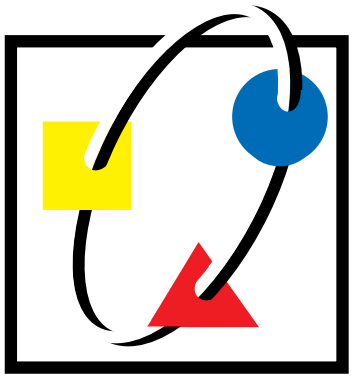


OPEN MEDIA FRAMEWORK®

OMF INTERCHANGE®

Specification

Version 2.1

OMF Developers' Desk
Avid Technology, Inc.
1925 Andover St.
Tewksbury, MA 01876
Phone: 800-949-OMFI
International: 978-640-3400
FAX: 978-640-0065 directed to OMF Developers' Desk
Internet URL: http://www.omfi.org
September 18, 1997

OMF Interchange Specification• Part 0130-00254-01 Rev. B• 9/97

Table of Contents

Preface

About this Document

The *OMF Interchange Specification* is written for two audiences. It is for system programmers who need to know the details of the file format for the purpose of reading or writing interchange files. It is also for system architects, system analysts, and others who want to understand the goals and overall semantics of OMF Interchange.

Version Information

This document describes OMF Interchange Version 2.1. Appendix D contains a summary of the changed features and new features in Version 2.1.

How to Use this Document

There are three major parts to this book. Depending on your purpose, you might want to read some sections and skip others.

- The first part provides an overview and an introduction to concepts.
  - — Chapter 1 gives an overview of the OMF Interchange goals, application domains, and file structures.
  - — Chapter 2 describes the basic media concepts that are used in OMF.
- The second part describes the OMF object model in detail and uses examples to show how to describe media with OMF. This part should be read in conjunction with the last part.
  - — Chapter 3 describes the OMF class model and class hierarchy.
  - — Chapter 4 describes Mobs and the Header object.
  - — Chapter 5 describes how to use OMF compositions.
  - — Chapter 6 describes how to use OMF to describe media.

- The third part contains the reference appendixes for OMF.
  - — Appendix A provides a comprehensive description of each OMF class.
  - — Appendix B describes the OMF data types and data kinds.
  - — Appendix C lists related reference documents and describes the formats used to store digital media data in files.
  - — Appendix D describes what's new in Version 2.0.
  - — Appendix E lists the OMF effects.
  - — Appendix F shows the OMF class hierarchy illustration.

At the end of the book, there are the following sections:

- Glossary
- Index
- OMF Participation Form

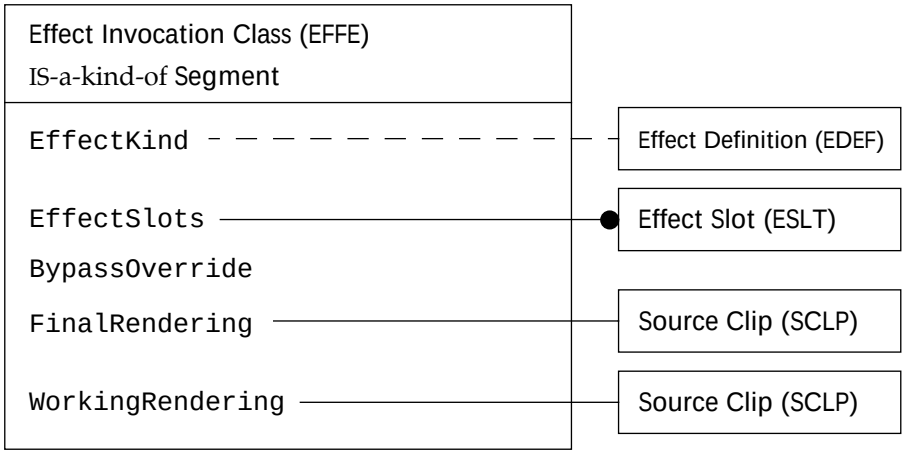
Documentation Conventions

This document follows these conventions:

- Class names are always capitalized and appear in Helvetica typeface. A class name can refer to the class or an object that belongs to the class.
- Class property names and property values appear in `Courier` typeface.

The class reference descriptions in Appendix A use a diagram to describe the data model. The following is an example data model diagram:

Data Model



The top portion of the large box in the Data Model identifies

- The class name, which is Effect Invocation in the example.
- The four-character Class ID, which is EFFE in the example.
- The parent class, which is Segment in the example.
- Optionally, the text *Abstract Class* indicates that the class is used to show what is shared among a set of subclasses; this is not illustrated in this example.

The bottom portion of the large box lists the properties that are defined for the particular class, but it does not include the properties that are inherited from a superclass. For properties with values that are specified directly, without an object, you will see only the property name listed in the large box, such as BypassOverride.

For properties with values that are specified by objects, the Data Model shows the kinds of objects that specify the value and the relationship between the objects and the properties. The properties appear in the large box, and the class names indicating the kinds of objects appear in the small boxes to the right. In the sample Data Model, an Effect Definition object specifies the value of the `EffectKind` property. The relationship between property and object is indicated by the type of line connecting them:

- A solid line *without* a solid circle indicates that the property has a single object as its value, as shown with the `FinalRendering` and `Working-Rendering` properties.
- A solid line *with* a solid circle indicates that the property has a set of objects as its value, as shown with the `EffectSlots` property. If the set is an ordered set, this is indicated by the word *ordered* above the solid line.
- A dashed line indicates that the property has a reference to the object, as shown by the `EffectKind` property and the Effect Definition object. This shows that more than one object can have a reference to a single Effect Definition object.

Technical changes from the *OMF Interchange Specification* Version 2.0 are marked with change bars.

For More Information

This section tells where you can get more information about OMF Interchange and OMF software.

There is a form at the end of this document that describes how you can register as an OMF Interchange Sponsor, Partner, or Champion.

To request a unique application or organization identifier, to order copies of this document, to order the OMF Interchange Toolkit software, or for more

information about OMF Interchange and the OMF Interchange Toolkit software, contact:

OMF Developers' Desk
Avid Technology, Inc.
1925 Andover St.
Tewksbury, MA 01876

Phone: 800-949-OMFI
International: 978-640-3400
FAX: 978-640-0065 Attention: OMF Developers' Desk

Email: `omf-request@omfi.org`
World Wide Web URL: `http://www.omfi.org`

For more information about the TIFF format for graphic images, the Bento container format and Bento API, and other file formats, see Appendix C, which lists standards documents and format specifications.

1 Overview

The Open Media Framework (OMF) Interchange format is a standard format for the interchange of digital media data among heterogeneous platforms. The format encapsulates all the information required to transport a variety of digital media such as audio, video, graphics, and still images, as well as the rules for combining and presenting the media. The format includes rules for identifying the original sources of the digital media data, and it can encapsulate both compressed and uncompressed digital media data.

This overview describes the goals and history of OMF, defines key terms, describes the overall file structure, and lists the required formats for the exchange of digital media data among OMF Interchange applications.

Goals of OMF Interchange

OMF Interchange is the product of many years of experience with the management and manipulation of digital media. Throughout the development of OMF Interchange, certain design goals were paramount:

- Portability and platform independence.

  Digital media processing systems exist on a multitude of platforms, each with different characteristics for storage capacity, throughput, multimedia hardware, and overall system architecture. OMF Interchange eases problems that arise due to particular platform characteristics. For example, OMF Interchange uses a numeric representation that can easily be converted to a platform's native format.
- Encoding of compositions and sources as well as digital media data.

OMF Interchange provides structures for three distinct elements: digital media data, media sources, and compositions:

— Digital media data such as audio, video, and graphics, is only part of the information that constitutes a media presentation.

— Sources describe the digital media data and the original, physical sources of the data.

— Compositions describe the arrangement of sections of sources and how they are played over time.

You can think of a composition as a "recipe" and digital media data as "ingredients." The source information identifies each ingredient and tells where the ingredient came from.

OMF Interchange keeps sources separate from compositions for several reasons. More than one composition can reference sections of the same digital media data, reducing storage requirements. An editor of a composition can choose to include more or less of the digital media data in a section. Storing the digital media data separately allows an editor the freedom to modify the selection at any time in the editing process. Because the OMFI file identifies the source of the digital media data, an editor can re-create the digital media data to use a different sampling frequency or compression. OMF Interchange provides a comprehensive set of formats for describing the original (often analog) source material, even when that material is outside the computer system.

- Program encapsulation.

  OMF Interchange provides for a variety of existing digital media types and the ability to easily support new types in the future. A single OMF Interchange file can encapsulate all the information required to create, edit, and play digital media presentations.

- Suitability for playback.

  Digital media for video and audio is typically large and stresses the capabilities of most of today's systems. While OMF Interchange is designed primarily for data interchange, it is structured to facilitate playback directly from an interchanged file when being used on platforms with characteristics and hardware similar to those of the source platform, without the need for expensive translation or duplication of the sample data.

- Application independence.

  Different applications will add value to OMF Interchange structures based on the application's purpose. OMF Interchange is specifically designed to permit different applications to create, edit, enhance, modify, play back, and then transmit the same compositions and media in object form without loss of information.

- Direct access to internal objects.

  OMF Interchange files may be large and contain many different kinds of media data structures, which are called "objects." Applications can determine the contents of an OMF Interchange file and extract objects of interest such as compositions and media source information, without reading the entire file.

- Encapsulation and specialization.

The classification of data objects in OMF Interchange, with the refinement or amplification of information through specialized properties, allows applications to deal with all known objects in a uniform fashion and bypass the specializations they do not need to interpret.

- External file references.

  The OMF Interchange standard anticipates that sample data or other data might exist in files external to an OMF Interchange file that references it. The external file can be another OMF Interchange file or a file containing "raw" sample data. The standard provides a mechanism for specifying a reference to the external file and, when necessary, identifying the originating system, the path to the file, and hints for locating the file.

- Extensibility.

  OMF Interchange provides for the development and integration of new media and composition types. In its general framework, OMF Interchange allows applications to add extensions without resorting to supplemental files.

- Application-specific data.

  Along with extensibility, OMF Interchange provides applications with a way of embedding annotations, auxiliary data structures, and data particular to the application. OMF Interchange provides a framework that allows other applications to deal with this data, even if they are not able to interpret its meaning.

- Incremental update.

  Because digital media files may be large, OMF Interchange specifically provides for incremental change to OMF Interchange files without requiring a complete recalculation and rewrite of an existing file.

History of OMF Interchange

The development of the Open Media Framework Interchange format is the result of the cooperative efforts of many industry and standards partners and Avid Technology, Inc.

Version 1.0 of the OMF Interchange format was published in November 1993. Version 2.0 was published in August 1996. This document describes Version 2.1.

The OMF Developer's Desk at Avid Technology, Inc., publishes and maintains the specification. In addition, the OMF Developer's Desk provides the OMF Interchange Toolkit, which is a software package that provides an Application Program Interface (API) that applications can use to read and write OMFI files. The purpose of the toolkit is to make it easier for applications to interchange media and compositions.

OMF uses a container format to store and access objects. OMF uses the Bento® software to define and access the container format. The Bento software defines

the container file format and provides an API that the OMF Interchange Toolkit uses to store and access the objects in a file. It is not necessary to have a detailed knowledge of the Bento software to understand the concepts and terminology of OMF Interchange. To find out whom to contact to get more information about the Bento software, see Appendix C. The OMF Interchange Developers' Toolkit Version 2.1, which is a reference implementation of the OMF Interchange Specification, is Bento compatible.

Version 2.1 and 2.0 Goals

The major goals for Version 2.1 and Version 2.0 of the OMF Interchange specification are to increase the capabilities available to the end user and to reduce the roadblocks to interchanging digital media data. These versions of the specification helps achieve these goals by:

- Allowing more kinds of information to be interchanged between applications
- Making it easier for an application to include OMF support, which will increase the number of applications available to the end user

Another goal for these versions is to enhance existing capabilities and to fix any errors that have been found in the previous version.

OMF Interchange Version 2.1 provides the following enhancements:

- Identification information in the Header object that allows you to determine the application that created an OMF file
- Support for large media files (greater than 2 gigabytes) with 64-bit positions, lengths, and frame indexes
- Support for user comments on Mobs and tagged user-defined information
- Improved support for film pulldown conversions
- Minor improvements to media handling and storage

OMF Interchange Version 2.0 provides the following enhancements:

- Interchange of effects
- Easier support of compositions
- Enhanced media support

Appendix D provides a detailed list of the new and changed features in Version 2.1 and 2.0.

Key Media Information

At the most general level, an OMF Interchange file contains the following kinds of media information:

- Compositions
- Sources of media data

This section defines these terms and describes the role of each type of information in an OMF Interchange file.

Compositions

A composition is a description of all the information required to play or re-edit a media presentation. A composition describes the logical organization of a time-based media presentation, and an application can combine and play the contents of the composition, relating the elements over time.

Typically, a composition comprises multiple types of media from a variety of sources, with references to the digital media data. It does not actually contain the digital media data such as the video, audio, graphics, or animation. Instead, it points to sections of source data. When an application plays a composition, it follows the composition's references to the digital data and then plays it.

The simplest type of composition represents how to play all the information from a single stream of media data. In contrast, a complex composition might contain the information required to play four tracks of audio and a video track with a graphic overlay track, for example. The audio tracks might contain volume and fader information, and the video track might be a combination of multiple video sources, including transition effects at the boundaries between sources (such as wipes, dissolves, or fades). The graphic overlay track could specify information needed to position the graphic, choose the keying characteristics, and fade in or out.

Sources of Media Data

An OMF Interchange file identifies sources of data and describes what kind of media data these sources provide. Is it a videotape source or a film source? Is it a file containing video frame data or audio data? What type of data does the source contain and what format is it in? The OMF composition names each source it uses and describes the section of the source's data that it needs.

Information About Previous Generations of Media

The OMF media source descriptions allow an application to trace a piece of media data from the digital data stored in a file to all previous generations of

media that it was derived from. For example, when an application digitizes the audio and video data from a videotape, it creates a source description for each generation of media. During the digitizing process, it creates a source description for each section of media that is digitized, a source description that identifies the videotape source, and, if applicable, source descriptions that identify the film or audio tape that were used to generate the videotape. Source information can identify any type of source, whether it is an analog source such as a videotape, film reel, or audio tape, or whether it is a digital source such as a file containing video data, audio data, animation data, or graphic data.

If the source media data was derived from another source of media, this information about the original source is stored in OMF so that an application can access the original media. For example, a file containing video data can be derived from the video from a videotape. Accessing the original source is required when you need to reconstruct the digital media data, for example to redigitize it at a higher resolution. Accessing source information is also required if you want to generate an edit decision list (EDL) or a cut list for assembling a finished production from the original sources.

The Source's Media Data

Media data is the digital or analog data you play. The data can be in the form of audio samples, video samples, computed effects, animation, and graphics. OMF Interchange files store digital media data. Digital media data is either data that a user created directly in digital form such as a graphic, or data that a user digitized from an analog source such as the video and audio from a videotape. An OMF Interchange file includes information about the format of the data such as the sample rate and compression technique, in addition to the data itself.

Although composition information is usually compact, a set of media data can be extremely large; therefore, OMF Interchange files allow applications to store the digital media data in separate files. Sometimes the pieces of media data used by a composition are spread across many files or disks.

The advantage of storing the digital media data separately from compositions is that any number of compositions can reference a single source without duplicating the storage requirements. Also, compositions can reference sections of a source's data any number of times. The fact that a composition uses only a portion of the digital media data does not reduce the availability of the data in any way, so an editor could reedit the composition to use more or less of the data at any time.

When digital media data is in a separate file, the source information includes hints for locating the file that contains the data. The external file can also contain the source information that identifies and describes the data, along with the media data itself. The information identifying and describing the source is important in case the data becomes separated from an OMF Interchange file that refers to it.

Mobs

OMF Interchange uses objects called Mobs (from "media objects") to uniquely describe compositions and sources. Each Mob contains an identifier, called a MobID. An OMFI file describes a composition by using a Composition Mob. An OMFI file describes a source of media data by using a Source Mob. File Source Mobs are Source Mobs that identify digital media data.

A Composition Mob identifies each digital source that it references by the source's MobID. Using the MobID, there are various ways that an application can locate the digital media data, depending on whether it is in the same file or in an external file.

Similarly, a file Source Mob stores the physical Source Mob MobID of its original source. An application can use the stored MobIDs to follow the information to the original media.

OMF Interchange and Media Applications

OMF Interchange supports a series of classes of applications with increasingly complex requirements for the manipulation of media:

1. Media producers and consumers

   This is the most basic class of application. These applications create digital media such as audio, video, or graphics, to be used by another application. This class includes applications that take digital media information from other applications in order to display it or store it. In general, these applications do not make use of the more expressive features of the composition information, but work with simple sequences of media. Examples of these applications are graphics drawing packages, animation programs, character generators, digitization packages, and display utilities (viewers).

2. Composition producers and consumers

   This is a more complex class of OMF application, which accepts information from media producer and consumer applications and combines the media to create a finished composition. The finished composition can contain complicated sequencing information such as multitrack audio, or it can combine complex transition effects such as crossfades, wipes, and digital video effects (DVEs). It may combine these components in the composition with finished audio information such as pan, volume, and equalization information.

   These applications make use of all of the information used by the media producers and consumers, and, in addition, they make use of the full expressive power of OMF Interchange compositions. However, these applications might not make full use of the media data information. For example, they could specify that a transition effect occur, but might not be able to produce the media necessary to display that effect.

3. Full-service media applications

At the highest level, a full-service application includes aspects of both the previous application types, but it also has the ability to create the finished product, including all media necessary to play the finished composition. These applications may also deal with collections of compositions and may have additional information regarding the compositions or media data.

File Structure and Access

An OMF Interchange file contains objects. An object can be simple such as a Filler object that describes a blank segment of media, or an object can include a complex structure of other objects. A Mob is an object that has a complex structure of other objects. A Mob describes editing information or media by means of the objects it contains and the structure that connects these objects.

Each file has a file Header object that provides efficient access to top-level objects such as mobs and Media Data objects. Figure 1 shows an OMF Interchange file containing a file Header object, Mobs, and Media Data. OMF Interchange defines many common types of objects and a data model for using these objects to represent composition and source information.

Objects, Properties, and Values

Each object consists of a set of properties. Properties contain data and identify the uses of the data in a way that resembles the role of fields in a database record. Each property has a name and a value. A property's name identifies the property, and a property's value has a type that specifies how the value should be treated. The OMF Interchange Specification defines a set of unique property names and a set of standard data types that can be used in interchange files.

The object class property identifies the kind of object. OMF Interchange defines a set of unique four-character class names to identify the objects needed in OMF Interchange files.

Although Bento allows a single property to have multiple types and a value for each type, OMF Interchange allows only a single type and value for each property in an object.

Access to Objects

OMF Interchange is a flexible format that allows applications to freely add objects, properties, and values to the file. And because each file has a file Header object containing file-wide information for locating objects, applications can easily access objects and their properties by using the OMF Interchange Toolkit API. OMF Interchange supports the use of private objects and properties by applications. If an application does not know how to process a

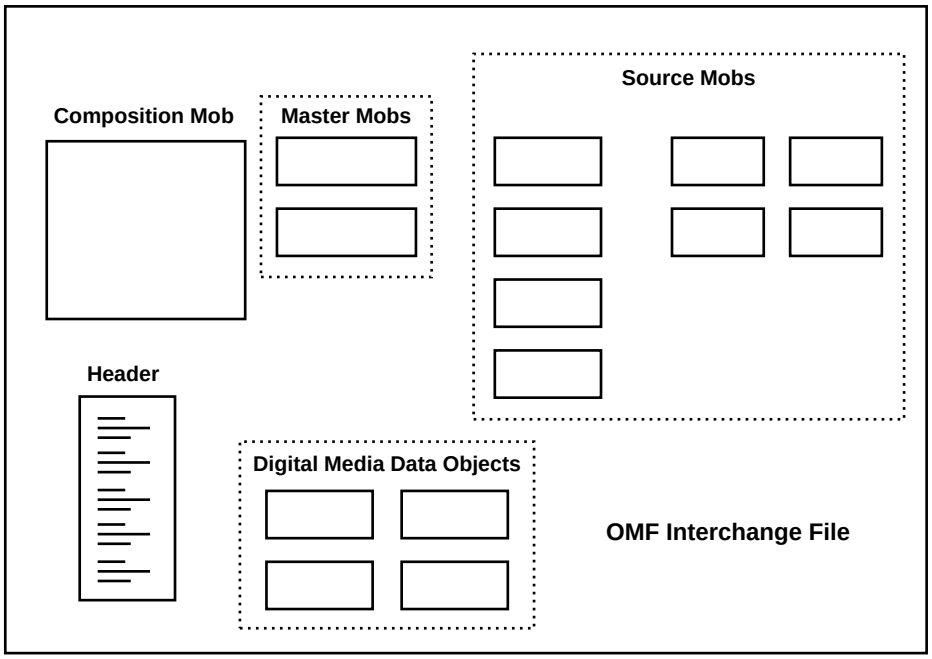


Figure 1: An OMF Interchange file can have any number of objects in any order. The file Header object provides access to objects in the file.

certain value, or chooses not to process it, the application can ignore that value. For example, given a property that represents pan settings for an audio media section, if an application has no way to adjust the pan settings dynamically, it can ignore the value.

The File Header

The file Header object has indexes to the locations of key objects in the file, including an index to the Mobs in the file and an index to the pieces of digital media data in the file. It also has additional properties such as ones that specify the OMF Interchange Specification version and when the file was last modified.

When it is necessary, the file Header contains a list of extensions to the OMF Interchange class hierarchy. OMF Interchange assumes that all applications support the standard class hierarchy; so when an application extends the class hierarchy, the file Header must define the extensions.

The Class Hierarchy and Object Properties

OMF Interchange defines a class hierarchy that allows objects to share properties with other similar objects. The hierarchy defines general classes with more specialized subclasses that all share the general properties. Based on the hierarchy, an object class that is a subclass of another class inherits the properties

its parent class. Chapter 3 provides a more detailed introduction to the OMF class model. Appendix A contains reference information about each object class and its properties.

Object Classes

OMF Interchange defines classes of objects for the elements in mobs, as well as classes of objects for digital media data and other objects that are used outside of mobs.

The OMF Interchange file describes compositions and sources by means of the relationships between mobs, the structure of the objects within the mob, the class of each object included in the mob, and the property values in each object. This object model allows you to create simple OMFI files to describe simple media and compositions but does not prevent you from describing extremely complex media and compositions.

Data Interchange Considerations

Interchanging data across heterogeneous systems may require data conversions due to different storage characteristics, including byte ordering, numeric format, and integer size. OMF objects describe how the data is stored, which enables applications to convert data to their native form when necessary.

The OMF Interchange file can contain objects having values in either of the two byte orders commonly used to store data. If OMFI allowed only one of the byte orders, then it would reduce the efficiency of using OMF on computer architectures that use the other byte order since it would require resource-intensive conversion from one byte order to another. By supporting both byte orders, OMF allows applications to create files efficiently using their native byte order. If another application that uses the same byte order reads the file, it can read it efficiently without converting the byte order. Applications that use a different byte order can still read the file by converting it to the correct byte order.

Digital Media Data Formats

OMF Interchange provides a basic set of required media formats that can be shared among many different products from multiple vendors. At the same time, it provides a structure for incorporating many different kinds of additional media formats into a standardized interchange mechanism. OMF Interchange provides this flexibility by maintaining these three levels of support for media data formats:

- Required interchange formats
- Registered interchange formats

- Private interchange formats

This section describes the purpose and use of each of the media format support categories, and it lists the required formats that OMF Interchange supports.

Required Interchange Formats

Required formats are a basic set of media data formats that are well defined and widely used for representing digital media. These formats are published and easily implemented by a wide range of products without relying on any proprietary hardware or software algorithms, and without bias toward a specific computer platform. Readers and writers for required formats must be either available or easily implementable.

For each required format, a descriptor class and a Media Data class is registered with OMF, and full support exists in the OMF Toolkit.

An application claiming OMF compliance *must* support required formats where applicable. In some cases, an OMF-compliant application may only support some media types such as audio, and not other media types such as video.

Registered Interchange Formats

Registered formats are also recognized by the OMF Developer's Desk as being widely used. However, implementation of registered formats is not required for an application to be considered OMF compliant.

A registered format must either be published so that vendors can implement their own readers and writers, or software must be available (source or object code) to support this format. Registered formats can rely on proprietary hardware or software, and can be specific to a certain computer platform or software product. A descriptor and a media data class identifier will be registered for each registered format. Support for these formats may or may not exist in the OMF Toolkit.

The OMF Developer's Desk provides information about all registered formats. This information includes the descriptor class, the digital data class, contact information for all providers of software and documentation, and the current level of OMF Toolkit support. Optionally, when a format has been widely adopted, the OMF Developer's Desk may directly provide software and documentation.

Private Interchange Formats

OMF Interchange allows media data in private formats so that applications can embed their own nonstandard media formats in an OMF Interchange file. Private formats can be considered internal to an organization or intended for use between two parties with a private agreement. If a format is intended to be

shared between products from different vendors, it probably belongs in the registered category.

Because of the proprietary use and nature of these formats, there should be no conflicts with private formats from other vendors. Reader and writer software is usually considered proprietary and is not available to the general public or provided by the OMF Toolkit. The internal nature of these formats implies that support in applications is obviously not required, and the OMF Developer's Desk need not support them.

OMF Interchange supports multiple coexisting representations of the same media. For example, an OMF Interchange file can represent digital audio in both a private format and in a required format.

The Required Formats

This section describes the required interchange formats included in the OMF Interchange standard. Table 1 summarizes the required formats.

Table 1: OMF Interchange Required Interchange Formats

Types of Media Data	Required Formats
video, graphic, and still image	RGBA Component Image (RGBA) Color Difference Component Image (CDCI) TIFF
animation	Stored as video
audio	Audio Interchange File Format (AIFC) RIFF Waveform Audio File Format (WAVE)

OMF Interchange Version 1.0 specified the TIFF video, graphic, and still format instead of the RGBA Component Image and Color Difference Component Image formats included in the current version. During a transition period, OMF-compliant applications must support the TIFF format to allow compatibility with existing OMF Interchange files and OMF Interchange-compliant applications. After this transition period, OMF-compliant applications do not have to support the TIFF format and should use the RGBA or CDCI required format instead.

Appendix C contains additional information about the digital media data storage formats including references to other documents that describe the TIFF, WAVE, and AIFC formats.

Required Video, Graphic, and Still Image Formats

The RGBA Component Image format (RGBA) and the Color Difference Component Image (CDCI) formats allow for great flexibility in the specification of video, graphic, and image data. These formats also contain the information needed to allow conversion to many other possible formats. The TIFF

format is included in this specification for compatibility with the previous version.

RGBA images are uncompressed component-based images where each pixel is made up of a red, a green, and a blue value. In addition, RGBA images can contain an alpha value. CDCI images are images made up of one luminance component and two color-difference components, commonly known as YC_bC_r, and can optionally be compressed with the Joint Photographic Experts Group (JPEG) algorithm.

The RGBA and CDCI formats allow you to specify the following:

- Dimensions of the image
- Relationship between the dimensions of the stored image and those of the analog media
- Information on how the images are interleaved
- Information on how the analog media was digitized

You can describe the following by using the RGBA image format:

- The color palette used for the component video
- The order of the components in the data

You can describe the following by using the CDCI image format:

- The ratio of luminance sampling to chrominance sampling
- The black and white reference levels
- The color range used to determine the color difference
- The JPEG tables used to compress the images

TIFF Video Format

The TIFF format is based on the TIFF specification, Version 6.0. The baseline TIFF, as specified in Version 6.0, is the minimum subset that an OMF Interchange application must be able to read and write. The TIFF format provides a means of exchanging video data stored in these formats:

- Red-green-blue (RGB) pixel arrays; the RGB data can be 24-bit or 8-bit data
- The JPEG File Interchange Format (JFIF) format
- The YCC compressed format

RGB pixel arrays are the simplest required format for representing video data in an OMF Interchange file.

The YCC format is a standard compression format for JPEG data. It uses a standard ratio of luminance information to chrominance information.

The JFIF standard is a compressed video format that results in high compression of the input data (sometimes 50:1 and higher) with minimal degradation in the appearance of the decompressed frames. Although this JPEG format is a “still image” format, it is suitable for representing video data as a series of frames, provided that there is some auxiliary information such as frame rate.

Animation Formats

There is currently no widely accepted, multiplatform, standard format for representing animation in an object-oriented fashion. For the most part, applications exchange animations by rendering them as a series of frames. As a result, the OMF Interchange required format allows the required video or graphics formats specified previously.

The lack of a common animation format limits the quality of the rendered animation to that of the source that originated the animation. As OMF Interchange evolves, a goal is the selection of an object-oriented animation data format.

Required Audio Formats

Because audio data is usually represented in a form that conforms to the byte ordering of the target platform, OMF Interchange provides the required audio formats AIFC and RIFF WAVE, which support the two OMF byte orderings.

Applications using the same byte ordering can exchange data in their native byte order without the need for translation. Applications using different byte ordering can still interchange audio data, but with a translation step required for playback.

AIFC Format

The Audio Interchange File Format (AIFF) is a widely used format based on the AIFF 85 standard. AIFF includes the AIFC format for both uncompressed and compressed data. It specifies parameters such as sample size, number of channels (interleaving), and sample rate, and it provides a wrapper for the raw audio data.

AIFC also specifies that the data will be in big-endian byte order. This provides a native format for applications running on systems that use this byte ordering. The uncompressed form of AIFC data is supported as an OMF Interchange required format; compressed audio format is not included in the required format in this version of OMF Interchange.

WAVE Format

The RIFF Waveform Audio File Format (RIFF WAVE) specifies data that will be in little-endian byte order. This provides a native audio format for applications running on systems that use this byte ordering. The uncompressed form of RIFF data will be supported as an OMF Interchange required format; compressed audio format is not included in the required format in this version of OMF Interchange.

The Registered Formats

Contact the OMF Developers' Desk for information on registered formats.

2 Media Concepts

Compositions and source information work together in an OMF Interchange file. This section introduces concepts and terms and describes the logical structure of compositions and source descriptions in an OMF Interchange file.

There are three main topics in this section:

- Composition building blocks

  OMF Interchange defines components that an application can create and arrange to form a composition.
- Physical source information building blocks

  OMF Interchange defines elements that describe a physical source and the digital media data to be played.
- Time management

  Compositions manage time in a way that allows a nonlinear approach to editing, unlimited access to physical source material, and guarantees synchronized playback of physical sources that do not share the same native sample rate.

Composition Building Blocks

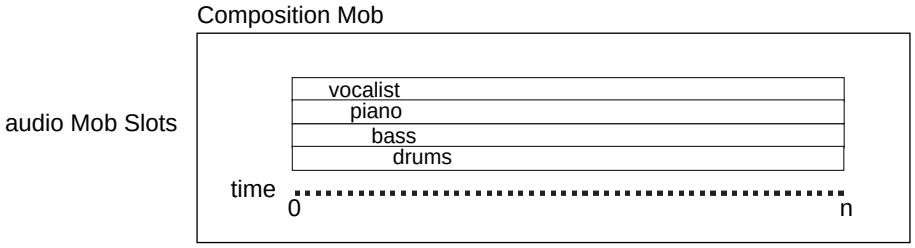
Composition Mobs contain all the information required to play a piece of media data or an entire media presentation. A single Composition Mob may represent something as small as a few frames of video or as large as an entire movie. The Composition Mob represents both playable media data and a network of behind-the-scenes information about how the media fits together.

Remember that the Composition Mob contains no actual media data. Rather, it is descriptive information that acts as a model of a media presentation. This section describes the parts of this model and shows how the parts are related over time.

Composition Mobs and Mob Slots

A Composition Mob is a collection of synchronized Mob Slots. Each Mob Slot represents a single type of media. A Mob Slot typically represents playable media, such as video and audio, but a Mob Slot can also represent descriptive information. For example, a Mob Slot can include timecode, edge code, or effect control arguments.

A Composition Mob, for instance, could represent a four-channel audio presentation, with one Mob Slot for each of the audio channels, as shown:



You can think of each Mob Slot as a logical playback channel. Placing the Mob Slots in a Composition Mob indicates that the four channels are to be played at the same time and synchronized.

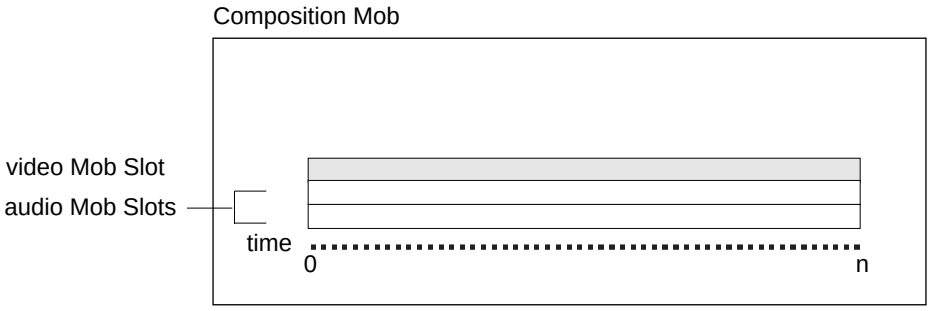
Typically, a Mob represents a number of Mob Slots for different media types, such as video and audio to be played together, starting at the same time and ending at the same time.

A single Mob Slot may contain a Sequence that represents multiple pieces of media. The order of the pieces within the sequence determines the playback order in that channel. For example, a video Mob Slot could consist of sections from the same or different sources, as shown:

A video Mob Slot consisting of six sections:

The Mob is a key mechanism in OMF Interchange. It imposes the synchronization of media; at the highest level, it synchronizes the Mob Slots in a composition.

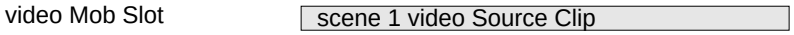
The Composition Mob controls the synchronization among the different Mob Slots. All Mob Slots in a Composition Mob must be played simultaneously, as demonstrated in the following diagram:



Segments

Mob Slots can contain a Segment of media or a Sequence of Segments of media. The simplest kind of Segment is a Source Clip, which represents a section of media. The following diagram illustrates the relationship between Mob Slots and Segments:

A Mob Slot containing only one Segment:



A Mob Slot containing a sequence of two Segments:

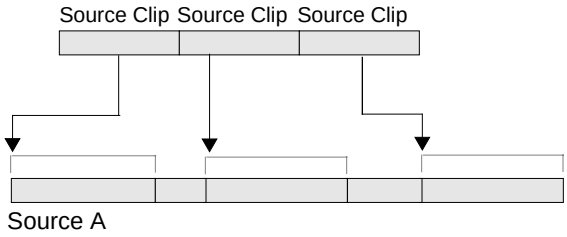
The order of the Segments in a Sequence specifies the order in which the Segments should appear when the composition is played.

Source Clips to Represent Media

Compositions use Source Clips to represent the media for a Segment. A Source Clip in a Composition Mob represents a section of digital media data in a file.

It identifies the data by storing the file source mob ID and describing the starting position and the length of the data.

A Sequence of three Source Clips from the same source:

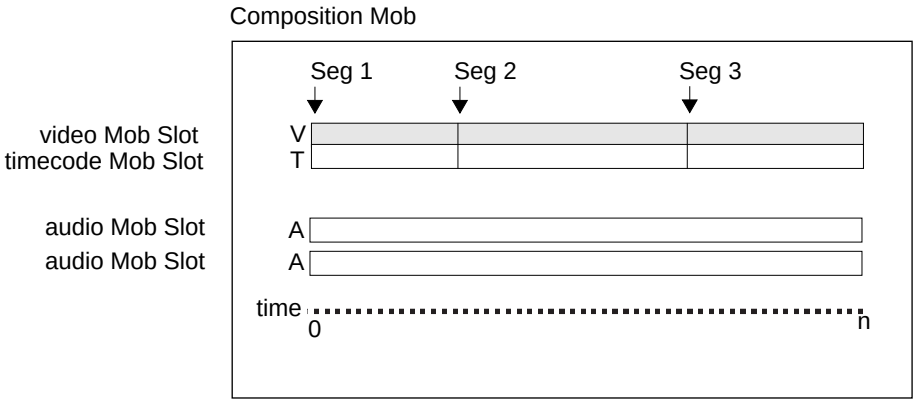


Source Clips are a mechanism for describing the actual media data that the Composition Mob uses. This data might be the digital media data contained in a file, live data being digitized from an external source, such as a camera or CD player, or a precomputed effect. Source Clips do not contain the media data itself, they reference the media data. The information in the source clip refers to a current digitized version of the media.

Timecode

Timecode provides a means of storing the timecode associated with a Segment. A Timecode can appear in a Mob Slot. For example, assume you are selecting synchronized segments of video and audio that were digitized from a videotape and adding them to a Mob Slot in a Composition Mob. To keep the SMPTE timecodes for each Segment accessible during editing, you could include a Timecode in a Mob Slot.

A composition that includes a Timecode Mob Slot:



The editing application might display the source timecode for each Segment as you view it.

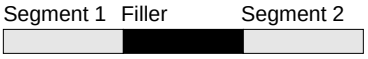
Similarly, an application can use Timecode to represent the composition-wide time associated with each Segment. In this way, the application could display the total Mob Slot length in timecode or the timecode associated with any point in the Mob Slot.

Edge Code

Edge Code is similar to Timecode, except that it stores the edge code associated with a film source. An Edge Code is used in a Source Mob describing film. Edge Code information is necessary to produce a description of the film source, such as when creating a cut list.

Filler

When there is no media specified in a Mob Slot or for a time period within a Sequence, the Filler object shows that there are no specified values.



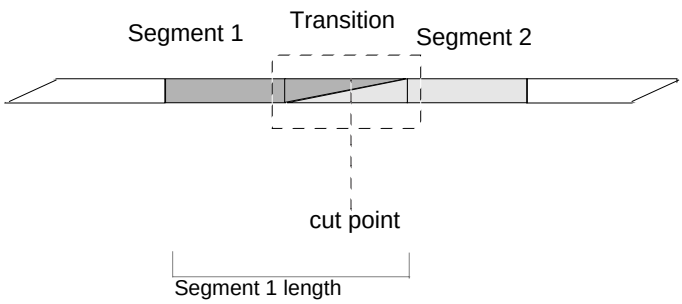
Effects as Segments

An Effect Invocation can be a Segment in a Mob Slot or Sequence within a Mob Slot. When played, an Effect Invocation produces a single stream of media data from one or more input Segments. Effect Invocations can also have control arguments. The application reading the Effect Invocation can interpret the information in the control arguments based on the effect documentation.

Transitions

A Transition specifies the way to go from the end of one Segment to the beginning of the next. It specifies an effect to use such as an audio cross-fade or a video dissolve.

A transition for a dissolve:



A cut is a simple divider between the two Segments in a Sequence. A Transition specifies an effect to be computed from the media in the Segments that precede and follow the Transition. OMF Interchange does not require a Transition between Segments; if one is not present, a simple cut from one Segment to the next is implied.

The Transition specifies an Effect Invocation to be used when going from the preceding Segment of media to the following Segment. The Effect Invocation object can specify a rendered or precomputed representation of the Transition effect. In this way, applications that can create the effect in real playback time may do so, while those that cannot may use the rendered version as a normal piece of digital media data.

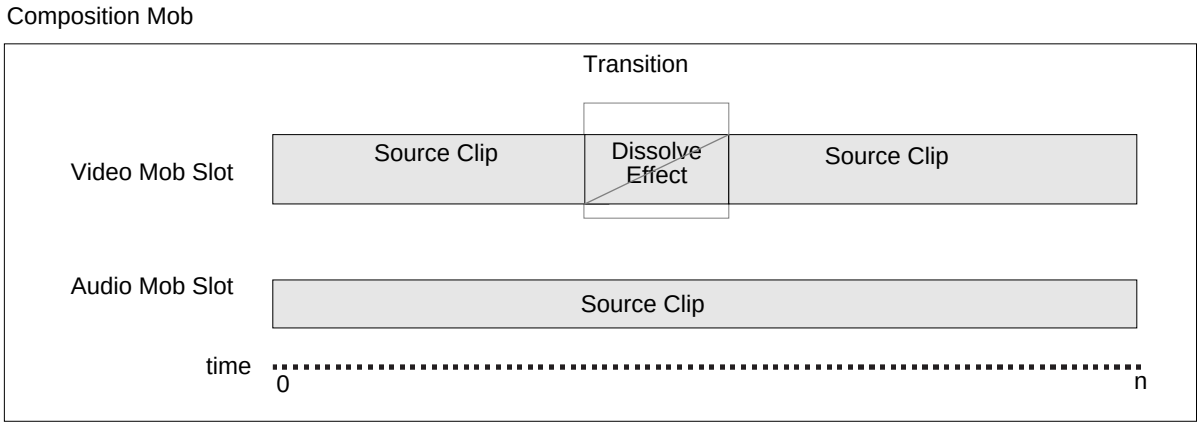
A Scope Reference

A Scope Reference gets its media by reference to another slot within the Composition Mob. For example, a Scope Reference in one video Mob Slot can reference a section of the image media described in another Mob Slot.

Scope References provide a general way of implementing layered effects in a Composition Mob. For example, the simplest way to put a title over a video Segment is to include the two Segments in an Effect Invocation that specifies that the title Segment should be superimposed over the video Segment. However, by using a Scope Reference, it becomes much easier to edit the Composition Mob because you can change the underlying video Segment without changing the Effect Invocation or the title Segment.

An Example Composition Mob

Here is a simple example that shows the Components in a Composition Mob. The Composition Mob synchronizes all the Mob Slots within it. The following figure shows a small Composition Mob containing a video Mob Slot and an audio Mob Slot. The audio Mob Slot consists of only one Component: a Source Clip for audio to be played. The video Mob Slot contains a Sequence of two Source Clips separated by a dissolve Transition.



Source Building Blocks

Recall that sources are represented by Source Mobs. A Source Mob can be a file Source Mob or a physical Source Mob such as a tape Source Mob, a film Source Mob, or an audio tape Source Mob. Each Source Mob describes its media data and can optionally provide information that may help the user or application locate the previous generation source.

Mob Slots in Source Mobs

A Source Mob that represents a physical source such as a film or tape has Mob Slots that represent the information contained by the source. These physical Source Mobs are simpler than Composition Mobs; they contain one Mob Slot for each type of information on the source. Typically, each Mob Slot contains a single Source Clip that represents the length of the source known to the application that created the Source Mob.

For example, a video cassette might have a video track and two audio tracks. If both video and audio information is digitized from this tape, then the application creates a tape Source Mob with three Mob Slots. The length of the Source Clips is set to the length of the digitized section.

It is also useful to include a timecode or edge code track in the Source Mob for a physical source that uses one of these measurement systems. This lets users of the source maintain precise positioning of samples relative to the source. A Source Mob contains this information in a Mob Slot that has a Timecode or Edge Code.

A file Source Mob that represents digital media data has a single Mob Slot with a single Source Clip that describes the nature and length of the digitized information in the file. Its source position represents the point of the first sample within the original physical source. The Source Clip references the Mob for the original source, if it exists, and the appropriate Mob Slot within that Mob.

In the example above, digitizing from the video tape produces three file Source Mobs, each with a single Mob Slot and Source Clip: one for the video track, and one for each audio track.

The digital data described by a file Source Mob must not be changed once it is created, because doing so would invalidate Mobs that reference it. If media data is digitized from the physical source again, new Mobs must be created.

Maintaining references in Composition Mobs while allowing the media data to be redigitized is accomplished by using an intermediate Master Mob. This Mob contains a Mob Slot for each of the synchronized media tracks digitized together, each with one Source Clip referencing one of the file Source Mobs. In the example above, the Mob would have three Mob Slots. Composition Mobs using the data reference the Master Mob instead of the file Source Mobs; if new

media data is digitized from the tape, only the Master Mob needs to be updated. The references within Composition Mobs remain intact.

Media Descriptors

A Source Mob includes a Media Descriptor that describes the kind of source and describes the format of the media data. A Media Descriptor for a film source might contain the film aspect ratio; a Media Descriptor for a videotape source might contain the tape format.

Media descriptors for file Source Mobs can include information about the location where the digital media data is stored. For example, a Media Descriptor can include the directory name and filename for a file.

Media Descriptors for Digital Media Data

In a file Source Mob, the Media Descriptor is specialized for the type of media data. An RGBA Component Image Descriptor describes the characteristics of component video digital media data. A Color Difference Component Image Descriptor describes the characteristics of image digital media data stored with one luminance component and two color difference components. An AIFC Audio Descriptor describes the characteristics of audio samples in the AIFC format. A WAVE Audio Descriptor describes the characteristics of audio data in the WAVE format. A TIFF Image Descriptor describes the characteristics of video, graphic, or still-image data in the TIFF format.

The information in these types of media descriptors is necessary for interpreting the sample data for editing or playback. For example, it is necessary to know the audio sample rate, the video frame rate, whether the video data is compressed, and the type of compression.

An important OMF Interchange feature is that the media descriptor provides enough information to enable an application to interpret and play the piece of data without a supporting composition.

Applications can add properties to the Media Descriptor to store their own information about the data it describes. These properties could contain information about the media content, the media quality, or any other information that an application may choose to record.

The Digital Media Data

The sample data in an OMF Interchange file is contained in a Media Data object. An Image Data object contains component video data, a JPEG Image object contains JPEG compressed video data, an AIFC object represents AIFC data, a WAVE object represents WAVE data, and a TIFF object represents TIFF data. The Media Data object contains the digital media data and the Mob ID of the file Source Mob that represents it. A Media Data object describes the

number of bytes occupied by the data, and it can also include version information.

File Source Mobs and Media Data Objects

An application can store digital media data in a file that is separate from Composition Mobs that reference it. OMF Interchange provides a framework that includes both recommended and required practices for maintaining references to the external digital media data.

Creating a File Source Mob

Once created, the digital media data associated with a file Source Mob is immutable: it cannot be updated or modified. If it is necessary to change the digital media data, an application must delete the Mob ID and create a new file Source Mob.

When the Digital Data Is External

When an OMF Interchange file contains a composition that references external digital media data, it must contain a copy of the file Source Mob and the Master Mob. Ideally, the file Source Mob provides hints as to the identity and location of the file that contains the Media Data object. However, it is the Mob ID that is the actual link to the data.

When the Digital Data Is Internal

OMF Interchange files containing digital media data must also contain the file Source Mob and Master Mob that represents this data. Multiple Mobs and the data they represent can be contained within a single file; for identification, the Media Data objects contain the Mob ID of the file Source Mob to which they belong.

Raw Digital Data in a Non-OMF Interchange External File

Situations can arise in which a Composition Mob references digital data in an external file that is not an OMF Interchange file. An application can create a file Source Mob and Master Mob for this external data, and it can use its own method of handling the data.

Time Management

Composition Mobs usually use a variety of media types from different physical sources. In order to combine sections of these sources and synchronize them, a composition must establish a common time unit for editing and playing all of the different types of media.

Edit Units and Edit Rate

Compositions measure sections of digital media data in abstract units called "edit units" (EUs). An EU is a unit of duration, representing the smallest interval of time that is meaningful to a composition. The number of EUs per second is called the "edit rate."

A Mob Slot in a Composition Mob uses an edit unit that is convenient for the editing process. When editing video data, for example, an editor edits frames, based on a certain display rate of frames per second.

But, while the composition describes media data in editable units, the physical source provides the data in sample units, which are the time duration of media represented by each sample in the media data file.

Another way of thinking of edit rate is as a "virtual" sample rate. This rate may or may not match the actual sample rate of the digital media. When it does not match, an OMF Interchange file provides the necessary data for converting EUs to sample units in order to access the data.

Sample Units and the Sample Rate

Media data consists of digitized samples of physical sources. When you digitize media data, you use a sample rate. The sample rate is the number of samples per second. The sample rate determines the duration of one sample.

A file Source Mob stores the sample rate of its digital media data. It also identifies the digitized section in the physical source by storing a start position measured in sample units.

3 The OMF Class Model

OMF uses a class model to describe compositions and media. The OMF class model specifies how to define the classes and objects used in OMF Interchange files.

Although this chapter provides a brief introduction to some object-oriented concepts, it assumes that you have a basic understanding of object-oriented programming.

Benefits of the OMF Class Model

The OMF Interchange Specification was designed according to an object-oriented paradigm because this paradigm provides an efficient abstract model to describe complex structures and relationships. This abstract model allows OMF to focus on the important data needed to describe media and compositions and to defer the details of the data formats used to store this information to a lower level of specification. This model provides the following benefits:

- It makes it easier to understand the OMF Interchange file format
- It makes it easier to write programs that create or read OMF Interchange files if the programmer chooses to adopt the class model in the program
- It allows you to extend OMF Interchange by defining new classes that are subclasses of the classes defined in this document

Elements of Object-Oriented Systems

This section provides a brief introduction to object-oriented modeling systems.

Note If you are familiar with object-oriented systems or languages, you can skip this section. If you are not familiar with object-oriented systems or languages, read this section to get a quick overview that will help you understand the descriptions in the following sections of this specification. Please be aware that this is an abridged overview and omits fundamental concepts not pertinent to OMF that are otherwise present in typical object-oriented systems.

Two basic concepts of object-oriented systems are objects and classes. An object is an individual item that can be a physical object or a model of a physical object. A class is an abstract description of a set of objects. A class can be defined in terms of properties and rules that the objects must follow. Once you have defined a class, an object that belongs to that class is said to be an instance of the class.

Class Model Terminology

This section defines the terms used in the OMF Interchange class model. Other object-oriented systems may define these terms differently.

object — An object has a collection of *properties,* each of which has a name and a value. An object is-an-instance-of of a class.

class — A class is a category of *objects*. The objects have common properties, relationships, and semantics. OMF objects do not have behavior because methods (code to perform actions on objects) are not stored in the object. Applications using OMF objects must supply the behavior based on the OMF Interface Specification.

inheritance — Inheritance is the mechanism that defines a relationship between *classes* where a *subclass* inherits the *properties,* relationships, and semantics of its superclass.

subclass — A subclass is a class that is defined as having the properties, relationships, and semantics as another class, which is called its *superclass*. The subclass can have additional properties, relationships, and semantics that are not in the superclass.

superclass — A superclass is a class that has another class, its *subclass,* that is defined as having the properties, relationships, and semantics as the superclass.

is-an-instance-of — The is-an-instance-of relationship is a relationship between an *object* and a *class*. An object is-an-instance-of a class if the object is in the *set* defined by the class.

is-a-kind-of — The is-a-kind-of relationship is a relationship between two *classes*. If a class has the properties, relationships, and semantics of a second class, then the first class is-a-kind-of the second class. The first class is called the *subclass* and the second class is the *superclass*.

substitutability — The rule of substitutability specifies that an *object* can be used in place of an object of any *class* that is a *superclass* of its class.

abstract class — An abstract class provides a way to refer to a group of classes. An *object* belonging to an abstract class must also belong to a nonabstract class that is a *subclass* of the abstract class.

data model — The data model is the high-level description that specifies the logical and semantic meaning. In contrast, the *implementation* is the lower-level description of a class that specifies the storage details.

implementation — The implementation is the lower-level description of a class that specifies the storage details. In contrast, the *data model* is the high-level description of a class that specifies meaning.

HAS — The HAS relationship is between an *object* and a property *value*. A value can be simple, such as a number or a string, or can be another object. If the value is another object, then that object is owned by the object that HAS it. In the data model diagrams, the HAS relationship is indicated by a solid line from a property to an object. The HAS relationship is also called CONTAINS or OWNS.

HAS-REFERENCE — The HAS-REFERENCE relationship is between an *object* and a property *value* that is another object. An object does not own the other object that it has a reference to, and more than one object can have a reference to a single object. In the data model diagrams, the HAS-REFERENCE relationship is indicated by a dashed line from a property to an object. The HAS-REFERENCE relationship is also called DEPENDS-ON or USES.

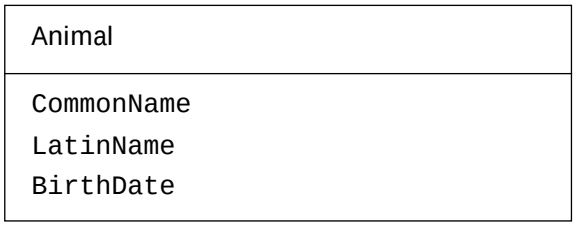
set — A set is an unordered collection of unique values. This is sometimes used in *class* definitions to store multivalued *properties.*

ordered set — An ordered *set* is an ordered collection of unique values. This is sometimes used in *class* definitions to store multivalued properties when ordering is important.

Example Class Model

The examples in this section are not part of OMF—these examples are just used to introduce object-oriented concepts. The example classes in this section are part of an object-oriented system designed for a zoo. The Animal class models any individual live animal. It has the following data model:

Data Model

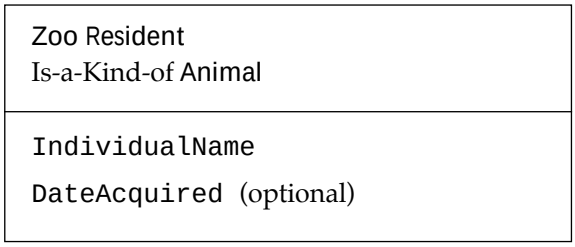


The class name is Animal and there are three properties: `CommonName`, `LatinName`, and `BirthDate`. Objects that belong to this class include a tiger in the zoo, a turtle on a Pacific island, and a fish in the author's aquarium.

Objects are not included in the class if they do not have one of the properties defined for the class or do not follow the rules and description of the class. For example, a fictional animal is not alive and does not belong to the class. In addition, if an animal's birth date is unknown, it would not belong to the class. To include animals with unknown birth dates, the `BirthDate` property could be specified as optional.

You can define subclasses that include additional properties or usage rules. For example, you can define the Zoo Resident class for live animals that are residents of the zoo with the following definition:

Data Model



An object in the Zoo Resident class is also in the Animal class. It has all the properties of both classes.

The subclass is a means of specializing the original or superclass so that an object in a subclass is a special case of the superclass. This object has all the properties of the superclass, as well as added information in its own specialized properties.

