AAF ASSOCIATION SPECIFICATION

Advanced Authoring Format (AAF) Object Specification v1.0.1

Table of Contents

1 Scope

This document defines the data structures used by the Advanced Authoring Format (AAF) for the interchange of audio-visual material and associated metadata. The data structures are defined using a class model, in terms that closely correspond to the authoring domain. The mapping of these data structures into a file (or other persistent storage) is defined by other AAF Association specifications.

2 Normative References

The following normative documents contain provisions that, through reference in this text, constitute provisions of this Document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this document are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative documents referred to applies.

SMPTE 298M – 1997, Television – Universal Labels for Unique Identification of Digital Data

SMPTE RP210 – Metadata Dictionary Contents

ISO/IEC 11578-1 – 1998, Information technology – Open Systems Interconnection – Remote Procedure Call (RPC) Annex A, Universally Unique Identifier

SMPTE RP224, SMPTE Labels Registry

IETF RFC 1738 – Uniform Resource Locators (URL)

IETF RFC 2396 – Uniform Resource Identifiers (URI)

3 Definition of Acronyms, Terms and Notation

3.1 Acronyms and Terms

Abstract class	A class that is not sufficient to define an object; an object must also belong to a concrete sub-class of the abstract class
AUID	A 16-byte unique identifier whose value is a SMPTE 298M Universal Label or a UUID or GUID.
CDCI	Color Difference Component Image
Class	Category of objects, which have common properties, relationships and semantics
Class hierarchy	Specification of the sub-class and super-class relationship among a set of classes
Concrete class	A class that is sufficient to define an object and may be instantiated as an object
Edit Rate	A rational number that specifies the units used to specify the duration of Components in a track; the edit rate is the number of units that equal one elapsed second. In a track which describes essence, the Edit Rate is usually chosen to be the number of Editable Units per second
Edit Unit	A period of time equal to 1/(Edit Rate)
Editable Unit	The smallest portion of essence which can be edited such as a field or frame.
Essence	The video, audio and data streams to be contained and described by AAF.
Inheritance	The mechanism that defines a relationship between classes where a sub-class inherits the properties, relationships, and semantics of its super-class
Interleaved-essence	An essence format that combines two or more channels of audio or video data into a single essence stream
Metadata	The description of essence or information on how to use the essence
Mob	An object that specifies the metadata for a piece of material and has a globally unique identity
MobID	The value of the unique identification of a Mob
Object	An instance of a class
Package	An alternative term for Mob used in MXF
RGBA	Red Green Blue Alpha
Sample Rate	For audio: The sample rate of the essence. For video: The field or frame rate of the essence (not the pixel clock rate)
Strong reference	A relationship between objects where one object is the owner of another object. An object can be owned only by a single object at one time. A strong reference defines a logical containment, where the owning object logically contains the owned object.
Sub-class	A class that is defined as having the same properties, relationships and semantics as another class, which is called its super-class, and may have additional properties, relationships, and semantics that are not present in the super-class
Super-class	A class that has another class as its sub-class
UML	Unified Modeling Language
Unicode	A form of character coding that allows a wide range of characters and ideograms to represent most major languages

Weak reference	A relationship between objects where one object has a reference to a second object; the second object is uniquely identified. In contrast with a strong reference, a weak reference specifies an association relationship but does not specify ownership. An object can be the target of weak references from more than one object.

3.2 Notation

The AAF Object Specification uses UML class diagrams to depict the class model. The UML class diagrams show the class name, properties, property types, object references and inheritance relationships. Figure 1 below provides a key to the UML class diagrams used in the AAF Object Specification.

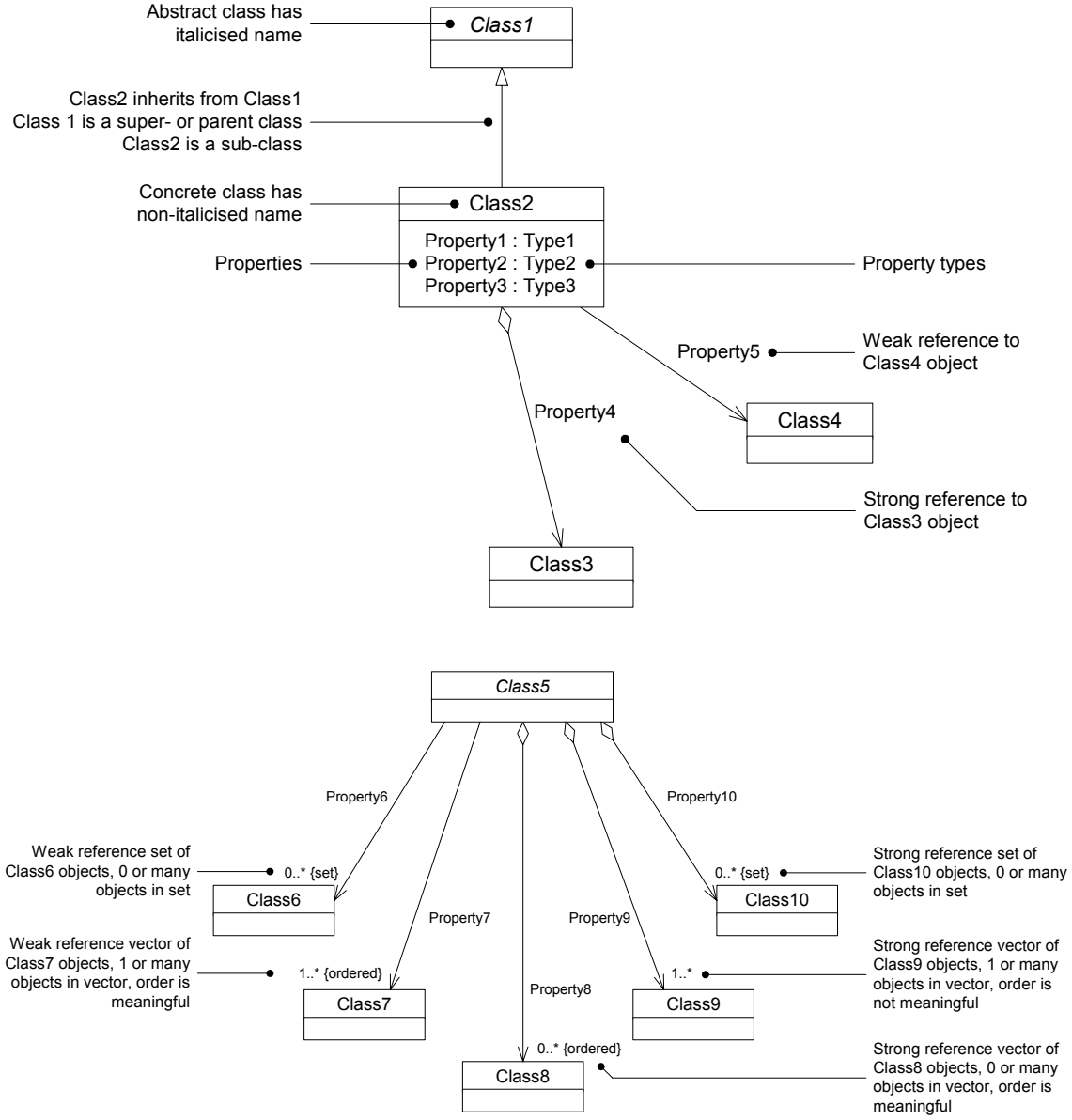


Figure 1 Key to UML class diagrams used in the AAF Object Specification

4 Introduction

4.1 Object Oriented Interchange

The Advanced Authoring Format provides an object-oriented mechanism to interchange multimedia information.

Object-oriented interchange has the following advantages:

- Objects provide a framework for containing and labelling different kinds of information
- Objects make it possible to treat different items in the same way for attributes they share.
- When the information becomes very complex, objects provide a mechanism to describe it in a structured way.

4.2 Header Object

An interchange file contains:

- A MetaDictionary defining the classes used in the AAF file
- One Header object and its related objects
- Mobs and the objects they have
- Essence data

The Header object and its related objects are in an interchange file so that Mobs and Essence data, which contain the useful information, may be accessed.

Each object in an interchange file belongs to a class. The class defines how the object may be used and the kind of information it stores. An object consists of a set of properties. Each property has a name, a type, and a value. An object's class defines the properties that it may have.

This specification defines classes using a class hierarchy, in which a subclass inherits the properties of its superclass. All classes are subclasses of the InterchangeObject class, except for the classes contained in the MetaDictionary which are subclasses of MetaDefinition.

There is exactly one Header object in an interchange file. The Header object owns all other objects in the file. This ownership relationship is specified by the StrongReference, StrongReferenceVector, and StrongReferenceSet property types.

The objects contained in an AAF file are depicted in Figure 2 below:

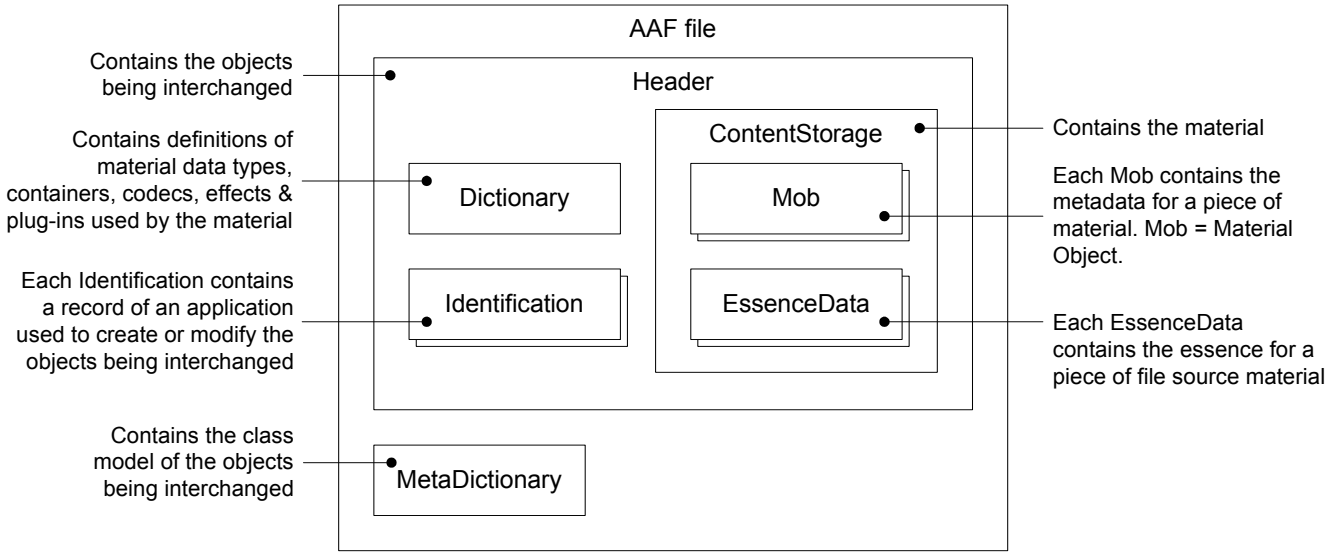


Figure 2 Objects contained in an AAF file

4.3 Mobs

A Mob is an object that has a universal identifier and consists of metadata. Mobs describe how essence data is formatted or how separate pieces of essence data are combined or composed. Mobs are very general and can be used to describe many different kinds of essence data: video, sound, still picture, text, animation, graphics, and other formats.

