapps.assetmanagement.import"
access="rw"/>
      <cocoa class="NSApplication"/>
      <property name="name" code="pnam" type="text" access="r" description="The
name of the application.">
        <access-group identifier="com.apple.proapps.assetmanagement.import"
access="rw"/>
      </property>
      <element type="document">
```

```
        <access-group identifier="com.apple.proapps.assetmanagement.import"
access="rw"/>
      </element>
    </class-extension>

   <class name="document" code="docu" description="A document." inherits="document">
      <access-group identifier="com.apple.proapps.assetmanagement.import"
access="rw"/>
      <cocoa class="SAMDocument"/>
      <property name="name" code="pnam" type="text" access="r" description="The
name of the application.">
        <access-group identifier="com.apple.assetmanagement.import" access="rw"/>
      </property>
      <property name="id" code="ID  " type="text" access="r" description="The
unique identifier of the asset">
        <access-group identifier="com.apple.assetmanagement.import" access="rw"/>
      </property>
      <element type="asset">
        <access-group identifier="com.apple.assetmanagement.import" access="rw"/>
        <cocoa key="assets"/>
      </element>
    </class>

    <class name="asset" code="aset" description="A media asset.">
      <access-group identifier="com.apple.assetmanagement.import" access="rw"/>
      <cocoa class="SAMAsset"/>
      <property name="id" code="ID  " type="text" access="r" description="The
unique identifier of the asset">
        <access-group identifier="com.apple.assetmanagement.import" access="rw"/>
        <cocoa key="uniqueID"/>
      </property>
      <property name="name" code="pnam" type="text" access="r" description="Its
name.">
        <access-group identifier="com.apple.assetmanagement.import" access="rw"/>
        <cocoa key="name"/>
      </property>
```

```
      <property name="location info" code="locn" type="asset location" access="r"
 description="Location information of the asset.">
        <access-group identifier="com.apple.assetmanagement.import" access="rw"/>
        <cocoa key="locationInfo"/>
      </property>
     <property name="metadata" code="meta" type="user defined record" access="rw"
 description="Metadata associated to the asset.">
        <access-group identifier="com.apple.assetmanagement.import" access="rw"/>
        <cocoa key="metadata"/>
      </property>
      <property name="data options" code="dopt" type="user defined record"
access="rw" description="Data creation options for the asset.">
        <access-group identifier="com.apple.assetmanagement.import" access="rw"/>
        <cocoa key="dataOptions"/>
      </property>
    </class>
  </suite>
</dictionary>
```

Appendix B: AppleScript Example

This appendix presents a simple AppleScript fragment and the resulting event trace that illustrates the interaction between Final Cut Pro X and another application through Apple events.

AppleScript fragment:

```
tell application "SimpleAssetManager"
    make new asset with properties {name:"MyNewAsset",
metadata:{|com.apple.proapps.share.episodeID|:"MyNewEpisode"}, data
options:{|availableMetadataSets|:{"Camera View", "General View"}}}
    set newAsset to result
    set theLocation to location info of newAsset
    set theMetadata to metadata of newAsset
    set theDataOptions to data options of newAsset
    theLocation
end tell
```

Apple event trace:

The following Apple event trace outlines events and replies when the AppleScript fragment above runs:

```
tell application "SimpleAssetManager"
    make new asset with properties {name:"MyNewAsset",
metadata:{|com.apple.proapps.share.episodeid|:"MyNewEpisode"}, data
options:{availableMetadataSets:{"Camera View", "General View"}}}
        --> asset id "SSO-1059-2" of document "Untitled"
    get location info of asset id "SSO-1059-2" of document "Untitled"
        --> {has media:true, has description:true, base name:"MyNewAsset",
folder:file “MacOS:Users:MacUser:Movies:"}
    get metadata of asset id "SSO-1059-2" of document "Untitled"
        --> {|com.apple.simpleassetmanager.managedasset|:"1",
|com.apple.quicktime.description|:"New asset for upcoming episode.",
|com.apple.simpleassetmanager.prepareasset|:"1",
|com.apple.simpleassetmanager.expeirationdate|:"2013-08-28 22:47:10 +0000",
|com.apple.proapps.share.episodeid|:"MyNewEpisode",
```

```
|com.apple.proapps.share.id|:"SAMS24598",
|com.apple.proapps.share.episodenumber|:"598"}

    get data options of asset id "SSO-1059-2" of document "Untitled"

        --> {metadataSet:"General View"}

end tell


Result:

{has media:true, has description:true, base name:"MyNewAsset", folder:file
“MacOS:Users:MacUser:Movies:"}
```

Appendix C: Share Metadata

This appendix lists the share metadata keys that Final Cut Pro X passes to and expects from the application.

Display Name	Key	Type
Actors	com.apple.quicktime.artist	String
Category	com.apple.proapps.share.category	String
Copyright	com.apple.quicktime.copyright	String
Creator	com.apple.quicktime.author	String
Date	com.apple.quicktime.creationdate	String
Description	com.apple.quicktime.description	String
Directors	com.apple.quicktime.director	String
Episode ID	com.apple.proapps.share.episodeID	String
Episode Number	com.apple.proapps.share.episodeNumber	String
Genre	com.apple.quicktime.genre	String
Media Kind	com.apple.proapps.share.mediaKind	String
Producers	com.apple.quicktime.producer	String
Screenwriters	com.apple.proapps.share.screenWriter	String
Season Number	com.apple.proapps.share.seasonNumber	String
Share ID	com.apple.proapps.share.id	String
Show	com.apple.quicktime.album	String
Tags	com.apple.quicktime.keywords	String
Title	com.apple.quicktime.title	String
TV Network	com.apple.proapps.share.tvNetwork	String

Display Name	Key	Type
US Rating	com.apple.quicktime.rating.user	String

Appendix D: Scripting Support Categories

This appendix lists a sample implementation of the categories discussed in Scripting support class extensions (page 20). For more information on the `NSAppleEventDescriptor` class, consult *NSAppleEventDescriptor Class Reference*.

```
/*
 File: ScriptingSupportCategories.m
 Abstract: Cocoa Scripting Support Categories implementations
 Version: 1.3

 Disclaimer: IMPORTANT:  This Apple software is supplied to you by Apple
 Inc. ("Apple") in consideration of your agreement to the following
 terms, and your use, installation, modification or redistribution of
 this Apple software constitutes acceptance of these terms.  If you do
 not agree with these terms, please do not use, install, modify or
 redistribute this Apple software.

 In consideration of your agreement to abide by the following terms, and
 subject to these terms, Apple grants you a personal, non-exclusive
 license, under Apple's copyrights in this original Apple software (the
 "Apple Software"), to use, reproduce, modify and redistribute the Apple
 Software, with or without modifications, in source and/or binary forms;
 provided that if you redistribute the Apple Software in its entirety and
 without modifications, you must retain this notice and the following
 text and disclaimers in all such redistributions of the Apple Software.
 Neither the name, trademarks, service marks or logos of Apple Inc. may
 be used to endorse or promote products derived from the Apple Software
 without specific prior written permission from Apple.  Except as
 expressly stated in this notice, no other rights or licenses, express or
 implied, are granted by Apple herein, including but not limited to any
 patent rights that may be infringed by your derivative works or by other
```

```
 works in which the Apple Software may be incorporated.

 The Apple Software is provided by Apple on an "AS IS" basis.  APPLE
 MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION
 THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY AND FITNESS
 FOR A PARTICULAR PURPOSE, REGARDING THE APPLE SOFTWARE OR ITS USE AND
 OPERATION ALONE OR IN COMBINATION WITH YOUR PRODUCTS.

 IN NO EVENT SHALL APPLE BE LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL
 OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 INTERRUPTION) ARISING IN ANY WAY OUT OF THE USE, REPRODUCTION,
 MODIFICATION AND/OR DISTRIBUTION OF THE APPLE SOFTWARE, HOWEVER CAUSED
 AND WHETHER UNDER THEORY OF CONTRACT, TORT (INCLUDING NEGLIGENCE),
 STRICT LIABILITY OR OTHERWISE, EVEN IF APPLE HAS BEEN ADVISED OF THE
 POSSIBILITY OF SUCH DAMAGE.

 Copyright (C) 2013 Apple Inc. All Rights Reserved.

 */

#import "ScriptingSupportCategories.h"

#import <Carbon/Carbon.h>



@implementation NSDictionary (UserDefinedRecord)

+(NSDictionary*)scriptingUserDefinedRecordWithDescriptor:(NSAppleEventDescriptor*)desc
{
    NSMutableDictionary* dict = [NSMutableDictionary dictionaryWithCapacity:0];
    NSAppleEventDescriptor* userFieldItems = [desc
descriptorForKeyword:keyASUserRecordFields];
    NSInteger numItems = [userFieldItems numberOfItems];
```

```
    for ( NSInteger itemIndex = 1; itemIndex <= numItems - 1; itemIndex += 2 ) {
        NSAppleEventDescriptor* keyDesc = [userFieldItems
descriptorAtIndex:itemIndex];
        NSAppleEventDescriptor* valueDesc = [userFieldItems
descriptorAtIndex:itemIndex + 1];
        NSString* keyString = [keyDesc stringValue];
        id value = [valueDesc objectValue];

        if ( keyString != nil && value != nil )
            [dict setObject:value forKey:keyString];
    }

    return [NSDictionary dictionaryWithDictionary:dict];
}

-(NSAppleEventDescriptor*)scriptingUserDefinedRecordDescriptor
{
   NSAppleEventDescriptor* recordDesc = [NSAppleEventDescriptor recordDescriptor];
   NSAppleEventDescriptor* userFieldDesc = [NSAppleEventDescriptor listDescriptor];
    NSInteger userFieldIndex = 1;

    for ( id key in self ) {
        if ( [key isKindOfClass:[NSString class]] ) {
            NSString* valueString = nil;
            id value = [self objectForKey:key];

            if ( ! [value isKindOfClass:[NSString class]] )
                valueString = [NSString stringWithFormat:@"%@", value];
            else
                valueString = value;