A property of a class can have a simple value, such as a number or text string, or it can have another object as its value. For example, in the following ZooHabitat class, the `Residents` property has a value that is a set of Zoo Residents.

Data Model

If the zoo also provided behavior training for pets that are privately owned and do not belong to or live at the zoo, they could be in the Pet Student class, which has the following properties:

- `CommonName`
- `LatinName`
- `BirthDate`

- `IndividualName`
- `DateAcquired`
- `Owner`
- `TrainingSession`
- `Grade`

Since this class has all of the properties that the Zoo Resident class has, one might be tempted to make the Pet Student class a subclass of it. Although objects in the Pet Student class have all the properties defined for the Zoo Resident class, Pet Student should *not* be defined as a subclass of Zoo Resident because the objects in the class do not fit the rules and description of the Zoo Resident class. They are not residents of the zoo.

The Pet Student class should be defined as a subclass of the Animal class with the following definition:

Data Model

Pet Student Is-a-Kind-of Animal

`IndividualName`
`DateAcquired` (optional)
`Owner`
`TrainingSession`
`Grade`

When defining a new class, you can reuse the name of a property in another class; reusing the name does not imply any inheritance.

OMF Interchange Class Model

OMF defines a set of classes. A class specifies a kind of object by defining its use and properties. The class definition does not itself appear in the OMF Interchange file; only objects that are instances of the class appear in the file.

OMF objects consist of a series of properties. An object stores information by having a property for that piece of information. Each property has a name and a value. The property value has a data type. The value of a property can be a simple data value or another object. For example, the Media File Descriptor (MDFL) object contains a `SampleRate` property that specifies the number of samples per second of the digital media data.

Class Hierarchy

OMF Interchange uses a class hierarchy to group objects that can be used in the same way and to define common properties through inheritance. Classes inherit the properties of classes above them in the hierarchy.

The hierarchy represents classes as a means of specializing superclasses, so that an object in a subclass lower in the hierarchy is a special case of the superclass. This object has all the properties of the superclass, as well as added information in its own specialized properties. The hierarchy makes it easier to define new classes and to create rules about how objects can be used.

A particular object has the required properties for its class. This includes all the required properties of its superclass and the required properties for each preceding superclass back to the root of the hierarchy.

Figure 2 illustrates the OMF class hierarchy. Appendix F contains a fold-out version of this illustration. Table 2 lists the four-character Class IDs shown in the class hierarchy diagrams and the corresponding full class name.

The names of the properties include the Class ID of the class that defines them. When a subclass inherits the properties from a superclass, the property name retains the superclass ID.

For example, both the Source Clip (SCLP) and Effect Invocation (EFFE) classes are subclasses of the Segment (SEGM) class, which is itself a subclass of the Component (CPNT) class. This means that anywhere a Segment is allowed, such as in a Sequence (SEQU) object, you can use a Source Clip or Effect Invocation object. Since the Component class has the `OMFI:CPNT:Length` and `OMFI:CPNT:DataKind` required properties, all Source Clip and Effect Invocation objects will have these properties.

Note

Some class definitions specify rules restricting substitutability. For example, the Source Mob and Master Mob classes have rules that state that Effect Invocations are not allowed within them.

How Classes Are Defined in OMF

Classes are defined in this specification with a data model and an implementation. The data model specifies the superclass and the properties that contain the specialized information for the class. The implementation lists the complete set of properties (including those that are inherited). The implementation definition provides a description of each property and specifies the data type of its value.

All classes in the OMF class hierarchy are subclasses of the OMFI Object (OOBJ) class. The OMFI Object class has one property: `OMFI:OOBJ:ObjClass`. The value of this property for all OMFI objects is the four-character class ID of the class of the object. The `ObjClass` value is the class ID of the most specialized class of which the object is an instance. To find

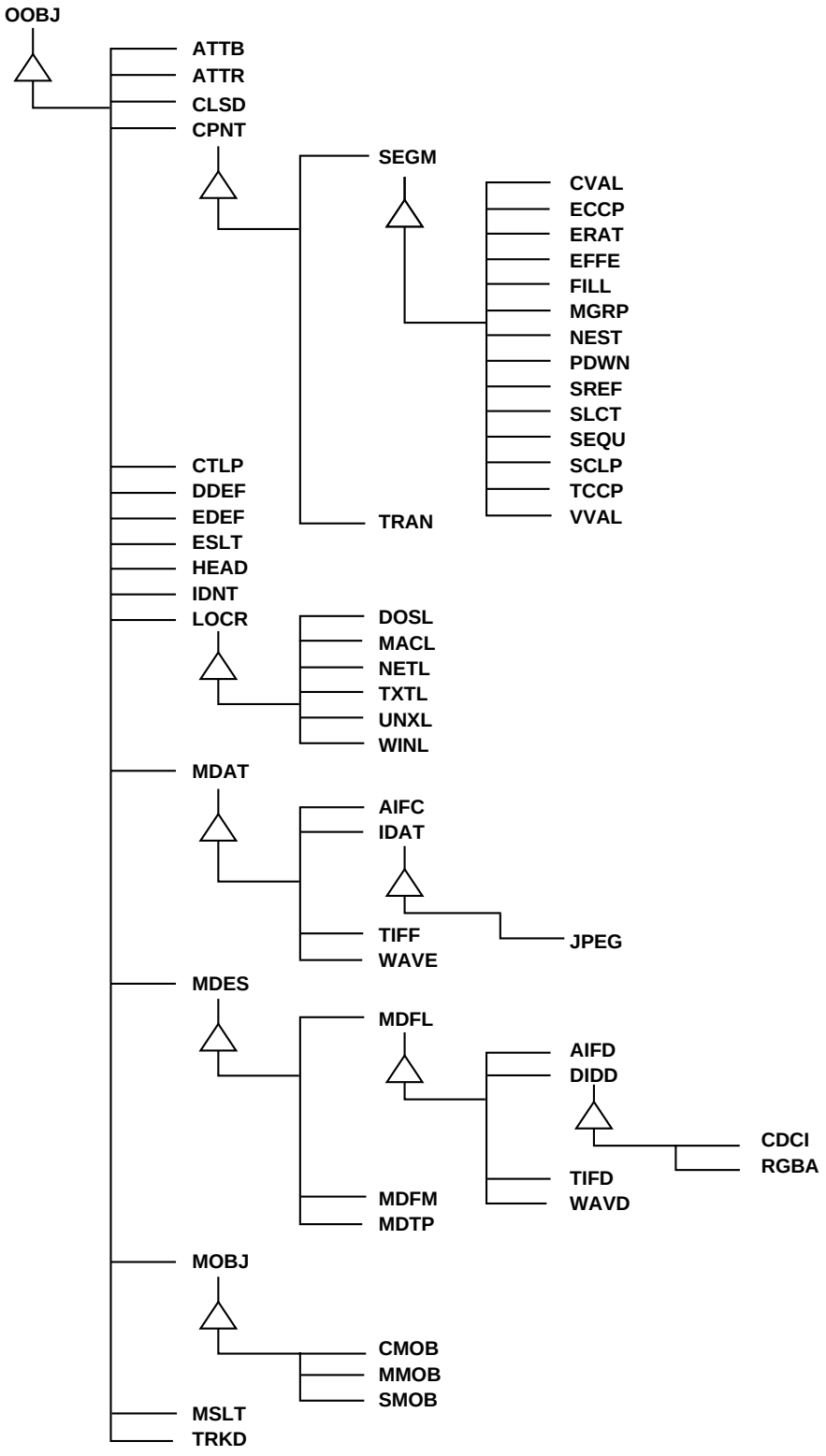


Figure 2: OMF Class Hierarchy

Table 2: Class IDs and Full Class Names

Class ID	Class Name
AIFC	AIFC Audio Data
AIFD	AIFC Audio Descriptor
CDCI	Color Difference Component Image Descriptor
CLSD	Class Dictionary
CMOB	Composition Mob
CPNT	Component (abstract)
CTLP	Control Point
CVAL	Constant Value
DDEF	Data Definition
DIDD	Digital Image Descriptor (abstract)
DOSL	DOS Locator
ECCP	Edge Code
EDEF	Effect Definition
EFFE	Effect Invocation
ERAT	Edit Rate Converter
ESLT	Effect Slot
FILL	Filler
HEAD	Header
IDAT	Image Data
JPEG	JPEG Image Data
LOCR	Locator (abstract)
MACL	MAC Locator
MDAT	Media Data (abstract)
MDES	Media Descriptor (abstract)
MDFL	Media File Descriptor (abstract)
MDFM	Media Film Descriptor
MDTP	Media Tape Descriptor
MGRP	Media Group
MMOB	Master Mob
MOBJ	Mob (abstract)
MSLT	Mob Slot
NEST	Nested Scope
OOBJ	OMFI Object (abstract)
RGBA	RBGA Component Image Descriptor
SCLP	Source Clip
SEGM	Segment (abstract)
SEQU	Sequence
SLCT	Selector
SMOB	Source Mob
SREF	Scope Reference
TCCP	Timecode
TIFD	TIFF Image Descriptor
TIFF	TIFF Image Data
TRAN	Transition
TRKD	Track Description
TXTL	Text Locator
UNXL	UNIX Locator
VVAL	Varying Value
WAVD	WAVE Audio Descriptor
WAVE	WAVE Audio Data
WINL	Windows Locator

out the class of any object in an OMFI file, you can examine the value of the `ObjClass` property.

Data Model and Implementation

The descriptions of the properties in the data model and the implementation are similar, but there are important differences. One difference is that the implementation includes all properties stored with the object, and the data model does not list the properties inherited from the superclass. Another difference is that the data model and implementation contain different information for each property.

For example, the data model and implementation for the OMFI Object class are as follows:

Data Model

OMFI Object Class (OOBJ) Abstract Class
`ObjClass`

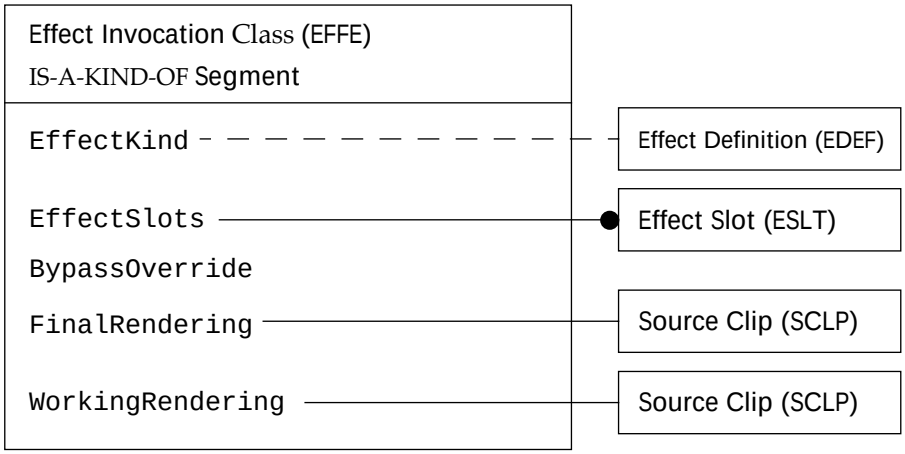
Implementation

Property Name	Type	Explanation
`OMFI:OOBJ:ObjClass`	`omfi:ClassID`	Class is OOBJ.

The implementation shows the full property and type names and lists the class ID, but otherwise contains the same information as the data model.

For properties whose value is another object, the data model and the implementation provide different information. For example, the following are the Effect Invocation data model and implementation are as follows:

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is EFFE.
OMFI:CPNT:DataKind	omfi:ObjRef	Specifies the data kind generated by the effect. Media effects typically have a data kind of omfi:data:Picture, omfi:data:PictureWithMatte, omfi:data:Sound, or omfi:data:StereoSound.
OMFI:CPNT:Length	omfi:Int32	Specifies the duration of the effect in edit units.
OMFI:EFFE:EffectKind	omfi:ObjRef	Identifies the kind of effect with an Effect Definition, which specifies the unique name of the effect.
OMFI:EFFE:EffectSlots	omfi:ObjRefArray	Specifies the Effect Slots that contain the input media Segments and the effect control argument Segments. Optional.
OMFI:EFFE:BypassOverride	omfi:ArgIDType	Specifies the ArgID value of the input media Effect Slot to be substituted for the Effect Invocation if the application cannot generate the effect. Overrides the bypass specified in the Effect Definition. Optional.
OMFI:EFFE:FinalRendering	omfi:ObjRef	Specifies a Source Clip that contains a rendered version of the effect intended for final use. Optional.
OMFI:EFFE:WorkingRendering	omfi:ObjRef	Specifies a Source Clip that contains a rendered version of the effect intended for viewing during editing but not intended for final use. Optional.

The data model illustrates a value that is another object by using a line from the property name to a box representing the other object. A solid line indicates that the property HAS the object as a value. A dashed line indicates that the property HAS-REFERENCE to the object. A filled-in black circle at the end of the

line indicates that the property HAS a set of the objects. In most cases, the HAS-REFERENCE relationship has the same implementation as the HAS relationship (using an `omfi:ObjRef`), but a few HAS-REFERENCE relationships are implemented with other mechanisms.

Types

OMF has two sets of types: the data type, which specifies the type of property values, and the data kind, which specifies the type of media represented by objects of the Component class. The implementation section describing a class lists the data type of each property that the class includes. Objects that belong to either the Component or Control Point class have a property that identifies the data kind of the object. Data kind is used to describe time-varying values produced by Components that describe media and Components that supply control arguments to Effect Invocations.

Data Type

The data type is identified by a globally unique text string that starts with the prefix `omfi:` and identifies the type. Table 3 lists the data types. See Appendix B for a complete explanation of the data types.

Table 3: Data Types

Data Type	Explanation
`omfi:Boolean`	Specifies either True or False.
`omfi:Char`	Specifies a single character value.
`omfi:ClassID`	Specifies the 4-character class identification.
`omfi:ColorSitingType`	Specifies how to compute subsampled values.
`omfi:CompCodeArray`	Specifies the order in which the RGBA components are stored.
`omfi:CompSizeArray`	Specifies the number of bits reserved for each component.
`omfi:DataValue`	Specifies media data or a block of data whose type is specified by a data kind.
`omfi:EdgeType`	Specifies the kind of film edge code.
`omfi:EditHintType`	Specifies hints to be used when editing Control Points.
`omfi:FadeType`	Specifies the kind of fade.
`omfi:FilmType`	Specifies the format of the film.
`omfi:InterpKind`	Specifies the method to use when interpolating between Control Points.

Table 3: Data Types (Continued)

Data Type	Explanation
omfi:Int8	Specifies an 8-bit integer value.
omfi:Int16	Specifies a 16-bit integer value.
omfi:Int32	Specifies a 32-bit integer value.
omfi:Int32Array	Specifies an array of 32-bit integer values.
omfi:JPEGTableIDType	Specifies the JPEG tables used in compressing TIFF data.
omfi:LayoutType	Specifies whether the image data is interleaved.
omfi:Length32	Specifies the length of a Component with a 32-bit integer value.
omfi:Length64	Specifies the length of a Component with a 64-bit integer value.
omfi:ObjRef	Specifies another object.
omfi:ObjRefArray	Specifies a set of other objects.
omfi:Position32	Specifies an offset into a Component with a 32-bit integer value.
omfi:Position64	Specifies an offset into a Component with a 64-bit integer value.
omfi:Rational	Specifies a rational number by means of an Int32 numerator and an Int32 denominator.
omfi:String	Specifies a string of characters.
omfi:TapeCaseType	Specifies the physical size and container of the videotape or audio tape.
omfi:TapeFormatType	Specifies the format used to store media on the videotape or audio tape.
omfi:TimeStamp	Specifies a date and time.
omfi:UID	Specifies a Mob ID.
omfi:UInt8	Specifies an unsigned 8-bit integer.
omfi:UInt16	Specifies an unsigned 16-bit integer.
omfi:UInt32	Specifies an unsigned 32-bit integer.
omfi:UniqueName	Specifies a qualified name which conforms to the OMFI naming conventions.
omfi:VersionType	Specifies a 2-byte unsigned OMFI version number.
omfi:VideoSignalType	Specifies the type of video signal on the videotape.

The `Rational` type can accurately represent the video edit rate of 29.97 frames per second as the numerator value `2997` and the denominator value `100.` This format avoids problems with round-off or imprecision for commonly used values in video and audio applications, and it does not bias the format toward any particular native format.

Applications use timestamps to record the current time, as when marking the last time a file is modified. OMF Interchange uses a timestamp format that allows applications to use any timestamp, whether it is local, GMT, or other. But, it makes it possible for applications to mark a timestamp as GMT in order to guarantee the ability to exchange files with a common reference to time.

Some properties can have one of two data types. These properties specify the length of a Component or an offset from the beginning of the Component. Properties specifying a length can have either the `omfi:Length32` or `omfi:Length64` data types and properties specifying an offset can have either the `omfi:Position32` or `omfi:Position64` data type. The 64-bit types allow OMF files to be larger than the maximum number of bytes that can be specified in 32 bits, which is approximately 4 gigabytes. Applications that cannot generate files that require the 64-bit types, may choose to use the 32-bit types to reduce the disk storage required by an OMF file.

Data Kind

The data kind is specified by a Data Definition object, which contains the globally unique text string of the data kind. The data kinds defined in this document start with the prefix `omfi:data:`. The meaning, internal format, and size of the data kind are not described in the Data Definition object. This information is provided in this document or in the documentation provided with registered or private media formats and Effect Definitions. Table 4 lists the data kinds. See Appendix B for a complete explanation.

Table 4: Data Kinds

Data Kind	Explanation
`omfi:data:Boolean`	Specifies either True or False.
`omfi:data:Char`	Specifies a single character value.
`omfi:data:Color`	Specifies a ColorSpace followed by a series of Rational, with one Rational for each color component in the color space.
`omfi:data:ColorSpace`	Specifies the color space used to describe color.
`omfi:data:DirectionCode`	Specifies one of 8 directions in a 2-dimensional plane.
`omfi:data:Distance`	Specifies a distance relative to the display dimensions of the image.

Table 4: Data Kinds (Continued)

Data Kind	Explanation
omfi:data:Edgecode	Specifies a stream of film edge code values.
omfi:data:Int32	Specifies a signed 32-bit integer.
omfi:data:Matte	Specifies a stream of media that contains an image of alpha values.
omfi:data:Picture	Specifies a stream of media that contains image data.
omfi:data:PictureWithMatte	Specifies a stream of media that contains image data and a matte.
omfi:data:Point	Specifies a point relative to the display dimensions of the image.
omfi:data:Polynomial	Specifies a polynomial value.
omfi:data:Rational	Specifies a rational number by means of an Int32 numerator and an Int32 denominator.
omfi:data:Sound	Specifies a stream of media that contains a single channel of sound.
omfi:data:StereoSound	Specifies a stream of media that contains two stereo channels of sound.
omfi:data:String	Specifies a string of characters.
omfi:data:Timecode	Specifies a stream of tape timecode values.
omfi:data:UInt8	Specifies an unsigned 8-bit integer.

Introduction to OMF Classes

This section contains a brief summary of some frequently used OMF classes. The complete list of classes and the complete description for each is in Appendix A.

Mob and Header Classes

- Composition Mob (CMOB)—describes the editing information and media that constitute a complete media production.
- Master Mob (MMOB)—provides an indirect way for a Composition Mob to reference a Source Mob and provides a way to synchronize data in mul-

tiple Source Mobs.

- Source Mob (SMOB)—describes media data; can be digital media data that is accessible to OMF (in which case it is a file Source Mob) or other media data stored in videotape, audio tape, film, or some other storage format.
- Header (HEAD)—specifies OMF Interchange file-wide information; there is exactly one Header object in each OMFI file.
- Class Dictionary Entry (CLSD)—extends the OMFI class hierarchy with a new class.
- Data Definition (DDEF)—identifies a globally defined data kind and so it can be referenced within an OMFI file.
- Effect Definition (EDEF)—identifies an effect so it can be referenced within an OMFI file.

Classes Used in All Mobs

- Mob Slot (MSLT)—specifies a track of media or other time-varying information.
- Track Description (TRKD)—provides the information needed to access a Mob Slot from outside the Mob.
- Source Clip (SCLP)—specifies a playable segment of media and identifies the Mob that describes the source of the media data.
- Sequence—combines a set of Components into a a single segment. The Sequence object is the basic mechanism for combining sections of media to be played in a sequential manner in an OMFI file.
- Filler (FILL)—specifies an unknown or unused section of media.

Classes Used in Composition Mobs

- Transition (TRAN)—specifies an effect that controls the transition between one piece of media and another within a Sequence.
- Effect Invocation (EFFE)—specifies an effect to be used in a Composition Mob; specifies kind of effect, input media slots and control arguments.
- Effect Slot (ESLT)—specifies the input media Segments and control arguments for an Effect Invocation.
- Constant Value (CVAL)—is typically used in Effect Invocations to supply control argument values.
- Varying Value (VVAL)—is typically used in Effect Invocations to supply control argument values.
- Nested Scope (NEST)—defines a scope of slots that can reference each other; the Nested Scope object produces the values of the last slot within it. Typically, Nested Scopes are used to enable layering or to allow a component to be shared.
- Scope Reference (SREF)—refers to a section in another Mob Slot or Nested Scope slot.

- Selector (SLCT)—specifies a selected Segment and preserves references to some alternative Segments that were available during the editing session. Typically, a Selector object is used to present alternative presentations of the same content, such as alternate camera angles of the same scene.
- Edit Rate Converter (ERAT)—converts part of a segment in one edit rate into a segment in another edit rate.

Classes Used in Master Mobs

- Media Group (MGRP)—provides alternative digital media representations.

Classes Used in Source Mobs

- Media Descriptor Classes (MDFM, MDTP)—describes physical media sources such as videotape, film, or audio tape.
- Media File Descriptor Classes (AIFD, CDCI, RGBA, TIFD, and WAVD)—describe sampled media sources.
- DOS, Mac, Text, UNIX, and Windows Locators (DOSL, MACL, TXTL, UNXL, WINL)—provide information to help find a file that contains the media data.
- Timecode (TCCP)—stores videotape or audio tape timecode information (also used in Composition Mobs).
- Edge Code (ECCP)—stores film edge code information.

Classes for Digital Media Data

- Image Data (IDAT)—contains sampled image media in RGBA Component or Color Difference Component Image format.
- JPEG Image Data (JPEG)—contains sampled image media that has been compressed using the JPEG algorithm.
- Audio Data Classes (AIFC and WAVE)—contain sampled audio media in the Audio Interchange Format (AIFC) or the RIFF Waveform (WAVE) format.
- TIFF Image Class (TIFF)—contains sampled image media in the TIFF format.

4

Mobs and the Header Object

This chapter describes Mobs and the OMF Header (HEAD) object. Mobs describe compositions and media. The Header provides the starting point to access objects in the OMFI file.

This chapter contains the following major sections:

- Mobs
- Header Object
- From Header to Media—an example OMF Interchange file showing the way to get from the Header object to the Composition Mob and the media that it uses

Mobs

A Mob is an OMF object that describes editing information or media by means of the kinds of objects it contains, the tree structure that connects these objects, and the property values within each object.There are several different kinds of Mobs used in OMF Interchange files, but all Mobs describe time-varying media information.

Mobs are the basic units of identity in the OMF world. They have a globally unique ID, and they are the only elements of an OMFI file that can be referenced from outside the file.

There are the following kinds of Mobs:

- Composition Mobs (CMOB)—describe editing information.
- Master Mobs (MMOB)—synchronize Source Mobs and provide a layer of

indirection to make it easy to change Source Mobs without changing Composition Mobs that reference them.

- Source Mobs (SMOB)—describe media. There are two kinds of Source Mobs:
  - File Source Mobs—describe digital media data stored in files.
  - Physical Source Mobs—describe media such as videotape, audio tape, and film.

A typical OMF Interchange file consists of one Composition Mob and the Master Mobs, Source Mobs, and media data used by the Composition Mob. However, there can be more than one Composition Mob in an OMFI file. Figure 3 illustrates an OMF Interchange file.

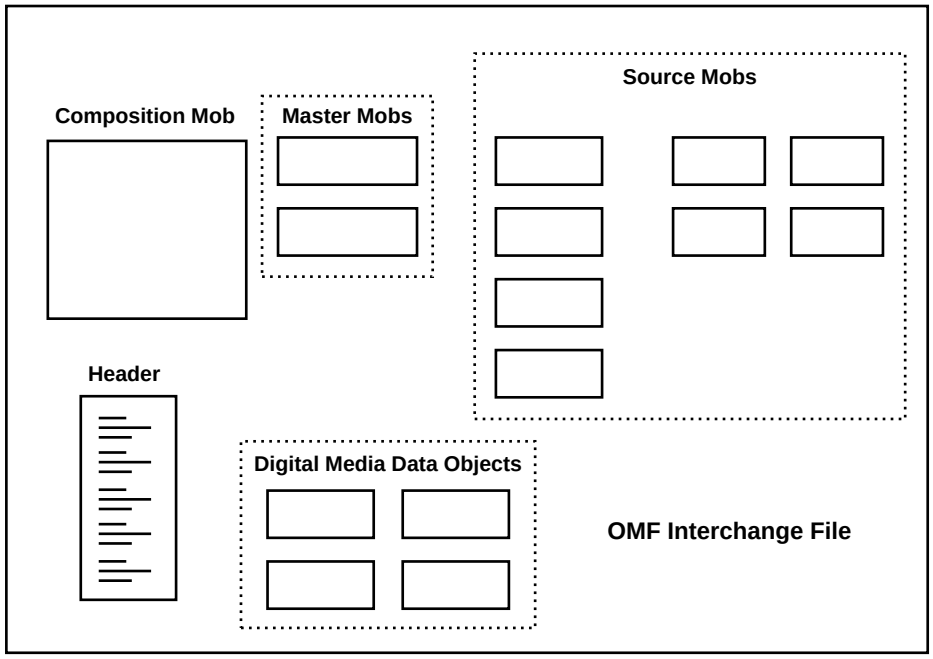


Figure 3: OMF Interchange File with Mobs and Digital Media Data

Mobs can have very simple or very complex structures. Typically, Master Mobs and Source Mobs have simple structures and Composition Mobs are more complex.

Figure 4 illustrates the structure of a Composition Mob and Figure 5 illustrates the structure of a file Source Mob. The meanings of the objects within the Mobs are described later in this document.

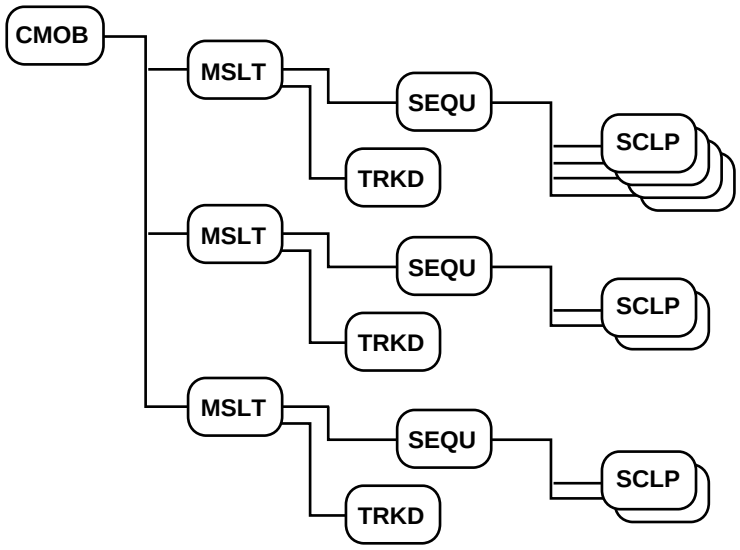


Figure 4: Structure Within a Composition Mob

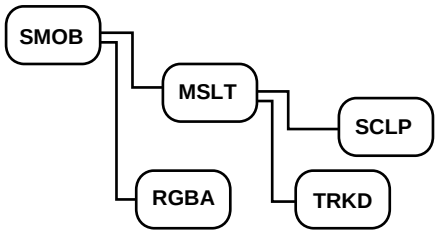


Figure 5: Structure Within a File Mob

Mobs and Immutable Media Data

Since Mobs can be referenced from outside the OMFI file containing them, it is important to avoid making any changes that invalidate these external references. In general, you can change Composition Mobs and Master Mobs but you are not allowed to change the Media Data associated with a file Source Mob. If the digital media data is changed in any way, you must create a new file Source Mob with a new `MobID`.

If you are revising the contents of a Composition Mob or Master Mob and intend the new contents to replace all existing copies of the Mob, you should use the same `MobID`. If you are revising the contents of the Mob but want the existing copy of the Mob to remain valid, you must create a new Mob with a new `MobID` and you must not reuse the `MobID` of the existing Mob.

Mob References

One Mob refers to another by having a Source Clip that specifies the second Mob's identifier. A Mob identifier, called a MobID, is a unique identification number assigned to a mob when it is first created. A Composition Mob can contain Source Clips that refer to many different Master Mobs. More than one Source Clip in a Composition Mob can refer to the same Master Mob, and Source Clips in different Composition Mobs can refer to the same Master Mob.

Mob Chain to Sources

In a typical OMF Interchange file, a Source Clip object in the Composition Mob points to a chain of Mobs. Following this chain enables you to get to the digital media data and to find out about the original source media. Figure 6 follows the Mob links from a Source Clip in a composition to the Master Mob, file Source Mob, and finally the physical Source Mobs describing the original media.

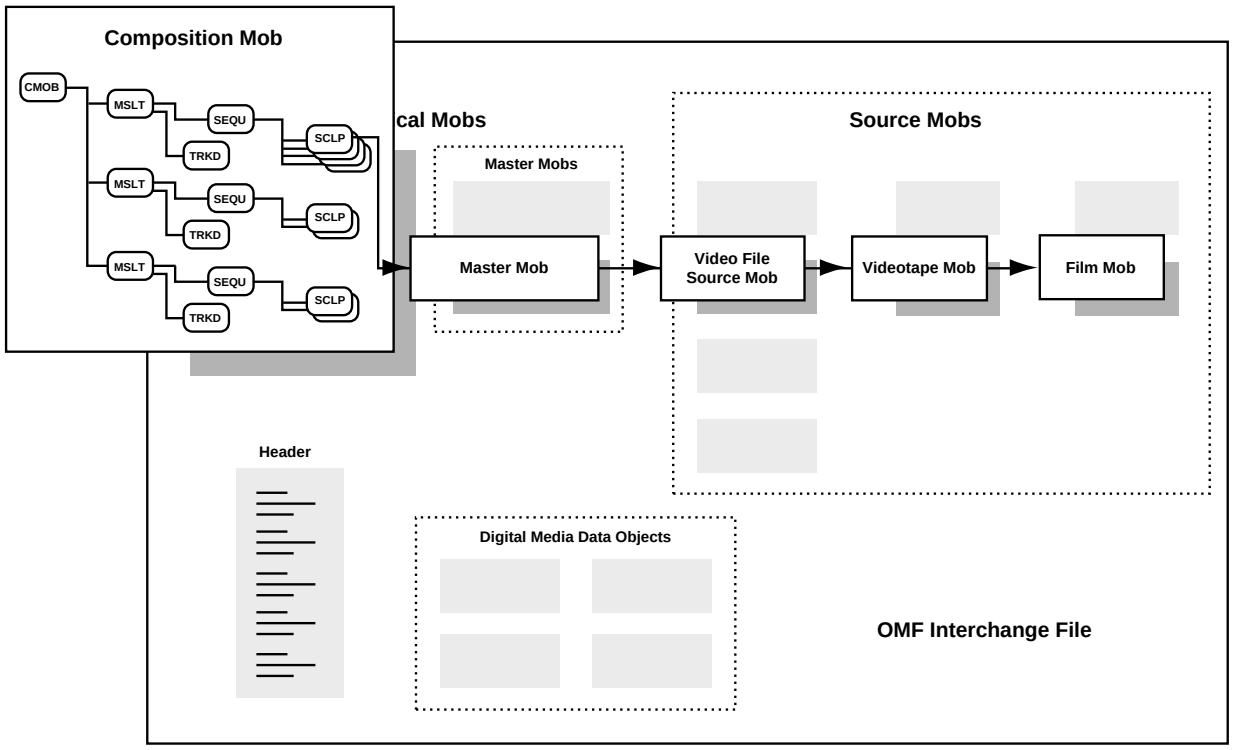


Figure 6: Mob Links from Composition to Physical Source

File Source Mobs and Media Data

A file Source Mob describes the digital media data and is used to access it, but does not contain the digital media data. OMF separates the description and the storage of the digital media data for the following reasons:

- Digital media data can be very large and may need to be stored in a separate file, on a different disk, over a network, or temporarily deleted to free disk space. Having the mob separate from the media data provides more flexible storage options while still allowing the composition to use the same access mechanism.
- Digital media data may be used in more than one Compositon Mob and these Composition Mobs can be in different OMF Interchange files. Separating the Source Mob from the digital media data means that only the information in the Source Mob needs to be duplicated in OMF Interchange files. Only one copy of the Media Data is needed.

The digital media data referenced in a Composition Mob can be stored in three ways:

1. In a Media Data object in the same OMFI file as the Composition Mob
2. In a Media Data object in a different OMFI file
3. In a raw data file

The `MobID` connects the file Source mob to the digital media data when the digital media data is stored in an OMFI file. The file Source Mob and its corresponding Media Data object have the same `MobID` value. This implies that there is always one and only one file Source Mob for each Media Data object, though there may be more than one copy of the file Source Mob.

Figure 7 illustrates how MobID is used to match the Media Data object with the file Source Mob.

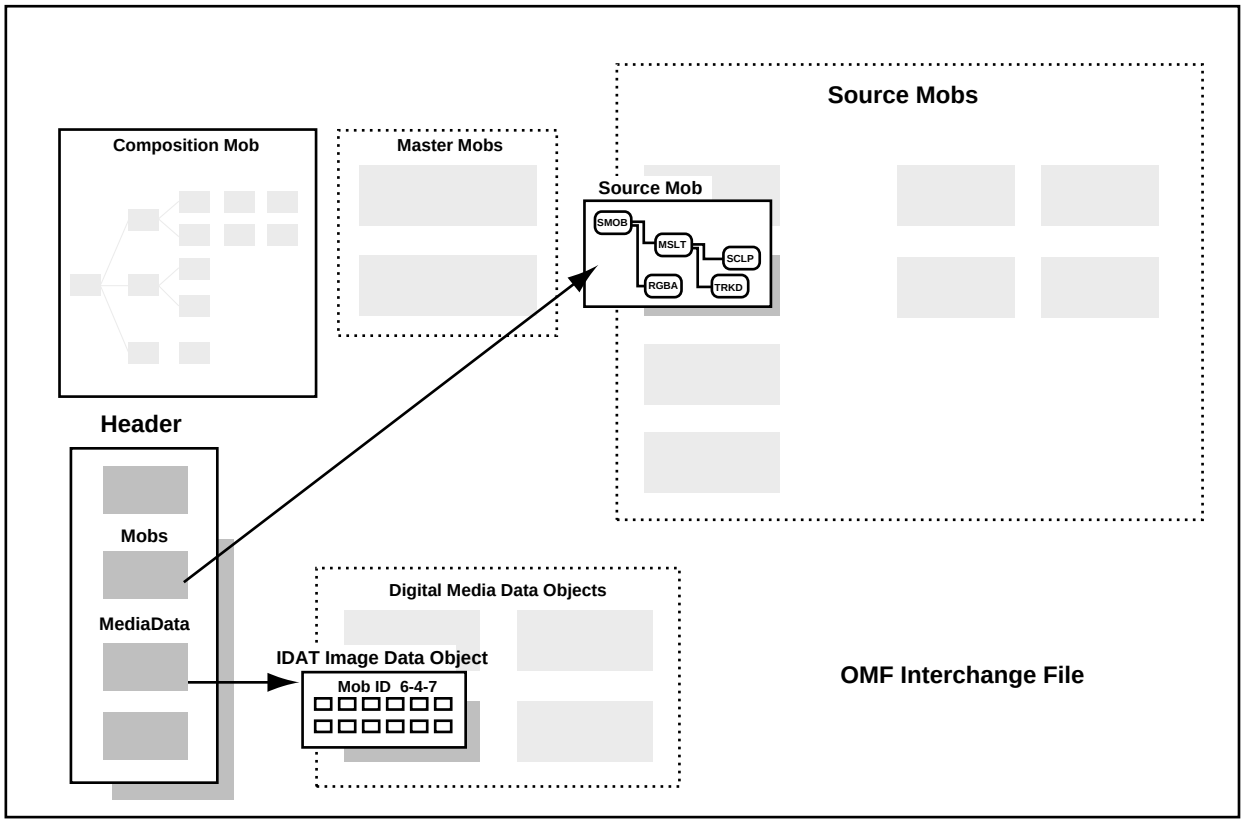


Figure 7: MobID Connection between File Source Mob and Media Data Object

A raw data file contains only digital media data; it does not contain any OMFI objects or structures. Storing digital data in a raw data file allows you to access the data both from applications that support OMF and those that do not. However, since there is no MobID stored with raw media data, it is difficult to identify a raw media data file if the file has been moved or renamed.

If a Composition Mob references Media Data in another OMFI file, the OMFI file containing the Composition Mob must contain a copy of the Master Mob and the file Source Mob. This copy can have Locator data that helps an application find the OMFI file containing the Media Data. The OMFI file containing the Media Data also must have a copy of the Master Mob and file Source Mob.

If a Composition Mob references digital media data stored in a raw data file, the OMFI file containing the Composition Mob contains a file Source Mob for the digital media data but does not contain a Media Data object. The file Source Mob contains Locator data that helps an application find the raw data file. If more than one Composition Mob references the same raw data file, they should use the same file Source Mob and Master Mob.

The Header Object (HEAD)

Each OMF Interchange file must have a single Header object (HEAD). The Header defines file-wide information and provides indexes for efficient access to the data in the file. It can also define extensions to the OMF classes, which can then be used in the OMFI file. The Header data refers to the file as a whole. Examples of file-wide information that are defined in the Header object are the byte order and the OMF Interchange Framework version number.

The Header object provides the starting point for accessing the information in the file. It lists Mobs and Media Data objects. In addition, it lists the primary Mobs— primary Mobs are the Mobs that the file is intending to communicate.

Byte Order

The value of `ByteOrder` property is either `'MM'` (hexadecimal `0x4d4d`) for big-endian byte order, which is used in some architectures such as the Motorola® 680x0 architecture, or `'II'` (hexadecimal `0x4949`) for little-endian byte order, which is used in some architectures, such as the Intel® x86 architecture.

Big-endian and little-endian refer to whether the most or least significant byte is stored first. In the big-endian byte order, the most significant byte is stored first (at the address specified, which is the lowest address of the series of bytes that comprise the value). In the little-endian byte order, the least significant byte is stored first. In both cases, bytes are stored with the most significant bit first.

Modification Date

The value of `LastModified` represents the last time the file was modified.

Class Dictionary Property

The `ClassDictionary` property defines file-wide extensions to the OMFI class hierarchy. It consists of an array of references to class dictionary objects. These objects are not required to define the standard class dictionary; they are required only for any extensions an application is adding for public or private objects.

The class dictionary property of the header object provides a mechanism for adding object classes to the standard set. Compliance with the standard objects is assumed for all applications and no class dictionary property is required for it. This property enables OMF Interchange readers to handle unknown objects.

Mob Index

Since references to Mobs are by MobID and not by object references, you need the Mobs property to have a list of all the Mobs in the file. This list contains the object references for all the Mobs in the OMFI file. You can examine each Mob to find its MobID and whether it is a Composition Mob, Master Mob, or Source Mob.

The PrimaryMobs index lists the mobs that you should examine first. If the OMFI file represents a single composition, the Composition Mob for it should be in the PrimaryMobs index. In addition, you may wish to include any other Mobs that you intend to be referenced independently by other Mobs.

The primary Mobs are the Mobs that the file is intending to communicate; all other Mobs are present only because they are dependencies of the primary Mobs. If an OMFI file has no primary Mobs, it is providing a repository for its Mobs but is not specifying which one should be used.

Media Data Index

Since Mobs are associated with Media Data objects by matching MobID values and not by object references, you need the MediaData property to be able to access all of the Media Data objects in the file. You can examine each Media Data object to find its MobID and to determine the subclass of Media Data that it belongs to.

Definition Objects

This index lists all Data Definition and Effect Definition objects in the OMFI file. Although these objects are referenced by other OMFI objects, they do not belong to any tree structure. This index is provided to improve efficiency in handling these definition objects.

Version Number

The OMFI:Version property specifies the version of OMF Interchange used to create the file. In the data type for this property the first byte indicates the major revision number. The second byte indicates the minor revision number. Any OMF Interchange file conforming to this version of the OMF Interchange specification must have a VersionNumber with the first byte having a value of 2.

Identification List

The omfi:HEAD:IdentificationList property identifies the application that created the OMF file and, optionally, other applications that have modi-

fied the OMF file. The identification list consists of an ordered set of Identification (IDNT) objects. The first Identification object describes the application that created the OMF file. Any following Identification objects describe applications that have modified the OMF file.

Mob Requirements for File Interchange

OMF Interchange files that are used for interchange purposes must include a complete set of Mobs. In other words, the file must include all the Mobs that are referenced in the file. The Media Data objects, which contain the actual media data, may be in external OMF Interchange files. Ideally, the file Source Mob provides location hints for finding external files.

From HEAD to Media—an Overview

This section provides an example of an OMF Interchange file to help you understand the overall structure of OMFI files. The example OMFI file described in this section contains the following objects:

- Header
- Composition Mob
- Master Mobs
- File Source Mobs
- Physical Source Mobs
- Media Data objects

The topics of this sections describe:

- Starting with the Header object to find the Mobs and Media Data objects
- Examining the Composition Mob and finding the Master Mob for a Source Clip
- Examining the Master Mob and then finding the corresponding file Source Mob
- Finding the Media Data object associated with the file Source Mob
- Following the links to the original Source Mobs

Starting with the Header Object

When you are importing or reading an OMF Interchange file, you first need to examine the data stored in the Header object. You use the Header object to find the Mobs and Media Data objects.

Figure 8 illustrates a Header object.

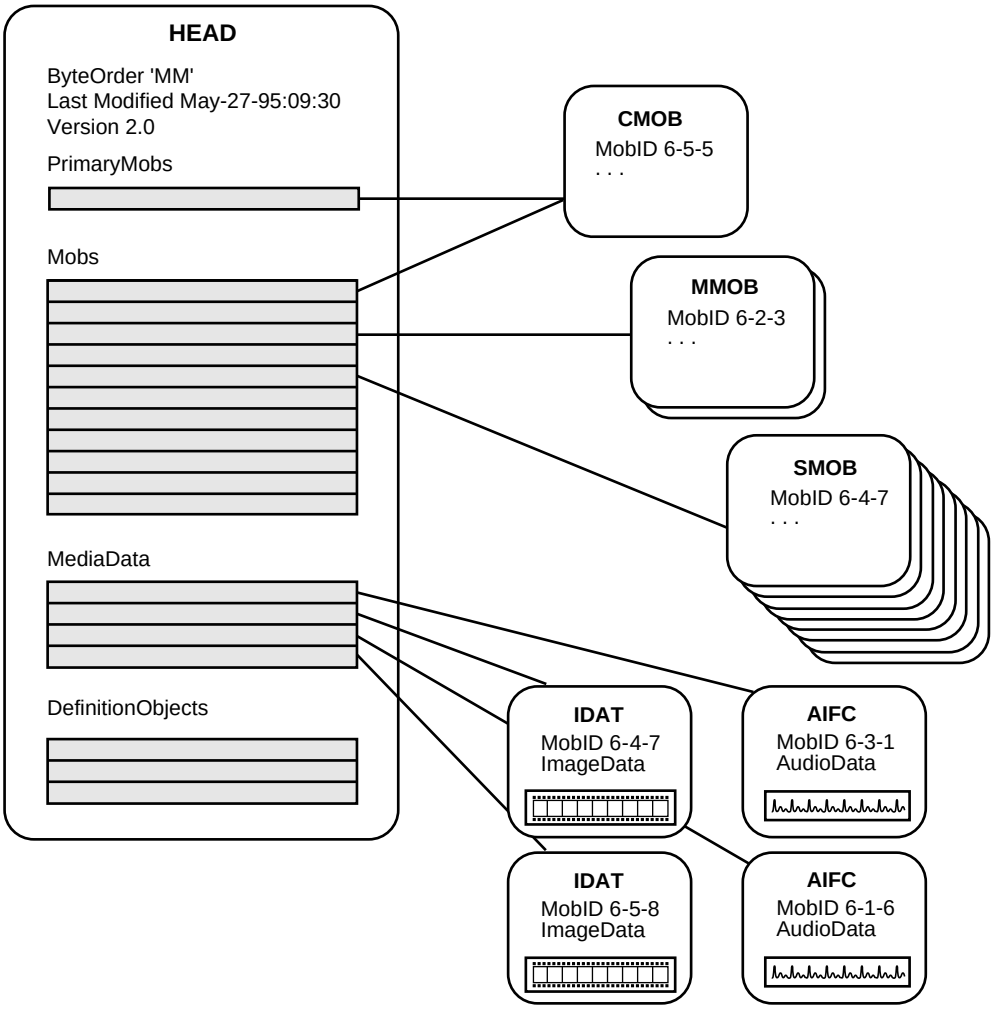


Figure 8: Header Object

The Header object has:

- A set of Mobs
- A set of Media Data objects
- References to a set of primary Mobs
- A set of OMFI Objects (OOBJ) that are Data Definition or Effect Definition objects (not shown in illustration)

MobID Match Between File Mob and Digital Data

Each Mob and Media Data object has a `MobID`. The `MobID`s, which are triplets of 32-bit integers, are displayed in the example as three integers separated by hyphens. The values of the integers in the example are not meaningful except to distinguish one `MobID` from another.

There is one Source Mob that has the same MobID as each digital Media Data object in the OMFI file. A file Source Mob is associated with the digital Media Data object that shares its MobID. For example, the Mob with the MobID 6-4-7 is the file Source Mob that corresponds to the Image Data (IDAT) object with the same MobID.

Examining the Composition Mob

By examining the structure of a Composition Mob and following the Mob links to the sources, you obtain access to all the information needed to play the Composition Mob.

Figure 9 shows the structure of the Composition Mob and shows the objects contained within it.

The Slots property in the Composition Mob has an ordered set of Mob Slot objects. These Mob Slots describe tracks of media or other time-varying information. In this example, they describe one video track and two audio tracks.

Within each Mob Slot , the Segment property has a Segment object. In this example as in most Composition Mobs, the Segment property has a Sequence Object, which contains a series of Components that are to be played sequentially. (The Sequence class is a subclass of Segment.) In the video Mob Slot in the example, there are four Source Clips in the Sequence, and in the two audio Mob Slots,there are two Source Clips in each Sequence.

In the Source Clip object, the SourceID property identifies the Master Mob that is used to find the media; the SourceTrackID property, identifies the Mob Slot within the Master Mob; and the Length and StartTime properties identify the section within the Mob Slot's Segment.

The first video Source Clip specifies the SourceID 6-2-3. This is the MobID of a Master Mob in the OMFI file.

Through the Master Mob to the File Mob

In this example, the first video Source Clip in the Composition Mob references the Master Mob that describes the media digitized from a section of a single videotape. When this section was digitized, there were three Media Data objects created: one for the video media and one for each of the two stereo audio channels. The Master Mob describes the synchronization of these three Media Data objects.

In addition to grouping these data objects, the Master Mob isolates the Composition Mob from any changes in the digitized media data. For example, you may need to change the data by redigitizing the original source media to include more data from the beginning or end of the section or to use a different compression to increase the quality of the image or to reduce the disk storage required. In order to ensure the integrity of compositions, OMF requires that you cannot reuse the same MobID for the new sampled media as for the

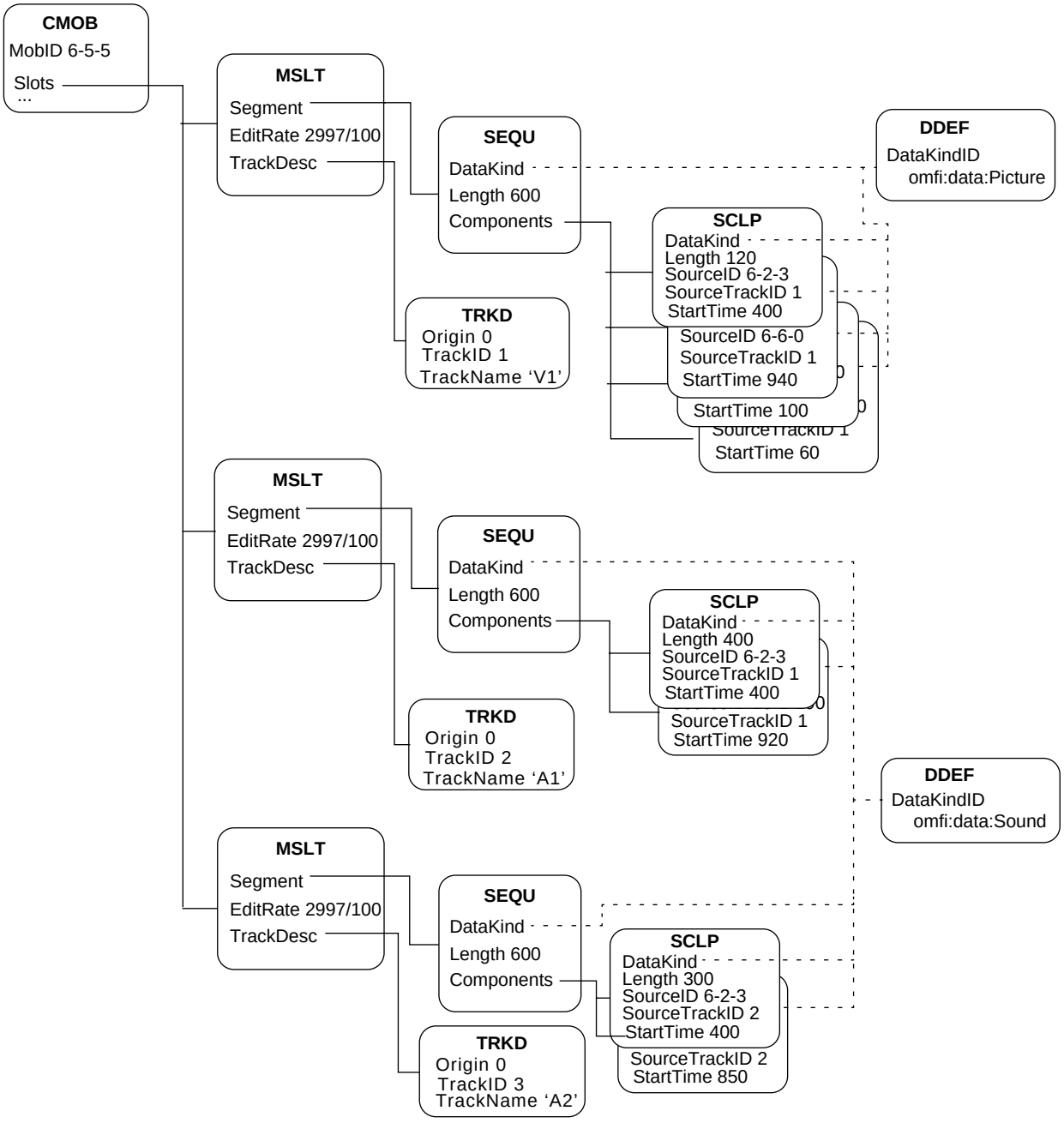


Figure 9: Composition Mob with Property Values

previous version. You must create a new file Source Mob with a new MobID. You must then update all references to the file Source Mob to the new MobID. OMF requires that the Composition Mob always access file Source Mobs through Master Mobs to minimize the number of updates required. Even if there are many Source Clips in a Composition Mob that use media from a redigitized Media Data object only the Master Mob needs to be updated.

Figure 10 shows the contents of a Master Mob from the example OMFI file.

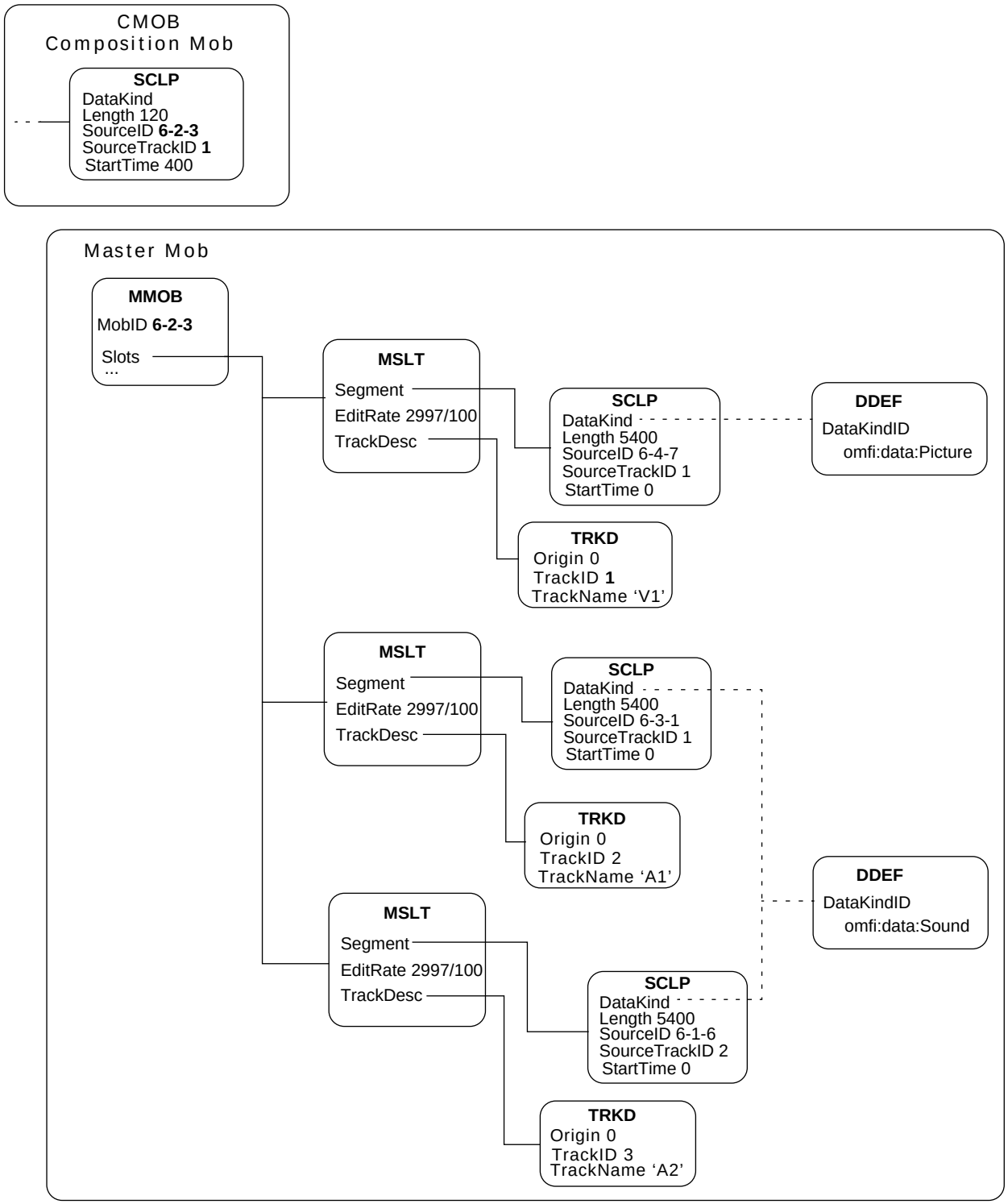


Figure 10: Source Clip to Master Mob

There are three Mob Slots in the Master Mobs, one video and two audio. The Source Clip in the Composition Mob identifies the first Mob Slot in this Master Mob because it has the SourceTrackID 1.