Mobs have names and descriptions, but are primarily identified by a unique identifier, which is called a MobID.

A Mob can describe more than one kind of essence. For example, a Mob can have audio, video, still image, and timecode data. A Mob has one or more Slots. Each Slot can describe only one kind of essence data. For example, a Mob can have two Slots with audio, one Slot with video, three Slots with still images, and two Slots with timecode.

Kind of Mob	Function
CompositionMob	Describes creative decisions on how to combine or modify essence: Decisions on order of essence data Decisions on placement of essence data Decisions on effects that modify or combine essence data
MasterMob	Collect and possibly synchronize related essence data; provides indirect access to essence data, which is independent of storage details
File SourceMob	Provides direct access to and describes format of digital essence data that is (or can be) stored in a computer file
Physical SourceMob	Describes physical media such as a videotape or film

CompositionMobs describe the creative editing and composing decisions that combine individual bits of essence data into a presentation. A Composition Mob can describe creative decisions like the following:

- The audio track contains "Also Sprach Zarathustra" when the video track showed the monolith in the Stanley Kubrick film 2001: A Space Odyssey
- The pacing of the cuts between shots in Alfred Hitchcock's thrillers

- How different still images are composed into a single image
- How a special effect distorts a video image to make it appear as if it were a reflection on a pool of water

MasterMobs provide an association between CompositionMobs, which describe the creative decisions, and SourceMobs, which describe and identify the essence data. MasterMobs insulate the CompositionMob from the detailed information about how the essence data is stored. MasterMobs can describe:

- How video and audio digital essence data are synchronized
- How multiple objects containing digital essence data represent different digital versions of the same original essence data - the versions may be different in the amount of compression or in the kind of format used to store it

File SourceMobs describe the format of the digital essence data and provide a mechanism to access the digital essence data. File SourceMobs have information such as:

- The format used to store the digital essence data, such as WAVE and AIFC for audio and CDCI and RGBA for video
- The number of samples or frames for digital audio and video data
- The kind of compression used
- The number of pixels and the aspect ratio for picture data

Physical SourceMobs have descriptive information that makes it possible to identify the actual videotape or film. They can also have timecode or edgecode information used to find the section of tape or film that corresponds to a frame in the associated digital essence data.

4.3.1 Immutability of Mobs

A MobID is globally unique. Two Mobs in an AAF file shall not have the same MobID. A Mob in one AAF file may have the same MobID as a Mob in another AAF file under either of the following conditions:

One Mob is a duplicate of the other

One Mob is a modified version of the other subject to the following restrictions on modifications. The type of Mob determines the kind of modifications that can be made to it and still retain its identity:

- The information in a file SourceMob describing the format of essence is immutable and cannot be modified. Modifications to a file SourceMob are limited to modifications to descriptive metadata and to the following limited modifications when the essence is being created or derived:
- When creating essence or deriving one form of essence from another form, it may be necessary to create or derive the essence in steps. In this case, the SourceMob can be modified to describe the additional essence that is incorporated in a step. This should be done in a manner that does not invalidate a reference to a section of the essence that was previously created.
- A MasterMob may be modified by replacing a reference to a SourceMob with a reference to another SourceMob or it may be modified by inserting a reference to an alternate SourceMob. The modifications are subject to the restriction that the replacement or alternate SourceMobs should be representations of the same physical media as the originally referenced SourceMob.
- A CompositionMob may be modified in any way.

4.4 MobSlots

A Mob shall have one or more MobSlots. Each MobSlot describes an element of essence that can be referenced. A MobSlot shall specify an integer identifier, which is called a SlotID.

Each kind of MobSlot defines a specific relationship between essence data and time. This specification currently defines the following kinds of MobSlots:

- StaticMobSlot
- TimelineMobSlot
- EventMobSlot

A StaticMobSlot describes essence that does not vary over time. A StaticMobSlot may describe a static image or some other static essence such as text.

A TimelineMobSlot describes essence that varies with a fixed, predictable interval or continuously over time. For example, digital audio, video, and film have a fixed, predictable sample or frame rate, and analog audio varies continuously over time.

An EventMobSlot describes essence that has an unpredictable relationship with respect to time. GPI (General Purpose Interface) events and CommentMarkers are examples of irregularly timed events.

Kind of Slot	Function
StaticMobSlot	Describes essence data that has no specific relationship to time, such as static images or static text.
TimelineMobSlot	Describes essence data that has a fixed or continuous relationship with time, such as audio, film, video, timecode, and edgecode
EventMobSlot	Describes essence data that has an irregular relationship with respect to time, such as GPI events and Comment Markers associated with specific times

4.5 Components

Components are essence elements. A component in a TimelineMobSlot has a duration expressed in edit units. The relation between edit units and clock time is determined by the edit rate of the TimelineMobSlot that has the component. A component provides a value for each edit unit of its duration.

The kind of value a component provides is determined by the component's data kind. A component can have a data kind that corresponds to a basic kind of essence, such as sound or picture or a kind of metadata such as timecode.

The Component class has two subclasses: Segment and Transition.

The Segment class subclasses include the following:

- SourceClip which references a section of a Slot in another Mob; for example a SourceClip in a TimelineMobSlot can describe video data
- Sequence which specifies that its set components are arranged in a sequential order; in a TimelineMobSlot, the components are arranged in sequential time order
- OperationGroup which specifies that either two or more Segments should be combined using a specified effect or that one Segment should be modified using a specified effect
- Filler which defines an unspecified value for its duration

A Transition causes the preceding and following Segments to be overlapped in time and to be combined by the Transition's effect. A Transition object shall be a member of a Sequence's set of Components and it shall be preceded by a Segment and followed by a Segment.

4.6 References between Mobs

A Mob can reference another Mob to indicate the source or derivation of the essence. A Mob refers to another Mob by having a SourceClip object. A SourceClip object has a weak reference to a Mob using its identifying MobID value; shall identify a MobSlot within the referenced Mob with a SlotID value; and when referencing a TimelineMobSlot shall specify an offset in time within the referenced TimelineMobSlot.

SourceClips in CompositionMobs specify the MobID of the MasterMob, and are used to represent pieces of digital essence data. The MasterMob provides a level of indirection between the digital essence data and the objects that refer to them.

SourceClips in File SourceMobs specify the MobID of a Physical SourceMob. For example, a video File SourceMob has a SourceClip that specifies the Physical SourceMob describing a videotape used to generate the digital video data.

SourceClips in Physical SourceMobs identify the MobID of a previous physical source of physical media. For example, a videotape SourceMob has a SourceClip that specifies the Physical SourceMob describing the film that was used to generate the videotape.

If there is no previous generation of essence, then the SourceMob shall contain a SourceClip that specifies a SourceID value of 0, a SourceSlotID value of 0, and, in TimelineSlots, a StartTime value of 0.

In summary, Mobs describe not only essence data, but through their relationships between one another, they describe how one form of essence data was derived from another.

Figure 3 below illustrates how a SourceClip in a CompositionMob references a MasterMob. The MasterMob references the File SourceMob, which references the tape SourceMob. Finally, the tape SourceMob references the film SourceMob.