            NSAppleEventDescriptor* valueDesc = [NSAppleEventDescriptor
descriptorWithString:valueString];
            NSAppleEventDescriptor* keyDesc = [NSAppleEventDescriptor
descriptorWithString:key];
```

```
            if ( valueDesc != nil && keyDesc != nil ) {
                [userFieldDesc insertDescriptor:keyDesc atIndex:userFieldIndex++];
              [userFieldDesc insertDescriptor:valueDesc atIndex:userFieldIndex++];
            }
        }
    }

    [recordDesc setDescriptor:userFieldDesc forKeyword:keyASUserRecordFields];

    return recordDesc;
}


@end

@implementation NSArray (UserList)

+(NSArray*)scriptingUserListWithDescriptor:(NSAppleEventDescriptor*)desc
{
    NSMutableArray* array = [NSMutableArray arrayWithCapacity:0];
    NSInteger numItems = [desc numberOfItems];

    for ( NSInteger itemIndex = 1; itemIndex <= numItems; itemIndex++ ) {
        NSAppleEventDescriptor* itemDesc = [desc descriptorAtIndex:itemIndex];

        [array addObject:[itemDesc objectValue]];
    }

    return [NSArray arrayWithArray:array];
}

-(NSAppleEventDescriptor*)scriptingUserListDescriptor
{
    NSAppleEventDescriptor* listDesc = [NSAppleEventDescriptor listDescriptor];
    NSInteger itemIndex = 1;
```

```
    for ( id item in self ) {
        NSAppleEventDescriptor* itemDesc = [NSAppleEventDescriptor
descriptorWithObject:item];

        [listDesc insertDescriptor:itemDesc atIndex:itemIndex++];
    }

    return listDesc;
}

@end



@implementation NSAppleEventDescriptor (GenericObject)

+(NSAppleEventDescriptor*)descriptorWithObject:(id)object
{
    NSAppleEventDescriptor* desc = nil;

    if ( [object isKindOfClass:[NSArray class]] ) {
        NSArray*    array = (NSArray*)object;

        desc = [array scriptingUserListDescriptor];
    }
    else if ( [object isKindOfClass:[NSDictionary class]] ) {
        NSDictionary*   dict = (NSDictionary*)object;

        desc = [dict scriptingUserDefinedRecordDescriptor];
    }
    else if ( [object isKindOfClass:[NSString class]] ) {
        desc = [NSAppleEventDescriptor descriptorWithString:(NSString*)object];
    }
    else if ( [object isKindOfClass:[NSURL class]] ) {
        desc = [NSAppleEventDescriptor descriptorWithURL:(NSURL*)object];
```

```
    }
    else {
        NSString* valueString = [NSString stringWithFormat:@"%@", object];

        desc = [NSAppleEventDescriptor descriptorWithString:valueString];
    }

    return desc;
}

-(id)objectValue
{
    DescType    descType = [self descriptorType];
    DescType    bigEndianDescType = 0;
    id          object = nil;


    switch ( descType ) {
        case typeUnicodeText:
        case typeUTF8Text:
            object = [self stringValue];
            break;
        case typeFileURL:
            object = [self urlValue];
            break;
        case typeAEList:
            object = [NSArray scriptingUserListWithDescriptor:self];
            break;
        case typeAERecord:
           object = [NSDictionary scriptingUserDefinedRecordWithDescriptor:self];
            break;
        case typeSInt16:
        case typeUInt16:
        case typeSInt32:
        case typeUInt32:
```

```
        case typeSInt64:
        case typeUInt64:
            object = [NSNumber numberWithInteger:(NSInteger)[self int32Value]];
            break;
        default:
            bigEndianDescType = EndianU32_NtoB(descType);
            NSLog(@"Creating NSData for AE desc type %.4s.",
(char*)&bigEndianDescType);
            object = [self data];
            break;
    }

    return object;
}

@end



@implementation NSAppleEventDescriptor (URLValue)

+ (NSAppleEventDescriptor *)descriptorWithURL:(NSURL *)url
{
    NSData*     urlData = (NSData *)CFBridgingRelease((CFURLCreateData(NULL,
(__bridge CFURLRef)(url), kCFStringEncodingUTF8, TRUE)));
    return [NSAppleEventDescriptor descriptorWithDescriptorType:typeFileURL
data:urlData];
}

- (NSURL*)urlValue
{
    NSData      *urlData = [self data];
    NSError     *theError = nil;
    NSURL       *theURLValue = nil;

    switch ( [self descriptorType] ) {
```

```
        case typeFileURL:
            theURLValue = (NSURL *)CFBridgingRelease(CFURLCreateWithBytes(NULL,
[urlData bytes], [urlData length], kCFStringEncodingUTF8, NULL));
            break;
    }

    if ( theError != nil ) {
        NSLog(@"Failed to retrieve URL value out of an NSAppleEventDescriptor %@,
 error %@.", self, theError);
    }

    return theURLValue;
}

@end
```

Document Revision History

This table describes the changes to *Final Cut Pro X Workflows Developer Guide* .

Date	Notes
2015-04-13	Made minor updates.
2013-12-19	New document that describes how Final Cut Pro X exchanges data with other applications.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010