Each Mob Slot in the Master Mob contains a single Source Clip. Each Source Clip specifies the MobID of the file Source Mob that describes the Media Data object from the videotape. For example, the Source Clip in the first Mob Slot specifies the file Source Mob with the MobID 6-4-7.

This connection from a Source Clip in one mob to a Mob Slot in another mob is the mechanism that is used to link mobs.

File Mob to Data Object

Figure 11 shows the contents of the Source Mob and its associated Media Data object that the Source Clip in the Master Mob represents. The Source Mob has one Mob Slot, which contains one Source Clip. The Source Clip shows how long the media data is and specifies its edit rate. In addition, it identifies the Source Mob that describes the previous generation of media.

The file Source Mob MediaDescriptor property describes the format of the media stored in the Media Data object or in the raw data file. In this example, it has an RGBA Component Image Descriptor. The Media Descriptor object specifies whether the digital media data is stored in an OMFI file or a raw data file. If the IsOMFI property has a True value, it is stored in a Media Data object in an OMFI file. If the IsOMFI property has a False value, it is stored in a raw data file.

The Media Descriptor object also specifies the sample rate of the digital media data and specifies the number of samples that it contains. See Chapter 6, Describing Media, for a description of the difference between sample units and edit units.

To find the digital media data itself, you need to find the Media Data object with the same MobID as the Source Mob. If you only need to play or access the digital media data, you do not have to follow the mob links to the original sources. However, if you need to redigitize the media, then you need to find the physical Source Mobs.

Following the Links to Original Sources

The Source Clip in the file Source Mob specifies the MobID of the previous generation of media. In Figure 11, the video file Source Mob's Source Clip identifies the Source Mob for the videotape. Figure 12 illustrates the videotape Source Mob. The videotape Source Mob has four Mob Slots. These Mob Slots describe the video, two audio tracks, and timecode track on the videotape.

The video and audio Mob Slots each typically contain a single Source Clip that shows how long the media data is and specifies its edit rate. In addition, it identifies the Source Mob that describes the previous generation of media for

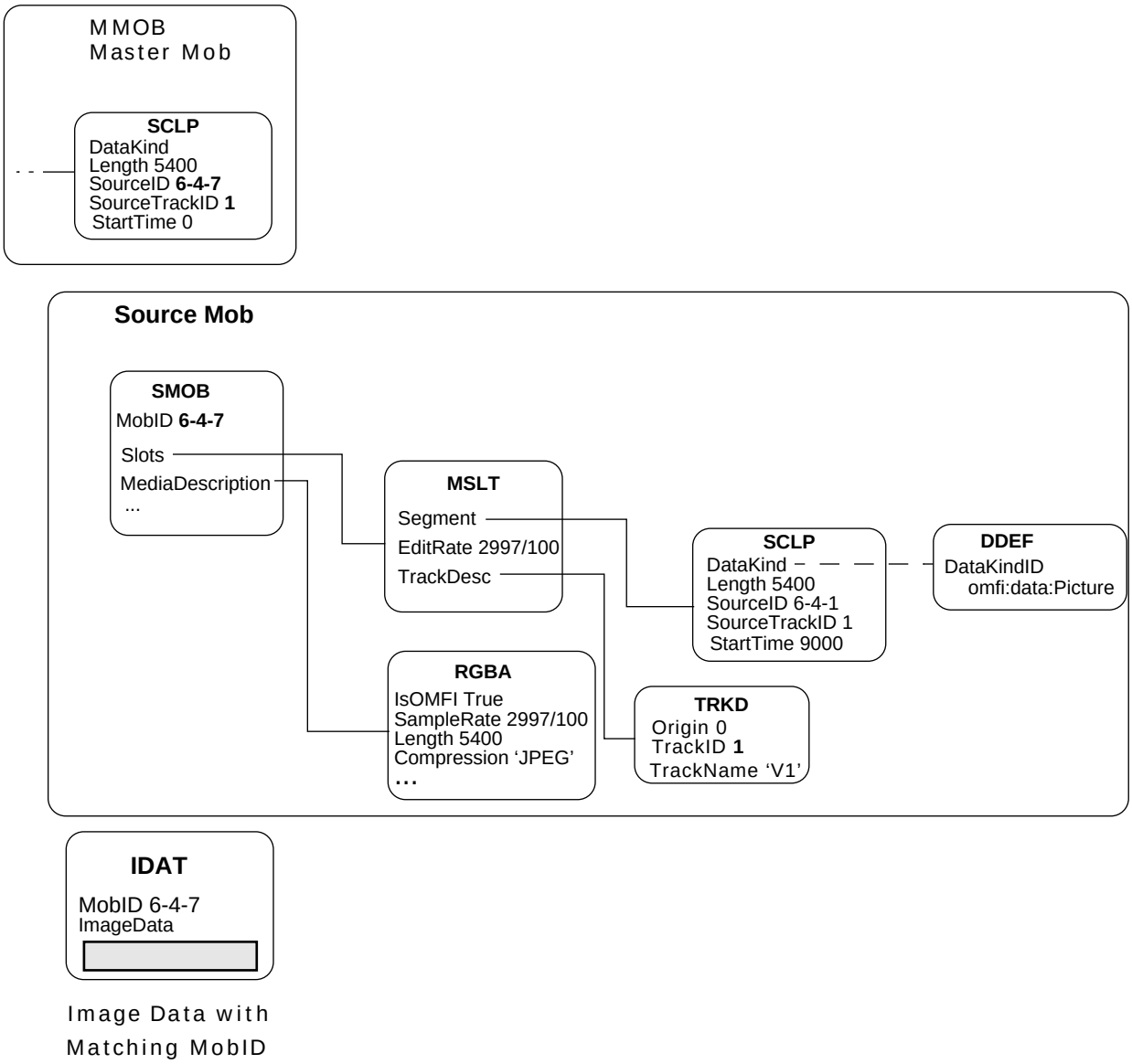


Figure 11: Master Mob Source Clip to Source Mob and Media Data

that track of media. If a video tape was transferred from more than one film or audio tape, its Mob Slots should contain a Sequence of Source Clips that identify the previous generation for each section of the videotape.

The timecode Mob Slot contains a single Time Code (TCCP).

The videotape Source Mob `MediaDescriptor` property describes the format of the media stored in the videotape.

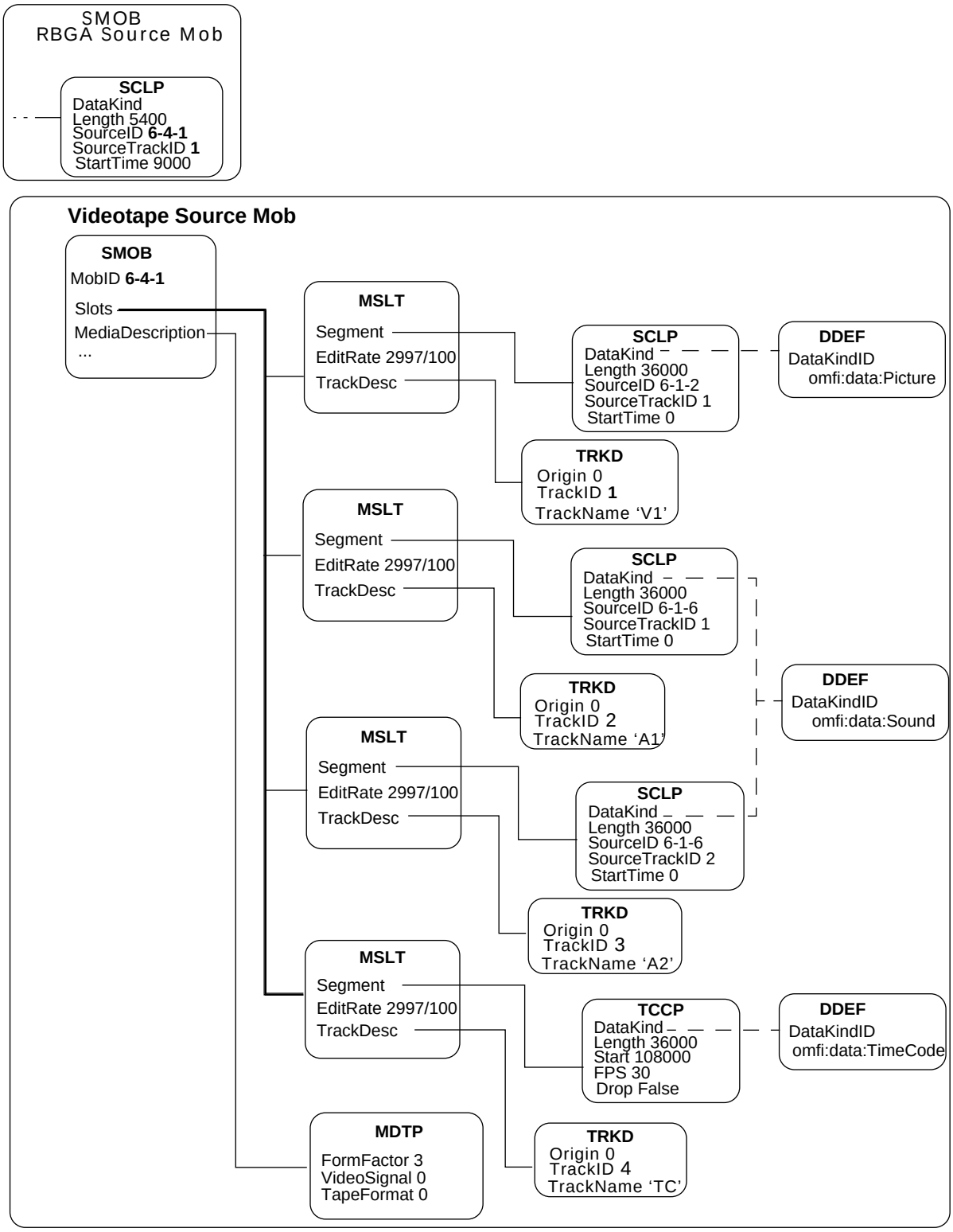


Figure 12: Source Clip to Videotape Source Mob

In videotape Source Mob, the video Source Clip identifies a film Source Mob and the two audio Source Clips identify the two Mob Slots in an audio tape Source Mob. Figure 13 illustrates the film Source Mob.

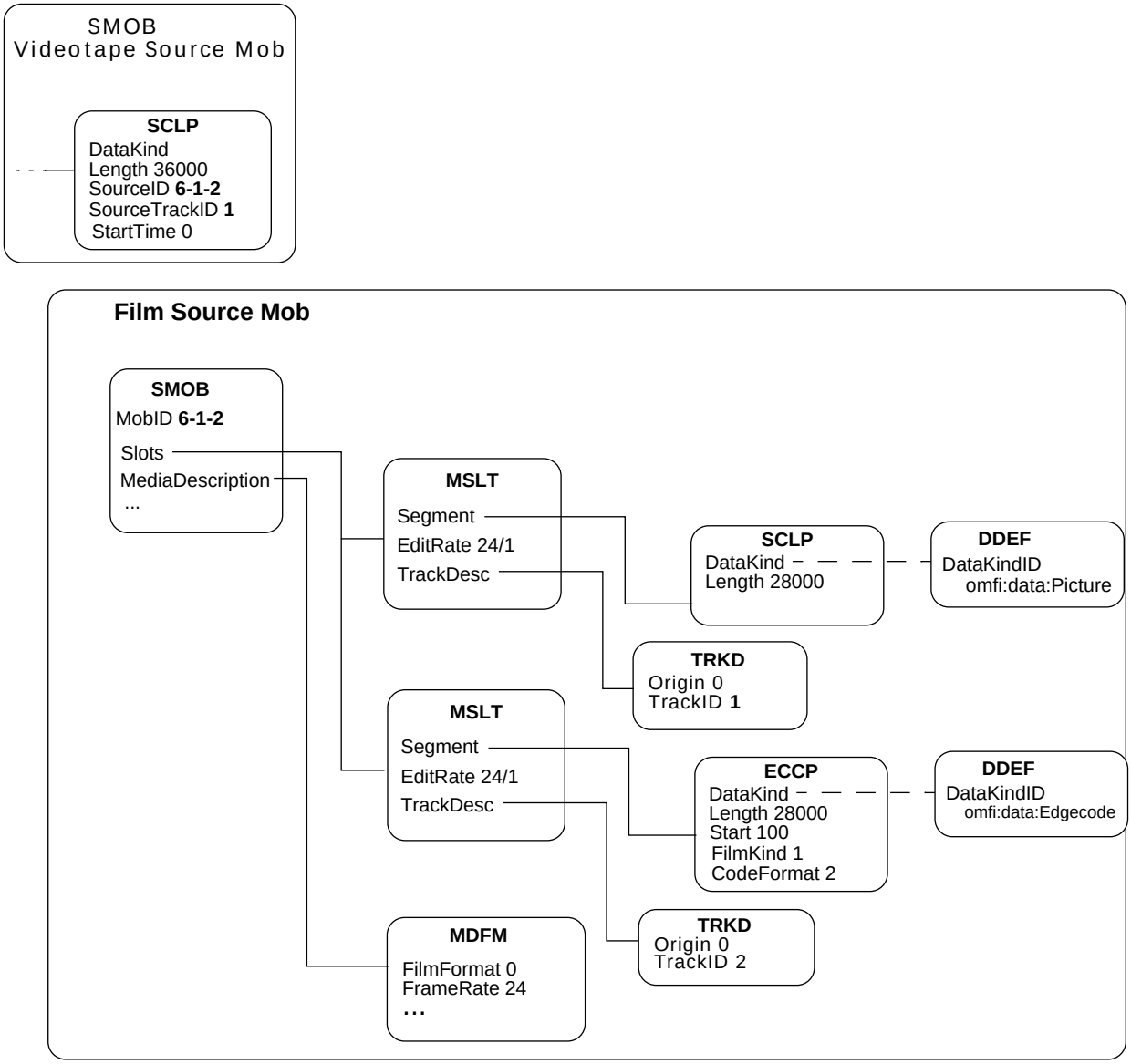


Figure 13: Source Clip to Film Source Mob

If a Source Mob describes the original media, then its Mob Slots contain Source Clips with `SourceID` properties that specify the `0-0-0 MobID`—there is no previous generation of media.

Note that if digital media data is created without a physical source, such as media data created by a graphics or animation application, then the file Source Mob is the original source and its Source Clips have a `SourceID` with a 0-0-0 value.

5 Composition Mobs

This chapter describes the OMF Composition Mob (CMOB)–the OMF object that describes editing information. Whereas the previous chapter introduced Mobs and described how the Composition Mob fits in the overall OMF Interchange file structure, this chapter focuses on the internal structure of the Composition Mob.

Composition Mob Basics

The Composition Mob describes editing information. It describes it by means of properties that have simple values and properties whose values are specified by other objects. The Composition Mob is the root of a tree structure containing these other objects. The kinds of objects in the tree, the structure that connects them, and the property values within each object describe the editing information.

Figure 14 illustrates a simple Composition Mob with three Mob Slots. Each Mob Slot in this figure represents a track of playable media and has a Sequence. Each Sequence object has a set of Source Clips. In this example, the tree structure has three main branches: the Mob Slots. The Source Clips are the leaves of the tree structure.

A Source Clip describes a section of media (or other time-varying data) and typically has a reference to a Mob Slot in a Master Mob, which identifies the media source. A Source Clip can also have a reference to a Mob Slot in another Composition Mob.

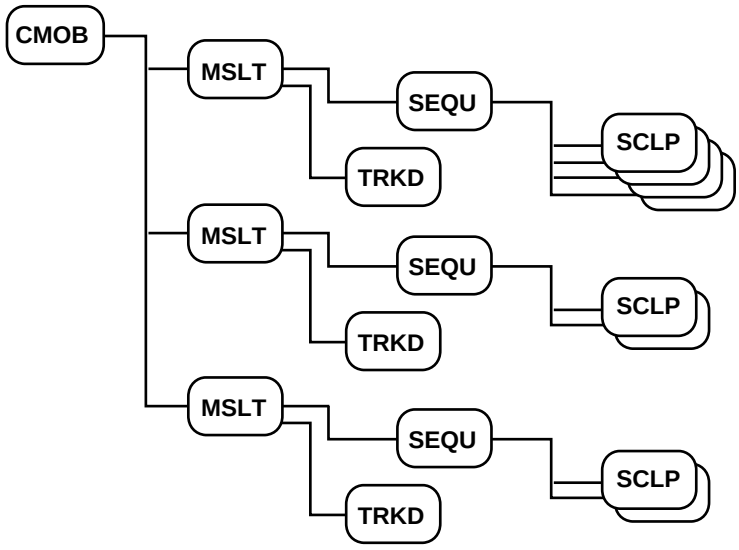


Figure 14:Composition Mob

In more complex Composition Mobs, each Mob Slot contains a Sequence that contains Transitions, Effect Invocations, Nested Scopes, Scope References, Selectors, and other embedded Sequences as well as Source Clips.

The Composition Mob structure describes how the pieces of the media production should go together. Because structures can sometimes be atypical, your application should read a Composition Mob by using the OMFI class model and class definitions, which determine what is legal in a Composition Mob. For example, although a typical Mob Slot in a Composition Mob has a Sequence, it can have any Segment, so it could contain a Source Clip, Effect Invocation, Nested Scope, or any other kind of Segment.

Mob Slots and Track Descriptions

Mob Slots contain Segments of media or other time-varying data. A Mob Slot may represent an externally accessible track or a slot that can only be referenced internally from another Mob Slot within the same Mob. By definition, a Mob Slot is an externally accessible track if and only if it has a Track Description (TRKD). The Track Description object specifies the `TrackID`, `TrackLabel`, and `Origin` of the track; these properties allow Source Clips in other Mobs to reference the Mob Slot.

A Segment is a section of time-varying data. Segment is an abstract class; its subclasses are Source Clip (SCLP), Filler (FILL), Sequence (SEQU), Effect Invocation (EFFE), Nested Scope (NEST), Scope Reference (SREF), Selector (SLCT), Timecode (TCCP), Constant Value (CVAL), Varying Value (VVAL), Edge Code (ECCP), Edit Rate Converter (ERAT), and Media Group (MGRP). These subclasses of Segment all have in common the attribute that each can stand alone. This

is in contrast with the other subclass of the Component class, Transition. Transition objects may only be used within a Sequence object and must be preceded and followed by a Segment object. The value produced by a Transition object depends on the values produced by the preceding and following Segments.

The Segment specified in a Mob Slot can be any of the following:

- Sequence—Combines a set of Components into a single segment. The Sequence object is the basic mechanism for combining sections of media to be played in a sequential manner in an OMFI file. The Components in a Sequence can be Segments or Transitions.
- Source Clip—Specifies a section of media or other time-varying data and identifies the Mob Slot in another Mob that describes the media. In a Composition Mob, Source Clip objects are used to reference the digitized media data to be played or manipulated.
- Timecode—In a Composition Mob, represents the timecode associated with the virtual media represented by the Composition Mob. If the Composition Mob is traversed in order to record the media data to a videotape, the Timecode should be used to generate the timecode for the videotape. Timecode can also be used in videotape Source Mobs.
- Filler—Specifies an unknown value for the Component's duration. Typically, a Filler is used in a Sequence to allow positioning of a Segment when not all of the preceding material has been specified. Another typical use of Filler objects is to fill time in Mob Slots and Nested Scope Segments that are not referenced or played.
- Effect Invocation—Specifies an effect to be used in a Composition Mob; specifies kind of effect, input media slots, and control arguments.
- Nested Scope—Defines a scope of slots that can reference each other; the Nested Scope object produces the values of the last slot within it. Typically, Nested Scopes are used to enable layering or to allow a component to be shared.
- Scope Reference—Refers to a section in another Mob Slot or Nested Scope slot.
- Selector—Specifies a selected Segment and preserves references to some alternative Segments that were available during the editing session. The alternative Segments can be ignored while playing a Composition Mob because they do not affect the value of the Selector object and cannot be referenced from outside of it. The alternative Segments can be presented to the user when the Composition Mob is being edited. Typically, a Selector object is used to present alternative presentations of the same content, such as alternate camera angles of the same scene.
- Edit Rate Converter—Converts part of a segment in one edit rate into a segment in another edit rate. In most cases, it is not necessary to convert from one edit rate to another. It is usually possible to express the same information in another edit rate. Two cases where you may want to use an Edit Rate Converter are when you want to make an exact copy of a Segment that is in another edit rate and when it is important to control the rounding errors caused by edit rate conversions.

- Constant Value—Provides a single value that does not vary over time and is typically used in Effect Invocations to supply control argument values.
- Varying Value—Provides values that vary over time and is typically used in Effect Invocations to supply control argument values.

In addition, there are two Segment subclasses, Media Group and Edge Code, that are not used in Composition Mobs. A Media Group object can only be in a Master Mob, and an Edge Code object can only be in a Source Mob.

The following sections describe

- Simple Composition Mobs with Sequence of Source Clips
- Composition Mobs with Transitions
- Composition Mobs with Effect Invocations
- Composition Mobs with Scope References
- Composition Mobs with Selectors, Audio Crossfades, and Edit Rate Converters

Simple Composition Mobs and Sequences

Typically, a Composition Mob (CMOB) has one or more Mob Slots (MSLT) and each Mob Slot has a Sequence (SEQU). The Sequence contains an ordered set of Components (CPNT). Within this ordered set any Transition (TRAN) object must be preceded and followed by a Segment (SEGM). Figure 15 illustrates the structure of a simple Composition Mob and shows the values of object properties. It has three Mob Slots, each of which contains a Sequence of Source Clips (SCLP).

The composition is defined by the structure of the Composition Mob, the kind of objects that appear in it, and the order in which the objects appear. The objects in the illustration have the following meaning:

- CMOB—the Composition Mob object identifies this as a composition.
- MSLT and TRKD—the Mob Slot represents an externally visible track of media if it has a Track Description.
- SEQU—the Sequence, which is a kind of Segment, specifies that the Mob Slot contains a series of Components (CPNT) that are to be played in sequential order.
- SCLP—the Source Clips represent sections of media.
- DDEF—the Data Definition objects show what kind of media each Component represents. Components within a Mob have a reference to a Data Definition but the Mob does not contain them; they can be referenced by Components in other Mobs also.

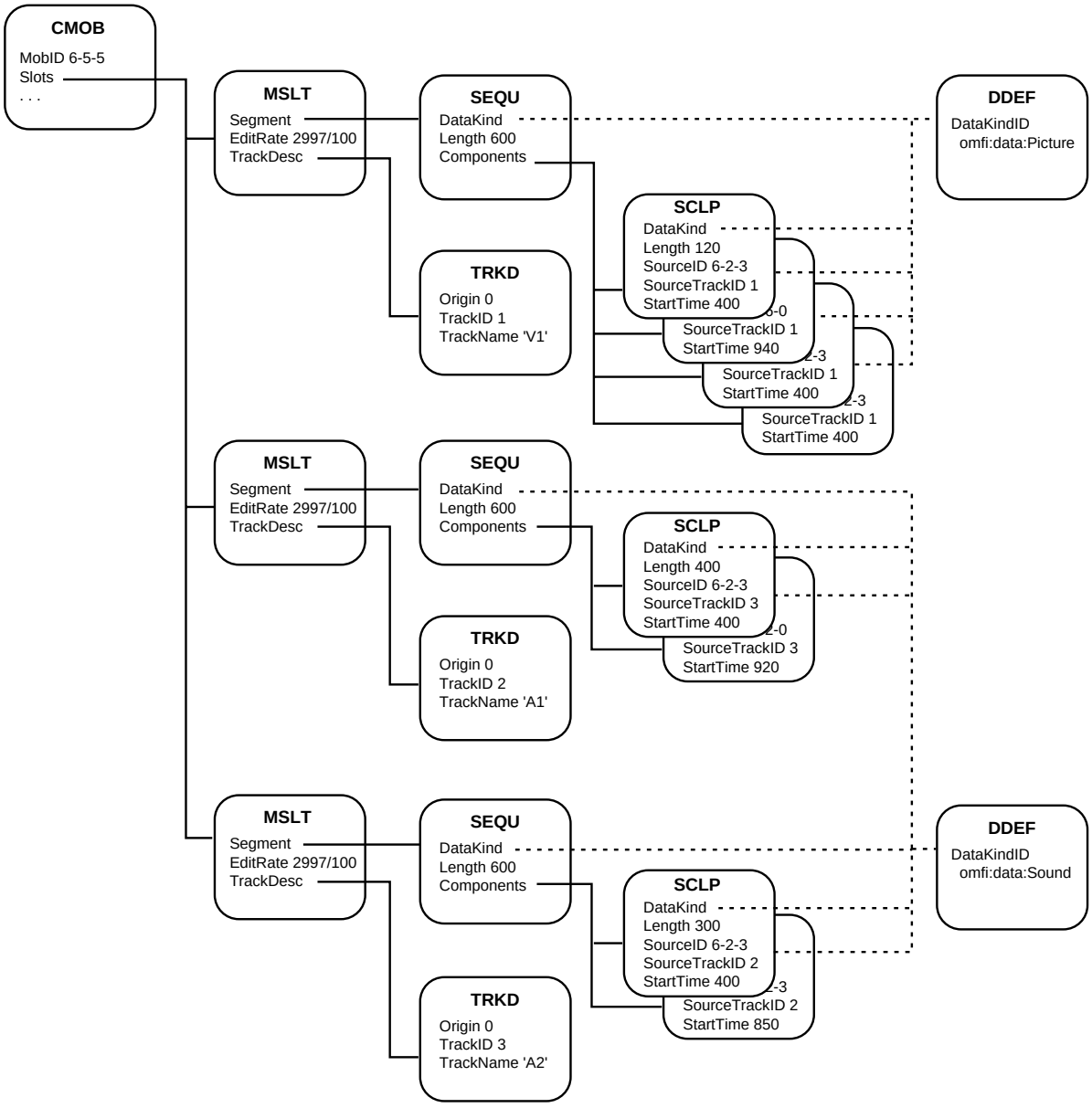


Figure 15: Composition Mob with Object Properties

Mob Slot and Component Properties

Figure 16 shows the objects in the Composition Mob and shows the properties of all the Source Clips in the video Mob Slot. Examining the property values in the objects provides the information needed to describe and play the composition. Some properties have simple values, such as a number or a string. Other properties have values that are themselves objects. When a property has a object, it is shown in the figures as a line from the property name to its object or objects. If it has a reference to an object (an object can be referenced by many

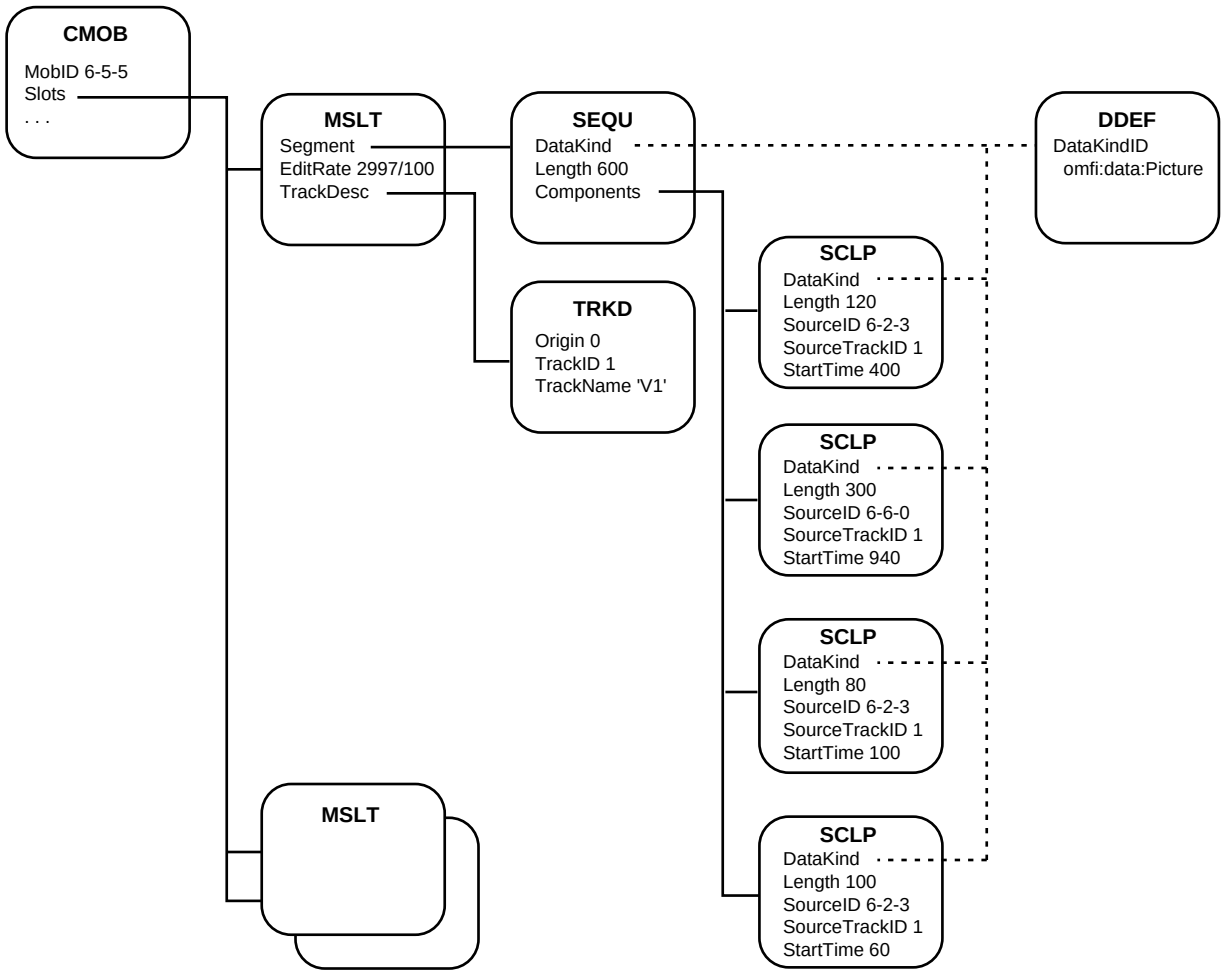


Figure 16: Composition Mob Showing Video Mob Slot

other objects), the line is a dashed line. The following list explains the meaning of the properties in Figure 16:

- The Composition Mob (CMOB) `Slots` property has a set of Mob Slots.
- The first Mob Slot (MSLT) describes the video media track.
  - The `Segment` property has a Sequence object. This property specifies the value of the Mob Slot.
  - The `EditRate` property specifies that the edit rate is 29.97 units per second. The Mob Slot `EditRate` property specifies the edit rate for all Components in the Mob Slot except for any objects within Edit Rate Converters (ERAT).
  - The `TrackDesc` property has a Track Description (TRKD) object, which identifies the Mob Slot as an externally visible track.
- The Sequence object specifies the media in the Mob Slot. It specifies the kind of media, the duration of the media, and the media itself.

— The DataKind property has a reference to a Data Definition (DDEF) object that identifies the media as omfi:data:Picture, which is video media.

— The Length property has a value of 600 edit units. Since each edit unit is approximately 1/30th of a second, the duration of the Sequence is approximately 20 seconds.

— The Components property has a set of four Source Clips.

- Each Source Clip specifies its data kind, duration, and a section of a Mob Slot in another Mob that describes the media.

  — The DataKind property of each Source Clip has a reference to the same Data Definition object that the Sequence references. They must either specify the same data kind as the Sequence or a data kind that can be converted to the Sequence's data kind.

  — The Length property of each Source Clip specifies its duration. The sum of these values equals the duration of the Sequence.

  — The SourceID property of each Source Clip specifies the MobID of a Master Mob. The first, third, and fourth Source Clips specify the MobID 6-2-3, and the second specifies 6-6-0.

  — The SourceTrackID property of each Source Clip specifies the Mob Slot within the specified Mob.

  — The StartTime property of each Source Clip specifies the offset within the Mob Slot's Segment where the section of media starts.

  The Source Clip's SourceID and SourceTrackID properties specify the Mob and the Mob Slot within the Mob; the Source Clip's Length and StartTime properties specify the section of the Mob Slot's Segment.

- The Track Description object identifies the Mob Slot as one that is intended to be externally referenced. If you are playing a Composition Mob, you can ignore the TrackID and Origin properties; they are only used if you reference the Composition Mob from a Source Clip in another Mob. A Composition Mob can be referenced by a Source Clip in either another Composition Mob or in a Source Mob.

To get the basic information about the contents of a Mob Slot, you must examine both the Mob Slot's properties and the properties of its Segment and Track Description. The properties that contain this basic information are:

- Mob Slot EditRate
- Segment DataKind
- Segment Length
- Track Description TrackID
- Track Description Origin

The EditRate, DataKind, and Length properties tell you the kind of media in the Mob Slot and its duration in absolute time. The Origin and TrackID properties tell you how the Mob Slot can be referenced from outside of the Composition Mob.

Figure 17 shows how the Sequence object structure relates to the timeline view of the Sequence. In this and other illustrations showing the structure of Mobs,

the first object in an ordered set is shown at the top and the last is at the bottom. In showing the relation between the structural view of the Sequence and the timeline view, Figure 17 shows the Source Clip objects with a horizontal dimension that is proportional to their duration.

Sequences with Transitions

Transitions (TRAN) take part of the adjacent Segments, overlap them, and combine them with an effect to produce a single Segment. A typical Transition produces a Segment that starts with media that appears to be the same as the preceding Segment and ends with media that appears the same as the following Segment. A Transition can thus appear to be a gradual change from the preceding Segment to the following Segment. However, a Transition can use an effect where the change is not gradual or linear.

The way a Transition combines these media segments depends on the effect specified by the Transition `Effect` property. Typical Transitions for image media include dissolves, wipes, and cuts. For more information on effects, see the section describing Effect Invocations on Page 74.

When playing a Transition, the last part of the preceding Segment is combined with the first part of the following Segment, effectively overlapping the two Segments. The duration of the transition specifies the amount of the overlap.

Figure 18 illustrates how a Transition object combines a part of the preceding and following media Segments and causes them to overlap.

To calculate the duration of a Sequence with Transitions, you add the durations of the Segments and then subtract the duration of the Transitions. In the example in Figure 18, the duration of the Sequence is 180 + 320 +80 +100 – 80, which equals 600.

If you are inserting a Transition between two Segments and you want to preserve the overall duration of the two Segments, you must adjust the Segments' `Length` and `StartTime` values. Table 5 compares the `Length`

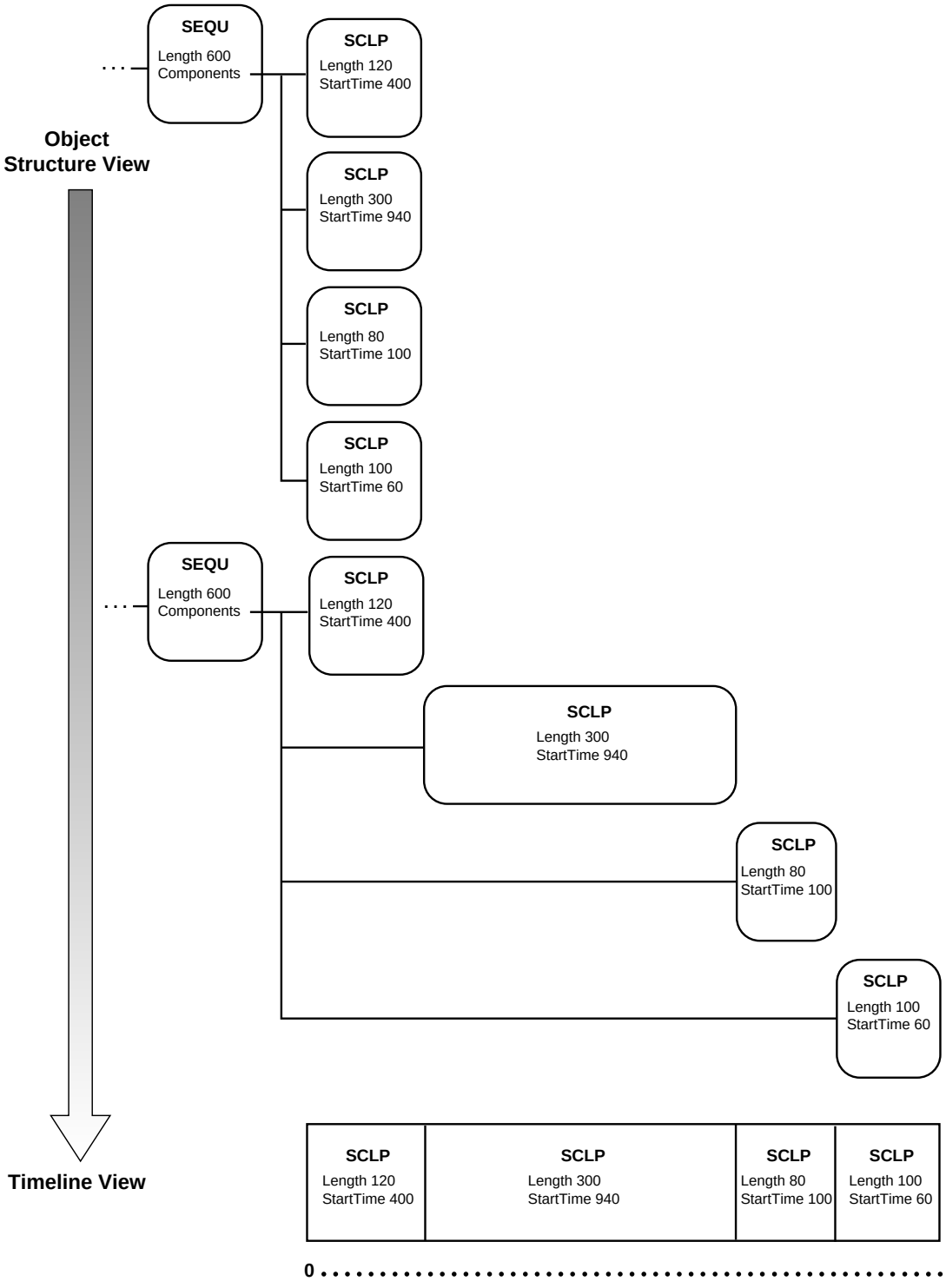


Figure 17: Sequence with Object Structure and Timeline View

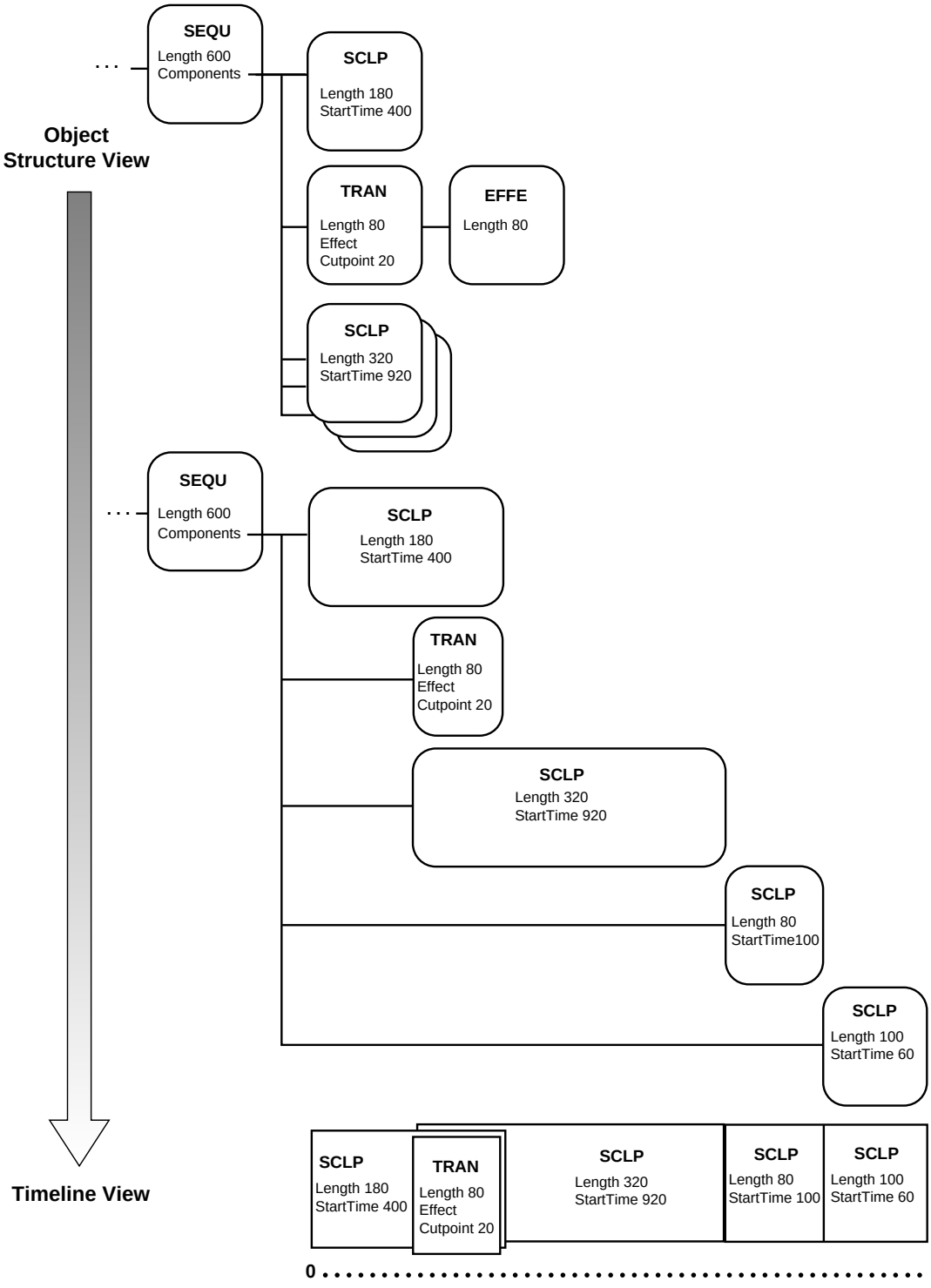


Figure 18: Sequence with Transition

and StartTime values of the first two Source Clips before inserting the Transition (Figure 17) and after inserting the Transition (Figure 18).

Table 5: Inserting a Transition and Preserving Overall Duration

	Without Transition	With Transition
First Source Clip	Length 120	Length 180
	StartTime 400	StartTime 400
Transition	—	Length 80
		CutPoint 20
Second Source Clip	Length 300	Length 320
	StartTime 940	StartTime 920

Note that the Transition has a Length of 80 and the Length of the preceding Source Clip was increased by 60 and the Length of the following Source Clip was increased by 20. The StartTime of the following Source Clip was decreased by 20. The Length of the Transition cancels the increase in the Lengths of the Source Clips because the Source Clips are overlapped.

Cuts and the Transition Cut Point

Transitions also specify a CutPoint. The CutPoint has no direct effect on the results specified by a Transition, but the CutPoint provides information that is useful if an application wishes to remove or temporarily replace the transition. The CutPoint represents the time within the Transition that the preceding Segment should end and that the following one begin if you remove the Transition and replace it with a cut. To remove a Transition and preserve the absolute time positions of both Segments, your application should trim the end of the preceding Segment by an amount equal to the Transition Length minus the CutPoint offset, and trim the beginning of the succeeding Segment by an amount equal to the CutPoint offset.

In this example transition, the CutPoint is 20 edit units. This means that to remove the Transition, you should trim the Length of the preceding Segment by 80 (Transition Length) minus 20 (CutPoint). This returns it to its original Length of 120. You should also need to trim the Length of the following Segment by 20 (CutPoint) and increase its StartTime by the same amount. This returns this Source Clip to its original Length of 300 and its original StartTime of 940.

Treating Transitions As Cuts

If you cannot play a Transition's effect, you should treat it as a cut. Treating is as a cut means that you should play the two Segments surrounding the transition as if they had been trimmed as described in the preceding paragraphs. If you play the two Segments without trimming, the total elapsed time for

them will be greater than it should be, which can cause synchronization problems.

Restriction on Overlapping Transitions

Transitions can occur only between two Segments. In addition, the Segment that precedes the Transition and the Segment that follows the Transition must each have a `Length` that is greater than or equal to the `Length` of the Transition. If a Segment has a Transition before it and after it, the Segment's `Length` must be greater than or equal to the sum of the `Length` of each of the two Transitions. This ensures that Transitions do not overlap. These restrictions allow applications to treat Transitions in a uniform manner and avoid ambiguous constructions.

It is possible to create Sequences that *appear* to start or end with Transitions or that *appear* to contain overlapping Transitions. To create the appearance of a Transition at the beginning of a Sequence, precede the Transition with a Filler object that has the same length as the Transition. To create the appearance of a Transition at the end of a Sequence, follow the Transition with a Filler object that has the same length as the Transition.

To create the appearance of overlapping Transitions, you nest the Transitions by using a Sequence within another Sequence. You can put two Segments separated by a Transition in the inner Sequence. Then you can use this Sequence object as the Segment before or after another Transition. The Transitions will appear to be overlapping.

Effect Invocations

An Effect Invocation (EFFE) combines or alters the appearance media from one or more input Segments and produces a single Segment of media. An example of an effect that combine different Segments of media to produce a single Segment is the picture-in-picture video effect, which inserts one image within another. An example that takes a single Segment of media as input and modifies it is the image flop effect, which reverses the left and right side of an image. Figure 19 illustrates these two effects.

The Effect Invocation identifies the kind of effect by having a reference to an Effect Definition object that specifies a global unique name for the effect. The global unique name is associated with a registered or private effect. In order for your application to process the effect, it must recognize the global unique name and have code to handle the recognized effect. If it does not recognize the effect, it can perform some operations on the Effect Invocation, but it cannot perform the operations required to generate the effect.

The description of the effect and its expected input are described in the effect's documentation. The OMF Developers' Desk provides this documention for

A-track video image

B-track Video Image

Picture-in-picture combined image

A-track video image

Image modified by flop effect

Figure 19: Examples of Effects

registered effects. The OMF Developers' Desk can help you find documentation on private effects.

An Effect Invocation typically has one or more Segments that provide the media to be combined or altered.

In addition to the input media, an effect can also have control arguments that determine how the effect is to be applied. These control arguments can specify an effect's location, size, transparency, and other parameters.

The Effect Invocation object specifies the input media Segments and control arguments with Effect Slot (ESLT) objects. Each Effect Slot identifies its purpose with an argument ID integer value. The effect's documentation describes the meaning of each integer value used for an effect.

An effect can have control arguments whose values vary over time. For example, a picture-in-picture effect where the size and transparency of the inserted picture stays constant throughout the effect has constant control arguments. In contrast, a picture-in-picture effect that starts with a small inserted picture that grows larger during the effect has control arguments with time-varying values.

A constant control argument can be specified with a Constant Value (CVAL) object in an Effect Slot. A time-varying value is specified with a Varying Value (VVAL) object in an Effect Slot. An Effect Slot can also contain a Sequence of Constant Value and Varying Value objects.

When an effect specifies time-varying control values, it can also specify how to interpolate these values. Interpolation is a mathematical method to calculate a value for a point based on the values of two surrounding points.

The documentation for an effect can describe how the effect is treated in a Transition. Typically, the effect uses the media from the two overlapping Segments and sets the default for a level control argument to specify a smooth transition from the preceding Segment to the following one.

An Effect Invocation can included rendered versions of the intended effect. These are Source Clips that identify digital media data that can be played instead of generating the effect from the input media. If an application does not recognize the effect, it should use the rendered media. It can also use the rendered media rather than generating the effect for reasons of efficiency.

The following sections describe:

- Example Effect Invocation in a Sequence
- Example Effect Invocation in a Transition
- Effect Invocations with Rendered Media
- Varying Value Objects in Effect Invocations

Example Effect Invocation in a Sequence

An Effect Invocation can be used anywhere in a Composition Mob that a Segment is allowed. Figure 20 illustrates an Effect Invocation as one of the components in a Sequence.

The Effect Invocation specifies the data kind of the Segment produced by the Effect Invocation, the duration of the Segment, the effect that is to be applied, and the media and control arguments.

The following explains the objects in Figure 20.

- EFFE—the Effect Invocation specifies that an effect is used in the Sequence. Its properties specify:
  - `DataKind` specifies the kind of media produced by the Effect Invocation. In this example, it specifies `omfi:data:Picture`, the data kind for video media.
  - `Length` specifies the duration of the Effect Invocation.
  - `EffectKind` specifies the Effect Definition (EDEF) object that specifies the effect to be used when combining or altering the media.
  - `EffectSlots` specify the media and control argument Effect Slots (ESLT).
- EDEF—the Effect Definition object specifies the `omfi:effect:VideoDissolve` effect.
- ESLT—the Effect Slots contain the input media segments and the control arguments. Its properties specify:
  - `ArgID` specifies the purpose of the Effect Slot. The `ArgID` property has an integer value, whose meaning is defined in the documentation

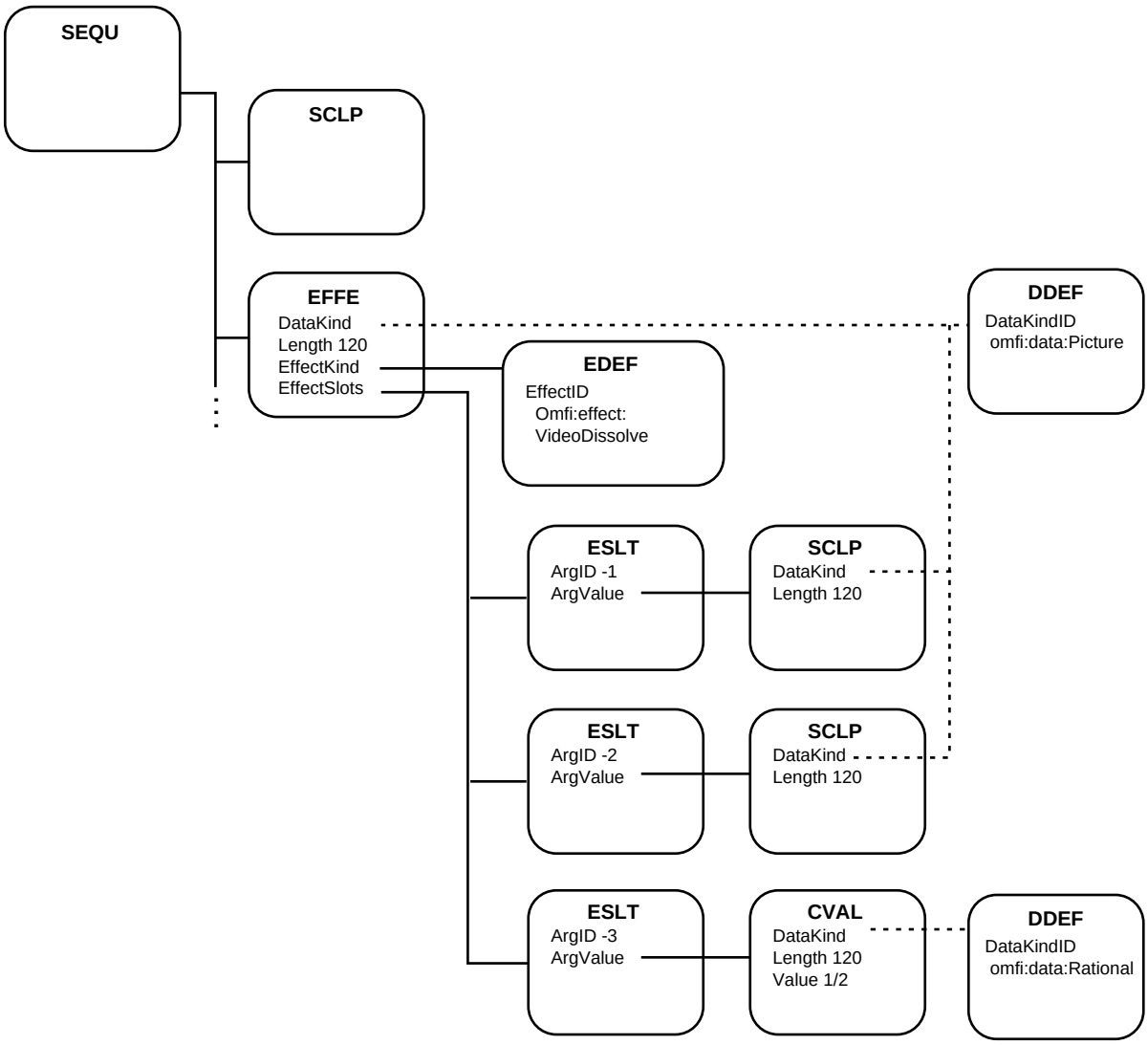


Figure 20: Video Dissolve Effect in Sequence

for the `omfi:effect:VideoDissolve` effect. Table 6 contains a summary of the `ArgID` description from this documentation.