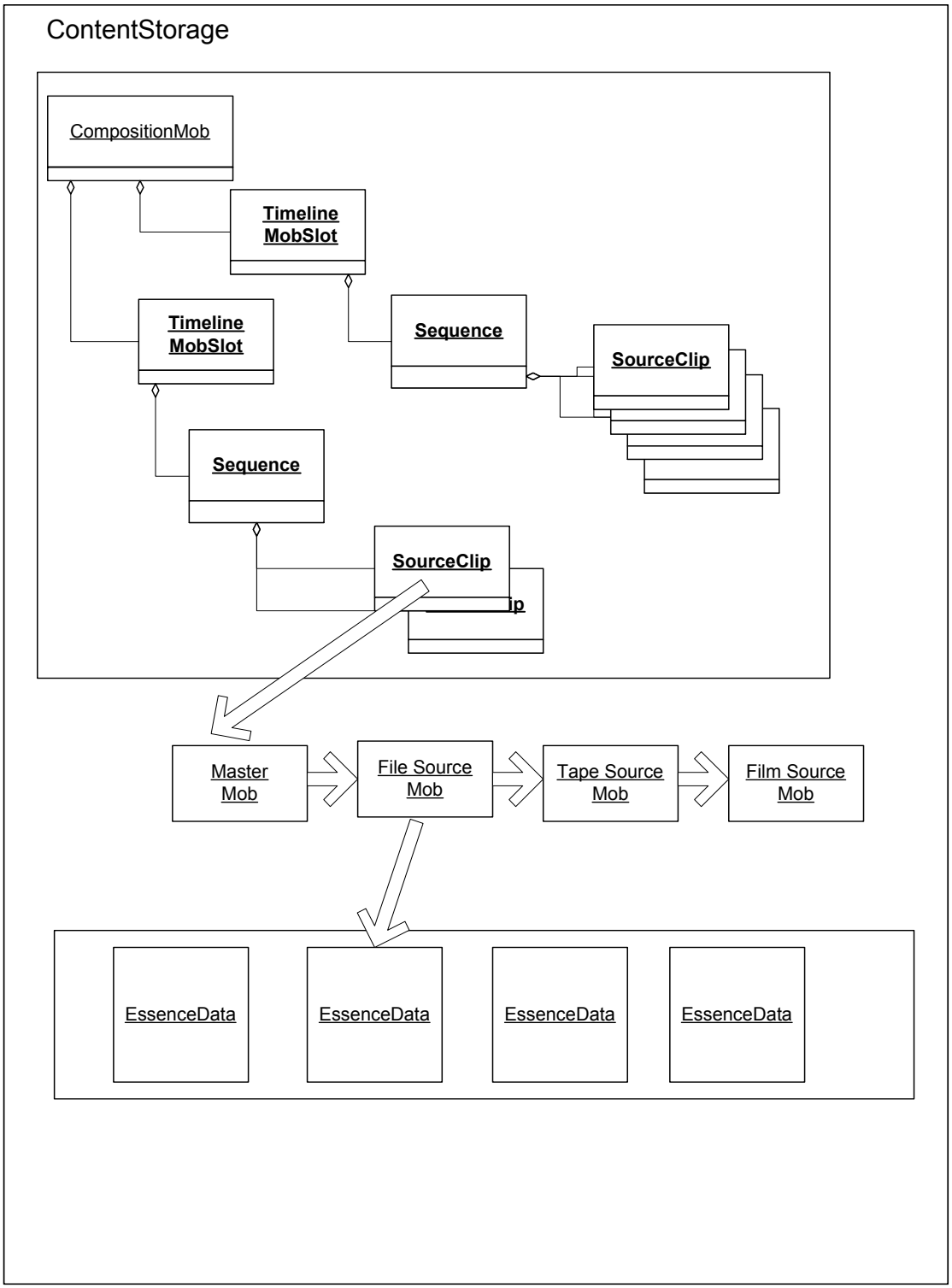


Figure 3 References between Mobs

4.7 File SourceMobs and EssenceData objects

A File SourceMob describes essence and is used to access it, but does not own it. This document separates the description and the storage of the essence for the following reasons:

- Audio and video data and other essence 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 File SourceMob separate from the essence provides more flexible storage options while still allowing the composition to use the same access mechanism.
- Audio and video data or other essence may be used in more than one CompositionMob and these CompositionMobs can be in different files. Separating the File SourceMob from the essence means that only the information in the File SourceMob needs to be duplicated.

The essence described by a File SourceMob can be stored in three ways:

- In an EssenceData object in the same file as the SourceMob
- In an EssenceData object in a different file, which must contain a duplicate of the SourceMob
- In a data file that is not a wrapper file

The MobID connects the File SourceMob to the essence if it is stored in an EssenceData object. The File SourceMob and its corresponding EssenceData object have the same MobID value. Since the ContentStorage object has all the Mobs and EssenceData objects in the file, applications can find the EssenceData object associated with a File SourceMob by searching for the appropriate MobID.

If the essence is stored in a data file that is not a wrapper file, then the data file is identified by locators in the essence descriptor. In some cases, the data file format has a mechanism to store a MobID. In these cases, applications can still use the MobID to associate the File SourceMob with the digital essence data. When there is no mechanism to store a MobID with the digital essence data, interchange files use specialized locator objects to associate a File SourceMob with the digital essence data. If there is no MobID stored with this essence, it is difficult to identify a data file if the file has been moved or renamed.

4.8 Static Image Essence in Mobs

Static image essence is described by a StaticMobSlot. Static image essence has no relation to time; consequently, StaticMobSlots do not have an edit rate and the objects that they have do not specify a duration.

In a StaticMobSlot, a SourceClip refers to another StaticMobSlot by specifying its MobID and SlotID but does not specify an offset in time or a duration as in a TimelineMobSlot.

Composition Mobs that have only StaticMobSlots specify the editing decisions involved in composing static images. Figure 4 below illustrates a typical Composition Mob that describes a static image and the static components used to compose it. The static images are combined by using static effects to transform the individual images and to combine them into a single image.

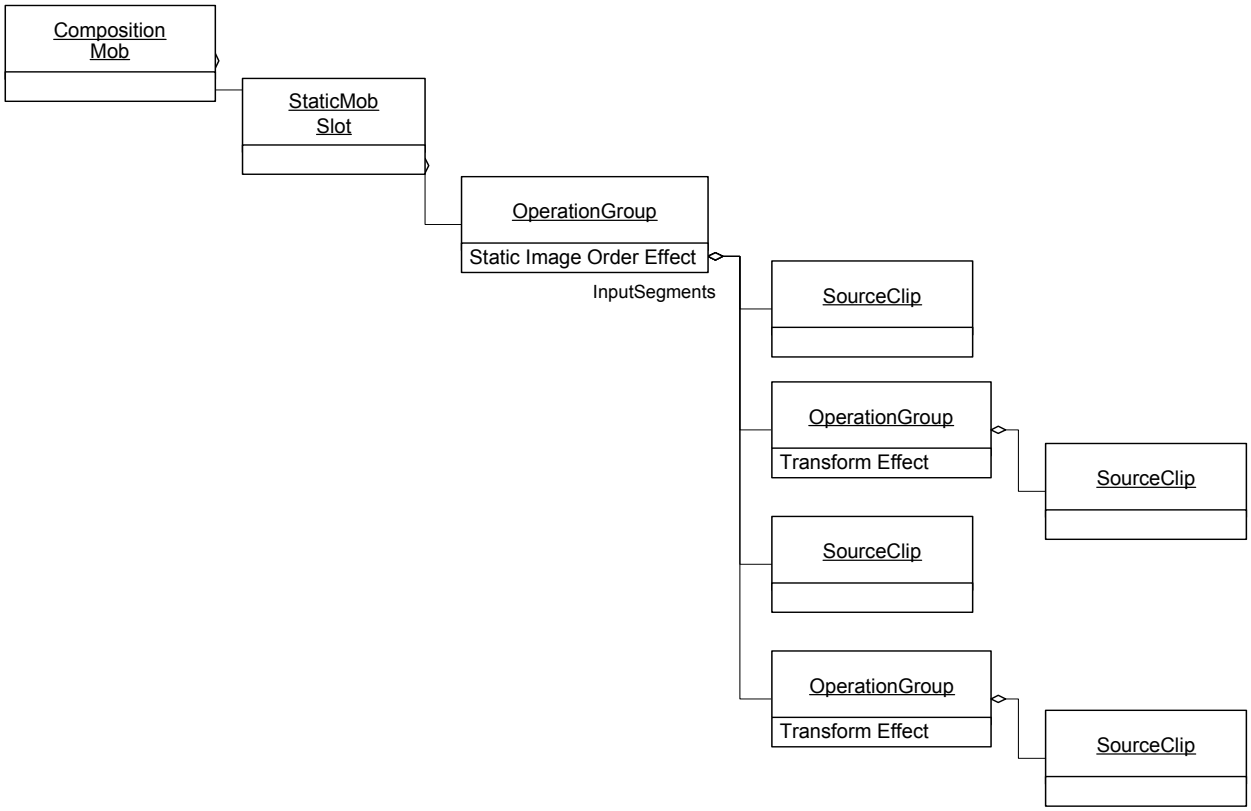


Figure 4 A Static MobSlot in a CompositionMob

4.9 Time-varying Video and Audio Essence in Mobs

Audio and video essence data is represented in TimelineMobSlots. These are Slots that represent time-varying data where this data has a fixed relationship with respect to time. For example, NTSC video has a frame rate of approximately 29.97 frames per second. Each TimelineMobSlot specifies an edit rate which defines the unit of time for objects referred to by that particular TimelineMobSlot. Edit rates are specified as a rational (a real number expressed as two integers: a denominator and a numerator). For example, NTSC video's edit rate is typically specified by an edit rate of 30000/1001.

In TimelineMobSlots, a SourceClip references a TimelineMobSlot in another Mob by specifying its MobID and TimelineMobSlot ID number and by specifying a subsection of the TimelineMobSlot with an offset in time and a duration. For example, a SourceClip in a composition Mob can reference a subsection of audio or video data by referencing a section of that essence data's MasterMob.

A simple Composition Mob has audio and video TimelineMobSlots where each TimelineMobSlot has a sequence of SourceClips. The sequence specifies that the SourceClips should be played consecutively, one after another. Each TimelineMobSlot in the Composition Mob is to be played simultaneously with other co-timed TimelineMobSlots.

The structure of a CompositionMob with TimelineMobSlots is shown in Figure 5 below.