Table 6: ArgID Values for Video Dissolve

`ArgID` Value	Meaning	Data Kind
`-1`	Specifies the A Track of video media, the background.	`omfi:data: PictureWithMatte`
`-2`	Specifies the B Track of video media that will be superimposed on the background.	`omfi:data PictureWithMatte`
`-3`	Specifies the level of superimposition of the images. If 0, then the image is 100% from A Track. If 1/2, then the image is 50% from A Track and B Track. If 1, then the image is 100% from B Track.	`omfi:data:Rational`

  - — `ArgValue` specifies a Segment that provides the data for the slot. In this example, the Effect Slots that specify input media (Effect Slots -1 and -2) have Source Clips that describe the media and the Effect Slot that specifies a control argument (Effect Slot -3) has a Constant Value (CVAL) object.
- SCLP—the Source Clip specifies the input media. The duration of the Source Clips must equal the duration of the Effect Invocation.
- CVAL—the Constant Value object specifies a single value for the Effect Invocation. Its properties specify:
  - — `DataKind` specifies that the value has a data kind `omfi:data:Rational`. This data kind must match or be convertible to the data kind specified in the effect's documentation.
  - — `Length` specifies the duration during which the constant value is defined. The duration of the Constant Value object must equal the duration of the Effect Invocation.
  - — `Value` specifies 1/2, which means that the resulting image is an equal mix of the images in A Track and in B Track.
- DDEF—the Data Definition objects specify the unique name that identifies the data kind of the Components.

Example Effect Invocation in a Transition

A Effect Invocation specifies the effect to perform when changing from the preceding media Segment to the following one in a Transition. Figure 21 illustrates using a Effect Invocation in a Transition.

The effect documentation specifies whether an effect is allowed in a Transition and how the Effect Slots should be treated. Typically, an effect that is allowed in a Transition will specify that the Effect Slot with the `ArgID` values -1 and -2

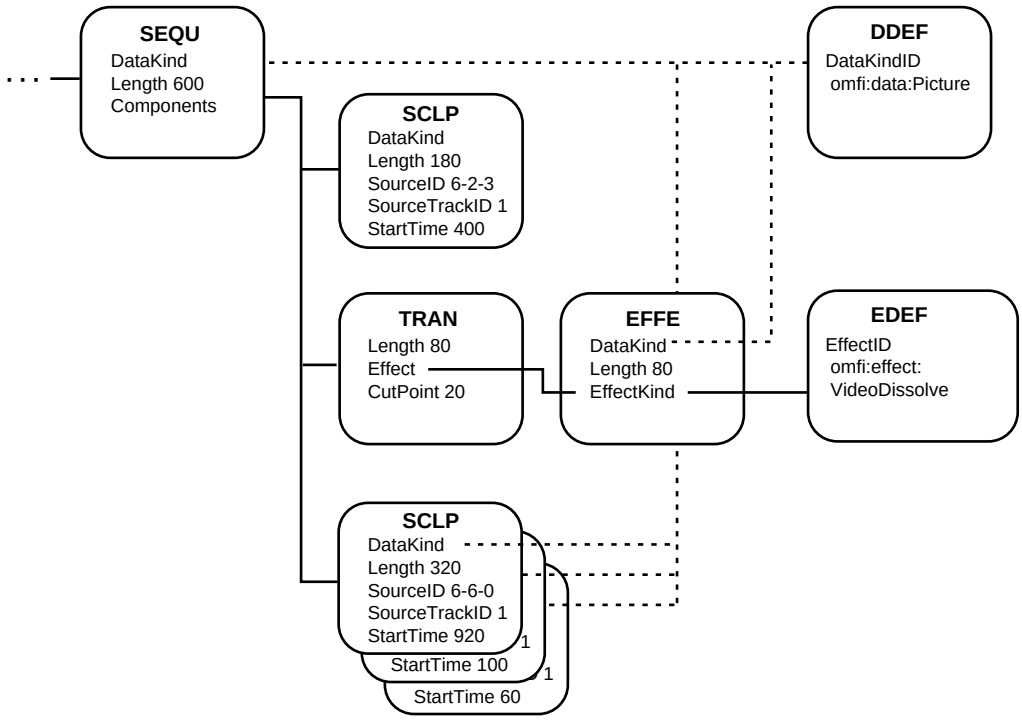


Figure 21: Video Dissolve Effect in Transition

should correspond to the overlapping sections from the preceding and following Segments. Typically, the default value for the Effect Slot with the `ArgID` value -3 varies from 0.0 to 1.0, but this Effect Slot can be explicitly overriden by specifying it in the Effect Invocation.

Table 7 describes the typical values for these Effect Slots when the Effect Invocation object is in a Transition object.

Table 7: Typical Value for Effect Slots in Transitions

ArgID Value	Typical Use in Effect	Value
-1	A Track	Last section of preceding segment; length of the section is equal to the Transition length.
-2	B Track	First section of following segment; length of the section is equal to Transition length.
-3	Level	Default value is a Rational value that varies from 0.0 at the beginning of the transition to 1.0 at the end of the transition and has a linear interpolation between. This is equivalent to a Varying Value object that has a length equal to the transition's length and linear interpolation and that contains two Control Point objects; the first specifies a time of 0.0 and a value of 0.0 and the second specifies a time of 1.0 and a value of 1.0.

In Figure 21, the Effect Invocation (EFFE) omits the `EffectSlots` property. Effect Invocations in Transitions only specify Effect Slots to override the default value for the Effect Slot with the `ArgID` value -3 and when the effect's documentation specifies additional `ArgID` values.

Rendered Effects

Sometimes it is desirable to compute the results of Effect Invocations once and store them. When the Effect Invocation is being played or accessed later, the results can be retrieved quickly and repeatedly without having to perform complex calculations.

A rendered version is digital media data that can be played to produce the effect. An application can produce a working rendering, a final rendering, or both. A working rendering is intended to store a partially rendered or approximately rendered implementation of the effect but is *not* intended for final production. A final rendering of the effect *is* intended for final production. Either a working rendering or a final rendering can be used to play the effect during editing.

Typically, an Effect Invocation is rendered and the result is stored in a Media Data object with an associated file Source Mob and Master Mob. The Effect Invocation `FinalRendering` or `WorkingRendering` property has a Source Clip that refers to the Master Mob. If there is more than one implementation of a rendering, the Master Mob could contain a Media Group (MGRP) object.

Dealing with Unrecognized Effects

If an application importing an OMFI file encounters an unknown effect, it can preserve the Effect Invocation and Effect Definition objects so that if the Composition Mob containing the effect is exported, the effect will be preserved. If any application cannot preserve the information, the OMF Developers' Desk recommends that the application inform the user that some effect information is being lost.

If an application is trying to play an unknown effect, the OMF Developers' Desk recommends that it perform the first action in the following list that applies:

1. If there is a rendered version of the effect, play the rendered version.
2. If the Effect Invocation specifies a `BypassOverride` property, play the media in the Effect Slot with the specified argument ID.
3. If the Effect Definition specifies a `Bypass` property, play the media in the Effect Slot with the specified argument ID.
4. Play blank media for the effect.

Varying Value Control Arguments

A Varying Value object is a Segment that represents time-varying values that are determined by an ordered set of Control Points. Each Control Point specifies the value for a specific time point within the Segment. The values for time points between two Control Points are calculated by interpolation.

Typically, Varying Value objects are used in Effect Slots to specify the value of a control argument for the Effect Invocation, but Varying Value objects can be used in any context where a Segment is allowed and the data kinds are compatible.

Figure 22 illustrates an Effect Invocation in a Sequence that has Varying Value control arguments.

Control Points

A Control Point that has a `Time` value equal to 0.0 represents the time at the beginning of the Varying Value object; one with a `Time` equal to 1.0 represents the time at the end of the Varying Value object. Control Points with `Time` values less than 0.0 and greater than 1.0 are meaningful but are only used to establish the interpolated values within the Varying Value object—they do not extend the duration of the Varying Value object.

Since time is expressed as a rational value, any arbitrary time can be specified—the specified time point does not need to correspond to the starting point of an edit unit of the Segment.

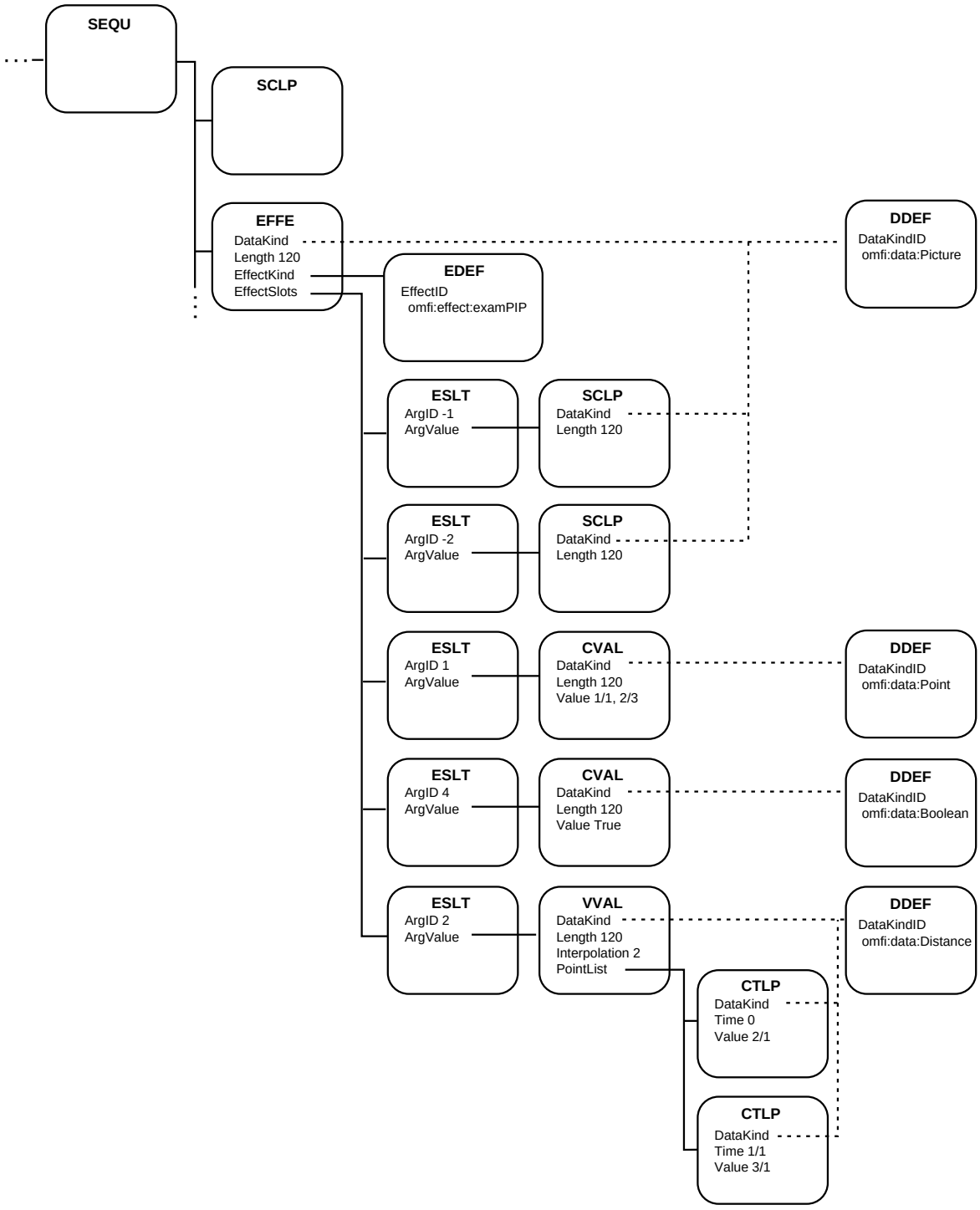


Figure 22: Picture-in-Picture Effect in Sequence

If two Control Point objects specify the same time value, the first Control Point is only used to interpolate for time points before this value. The second Control Point determines the value for the time point specified and is also used to interpolate values after this time point. It is not legal for more than two Control Points to specify the same time value.

Interpolating Control Points

For some control arguments, it is very important to be able to specify exactly how the control argument varies. For example, minor changes in the value can have a significant impact on the visual appearance of an effect that moves an insert in a spiral into the center of the image.

You can specify how the control varies by specifying many Control Points, but this can be hard to maintain when an Effect Invocation is changed. An alternative is to specify the interpolation method that governs how the control is calculated between Control Points.

An interpolation method is a mathematical formula for determining a value for a variable at any point based on values specified for points that surround that point. OMF Interchange Version 2.1 includes two interpolation methods: linear and constant. A linear interpolation means that a control argument varies in a straight line between the two specified values. A constant interpolation means that a control argument maintains a constant value until a new value is specified.

The following equation specifies the value at time X by using a linear interpolation and the values specified for time A and time B.

$$Value_X = \frac{(Time_X - Time_A)}{(Time_B - Time_A)} \times (Value_B - Value_A) + Value_A$$

Extrapolation of Control Values

Extrapolation is a mathematical method to calculate a value for a point that has values defined only on one side. Extrapolation is used if values are not specified for the start or end of a Varying Value object. If the first Control Point in a Varying Value object specifies a time value greater than 0, this value is extrapolated to the 0 time point by holding the value constant. If the last Control Point in a Varying Value object specifies a time value less than 1.0, this value is extrapolated to the 1.0 time point by holding the value constant. This holding extrapolation method is used if the interpolation method specified for the Varying Value object is constant or linear interpolation.

Sequence of Varying Value Objects

If you need to specify values for an Effect Slot using more than one kind of interpolation, you must use a Sequence object that contains a series of Varying Value objects in the Effect Slot. Each Varying Value object can have its own in-

terpolation method. Each Varying Value object defines the control argument values for its section of the Sequence. A time value of 0 specifies the beginning of the Varying Value object, which, if it is in a Sequence, may not correspond to the beginning of the Effect Slot. A time value of 1 specifies the end of the Varying Value object, which, if it occurs in a Sequence, may not always correspond to the end of the Effect Slot.

If an Effect Slot has a Sequence the value to be used at the time point where one Segment ends and another begins is specified by the following Segment.

Quantization Adjustments

The Varying Value object specifies a value for each time point within the Varying Value object; however, if you are generating a stream of media from the Composition Mob containing the Varying Value object, it can be important to adjust values produced by the Varying Value object based on sample-rate quantization. Within a media sample unit, there can only be a single value used from the Varying Value object when generating that sample.

When the number of samples is large (when quantization error is not noticeable) it is usually sufficient to express a curve in a sample-rate independent form, which is converted to the appropriate sampled values when needed. However, there are often times when the desired sample rate is low enough that some precise control over how the curve gets sampled is needed for the right result. In particular, this occurs at video sample rates.

An example using a dissolve can illustrate the quantization problem. It is natural to think of a dissolve as a mixture between video stream A and video stream B, where the mix is controlled by a level parameter that goes from 0 to 1 over the duration of the dissolve. However, since the frame at any particular time freezes the value of level that was specified at the beginning of the frame, the first frame of the dissolve will have a value of 0, and the last frame of the dissolve will be slightly less than 1. This result is incorrect, because the resulting frame sequence has a value of level which is asymmetrical. Changing the definition so that the middle of a frame is sampled instead of the beginning does not solve the problem; instead, it just transforms it into a case where the level change on the first and last frame of the dissolve is half of that for all the other frames. This is not correct because it is not uniform.

This error is due to quantization and becomes vanishingly small as the sample rate increases. But because it is sample-rate dependent, it is not something that can be accounted for in a sample-rate independent way simply by adjusting the Time values of the Control Points. Instead, it is up to the software that implements a particular effect to adjust the Control Point mapping for the actual sample rate at the time of rendering. This mapping adjustment is done by scaling the curve represented by the VaryingValue so that the 0 point is moved back by one sample time before interpolation and quantization is performed.

The following formula scales a Control Point's Time value from its stored number in edit units to its actual number in sample units, relative to the beginning of the VaryingValue component.

$$SampleTime = \left(\left(\left(\left(\frac{Length \times SampleRate}{EditRate}\right) + 1\right) \times ControlPointTime\right) - 1\right)$$

This algorithm makes the level 0 sample be the sample before the Effect Invocation starts and the level 1 sample be the sample after the Effect Invocation ends. For most effects, this is the desired results. However, some effects, such as fade-to-black or fade-from-color may need to modify the algorithm so that the level 0 or level 1 sample is included within the Effect Invocation. The effect documentation must specify a modified scaling algorithm if it should be used for the effect.

Control Point Editing Hints

The Control Point objects can specify editing hints that help applications decide how to handle edits to the Effect Invocation. These editing hints do not have an impact on how the Effect Invocation should be generated; they only provide hints that can help preserve the original intent of the Control Points when the Effect Invocation is edited and its length or starting or ending position is changed. Table 8 lists the editing hints that you can specify:

Table 8: Editing Hints

Hint	Meaning
CC_HINT_PROPORTIONAL	Keep point in time proportional to effect length (default).
CC_HINT_RELATIVE_LEFT	Keep point in time relative to beginning of effect.
CC_HINT_RELATIVE_RIGHT	Keep point in time relative to ending of effect.
CC_HINT_FIXED	Keep point in time fixed to current position.

Scope and References

Scope Reference (SREF) objects allow you to reference from within one slot the values produced by another slot. A Scope Reference can refer to a Mob Slot in the Composition Mob or it can refer to a Segment in a Nested Scope (NEST) `Slots` property. Both Composition Mobs and Nested Scope objects define a scope and an ordered set of slots. You can put one Nested Scope object within another. Within any Nested Scope object you can reference the slots within its scope and the slots within any Nested Scope object that contains it. You cannot reference a slot in a Nested Scope object from outside of the Nested Scope object. The following sections describe:

- Why to use Scope References

- How to specify a Scope Reference
- How to choose between Mob scope and Nested Scope

Why Use Scope References

Two reasons to use Scope References are:

- To layer sections of media that overlap
- To share the values produced by a slot in different contexts

Although you can layer overlapping sections of media without using Scope References, you lose some information that makes it harder for the user to make changes. For example, consider the following sequence of shots that a user wants to appear in a production:

1. An long shot of a Mediterranean island, with waves breaking on the shore
2. A title superimposed on the island
3. A shot of the star inserted in a picture-in-picture effect over the island shot
4. Ending with the island shot

You could get this sequence of shots without using Scope References by creating the following Sequence:

1. Source Clip for the island shot
2. Effect Invocation for title effect
3. Effect Invocation for picture-in-picture effect
4. Another Source Clip for the island shot

Within each of the Effect Invocations, you would specify one of the Effect Slots to have a Source Clip of the island shot. The problem with this way of implementing the Sequence is that there are four Source Clips that refer to adjacent sections of the same scene with no linkage indicated in the OMFI file. If you change the length of one of the Source Clips or Effect Invocations, you need to change the other Segments in the Sequence to ensure continuity.

To implement this sequence of shots using Scope Reference, you would have:

- The first slot contain a Source Clip for the entire island shot
- The second slot contain the following Sequence:
    1. Scope Reference to the first slot
    2. Effect Invocation for title effect including a Scope Reference to the first slot
    3. Effect Invocation for picture-in-picture effect including a Scope Reference to the first slot
    4. Scope Reference to the first slot

The length of any of the Segments in the second slot can be changed without losing the continuity of the background island scene. The user can also easily replace the background island scene and retain the edits in the second slot.

Another reason to use Scope References is to share the values produced by one slot in different contexts. An example of this is an effect that produces a rotating cube where each side of the cube shows the Segment from a different Effect Slot. If you want some of the sides to show the same Segment, you can use Scope References and put the desired Segment in another slot.

How to Specify Scope References

The Mob defines a scope consisting of the ordered set of Mob Slots. A Scope Reference object in a Mob Slot can specify any Mob Slot that precedes it within the ordered set. Nested Scope objects define scopes that are limited to the Components contained within the Nested Scope object's slots. A Scope Reference is specified with a relative scope and a relative slot.

Relative scope is specified as an unsigned integer. It specifies the number of Nested Scopes that you must pass through to find the referenced scope. A value of 0 specifies the current scope, which is the innermost Nested Scope object that contains the Scope Reference or the Mob scope if no Nested Scope object contains it. A relative scope value of 1 specifies that you must past through the Nested Scope object containing the Scope Reference to find the Nested Scope or Mob scope that contains it.

Relative slot is specified as a positive integer. It specifies the number of preceding slots that you must pass to find the referenced slot within the specified relative scope. A value of 1 specifies the immediately previous slot. Figure 23 illustrates Scope References. The slots are shown in a timeline view.

The first Scope Reference has a `RelativeScope` 2 and a `RelativeSlot` 2. This Scope Reference passes through 2 scopes: the current Nested Scope and the Nested Scope that contains it. By passing through two scopes, it refers to the scope defined by the Mob. Within that scope, it appears in slot 3. Since `RelativeSlot` has a value 2, it passes 2 slots to refer to a section in slot 1.

The second Scope Reference has a `RelativeScope` 0 and a `RelativeSlot` 3. Since `RelativeScope` is 0, it refers to a slot within its own scope. It appears in slot 4 and passes 3 slots to refer to a section of slot 1.

A Scope Reference object returns the same time-varying values as the corresponding section of the slot that it references. The corresponding section is the one that occupies the same time period as the Scope Reference.

If a Scope Reference specifies a Mob Slot, the corresponding section of the slot is the time span that has the equivalent starting position from the beginning of the Mob Slot and the equivalent length as the Scope Reference object has within its Mob Slot. If the specified Mob Slot has a different edit rate from the Mob Slot containing the Scope Reference, the starting position and duration are converted to the specified Mob Slot's edit units to find the corresponding section.

If a Scope Reference specifies a Nested Scope slot, the corresponding section of the slot is the one that has the same starting position offset from the begin-

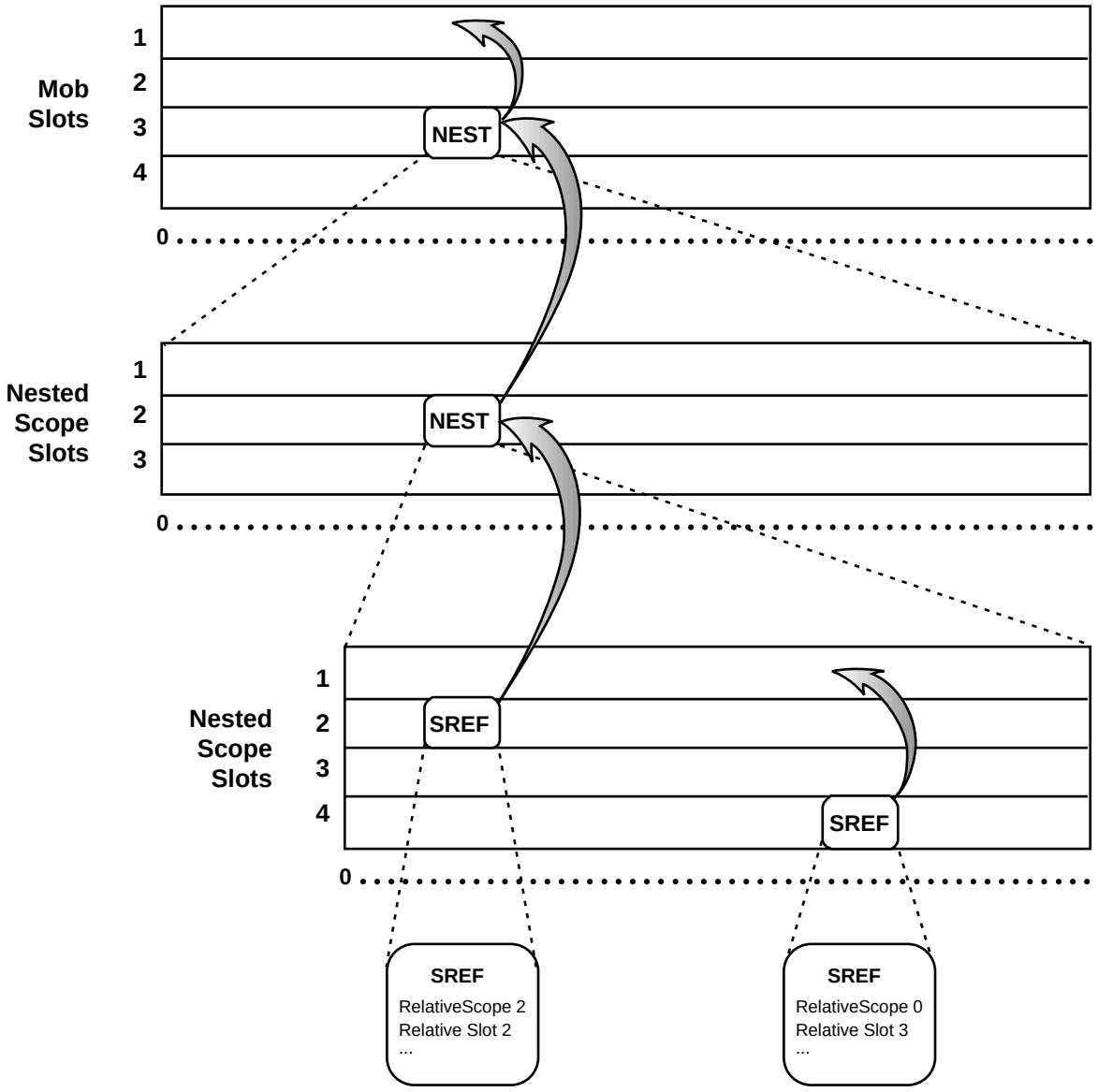


Figure 23: Relative Scope and Slot

ning of the Nested Scope segments and the same duration as the Scope Reference object has in the specified scope.

Mob Scope and Nested Scope

The following describes the tradeoffs between using a Nested Scope or Mob scope to share segments:

- It may be easier to synchronize the Scope Reference to a slot within a Nested Scope than to a Mob Slot. The synchronization within the Nested Scope is isolated from any changes made outside of it.
- Mob Slots allow you to reference a Segment that has a different edit rate, but all Nested Scope slots have the same edit rate.
- If two different Mob Slots need to share access to a Component, that Component must be in a Mob Slot—a Component in a Nested Scope slot can be referenced only from within the Nested Scope object.
- The Nested Scope object limits the scope in which a Component can be shared. It is limited by the duration of the Nested Scope object. Access is limited to the objects contained within the Nested Scope. This limited access can enable applications to process the sharing in a more efficient manner.

In summary, either Mob Slot objects or Nested Scope objects can be used to allow sharing within a Composition Mob. If you need to reference a Component that has a different edit rate, or if you need to share access to a Component from more than one Mob Slot, you must use a Mob Slot. In all other cases, you can use either method of sharing.

Other Composition Mob Features

This section describes how to perform the following in Composition Mobs:

- Preserve editing choices
- Use audio fades
- Convert edit rates

Preserving Editing Choices—Selectors

In some cases, an application may need to preserve alternatives that were presented to the user and *not* chosen. For example, if a scene was shot with multiple cameras simultaneously, the user can choose the video from the preferred camera angle. In a future editing session, the user may wish to change the video to one that was shot from another camera. By preserving the original choices in the Composition Mob, your application can make it easier for the user to find the alternatives.

The Selector object specifies a selected Segment and a set of alternative Segments. When playing a Composition Mob, an application treats the Selector object as if it were the selected Segment. However, when a user wants to edit the Composition Mob, the application can present the alternative Segments as well as the selected one.

Using Audio Fade In and Fade Out

The Source Clip `FadeInLength`, `FadeInType`, `FadeOutLength`, and Fade-OutType properties allow you to specify audio fades without an Effect Invocation object. Audio fades use these Source Clip properties instead of Effect Invocations for the following reasons:

- Some applications use audio fades on every Segment of audio to avoid noise when cutting from one audio Segment to another—using the Source Clip properties rather than Effect Invocations simplifies the Composition Mob structure.
- Audio fades typically have simple controls arguments and do not need the time-varying control arguments that are allowed in Effect Invocations.

The Composition Mob can contain a default value for audio fade in and fade out. This default value should be used for any audio Source Clip in the Composition Mob that does not explicitly specify a fade in or fade out property.

However, if you want to create a crossfade, you need to do one of the following:

- Insert a Transition object with the `omfi:effect:MonoAudioMixdown` effect between the two audio source clips to cause them to overlap. If the `FadeOutLength` of the preceding Source Clip is not equal to the `FadeInLength` of the following Source Clip, the crossfade will be asymmetric.
- Store overlapping audio Source Clips on different Effect Slots in a `omfi:effect:MonoAudioMixdown`.

Converting Edit Rates

An Edit Rate Converter object converts part of a segment in one edit rate into a segment in another edit rate. You may want to use an Edit Rate Converter when you want to make an exact copy of a Segment in another edit rate and it is important to avoid the rounding errors caused by edit rate conversions.

Edit rate conversions can also be performed by the following mechanisms:

- Scope References to a Mob Slot with a different edit rate
- Source Clip reference to a Mob Slot in another Mob with a different edit rate

6 Describing Media

This chapter shows how OMF Interchange files describe media. OMFI files can include digital media data stored in formats unique to OMF and in formats based on media interchange formats that are also used independently of OMF. Consequently, the way digital media data is described is, in part, dependent on the format used to store it. OMF makes it easier for your program to handle these different formats by providing a layer that is common to all, but your program must know some details about the media format.

OMFI uses the following mechanisms to describe media:

- Source Mobs—describe digital media data stored in files or a physical media source such as videotape, audio tape, and film. The Source Mob contains the following objects that provide information about the media:
  - Mob Slots—specify the number of tracks in the media source, the duration of each track, the edit rate, and the Source Mob that describes the previous generation of media. In addition, Mob Slots contain timecode and edge code information.
  - Media Descriptors—describe the kind of media and the format of the media and specify whether the Source Mobs describe digital media data stored in files or a physical media source.
  - Pulldown objects—describe how media is converted between a film speed and a video speed.
- Media Data objects—contain the digital media data and provide a frame index for compressed digital media data.
- Digital Media Data—contains additional descriptive information for some media formats.

In addition, Master Mobs (MMOB) synchronize Source Mobs and provide a layer of indirection to make it easy to change Source Mobs without changing Composition Mobs that reference them.

This chapter contains the following sections:

- Describing Media with Mob Slots
- Describing Media with Master Mobs
- Describing Timecode with Source Mobs
- Describing Media with Pulldown objects
- Describing Media with Media Descriptors

Appendix C contains a description of the media formats used to store digital media data. It lists references that specify the media formats. In addition, it summarizes some of the descriptive information that is stored in the digital media data.

Describing Media with Mob Slots

A Source Mob represents a file containing digitized media or a physical media source, such as an audio tape, film, or videotape.

If the media described by the Source Mob has been derived from a previous generation of media, the Mob Slots should contain Source Clips that identify the Mob that describes the previous generation. The Source Clip `SourceID`, `SourceTrackID`, and `StartTime` properties identify the Mob. If the Source Mob describes media that is not derived from a previous generation, the Mob Slots should contain Source Clips that omit these properties.

Timecode and Segment Length

A Timecode object in a Source Mob typically appears in a Mob Slot in a Source Mob that describes a videotape or audio tape. In this context it describes the timecode that exists on the tape.

If a tape has a contiguous timecode, the Source Mob should have:

- A Mob Slot with a Track Description for each track of media on the tape; the Mob Slot should have a single Source Clip whose `Length` equals the duration of the tape.
- A Mob Slot with a Track Description for the timecode track that has a `Start` value equal to the timecode at the beginning of the tape and whose `Length` equals the duration of the tape.

If a tape contains noncontiguous timecodes, then the Mob Slot should contain a Sequence of Timecode (TCCP) objects; each representing a contiguous section of timecode on the tape.

In some cases the information required to accurately describe the tape's timecode may not be available. For example, if only a section of a videotape is digitized, the application may not have access to the timecode at the start of the

videotape. In these cases, applications may create a Source Mob in which the duration of the Source Clip does not necessarily match the duration of the videotape.

Sample Rate and Edit Rate

In many cases the sample rate and edit rate in a file Source Mob will be the same. However, it is possible to use different edit rates and sample rates in a Source Mob. For example, you can create a Source Mob for digital audio data, where the edit rate matches the edit rate of the associated video but the sample rate is much higher. The sample rate is specified in the `SampleRate` property in the Media File Descriptor (MDFL). When accessing the digital media data, your application must convert from the edit rate to the sample rate.

The Source Origin

When an application accesses the digital media data, it locates the starting position by measuring from a position known as the source origin. Each file Source Mob indicates this position for each Mob Slot with a Track Description in order to provide a reference point for measurements of its media data.

For example, when you first digitize the audio from a tape, your application would most likely assign a value of 0 to the Track Description `Origin` property. In this case the source origin corresponds to the beginning of the data. Any Source Clip that references this audio will specify a `StartTime` value that is relative to the start of the media.

However, the location of the origin does not necessarily correspond to the actual beginning of the source. For example, if a user redigitizes the audio data in the previous example to add more data at the beginning, the new Media Data object starts at a different point. However, the application will ensure that existing Source Clips in Composition Mobs remain valid by changing the value of the `Origin` property in the Master Mob. By setting the `Origin` to the current offset of the original starting point, the application ensures that existing Composition Mobs remain valid.

Converting Edit Units to Sample Units

A Mob Slot uses its own edit rate. So, a Source Clip in a Composition Mob indicates the starting position in the source and the length of the Segment in edit units. When an application plays a Composition Mob, it maps the Composition Mob's references to the source material into references to the corresponding digital media data.

To play the digital media data referenced by a Composition Mob, the application uses the `StartTime` and `Length` values of the Composition Mob's Source Clip, which are specified in edit units, along with the edit rate to determine the samples to be taken from the media data. The application converts EUs to sample durations, adds the file Mob Slot's `Origin` to the Source Clip's `StartTime`, then converts the resulting sample time offset to a sample byte

offset. Performing the final calculation for some media data formats involves examining the data to find the size in bytes of the particular samples involved. (All samples need not be the same size.) For example, the JPEG Image Data object contains a frame index.

An application would not need to reference the original physical Source Mob of the digitized data unless it is necessary to redigitize or generate a source-relative description, such as an EDL or cut list.

In summary:

- Composition Mobs deal entirely in edit units, which are application-defined time units.
- Digital media data such as video frames, animation frames, and audio samples are stored in a stream of bytes, measured in sample units that represent the time duration of a single sample.
- Applications access media data by converting edit units to sample units and then to byte offsets.
- Master Mobs maintain a reference point in the digitized media data called the source origin. Composition Mobs reference positions in the media data relative to the origin.

Describing Media with Master Mobs

A Master Mob object provides a level of indirection for accessing Source Mobs from Composition Mobs. The media associated with a Source Mob is immutable. Consequently, if you must make any changes to the media data, you must create a new Source Mob with a new unique `MobID`. Typical reasons to change the media data include redigitizing to extend the section of the media included in the file, redigitizing to change the compression used to create the digital media data, and redigitizing to change the format used to store the media data, such as from AIFF audio data to WAVE audio data. A Composition Mob may have many Source Clip objects that reference media data—updating every Source Clip in the Composition Mob each time the media is redigitized would be inefficient. By having the Composition Mob access a Source Mob only through a Master Mob, OMF ensures that you have to change only a single Master Mob when you make changes to the media data.

In addition, a Master Mob can synchronize media data in different Source Mobs. For example, when an application digitizes a videotape, it creates separate Source Mobs for the video and audio data. By having a single Master Mob with one Mob Slot for each Source Mob, the Composition Mob avoids having to synchronize the audio and video tracks each time it references media from different tracks of the videotape.

The same media data can exist in more than one digital media data implementation. Different implementations represent the same original media data but can differ in media format, compression, or byte order. If there are multiple

implementations of digitized media, the Master Mob can contain a Media Group object. The Media Group object contains a set of Source Clip objects, each of which identifies a Source Mob associated with a different implementation of the media data. An application can examine these implementations to find the one that it is able to play or that it can play most efficiently. Media Groups may be needed if you have systems with different architectures or compression hardware accessing a single OMFI file.

If, when a media data file is redigitized, it has to be broken into multiple files, this can be represented by a Sequence object in the Master Mob that contains a series of Source Clip objects, each identifying the Source Mob associated with one of the files.

Typically, Master Mobs have a very simple structure. They have an externally visible Mob Slot for each track of media and do not have any other slots. Typically, each Mob Slot contains a single Source Clip object that identifies the Source Mob. Master Mobs cannot have Effect Invocations, Nested Scopes, Selectors, Edit Rate Converters, or Transitions.

The following lists the reasons for having a Mob Slot in a Master Mob contain an object other than a Source Clip:

- If there are multiple implementations of the same media, the Mob Slot can contain a Media Group instead of a Source Clip object.
- If the media source has been broken into several Source Mobs, the Mob Slot can contain a Sequence object. The Sequence object cannot contain any component other than a Source Clip object or a Media Group object.

Describing Timecode with Source Mobs

The timecode information for digital media data and file Source Mobs is contained in the videotape Source Mob that describes the videotape used to generate the digital media data. Figure 24 illustrates how timecode information is stored in videotape Source Mobs.

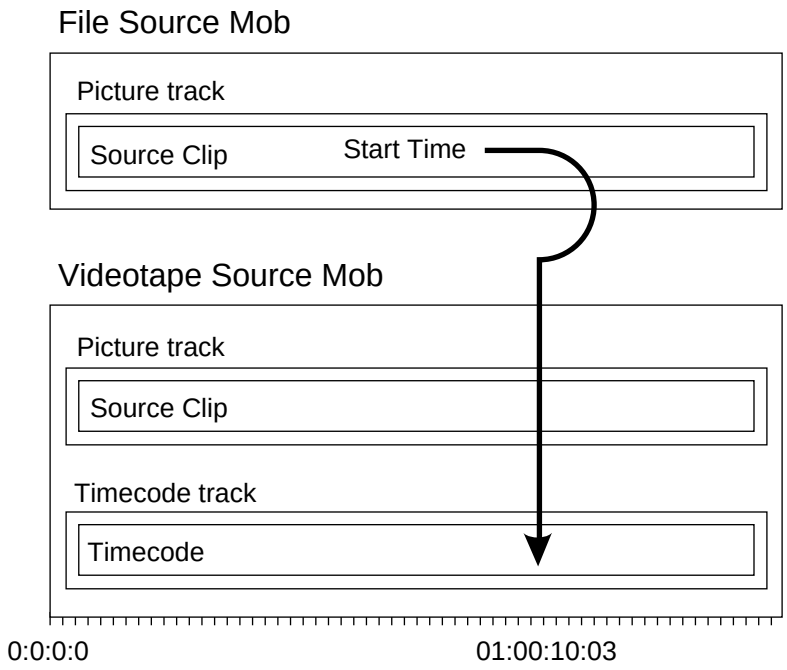


Figure 24: Describing Timecode in Source Mobs

The starting timecode for digital media data is specified by the Source Clip in the File Source Mob and by the timecode track in the videotape Source Mob. The Source Clip specifies the MobID of the videotape Source Mob, the TrackID for the track describing the media data, and the offset in that track. To find the timecode value, you must find the value specified for that offset in the timecode track of the videotape Source Mob.

If a videotape has continuous timecode for the entire tape, it is specified by a single Timecode object. If a videotape has discontinuous timecode, OMF files typically describe it with a single Timecode object that encompasses all timecode values that are used on the videotape. An alternative mechanism to describe discontinuous timecode is to use a timecode track that contains a sequence of Timecode objects, each of which specifies the starting timecode and the duration of each section of continuous timecode on the videotape.

If the timecode track has a single Timecode object, you add the offset to the starting timecode value specified by the Timecode object.

If the timecode track has a sequence of Timecode objects, you calculate the timecode by finding the Timecode object that covers the specified offset in the track and add to its starting timecode the difference between the specified offset and the starting position of the Timecode object in the track.

Describing Media with Pulldown Objects

Pulldown is a process to convert media with one frame rate to media with another frame rate. OMF describes how media has been converted with Pulldown objects in File Source Mobs and videotape Source Mobs.

What is Pulldown?

Pulldown is a process to convert between media at film speed of 24 frames per second (fps) and media at a videotape speed of either 29.97 fps or 25 fps. It is important to track this conversion accurately for two reasons:

- If the final media format is film and the edits are being done in video, you must be able to accurately identify a film frame or the cut may be done at the wrong frame in the film.
- You need to be able to maintain the synchronization between picture and audio.

There are two processes that are used to generate a videotape that matches the pictures on film:

- Telecine—after the film has been processed a videotape is generated from the film negative or workprint.
- Video tap during filming—a video camera taps the images being filmed and records a videotape as the film camera shoots the take. The video camera gets the same image as the film camera either by means of a half-silvered mirror or a parallel lens.

The videotape can then be digitized to produce a digital video data that can be edited on a nonlinear editing system.

It is also possible to digitize a film image without creating a videotape. The film image can be digitized at film resolution, video resolution, or both.

The audio tracks also are transferred from the original recording media to digital audio data stored on a nonlinear editing system. The audio tracks can be transferred by the same mechanism as the video tracks or by a different mechanism.

Nonlinear editing of material that originated on film can use any of the following workflows:

- Offline film project—film to tape to digital to film cut list
- Offline video project—film to tape to digital with matchback to videotape EDL and/or film cut list
- Online video project—film to tape to digital, recording a final cut from digital to tape

Each of these workflows has a different requirement for synchronizing the digital, tape, and film media for both audio and video.

NTSC Three-Two Pulldown

The relation between film speed (24 fps) and NTSC (29.97) is *approximately* 4 to 5. A videotape will have five frames for each four frames of film. Three-Two pulldown accomplishes this by creating three fields from half of the frames and two fields from the other frames. The SMPTE standard specifies that the A and C frames are transferred into two fields and the B and D frames are transferred into three fields.

Since NTSC videotape has a speed of 29.97 fps, in order to get an exact ratio of 4 to 5, the film is played at 23.976 fps in the telecine machine instead of its natural speed of 24 fps.

Figure 25 illustrates how four film frames are converted to five video frames in Three-Two pulldown by converting film frames to either two or three video fields.

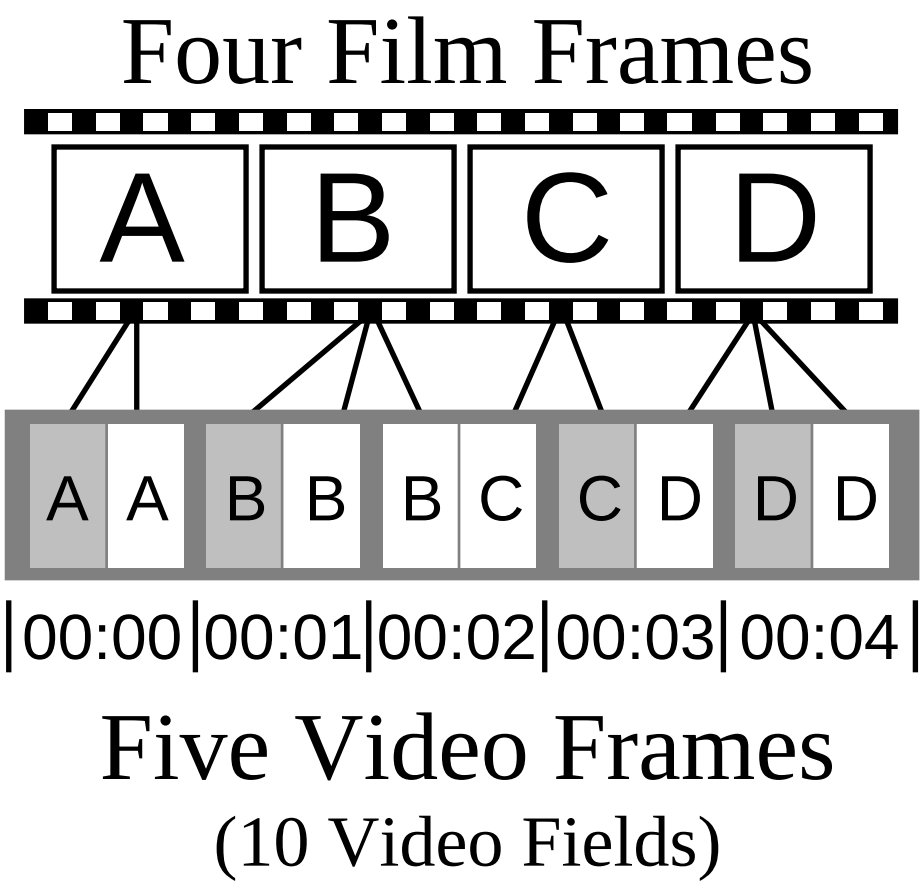


Figure 25: Telecine Three-Two Pulldown

During the telecine process, a white flag can be added to the vertical blanking interval of the first field of video that corresponds to a new film frame. Figure 26 illustrates the fields in Three-Two pulldown that are marked with the white flag.

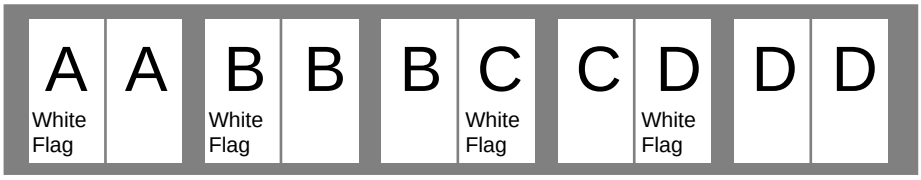


Figure 26: White Flag in Three-Two Pulldown

A tape Mob describing a tape produced by telecine should have edit rates of 30 fps for its tracks. Although the videotape is always played at 29.97 fps, the content has a speed of 30 fps.

If the final distribution format is being generating from film, there are advantages to digitizing the videotape to digital video media that has a film sample rate. This is done by a reverse telecine process where only 4 digital fields are created from 5 video frames, which contain 10 video fields. Figure 27 illustrates using reverse three-two pulldown to generate 24 fps digital video media from a videotape.

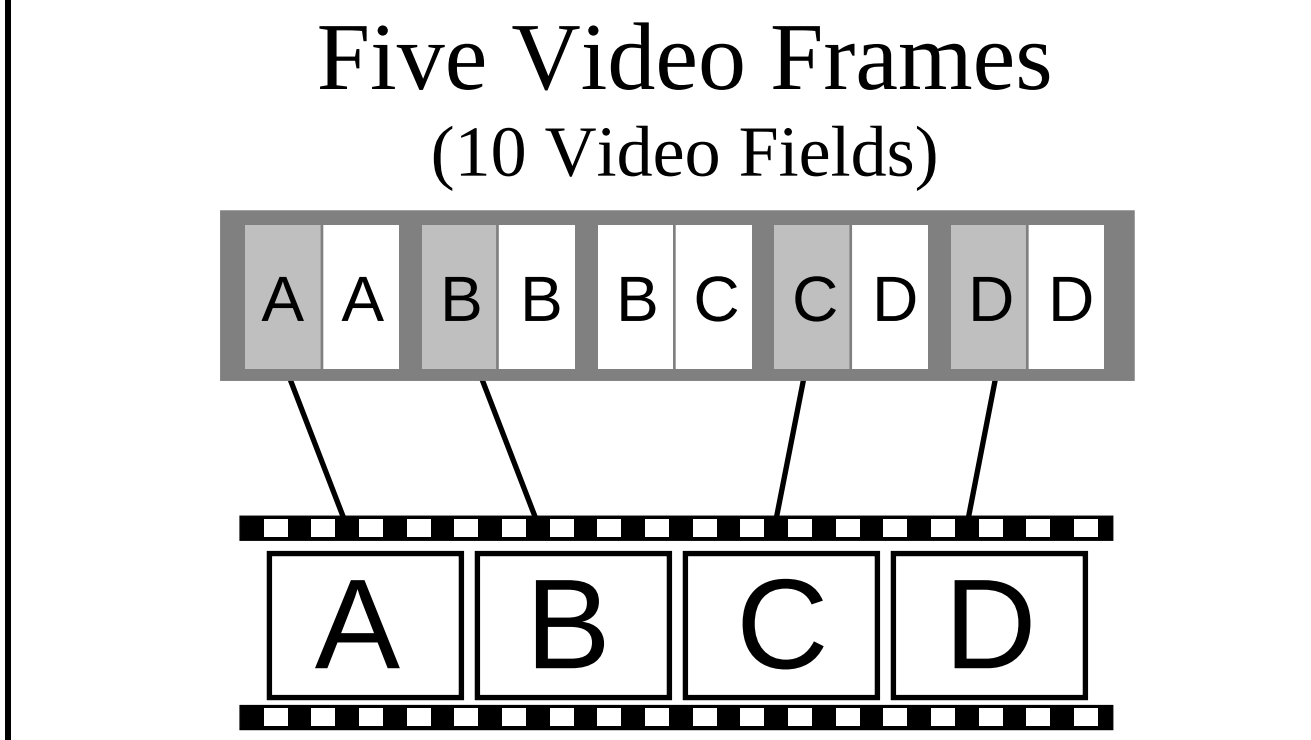


Figure 27: Reverse Three-Two Pulldown

Other Forms of Pulldown

If an NTSC videotape is generated by a video camera running in synchronization with the film camera, the film camera runs at 24 fps and the video runs at 29.97 fps. Four film frames do not correspond to exactly five video frames; they correspond to slightly more than five video frames. The video tap uses a white flag in the vertical blanking area to indicate when a new film frame starts. The first field that starts after the film frame starts is indicated by a white flag.

PAL video and 24 fps film can be converted by simply speeding up the film to PAL's 25 fps rate or can be converted by a pulldown process by converting all 24 frames except the twelfth and twenty-fourth into two fields of video and converting the twelfth and twenty-fourth film frames into three fields of video.

Pulldown Objects in Source Mobs

If NTSC video is digitized to a 24-fps film rate using a reverse Three-Two pull-down, both the File Source Mob and the Videotape Source Mob contain Pull-down objects. Figure 28 illustrates how Pulldown objects describe the process of converting film to video and the video to digitized video media at a film frame rate.

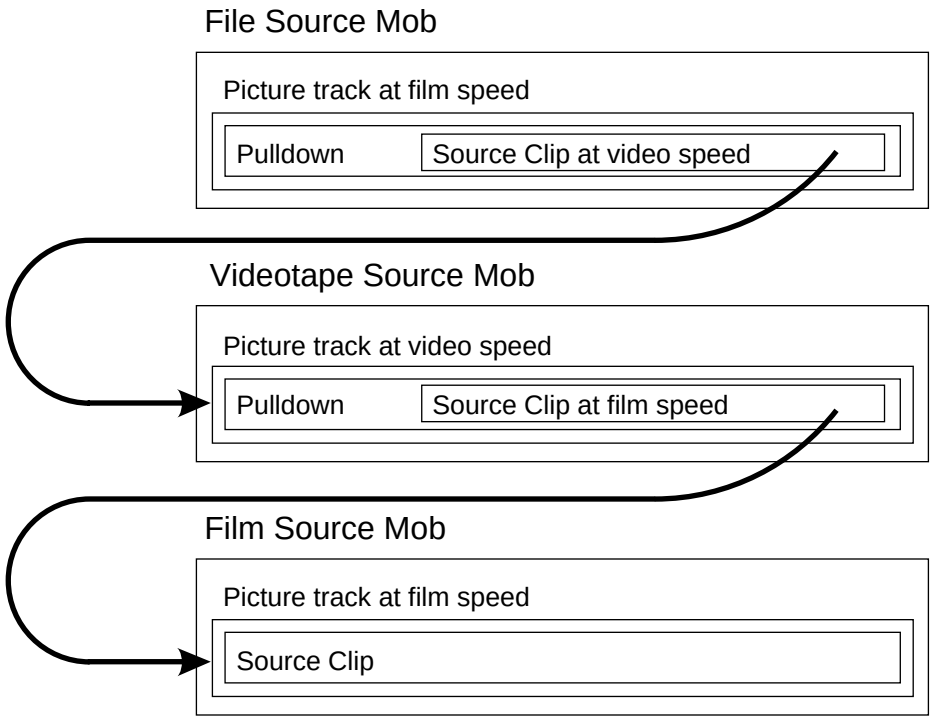


Figure 28: Describing Pulldown in Source Mobs

The Pulldown object in the File Source Mob describes how the videotape was digitized. The track in the File Source Mob has an edit rate of 24/1 but the Source Clip in the Pulldown object has an edit rate of 30/1. The Pulldown object specifies the phase of the first frame of the digital media data. The phase has a value in the range 0 to 3, where 0 specifies the A frame and 3 specifies the D frame.

The Pulldown object in the videotape Source Mob describes how the video was generated from film. The track in the videotape Source Mob has an edit rate of 30/1 but the Source Clip in the Pulldown object has an edit rate of 24/1. The phase specifies where the first frame of the section of videotape is in the 5-frame repeating pattern. The phase has a value in the range 0 to 4, where 0 specifies that the first frame is the AA frame.

You need to use the phase information to convert an offset in the Mob track containing the Pulldown object to an offset in the previous generation Mob. To

convert a film-rate offset, you multiply it by 5/4 to get a video rate offset, but if the result is not an integer, you use the phase information to determine whether you round up or down to get an integer value.

Typically a videotape is generated from more than one piece of film. In this case, the picture track in the videotape Source Mob contains a Sequence object which contains a Pulldown object for each section of film that has been telecined. If the videotape has discontinuous timecode and the videotape Source Mob timecode track contains a single Timecode object, then the Pulldown objects in the Sequence are separated by Filler objects that correspond to the skipped timecodes on the videotape. Figure 29 illustrates how pulldown information can be described with discontinuous timecode which is specified by a single Timecode object.

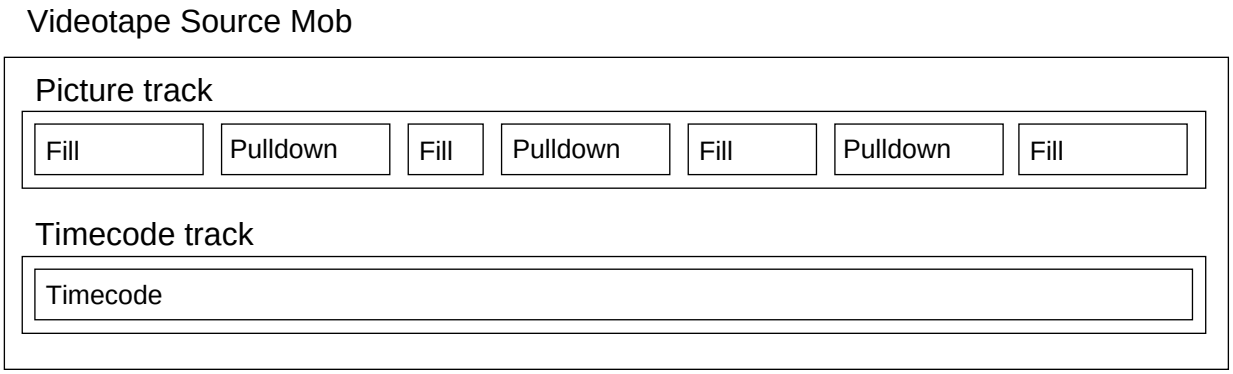


Figure 29: Describing Discontinuous Timecode in Videotape Source Mobs

Describing Media with Media Descriptors

Media Descriptor is an abstract class that describes the format of the media data. The media data can be digitized media data stored in a file or it can be media data on audio tape, film, videotape, or some other form of media storage.

There are two kinds of Media Descriptors:

- Media File Descriptors that describe digital media data stored in Media Data objects or in raw data files. The Media File Descriptor class is also an abstract class; its subclasses describe the various formats of digitized media. If a Media Descriptor object belongs to a subclass of Media File Descriptor, it describes digital media data. If a Media Descriptor object does not belong to a subclass of Media File Descriptor, it describes a physical media source.
- Media Descriptors that describe a physical media source. This specification defines the Media Film Descriptor and Media Tape Descriptor, but addi-

tional private or registered subclasses of Media Descriptors can be defined.

The Media File Descriptor class has the following subclasses defined in this specification:

- AIFC Audio Descriptor (AIFD) class
- Digital Image Descriptor (DIDD) class, which is itself an abstract class and has the following subclasses:
  - Color Difference Component Image Descriptor (CDCI) class—images stored using one luminance and two color-difference components
  - RGBA Component Image Descriptor class—images stored using RGB component color data optionally including an alpha component
- TIFF Image Descriptor class
- WAVE Audio Descriptor class

If the digital media data is stored in an OMFI file, the value of the `IsOMFI` property in the Media Descriptor must be true. If the digital media data is stored in a raw data file, the value of the `IsOMFI` property must be false. Digital media data can be stored in a raw data file to allow an application that does not support OMFI to access it or to avoid duplicating already existing digital media data. However, since there is no `MobID` stored with raw media data, it is difficult to identify a raw media data file if the Locator information is no longer valid. The format of the digital media data in the raw file is the same as it would be if it were stored in an OMFI Media Data object.

The Media File Descriptor specifies the sample rate and length of the media data. The sample rate of the data can be different from the edit rate of the Source Clip object that references it.

Describing Image Media

The goal of the OMF image format is to simplify the representation of image data and to be able to store the information required by video formats in common use. It can support compressed and uncompressed video and can store images in either a color difference component or RGBA component image format. It provides a rich description of the sampling process used to create the digital media from an analog source. This information allows applications to interpret the digital data to represent the original media.

This section explains the image media descriptions that are common to all image media descriptors that are subclasses of the Digital Image Descriptor class.

In order to correctly process or regenerate images, you need access to a complete description of the layout of the images in the file. This description allows applications to extract the relevant information from the files, or, if the images have been lost, restore images to their original digital form. At the most generic level, the description of the images is conveyed by a combination of the following properties: dimensional properties (geometries), sampling properties and colorspace properties.

These properties specify the following about the image format:

- Properties describing interleaving
- Properties describing geometry
- Properties describing sampling
- Properties describing alpha transparency
- Properties describing compression

Properties Describing Interleaving

The major structure of the images is determined by how the images are collated. Images can be compound or atomic. Atomic images contain the entire frame in one contiguous segment. Examples of atomic images include computer graphic frames, digitized film frames, progressive-scan video, two-field interlaced video (even and odd fields mixed together), and single-field video (video where one of the fields is discarded). Compound images are, at this time, limited to two-field non-interlaced video, in which the fields are stored separately.

Since compound video images represent two sub-images, each with the same characteristics, the properties describe the individual fields, and will apply equally to both fields. This is important for applications to recognize, since compound video images have a listed height that is half of the entire frame.

Some image formats allow some form of selection between "interleaved" and "blocked" component order. Interleaved ordering has the data organized by pixels, with each pixel containing all of the components it comprises.

Properties Describing Geometry

The geometry properties describe the dimensions and meaning of the stored pixels in the image. The geometry describes the pixels of an uncompressed image. Consequently, the geometry properties are independent of the compression and subsampling.

Three separate geometries, stored view, sampled view, and display view, are used to define a set of different views on uncompressed digital data. All views are constrained to rectangular regions, which means that storage and sampling have to be rectangular.

The relationships among the views are described in Figure 30.

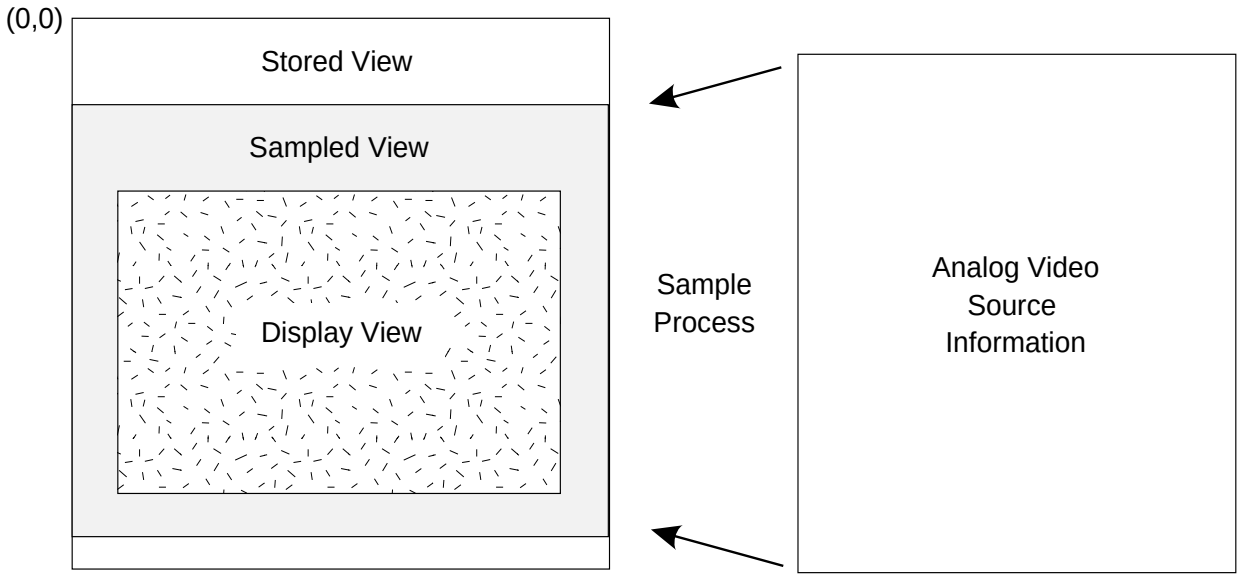


Figure 30: Stored, Sampled, and Displayed View

The stored view is the entire data region corresponding to a single uncompressed frame or field of the image, and is defined by its horizontal and vertical dimension properties. The stored view may include data that is not derived from and would not usually be translated back to analog data.

The sampled view is defined to be the rectangular dimensions in pixels corresponding to the digital data derived from an analog or digital source. These pixels reside within the rectangle defined by the stored view. This would include the image and auxiliary information included in the analog or digital source. For the capture of video signals, the mapping of these views to the original signal is determined by the `VideoLineMap` property.

The display view is the rectangular size in pixels corresponding to the viewable area. These pixels contain image data suitable for scaling, display, warping, and other image processing. The display view offsets are relative to the stored view, not to the sampled view.

Although typically the display view is a subset of the sampled view, it is possible that the viewable area may not be a subset of the sampled data. It may overlap or even encapsulate the sampled data. For example, a subset of the input image might be centered in a computer-generated blue screen for use in a chroma key effect. In this case the viewable pixels on disk would contain more than the sampled image.

Each of these data views has a width and height value. Both the sampled view and the display view also have offsets relative to the top left corner of the stored view.

Properties Describing Sampling

The sampling properties describe the parameters used during the analog-to-digital digitization process. The properties detail the mapping between the signals as well as the format of the source analog signal. If the media originated in a digital format, these properties do not apply.

The `VideoLineMap` property is necessary for images that are derived from or will be converted to video (television) signals. For each field, it describes the mapping, relative to the Sampled View in the digital media, of the digital image lines to the analog signal lines.

The `VideoLineMap` specifies the relationship between the scan lines in the analog signal and the beginning of the digitized fields. The analog lines are expressed in scan line numbers that are appropriate for the signal format. For example, a typical PAL two-field mapping might be {20,332}, where scan line 20 corresponds to the first line of field 1, and scan line 332 corresponds to the first line of field 2. Notice that the numbers are based on the whole frame, not on offset from the top of each field, which would be {20,20}.

A value of 0 is allowed only when computer-generated media has to be treated differently. If the digital media was computer generated (RGB), the values can be either {0,1} (even field first) or {1,0} (odd field first).

Properties Describing Alpha Transparency

The `AlphaTransparency` property determines whether the maximum alpha value or the 0 value indicates that the pixel is transparent. If the property has a value of 1, then the maximum alpha value is transparent and a 0 alpha value is opaque. If the property has a value of 0, then the maximum alpha value is opaque and the 0 alpha value is transparent.

Properties Describing Compression

The `Compression` property specifies that the image is compressed and the kind of compression used. Applications are required to support JPEG and no compression. Registered and private compression kinds are described in documentation available separately from the OMF Developers' Desk. A value of JPEG specifies that the image is compressed according to the following:

- Each image frame conforms to ISO DIS 10918-1. If the frame has two fields then each field is stored as a separate image.
- Images may be preceded or followed by fill bytes.
- Quantization tables are required; they may not be omitted.
- Huffman tables are optional; if omitted, tables from the ISO standard are used.

JPEG image data are color difference component images that have been compressed using the JPEG compression algorithm. The JPEG descriptor specifies a general set of quantization tables for restoring images from the original media. While tables may vary per image, these tables will represent a starting point.

The JPEG Image Data object (JPEG) contains a frame index that allows you to access the frames without searching through the file sequentially. Since the size of the compressed frame is different depending on the image stored on the frame, the frame index is needed to directly access data for a frame.

RGBA Component Image (RGBA) Descriptors

An RGBA Component Image (RGBA) object describes media data that contains component-based images where each pixel is made up of a red, a green, and a blue value. Each pixel can be described directly with a component value or by an index into a pixel palette.

Properties in the RGBA descriptor allow you to specify the order that the color components are stored in the image, the number of bits needed to store a pixel, and the bits allocated to each component.

If a color palette is used, the descriptor allows you to specify the color palette and the structure used to store each color in the palette.

Color Difference Component (CDCI) Image Descriptors

Color Difference Component Image objects specify pixels with one luminance component and two color-difference components. This format is commonly known as YC_bC_r.

It is common to reduce the color information in luma/chroma images to gain a reasonable data reduction while preserving high quality. This is done through chrominance subsampling. Subsampling removes the color information from a fraction of the pixels, leaving the luminance information unmodified. This removal has the effect of cutting the sampling rate of the chrominance to a fraction of the luminance sampling rate. The fraction is controlled by the subsampling specification property. The subsampling factor specifies the number of pixels that will be combined down to one for chrominance components.

Since the color information is reduced across space, it is useful to be able to specify where in the space the stored pixel is sited. Understanding the siting is important because misinterpretation will cause colors to be misaligned.

For uncompressed images, subsampling is limited to horizontal, since the pixels are interleaved.

Describing Audio Media

An AIFC object contains digitized audio data in the big-endian byte ordering. It contains data formatted according to the Audio Interchange File Format (AIFF), Apple Computer, Inc., Version 1. The audio data and the AIFC descriptor data are contained in the AIFC object.

Note that, although the AIFC standard is designed to support compressed audio data, the OMF Interchange required AIFC media format does not include any compressed audio formats. The only AIFC compression form supported is `NONE` and the only AIFC data items that are necessary are the "COMM" and "SSND" data items. All other AIFC data items can be ignored. The descriptive information is contained directly in the OMF `AIFC` object. The AIFC SSND data is duplicated in the AIFC Audio Descriptor to make it more efficient to access this information.

A WAVE object contains digitized audio data in the little-endian byte ordering. It contains data formatted according to the Microsoft/IBM Multimedia Programming Interface and Data Specifications, Version 1.0, but limited to the section describing the RIFF Waveform Audio File Format audio data. The WAVE file information (without the sample data) is duplicated in the WAVE Audio Descriptor to make it more efficient to access this information.

The descriptive information is contained directly in the WAVE object. No additional data properties or objects are defined for WAVE data, because this format contains all of the information needed for playback.

If a Master Mob or Source Mob contains two audio media tracks , the TrackIDs indicate the physical input channel according to the following convention: an odd TrackID indicates the left channel and an even TrackID indicates the right channel.

Describing TIFF Image Media

A TIFF Image Descriptor object describes the TIFF image data associated with the Source Mob. The image data is formatted according to the TIFF specification, Revision 6.0, available from Aldus Corporation. The `TIFF` object type supports only the subset of the full TIFF 6.0 specification defined as baseline TIFF in that document.

Note

The TIFF image format has been superseded by the Color Difference Component Image Descriptor (CDCI) format and the RGBA Component Image Descriptor (RGBA) format in the current version of the specification. The TIFF format is included in this specification for compatibility with OMF Interchange Version 1.0.

The `JPEGTableID` is an assigned type for preset JPEG tables. The table data must also appear in the TIFF object along with the sample data, but cooperating applications can save time by storing a preapproved code in this property that presents a known set of JPEG tables.

See Appendix C for the descriptive information stored in the TIFF digital media data.

Describing Tape and Film

The Media Tape Descriptor (MDTP) describes videotape and audio tape media sources. The Media Film Descriptor (MDFM) describes film sources. Their properties describe the physical storage format used for the media. When you create a tape or film Source Mob, you can include as many of these properties as your application has access to. Since these properties are optional, they can be omitted when they are unknown.

Appendix A

OMF Object Classes

This appendix contains the reference descriptions of the OMF Interchange classes. The reference pages are arranged alphabetically by the full class name.

AIFC Audio Data Class (AIFC)

Contains AIFC audio data.

Data Model

AIFC Audio Data Class (AIFC) Is-a-Kind-of Media Data
AudioData

Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is AIFC.
OMFI:MDAT:MobID	omfi:UID	MobID of the file Source Mob describing the media data.
OMFI:AIFC:AudioData	omfi:DataValue	AIFC format audio data.

Description

An AIFC Audio Data object contains digitized audio data in the big-endian byte ordering (used, for example, on the Motorola 680x0 architecture). It contains data formatted according to the Audio Interchange File Format (AIFF), Apple Computer, Inc., Version 1. The audio data and the AIFC descriptor data are contained in the AIFC object.

Note that, although the AIFC standard is designed to support compressed audio data, the OMF Interchange required AIFC media format does not include any compressed audio formats. The only AIFC compression form supported is NONE, and the only properties that are necessary to specify the data are the COMM and SSND AIFC objects. All other AIFC objects can be ignored.

Related Classes

AIFC Audio Descriptor (AIFD), Header (HEAD), Media Data (MDAT), Source Mob (SMOB), WAVE Audio Data (WAVE), WAVE Audio Descriptor (WAVD)

AIFC Audio Descriptor Class (AIFD)

Describes the AIFC media associated with a Source Mob.

Data Model

AIFC Audio Descriptor Class (AIFD) Is-a-Kind-of Media File Descriptor
Summary

Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is AIFD.
OMFI:MDES:Locator	omfi:ObjRefArray	Set of locators that provide hints to help find the OMFI file or the raw data file that contains the media data. Optional.
OMFI:MDFL:IsOMFI	omfi:Boolean	A True value indicates that the media data is stored in an OMFI file; a False value indicates that the media data is stored in a raw data file.
OMFI:MDFL:SampleRate	omfi:Rational	The native sample rate of the digitized media data.
OMFI:MDFL:Length	omfi:Length32 omfi:Length64	Duration of the media in sample units.
OMFI:AIFD:Summary	omfi:DataValue	A copy of the descriptive information in the associated AIFC Audio Data value.

Description

An AIFC Audio Data object contains digitized audio data in the big-endian byte ordering (used, for example, on the Motorola 68000 architecture). It contains data formatted according to the Audio Interchange File Format (AIFF), Apple Computer, Inc., Version 1. The data is contained directly in the AIFC object. The descriptive data from the AIFC object is duplicated in the AIFC Audio Descriptor to make it more efficient to access this information.

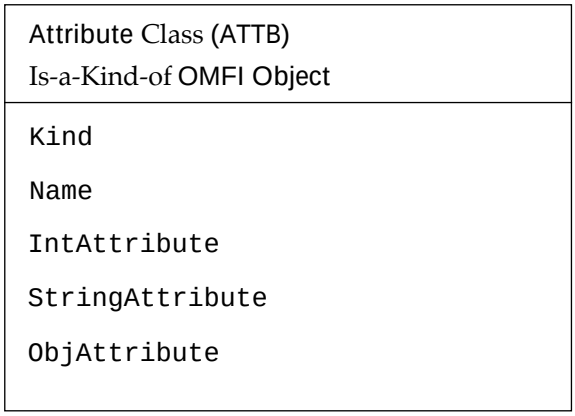
Related Classes

AIFC Audio Data (AIFC), Media File Descriptor (MDFL), Locator (LOCR), Source Mob (SMOB), WAVE Audio Data (WAVE), WAVE Audio Descriptor (WAVD)

Attribute Class (ATTB)

Contains user-specified data.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is ATTB.
OMFI:ATTB:Kind	omfi:AttrKind	Specifies the kind of Attribute value. May have the following values: 0 kOMFNullAttribute — Unspecified type 1 kOMFIntegerAttribute — Integer value 2 kOMFStringAttribute — String value 3 kOMFObjectAttribute — Object reference value
OMFI:ATTB:Name	omfi:String	User-specified name.
OMFI:ATTB:IntAttribute	omfi:Int32	User-specified integer value. Optional, but if the Kind property has a kOMFIntegerAttribute value, this property is required.
OMFI:ATTB:StringAttribute	omfi:String	User-specified string value. Optional, but if the Kind property has a kOMFStringAttribute value, this property is required.
OMFI:ATTB:ObjAttribute	omfi:ObjRef	User-specified value contained in an OMF object. Optional, but if the Kind property has a kOMFObjectAttribute value, this property is required.

Description

An Attribute object specifies a category name and a value for the category. Each Attribute object has a category name and a value. The Mob object User-Attributes property provides a mechanism to associate user-specified data

with a Mob by using Attribute Arrays and Attributes. This mechanism allows the user to define categories and provide values for one or more Mobs in the OMF file. User attributes can be used to specify information such as the take number or lighting conditions of a section of media.

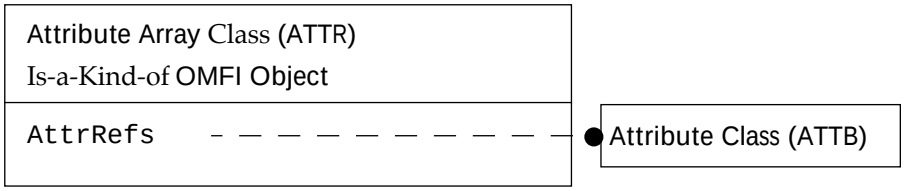
Related Classes

Attribute Array (ATTR), Mob (MOBJ)

Attribute Array Class (ATTR)

Contains a set of Attribute objects.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is ATTR.
OMFI:ATTR:AttrRefs	omfi:ObjRefArray	HAS a set of Attribute objects.

Description

An Attribute Array object contains a set of Attribute objects. Each Attribute object has a category name and a value. Attribute Arrays are typically used to store user-specified data. The Mob object UserAttributes property provides a mechanism to associate user-specified data with a Mob.

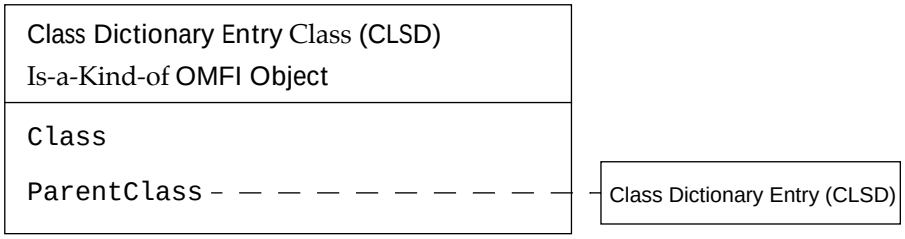
Related Classes

Attribute (ATTB), Mob (MOBJ)

Class Dictionary Entry Class (CLSD)

Extends the OMFI class hierarchy with a new class.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is CLSD.
OMFI:CLSD:Class	omfi:ClassID	Specifies the class being defined.
OMFI:CLSD:ParentClass	omfi:ObjRef	HAS-REFERENCE to a Class Dictionary Entry that specifies the parent of the class being defined. Optional; if the class being defined is a class described in this specification, this property must be omitted.

Description

The Header (HEAD) object `ClassDictionary` property has a set of Class Dictionary Entry objects. This set describes extensions to the classes described in this specification that are used in the OMFI file. Each Class Dictionary Entry object describes a new class or a parent class of a new class.

If the class being defined by the Class Dictionary Entry object is defined in this document, the `ParentClass` property must be omitted. You are not allowed to define a new parent class for a class defined in this specification. Any class extension must be descended from the OMFI Object class.

For example, to define a new class called the Modulated Source Clip (MSCP) class that is a subclass of the Source Clip class, you create two Class Dictionary Entry objects:

- The first Class Dictionary Entry object `Class` property has a value of `MSCP`, and its `ParentClass` property has a value that references the second Class Dictionary Entry object.
- The second Class Dictionary Entry object `Class` property has a value of `SCLP`, and it does not have a `ParentClass` property.

This relationship defines the class identified by the Class ID `MSCP` as a subclass of the class identified by `SCLP`. Since the Source Clip class is defined in this specification, you must not specify its parent class.

An application that encounters a Modulated Source Clip object and does not recognize the new class can treat the object as a Source Clip object based on the information in the Header `ClassDictionary` property. This application will ignore any additional properties defined for the class.

Related Classes

Header (HEAD)

Color Difference Component Image Descriptor Class (CDCI)

Describes the media stored with one luminance component and two color-difference components that is associated with a Source Mob.

Data Model

Color Difference Component Image Descriptor Class (CDCI) Is-a-Kind-of Digital Image Descriptor
ComponentWidth HorizontalSubsampling ColorSiting BlackReferenceLevel WhiteReferenceLevel ColorRange PaddingBits

Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is CDCI.
OMFI:MDES:Locator	omfi:ObjRefArray	Set of locators that provide hints to help find the OMFI file or the raw data file that contains the media data. Optional.
OMFI:MDFL:IsOMFI	omfi:Boolean	A True value indicates that the media data is stored in an OMFI file; a False value indicates that the media data is stored in a raw data file.
OMFI:MDFL:SampleRate	omfi:Rational	The native sample rate of the digitized media data.
OMFI:MDFL:Length	omfi:Length32 omfi:Length64	Duration of the media in sample units.
OMFI:DIDD:Compression	omfi:String	Kind of compression and format of compression information; a string with the following values that must be recognized by all OMFI-compatible applications: JPEG ISO JPEG stream Other registered and private strings may be defined. Optional; if there is no compression, the property is omitted.

Property Name	Type	Explanation
OMFI:DIDD:StoredHeight	omfi:UInt32	Number of pixels in vertical dimension of stored view. See the Description section of DIDD class for an explanation of image geometry.
OMFI:DIDD:StoredWidth	omfi:UInt32	Number of pixels in horizontal dimension of stored view.
OMFI:DIDD:SampledHeight	omfi:UInt32	Number of pixels in vertical dimension of sampled view. Optional; the default value is StoredHeight. See the Description section of DIDD class for an explanation of image geometry.
OMFI:DIDD:SampledWidth	omfi:UInt32	Number of pixels in horizontal dimension of sampled view. Optional; the default value is StoredWidth.
OMFI:DIDD:SampledXOffset	omfi:Int32	X offset, in pixels, from top left corner of stored view. Optional; the default value is 0.
OMFI:DIDD:SampledYOffset	omfi:Int32	Y offset, in pixels from top left corner of stored view. Optional; the default value is 0.
OMFI:DIDD:DisplayHeight	omfi:UInt32	Number of pixels in vertical dimension of display view. Optional; the default value is StoredHeight. See the Description section of DIDD class for an explanation of image geometry.
OMFI:DIDD:DisplayWidth	omfi:UInt32	Number of pixels in vertical dimension of display view. Optional; the default value is StoredWidth.
OMFI:DIDD:DisplayXOffset	omfi:Int32	X offset, in pixels, from top left corner of stored view. Optional; the default value is 0.
OMFI:DIDD:DisplayYOffset	omfi:Int32	Y offset, in pixels, from top left corner of stored view. Optional; the default value is 0.

Property Name	Type	Explanation
OMFI:DIDD:FrameLayout	omfi:LayoutType	Describes whether all data for a complete sample is in one frame or is split into more than one field. Values are 0 FULL_FRAME: frame contains full sample in progressive scan lines. 1 SEPARATE_FIELDS: sample consists of two fields, which when interlaced produce a full sample. 2 SINGLE_FIELD: sample consists of two interlaced fields, but only one field is stored in the data stream. 3 MIXED_FIELDS.
OMFI:DIDD:VideoLineMap	omfi:Int32Array	Specifies the scan line in the analog source that corresponds to the beginning of each digitized field. For single field video, there is 1 value in the array and for interleaved video, there are 2 values in the array.
OMFI:DIDD:ImageAspectRatio	omfi:Rational	Describes the ratio between the horizontal size and the vertical size in the intended final image.
OMFI:DIDD:AlphaTransparency	omfi:Int32	A value of 1 means that the maximum Alpha value is transparent. A value of 0 means that the 0 Alpha value is transparent. Optional.
OMFI:DIDD:Gamma	omfi:Rational	Specifies the expected output gamma setting on the video display device. Optional.
OMFI:DIDD:ImageAlignmentFactor	omfi:Int32	Specifies the alignment when storing the digital media data. For example, a value of 16 means that the image is stored on 16-byte boundaries. The starting point for a field will always be a multiple of 16 bytes. If the field does not end on a 16-byte boundary, the remaining bytes are unused. Optional.
OMFI:CDCI:ComponentWidth	omfi:Int32	Specifies the number of bits used to store each component. Can have a value of 8, 10, or 16. Each component in a sample is packed contiguously; the sample is filled with the number of bits specified by the optional OMFI:CDCI:PaddingBits property. If the OMFI:CDCI:PaddingBits property is omitted, samples are packed contiguously.

Property Name	Type	Explanation
OMFI:CDCI:HorizontalSubsampling	omfi:UInt32	Specifies the ratio of luminance sampling to chrominance sampling in the horizontal direction. For 4:2:2 video, the value is 2, which means that there are twice as many luminance values as there are color-difference values. The other legal value is 1.
OMFI:CDCI:ColorSiting	omfi:ColorSitingType	Specifies how to compute subsampled chrominance component values. Values are: 0 coSiting To calculate subsampled pixels, take the preceding pixel's color value, discard the other color values, and cosite the color with the first luminance value. 1 averaging To calculate subsampled pixels, take the average of the two adjacent pixels' color values, and site the color in the center of the luminance pixels. 2 threeTap To calculate subsampled pixels, take 25 percent of the previous pixel's color value, 50 percent of the first value, and 25 percent of the second value. For the first value in a row, use 75 percent of that value since there is no previous value. The threeTap value is only meaningful when the HorizontalSubsampling property has a value of 2. Optional; when omitted, treat as coSiting.
OMFI:CDCI:BlackReferenceLevel	omfi:UInt32	Specifies the digital luminance component value associated with black. For CCIR-601/2, the value is 16; for YUV, the value is 0. The same value is used in CDCI and RGBA when the standard CCIR colorspace conversion is used. Optional; if omitted the default value is 0.

Property Name	Type	Explanation
OMFI:CDCI:WhiteReferenceLevel	omfi:UInt32	Specifies the digital luminance component value associated with white. For CCIR-601/2, 8-bit video, the value is 235; for YUV 8-bit video, the value is 255. Optional; if omitted, the default value is maximum unsigned integer value for component size.
OMFI:CDCI:ColorRange	omfi:UInt32	Specifies the range of allowable digital chrominance component values. Chrominance values are signed and the range specified is centered on 0. For CCIR-601/2, the value is 225; for YUV the value is 255. This value is used for both chrominance components. Optional; the default value is the maximum unsigned integer value for the component size.
OMFI:CDCI:PaddingBits	omfi:Int16	Specifies the number of bits padded to each pixel. Optional; default is 0.

Description

Color Difference Component Image objects specify pixels with one luminance component and two color-difference components. This format is commonly known as YC_bC_r.

Chrominance subsampling reduces storage requirements by omitting the color difference information for some pixels. When reading the image, the color difference information for these pixels is calculated from the color difference information of the adjacent pixels. Color siting specifies how to calculate the color difference information when the two pixels have unequal color difference information.

See the description of the Digital Image Descriptor class for a description of the compression and geometry properties.

If the PaddingBits property is set to 0 or omitted, the image data contains one pixel immediately after another. If one pixel ends within a byte, the remaining bits in that byte are used for the following pixel. In all cases, the video frame is padded to a byte boundary.

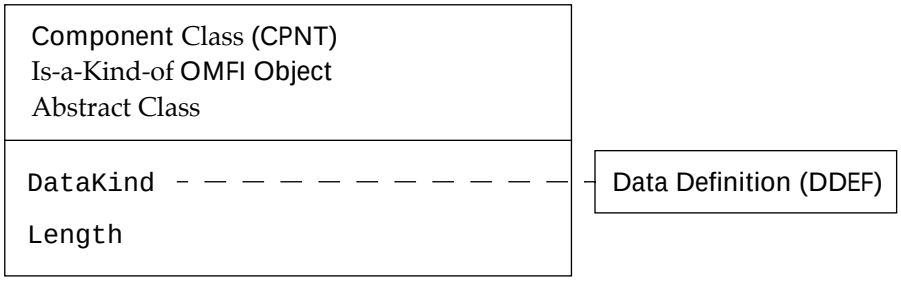
Related Classes

Digital Image Descriptor (DIDD), Image Data (IDAT), JPEG Image Data (JPEG), RGBA Component Image Descriptor (RGBA), Source Mob (SMOB)

Component Class (CPNT)

Represents time-varying data and has a type and length.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is CPNT.
OMFI:CPNT:DataKind	omfi:ObjRef	HAS-REFERENCE to a Data Definition object that specifies the data kind of the component.
OMFI:CPNT:Length	omfi:Length32 omfi:Length64	Specifies the duration in edit units of the component.

Description

The Component class is an abstract class that represents a section of media or other time-varying data. Since it is an abstract class, there are no objects in an OMFI file that have an ObjClass value of CPNT; all objects in any OMFI file that belong to the Component class also belong to a subclass of Component. The two subclasses of Component defined in this specification are Segment (SEGM) and Transition (TRAN). A Segment is a time-varying object that has an independent meaning and does not have to be followed or preceded by another object in a Sequence. Segment is also an abstract class—it has many subclasses such as Effect Invocation, Filler, Sequence, and Source Clip. A Transition object describes the way to change from one segment to another; consequently, a Transition can only appear in a Sequence between two Segments.

The two properties of the Component class, DataKind and Length, specify the fundamental attributes of any OMFI time-varying object. DataKind identifies the format and meaning of the data values, and Length specifies the duration in edit units of the component. The duration is the time period in which the component defines data values. The component's edit rate is defined by the Mob Slot, Effect Slot, or Edit Rate Converter that contains the Component. With the information from these two properties, an application can determine its level of support for a Component. If it cannot accurately play

the Component, at least it can use the basic information to decide how to proceed.

A typical `DataKind` value of a Component in a video slot would be `omfi:data:Picture`. The data kind for a Component must match or be convertible to the data kind required by the context of the Component. For example, a Component object in a Sequence object must have a data kind that is the same as or convertible to the data kind of the Sequence.

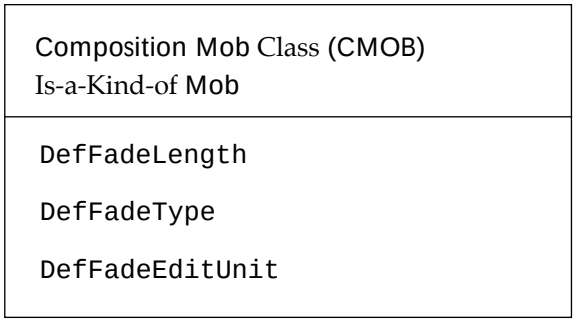
Related Classes

Data Definition (DDEF), Edit Rate Converter (ERAT), Effect Slot (ESLT), Mob Slot (MSLT), OMFI Object (OOBJ), Segment (SEGM), Transition (TRAN)

Composition Mob Class (CMOB)

Describes the editing information that combines segments of media into a presentation.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is CMOB.
OMFI:MOBJ:MobID	omfi:UID	Unique Mob Identification.
OMFI:MOBJ:Name	omfi:String	Name of mob for display to end user. Optional.
OMFI:MOBJ:Slots	omfi:ObjRefArray	Contains the externally visible Mob Slots and the internal slots containing media and other time-varying information.
OMFI:MOBJ:LastModified	omfi:TimeStamp	Date and time when mob was last modified.
OMFI:MOBJ:CreationTime	omfi:TimeStamp	Date and time when the mob was originally created.
OMFI:MOBJ:UserAttributes	omfi:ObjRef	Specifies a set of user attributes, which provide additional information about the Mob. The Attribute Array contains a set of Attributes. Optional.
OMFI:CMOB:DefFadeLength	omfi:Length32 omfi:Length64	Specifies the default length of the audio fade-in and fade-out to be applied to all audio Source Clips that do not specify the audio fade properties. Optional; if specified, then the default fade type and the default fade edit units must also be specified.

Property Name	Type	Explanation
OMFI:CMOB:DefFadeType	omfi:FadeType	Specifies the default type of audio fade. Optional; if specified, then the default length and default edit units must also be specified. Specifies the type of the audio fade in; may have one of the following values: 0 No fade 1 Linear amplitude fade 2 Linear power fade 3 Linear dB fade Additional registered and private fade in types may be defined. Optional.
OMFI:CMOB:DefFadeEditUnit	omfi:Rational	Specifies the edit units in which the default fade length is specified. Optional; if specified, then the default fade length and default fade type must also be specified.

Description

A Composition Mob object contains a complete multimedia presentation. Composition Mob objects can range from a very simple structure containing only a single Segment of media to extremely complex structures containing multiple Mob Slots representing different media kinds; each Mob Slot containing complex Sequences, Transitions, and multilayer effects.

Mob Slots and Track Descriptions

Mob Slots contain Segments of media or other time-varying data. A Mob Slot can be externally visible or externally hidden. Mob Slots that are externally visible by definition have a Track Description (TRKD); these Mob Slots represent outputs of the Mob. All slots can be referenced by other slots within the same Mob via Scope References (SREF).

Mob Slots are ordered to allow Scope References with the Mob.

Default Audio Fade Properties

Specify the default values to create audio "soft cuts," which are created by applying a fade-in effect at the start of an audio Source Clip and a fade-out effect at its end. If the edit rate specified in the `DefFadeEditUnit` property is different from the edit rate of the track containing the audio Source Clip, the length specified by the `DefFadeLength` property must be converted to the track's edit rate. The fade length and fade type specified by these properties specify both the default fade-in and fade-out.

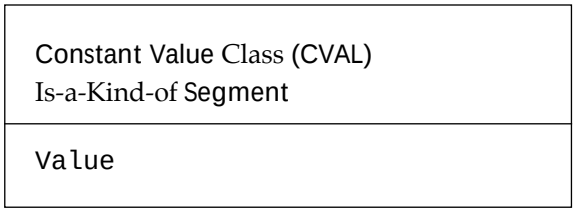
Related Classes

Header (HEAD), Master Mob (MMOB), Mob (MOBJ), Mob Slot (MSLT), Source Clip (SCLP), Source Mob (SMOB), Track Description (TRKD)

Constant Value Class (CVAL)

Specifies a constant data value for the duration of the component; typically used to specify a constant value for an effect control slot.

Data Model



Implementation

Property Name	Type	Explanation
`OMFI:OOBJ:ObjClass`	`omfi:ClassID`	Class is CVAL.
`OMFI:CPNT:DataKind`	`omfi:ObjRef`	HAS-REFERENCE to a Data Definition object that specifies the data kind of the value.
`OMFI:CPNT:Length`	`omfi:Length32` `omfi:Length64`	Specifies the duration in edit units of the constant value.
`OMFI:CVAL:Value`	`omfi:DataValue`	Specifies the value.

Rule

A Constant Value object can only have a data kind that has a defined data constant format. A Constant Value object cannot have a data kind that specifies a media stream because these formats do not have a defined constant format. Data kinds that specify a media stream include `omfi:data:EdgeCode`, `omfi:data:Picture`, `omfi:data:PictureWithMatte`, `omfi:data:Matte`, `omfi:data:Sound`, `omfi:data:StereoSound`, and `omfi:data:Timecode`.

Description

A Constant Value object can be used to supply a constant value for a control argument in an Effect Slot object. Typically, this is done by having the Effect Slot Segment be the Constant Value object.

Although Constant Value objects are typically used as effect control arguments, they can be used in any context that a Segment object is allowed. For example, you can use a Constant Value object with an `omfi:data:Color` value in any place where an `omfi:data:Picture` Segment is allowed. This is allowed because the `Color` data kind is automatically converted to the `Picture` data kind.

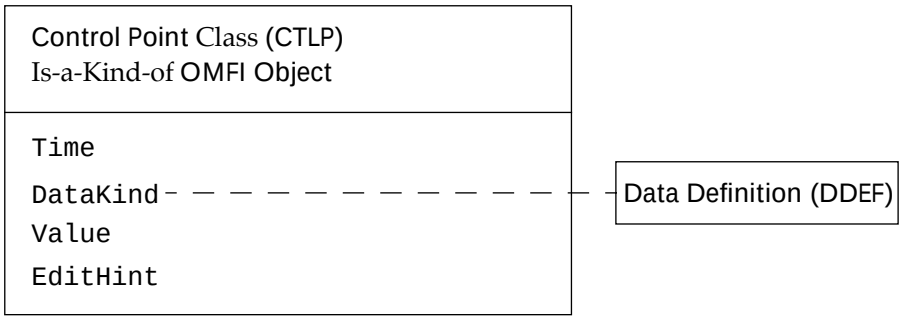
Related Classes

Effect Invocation (EFFE), Effect Slot (ESLT), Segment (SEGM), Varying Value (VVAL)

Control Point Class (CTLP)

Specifies a value for the specified time point in a Varying Value segment; typically used to specify a value in an effect control slot.

Data Model



Implementation

Property Name	Type	Explanation
`OMFI:OOBJ:ObjClass`	`omfi:ClassID`	Class is CTLP.
`OMFI:CTLP:Time`	`omfi:Rational`	Specifies the time within the Varying Value segment for which the value is defined.
`OMFI:CTLP:DataKind`	`omfi:ObjRef`	HAS-REFERENCE to a Data Definition object that specifies the data kind of the value.
`OMFI:CTLP:Value`	`omfi:DataValue`	Specifies the value.
`OMFI:CTLP:EditHint`	`omfi:EditHintType`	Specifies a hint to be used if the Effect Invocation starting time or length is changed during editing. Can be `EH_Proportional`, `EH_RelativeLeft`, or `EH_RelativeRight`. Optional.

Rule

A Control Point object can only have a data kind that has a defined data constant format. A Control Point object cannot have a data kind that specifies a media stream because these formats do not have a defined constant format. Data kinds that specify a media stream include `omfi:data:EdgeCode`, `omfi:data:Picture`, `omfi:data:PictureWithMatte`, `omfi:data:Matte`, `omfi:data:Sound`, `omfi:data:StereoSound`, and `omfi:data:Timecode`.

Description

A Control Point object specifies the value at a specific time in a Varying Value object. The Control Point object must have the same data kind as the Varying Value object containing it.

A `Time` equal to 0.0 represents the time at the beginning of the Varying Value object; a `Time` equal to 1.0 represents the time at the end of the Varying Value object.

Related Classes

Data Definition (DDEF), Effect Invocation (EFFE), Effect Slot (ESLT), Varying Value (VVAL)

Data Definition Class (DDEF)

Identifies a globally defined data kind and makes it referenceable within an OMFI file.

Data Model

Data Definition Class (DDEF) Is-a-Kind-of OMFI Object
`DataKindID`

Implementation

Property Name	Type	Explanation
`OMFI:OOBJ:ObjClass`	`omfi:ClassID`	Class is DDEF.
`OMFI:DDEF:DataKindID`	`omfi:UniqueName`	A string of characters that specify a qualified name. Segments of the name are separated by: (colon characters).

Description

A Data Definition object identifies the kind of the time-varying data produced by a Component object. Each Component object in an OMFI file has a reference to a Data Definition object. Many Component objects can have a reference to a single Data Definition object.

The Data Definition object specifies the unique name of the data kind. The meaning, internal format, and size of the data kind are not described in the Data Definition object. This information is provided in this specification or in the documentation provided with registered or private formats.

A Data Definition can identify either a kind of data with a constant value format or a kind of data associated with a media stream.

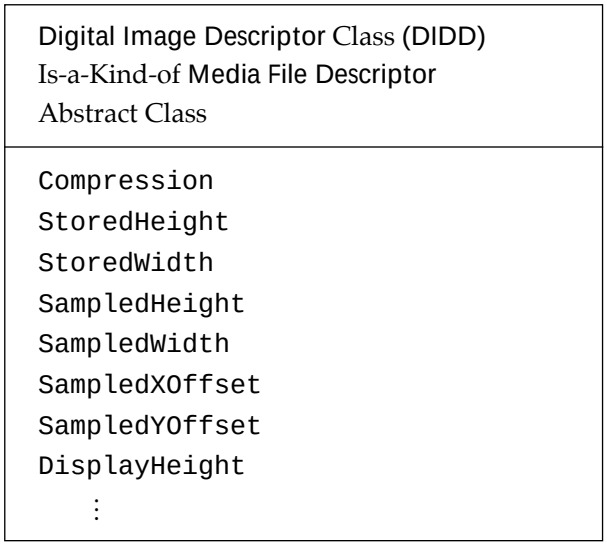
Related Classes

Component (CPNT), Control Point (CTLP), Header (HEAD)

Digital Image Descriptor Class (DIDD)

Describes the image media data associated with a Source Mob.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is DIDD.
OMFI:MDES:Locator	omfi:ObjRefArray	Set of locators that provide hints to help find the OMFI file or the raw data file that contains the media data. Optional.
OMFI:MDFL:IsOMFI	omfi:Boolean	A True value indicates that the media data is stored in an OMFI file; a False value indicates that the media data is stored in a raw data file.
OMFI:MDFL:SampleRate	omfi:Rational	The native sample rate of the digitized media data.
OMFI:MDFL:Length	omfi:Int32	Duration of the media in sample units.
OMFI:DIDD:Compression	omfi:String	Kind of compression and format of compression information; a string with the following values that must be recognized by all OMFI-compatible applications: JPEG ISO JPEG stream Other registered and private strings may be defined. Optional; if there is no compression, the property is omitted.

Property Name	Type	Explanation
OMFI:DIDD:StoredHeight	omfi:UInt32	Number of pixels in vertical dimension of stored view. See the Description section for an explanation of image geometry.
OMFI:DIDD:StoredWidth	omfi:UInt32	Number of pixels in horizontal dimension of stored view.
OMFI:DIDD:SampledHeight	omfi:UInt32	Number of pixels in vertical dimension of sampled view. Optional; the default value is StoredHeight. See the Description section for an explanation of image geometry.
OMFI:DIDD:SampledWidth	omfi:UInt32	Number of pixels in horizontal dimension of sampled view. Optional; the default value is StoredWidth.
OMFI:DIDD:SampledXOffset	omfi:Int32	X offset, in pixels, from top-left corner of stored view. Optional; default value is 0.
OMFI:DIDD:SampledYOffset	omfi:Int32	Y offset, in pixels from top-left corner of stored view. Optional; default value is 0.
OMFI:DIDD:DisplayHeight	omfi:UInt32	Number of pixels in vertical dimension of display view. Optional; the default value is StoredHeight. See the Description section for an explanation of image geometry.
OMFI:DIDD:DisplayWidth	omfi:UInt32	Number of pixels in vertical dimension of display view. Optional; the default value is StoredWidth.
OMFI:DIDD:DisplayXOffset	omfi:Int32	X offset, in pixels, from top-left corner of stored view. Optional; the default value is 0.
OMFI:DIDD:DisplayYOffset	omfi:Int32	Y offset, in pixels, from top-left corner of stored view. Optional; the default value is 0.
OMFI:DIDD:FrameLayout	omfi:LayoutType	Describes whether all data for a complete sample is in one frame or is split into more than one field. Values are 0 FULL_FRAME: frame contains full sample in progressive scan lines. 1 SEPARATE_FIELDS: sample consists of two fields, which when interlaced produce a full sample. 2 SINGLE_FIELD: sample consists of two interlaced fields, but only one field is stored in the data stream. 3 MIXED_FIELDS.
OMFI:DIDD:VideoLineMap	omfi:Int32Array	Specifies the scan line in the analog source that corresponds to the beginning of each digitized field. For single-field video, there is 1 value in the array; for interleaved video, there are 2 values in the array.

Property Name	Type	Explanation
OMFI:DIDD:ImageAspectRatio	omfi:Rational	Describes the ratio between the horizontal size and the vertical size in the intended final image.
OMFI:DIDD:AlphaTransparency	omfi:Int32	A value of 1 means that the maximum Alpha value is transparent. A value of 0 means that the 0 Alpha value is transparent. Optional.
OMFI:DIDD:Gamma	omfi:Rational	Specifies the expected output gamma setting on the video display device. Optional.
OMFI:DIDD:ImageAlignmentFactor	omfi:Int32	Specifies the alignment when storing the digital media data. For example, a value of 16 means that the image is stored on 16-byte boundaries. The starting point for a field will always be a multiple of 16 bytes. If the field does not end on a 16-byte boundary, the remaining bytes are unused. Optional.

Rules

1. If you specify any of the sampled geometry properties, `SampledHeight`, `SampledWidth`, `SampledXOffset`, and `SampledYOffset`, you must specify all of them.
2. If you specify any of the display geometry properties, `DisplayHeight`, `DisplayWidth`, `DisplayXOffset`, and `DisplayYOffset`, you must specify all of them.

Description

The Digital Image Descriptor (DIDD) is an abstract class. Objects of its subclasses, Color Difference Component Image Descriptor (CDCI) and RBGA Component Image Descriptor (RGBA) are used in a Source Mob and describe the format of the digital image data associated with the Source Mob. The Digital Image Descriptor specifies the properties common to these formats. These properties specify the following about the image format:

- Compression
- Geometry
- Alpha Transparency Value

Compression

The `Compression` property specifies that the image is compressed and the kind of compression used. Applications are required to support JPEG and no compression. Registered and private compression kinds are described in documentation available separately from the OMF Developers' Desk. A value of `JPEG` specifies that the image is compressed according to the following:

- Each image frame conforms to ISO DIS 10918-1. If the frame has two fields,

then each field is stored as a separate image.

- Images may be preceded or followed by fill bytes.
- Quantization tables are required; they may not be omitted.
- Huffman tables are optional; if omitted, tables from the ISO standard are used.

Geometry

The geometry properties describe the dimensions and meaning of the stored pixels in the image. The geometry describes the pixels of an uncompressed image. Consequently, the geometry properties are independent of the compression and subsampling.

Three separate geometries—stored view, sampled view, and display view—are used to define a set of different views on uncompressed digital data. All views are constrained to rectangular regions, which means that storage and sampling has to be rectangular.

Each of these data views will have a width and height value. Both the sampled view and the display view also have offsets relative to the top left corner of the stored view.

The `FrameLayout` property describes whether a complete image is contained in one full field or in two separate fields.

The `ImageAspectRatio` describes the ratio between the horizontal size and the vertical size in the intended final image.

The `VideoLineMap` specifies the relationship between the scan lines in the analog signal and the beginning of the digitized fields. The analog lines are expressed in scan line numbers that are appropriate for the signal format. For example, a typical PAL two-field mapping might be {20,332}, where scan line 20 corresponds to the first line of field 1, and scan line 332 corresponds to the first line of field 2. Notice that the numbers are based on the whole frame, not on offsets from the top of each field, which would be {20,20}

A value of 0 is allowed only when computer-generated media has to be treated differently. If the digital media was computer generated (RGB), the values may be either {0,1} (even field first) or {1,0} (odd field first).

Alpha Transparency

The `AlphaTransparency` property determines whether the maximum alpha value or the 0 value indicates that the pixel is transparent. If the property has a value of 1, then the maximum alpha value is transparent and a 0 alpha value is opaque. If the property has a value of 0, then the maximum alpha value is opaque and the 0 alpha value is transparent.

Related Classes

Color Difference Component Image Descriptor (CDCI), Image Data (IDAT), JPEG Image Data (JPEG), RGBA Component Image Descriptor (RGBA), Source Mob (SMOB)

DOS Locator Class (DOSL)

Provides information to help find the DOS file containing media data.

Data Model

DOS Locator Class (DOSL) Is-a-Kind-of Locator
PathName

Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is DOSL.
OMFI:DOSL:PathName	omfi:String	MS DOS pathname for raw data file or OMFI file containing the media data.

Description

The DOS Locator (DOSL) provides a DOS pathname that contains a hint to help an application find the OMFI file or raw data file containing the media data.

The Source Mob (SMOB) describing the digital media data HAS a Media Descriptor object that optionally HAS a set of Locators.

Related Classes

Locator (LOCR), Mac Locator (MACL), Network Locator (NETL), Text Locator (TXTL), UNIX Locator (UNXL), Windows Locator (WINL)

Edgecode Class (ECCP)

Stores film edge code information.

Data Model

Edge Code Class (ECCP) Is-a-Kind-of Segment

`Start`
`FilmKind`
`CodeFormat`

Implementation

Property Name	Type	Explanation
`OMFI:OOBJ:ObjClass`	`omfi:ClassID`	Class is ECCP.
`OMFI:CPNT:DataKind`	`omfi:ObjRef`	HAS-REFERENCE to a Data Definition object that specifies the data kind `omfi:data:Edgecode`.
`OMFI:CPNT:Length`	`omfi:Length32` `omfi:Length64`	Duration of contiguous edge code values.
`OMFI:ECCP:Start`	`omfi:Position32` `omfi:Position64`	Specifies the edge code at the beginning of the segment.
`OMFI:ECCP:FilmKind`	`omfi:FilmType`	Specifies the type of film; one of these: 0 FT_NULL 1 FT_35MM 2 FT_16MM 3 FT_8MM 4 FT_65MM
`OMFI:ECCP:CodeFormat`	`omfi:EdgeType`	Specifies the edge code format; one of these: 0 ET_NULL 1 ET_KEYCODE 2 ET_EDGENUM4 3 ET_EDGENUM5
`OMFI:ECCP:Header`	`omfi:DataValue`	Specifies the text prefix that identifies the film. Typically, this is a text string with no more than 8 characters; optional.

Description

An Edge Code object typically appears in a Mob Slot in a Source Mob that describes a film source. For a film source where the edge codes are contiguous

for the entire source, the Mob Slot should contain a single Edge Code object with a length equal to the length of the Source Clip in the video Mob Slot. For a film source where the edge codes are not contiguous, the Mob Slot should contain a Sequence object that contains a series of Edge Code objects.

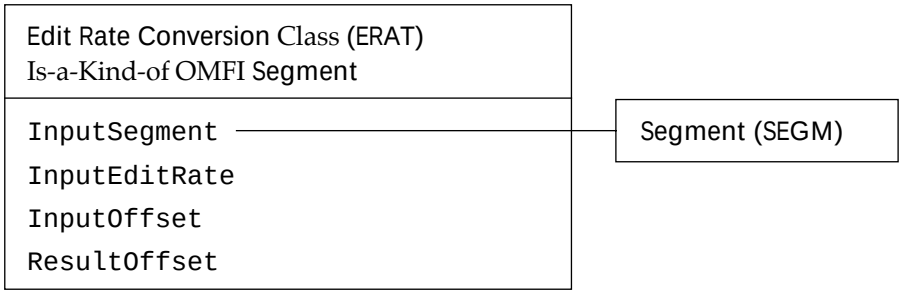
Related Classes

Segment (SEGM), Source Mob (SMOB), Timecode (TCCP)

Edit Rate Converter Class (ERAT)

Converts from one edit rate to another.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is ERAT.
OMFI:CPNT:DataKind	omfi:ObjRef	HAS-REFERENCE to a Data Definition object that specifies the data kind of the Edit Rate Converter.
OMFI:CPNT:Length	omfi:Length32 omfi:Length64	Duration of the Edit Rate Converter in output edit units.
OMFI:ERAT:InputSegment	omfi:ObjRef	HAS a Segment that is to be converted from input edit rate to output edit rate.
OMFI:ERAT:InputEditRate	omfi:Rational	Edit rate of the Segment being converted.
OMFI:ERAT:InputOffset	omfi:Position32 omfi:Position64	Offset in input edit units from an arbitrary synchronization point to the start of the input Segment.
OMFI:ERAT:ResultOffset	omfi:Position32 omfi:Position64	Offset in output edit units from the same arbitrary synchronization to the start of the input Segment.