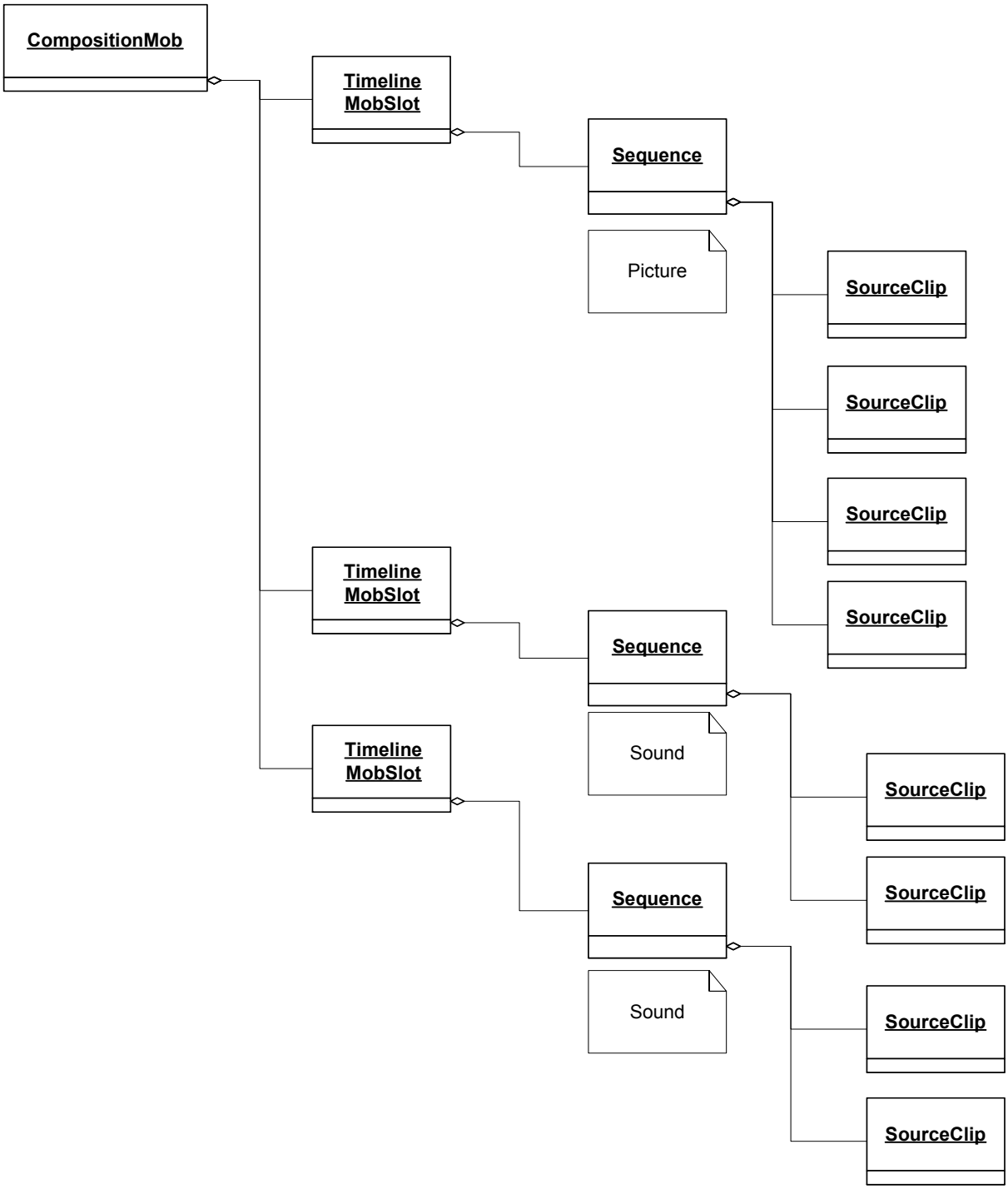


Figure 5 TimelineMobSlots in a CompositionMob

Each SourceClip in a sequence identifies the audio or video data to be played and specifies its duration, but does not specify the time at which it should be played in the composition. The starting time of a section in a sequence depends on the number and duration of the sections that precede it. A SourceClip can be thought of as a section of videotape or film to be spliced with other sections. By examining the section itself, one may listen to its audio or view its frames, but one cannot tell where it will appear in the finished piece until the preceding sections in the sequence are examined.

Figure 6 below illustrates how the SourceClips in a sequence appear in a timeline view of a composition.

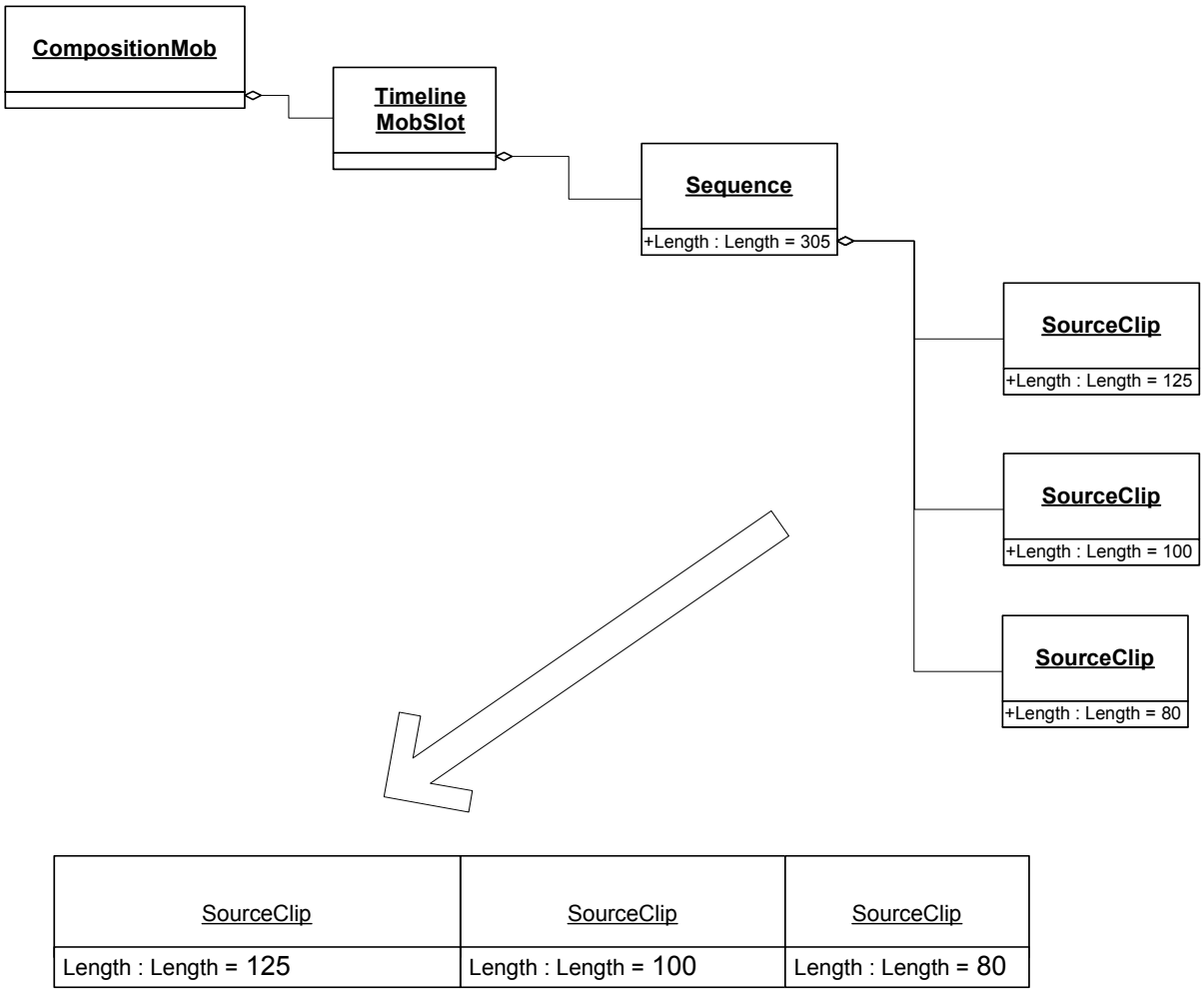


Figure 6 Timeline View of a Sequence of SourceClips

4.10 Event Data in Mobs

Events typically specify an action or define a behavior that takes place at a specified time. Typically, EventMobSlots specify events that are associated with the time-varying essence in a parallel TimelineMobSlot. Each EventMobSlot describes one kind of event. Figure 7 below illustrates a Composition Mob that has a TimelineMobSlot with video essence data, and an EventMobSlot that has comments defined for specific points in time.

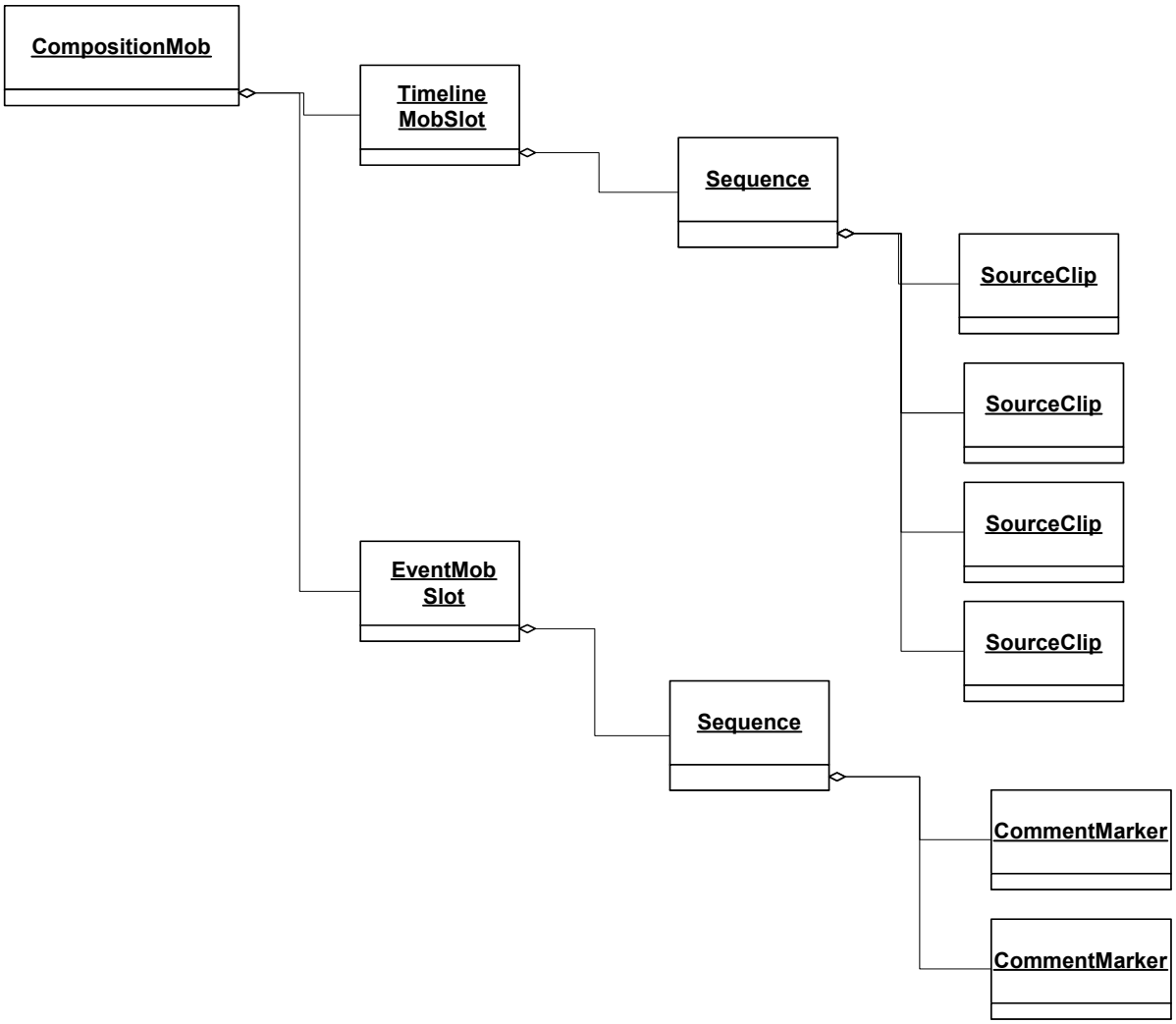


Figure 7 EventMobSlots in a CompositionMob

5 AAF Class Hierarchy

This chapter defines the AAF class hierarchy, which is used to describe multimedia compositions and data. A class specifies an AAF object by defining what kind of information it may contain and how it is to be used. Each AAF class inherits from its superclass. The AAF class hierarchy does not define any classes that inherit from more than one immediate superclass thereby avoiding the problems associated with multiple inheritance.

An AAF object consists of a set of properties. A property consists of a property name, a property type, and a property value.

Each class defines an object that has a set of properties. An object shall contain all the required properties of all classes from which it inherits. There are two root classes in the AAF class hierarchy: the InterchangeObject class and the MetaDefinition class.

The InterchangeObject class is the root for most of the classes in AAF including those for Mobs and Essence Data. The InterchangeObject class defines one required property, the ObjClass property. An AAF object specifies its class by the value of the ObjClass property. The InterchangeObject class and its subclasses defined by this specification may be extended by defining additional optional properties for existing classes or by defining new subclasses.

The MetaDefinition class is the superclass of the ClassDefinition, PropertyDefinition, and TypeDefinition classes. Since these classes provide the mechanism for describing and extending AAF classes, it is not possible to add optional properties or define new subclasses to the MetaDefinition classes described in this document.

An AAF file shall contain the class model being used by the file in the MetaDictionary object's ClassDefinitions and TypeDefinitions properties.

This specification describes classes, property names, and property types by name, but classes, property names, and property types are uniquely defined in an AAF file by an AUID.

These AUIDs are listed in the AAF reference implementation in the following files:

- AAFClassDefUIDs.h
- AAFPropertyDefs.h
- AAFTypeDefUIDs.h

AAF objects are stored in an AAF file using a structured container format. The AAF reference implementation uses Microsoft's Structured Storage as its container format, and implements an object management layer for extended property set management.

5.1 Object model goals

Applications that process essence and metadata exist on a multitude of platforms, each with different characteristics for storage capacity, throughput, multimedia hardware, and overall system architecture. This document defines a format for the interchange of essence and metadata across applications and across platforms.

This document provides a mechanism to encapsulate essence and metadata. It defines objects to store and describe the essence that allow an application to determine the format of the essence and to determine what conversions, if any, it needs to apply to the essence to process the essence.

This document provides a mechanism to synchronize essence and to describe the format of essence that contains interleaved streams. This mechanism allows an application to synchronize separate streams of essence that were originally derived from original media sources, such as film, audio tape, and videotape, that were created in synchronization.

This document provides a mechanism to describe the derivation of essence from the original media sources. This mechanism allows applications to reference tape timecode and film edgecode that correspond to the essence and allows applications to regenerate essence from the original media sources.

This document provides a mechanism to describe compositions. Compositions contain information about how sections of essence should be combined in sequence, how to synchronize parallel tracks of sequences, and how to alter sections of essence or combine sections of essence by performing effects.

This document provides a mechanism to define new classes or to add optional information to existing classes. This mechanism allows applications to store additional information in an interchange file without restricting the interchange of the information specified by this document.

5.2 Classes and semantic rules

This document defines classes that specify the kinds of objects that can be included in a storage wrapper file and it defines the semantic rules for including objects in a storage wrapper file.

An object consists of a set of properties. Each property has a property name, a property type, and a property value. Each object belongs to a class that specifies the properties that it is required to have and optional properties that it may have.

This document defines classes by defining a class hierarchy and by defining the properties for each class in the hierarchy. This document also defines a mechanism for extending the class hierarchy by defining new classes that are subclasses of classes defined in this document.

An object shall have the required properties specified for all classes that it is a member of. An object may have the optional properties specified for all classes that it is a member of. This specification lists the classes in the class hierarchy and specifies the properties that are required and the properties that are optional for each class. This specification also lists semantic rules, restrictions, and requirements on objects based on the object's class and the context in which the object is used.

The class of an object is specified by the ObjClass property of the InterchangeObject class.

5.3 Class Hierarchy

Figure 8 through Figure 12 illustrates the InterchangeObject class hierarchy and the class packages incorporated in it. The classes in each figure are specified in succeeding chapters, with each chapter corresponding to one figure.

Note that additional subclasses of any of the classes shown in these diagrams may be defined, as extension classes or as new built-in classes in future versions of this specification.

Figure 13 illustrates the MetaDefinition class package.