Rule

The span of time represented by the input Segment must completely include the span of time represented by the Edit Rate Converter object.

Description

An Edit Rate Converter object converts part of a Segment in one edit rate into a Segment in another edit rate. You may want to use an Edit Rate Converter when you want to make an exact copy of a Segment that is in another edit rate and it is important to avoid the rounding errors caused by edit rate conversions.

Edit rate conversions can also be performed by the following mechanisms:

- Scope References to a Mob Slot with a different edit rate
- Source Clip reference to a Mob Slot with a different edit rate

Related Classes

Scope Reference (SREF), Segment (SEGM), Source Clip (SCLP)

Effect Definition Class (EDEF)

Identifies an effect and allows it to be referenced within an OMFI file.

Data Model

Effect Definition Class (EDEF) Is-a-Kind-of OMFI Object

`EffectID`
`EffectName`
`EffectDescription`
`Bypass`
`IsTimeWarp`

Implementation

Property Name	Type	Explanation
`OMFI:OOBJ:ObjClass`	`omfi:ClassID`	Class is EDEF.
`OMFI:EDEF:EffectID`	`omfi:UniqueName`	Specifies the unique name registered for the effect; for example, `omfi:effect:VideoSpeedControl`.
`OMFI:EDEF:EffectName`	`omfi:String`	Name for displaying to users. Optional.
`OMFI:EDEF:EffectDescription`	`omfi:String`	Description for displaying to users. Optional.
`OMFI:EDEF:Bypass`	`omfi:ArgIDType`	The Argument ID value that identifies the primary input slot. Optional.
`OMFI:EDEF:IsTimeWarp`	`omfi:Boolean`	Specifies that the duration for the Effect Slots can be different from the duration of the Effect Invocation. Optional.

Description

An Effect Definition object identifies an effect by its effect ID, a unique effect name. For registered effect IDs, you can use the effect documentation available separately from the OMF Developers' Desk. Effect documentation for private effects should be available from the organization that defined the effect.

The effect documentation provides a brief description of how the effect should modify or combine media and of the purpose of the data in the Effect Invocation slots. It lists the argument ID values allowed in an Effect Invocation and the data kind and meaning of the time-varying data in the Effect Slot. It can also specify the allowed range of values for a Effect Slot with the argument ID value. This information is *not* encoded in the Effect Definition itself.

The `EffectName` and `EffectDescription` properties are intended for display to end users to help them choose a substitute for unknown effects. The `Bypass` property is intended to provide applications a way to play effects that they do not know or cannot generate.

Each Effect Invocation object HAS-REFERENCE to an Effect Definition. If your application is processing an Effect Invocation object and has code to handle the effect specified by the effect ID unique name, it does not need to examine any of the other properties of the Effect Definition object—this information is described in the effect documentation and cannot be changed.

All Effect Definition objects in an OMFI file are part of the Header `DefinitionObjects` property.

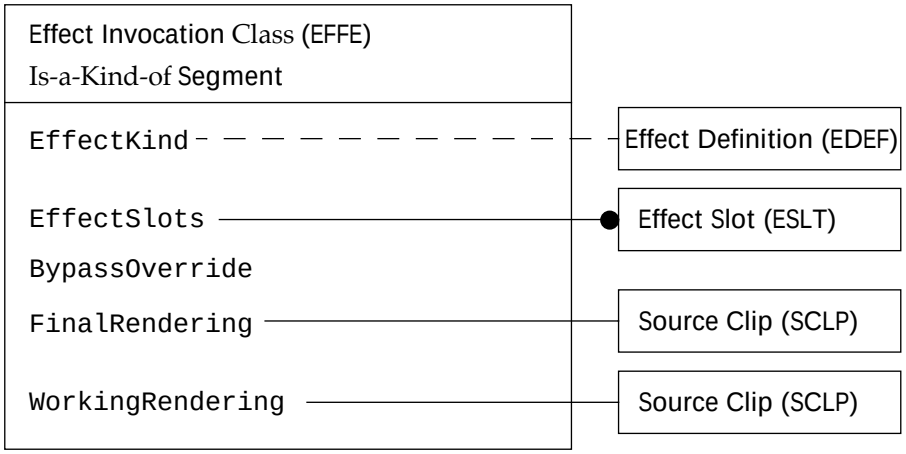
Related Classes

Effect Invocation (EFFE), Effect Slot (ESLT), Header (HEAD)

Effect Invocation Class (EFFE)

Specifies an effect in a Composition Mob.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is EFFE.
OMFI:CPNT:DataKind	omfi:ObjRef	HAS-REFERENCE to a Data Definition object that specifies the data kind of the Effect Invocation. Media effects typically have a data kind of: omfi:data:Picture omfi:data:PictureWithMatte omfi:data:Sound omfi:data:StereoSound
OMFI:CPNT:Length	omfi:Length32 omfi:Length64	Specifies the duration of the effect in edit units.
OMFI:EFFE:EffectKind	omfi:ObjRef	HAS-REFERENCE to an Effect Definition that identifies the kind of effect with a unique name.
OMFI:EFFE:EffectSlots	omfi:ObjRefArray	HAS a set of Effect Slots containing the input media Segments and the effect control argument Segments. Optional.
OMFI:EFFE:BypassOverride	omfi:ArgIDType	Specifies the ArgID value of the input media Effect Slot to be substituted for the Effect Invocation if the application cannot generate the effect. Overrides the bypass specified in the Effect Definition. Optional.

Property Name	Type	Explanation
OMFI:EFFE:FinalRendering	omfi:ObjRef	HAS a Source Clip that contains a rendered version of the effect intended for final use. Optional.
OMFI:EFFE:WorkingRendering	omfi:ObjRef	HAS a Source Clip that contains a rendered version of the effect intended for viewing during editing but not intended for final use. Optional.

Rules

1. In all Effect Invocation objects, the length of the FinalRendering Source Clip and the length of the WorkingRendering Source Clip must each equal the length of the Effect Invocation.
2. In Effect Invocation objects whose Effect Definition object does not specify a time warp, the length of the Segment in each Effect Slot must each equal the length of the Effect Invocation.

Description

An Effect Invocation object specifies the kind of effect, the input media Segments, and the values of control arguments that specify how the effect should be applied. The EffectSlots property contains the input media Segments and values of control arguments.

Effect Definition

The Effect Invocation EffectKind property HAS-REFERENCE to an Effect Definition object that specifies the effect ID, the unique name of the effect. For registered effect IDs, you can use the effect documentation available separately from the OMF Developers' Desk. Effect documentation for private effects should be available from the organization that defined the effect. This effect documentation provides a brief description of how the effect should modify or combine media and of the meaning of effect control arguments in the Effect Invocation object. This information is *not* encoded in the Effect Definition object.

If your application is processing an Effect Invocation object and has code to handle the effect specified by the effect ID unique name, it does not need to examine any of the other properties of the Effect Definition object—this information is described in the effect documentation and cannot be changed.

The effect documentation provides a brief description of how the effect should modify or combine media and of the purpose of the data in the Effect Invocation slots. It lists the argument ID values allowed in an Effect Invocation and the data kind and meaning of the time-varying data in the Effect Slot. It can also specify the allowed range of values for a Effect Slot for each argument ID value.

Effect Slots

The `EffectSlots` property contains both input media Segments and control arguments. Input media Segments provide the media that is altered or combined to produce the intended effect. Control arguments contain values that specify parameters for adjustments in the way the effect should be performed. The Effect Slot `ArgID` property contains an integer that identifies the purpose of the slot. The effect documentation lists all valid argument ID integers and their meaning for an effect.

An effect can have constant control arguments or have control arguments whose values vary over time. For example, a picture-in-picture effect where the size and transparency of the inserted picture stays constant throughout the effect has constant control arguments, but a picture-in-picture effect that starts with a small inserted picture that grows larger during the effect has control arguments with time-varying values.

A constant control argument can be specified with a Constant Value (CVAL) object in an Effect Slot. A time-varying value is specified with a Varying Value (VVAL) object in an Effect Slot.

When an effect specifies time-varying control values, it can also specify how to interpolate these values. Interpolation is a mathematical method to calculate a value for a point based on the values of two surrounding points. In effects, you can use interpolation to find the value for control arguments for any point in time in an effect or transition object by using the values that are specified.

Rendered Effects

When an application creates an Effect Invocation object in an OMFI file, it can also include rendered versions of the effect. A rendered version is media data that can be played to produce the effect. An application can produce a working rendering, a final rendering, or both. A working rendering is intended to store a partially rendered or approximately rendered implementation of the effect but is not intended for final production. A final rendering of the effect *is* intended for final production; it can also be used to play the effect during editing.

If there are more than one implementation of a rendered effect, you should use a Media Group object in the Master Mob.

Dealing with Unrecognized Effects

If an application importing an OMFI file encounters an unknown effect, it can preserve the Effect Invocation and Effect Definition objects so that if the Composition Mob containing the effect is exported, the effect will be preserved. If any application cannot preserve the information, it should inform the user that some effect information is being lost.

If an application is trying to play an unknown effect, the OMF Developers' Desk recommends that it perform the first action in the following list that applies:

1. If there is a rendered version of the effect, play the rendered version.
2. If the Effect Invocation specifies a `BypassOverride` property, play the media in the Effect Slot with the specified argument ID.
3. If the Effect Definition specifies a `Bypass` property, play the media in the Effect Slot with the specified argument ID.
4. Play blank media for the effect.

Related Classes

Constant Value (CVAL), Control Point (CTLP), Data Definition (DDEF), Effect Definition (EDEF), Effect Slot (ESLT), Media Group (MGRP), Segment (SEGM), Source Clip (SCLP), Varying Value (VVAL)

Effect Slot Class (ESLT)

Specifies an input media segment or a control argument for an effect.

Data Model

Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is ESLT.
OMFI:ESLT:ArgID	omfi:ArgIDType	Integer that identifies the role of the slot in the effect. The meaning of the integer is described in the effect's documentation.
OMFI:ESLT:ArgValue	omfi:ObjRef	HAS a Segment that contains the input media or the control argument values.

Description

An Effect Slot object can contain either input media segments or control arguments. Input media segments provide the media that is altered or combined to produce the intended effect. Control arguments contain values that specify parameters for adjustments in the way the effect should be performed. The ArgID property has an integer value that identifies the purpose of the slot. The effect documentation lists all valid argument ID integers and their meaning for an effect.

The effect documentation provides a brief description of how the effect should modify or combine media and of the purpose of the data in the Effect Invocation slots. It lists the argument ID values allowed in an Effect Invocation and the data kind and meaning of the time-varying data in the slot. It can also specify the allowed range of values for a slot. This information is *not* encoded in the Effect Definition itself.

An effect can have constant control arguments or have control arguments whose values vary over time. For example, a picture-in-picture effect where the size and transparency of the inserted picture stays constant throughout the effect has constant control arguments, but a picture-in-picture effect that starts with a small inserted picture that grows larger during the effect has control arguments with time-varying values.

A constant control argument can be specified with a Constant Value (CVAL) object in an Effect Slot. A time-varying value is specified with a Varying Value (VVAL) object in an Effect Slot.

Related Classes

Constant Value (CVAL), Control Point (CTLP), Data Definition (DDEF), Effect Definition (EDEF), Effect Invocation (EFFE), Source Clip (SCLP), Varying Value (VVAL)

Filler Class (FILL)

Represents an unknown or unspecified value for the duration of the object.

Data Model

Filler Class (FILL) Is-a-Kind-of Segment
no additional properties

Implementation

Property Name	Type	Explanation
`OMFI:OOBJ:ObjClass`	`omfi:ClassID`	Class is FILL.
`OMFI:CPNT:DataKind`	`omfi:ObjRef`	HAS-REFERENCE to a Data Definition object that specifies the data kind of the unspecified value. A Filler object can have any data kind.
`OMFI:CPNT:Length`	`omfi:Length32` `omfi:Length64`	Specifies the duration of the unspecified value.

Description

A Filler object is a placeholder for an unknown value for the component's duration.

Typically, a Filler object is used in a Sequence to allow positioning of a Segment when not all of the preceding material has been specified. Another typical use of Filler objects is to fill time in Mob Slots and Nested Scope `Slots` that are not referenced or played.

If a Filler object is played, applications can choose any appropriate blank media to play. Typically, a video Filler object would be played as a black section, and an audio Filler object would be played as a silent section.

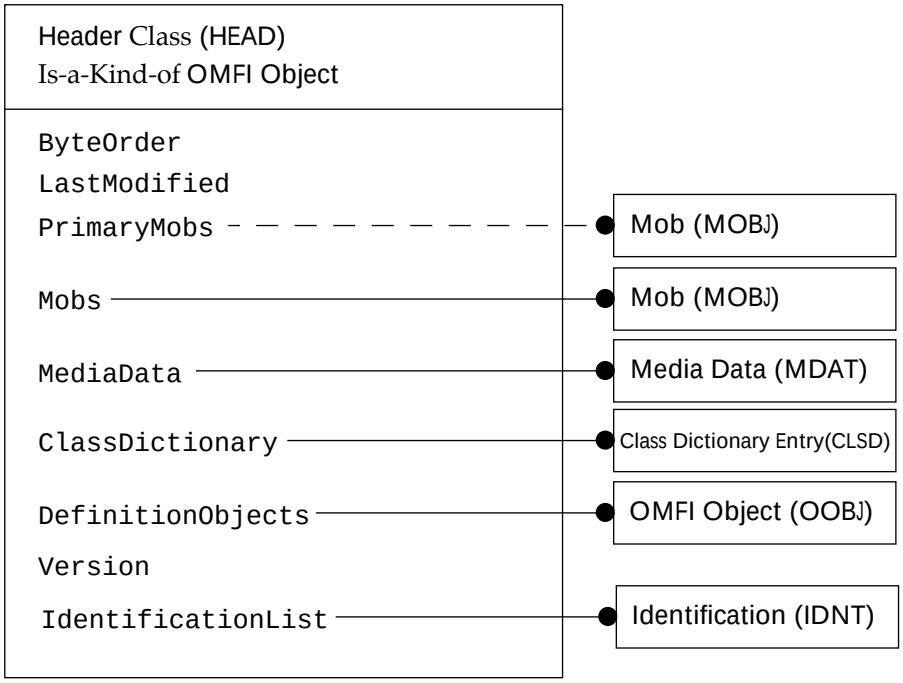
Related Classes

Component (CPNT), Scope Reference (SREF), Segment (SEGM)

Header Class (HEAD)

Provides file-wide information and indexes to the OMF Interchange file.

Data Model



Implementation

Property Name	Type	Explanation
`OMFI:OOBJ:ObjClass`	`omfi:ClassID`	Class is HEAD.
`OMFI:HEAD:ByteOrder`	`omfi:Int16`	Specifies the byte order for the OMFI file. One of the following: `'II'` Little-endian byte order `'MM'` Big-endian byte order
`OMFI:HEAD:LastModified`	`omfi:TimeStamp`	Time and Date the file was last modified.
`OMFI:HEAD:PrimaryMobs`	`omfi:ObjRefArray`	HAS-REFERENCE to a set of Mobs that should be examined first to determine contents of OMFI file. Optional.
`OMFI:HEAD:Mobs`	`omfi:ObjRefArray`	HAS a set of all Mobs defined in the OMFI file.
`OMFI:HEAD:MediaData`	`omfi:ObjRefArray`	HAS a set of all Media Data objects defined in the OMFI file.
`OMFI:HEAD:ClassDictionary`	`omfi:ObjRefArray`	HAS a set of Class Dictionary Entry objects that extend the OMFI class hierarchy. Optional.

Property Name	Type	Explanation
OMFI:HEAD:DefinitionObjects	omfi:ObjRefArray	HAS a set of all Data Definition and Effect Definition objects defined in the OMFI file.
OMFI:HEAD:Version	omfi:VersionType	OMF Interchange specification version number that the file conforms to; must be 2.0 or higher.
OMFI:HEAD:IdentificationList	omfi:ObjRefArray	HAS an ordered set of Identification objects, which identify the application that created or modified the OMF file. Optional.

Rules

1. Each OMF Interchange file must have exactly one Header object.
2. All mobs defined in the OMFI file must be included in the Mobs property set of mobs.
3. All Media Data objects in the OMFI file must be included in the MediaData set of Media Data objects.
4. Only Data Definition (DDEF) and Effect Definition (EDEF) objects can be included in the DefinitionObjects set.

Description

The Header object stores information about the OMFI file as a whole, such as byte order, Mob indexes, Media Data index, version number, and information about the applications that created or modified the file.

Byte Order

The value of the ByteOrder property is either 'MM' (hexadecimal 0x4d4d) for big-endian byte order, which is used in some architectures such as the Motorola 680x0 architecture, or 'II' (hexadecimal 0x4949) for little-endian byte order, which is used in some architectures such as the Intel x86 architecture.

Big-endian and little-endian refer to whether the most- or least-significant byte is stored first. In the big-endian byte order, the most-significant byte is stored first (at the address specified, which is the lowest address of the series of bytes that constitute the value). In the little-endian byte order, the least-significant byte is stored first. In both cases, bytes are stored with the most-significant bit first.

Modification Date

The value of LastModified represents the last time the file was modified.

Class Dictionary Property

The `ClassDictionary` property defines file-wide extensions to the OMFI class hierarchy. It HAS a set of Class Dictionary objects. These objects are not required to define the standard class dictionary; they are required only for any extensions an application is adding for public or private objects.

The `ClassDictionary` property of the Header object provides a mechanism for adding object classes to the standard set. Compliance with the standard objects is assumed for all applications and no `ClassDictionary` property is required for it. This property enables OMF Interchange readers to handle unknown objects.

Mob Indexes

Since references to Mobs are by Mob ID and not by object references, you need the set of all mobs in the file provided in the Mobs property. This property has the object references for all the Mobs in the OMFI file. You can examine each Mob to find its Mob ID and whether it is a Composition Mob, Master Mob, or Source Mob.

The `PrimaryMobs` index lists the Mobs that you should examine first. If the OMFI file represents a single composition, the Composition Mob for it should be in the `PrimaryMobs` index. In addition, you may wish to include any other Mobs that you intend to be referenced independently of the single composition.

The primary Mobs are the Mobs that the file is intending to communicate; all other Mobs are present because the primary Mobs reference them. If an OMFI file has no primary mobs, it is providing a repository for its Mobs but is not specifying which one should be used.

Media Data Index

Since Mobs are associated with media objects by matching Mob ID values and not by object references, you need the `MediaData` property to be able to access all of the Media Data objects in the file. You can examine each Media Data object to find its Mob ID and to determine the subclass of Media Data that it belongs to.

Definition Objects

This index lists all Data Definition and Effect Definition objects in the OMFI file. Although these objects are referenced by other OMFI objects, they do not belong to any tree structure. This index is provided to improve efficiency in handling these definition objects.

Version Number

The `Version` property specifies the version of OMF Interchange used to create the file. In the data type for this property, the first byte indicates the major revision number. The second byte indicates the minor revision number. Any OMF Interchange file conforming to this version of the OMF Interchange specification must have a `VersionNumber` with the first byte having a value of 2.

In OMF Interchange Version 1.0, the `VersionNumber` property was optional. Any OMFI file that omits the Header `VersionNumber` must be an OMFI Version 1.0 file.

Identification List

The `IdentificationList` property identifies the application that created the OMF file and, optionally, other applications that have modified the OMF file. The identification list consists of an ordered set of Identification (IDNT) objects. The first Identification object describes the application that created the OMF file. Any following Identification objects describe applications that have modified the OMF file.

OMFI Tree Structure

In the same way that a mob consists of a tree structure, the entire OMFI file consists of a tree structure with the Header object being the root of the tree. The `ClassDictionary`, `Mobs`, `MediaData`, and `DefinitionObjects` properties represent the major branches of the tree. By following the branches of this tree, you can find all the required OMFI objects in the file. Note that, when following the tree, you should not follow any property that has a reference to another object (indicated by dashed lines in the Data Model syntax). It is possible that there are private objects in the OMFI file that are not included in this tree.

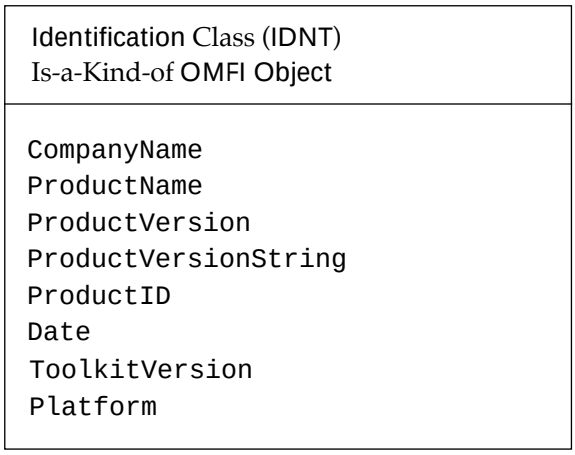
Related Classes

Class Dictionary Entry (CLSD), Composition Mob (CMOB), Data Definition (DDEF), Effect Definition (EDEF), Master Mob (MMOB), Media Data (MDAT), Mob (MOBJ), Source Mob (SMOB)

Identification Class (IDNT)

Provides information about the application that created or modified the OMF file.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is IDNT.
OMFI:IDNT:CompanyName	omfi:String	Specifies the name of the company or organization that created the application. Optional.
OMFI:IDNT:ProductName	omfi:String	Specifies the name of the application. Optional.
OMFI:IDNT:ProductVersion	omfi:ProductVersion	Specifies the version number of the application. Consists of 5 16-bit integer values that specify the version of an application. The first four integers specify the major, minor, tertiary, and patch version numbers. The fifth integer has the following values: 0 kVersionUnknown: No additional version information 1 kVersionReleased: Released product 2 kVersionDebug: Development version 3 kVersionPatched: Released product with patches 4 kVersionBeta: Prerelease beta test version 5 kVersionPrivateBuild: Optional.
OMFI:IDNT:ProductVersion-String	omfi:String	Specifies the version number of the application in string form. Optional.

Property Name	Type	Explanation
OMFI:IDNT:ProductID	omfi:Int32	Specifies the identification number assigned to the application and vendor. This number specifies the first integer field in MobIDs for Mobs created by the application.
OMFI:IDNT:Date	omfi:TimeStamp	Time and date the application created or modified the OMF file.
OMFI:IDNT:ToolkitVersion	omfi:ProductVersion	Specifies the version number of the OMF Toolkit library. Optional.
OMFI:IDNT:Platform	omfi:String	Specifies the platform on which the application is running. Optional.

Description

The Identification (IDNT) class identifies the application that created or modified the OMF file.

Related Classes

Header (HEAD)

Image Data Class (IDAT)

Contains image media data.

Data Model

Image Data Class (IDAT) Is-a-Kind-of Media Data
`ImageData`

Implementation

Property Name	Type	Explanation
`OMFI:OOBJ:ObjClass`	`omfi:ClassID`	Class is IDAT.
`OMFI:MDAT:MobID`	`omfi:UID`	MobID of source mob that describes media data.
`OMFI:IDAT:ImageData`	`omfi:DataValue`	Media data stream.

Description

An Image Data object contains a stream of image media data. It has the same `MobID` as a Source Mob object in the OMFI file. This Source Mob describes the format of the image media. The required OMFI formats for image media stored in Image Data objects are Color Difference Component Image Descriptor and RGBA Component Image Descriptor.

TIFF image media data is not stored in an Image Data object but is stored instead in a TIFF Image Data object. The TIFF format is included for compatibility with OMF Interchange Version 1.0.

Related Classes

Color Difference Component Image Descriptor (CDCI), JPEG Image Media Data (JPEG), Media Data (MDAT), RGBA Component Image Descriptor (RGBA), Source Mob (SMOB), TIFF Image Data (TIFF)

JPEG Image Data Class (JPEG)

Contains image media that has been compressed with the JPEG algorithm.

Data Model

JPEG Image Data Class (JPEG) Is-a-Kind-of Image Data
`FrameIndex`

Implementation

Property Name	Type	Explanation
`OMFI:OOBJ:ObjClass`	`omfi:ClassID`	Class is JPEG.
`OMFI:MDAT:MobID`	`omfi:UID`	Mob ID of Source Mob that describes media data.
`OMFI:IDAT:ImageData`	`omfi:DataValue`	Media data stream.
`OMFI:JPEG:FrameIndex`	`omfi:Position32Array` `omfi:Position64Array`	Array containing an offset into the media stream for each frame in the stream.

Description

A JPEG Image Data object contains digital images stored in variable-length frames; consequently, the JPEG Image Data object includes the `FrameIndex` property that provides the offset for each frame in the stream.

Every JPEG Image Data object in the OMF Interchange file must have a corresponding Source Mob object in the file that has the same value for the `MobID` property. The Source Mob object describes the media in the JPEG Image Data object and allows Composition Mobs to reference it.

The Media Descriptor that describes media stored in a JPEG Image Data object is the Color Difference Component Image Descriptor (CDCI).

TIFF image media data that has been compressed by using the JPEG algorithm is not stored in a JPEG Image Data object but is stored instead in a TIFF Image Media Data object. The TIFF format is included for compatibility with OMF Interchange Version 1.0.

Related Classes

Color Difference Component Image Descriptor (CDCI), Image Data (IDAT), Media Data (MDAT), RGBA Component Image Descriptor (RGBA), Source Mob (SMOB), TIFF Image Data (TIFF)

Locator Class (LOCR)

Provides information to help find a file that contains the media data.

Data Model

Locator Class (LOCR) Is-a-Kind-of OMFI Object Abstract Class
no additional properties

Implementation

Property Name	Type	Explanation
`OMFI:OOBJ:ObjClass`	`omfi:ClassID`	Class is LOCR.

Description

The Locator (LOCR) class is an abstract class that provides information on finding an OMFI file or a raw data file that contains the media data associated with a source mob. The Locator provides hints to help an application find the file containing the media data.

The Source Mob (SMOB) describing the digital media data HAS a Media Descriptor object that optionally HAS a set of Locators.

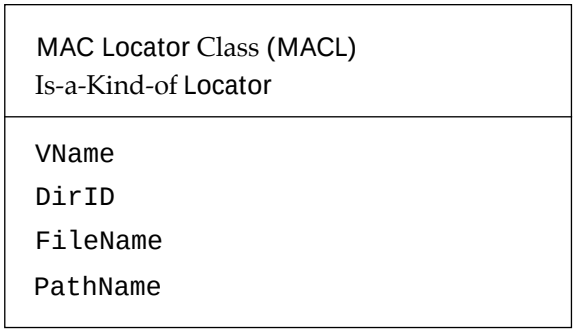
Related Classes

DOS Locator (DOSL), Mac Locator (MACL), Network Locator (NETL),Text Locator (TXTL), UNIX Locator (UNXL), Windows Locator (WINL)

Mac Locator Class (MACL)

Provides information to help find a Mac file containing media data.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is MACL.
OMFI:MACL:VName	omfi:String	Identifies the volume name. Optional.
OMFI:MACL:DirID	omfi:Int32	Identifies the Mac directory. Optional.
OMFI:MACL:FileName	omfi:String	Identifies the filename. Optional.
OMFI:MACL:PathName	omfi:String	Identifies the pathname of the file containing the media data.

Description

The MAC Locator (MACL) provides a Macintosh® volume name, directory, and file name that contain hints to help an application find the OMFI file or raw data file containing the media data.

The Source Mob (SMOB) describing the digital media data HAS a Media Descriptor object that optionally HAS a set of Locators.

Related Classes

DOS Locator (DOSL), Locator (LOCR), Network Locator (NETL), Text Locator (TXTL), UNIX Locator (UNXL), Windows Locator (WINL)

Master Mob Class (MMOB)

Provides an indirect way for a Composition Mob to reference a Source Mob and synchronizes data in multiple Source Mobs.

Data Model

Master Mob Class (MMOB) Is-a-Kind-of Mob
no additional properties

Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is MMOB.
OMFI:MOBJ:MobID	omfi:UID	Unique Mob identification.
OMFI:MOBJ:Name	omfi:String	Name of the mob for display to end user. Optional.
OMFI:MOBJ:Slots	omfi:ObjRefArray	Contains the externally visible Mob Slots and the internal Mob Slots containing media and other time-varying information.
OMFI:MOBJ:LastModified	omfi:TimeStamp	Date and time when the Mob was last modified.
OMFI:MOBJ:CreationTime	omfi:TimeStamp	Date and time when the Mob was originally created.
OMFI:MOBJ:UserAttributes	omfi:ObjRef	Specifies a set of user attributes, which provide additional information about the Mob. The Attribute Array contains a set of Attributes. Optional.

Description

A Master Mob object provides a level of indirection for accessing Source Mobs from Composition Mobs. The media associated with a Source Mob is immutable; consequently, if you must make any changes to the media data, you must create a new Source Mob with a new unique Mob ID. Typical reasons to change the media data include redigitizing to extend the section of the media included in the file, redigitizing to change the compression used to create the digital media data, and redigitizing to change the format used to store the media data, such as from AIFF audio data to WAVE audio data. A Composition Mob may have many Source Clip objects that reference media data—updating every Source Clip in the Composition Mob each time the media was redigitized would be inefficient. By having the Composition Mob only access a

Source Mob through a Master Mob, OMF ensures that you only have to change a single Master Mob when you make changes to the media data.

In addition, a Master Mob can synchronize media data in different Source Mobs. For example, when an application digitizes a videotape, it creates separate Source Mobs for the video and audio data. By having a single Master Mob with one Mob Slot for each Source Mob, the Composition Mob avoids having to synchronize the audio and video tracks each time it references media from different tracks of the videotape.

If there are multiple implementations of digitized media, the Master Mob can contain a Media Group (MGRP) object. The Media Group object contains a set of Source Clip objects, each of which identifies a Source Mob containing a different implementation of the media data. An application can examine these implementations to find the one that it is able to play or that it can play most efficiently. Media Groups may be needed if you have systems with different architectures or compression hardware accessing a single OMFI file.

If when a media data file is redigitized and it has to be broken into multiple files, this can be represented by a Sequence object in the Master Mob that contains a series of Source Clip objects, each identifying the Source Mob associated with one of the files.

Typically, Master Mobs have a very simple structure. They have a Mob Slot with a Track Description for each track of media and do not have any other slots. Typically, each Mob Slot contains a single Source Clip object that identifies the Source Mob. Master Mobs cannot have Effect Invocation objects, Nested Scope objects, Selector objects, Edit Rate Converter objects, or Transition objects.

The following lists the reasons for having a Master Mob slot contain an object other than a Source Clip:

- If there are multiple implementations of the same media, the Mob Slot can contain a Media Group instead of a Source Clip object.
- If the media source has been broken into several source mobs, the Mob Slot can contain a Sequence object. The Sequence object cannot contain any Component other than a Source Clip object or a Media Group object.

Related Classes

Composition Mob (CMOB), Header (HEAD), Master Mob (MMOB), Media Group (MGRP), Mob (MOBJ), Mob Slot (MSLT), Source Clip (SCLP), Source Mob (SMOB), Track Description (TRKD)

Media Data Class (MDAT)

Contains digital media data.

Data Model

Media Data Class (MDAT) Is-a-Kind-of OMFI Object Abstract class
`MobID`

Implementation

Property Name	Type	Explanation
`OMFI:OOBJ:ObjClass`	`omfi:ClassID`	Class is MDAT.
`OMFI:MDAT:MobID`	`omfi:UID`	MobID of the source mob that describes the media data.

Description

The Media Data class is an abstract class that has digital media data. Any private class that contains digital media data and has a Source Mob associated with it should be a subclass of Media Data.

The Header (HEAD) object contains an index to all Media Data objects in the OMFI file. For each Media Data object in the OMFI file, there is a corresponding Source Mob that specifies the same `MobID` value.

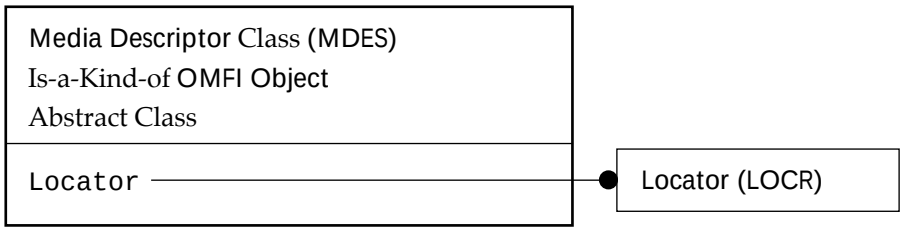
Related Classes

AIFC Audio Data (AIFC), Image Data (IDAT), Media Descriptor (MDES), Source Mob (SMOB), TIFF Image Data (TIFF), WAVE Audio Data (WAVE)

Media Descriptor Class (MDES)

Describes the media associated with a Source Mob.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is MDES.
OMFI:MDES:Locator	omfi:ObjRefArray	HAS a set of Locator objects that contain operating-system-dependent data or text information that provide hints for finding raw data files or OMFI files that contain the media data. Optional.

Description

Media Descriptor is an abstract class that describes the format of the media data. The media data can be digital media data stored in a file, or it can be media data on audio tape, film, videotape, or some other form of media storage.

The Source Mob MediaDescription property has the Media Descriptor that describes the media data. There are two kinds of Media Descriptors:

- Media File Descriptors that describe digitized media stored in raw data files or in OMFI files. The Media File Descriptor class is also an abstract class; its subclasses describe the various formats of digitized media.
- Physical Media Descriptors that describe a physical media source. This specification defines two physical Media Descriptors: Media Film Descriptor and Media Tape Descriptor, but additional private or registered subclasses can be defined.

The Locator objects provide hints to help find media data files associated with the Source Mob, but they are only hints because their correctness cannot be guaranteed, since users may rename or delete files. Typically, this can happen if the OMFI file is renamed after being created. If your application cannot find an OMFI file by using the hint, it can search through all accessible OMFI files to find the media data object with the MobID value.

A Media Descriptor may have more than one Locator object if the file may be accessed from more than one operating system or for any reason that multiple Locators may make it more likely that the application can find the file.

Related Classes

Locator (LOCR), Media File Descriptor (MDFL), Media Film Descriptor (MDFM), Media Tape Descriptor (MDTP), Source Mob (SMOB)

Media File Descriptor Class (MDFL)

Describes digital media data associated with a Source Mob.

Data Model

Media File Descriptor Class (MDFL) Is-a-Kind-of Media Descriptor Abstract Class
IsOMFI SampleRate Length

Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is MDFL.
OMFI:MDES:Locator	omfi:ObjRefArray	HAS a set of Locator objects that contain operating-system-dependent data or text information that provide hints for finding raw data files or OMFI files that contain the media data. Optional.
OMFI:MDFL:IsOMFI	omfi:Boolean	A True value indicates that the media data is stored in an OMFI file; a False value indicates that the media data is stored in a raw data file.
OMFI:MDFL:SampleRate	omfi:Rational	The native sample rate of the digitized media data.
OMFI:MDFL:Length	omfi:Length32 omfi:Length64	Duration of the media in sample units.

Description

The Media File Descriptor class is an abstract class that describes the format of digital media data; its subclasses describe the various formats of digitized media. The Media File Descriptor class has the following subclasses defined in this specification:

- AIFC Audio Descriptor class
- Digital Image Descriptor class, which is itself an abstract class and has the following subclasses:
  - Color Difference Component Image Descriptor class
  - RGBA Component Image Descriptor class
- TIFF Image Descriptor class
- WAVE Audio Descriptor class

If the media data is stored in an OMFI file, the value of the `IsOMFI` property must be true. If the media data is stored in a raw data file, the value of the `IsOMFI` property must be false. Media data can be stored in a raw data file to allow an application that does not support OMFI to access it or to avoid duplicating already existing digital media data. However, since there is no Mob ID stored with raw media data, it is difficult to identify a raw media data file if the Locator information is no longer valid. The format of the media data in the raw file is the same as it would be if it were stored in an OMFI Media Data object.

The Media File Descriptor specifies the sample rate and the length in the sample rate of the media data. The sample rate of the data can be different from the edit rate of the Source Clip object that references it.

Related Classes

AIFC Audio Descriptor (AIFD), Color Difference Component Image Descriptor (CDCI), Digital Image Descriptor (DIDD), Locator (LOCR), Media Descriptor (MDES), RGBA Component Image Descriptor (RGBA), Source Mob (SMOB), TIFF Image Descriptor (TIFD), WAVE Audio Descriptor (WAVD)

Media Film Descriptor Class (MDFM)

Describes film media.

Data Model

Media Film Descriptor Class (MDFM) Is-a-Kind-of Media Descriptor

`FilmFormat`
`FrameRate`
`PerforationsPerFrame`
`FilmAspectRatio`
`Manufacturer`
`Model`

Implementation

Property Name	Type	Explanation
`OMFI:OOBJ:ObjClass`	`omfi:ClassID`	Class is MDFM.
`OMFI:MDES:Locator`	`omfi:ObjRefArray`	Text locators contain hints to indicate where the film is stored. Optional.
`OMFI:MDFM:FilmFormat`	`omfi:FilmType`	Identifies the format of the film; one of the following: 0 FT_NULL 1 FT_35MM 2 FT_16MM 3 FT_8MM 4 FT_65MM Optional.
`OMFI:MDFM:FrameRate`	`omfi:UInt32`	Specifies the frame rate in frames per second. Optional.
`OMFI:MDFM:PerforationsPerFrame`	`omfi:UInt8`	Specifies the number of perforations per frame on the film stock. Optional.
`OMFI:MDFM:FilmAspectRatio`	`omfi:Rational`	Ratio between the horizontal size of the frame image and the vertical size of the frame image. Optional.
`OMFI:MDFM:Manufacturer`	`omfi:String`	A string to display to end users, indicating the manufacturer of the film. Optional.
`OMFI:MDFM:Model`	`omfi:String`	A string to display to end users, indicating the manufacturer's brand designation for the film. Optional.

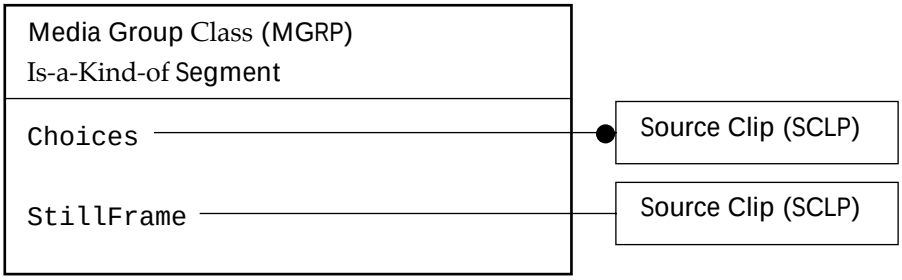
Related Classes

Locator (LOCR), Media Descriptor (MDES), Source Mob (SMOB)

Media Group Class (MGRP)

Provides alternative digital media representations in a Master Mob.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is MGRP.
OMFI:CPNT:DataKind	omfi:ObjRef	HAS-REFERENCE to Data Defintion object that specifies the data kind of the media.
OMFI:CPNT:Length	omfi:Length32 omfi:Length64	Duration of the media in edit units.
OMFI:MGRP:Choices	omfi:ObjRefArray	HAS a set of Source Clips that identify the alternate representations that may be chosen.
OMFI:MGRP:StillFrame	omfi:ObjRef	HAS a Source Clip that identifies the media for a single-frame image representation of the media. Optional.

Rules

1. A Media Group object can only be used in a Mob Slot in a Master Mob.
2. The length of each Source Clip in the Choices set must be the same as the length of the Media Group object.
3. The length of the StillFrame Source Clip must be 1.

Description

A Media Group object provides a way to access different implementations of the same media. A Media Group can only occur in a Mob Slot of a Master Mob. Typically the different implementations vary in media format, compression, or frame size. The application is responsible for choosing the appropriate implementation of the media.

Related Classes

Master Mob (MMOB), Segment (SEGM), Source Clip (SCLP), Source Mob (SMOB)

Media Tape Descriptor Class (MDTP)

Describes audio tape or videotape media.

Data Model

Media Tape Descriptor Class (MDTP) Is-a-Kind-of Media Descriptor
FormFactor VideoSignal TapeFormat Length Manufacturer Model

Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is MDTP.
OMFI:MDES:Locator	omfi:ObjRefArray	Text Locators provide hints to help find the physical location of the master videotape. Optional.
OMFI:MDTP:FormFactor	omfi:TapeCaseType	Describes the physical size of the tape; may have one of the following values: 0 3/4 inch videotape 1 VHS video tape 2 8mm videotape 3 Betacam videotape 4 Compact cassette 5 DAT cartridge 6 Nagra audio tape Optional.
OMFI:MDTP:VideoSignal	omfi:VideoSignalType	Describes the video signal type; may have one of the following values: 0 NTSC 1 PAL 2 SECAM Optional.

Property Name	Type	Explanation
OMFI:MDTP:TapeFormat	omfi:TapeFormatType	Describes the format of the tape; may have one of the following values: 0 Betacam 1 BetacamSP 2 VHS 3 S-VHS 4 8mm 5 Hi8 Optional.
OMFI:MDTP:Length	omfi:UInt32	Tape capacity in minutes. Optional.
OMFI:MDTP:Manufacturer	omfi:String	Text string to display to end users, identifying the manufacturer of the tape. Optional.
OMFI:MDTP:Model	omfi:String	Text string to display to end users, identifying the manufacturer's brand designation of the tape. Optional.

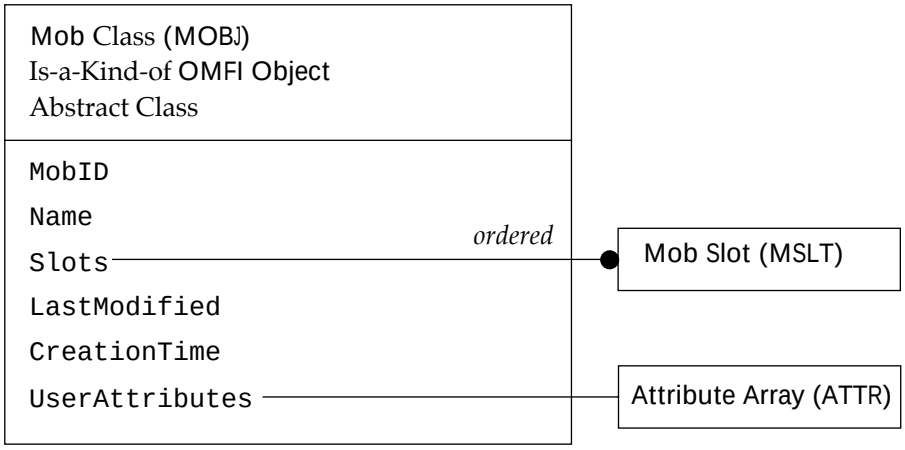
Related Classes

Locator (LOCR), Media Descriptor (MDES), Source Mob (SMOB)

Mob Class (MOBJ)

Describes editing information or media.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is MOBJ.
OMFI:MOBJ:MobID	omfi:UID	Unique Mob Identification.
OMFI:MOBJ:Name	omfi:String	Name of mob for display to end user. Optional.
OMFI:MOBJ:Slots	omfi:ObjRefArray	HAS a set of Mob Slots containing media and other time-varying information.
OMFI:MOBJ:LastModified	omfi:TimeStamp	Date and time when the Mob was last modified.
OMFI:MOBJ:CreationTime	omfi:TimeStamp	Date and time when the Mob was originally created.
OMFI:MOBJ:UserAttributes	omfi:ObjRef	Specifies a set of user attributes, which provide additional information about the Mob. The Attribute Array contains a set of Attributes. Optional.

Description

The Mob class is an abstract class that describes editing information or media. There are several different kinds of Mobs used in OMF Interchange files, but all Mobs describe time-varying media information.

Mobs have a globally unique ID, and they are the only elements of an OMF file that can be referenced from outside the file. It is possible to copy Mobs from

one file to another file, but they must be copied in their entirety. It is not possible for a single Mob to contain OMF objects that are in different files.

The Mob class has the following subclasses:

- Composition Mobs, which describe editing information
- Master Mobs, which synchronize Source Mobs and provide a layer of indirection to make it easy to change Source Mobs without changing Composition Mobs that reference them
- Source Mobs, which describe media
  - File Source Mobs, which describe digital media stored in files
  - Physical Source Mobs, which describe physical media such as videotape, audio tape, and film

Mob Slots and Scope

Mob Slots contain segments of media or other time-varying data. A Mob Slot can contain an externally visible segment or an externally hidden segment. Mob Slots that are externally visible by definition have a Track Description; these Mob Slots represent outputs of the Mob. All Mob Slots can be referenced by other Mob Slots by using Scope References.

Mob Slots are ordered to allow Scope References within one slot to reference another slot. The Mob defines a scope consisting of the ordered set of Mob Slots. A Scope Reference object in a Mob Slot can refer to any Mob Slot that precedes it. A Scope Reference returns the same time-varying values as the section in the specified Mob Slot that corresponds to the starting point of the Scope Reference in the Mob Slot and the duration of the Scope Reference. In addition to Mobs, Nested Scope objects define scopes; however, their scope is limited to the Components contained within the Nested Scope object's slots.

User Information

The user can specify a name and additional information to identify and provide information about the Mob. The `Name` property contains the name specified by the user and the `UserAttributes` property contains additional descriptive information. For each attribute, the user can specify the category name and the value for the category. This mechanism allows the user to define categories and provide values for one or more Mobs in the OMF file. User attributes can be used to specify information such as the take number or lighting conditions of a section of media.

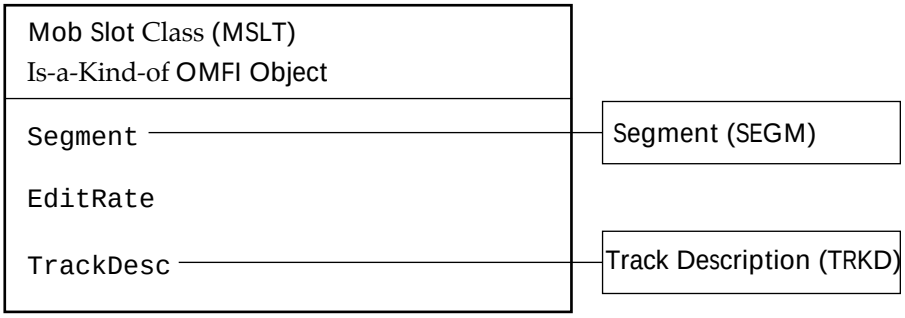
Related Classes

Composition Mob (CMOB), Header (HEAD), Master Mob (MMOB), Mob Slot (MSLT), Scope Reference (SREF), Source Clip (SCLP), Source Mob (SMOB), Track Description (TRKD)

Mob Slot Class (MSLT)

Represents a stream of media in a Mob.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is MSLT.
OMFI:MSLT:Segment	omfi:ObjRef	HAS a Segment that specifies the media or other time-varying data contained in the Mob Slot.
OMFI:MSLT:EditRate	omfi:Rational	Specifies the edit rate for the segment.
OMFI:MSLT:TrackDesc	omfi:ObjRef	HAS a Track Description that specifies that the Mob Slot is externally accessible and identifies it. Optional.

Description

A Mob Slot object contains a Segment; the Mob Slot binds the Segment to real time. The Segment by itself has a virtual duration, expressed in edit units. The Mob Slot defines the edit rate and thus maps the virtual edit units to real time. If the Mob Slot has a Track Description object, it specifies a track ID and external origin to allow Source Clips in other Mobs to reference the Mob Slot .

The Mob Slot specifies the edit rate for the Segment it contains. The Segment specifies its length in the edit rate set by the Mob Slot . The Segment also specifies its own data kind.

A Mob Slot may be externally accessible or may be limited to access from within the same Mob The Track Description specifies that the Mob Slot is externally accessible and specifies the TrackID, TrackLabel, and Origin. These Track Description properties allow the Mob Slot to be referenced from outside the Mob.

Mob Slots that are not externally referenceable are typically included in a Mob to allow more than one other slot to reference the Segment in it. If the referenc-

es to a Component are in more than one Mob Slot, then you must use a Mob Slot to share access. If all of the references to a Component are in the same Mob Slot, then you can use either a Mob Slot or a Nested Scope slot to share access.

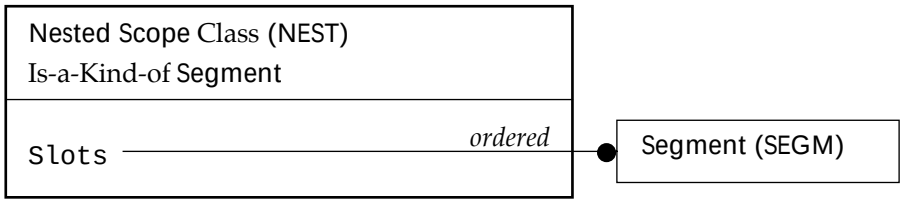
Related Classes

Mob (MOBJ), Nested Scope (NEST), Scope Reference (SREF), Segment (SEGM), Source Clip (SCLP), Track Description (TRKD)

Nested Scope Class (NEST)

Defines a scope that contains an ordered set of Segments that can be referenced from within the scope but that cannot be referenced from outside of the scope.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is NEST.
OMFI:CPNT:DataKind	omfi:ObjRef	HAS-REFERENCE to a Data Definition object that specifies the data kind of the Nested Scope object.
OMFI:CPNT:Length	omfi:Length32 omfi:Length64	Specifies the length of the segment produced by the Nested Scope object.
OMFI:NEST:Slots	omfi:ObjRefArray	HAS an ordered set of Segments; the last segment provides the value for the Nested Scope object.

Rules

1. The length of each Segment object in the set must be equal to the length of the Nested Scope object.
2. The data kind of the last Segment in the set must be the same as or convertible to the data kind of the Nested Scope object.

Description

A Nested Scope object defines a scope that contains an ordered set of Segments and produces the value specified by the last Segment in the ordered set. Nested Scopes are typically included in Composition Mobs to allow more than one Component to share access to a Segment. You can allow this sharing by using a Nested Scope object or by using the scope defined by a Mob.

Scopes and Scope References

The Mob defines a scope consisting of the ordered set of Mob Slots. A Scope Reference object in a Mob Slot can specify any Mob Slot that precedes it. Nest-

ed Scope objects define scopes that are limited to the Components contained within the Nested Scope object's slots.

A Scope Reference object returns the same time-varying values as the corresponding section of the slot that it references. The time-varying values of a section of a slot can be shared by having more than one Scope Reference specify it. A Scope Reference object specifies the slot by specifying relative scope and relative slot. The relative scope specifies whether the slot is in the current scope that contains the Scope Reference, a preceding Nested Scope, or the Mob scope which is the outermost scope. The relative slot specifies one of the slots that precedes the slot containing the Scope Reference within the scope specified by the relative scope.

If a Scope Reference specifies a Mob Slot, the corresponding section of the slot is the time span that has the equivalent starting position from the beginning of the Mob Slot and the equivalent length as the Scope Reference object has within its Mob Slot. If the specified Mob Slot has a different edit rate than the Mob Slot containing the Scope Reference, the starting position and duration are converted to the specified Mob Slot's edit units to find the corresponding section.

If a Scope Reference specifies a Nested Scope slot, the corresponding section of the slot is the one that has the same starting position offset from the beginning of the Segments in the Nested Scope and the same duration as the Scope Reference object.

Related Classes

Mob Slot (MSLT), Scope Reference (SREF), Segment (SEGM)

Network Locator Class (NETL)

Provides information to help find the file containing media data.

Data Model

Network Locator Class (NETL) Is-a-Kind-of Locator
URLString

Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is NETL.
OMFI:NETL:URLString	omfi:String	Universal Resource Locator (URL) for file containing the media data.

Description

The Network Locator (NETL) provides a URL that contains a hint to help an application find the file containing the media data.

The Source Mob (SMOB) describing the digital media data HAS a Media Descriptor object that optionally HAS a set of Locators.

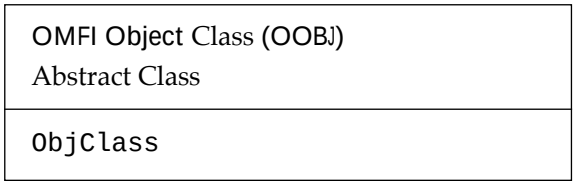
Related Classes

DOS Locator (DOSL), Locator (LOCR), Mac Locator (MACL), Text Locator (TXTL), UNIX Locator (UNXL), Windows Locator (WINL)

OMFI Object Class (OOBJ)

Is the root of the OMF class hierarchy.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is OOBJ.

Description

OMFI Object (OOBJ) is an abstract class, which defines the `ObjClass` property. The `ObjClass` property specifies the class of an object. An object's class is the most specialized class that it belongs to; you can determine the ancestor classes that the object belongs to by examining the class hierarchy.

If you are defining private classes, you should make the private classes descendents of the OMFI Object class.

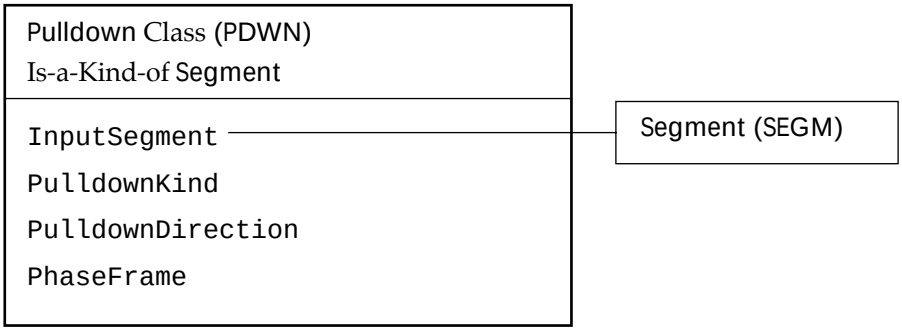
Related Classes

Class Dictionary Entry (CLSD), Header (HEAD)

Pulldown Class (PDWN)

Converts between film-speed and tape-speed.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is PDWN.
OMFI:CPNT:DataKind	omfi:ObjRef	HAS-REFERENCE to Data Defintion object that specifies the data kind of the media.
OMFI:CPNT:Length	omfi:Length32 omfi:Length64	Duration of the media in edit units.
OMFI:PDWN:InputSegment	omfi:ObjRef	HAS a Segment that is either a Source Clip or Timecode. The length of the Source Clip or Timecode is in the edit units determined by the PulldownKind and PulldownDirection.