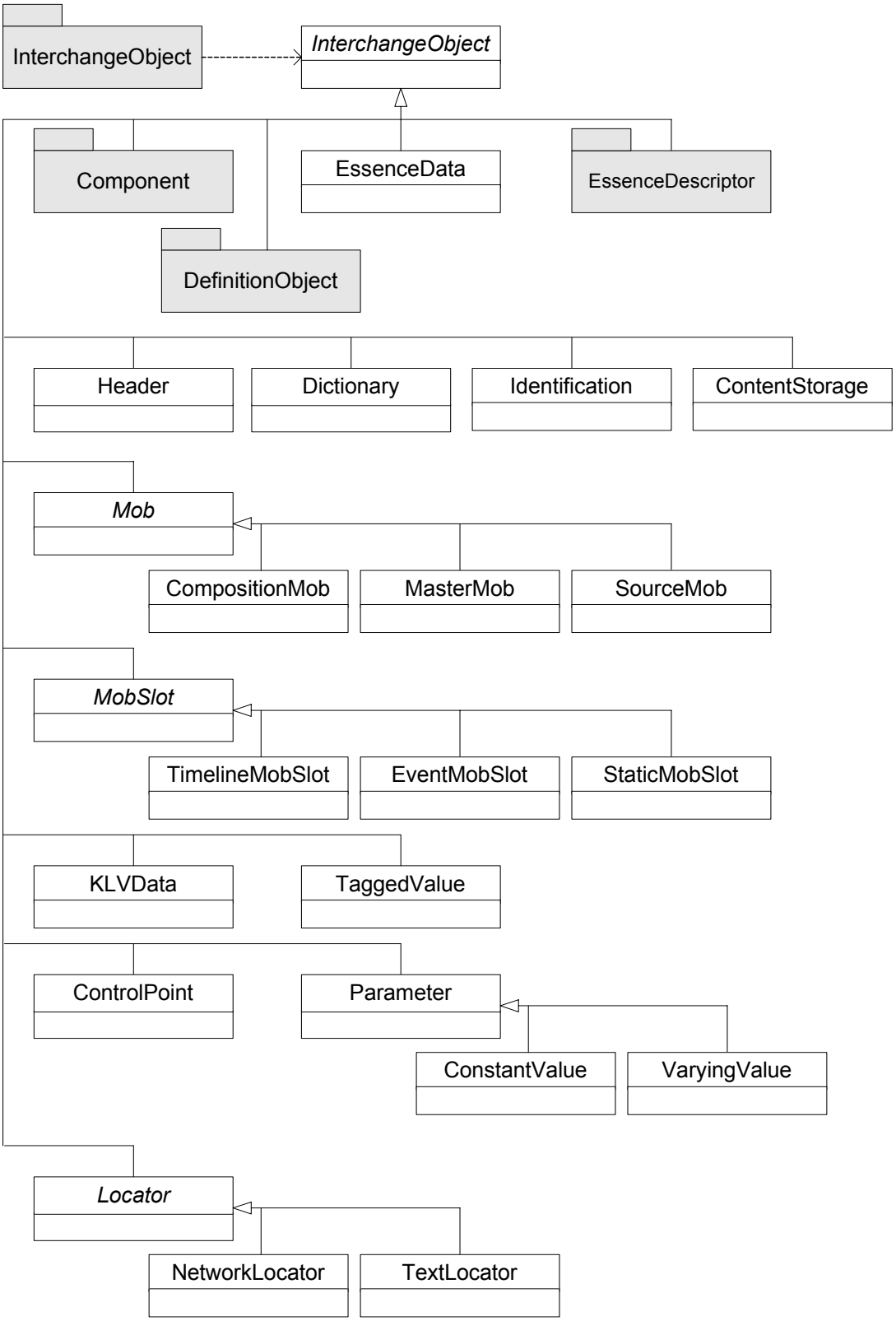


Figure 8 InterchangeObject Package

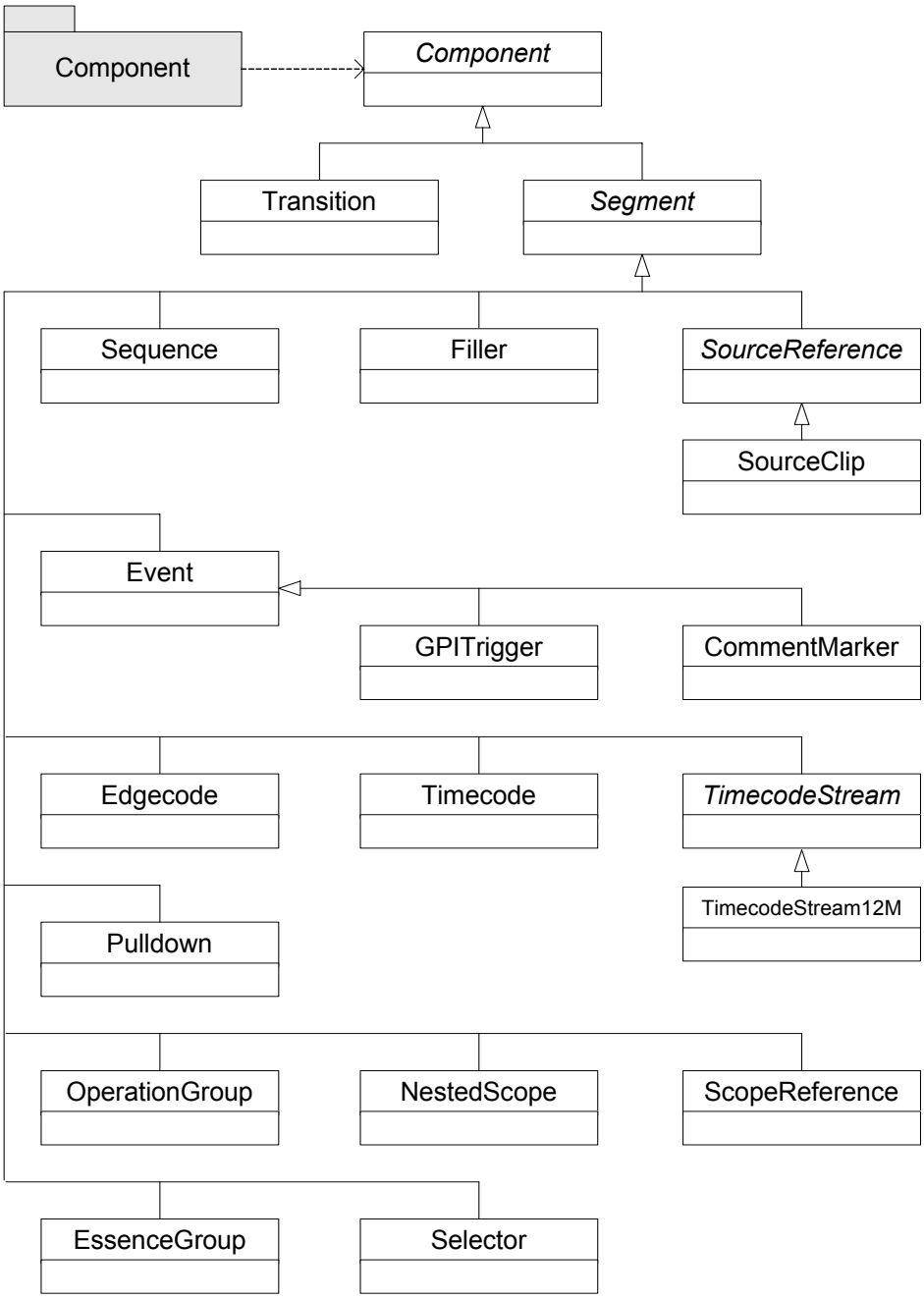


Figure 9 Component Package

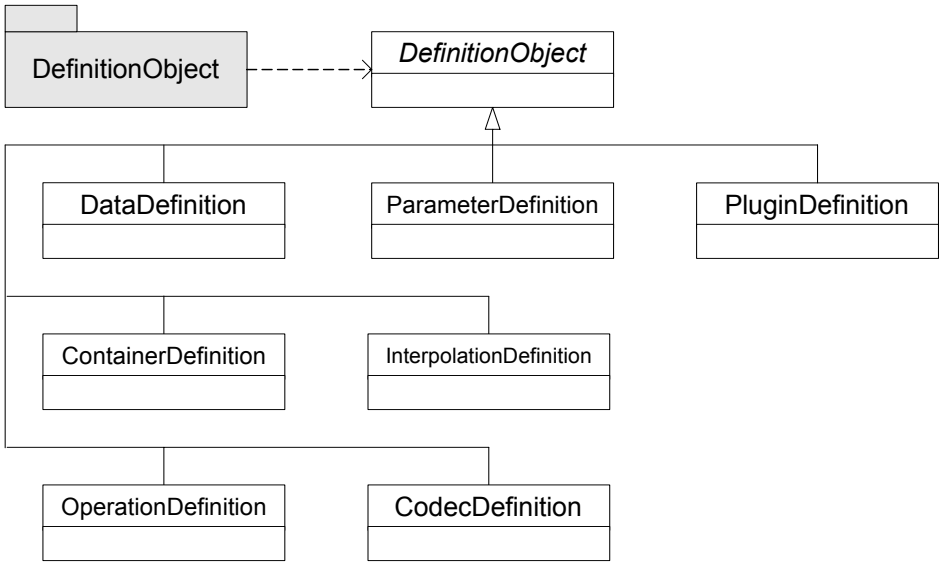


Figure 10 DefinitionObject Package

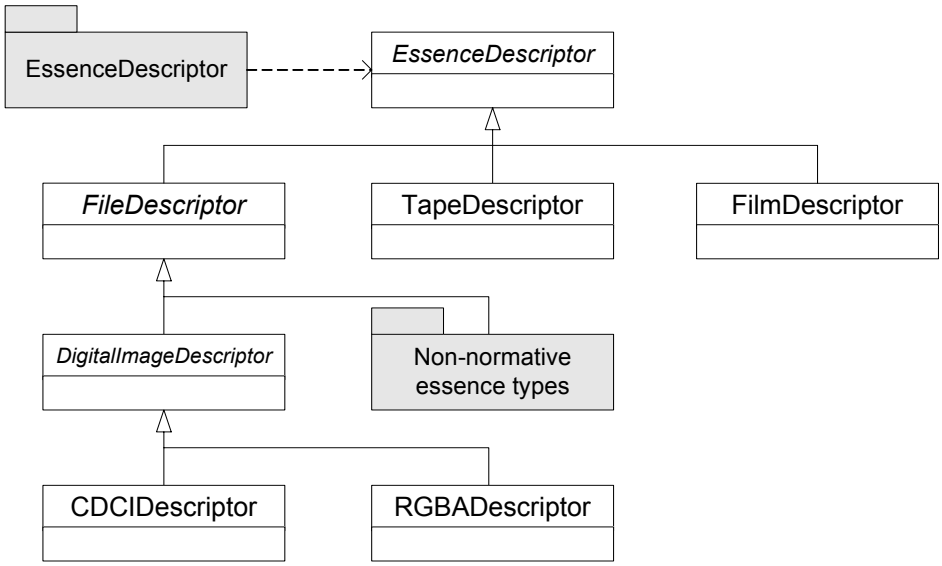


Figure 11 EssenceDescriptor Package

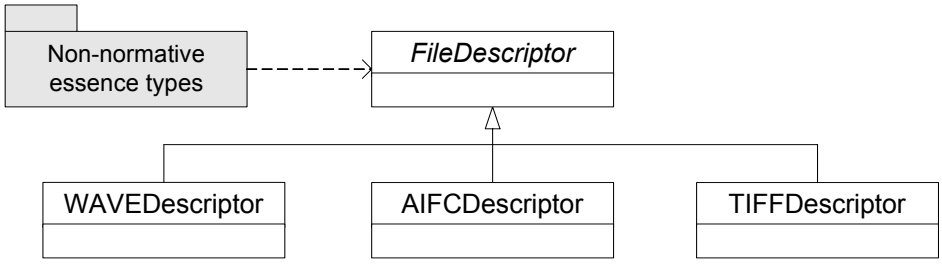


Figure 12 Non-normative essence types Package

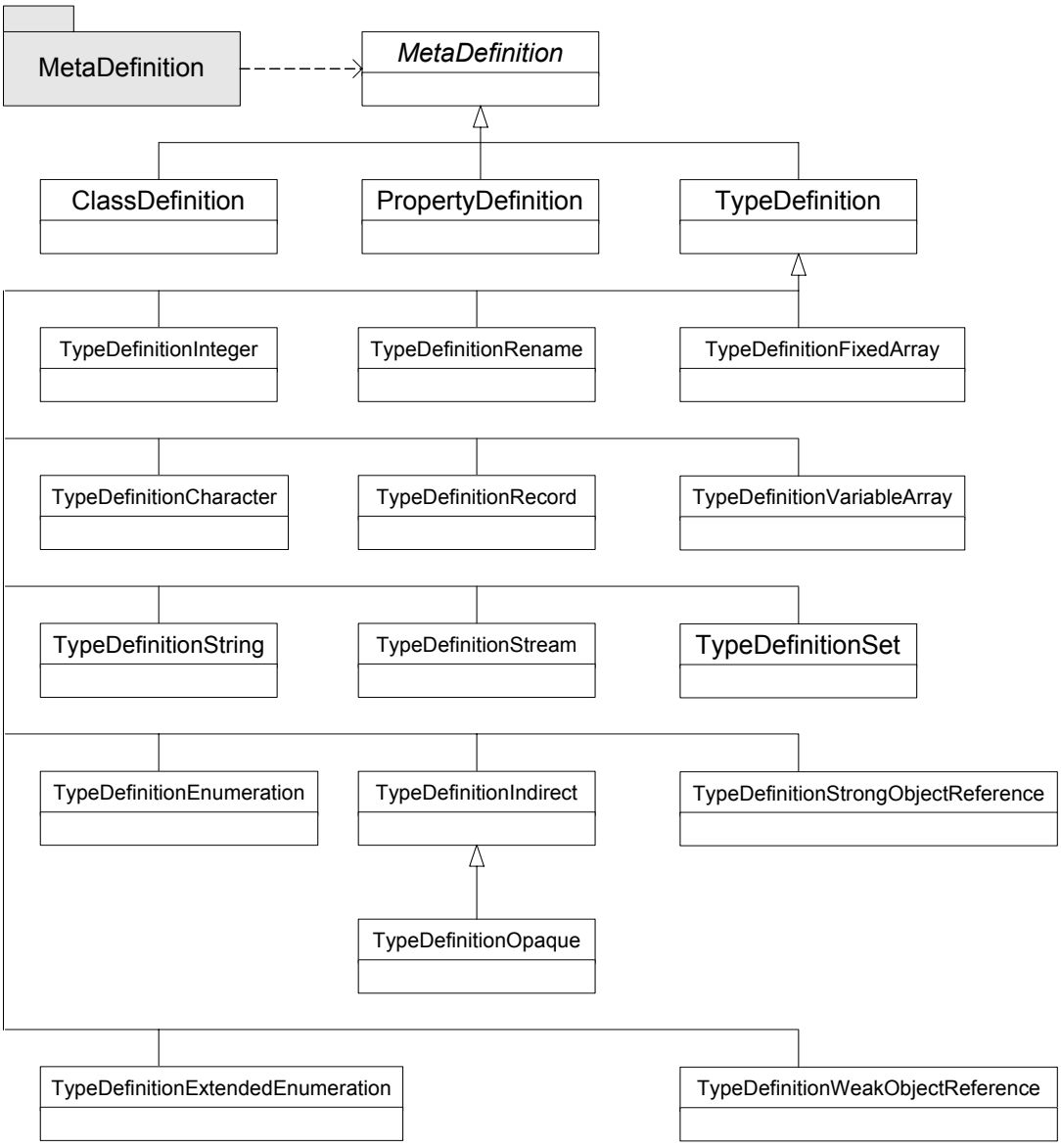


Figure 13 MetaDefinition Package

In the Figures above, *italic* typeface identifies the classes in the class hierarchy that are abstract classes. An object that belongs to an abstract class shall also belong to a subclass of the abstract class.

An object can be used in any context where an object of its class or one of its superclasses is allowed subject to any restrictions listed in the class specification.

6 InterchangeObject Classes

This chapter includes the class specifications for classes in the InterchangeObject package. Figure 14 shows the class hierarchy for the InterchangeObject package.

The class specification pages are presented in order according to the class hierarchy, reading the Figure left to right, depth-first.

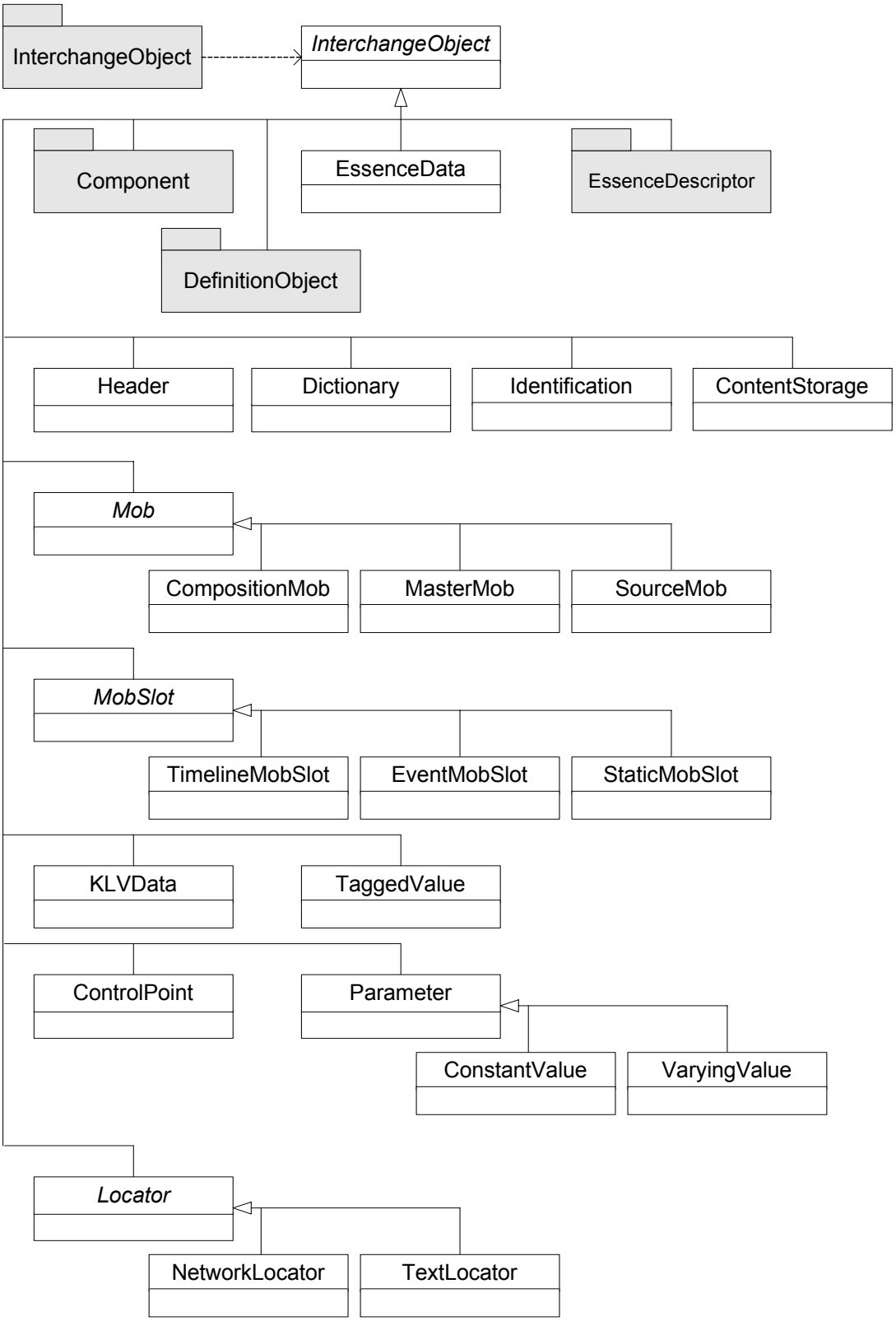
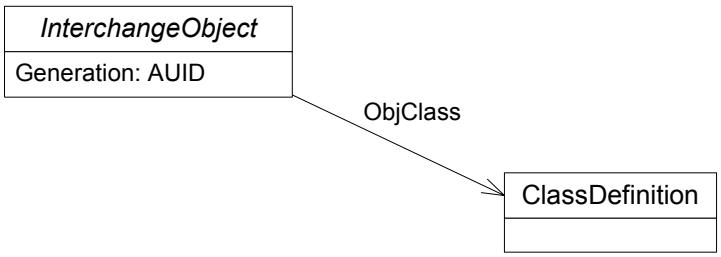


Figure 14 InterchangeObject Package

6.1 InterchangeObject class

The InterchangeObject class is a root class. All classes defined in an AAF file shall be sub-classes of InterchangeObject with the exception of the MetaDefinition classes defined by this document.

The InterchangeObject class is an abstract class.

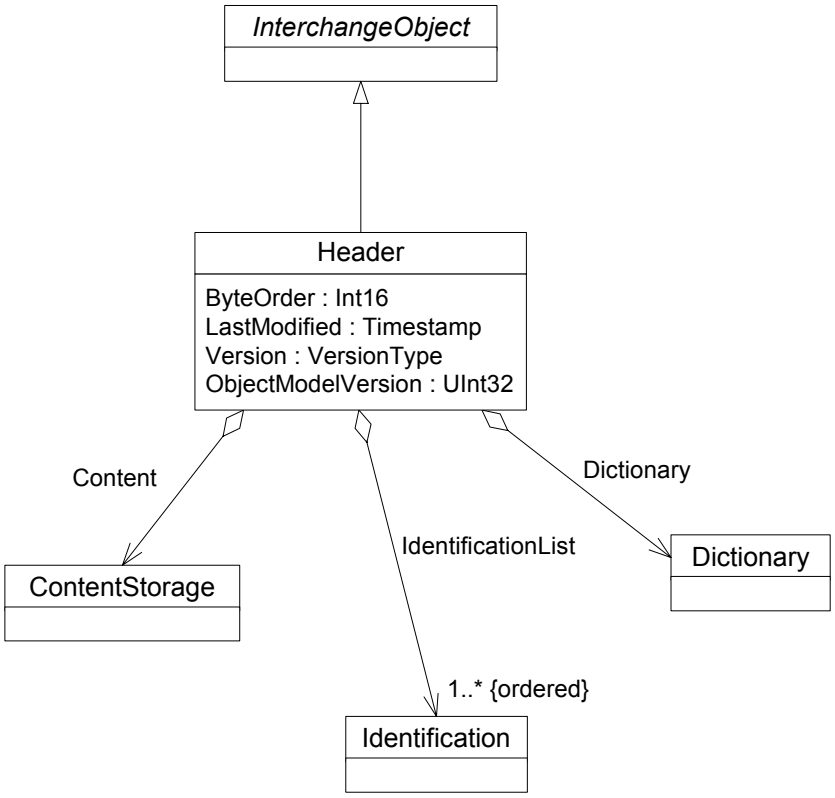


Property Name	Type	Req ?	Meaning
ObjClass	WeakReference to ClassDefinition	Req	Identifies the class of the object
Generation	AUID	Opt	Identifies when the object was created or last modified. If omitted, the object was created or last modified in the first generation of the file.

6.2 Header class

The Header class provides file-wide information and indexes. An AAF file shall have exactly one Header object.

The Header class is a sub-class of InterchangeObject.



Property Name	Type	Req ?	Meaning	Default
ByteOrder	Int16	Req	Specifies the byte order for the AAF file. One of the following: 'II' Little-endian byte order 'MM' Big-endian byte order	
LastModified	TimeStamp	Req	Time and Date the file was last modified	
Version	VersionType	Req	Version number of the AAF Object Specification document that the file is compatible with; shall be 1.0 or higher.	
Content	StrongReference to ContentStorage	Req	Has the ContentStorage object that has all Mobs and Essence Data in the file	
Dictionary	StrongReference to Dictionary	Req	Has a Dictionary object that has the DefinitionObjects defined in the AAF file	
IdentificationList	StrongReferenceVector of Identification	Req	Has an ordered set of Identification objects, which identify the application that created or modified the AAF file	
ObjectModelVersion	UInt32	Opt	The version of the persistent storage format for objects. Automatically maintained	AAF v1.0

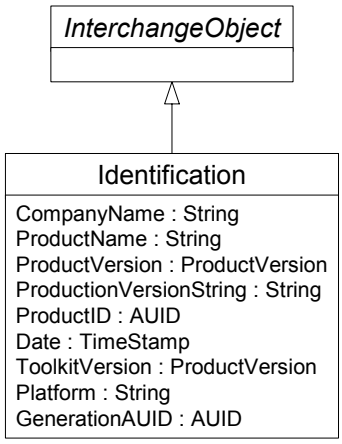
The ByteOrder property records the byte order of the computer platform on which the file was created. Subsequent modification of the file may create objects with foreign byte order; the byte order of individual modified objects shall be properly maintained by the storage format. In the big-endian byte 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.

6.3 Identification class

The Identification class provides information about the application that created or modified the file.

The Identification class is a sub-class of InterchangeObject.

All Identification objects in a file shall be included in the IdentificationList of the Header object.