Property Name	Type	Explanation
OMFI:PDWN:PulldownKind	omfi:PulldownKindType	Specifies whether the Pulldown object is converting from NTSC or PAL video and whether frames are dropped or the video is played at another speed. Values are: 0 kOMFTwoThreePD Converting between NTSC video and film by dropping or adding frames 1 kOMFPalPD Converting between PAL video and film by dropping or adding frames 2 kOMFOneToOneNTSC Converting between NTSC video and film by speeding up or slowing down the frame rate. 3 kOMFOneToOnePAL Converting between PAL video and film by speeding up or slowing down the frame rate. 4 kOMFVideoTapNTSC Converting between NTSC video and film by recording original film and video sources simultaneously.
OMFI:PDWN:PulldownDirection	omfi:PulldownDirectionType	Specifies whether the Pulldown object is converting from tape to film speed or from film to tape speed. Values are: 0 kVideoToFilmSpeed The InputSegment is at video speed and the Mob track containing the PDWN object is at film speed. 1 kFilmToVideoSpeed The InputSegment is at film speed and the Mob track containing the PDWN object is at video speed.
OMFI:PDWN:PhaseFrame	omfi:PhaseFrameType	Specifies the phase within the repeating pulldown pattern of the first frame after the pulldown conversion. A value of 0 specifies that the Pulldown object starts at the beginning of the pulldown pattern.

Description

A Pulldown object provides a mechanism to convert from media between video and film rates and describe the mechanism that was used to convert the media. Pulldown objects are typically used in three ways:

- In a tape Source Mob to describe how the videotape was created from film
- In a file Source Mob that has digital media at film speed to describe how the digital media was created from the videotape

- In a Mob to create Timecodes tracks at different edit rates

A Pulldown object that contains a Source Clip defines a relationship between two Source Mobs, a derived Source Mob and a previous generation Source Mob. The derived Source Mob describes the media that was generated from the media described by the previous generation Source Mob using a pulldown mechanism. The derived Source Mob contains a track (Mob Slot) that contains a Pulldown object. The Pulldown object describes the pulldown mechanism and has a Source Clip that identifies the previous generation Source Mob. The Source Clip is specified using the edit rate of the previous generation Source Mob and does not use the edit rate of the track that contains the Pulldown object.

Each kind of pulldown identifies the speed of the tape. If two Source Mobs have a pulldown relationship, the edit rates of the video tracks should correspond to the frame rate of the media.

Related Classes

Mob Slot (MSLT), Source Clip (SCLP), Source Mob (SMOB)

RGBA Component Image Descriptor Class (RGBA)

Describes the media stored with separate color components and a separate alpha component associated with the Source Mob.

Data Model

RGBA Component Image Descriptor Class (RGBA) Is-a-Kind-of Digital Image Descriptor

`PixelLayout`
`PixelStructure`
`Palette`
`PaletteLayout`
`PaletteStructure`

Implementation

Property Name	Type	Explanation
`OMFI:OOBJ:ObjClass`	`omfi:ClassID`	Class is RGBA.
`OMFI:MDES:Locator`	`omfi:ObjRefArray`	Set of locators that provide hints to help find the OMFI file or the raw data file that contains the media data. Optional.
`OMFI:MDFL:IsOMFI`	`omfi:Boolean`	A True value indicates that the media data is stored in an OMFI file; a False value indicates that the media data is stored in a raw data file.
`OMFI:MDFL:SampleRate`	`omfi:Rational`	The native sample rate of the digitized media data.
`OMFI:MDFL:Length`	`omfi:Length32` `omfi:Length64`	Duration of the media in sample units.
`OMFI:DIDD:Compression`	`omfi:String`	Kind of compression and format of compression information; no compresssion is allowed for RGBA data; this property should not be present in an RGBA object.
`OMFI:DIDD:StoredHeight`	`omfi:UInt32`	Number of pixels in vertical dimension of stored view. See the Description section of DIDD class for an explanation of image geometry.
`OMFI:DIDD:StoredWidth`	`omfi:UInt32`	Number of pixels in horizontal dimension of stored view.

Property Name	Type	Explanation
OMFI:DIDD:SampledHeight	omfi:UInt32	Number of pixels in vertical dimension of sampled view. Optional; the default value is StoredHeight. See the Description section of DIDD class for an explanation of image geometry.
OMFI:DIDD:SampledWidth	omfi:UInt32	Number of pixels in horizontal dimension of sampled view. Optional; the default value is StoredWidth.
OMFI:DIDD:SampledXOffset	omfi:Int32	X offset, in pixels, from top left corner of stored view. Optional; the default value is 0.
OMFI:DIDD:SampledYOffset	omfi:Int32	Y offset, in pixels from top left corner of stored view. Optional; the default value is 0.
OMFI:DIDD:DisplayHeight	omfi:UInt32	Number of pixels in vertical dimension of display view. Optional; the default value is StoredHeight. See the Description section of DIDD class for an explanation of image geometry.
OMFI:DIDD:DisplayWidth	omfi:UInt32	Number of pixels in vertical dimension of display view. Optional; the default value is StoredWidth.
OMFI:DIDD:DisplayXOffset	omfi:Int32	X offset, in pixels, from top left corner of stored view. Optional; the default value is 0.
OMFI:DIDD:DisplayYOffset	omfi:Int32	Y offset, in pixels, from top left corner of stored view. Optional; the default value is 0.
OMFI:DIDD:FrameLayout	omfi:LayoutType	Describes whether all data for a complete sample is in one frame or is split into more than one field. Values are 0 FULL_FRAME: frame contains full sample in progressive scan lines. 1 SEPARATE_FIELDS: sample consists of two fields, which when interlaced produce a full sample. 2 SINGLE_FIELD: sample consists of two interlaced fields, but only one field is stored in the data stream. 3 MIXED_FIELDS.
OMFI:DIDD:VideoLineMap	omfi:Int32Array	Specifies the scan line in the analog source that corresponds to the beginning of each digitized field. For single-field video, there is 1 value in the array and for interleaved video, there are 2 values in the array.
OMFI:DIDD:ImageAspectRatio	omfi:Rational	Describes the ratio between the horizontal size and the vertical size in the intended final image.

Property Name	Type	Explanation
OMFI:DIDD:AlphaTransparency	omfi:Int32	A value of 1 means that the maximum Alpha value is transparent. A value of 0 means that the 0 Alpha value is transparent. Optional.
OMFI:DIDD:Gamma	omfi:Rational	Specifies the expected output gamma setting on the video display device. Optional.
OMFI:DIDD:ImageAlignmentFactor	omfi:Int32	Specifies the alignment when storing the digital media data. For example, a value of 16 means that the image is stored on 16-byte boundaries. The starting point for a field will always be a multiple of 16 bytes. If the field does not end on a 16-byte boundary, the remaining bytes are unused. Optional.
OMFI:RGBA:PixelLayout	omfi:CompCodeArray	An array of characters that specifies the order that the color components of a pixel are stored in the image. Each element in the array represents a different color component. The array can contain the following characters: ‘A’ Alpha component ‘B’ Blue component ‘F’ Fill component ‘G’ Green component ‘P’ Palette code ‘R’ Red component ‘0’ no component Each character except ‘0’ can appear no more than one time in the array. The array is terminated by a 0 byte and has a maximum of 8 elements (including the terminating byte). Note that a byte with the ASCII ‘0’ indicates no component, and a byte with a 0 (ASCII NULL) terminates the string.
OMFI:RGBA:PixelStructure	omfi:CompSizeArray	An array of UInt8 that specifies the number of bits allocated to store each component in the order specified in the PixelLayout property. The array is terminated by a 0 byte and has a maximum of 8 elements (including the terminating byte). Optional.
OMFI:RGBA:Palette	omfi:DataValue	An array of color values that are used to specify an image. Optional.

Property Name	Type	Explanation
OMFI:RGBA:PaletteLayout	omfi:CompCodeArray	An array of characters that specifies the order that the color components are stored in the palette. Each element in the array represents a different color component. The array can contain the following characters: 'A' Alpha component 'B' Blue component 'F' Fill component 'G' Green component 'R' Red component '0' no component Each character except '0' can appear no more than one time in the array. The array is terminated by a 0 byte and has a maximum of 8 elements (including the terminating byte). Note that a byte with the ASCII '0' indicates no component, and a byte with a 0 (ASCII NULL) terminates the string. Optional.
OMFI:RGBA:PaletteStructure	omfi:CompSizeArray	An array of UInt8 that specifies the number of bits allocated to store each component in the order specified in the PaletteLayout property. The array is terminated by a 0 byte and has a maximum of 8 elements (including the terminating byte). Optional.

Description

An RGBA Component Image Descriptor (RGBA) object describes media data that contains component-based images where each pixel is made up of a red, a green and a blue value. Each pixel can be described directly with a component value or a by an index into a pixel palette.

Properties in the RGBA Component Image Descriptor allow you to specify the order that the color components are stored in the image, the number of bits needed to store a pixel, and the bits allocated to each component.

To describe pixels with a pixel palette, the RGBA object should:

- Have a 'P' character in the PixelLayout property value and not have a 'B', 'G', or 'R' character in the PixelLayout property value
- Include the Palette, PaletteLayout, and PaletteStructure properties

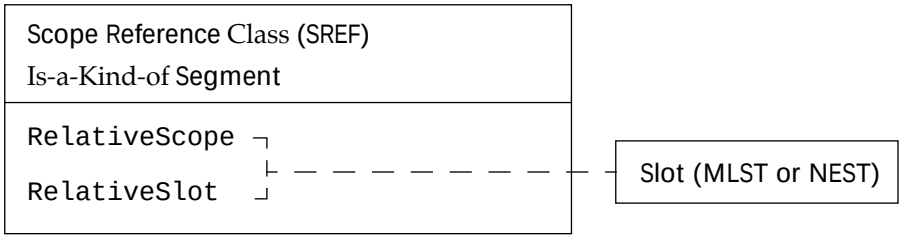
Related Classes

Color Difference Component Image Descriptor (CDCI), Digital Image Descriptor (DIDD), Image Data (IDAT), JPEG Image Data (JPEG), Source Mob (SMOB)

Scope Reference Class (SREF)

Refers to a section in the specified Mob Slot or Nested Scope slot.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is SREF.
OMFI:CPNT:DataKind	omfi:ObjRef	HAS-REFERENCE to a Data Definition object that specifies the data kind of the Scope Reference object.
OMFI:CPNT:Length	omfi:Length32 omfi:Length64	Specifies the duration of the Scope Reference in the edit units defined by its context.
OMFI:SREF:RelativeScope	omfi:UInt32	Specifies the number of Nested Scopes to pass to find the Nested Scope or Mob containing the slot.
OMFI:SREF:RelativeSlot	omfi:UInt32	Specifies the number of slots that precede the slot containing the Scope Reference to pass to find the slot referenced.

Rules

1. The data kind of the Segment in the referenced slot must be the same as or convertible to the data kind of the Scope Reference object.
2. The value of RelativeScope must be less than or equal to the number of Nested Scope objects that contain the Scope Reference. If the Scope Reference is not contained in any Nested Scope object, then the RelativeScope must have a value of 0.
3. The value of RelativeSlot must be greater than 0 and less than or equal to the number of slots that precede it within the scope specified by RelativeScope.

Description

A Scope Reference object has the same time-varying values as the section of the Nested Scope slot or Mob Slot that it references. Scope Reference objects allow one or more objects to share the values produced by a section of a slot.

Scopes and Scope References

The Mob defines a scope consisting of the ordered set of Mob Slots. A Scope Reference object in a Mob Slot can specify any Mob Slot that precedes it. Nested Scope objects define scopes that are limited to the Components contained within the Nested Scope.

A Scope Reference object returns the same time-varying values as the corresponding section of the slot that it references. The time-varying values of a section of a slot can be shared by having more than one Scope Reference specify it. A Scope Reference object specifies the slot by specifying relative scope and relative slot. The relative scope specifies whether the slot is in the current scope that contains the Scope Reference, a preceding Nested Scope, or the Mob scope which is the outermost scope. The relative slot specifies one of the slots that precedes the slot containing the Scope Reference within the scope specified by the relative scope.

If a Scope Reference specifies a mob slot, the corresponding section of the slot is the one that has the equivalent starting position from the beginning of the mob slot and the equivalent length as the Scope Reference object has within its mob slot. If the specified Mob Slot has a different edit rate than the Mob Slot containing the Scope Reference, the starting position and duration are converted to the specified Mob Slots edit units to find the corresponding section.

If a Scope Reference specifies a Nested Scope slot, the corresponding section of the slot is the one that has the same starting position offset from the beginning of the Nested Scope segments and the same duration as the Scope Reference object has in the specified scope.

Specifying Relative Scope and Relative Slot

Relative scope is specified as an unsigned integer. It specifies the number of nested scopes that you must pass through to find the referenced scope. A value of 0 specifies the current scope, that is the innermost Nested Scope object that contains the Scope Reference or the Mob scope if no Nested Scope object contains it. A value of 1 specifies the scope level that contains the Nested Scope object that contains the Scope Reference.

Relative slot is specified as a positive integer. It specifies the number of preceding slots that you must pass to find the referenced slot within the specified relative scope. A value of 1 specifies the immediately preceding slot.

Related Classes

Mob (MOBJ), Mob Slot (MSLT), Nested Scope (NEST), Segment (SEGM)

Segment Class (SEGM)

Represents a Component that is independent of any surrounding object.

Data Model

Segment Class (SEGM) Is-a-Kind-of Component Abstract Class
no additional properties

Implementation

Property Name	Type	Explanation
`OMFI:OOBJ:ObjClass`	`omfi:ClassID`	Class is SEGM.
`OMFI:CPNT:DataKind`	`omfi:ObjRef`	HAS-REFERENCE to a Data Definition object that specifies the data kind of the Segment.
`OMFI:CPNT:Length`	`omfi:Length32` `omfi:Length64`	Specifies the duration of the Segment in edit units.

Description

Segment is an abstract class; its subclasses include all Components with well-defined boundaries, that is, those that have well-defined values within their boundary without needing to be surrounded by other Components in a Sequence. This is in contrast with the other subclass of the Component class, the Transition class. Transition objects may only be used within a Sequence object and must be preceded and followed by a Segment object. The value produced by a Transition object is dependent on the values produced by the preceding and following Segments.

Related Classes

Component (CPNT), Transition (TRAN)

Selector Class (SLCT)

Provides the value of a single Segment while preserving references to unused alternatives.

Data Model

Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is SLCT.
OMFI:CPNT:DataKind	omfi:ObjRef	HAS-REFERENCE to a Data Definition object that specifies the data kind of the Selector object.
OMFI:CPNT:Length	omfi:Length32 omfi:Length64	Specifies the duration of the Selector object.
OMFI:SLCT:Selected	omfi:ObjRef	HAS the selected Segment.
OMFI:SLCT:Alternates	omfi:ObjRefArray	HAS a set of unused alternative Segments. Optional.

Rules

1. The duration of the selected Segment and of each alternative Segment must equal the duration of the Selector object.
2. The data kind of the selected Segment and of each alternative Segment must be the same as or convertible to the data kind of the Selector object.

Description

A Selector object provides the value of the selected Segment and preserves references to some alternative Segments that were available during the editing session. The alternative Segments can be ignored while playing a Composition Mob because they do not affect the value of the Selector object and cannot be referenced from outside of it. The alternative Segments can be presented to the user when the Composition Mob is being edited. Typically, a Selector object is used to present alternative presentations of the same content, such as alternate camera angles of the same scene.

A Selector object represents an editing decision. This is in contrast with a Media Group object which presents a group of alternative implementations of the same media that the application can choose from based on the most appropriate or efficient media format among the alternatives.

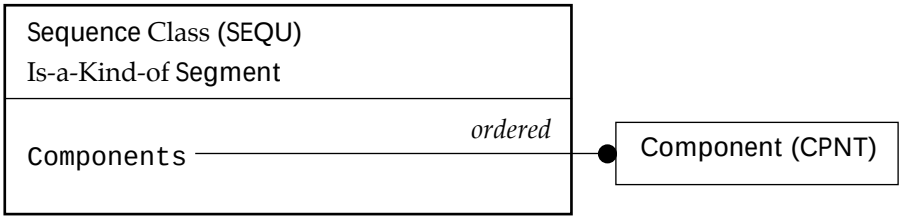
Related Classes

Media Group (MGRP), Segment (SEGM)

Sequence Class (SEQU)

Combines a series of Segments into a single longer Segment, optionally with Transitions between the Segments.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is SEQU.
OMFI:CPNT:DataKind	omfi:ObjRef	HAS-REFERENCE to a Data Definition object that specifies the data kind of the Sequence object.
OMFI:CPNT:Length	omfi:Length32 omfi:Length64	Specifies the duration in edit units of the Sequence object.
OMFI:SEQU:Components	omfi:ObjRefArray	HAS an ordered set of Segment objects; optionally separated by Transition objects.

Rules

1. The first and last Component in the ordered set must both be Segment objects. (Neither can be a Transition object.)
2. A Transition object can only appear in a Sequence between two Segment objects. The length of each of these Segments must be greater than or equal to the length of the Transition.
3. If a Segment object has a Transition before it and after it, the sum of the lengths of the surrounding Transitions must be less than or equal to the length of the Segment that they surround.
4. The length of the Sequence must equal the sum of the length of all Segments contained in the Sequence minus the sum of the lengths of all Transitions contained in the Sequence.
5. The data kind of each Component in the Sequence object must be the same as or convertible to the data kind of the Sequence.

Description

A Sequence object allows you to specify an ordered set of Components in any context that you can specify a single Segment. The Sequence object is the basic

mechanism for combining sections of media to be played in a sequential manner in a Mob Slot in an OMFI file.

If a Sequence object contains a Segment followed by another Segment, after the first Segment is played, the following one begins immediately. If a Sequence object contains Segment objects only with no Transition objects, the duration of the Sequence is equal to the sum of the duration of each Segment that it contains.

If a Sequence object contains a Transition object, the last section of the Segment that precedes the Transition, the Transition, and the first section of the Segment that follows the Transition are overlapped. The duration of the Transition determines the duration of the section of the preceding and following Segments that are overlapped.

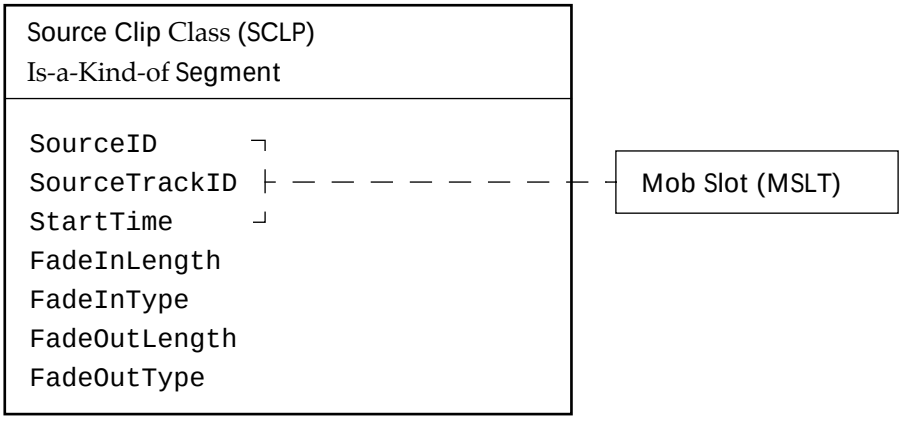
Related Classes

Component (CPNT), Segment (SEGM), Transition (TRAN)

Source Clip Class (SCLP)

Represents the media or other time-varying data described by a section of a Mob Slot in another Mob.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is SCLP.
OMFI:CPNT:DataKind	omfi:ObjRef	HAS-REFERENCE to a Data Definition object that specifies the data kind of the Source Clip object.
OMFI:CPNT:Length	omfi:Length32 omfi:Length64	Specifies the duration of the Source Clip object and of the referenced section in edit units determined by the Source Clip object's context.
OMFI:SCLP:SourceID	omfi:UID	In conjunction with the SourceTrackID and StartTime properties, HAS-REFERENCE to a section of a Mob Slot within another Mob. The SourceID specifies the MobID. If the value is 0-0-0, it means that this Mob describes the original source.
OMFI:SCLP:SourceTrackID	omfi:UInt32	Specifies the TrackID of the referenced Mob Slot within the other mob. If the SourceID has a value 0-0-0, then SourceTrackID must have a 0 value.
OMFI:SCLP:StartTime	omfi:Position32 omfi:Position64	Specifies the offset from the origin of the referenced Mob Slot in edit units determined by the Source Clip object's context. If the SourceID has a value 0-0-0, then StartTime must have a 0 value.
OMFI:SCLP:FadeInLength	omfi:Length32	Specifies the length of an audio fade-in to be applied to the Source Clip. Optional.

Property Name	Type	Explanation
OMFI:SCLP:FadeInType	omfi:FadeType	Specifies the type of the audio fade; may have one of the following values: 0 No fade 1 Linear amplitude fade 2 Linear power fade 3 Linear dB fade Additional registered and private fade in types may be defined. Optional.
OMFI:SCLP:FadeOutLength	omfi:Length32	Specifies the length of an audio fade-out to be applied to the Source Clip. Optional.
OMFI:SCLP:FadeOutType	omfi:FadeType	Specifies the type of the audio fade; may have one of the following values: 0 No fade 1 Linear amplitude fade 2 Linear power fade 3 Linear dB fade Additional registered and private audio fade types may be defined. Optional.

Rules

1. The data kind of the Segment contained in the referenced Mob Slot must be the same as or convertible to the data kind of the Source Clip object.
2. The fade properties are only allowed when the data kind is omfi:data:Sound or omfi:data:StereoSound.

Description

A Source Clip object represents a section of media or other time-varying data described by a Mob Slot in another Mob. In a Composition Mob, Source Clip objects are used to reference the digitized media data to be played or manipulated.

Although the Source Clip class data model shows that it has a reference to the Mob Slot object, the implementation does not use an object reference to accomplish this. The implementation uses two properties, SourceID and SourceTrackID, to identify the external Mob and the Mob Slot within it. The application must find the Mob with the specified ID and find the Mob Slot object that has the specified track ID in its Track Description.

The Source Clip represents a section of the media in the Mob Slot. The section is determined by the Source Clip StartTime and Length properties and the Track Description Origin property in the referenced Mob Slot. The Track Description Origin represents an offset from the beginning of the Segment contained in the Mob Slot and is expressed in the edit rate of the Mob Slot. The time specified by this offset is the external reference point. The Source Clip StartTime property represents an offset from this external reference point and is expressed in the Source Clip's edit rate. The Source Clip Length property represents the duration of the referenced section and is expressed in the Source Clip's edit rate.

Source Clip objects are leaf objects in the Mob structure. Although they have a reference to a Mob Slot in another Mob, that Mob Slot object is not part of the Mob containing the Source Clip object.

Mob References and Chains

One Source Clip object in a Mob refers to a Mob Slot in another Mob by specifying the second mob's Mob ID and the track ID of the Mob Slot in it.

These Mob references, from a Source Clip in one Mob to a Mob Slot in another Mob form Mob chains that enable one to go from a Source Clip in a Composition Mob to the digitized media and then to the physical sources that make up the previous media generation. Mob chains allow you to go

- From a Source Clip in a Composition Mob to the Master Mobs that describe the media; in addition a Source Clip in a Composition Mob can reference another Composition Mob
- From a Source Clip in a Master Mob to the file Source Mob that describes and locates the digital media
- From a Source Clip in the file Source Mob to the physical Source Mob that describes the previous generation of media
- From a Source Clip in the physical Source Mob to another physical Source Mob that describes a preceding generation media, such as from a videotape Source Mob to a film Source Mob

If a Source Mob represents the original media source and there is no previous generation, then its Source Clips must specify a value 0-0-0 for its `SourceID` and 0 values for `SourceTrackID` and `StartTime`.

In summary, Source Clips allow you to find the digitized media and the original sources used to create the digitized media.

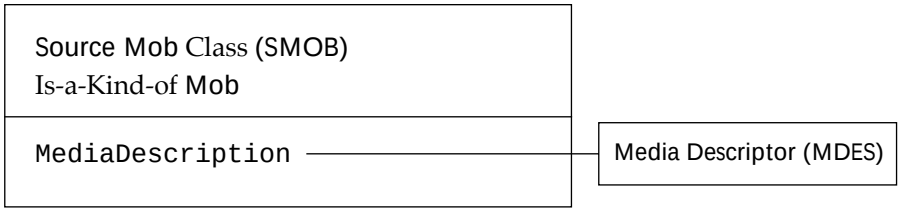
Related Classes

Mob (MOBJ), Mob Slot (MSLT), Track Description (TRKD)

Source Mob Class (SMOB)

Represents and describes media data that is either contained in a physical source, such as tape or film, or contained as digitized media data stored in a file.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is SMOB.
OMFI:MOBJ:MobID	omfi:UID	Unique Mob identification.
OMFI:MOBJ:Name	omfi:String	Name of the Mob for display to end user. Optional.
OMFI:MOBJ:Slots	omfi:ObjRefArray	HAS an ordered set of Mob Slots that contain media and other time-varying information.
OMFI:MOBJ:LastModified	omfi:TimeStamp	Date and time when the Mob was last modified.
OMFI:MOBJ:CreationTime	omfi:TimeStamp	Date and time when the Mob was originally created.
OMFI:MOBJ:UserAttributes	omfi:ObjRef	Specifies a set of user attributes, which provide additional information about the Mob. The Attribute Array contains a set of Attributes. Optional.
OMFI:SMOB:MediaDescription	omfi:ObjRef	Describes the format of the media associated with the Source Mob.

Description

A Source Mob represents a file containing digitized media or a physical media source, such as an audio tape, film, or videotape.

If the media described by the Source Mob has been derived from a previous generation of media, the Mob Slots should contain Source Clips that identify the Mob that describes the previous generation. The Source Clip SourceID, SourceTrackID, and StartTime properties identify the Mob. If the Source Mob describes media that is not derived from a previous generation, the Mob Slots should contain Source Clips that specify the special MobID 0-0-0.

The length of the Segment in the Mob Slot indicates the duration of the media. If you create a Source Mob for a physical media source and you do not know the duration of the media, specify a length of 24 hours.

If the Source Mob describes media that was derived from a Composition Mob, the Source Clips should identify the Composition Mob as the source. For example, if you create a Source Mob to describe a videotape that was written from a Composition Mob, the Composition Mob is the previous generation.

Source Media Is Immutable

The media data represented by a Source Mob is immutable. If the media changes, such as if a videotape is redigitized, you must create a new Source Mob with a new Mob ID.

Related Classes

Composition Mob (CMOB), Header (HEAD), Master Mob (MMOB), Mob (MOBJ), Mob Slot (MSLT), Source Clip (SCLP), Track Description (TRKD)

Text Locator Class (TXTL)

Provides information to help find a file containing the media data.

Data Model

Text Locator Class (TXTL) Is-a-Kind-of Locator
Name

Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is TXTL.
OMFI:TXTL:Name	omfi:String	Text string containing information to help find the raw data file or OMFI file containing the media data.

Description

A Text Locator (TXTL) object provides information to the user to help locate the OMFI file or raw data file containing the digital media data; it is not handled by applications automatically because the meaning or format of the Name property value is not defined.

The Source Mob (SMOB) describing the digital media data HAS a Media Descriptor object that optionally HAS a set of Locators.

Related Classes

DOS Locator (DOSL), Locator (LOCR), Mac Locator (MACL), Media Descriptor (MDES), UNIX Locator (UNXL), Windows Locator (WINL)

TIFF Image Data Class (TIFF)

Contains TIFF image data.

Data Model

TIFF Image Data Class (TIFF) Is-a-Kind-of Media Data
`ImageData`

Implementation

Property Name	Type	Explanation
`OMFI:OOBJ:ObjClass`	`omfi:ClassID`	Class is TIFF.
`OMFI:MDAT:MobID`	`omfi:UID`	MobID of the Source Mob describing the media data.
`OMFI:TIFF:ImageData`	`omfi:DataValue`	TIFF format image data.

Description

A TIFF Image Data object contains a stream of image media data. It has the same Mob ID as a Source Mob object in the OMFI file. This Source Mob describes the format of the image media in its `MediaDescription` property. The TIFF format is included for compatibility with OMF Interchange Version 1.0.

The image data is formatted according to the TIFF specification, Revision 6.0, available from Aldus Corporation. The `TIFF` object type supports only the subset of the full TIFF 6.0 specification defined as baseline TIFF in that document.

See Appendix C for more information on the TIFF format.

Related Classes

Color Difference Component Image Descriptor (CDCI), Image Data (IDAT), JPEG Image Data (JPEG), Media Data (MDAT), RGBA Component Image Data (RGBA), Source Mob (SMOB), TIFF Image Descriptor (TIFD)

TIFF Image Descriptor Class (TIFD)

Describes the TIFF media associated with a Source Mob.

Data Model

TIFF Image Descriptor Class (TIFD) Is-a-Kind-of Media File Descriptor
IsUniform IsContiguous LeadingLines TrailingLines JPEGTableID Summary

Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is TIFD.
OMFI:MDES:Locator	omfi:ObjRefArray	HAS a set of Locators that provide hints to help find the OMFI file or the raw data file that contains the media data. Optional.
OMFI:MDFL:IsOMFI	omfi:Boolean	A True value indicates that the media data is stored in an OMFI file; a False value indicates that the media data is stored in a raw data file.
OMFI:MDFL:SampleRate	omfi:Rational	The native sample rate of the digitized media data.
OMFI:MDFL:Length	omfi:Length32 omfi:Length64	Duration of the media in sample units.
OMFI:TIFD:IsUniform	omfi:Boolean	True for data having the same number of rows per strip throughout.
OMFI:TIFD:IsContiguous	omfi:Boolean	True for data stored in contiguous bytes.
OMFI:TIFD:LeadingLines	omfi:Int32	Number of leading lines to be thrown away. Optional.
OMFI:TIFD:TrailingLines	omfi:Int32	Number of trailing lines to be thrown away. Optional.
OMFI:TIFD:JPEGTableID	omfi:JPEGTableIDType	Registered JPEG table code or JT_NULL. Optional.
OMFI:TIFD:Summary	omfi:DataValue	A copy of the TIFF IFD (without the sample data).

Description

A TIFF Image Descriptor object describes the TIFF image data associated with the Source Mob. The image data is formatted according to the TIFF specification, Revision 6.0, available from Aldus Corporation. The `TIFF` object type supports only the subset of the full TIFF 6.0 specification defined as baseline TIFF in that document.

Note

The TIFF image format has been replaced by the Color Difference Component Image Descriptor (CDCI) format and the RGBA Component Image Descriptor (RGBA) format in the current version of the specification. The TIFF format is included in this specification for compatibility with OMF Interchange Version 1.0.

The `JPEGTableID` is an assigned type for preset JPEG tables. The table data must also appear in the TIFF object along with the sample data, but cooperating applications can save time by storing a preapproved code in this property that presents a known set of JPEG tables.

Related Classes

Color Difference Component Image Descriptor (CDCI), Image Data (IDAT), JPEG Image Data (JPEG), RGBA Image Descriptor (RGBA), Source Mob (SMOB), TIFF Image Data (TIFF)

Timecode Class (TCCP)

Stores videotape or audio tape timecode information.

Data Model

Timecode Class (TCCP) Is-a-Kind-of Segment

`Start`
`FPS`
`Flags`

Implementation

Property Name	Type	Explanation
`OMFI:OOBJ:ObjClass`	`omfi:ClassID`	Class is TCCP.
`OMFI:CPNT:DataKind`	`omfi:ObjRef`	HAS-REFERENCE to a Data Definition object that specifies the data kind `omfi:data:Timecode`.
`OMFI:CPNT:Length`	`omfi:Length32` `omfi:Length64`	Duration of contiguous timecode values.
`OMFI:TCCP:Start`	`omfi:Position32` `omfi:Position64`	Specifies the timecode at the beginning of the segment.
`OMFI:TCCP:FPS`	`omfi:UInt16`	Frames per second of the videotape or audio tape.
`OMFI:TCCP:Drop`	`omfi:Boolean`	Indicates whether the timecode is drop (True value) or nondrop (False value).

Description

A Timecode object can typically appear in either a Source Mob or in a Composition Mob. In a Source Mob, it typically appears in a Mob Slot in a Source Mob that describes a videotape or audio tape. In this context, it describes the timecode that exists on the tape. In a Composition Mob, it represents the timecode associated with the virtual media represented by the Composition Mob. If the Composition Mob is rendered to a videotape, the Timecode should be used to generate the timecode on the videotape.

Related Classes

Composition Mob (CMOB), Edge Code (ECCP), Segment (SEGM), Source Mob (SMOB)

Track Description Class (TRKD)

Specifies that the Mob Slot is externally accessible and the `TrackID` value needed to reference it.

Data Model

Track Description Class (TRKD) Is-a-Kind-of OMFI Object

`Origin`
`TrackID`
`TrackName`
`PhysicalTrack`

Implementation

Property Name	Type	Explanation
`OMFI:OOBJ:ObjClass`	`omfi:ClassID`	Class is TRKD.
`OMFI:TRKD:Origin`	`omfi:Position32` `omfi:Position64`	Specifies the offset in the Mob Slot segment expressed in the Mob Slot's edit rate that all external references are relative to.
`OMFI:TRKD:TrackID`	`omfi:UInt32`	Specifies a positive integer that is unique for all Mob Slots in the same Mob.
`OMFI:TRKD:TrackName`	`omfi:String`	Specifies a label that can be used to identify the track when displaying it to the user. Optional.
`OMFI:TRKD:PhysicalTrack`	`omfi:UInt32`	Specifies the physical output channel for tracks in Source Mobs.

Description

The Track Description specifies that the Mob Slot is externally visible and specifies the `TrackID`, `TrackLabel`, and `Origin` for the Mob Slot. These Track Description properties allow the Mob Slot to be referenced from outside the Mob.

The `Origin` property specifies an offset in the `Segment` contained in the Mob Slot that is the reference point for all Source Clip references to this Mob Slot. The `Origin` typically has a value of 0 except in Master Mob objects that refer to media that has changed. If you are redigitizing data and changing the starting point, you should set the `Origin` property in the Mob Slot in the Master Mob so that Source Clips reference the equivalent media in the new Source Mob as they did in the Source Mob describing the previous version of the digitized data.

The `PhysicalTrack` property identifies the physical track associated with the media. For File Source Mobs that describe stereo audio media, the left channel should have a `PhysicalTrack` of 1 and the right channel should have a `PhysicalTrack` of 2.

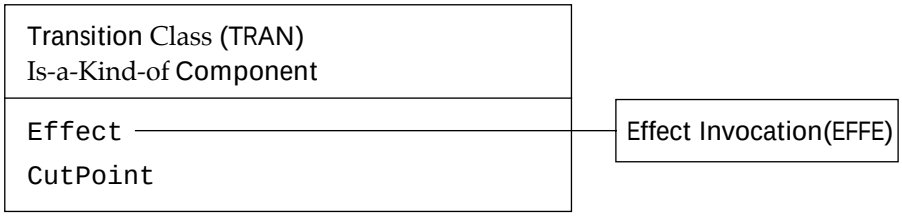
Related Classes

Mob (MOBJ), Mob Slot (MSLT), Source Clip (SCLP)

Transition Class (TRAN)

Specifies the way to change from one Segment to another in a Sequence.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is TRAN.
OMFI:CPNT:DataKind	omfi:ObjRef	HAS-REFERENCE to a Data Definition object that specifies the data kind of the Transition.
OMFI:CPNT:Length	omfi:Length32 omfi:Length64	Specifies the duration of the Transition.
OMFI:TRAN:Effect	omfi:ObjRef	HAS an Effect Invocation that is to be performed during the Transition.
OMFI:TRAN:CutPoint	omfi:Position32	Specifies a cut point to use if replacing the Transition with a cut.

Description

A Transition object specifies the way to change from one Segment to another when playing the Sequence in which they occur. A Transition object specifies the effect to use during the Transition. A Transition object occurs in a Sequence and must be preceded and followed by a Segment object.

A Transition object specifies that sections of the preceding and following segments overlap for the duration of the Transition. The effect combines the media from the overlapping sections in some way.

Transition and Effect Control Arguments

The effect documentation specifies whether an effect is allowed in a Transition and how the Effect Slots should be treated. Typically, an effect that is allowed in a Transition will specify that the Effect Slot with the argument ID values -1 and -2 should correspond to the overlapping sections from the preceding and following Segments. Typically, the default value for the Effect Slot with the ar-

gument ID value -3 varies from 0.0 to 1.0, but this Effect Slot can be explicitly overriden by specifying it in the Effect Invocation.

Cut Point

The Transition `CutPoint` has no direct effect on the results produced by a Transition. However, the cut point provides information that is useful if an application wishes to remove the Transition or substitute a cut when playing the Transition. The cut point is represented as an offset from the beginning of the Transition. When removing the Transition, an application would change the Composition Mob so that the preceding Segment ends where the cut point is located, and the succeeding Segment starts at that location. This can be done by trimming the end of the preceding Segment by an amount equal to the Transition length minus the cut point offset, and trimming the beginning of the succeeding Segment by an amount equal to the cut point offset.

Related Classes

Component (CPNT), Effect Invocation (EFFE), Segment (SEGM)

UNIX Locator Class (UNXL)

Provides information to help find a UNIX file containing media data.

Data Model

UNIX Locator Class (UNXL) Is-a-Kind-of Locator
PathName

Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is UNXL.
OMFI:UNXL:PathName	omfi:String	UNIX pathname for raw data file or OMFI file containing the media data.

Description

The UNIX Locator (UNXL) provides a UNIX pathname that contains a hint to help an application find the OMFI file or raw data file containing the media data.

The Source Mob (SMOB) describing the digital media data HAS a Media Descriptor object that optionally HAS a set of Locators.

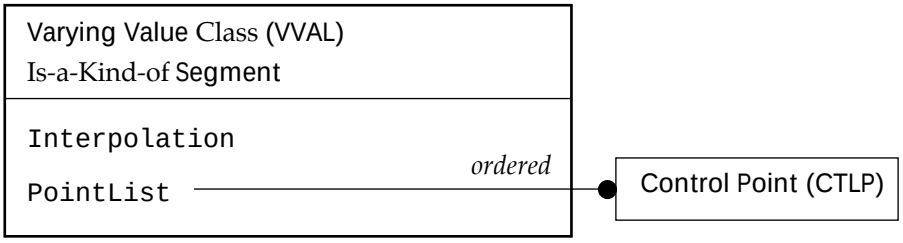
Related Classes

DOS Locator (DOSL), Locator (LOCR), Mac Locator (MACL), Media Descriptor (MDES), Text Locator (TXTL), Windows Locator (WINL)

Varying Value Class (VVAL)

Specifies a changing data value for the duration of the Component; typically used to specify values for an effect control slot.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is VVAL.
OMFI:CPNT:DataKind	omfi:ObjRef	HAS-REFERENCE to a Data Definition object that specifies the data kind of the Varying Value object.
OMFI:CPNT:Length	omfi:Length32 omfi:Length64	Specifies the duration of the Varying Value object in edit units.
OMFI:VVAL:Interpolation	omfi:InterpKind	Specifies the kind of interpolation to be used to find the value between Control Points; may have one of the following values: 1 Constant interpolation 2 Linear interpolation
OMFI:VVAL:PointList	omfi:ObjRefArray	HAS a set of Control Points, each of which specifies a value and a time point at which the value is defined.

Rules

1. Control Points must be ordered by their time value.
2. A Varying Value object can only have a data kind that has a defined data constant format. A Varying Value object cannot have a data kind that specifies a media stream because these formats do not have a defined constant format. Data kinds that specify a media stream include: omfi:data:EdgeCode, omfi:data:Picture, omfi:data:PictureWithMatte, omfi:data:Matte, omfi:data:Sound, omfi:data:StereoSound, and omfi:data:Timecode.

Description

A Varying Value object is a segment that returns time-varying values that are determined by an ordered set of Control Points. Each Control Point specifies the value for a specific time point within the Segment. The values for time points between two Control Points are calculated by interpolating between the two values.

Typically, Varying Value objects are used in Effect Slots to specify the value of a control argument for the effect, but Varying Value objects can be used in any context where a Segment is allowed.

Control Points

A Control Point that has a `Time` value equal to 0.0 represents the time at the beginning of the Varying Value object; one with a time equal to 1.0 represents the time at the end of the Varying Value object. Control Points with `Time` values less than 0.0 and greater than 1.0 are meaningful but are only used to establish the interpolated values within the Varying Value object—they do not affect values outside of the duration of the Varying Value object.

Since time is expressed as a rational value, any arbitrary time can be specified—the specified time point does not need to correspond to the starting point of an edit unit of the Segment.

If more than two Control Point objects specify the same value, the last Control Point determines the value for the time point specified and is used to interpolate values after this time point.

Interpolation of Control Values

The following equation specifies the value at time X, by using a linear interpolation and the values specified for time A and time B.

$$Value_X = \frac{(Time_X - Time_A)}{(Time_B - Time_A)} \times (Value_B - Value_A) + Value_A$$

Extrapolation of Control Values

If the first Control Point in a Varying Value object specifies a time value greater than 0, this value is extrapolated to the 0 time point by holding the value constant. If the last Control Point in a Varying Value object specifies a time value less than 1.0, this value is extrapolated to the 1.0 time point by holding the value constant. This extrapolation method of holding values is used if the interpolation method specified for the Varying Value object is constant or linear interpolation.

Sequence of Varying Value Objects

If you need to specify values for an Effect Slot by using more than one kind of interpolation, you must use a Sequence object that contains a series of Varying Value objects in the Effect Slot. Each Varying Value object can have its own interpolation method. Each Varying Value object defines the control argument values for its section of the Sequence. A time value of 0 specifies the beginning of the Varying Value object, which, if it is in a Sequence, may not correspond to the beginning of the Effect Slot . A time value of 1 specifies the end of the Varying Value object, which, if it occurs in a Sequence, may not correspond to the end of the Effect Slot .

Quantization Adjustments

The Varying Value object specifies a value for each time point within the Varying Value object; however, if you are generating a stream of media from the Composition Mob containing the Varying Value object, it can be important to adjust values produced by the Varying Value object based on sample-rate quantization. Within a media sample unit, there can only be a single value of the Varying Value object when generating that sample.

When the number of samples is large (when quantization error is not noticeable) it is usually sufficient to express a curve in a sample-rate independent form, which is converted to the appropriate sampled values when needed. However, there are often times when the desired sample rate is low enough that some precise control over how the curve gets sampled is needed for the right result. In particular, this occurs at video sample rates.

An example using a dissolve can illustrate the quantization problem. It is natural to think of a dissolve as a mixture between video stream A and video stream B, where the mix is controlled by a level parameter that goes from 0 to 1 over the duration of the dissolve. However, since the frame at any particular time freezes the value of the level that was specified at the beginning of the frame, the first frame of the dissolve will have a value of 0, and the last frame of the dissolve will be slightly less than 1. This result is incorrect, because the resulting frame sequence has a value of level which is asymmetrical. Changing the definition so that the middle of a frame is sampled instead of the beginning does not solve the problem; instead, it just transforms it into a case where the level change on the first and last frame of the dissolve is half of that for all the other frames. This is not correct because it is not uniform.

This error is due to quantization and becomes vanishingly small as the sample rate increases. But because it is sample-rate dependent, it is not something that can be accounted for in a sample-rate independent way simply by adjusting the Time values of the Control Points. Instead, it is up to the software that implements a particular effect to adjust the Control Point mapping for the actual sample rate at the time of rendering. This mapping adjustment is done by scaling the curve represented by the Varying Value so that the 0 point is moved back by one sample time before interpolation and quantization is performed.

The following formula scales a Control Point's `Time` value from its stored number in edit units to its actual number in sample units, relative to the beginning of the Varying Value component.

$$SampleTime = \left(\left(\left(\left(\frac{Length \times SampleRate}{EditRate}\right) + 1\right) \times ControlPointTime\right) - 1\right)$$

This algorithm makes the level 0 sample be the sample before the Effect Invocation starts and the level 1 sample be the sample after the Effect Invocation ends. For most effects, this is the desired result. However, some effects, such as fade-to-black or fade-from-color may need to modify the algorithm so that the level 0 or level 1 sample is included within the Effect Invocation. The effect documentation must specify a modified scaling algorithm if it should be used for the effect.

Related Classes

Constant Value (CVAL), Control Point (CTLP), Data Definition (DDEF), Effect Definition (EDEF), Effect Invocation (EFFE), Effect Slot (ESLT), Source Clip (SCLP)

WAVE Audio Data Class (WAVE)

Contains WAVE audio data.

Data Model

WAVE Audio Data Class (WAVE) Is-a-Kind-of Media Data
AudioData

Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is WAVE.
OMFI:MDAT:MobID	omfi:UID	MobID of the Source Mob describing the media data.
OMFI:WAVE:AudioData	omfi:DataValue	WAVE format data.

Description

A WAVE Audio Data object contains digitized audio data in the little-endian byte ordering (used on the Intel architecture). It contains data formatted according to the Microsoft/IBM Multimedia Programming Interface and Data Specifications, Version 1.0, but limited to the section describing the RIFF Waveform Audio File Format audio data.

The data is contained directly in the WAVE object. No additional data properties or objects are defined for WAVE data because this format contains all of the information needed for playback.

Related Classes

AIFC Audio Data (AIFC), AIFC Audio Descriptor (AIFD), Header (HEAD), Media Data (MDAT), Source Mob (SMOB), WAVE Audio Descriptor (WAVD)

WAVE Audio Descriptor Class (WAVD)

Describes the WAVE audio media associated with a Source Mob.

Data Model

WAVE Audio Descriptor Class (WAVD) Is-a-Kind-of Media File Descriptor
Summary

Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is WAVD.
OMFI:MDES:Locator	omfi:ObjRefArray	Set of Locators that provide hints to help find the OMFI file or the raw data file that contains the media data. Optional.
OMFI:MDFL:IsOMFI	omfi:Boolean	A True value indicates that the media data is stored in an OMFI file; a False value indicates that the media data is stored in a raw data file.
OMFI:MDFL:SampleRate	omfi:Rational	The native sample rate of the digitized media data.
OMFI:MDFL:Length	omfi:Length32 omfi:Length64	Duration of the media in sample units.
OMFI:WAVD:Summary	omfi:DataValue	A copy of the WAVE file information without the sample data.

Description

A WAVE Audio Descriptor describes a WAVE object that contains digitized audio data in the little-endian byte ordering (used on the Intel architecture). It contains data formatted according to the Microsoft/IBM Multimedia Programming Interface and Data Specifications, Version 1.0, but limited to the section describing the RIFF Waveform Audio File Format audio data. The WAVE file information (without the sample data) is duplicated in the WAVE Audio Descriptor Summary property to make it more efficient to access this information.

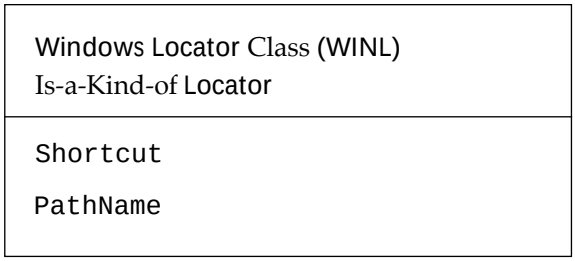
Related Classes

AIFC Audio Data (AIFC), AIFC Audio Descriptor (AIFD), Locator (LOCR), Media File Descriptor (MDFL), Source Mob (SMOB), WAVE Audio Data (WAVE)

Windows Locator Class (WINL)

Provides a Windows shortcut object to help find the file containing the media data.

Data Model



Implementation

Property Name	Type	Explanation
OMFI:OOBJ:ObjClass	omfi:ClassID	Class is WINL.
OMFI:WINL:Shortcut	omfi:DataValue	Contains a Windows shortcut to finding a file. Optional.
OMFI:WINL:PathName	omfi:String	Windows pathname for file containing the media data.

Description

The Windows Locator (WINL) provides data that contains a hint to help an application find the OMFI file or raw data file containing the media data.

The Source Mob (SMOB) describing the digital media data HAS a Media Descriptor object that optionally HAS a set of Locators.

Related Classes

DOS Locator (DOSL), Locator (LOCR), Mac Locator (MACL), Media Descriptor (MDES), Text Locator (TXTL), UNIX Locator (UNXL)

Appendix B

Data Types

OMF has two sets of types: the data type, which specifies the type of property values and the data kind, which specifies the type for objects in the Component class. The implementation section describing a class lists the data type of each property that the class includes. Objects that belong either to the class Component or Control Point have a property that identifies the data kind of the object. Data kind is used to describe time-varying values produced by Components that describe media and Components that supply control arguments to Effect Invocations.

The data type is identified by a globally unique text string that starts with the prefix `omfi:` and identifies the type. Table 9 lists the data types.

The data kind is specified by a Data Definition object, which contains the globally unique text string of the data kind. The data kinds defined in this document start with the prefix `omfi:data:`. The meaning, internal format, and size of the data kind are not described in the Data Definition object. This information is provided in this document or in the documentation provided with registered or private media formats and Effect Definitions. Table 10 lists the data kinds.

Table 11 describes the data kind conversions that are allowed in an OMFI file.

Table 9: Data Types

Data Type	Explanation
omfi:ArgIDType	Specifies an integer that identifies an Effect Slot.
omfi:AttrKind	A 16-bit integer. Values are 0 kOMFNullAttribute — Unspecified type 1 kOMFIntegerAttribute — Integer value 2 kOMFStringAttribute — String value 3 kOMFObjectAttribute — Object reference value
omfi:Boolean	Specifies either True or False.
omfi:Char	Specifies a single character value.
omfi:ClassID	Specifies the 4-character class identification.
omfi:ColorSitingType	Specifies how to compute subsampled values as a 16-bit enumerated type. Values are 0 coSiting — To calculate subsampled pixels, take the preceding pixel's color value, discard the other color values, and cosite the color with the first luminance value. 1 averaging — To calculate subsampled pixels, take the average of the two adjacent pixel's color values, and site the color in the center of the luminance pixels. 2 threeTap — To calculate subsampled pixels, take 25 percent of the previous pixel's color value, 50 percent of the first value, and 25 percent of the second value. For the first value in a row, use 75 percent of that value since there is no previous value. The threeTap value is only meaningful when the HorizontalSubsampling property has a value of 2.
omfi:CompCodeArray	Specifies the order in which the RGBA components are stored as an array of character. Each element in the array represents a different color component. The array can contain the following characters: 'A' Alpha component 'B' Blue component 'F' Fill component 'G' Green component 'P' Palette code 'R' Red component '0' no component Each character except '0' can appear no more than one time in the array. The array is terminated by a 0 byte and has a maximum of 8 elements (including the terminating byte). Note that a byte with the ASCII '0' indicates no component and a byte with a 0 (ASCII NULL) terminates the string.
omfi:CompSizeArray	Specifies the number of bits reserved for each component as an array of UInt8 in the order specified in the CompCodeArray. The array is terminated by a 0 byte and has a maximum of 8 elements (including the terminating byte).
omfi:DataValue	Specifies media data or a block of data whose type is specified by a data kind.

Table 9: Data Types (Continued)

Data Type	Explanation
omfi:EdgeType	Specifies the kind of film edge code as an enumerated Int16. Values are: 0 ET_NULL Invalid edge code 1 ET_KEYCODE Eastman Kodak KEYKODE ™ format. 2 ET_EDGENUM4 edge code format: nnnn+nn. 3 ET_EDGENUM5 edge code format: nnnnn+nn.
omfi:EditHintType	Specifies hints to be used when editing Control Points. Values are: 0 EH_Proportional 1 EH_RelativeLeft 2 EH_RelativeRight
omfi:FadeType	Specifies the type of the audio fade; may have one of the following values: 0 No fade 1 Linear amplitude fade 2 Linear power fade 3 Linear dB fade Additional registered and private fade types may be defined.
omfi:FilmType	Specifies the format of the film as an Int16 enumerated value. Values are: 0 FT_NULL invalid film type 1 FT_35MM 35 millimeter film 2 FT_16MM 16 millimeter film 3 FT_8MM 8 millimeter film 4 FT_65MM 65 millimeter film
omfi:InterpKind	Specifies the method to use when interpolating between Control Points. Values are: 1 Constant interpolation 2 Linear interpolation
omfi:Int8	Specifies an 8-bit integer value.
omfi:Int16	Specifies a 16-bit integer value.
omfi:Int32	Specifies a 32-bit integer value.
omfi:Int32Array	Specifies an array of 32-bit integer values.
omfi:JPEGTableIDType	Specifies the JPEG tables used in compressing TIFF data.
omfi:LayoutType	Describes whether all data for a complete sample is in one frame or is split into more than one field as an inumerated Int16. Values are: 0 FULL_FRAME: frame contains full sample in progressive scan lines 1 SEPARATE_FIELDS: sample consists of two fields, which when interlaced produce a full sample 2 SINGLE_FIELD: sample consists of two interlaced fields, but only one field is stored in the data stream 3 MIXED_FIELDS
omfi:Length32	Specifies the length of a Component with a 32-bit integer.
omfi:Length64	Specifies the length of a Component with a 64-bit integer.
omfi:ObjRef	Specifies another object.
omfi:ObjRefArray	Specifies a set of other objects.

Table 9: Data Types (Continued)

Data Type	Explanation
omfi:Position32	Specifies an offset into a Component with a 32-bit integer.
omfi:Position32Array	Specifies an array of 32-bit offsets.
omfi:Position64	Specifies an offset into a Component with a 64-bit integer.
omfi:Position64Array	Specifies an array of 64-bit offsets.
omfi:ProductVersion	Specifies the version number of the application. Consists of 5 16-bit integer values that specify the version of an application. The first four integers specify the major, minor, tertiary, and patch version numbers. The fifth integer has the following values: 0 kVersionUnknown — No additional version information 1 kVersionReleased — Released product 2 kVersionDebug — Development version 3 kVersionPatched — Released product with patches 4 kVersionBeta — Prerelease beta test version 5 kVersionPrivateBuild
omfi:Rational	Specifies a rational number by means of an Int32 numerator and an Int32 denominator.
omfi:String	Specifies a string of characters.
omfi:TapeCaseType	Describes the physical size of the tape; may have one of the following values: 0 3/4 inch videotape 1 VHS video tape 2 8mm videotape 3 Betacam videotape 4 Compact cassette 5 DAT cartridge 6 Nagra audio tape
omfi:TapeFormatType	Describes the format of the tape; may have one of the following values: 0 Betacam 1 BetacamSP 2 VHS 3 S-VHS 4 8mm 5 Hi8
omfi:TimeStamp	Specifies a date and time. The timestamp is a four-byte unsigned integer, followed by a one-byte Boolean value. The integer represents the number of seconds since January 1, 1970. The Boolean value indicates whether or not the value is based on Greenwich Mean Time (GMT) time.
omfi:UID	Specifies a Mob ID value; a 96-bit value to be interpreted as 3 long values. The value of the first 32 bits must be a registered application or organization code. The remainder of the value is assigned by an application.
omfi:UInt8	Specifies an unsigned 8-bit integer.
omfi:UInt16	Specifies an unsigned 16-bit integer.
omfi:UInt32	Specifies an unsigned 32-bit integer.

Table 9: Data Types (Continued)

Data Type	Explanation
`omfi:UniqueName`	Specifies a qualified name which conforms to the OMFI naming conventions.
`omfi:VersionType`	Specifies a 2-byte unsigned OMFI version number.
`omfi:VideoSignalType`	Specifies the type of video signal on the videotape. Values are: 0 NTSC 1 PAL 2 SECAM

Table 10: Data Kinds

Data Kind	Explanation
`omfi:data:Boolean`	Specifies either True or False.
`omfi:data:Char`	Specifies a single character value.
`omfi:data:Color`	Specifies a ColorSpace followed by a series of Rational, with one Rational for each color component in the color space. The size of this data type depends on the color space. If the color space is a 3-component color space, the size will be 25 bytes (1 byte for the color space, followed by 3 8-byte rationals.
`omfi:data:ColorSpace`	Specifies the color space used to describe color as an Int16 enumerated type. Values are: 0 RGB 1 YUV 2 YIQ 3 HSI 4 HSV 5 YCrCb 6 YDrDb 7 CMYK
`omfi:data:DirectionCode`	Specifies one of 8 directions in a 2-dimensional plane as an Int16 enumerated type. Values are: 0 Right 1 UpperRight 2 Up 3 UpperLeft 4 Left 5 LowerLeft 6 Down 7 LowerRight
`omfi:data:Distance`	Specifies a distance which is relative to the display dimensions of the image as an `omfi:Rational` value. A value of 1.0 represents 1/6 the vertical dimension of the image. For the common 4x3 aspect ration, the horizontal distance equals 8 units and the vertical distance equals 6 units.
`omfi:data:Edgecode`	Specifies a stream of film edge code values.
`omfi:data:Int32`	Specifies a signed 32-bit integer.
`omfi:data:Matte`	Specifies a stream of media that contains an image of alpha values.
`omfi:data:Picture`	Specifies a stream of media that contains image data.
`omfi:data:PictureWithMatte`	Specifies a stream of media that contains image data and a matte.

Table 10: Data Kinds (Continued)

Data Kind	Explanation
omfi:data:Point	Specifies a point relative to the display dimensions of the image as two omfi:Distance values. The origin is the center of the image; the first distance represents the horizontal distance (positive values increase to the right); and the second distance represents the vertical distance (positive values increase up). For example, for an aspect ration of 4x3, the upper right corner has a value (4.0,3.0) and the lower left corner has a value (-4.0,-3.0).
omfi:data:Polynomial	Specifies a polynomial value.
omfi:data:Rational	Specifies a rational number by means of an Int32 numerator and an Int32 denominator.
omfi:data:Sound	Specifies a stream of media that contains a single channel of sound.
omfi:data:StereoSound	Specifies a stream of media that contains two stereo channels of sound.
omfi:data:String	Specifies a string of characters.
omfi:data:Timecode	Specifies a stream of tape timecode values.
omfi:data:UInt8	Specifies an unsigned 8-bit integer.

Table 11: Data Kind Conversions

Convert from Data Kind	Convert to Data Kind	Explanation
PictureWithMatte	Picture	If a PictureWithMatte is used where a Picture is specified, the matte part is ignored and the picture part is used.
PictureWithMatte	Matte	If a PictureWithMatte is used where a Matte is specified, the picture part is ignored and the matte is used.
Picture	PictureWithMatte	If a Picture is used where a PictureWithMatte is specified, the picture is used and the matte is assumed to be all fore-ground.
Matte	PictureWithMatte	If a Matte is used where a PictureWithMatte is specified, the Matte is used and the Picture is assumed to be a grayscale image equivalent to the matte with white as background, black foreground.
Matte	Picture	If a Matte is used where a Picture is specified, the Picture is assumed to be a greyscale image equivalent to the matte with white as background, black as foreground.
Picture	Matte	If a Picture is used where a Matte is specified, the matte is assumed to be the equivalent of using only the luminance component of the Picture.
Color	Picture	If a Color is used where a Picture is specified, the Picture is assumed to have all pixels of the specified color.

Table 11: Data Kind Conversions (Continued)

Convert from Data Kind	Convert to Data Kind	Explanation
`Color`	`Matte`	If a Color is used where a Matte is specified, the Matte is assumed to have all pixels be the luminance component of the specified color.
`Color`	`PictureWithMatte`	If a Color is used where a PictureWithMatte is specified, the Picture is assumed to have all pixels of the specified color, and the matte is assumed to be all foreground. A Picture, Matte, or PictureWithMatte cannot be implicitly converted to a Color.
`Color`	`Rational`	A color is converted to a Rational by using only the luminance component
`Rational`	`Color`	A Rational is converted to a color by making a color whose luminance component is the Rational and whose chrominance components are 0.
`Sound`	`StereoSound`	A Sound is converted to a StereoSound by using half the original sound on each of the Left and Right parts.
`StereoSound`	`Sound`	A StereoSound is converted to a Sound by adding the Left and Right parts.
`Int32, Int16, Int8, Uint32, Uint16, UInt8`	`Rational`	Any of the integer types is converted to a Rational by first converting it to an Int32, and then using the result as the numerator and setting the denominator to 1.
`Rational`	`Int32`	A Rational is converted to a Int32 by dividing numerator by denominator and rounding any fractional part toward negative infinity.

Appendix C
References and Media Formats

This appendix lists references to specifications and standards documents and describes the formats used to store media data in disk files.

References

Bento®

Bento Specification, Revision 1.0d5 by Jed Harris and Ira Ruben. Apple Computer, Inc.

For a copy of the Bento Specification and the Bento software, you can contact the OMF Developers' Desk.

TIFF

TIFF, Revision 6.0, Final - June 3, 1992. Aldus Developers Desk.

OMF Interchange supports the use of TIFF as a media data format for interchange of graphic data and video frame data. The TIFF specification, sample TIFF files, and other TIFF developer information are available on CompuServe ("Go ALDSVC", Library 10) and on AppleLink® (Aldus Developers Icon). For a copy of the TIFF 6.0 specification, call (206) 628-6593. For other information about the TIFF format, contact:

Aldus Developers Desk
411 First Avenue South
Seattle, WA 98104-2871

AIFF

Audio Interchange File Format, Apple Computer, Inc., Version 1.

For information on how to get the Audio Interchange File Format (AIFC) specification, call Apple Computer Customer Assistance at (800) 776-2333.

WAVE Audio File Format

For information or a copy of the WAVE audio file format specification, contact Microsoft Developer's Services at (800) 227-4679.

Other References

Joint Photographic Experts Group Standard JPEG-8-R8.

Microsoft/IBM Multimedia Programming Interface and Data Specifications, Version 1.0.

SMPTE Recommended Practice, RP 146-1987, "Transfer of Edit Decision Lists," Society of Motion Picture and Television Engineers.

Media Formats

TIFF File Format

OMF Interchange extends baseline TIFF 6.0 in order to efficiently store video frame sequences. OMF Interchange includes a version of the `TIFF` object type for TIFF format video data that includes JPEG full motion video media data and frame information.

The object contains the TIFF image file Header (IFH), followed by a series of frames, the TIFF image file directory (IFD), reference data including the JPEG tables, and a frame index.

The IFD can contain baseline TIFF fields, plus JPEG and YCbCr fields, along with some private tags that OMF Interchange adds to extend TIFF to represent video. There are restrictions on some of the standard TIFF fields, but these restrictions limit only the kinds of data that can be written. They do not modify the meanings of the tags in any way.

Additional IFD Fields

OMF Interchange defines the following additional IFD fields.

- `VideoFrameOffsets`

  Tag = `34432` (8680.H)

  Type = `LONG`

  N = number of frames

  This field contains an array of byte offsets to the video frames. The order of frames in time corresponds to the order of the offsets.

  If this field is not present, the file cannot be interpreted as video.

- `VideoFrameByteCounts`

  Tag = `34433` (8681.H)

  Type = `LONG`

  N = number of frames

  This field contains an array of byte lengths for the video frames. The order of frames in time corresponds to the order of the lengths.

  If this field is not present, the lengths have to be computed by interpreting the video data.

- `VideoFrameLayout`

  Tag = `34434` (8682.H)

  Type = `SHORT`

  N = `1`

  This field contains the layout of the pixels with respect to a broadcast video image.

  `1`= Full Frame picture.
  The image is not separated into fields, and represents the entire picture.

  `2`= Single Field, Odd.
  The image corresponds to the odd broadcast field, and must be doubled to be displayed on a monitor.

  `3`= Single Field, Even.
  The image corresponds to the even broadcast field, and must be doubled to be displayed on a monitor.

  `4`= Mixed fields.
  The image contains two fields, the odd field first, followed by the even field. Together, the two images make up a complete broadcast image.

- `VideoFrameRate`

  Tag = `34435` (8683.H)

  Type = `RATIONAL`

  N = `1`

  This field contains the sample rate of the video data. Typical values would be 2997/100 for NTSC, or 25/1 for PAL.

- `JPEGQTables16`

  Tag = `34436` (8684.H)

  Type = `LONG`

  N = number of components

This field points to a list of offsets to high-precision quantization tables, one per component. Each table consists of 128 BYTES (one 16-bit value for each DCT coefficient in the 8x8 block). The quantization tables are stored in zig-zag order.

The `JPEGQTables` field is still required, as they are mandatory with the JPEG extension. Applications that recognize the 16-bit tables can use them if they choose to.

OMF Interchange imposes the following restrictions on the representation of video TIFF/JPEG data. These restrictions limit what kinds of data can be written, but they do not modify the meanings of the tags in any way.

- Lossless JPEC (`JPECProc` = 14) is not supported; `JPECProc` must equal 1.
- `JPEGInterchangeFormat` and `JPEGInterchangeFormatLength` are not supported.
- `PlanarConfiguration` must be 1; only interleaved components are supported.
- Only one strip per image is allowed; `RowsPerStrip` must equal the length of the images.
- Tiles are not supported.
- `Compression` must be `1` or `6` (none or JPEG).
- If Compression is `6` (JPEG), then `PhotometricInterpretation` must be `6` (YC_bC_r).
- YC_bC_r `Subsampling` must be `(2,1)`; this correspond to the industry-standard format YUV422.
- `JPEGRestartInterval` is not allowed; there must be explicit markers in the frame data to prevent wrong interpretation of the data.
- All frames must conform to the values specified in the IFD, such as height and length.

Appendix D

Changes in Version 2.1

List of Changes

The following sections summarize the technical changes in OMF class hierarchy in this version of the specifications.

File Version Number

The changes introduced the OMF Interchange Specification Version 2.1 are minor enhancements to Version 2.0. Consequently, the file version number stored in the Header object remains 2.0 and is not changed. Any application that is built using the OMF Interchange Developers' Toolkit Version 2.0 should be able to read a file that conforms to the OMF Interchange Specification Version 2.1 with a few specific exceptions. The exceptions are that it may ignore the identification information in the Header object, that it cannot handle any data item or frame index that requires a 64-bit integer, and that it will treat Pulldown objects as unknown objects.

What's New?

The following are new features:

- Identification information in the Header object
  - New `OMFI:HEAD:IdentificationList` property
  - New Identification (IDNT) class
  - New `omfi:ProductVersion` data type

- Support for large media files (greater than 2 gigabytes)
  - The following properties can have either an `omfi:Length32` or an `omfi:Length64` data type: `OMFI:CPNT:Length`, `OMFI:MDFL:Length`, `OMFI:CMOB:DefFadeLength`, `OMFI:SCLP:FadeInLength`, and `OMFI:SCLP:FadeOutLength`
  - The following properties can have either an `omfi:Position32` or an `omfi:Position64` data type: `OMFI:ECCP:Start`, `OMFI:ERAT:InputOffset`, `OMFI:ERAT:ResultOffset`, `OMFI:SCLP:StartTime`, `OMFI:TCCP:Start`, and `OMFI:TRKD:Origin`
  - The `OMFI:JPEG:FrameIndex` property can have either an `omfi:Position32Array` or an `omfi:Position64Array` data type
- Support for user comments on Mobs and tagged user-defined information
  - New `OMFI:MOBJ:UserAttribute` property
  - New Attribute (ATTB) class
  - New Attribute Array (ATTR) class
  - New `omfi:AttrKind` data type
- Improved support for film pulldown conversions
  - New Pulldown (PDWN) class
  - New `omf:PulldownKindType`, `omfi:PulldownDirection-Type`, and `omfi:PhaseFrameType` data types
- Minor improvements for media handling and storage
  - Default audio fades in Composition Mobs; new `OMFI:CMOB:Def-FadeLength`, `OMFI:CMOB:DefFadeType`, and `OMFI:CMOB:Def-FadeEditUnits` properties
  - Ability to store the film edgecode header; new `OMFI:ECCP:Header` property
  - Ability to specify CDCI padding bits; new `OMFI:CDCI:Padding-Bits` properties
  - Ability to specify physical track; new `OMFI:TRKD:PhysicalTrack` property
  - Ability to specify image geometry using signed offsets; changed data type for `OMFI:DIDD:SampledXOffset`, `OMFI:DIDD:SampledY-Offset`, `OMFI:DIDD:DisplayXOffset`, and `OMFI:DIDD:Dis-playYOffset`
  - Improved set of Locators; new Network (NETL) locator, new `OMFI:MACL:PathName`, and `OMFI:WINL:PathName` properties

What's Removed?

A few effect definitions have been removed from the Effect Dictionary. The omfi:effect:VideoFrameMask effect has been replaced by the Pulldown object. The stereo audio effects have been removed.

Changes in Version 2.0

This section describes the changes between Version 1.0 and Version 2.0.

The major goals for Version 2.0 of the OMF Interchange specification are to increase the capabilities available to the end user and to reduce the roadblocks to digital media data interchange. This version of the specification helps achieve these goals in the following ways:

- Version 2.0 allows more kinds of information to be interchanged between applications. The most important new kinds of information in this version are
  - — Effects
  - — More comprehensive descriptions of image and video media
- Version 2.0 makes it easier for an application to include OMF support, which subsequently increases the number of applications available to the end user. The major changes in this version that make it easier to include OMF support are
  - — Better class model
  - — Better description of Mobs and compositions
- Version 2.0 enhances existing OMF capabilities and remedies problems found in earlier versions.

Interchanging Effects

OMF Version 2.0 allows interchange of effects, where as the previous version allowed interchange for only a very limited number of transition effects. Effects represent a major part of the value added to the media during the post-production process. Effects are used as transitions, to layer segments of media, and to alter segments of media. By allowing effects that are defined in one application to be generated or altered in another, OMF reduces the amount of work that must be duplicated when going from one application to another.

The OMF Interchange file has a new model for interchanging effects. The new effect model allows an effect generated by one application to be understood, viewed and edited by a second application, significantly simplifying media production in a multi-vendor environment. Editors will be able to describe and preview effects on a nonlinear editing system, render and alter them on a high-end system and then send them back to the editing system for finishing. Because the OMF specification represents all effects in a common way, details are retained throughout the transfer process, ultimately saving editors time and money. Developed in close cooperation with OMF Champions and Partners, the new effect model offers post professionals quicker, more collaborative effects generation.

OMF 2.0's effect representation is sample rate and resolution independent, and therefore portable to media formats like NTSC, PAL or film, and to different

play rates. Because effects are represented descriptively rather than rendered as video, the importing applications can alter parameters to modify the effect, such as make a page turn faster or alter the curve of the page.

There are also well-defined mechanisms to include rendered versions of effects. This allows applications that cannot generate an effect to play a prerendered version of the effect.

Enhancing Media Support

Certain areas of the 1.0 version of the specification have been significantly improved as part of OMF 2.0. These areas include: describing video image media, handling origins of media with different sampling rates, describing media with interleaved channels, and describing different representations of the same media.

New Image Format

This version includes a new image format (for moving and still images) that supersedes and enhances the functionality provided with the TIFF image format used in the 1.0 version of the OMF specification. The TIFF format is still supported in OMF 2.0 for compatibility with version 1.0 OMF images.

The goal of the new image format is to simplify the representation of image data and to be able to store the information required by video formats in common use. The Version 2.0 image format supports component video, which is not supported by the previous version's TIFF format. It also eliminates redundancy and is an efficient storage format for both single and moving images. It can easily support compressed and uncompressed video, store images in either a color difference or component video format, and provide additional information about the sampling process used to create the digital media from an analog source.

Applications that support OMF must support the TIFF format as well as the new image format during the 1.0 to 2.0 transition period (this functionality will be provided in the OMF 2.0 Toolkit). This will allow complete compatibility with applications that are still using the 1.0 format. After the transition period ends, which is a year after the publication of the version 2.0 specification, applications will not be required to continue to support the TIFF format.

Better Support for Media

This version of OMF Interchange has better support for media with different native sampling rates, media with interleaved channels, and media with more than one digital representation. The starting time for each track of media can now be specified independently. This allows sample-accurate starting times to be specified for each kind of media.

New structures allow you to include media data that interleaves multiple channels. For example, media data that has interleaved stereo audio contains the information required to play two channels of audio.

Sometimes it is desirable to have more than one digital representation for media. For example, media may need to be stored differently on different systems for playing efficiency, or media may be stored with different compressions depending on the storage available and the stage of production.

Making It Easier to Add OMF Support

Version 2.0 of the OMF specification (along with the OMF 2.0 Toolkit) is designed to make it easier to add OMF support to applications by providing more thorough descriptions and information. The improvements added to this version were chosen based on the feedback from the OMF Champions who have already added OMF support to their applications. This version of the OMF Interchange Specification:

- Provides a new class model that makes it possible for a more rigorous definition of the legal structure of OMFI files.
- Provides more information on defining compositions
- Provides better explanations and examples
- Eliminates some object classes

List of Changes

The following sections summarize the technical changes in OMF class hierarchy in this version of the specifications.

What's New?

The following new classes are used to describe effects:

- Effect Definition (EDEF) class
- Effect Invocation (EFFE) class
- Effect Slot (ESLT) class
- Varying Value (VVAL) class
- Constant Value (CVAL) class
- Control Point (CTLP) class
- Data Definition (DDEF) class

The following new features also help provide effect interchange:

- Extended set of data types and data kinds
- Effects Dictionary

The following new classes are used for the new image format:

- Media Data (MDAT) class
- Image Data (IDAT) class
- JPEG Image Data (JPEG) class
- Digital Image Descriptor(DIDD) class
- RGBA Component Image Descriptor (RGBA) class
- Color Difference Component Image Descriptor (CDCI) class

The following new classes are used to improve references from one section of a Composition Mob to another:

- Scope Reference (SREF) class
- Nested Scope (NEST) class
- Mob Slot (MSLT) class
- Edit Rate Converter (ERAT) class
- Track Description (TRKD) class

The following new class allows a Master Mob to provide access to multiple implementations of the same media:

- Media Group (MGRP) class

The following new classes allow you to specify more descriptive information about physical sources and include Windows format locator information:

- Media Film Descriptor (MDFM) class
- Media Tape Descriptor (MDTP) class
- Windows Locator (WINL) class

The following classes help provide a better class hierarchy, which makes OMF easier to understand and use:

- Composition Mob (CMOB) class
- Master Mob (MMOB) class
- Source Mob (SMOB) class
- OMFI Object (OOBJ) class
- Segment (SEGM) class

Changed Properties

This section highlights property changes for classes that are in both 1.0 and 2.0.

The `OMFI:ObjID` property has been renamed the `OMFI:OOBJ:ObjClass` property, which eliminates the concept of generic properties and makes the class model easier to understand. The name `ObjClass` is more descriptive because the property identifies the class of the object rather than containing an ID number for the object.

The `OMFI:Version` property has been removed from all objects except the HEAD object. All objects in a single OMFI file must conform to the same version of the specification.

The OMFI:ObjectSpine and OMFI:ExternalFiles properties have been removed from the HEAD object. Instead of using the object spine to find objects, you use the mob and media data indexes in the HEAD object. The external files index provided information that is easily accessible through the mob indexes. The Mobs index and the MediaData are now arrays of object references rather than also containing the MobID values. This reduces the redundancy in the OMFI file, which minimizes the possibility of creating inconsistent OMFI files.

The omfi:TrackType data type is no longer an enumerated data type. It is now an object reference to a Data Definition object.

The OMFI:TRAK:FillerProxy property has been replaced with Nested Scope and Scope References. The Filler class is not used for implicit references.

What's Removed?

The following object classes are not included in OMF Interchange Specification Version 2.0:

- The TRKG track group class is replaced by Effect Invocation (EFFE), Nested Scope (NEST), and Media Group (MGRP) classes. In addition, the following subclasses of TRKG are also replaced by EFFE: WARP time warp class, SPED speed class, MASK mask class, REPT repeat class.
- The ATTR attribute array class, the ATTB attribute class, and the ATCP clip attribute class have been removed. These were intended to store private information. Effects interchange in Version 2.0 eliminates the need for some of this private data. The remaining private data can be stored using private properties or new private classes.
- The TRAK class has been replaced by the Mob Slot (MSLT) and the Track Description (TRKD) classes.
- The TRKR class has been replaced by the Scope Reference (SREF) class.

Appendix E

Effects Dictionary

This appendix describes the registered effects that were specified at the time this document was published. Contact the OMF Developers' Desk for documentation on additional registered effects.

This section provides documentation on the following audio effects:

- Mono Audio Dissolve
- Mono Audio Gain
- Mono Audio Mixdown
- Mono Audio Pan

This section provides documentation on the following video effects:

- SMPTE Video Wipe
- Video Dissolve
- Video Fade To Black
- Video Frame Mask
- Video Repeat
- Video Speed Control

The Effect Definition section in each dictionary page lists the values of the Effect Definition (EDEF) properties. It also lists the result data kind of the effect. Effects dictionary pages are arranged alphabetically by the effect name.

Mono Audio Dissolve Effect

Combines two mono audio streams.

Effect Definition

EffectID	omfi:effect:SimpleMonoAudioDissolve
EffectName	Simple Mono Audio Dissolve
EffectDescription	Combines two mono audio streams by using a simple linear equation.
Bypass	-1 (in Transitions, substitute a cut)
IsTimeWarp	False
Result Data Kind	omfi:data:Sound

Control Arguments

ArgID	Data Kind	Description
-1	omfi:data:Sound	Input audio Segment "A", may also be known as "outgoing" for Transitions. This control is required. If it is not present, then the effect must be a transition effect, so derive the control from the preceding Segment in the containing sequence. If the control is not present, and the effect is not in a transition, the effect is in error.
-2	omfi:data:Sound	Input audio segment "B", may also be known as "incoming" for transitions. This control is required. If it is not present, then the effect must be a Transition effect, so derive the control from the preceding segment in the containing sequence. If the control is not present, and the effect is not in a transition, the effect is in error.
-3	omfi:data:Rational	Level, equal to mix ratio of B/A. Range is 0 to 1. The formula: P = (Level*B) + ((1-Level)*A) This control is optional. The default for Effect Invocations that are not in Transitions is a constant 1/2. The default for Effect Invocations in Transition is a VVAL with two control points: Value 0 at time 0, and value 1 at time 1.

Notes

This effect is intended primarily for audio Transitions, but it can be used outside of Transitions also.

Mono Audio Gain Effect

Adjusts volume of mono audio Segment.

Effect Definition

EffectID	omfi:effect:MonoAudioGain
EffectName	Mono Audio Gain
EffectDescription	Adjusts the volume of an audio Segment.
Bypass	-1
IsTimeWarp	False
Result Data Kind	omfi:data:Sound

Control Arguments

ArgID	Data Kind	Description
-1	omfi:data:Sound	Input audio segment.
1	omfi:data:Rational	Amplitude multiplier. Range is from 0 to 2^{32}-1. Unity gain is 1.

Notes

This effect is not allowed in Transitions.

Mono Audio Mixdown Effect

Mixes any number of mono audio Segments into a single mono audio output Segment.

Effect Definition

EffectID	omfi:effect:MonoAudioMixdown
EffectName	Mono Audio Mixdown
EffectDescription	Combines any number of mono audio Segments into a single mono audio output.
Bypass	1
IsTimeWarp	False
Result Data Kind	omfi:data:Sound

Control Arguments

ArgID	Data Kind	Description
1	omfi:data:Sound	First input mono audio Segment. Required except when effect is used in a Transition.
2	omfi:data:Sound	Second input mono audio Segment. Optional.
n	omfi:data:Sound	Nth input mono audio Segment. Optional.

Notes

1. All Effect Slots must have the data kind omfi:data:Sound or a data kind that can be converted to omfi:data:Sound.
2. The audio Segments are added together.
3. In a Transition, no Effect Slots may be specified. The effect mixes the overlapping sections of the adjacent Segments.
4. Except when the effect is used in a Transition, there must be at least one Effect Slot.
5. The ArgID values specified in all Effect Slots must form a set of sequential integers starting at 1 (no duplicates are allowed). Since the Effect Invocation has an unordered set of Effect Slots, they can appear in any order.

Mono Audio Pan Effect

Converts a mono audio Segment into stereo audio. The amount of data sent to each channel is determined by the pan control.

Effect Definition

`EffectID`	`omfi:effect:MonoAudioPan`
`EffectName`	Mono Audio Pan
`EffectDescription`	Splits a mono audio Segment into a L/R stereo pair.
`Bypass`	`-1`
`IsTimeWarp`	False
Result Data Kind	`omfi:data:StereoSound`

Control Arguments

ArgID	Data Kind	Description
`-1`	`omfi:data:Sound`	Specifies the input audio Segment.
`1`	`omfi:data:Rational`	Specifies the pan value; it has a range from 0.0 to 1.0. This value represents the ratio of left/right apportionment of sound. A value of 0 is full left; a value of 1 is full right, and a value of 1/2 is half left and half right.

Notes

This effect is not allowed in Transitions.

SMPTE Video Wipe Effect

Combines two video streams by using a SMPTE video wipe.

Effect Definition

EffectID	omfi:effect:SMPTEVideoWipe
EffectName	SMPTE Video Wipe
EffectDescription	Combines two video streams according to the SMPTE Recommended Practice for Transfer of Edit Decision Lists.
Bypass	-1
IsTimeWarp	False
Result Data Kind	omfi:data:Picture

Control Arguments

ArgID	Data Kind	Description
-1	omfi:data:PictureWithMatte	Input video segment “A”, may also be known as “outgoing” for transitions. This control is required. If it is not present, then the effect must be a transition effect, so derive the control from the preceding segment in the containing sequence. If the control is not present, and the effect is not in a transition, the effect is in error.
-2	omfi:data:PictureWithMatte	Input video segment “B”, may also be known as “incoming” for transitions. This control is required. If it is not present, then the effect must be a transition effect, so derive the control from the preceding segment in the containing sequence. If the control is not present, and the effect is not in a transition, the effect is in error.
-3	omfi:data:Rational	Level, equal to “percentage done.” Range 0 to 1.
1	omfi:data:Int32Boolean	SMPTE Wipe Number. No default, must be specified.
2	omfi:data:Boolean	Reverse, default FALSE.
3	omfi:data:Boolean	Soft, default FALSE.
4	omfi:data:Boolean	Border, default FALSE.
5	omfi:data:Boolean	Position, default FALSE.
6	omfi:data:Boolean	Modulator, default FALSE.
7	omfi:data:Boolean	Shadow, default FALSE.
8	omfi:data:Boolean	Tumble, default FALSE.
9	omfi:data:Boolean	Spotlight, default FALSE.
10	omfi:data:Int32	ReplicationH.

ArgID	Data Kind	Description
11	omfi:data:Int32	ReplicationV
12	omfi:data_Boolean	Checkerboard

Notes

See the SMPTE Recommended Practice, Transfer of Edit Decision Lists, RP-146-1987 more for information on SMPTE wipes.

Video Dissolve Effect

Combines two video streams by using a simple linear equation.

Effect Definition

`EffectID`	`omfi:effect:SimpleVideoDissolve`
`EffectName`	Simple Video Dissolve
`EffectDescription`	Combines two video streams by using a simple linear equation.
`Bypass`	`-1` (in Transitions, substitute a cut)
`IsTimeWarp`	False
Result Data Kind	`omfi:data:PictureWithMatte`

Control Arguments

ArgID	Data Kind	Description
`-1`	`omfi:data:PictureWithMatte`	Input video Segment "A", may also be known as "outgoing" for Transitions. This control is required. If it is not present, then the effect must be a transition effect, so derive the control from the preceding segment in the containing sequence. If the control is not present, and the effect is not in a transition, the effect is in error.
`-2`	`omfi:data:PictureWithMatte`	Input video Segment "B", may also be known as "incoming" for Transitions. This control is required. If it is not present, then the effect must be a Transition effect, so derive the control from the preceding segment in the containing sequence. If the control is not present, and the effect is not in a transition, the effect is in error.
`-3`	`omfi:data:Rational`	Level, equal to mix ratio of B/A. Range is 0 to 1. For each pixel in the result, the output pixel value P is computed by combining the input pixels A and B with the formula: P = (Level*B) + ((1-Level)*A) This control is optional. The default for segment effects is a constant 1/2. The default for transition effects is a VVAL with two control points: Value 0 at time 0, and value 1 at time 1.

Notes

This effect is intended primarily for video Transitions, but can also be used outside of Transitions.

Video Fade To Black Effect

Combines a video stream with black by using a simple linear equation.

Effect Definition

EffectID	omfi:effect:VideoFadeToBlack
EffectName	Video Fade to Black
EffectDescription	Combines a video stream with black by using a simple linear equation
Bypass	-1
IsTimeWarp	False
Result Data Kind	omfi:data:PictureWithMatte

Control Arguments

ArgID	Data Kind	Description
-1	omfi:data:PictureWithMatte	Input video Segment A.
1	omfi:data:Rational	Level, equals mix ratio of Black/A. For each pixel in the result, the output pixel value P is computed by combining the input pixel A and black pixel Black with the formula: P = (Level*Black) + ((1-Level)*A) This control is optional. If Reverse is FALSE, the default for segment effects is a VVAL with two control points: Value 0 at time 0, and value 1 at time 1. If Reverse is TRUE, the default for segment effects is a VVAL with two control points: Value 1 at time 0, and value 0 at time 1.
2	omfi:data:Boolean	Reverse (fade from black). If FALSE, then start with the input video, finish with black. If TRUE, start with black, end with the input video. This control must be constant (CVAL). This control is optional. The default is FALSE.

Notes

1. This effect must map control time 0 to the start of the first frame, and control time 1 to the start of the last frame.

Video Pull-Down Frame Mask Effect

Maps an input video Segment to an output video Segment according to a cyclical pattern.

Pulldown objects should be used to describe film to video and video to film conversions. Existing applications will continue to use the Video Frame Mask effect to describe pulldown, but future releases should use the Pulldown object.

Effect Definition

EffectID	omfi:effect:VideoFrameMask
EffectName	Video Frame Mask
EffectDescription	Changes a Segment of video by duplicating or subtracting frames according to the repeating bits in a control mask.
Bypass	None, use Filler
IsTimeWarp	True
Result Data Kind	omfi:data:PictureWithMatte

Control Arguments

ArgID	Data Kind	Description
-1	omfi:data:PictureWithMatte	Input video Segment. This control is required.
1	omfi:data:UInt32	Pull-down mask bits. The bits are processed in order, from the most-significant bit down toward the least-significant bit as follows: If at any time all of the remaining bits in the mask are 0, then processing resets to the first bit in the mask. This value must be constant (CVAL). This control is required, and must not equal 0.
2	omfi:data:UInt8	Phase offset. Indicates where (as an offset in bits from the most-significant bit) to start reading the mask on the first cycle. Range is 0 to 31. This value must be constant (CVAL). This control is optional, the default value is constant 0.
3	omfi:data:Boolean	Add/Drop indicator. TRUE means when a 1-bit is reached in the mask, hold the previous frame. FALSE means when a 1-bit is reached in the mask, skip to the next frame. In both cases, a 0-bit indicates that the input frame should be copied to the output segment. This value must be constant (CVAL). This control is required.

Notes

1. Control 2 (Add/Drop) is the general indicator of the ratio. If the value is TRUE, the output will have more edit units than the input. If the value is FALSE, the output will have fewer edit units than the input.
2. This is a single-video-input effect, and is not legal as a transition effect.
3. In Version 2.1 this effect has been replaced by the Pulldown object.

Video Repeat Effect

Repeats a segment of video for a given amount of time.

Effect Definition

`EffectID`	`omfi:effect:VideoRepeat`
`EffectName`	Video Repeat
`EffectDescription`	Loops a segment of video as long as needed to fill the output duration.
`Bypass`	None, use Filler
`IsTimeWarp`	True
Result Data Kind	`omfi:data:PictureWithMatte`

Control Arguments

ArgID	Data Kind	Description
`-1`	`omfi:data:PictureWithMatte`	Input video segment. This control is required.
`1`	`omfi:data:UInt32`	Phase Offset. Indicates where (as an offset) to start reading the input segment on the first loop. Range is 0 to the length of the input segment. This control must be constant (CVAL). This control is optional, the default value is constant 0.

Notes

This is a single-video-input effect, and is not legal as a transition effect.

Video Speed Control Effect

Changes the speed at which the media is to be played by duplicating or eliminating edit units by using a constant ratio.

Effect Definition

`EffectID`	`omfi:effect:VideoSpeedControl`
`EffectName`	Video Speed Control
`EffectDescription`	Changes a Segment of video by duplicating or subtracting frames according to the repeating bits in a control mask.
`Bypass`	None, use Filler
`IsTimeWarp`	True
Result Data Kind	`omfi:data:PictureWithMatte`

Control Arguments

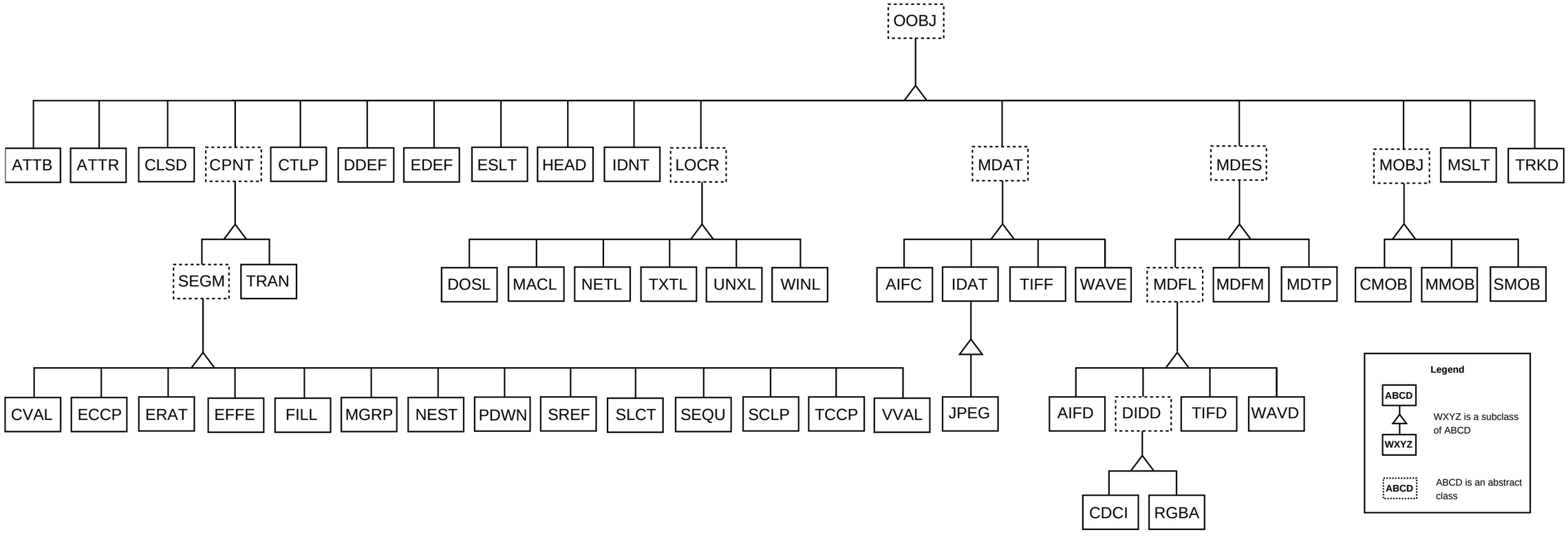
ArgID	Data Kind	Description
`-1`	`omfi:data:PictureWithMatte`	Input video segment. This control is required.
`1`	`omfi:data:Rational`	Edit ratio. Defines the ratio of output length to input length. Range is -infinity to +infinity. For example, a ratio of 2/1 means that the output is twice as long as the input, appearing as a slow motion effect. A ratio of 1/2 would appear as fast motion since the input segment would be reduced to half its time. A negative value means that the frames are played in reverse order. A ratio of -1/1 would appear as backwards but at the normal frame rate. A ratio of -1/2 would appear as backwards and fast motion. This value must be constant (CVAL). This control is required.
`2`	`omfi:data:UInt32`	Phase Offset. Specifies that the first edit unit of the output is actually offset from the theoretical output. For example, if an input ABCD gets converted with a 2/1 ratio to AABBCCDD, a phase offset of 1 would make the output ABBCCDD. This control must be constant (CVAL). This control is optional, the default value is constant 0.

Notes

This is a single-video-input effect, and is not legal as a Transition effect.

Appendix F

Class Hierarchy



Class ID	Class Name
ATTB	Attribute
ATTR	Attribute Array
AIFC	AIFC Audio Data
AIFD	AIFC Audio Descriptor
CDCI	Color Difference Component Image Descriptor
CLSD	Class Dictionary
CMOB	Composition Mob
CPNT	Component (abstract)
CTLP	Control Point
CVAL	Constant Value
DDEF	Data Definition
DIDD	Digital Image Descriptor (abstract)
DOSL	DOS Locator
ECCP	Edge Code

Class ID	Class Name
EDEF	Effect Definition
EFFE	Effect Invocation
ERAT	Edit Rate Converter
ESLT	Effect Slot
FILL	Filler
HEAD	Header
IDNT	Identification
IDAT	Image Data
JPEG	JPEG Image Data
LOCR	Locator (abstract)
MACL	MAC Locator
MDAT	Media Data (abstract)
MDES	Media Descriptor (abstract)
MDFL	Media File Descriptor (abstract)

Class ID	Class Name
MDFM	Media Film Descriptor
MDTP	Media Tape Descriptor
MGRP	Media Group
MMOB	Master Mob
MOBJ	Mob (abstract)
MSLT	Mob Slot
NEST	Nested Scope
NETL	Network Locator
OOBJ	OMFI Object (abstract)
PDWN	Pulldown
RGBA	RBGA Component Image Descriptor
SCLP	Source Clip
SEGM	Segment (abstract)
SEQU	Sequence

Class ID	Class Name
SLCT	Selector
SMOB	Source Mob
SREF	Scope Reference
TCCP	Timecode
TIFD	TIFF Image Descriptor
TIFF	TIFF Image Data
TRAN	Transition
TRKD	Track Description
TXTL	Text Locator
UNXL	UNIX Locator
VVAL	Varying Value
WAVD	WAVE Audio Descriptor
WAVE	WAVE Audio Data
WINL	Windows Locator

Glossary

abstract class	An abstract class provides a way to refer to a group of classes. An *object* belonging to an abstract class must also belong to a nonabstract class that is a *subclass* of the abstract class.
AIFC	The version of the Audio Interchange File Format (AIFF) for both compressed and uncompressed audio data in big endian byte order. OMF Interchange includes AIFC as a common interchange format for uncompressed audio data.
Bento	A general container format and software API developed by Apple Computer, Inc. OMF Interchange uses Bento as a storage and access system for the information in an OMF Interchange file.
class	A class is a category of *objects*. The objects have common properties, relationships, and semantics. OMF objects do not have behavior because methods (code to perform actions on objects) are not stored in the object. Applications using OMF objects must supply the behavior based on the OMF Interface Specification.
class dictionary	A property in an OMF Interchange file header for extending the class hierarchy. Applications can add a private class dictionary object that extends the common hierarchy of classes and subclasses.
class hierarchy	The hierarchical list of object classes and subclasses in OMF Interchange that determines the inherited properties of each class.
Component	An abstract class of objects in the OMF Interchange class hierarchy representing the basic time-varying building block of a composition.
composition	An OMF Interchange data structure that logically represents the organization of all of the components in a time-based media presentation. It contains all the information necessary for playing or editing a media presentation. A composition does not contain digital media data. Instead, it includes references to physical sources, including both sources of digital data and the original sources.
Composition Mob	A mob that describes a composition.
data model	The data model is the high-level description that specifies the logical and semantic meaning. In contrast, the *implementation* is the lower-level description of a class that specifies the storage details.
digital media data	Digital data stored in a file. It can be either data that was digitized, such as video frame data and audio samples, or data created in digital form, such as title graphics or animation frames. It can be stored in either a *Media Data object* or a *raw data file*.

dissolve	A transition video effect in which the pixel-by-pixel display of one segment is gradually interspersed with and then replaced by the display of another segment.
DVE	Digital video effect.
EDL	See *edit decision list*.
edge code	Also called edge number and key number. An alphanumeric identifier that a film manufacturer places along the edge of film at regular intervals to identify the frames. One edge number identifies a series of frames. See also *frame offset*.
edit decision list (EDL)	A list of edits made during offline editing and used to direct the online editing of the master.
edit rate	In compositions, a measure of the number of editable units per second in a piece of media data. See also *edit unit*; compare with *sample rate*.
edit unit (EU)	A unit of duration representing the smallest interval of time that is recognized by a composition. A composition's content determines the edit unit. For example, a composition describing an NTSC presentation would represent both video data and audio data in units corresponding to the length of a video frame display rate of 29.97 frames per second.
effects	Composition information for combining media tracks and altering the playback of media tracks, such as wipes, dissolves, or volume control data. A composition can include transition effects and track effects.
file header	An object in each OMF Interchange file containing file-wide information. It can have properties that provide efficient access to all the mobs in the file and to all the top-level objects in the file.
file Source Mob	A mob for digital media data. A file Source Mob can appear in an OMF Interchange file along with the sample data it identifies. If the file Source Mob and sample data are in an external file, any file containing a composition that uses the data must include a copy of the file Source Mob. Applications cannot update or change the data associated with file Source Mobs; they must create a new file Source Mob when the information changes.
Filler	A component in a composition that represents an empty portion of a track.
fps	Frames per second; a measure of the video display rate.
frame offset	A way of indicating a particular frame within the group of frames identified by the edge number on a piece of film. For example, a frame offset of +12 indicates the twelfth frame from the frame marked by the edge code.
freeze frame	A name sometimes used for a video effect that appears to stop the action. Compositions can create this effect using a track repeat object, which can specify the display of a single frame for a period of time.
HAS	The HAS relationship is between an *object* and a property *value*. A value can be simple, such as a number or a string, or can be another object. If the value is another object, then that object is owned by the object that HAS it. In the data model diagrams, the HAS relationship is indicated by a solid line from a property to an object. The HAS relationship is also called CONTAINS or OWNS.
HAS-REFERENCE	The HAS-REFERENCE relationship is between an *object* and a property *value* that is another object. An object does not own the other object that it has a reference to, and more than one object can have a reference to a single object. In the data model diagrams, the HAS-REFERENCE relationship is indicated by a

	dashed line from a property to an object. The HAS-REFERENCE relationship is also called DEPENDS-ON or USES.
implementation	The implementation is the lower-level description of a class that specifies the storage details. In contrast, the *data model* is the high-level description of a class that specifies meaning.
inheritance	Inheritance is the mechanism that defines a relationship between *classes* where a *subclass* inherits the *properties,* relationships, and semantics of its superclass.
is-an-instance-of	The is-an-instance-of relationship is a relationship between an *object* and a *class*. An object Is-an-instance-of a class if the object is in the *set* defined by the class.
is-a-kind-of	The is-a-kind-of relationship is a relationship between two *classes*. If a class has the properties, relationships, and semantics of a second class, then the first class is-a-kind-of the second class. The first class is called the *subclass* and the second class is the *superclass*.
JFIF	JPEG File Interchange Format, a compressed image file format.
JPEG	Joint Photographic Experts Group.
layered media	A method of creating an effect by combining two Segments of media in a specified way. For example an application can use this method to superimpose a chroma key video Segment over another video Segment.
Locator	An object that contains information an application or its user could use to help locate and identify a source of media data.
media	The video, audio, graphics, computed effects that combine to form a presentation.
media data	Data from a media source. It can be analog data, such as film frames, tape audio, or videotape video and audio. It can be digital data; either data that was digitized, such as video frame data and audio samples, or data created in digital form, such as title graphics or animation frames. See also *digital media data*.
Media Data object	An OMFI *object* that contains *digital media data*.
Media Descriptor	The element in Source Mob that describes the data from the source. For example, a Media Descriptor for video data describes the format of the data and the compression type. A Media Descriptor for digital data in a file can have location hints.
media source	See *physical source*.
media sample data	See *sample data*.
media data index	An property in the file header; it is an index to the objects in the file that contain digital media data.
Mob	A primary OMF Interchange data structure that includes a unique ID and can identify a composition or a physical source of media data. The use of mobs makes it possible to create compositions separately from the digital media data and to store source information applications can use to recreate media. Composition mobs store persistent references to their physical source mobs. A digital file mob can store a reference to the videotape mob describing its original analog source.
mob ID	The unique ID associated with an OMF Interchange file Mob, having OMF Interchange data type `omfi:UID`.

motion effect	An effect that speeds up or slows down the presentation of media in a track.
NAB	National Association of Broadcasters.
NTSC	National Television Standards Committee, which established the color TV transmission system used in the U.S.
NTSC video	The color video standard established by the National Television Standards Committee. This standard calls for 525 lines of information, scanned at a rate of approximately 29.97 frames per second.
object	An object is a member of a *class*. It has a collection of *properties*, each of which has a name and a value. An object is-an-instance-of of a class.
object reference	The file-specific, unique, persistent ID value an application uses to access an object in an OMF Interchange file. During processing, an application can convert an object reference into a pointer to the object, such as a Source Clip.
ordered set	An ordered *set* is an ordered collection of unique values. This is sometimes used in *class* definitions to store multivalued properties when ordering is important.
origin	A reference point for measuring sections of digitized sample data. A file mob value for the start position in the media is expressed in relation to the origin. Although the same sample data can be redigitized and more sample data might be added, the origin remains the same so that composition source clips that reference it remain valid.
PAL	Phase Alternating Line, a color TV standard used in many countries. PAL calls for 625 lines, scanned at a rate of 25 frames per second.
physical Source Mob	A mob that represents a physical source, such as a videotape or an audio tape.
physical source	The physical source of digital media data. A physical source can be an analog source from which media data is digitized, such as a videotape, an audio tape, or a film reel. It can also be a source that is digital in its original form, such as animation frames or a graphics file.
property	Identifies an object's value and the data type of the value. Objects have their own set of required or optional properties. Many objects have properties that are inherited from a another class that is higher up in the class hierarchy.
raw media file	A file containing *digital media data* that does not contain any OMFI objects.
repeat effect	A type of effect for repeating a frame, so that it appears to "freeze" or stop the frame, or for repeating a series of frames, such as a series of animation frames.
rendered media	A computed effect stored in a file and referenced by a composition. Applications can render effects that they cannot create in during playback.
RGB	RGB pixel arrays, a widely used file format for representing the primary colors red, green, and blue image data for digital use.
RIFF WAVE	See *WAVE*.
sample data	Media data created by digitizing from a physical source. A sample is a unit of data that the digitizing device can measure. Applications can play digital sample data from files on disk.
sample rate	The frequency of the sample units.

sample units	A unit of measure used in digitizing media data from a physical source, such as a videotape. Media data contains its own sample rate and the size of each sample in bytes.
SECAM	Séquential Couleur á Memoire, a color TV standard developed in France and used there and in other countries.
Segment	Within a composition, a Segment is specifies time-varying media or other information. A Segment can be used alone in contrast with a Transition, which can only be used when surrounded by Segments in a Sequence. A typical Segment is a Source Clip.
Sequence	An *ordered list* of Segments that are optionally connected by Transitions.
set	A set is an unordered collection of unique values. This is sometimes used in *class* definitions to store multivalued *properties*.
SMPTE	The Society of Motion Picture and Television Engineers.
SMPTE timecode	A frame numbering system developed by SMPTE and used primarily for electronic editing and timing of video programs. It assigns a number to each frame of video, telling the number of hours, minutes, seconds, and frames; for example, 01:42:13:26.
Source Clip	One of the lowest level building blocks of a composition. It identifies the mob ID of a physical source and describes a section of the source.
subclass	A subclass is a class that is defined as having the properties, relationships, and semantics as another class, which is called its *superclass*. The subclass can have additional properties, relationships, and semantics that are not in the superclass.
substitutability	The rule of substitutability specifies that an *object* can be used in place of an object of any *class* that is a *superclass* of its class.
superclass	A superclass is a class that has another class, its *subclass,* that is defined as having the properties, relationships, and semantics as the superclass.
TIFF	A tag-based file format for storing and interchanging raster images developed by Aldus Corporation. The OMF Interchange standard includes TIFF as a common format for graphic interchange, and it includes TIFF with extensions as a common format for video frame data.
time code	Usually SMPTE timecode, but media data can have any time code associated with it. During editing, applications can display many types of time code, such as the time code of a physical source you are editing or the time code for the point at which an editor is inserting a new segment.
time warp effect	A class of effects, where the duration of the input media segments does not equal the duration of the effect. See also, *motion effect*, *capture mask effect*, and *repeat track effect*.
track	A externally referencable channel in a composition, such as a video track or audio track.
Transition	A Transition represents what is to take place as one segment ends and the next one begins. The simplest transition is a cut, which, in video, is when the first frame of the starting segment directly follows the last frame of the segment that is ending.
UID	See *mob ID*.

value	The actual data associated with a particular property in an OMF Interchange object.
videotape	Oxide-coated, plastic-based magnetic tape used for recording video and audio signals.
VTR	Videotape recorder.
WAVE	RIFF Waveform Audio File Format. A widely used format for audio data using little endian byte order. OMF Interchange includes it as a common interchange format for audio data.
wipe	A transition in which a margin or border moves across the screen, wiping out the image of one segment and replacing it with the image of the next one.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

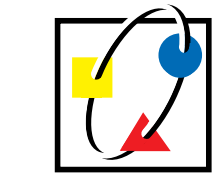
V

W

Y

OMF™ Participation Form

To get involved and become an OMF Interchange Sponsor, Partner, or Champion, fill in this form and return it to the OMF Developers' Desk today.



Sponsor: Digital media users who share the vision of digital media interchange and are actively encouraging vendors to integrate OMF Interchange.

Partner: Manufacturers actively contributing to the creation and evolution of OMF Interchange.

Champion: Manufacturers supporting OMF Interchange by offering OMF Interchange compatibility in their products. You may request a Champion application by returning this form.

- As a *Sponsor* of the Open Media Framework we understand the importance of a universal digital media interchange format. We support the OMF Partners and Champions in their effort and agree to help the OMF effort by encouraging manufacturers to put OMF Interchange compatibility in their products.
- As a *Partner* we agree to participate in the process of developing and revising the OMF Interchange specification by providing comments and suggestions. We agree to work with Avid engineers to resolve technical issues and ideas. We recognize that our comments will be factored into the publicly available specification.
- As a *Champion* of OMF Interchange, we will act as a partner to help support and evolve OMF Interchange. We will include OMF Interchange compatibility in our products and applications. If we are accepted as a Champion, we agree to be listed by Avid as a supplier of OMF compatible products.

As a Sponsor, Partner, or Champion, we understand that it is Avid's intent to publish the OMF Interchange format descriptions and make them freely available to anyone interested in adopting these proposed standards. We also understand that Avid will retain copyrights to the *OMF Interchange Specification* in order to protect the integrity of OMF and Avid's intellectual property rights.

We understand that Avid will make public the names of all organizations participating in the process of establishing OMF standards. We further understand that as a Sponsor or Partner, this agreement does not obligate our organization to support OMF Interchange standards in our products. We will have the option to adopt the proposed OMF standards and support them in our products. We may then apply to become an OMF Champion.

We wish to be an OMF: Sponsor ☐ Partner ☐ Champion Applicant ☐

We support OMF Interchange as a: Vendor ☐ User ☐

Name: Title:

Company: World Wide Web URL:

Address:

Address: Country:

Phone: Fax: Email Address:

Description of products/services:

Signature Date:

Please mail or fax this form to: Avid Technology, Inc. OMF Developers' Desk
1925 Andover St.
Tewksbury, MA 01876
Phone: (800) 949-OMFI International (978) 640-3400 Fax: (978) 640-0065

Please attach a sheet identifying technical and marketing contacts for your organization.