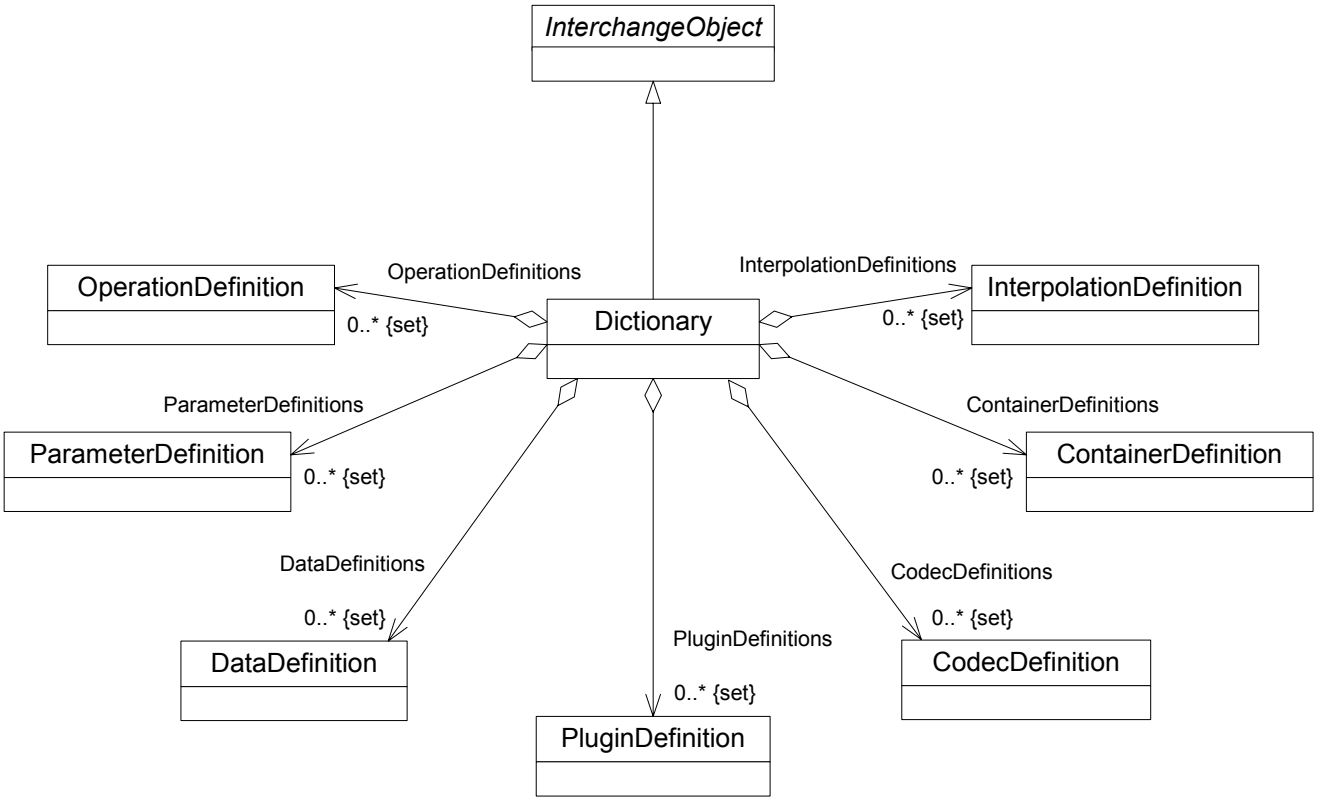


Property Name	Type	Req ?	Meaning
CompanyName	String	Req	Specifies the name of the company or organization that created the application
ProductName	String	Req	Specifies the name of the application
ProductVersion	ProductVersion	Opt	Specifies the version number of the application
ProductVersionString	String	Req	Specifies the version number of the application in string form
ProductID	AUID	Req	Identifies the application
Date	TimeStamp	Req	Time and date the application created or modified the AAF file
ToolkitVersion	ProductVersion	Opt	Specifies the version number of the SDK library
Platform	String	Opt	Specifies the platform on which the application is running
GenerationAUID	AUID	Req	AUID generated at the time the application created or opened for modification the file

6.4 Dictionary class

The Dictionary class has DefinitionObject objects. An AAF file shall have exactly one Dictionary object.

The Dictionary class is a sub-class of InterchangeObject.



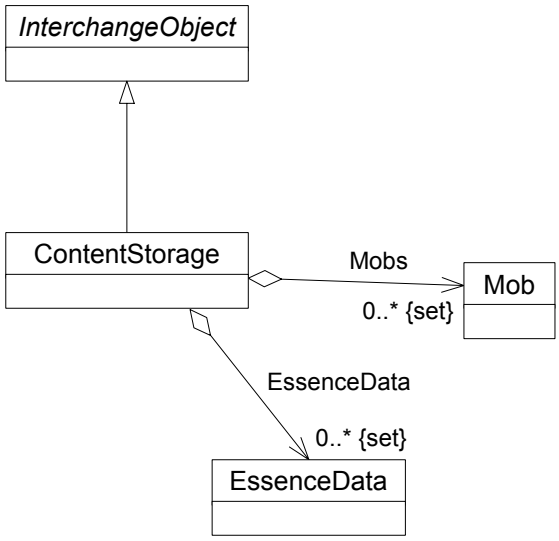
Property Name	Type	Req ?	Meaning
OperationDefinitions	StrongReferenceSet of OperationDefinition	Opt	Specifies the OperationDefinitions that are used in the file
ParameterDefinitions	StrongReferenceSet of ParameterDefinition	Opt	Specifies the ParameterDefinitions that are used in the file
DataDefinitions	StrongReferenceSet of DataDefintion	Opt	Specifies the DataDefinitions that are used in the file
PluginDefinitions	StrongReferenceSet of PluginDefinition	Opt	Identifies code objects that provide an implementation for a DefinitionObject, such as a CodecDefinition or an InterpolationDefinition
CodecDefinitions	StrongReferenceSet of CodecDefinition	Opt	Specifies CodecDefinitions that describe code that can compress or uncompress samples of EssenceData or that can convert samples to another format
ContainerDefinitions	StrongReferenceSet of ContainerDefinition	Opt	Specifies ContainerDefinitions that describe container mechanisms used to store essence
InterpolationDefinitions	StrongReferenceSet of InterpolationDefinition	Opt	Specifies InterpolationDefinitions that can calculate values in a VaryingValue based on the values specified by the ControlPoints

Informative note: The Dictionary object API in the AAF reference implementation (IAAFDictionary) is used to access the MetaDictionary.

6.5 ContentStorage class

The ContentStorage class has the Mob and EssenceData objects. An AAF file shall have exactly one ContentStorage object.

The ContentStorage class is a sub-class of InterchangeObject.



Property Name	Type	Req ?	Meaning
Mobs	StrongReferenceSet of Mob	Req	Has a set of all Mobs in the file
EssenceData	StrongReferenceSet of EssenceData	Req	Has a set of all EssenceData objects in the file

A ContentStorage may have any number of Mobs.

6.6 Mob class

The Mob class specifies a Mob, which can describe a composition, essence, or physical media.

The Mob class is a sub-class of InterchangeObject.

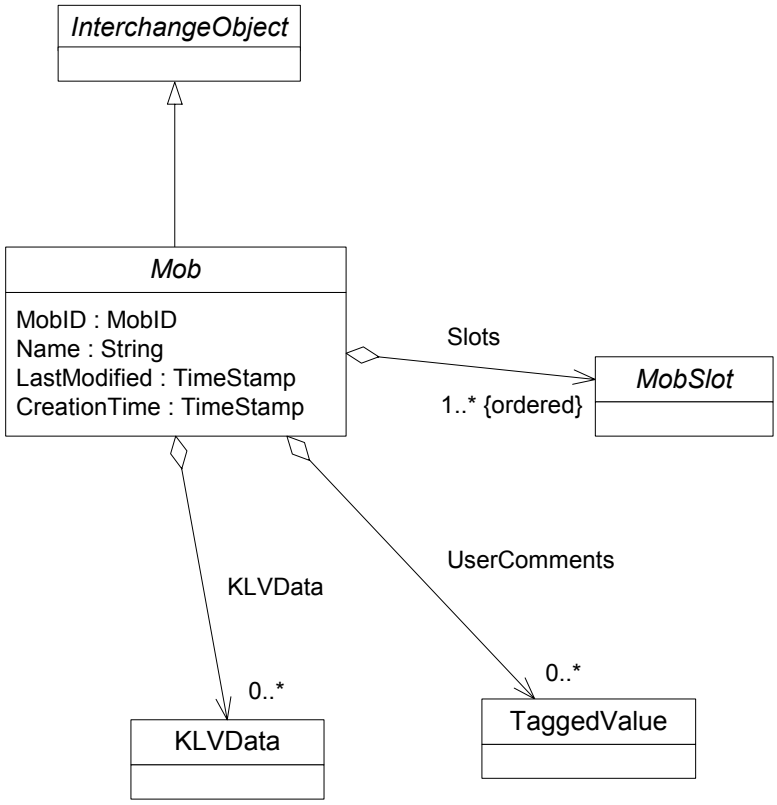
The Mob class is an abstract class.

All Mob objects shall be owned by the ContentStorage object.

Mobs have a globally unique ID, and they are the only elements of an AAF file that can be referenced from outside the file.

A Mob object shall have one or more MobSlots

Informative note: However, this is not enforced by the AAF reference implementation, and applications should expect to encounter some Mobs with no MobSlots



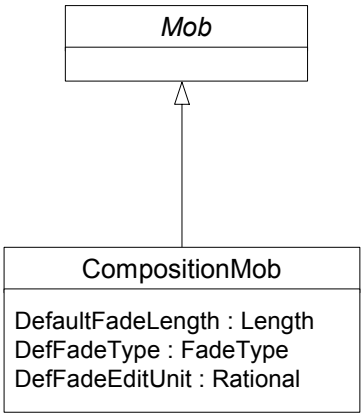
Property Name	Type	Req ?	Meaning
MobID	MobID	Req	Unique Mob Identification
Name	String	Opt	Name of Mob for display to end user
Slots	StrongReferenceVector of MobSlot	Req	Has an ordered set of MobSlots
LastModified	TimeStamp	Req	Date and time when the Mob was last modified
CreationTime	TimeStamp	Req	Date and time when the Mob was originally created
UserComments	StrongReferenceVector of TaggedValue	Opt	Has a set of TaggedValues that specify user comments
KLVData	StrongReferenceVector of KLVData	Opt	Contains a set of user KLV data consisting of a key (a SMPTE label), a length, and a value

MobSlots are ordered to allow ScopeReferences within one slot to reference another slot. The Mob defines a scope consisting of the ordered set of MobSlots. A ScopeReference object in a MobSlot can refer to any MobSlot that precedes it. A ScopeReference returns the same time-varying values as the section in the specified Mob MobSlot that corresponds to the starting point of the ScopeReference in the MobSlot and the duration of the ScopeReference. In addition to Mobs, NestedScope objects define scopes; however, their scope is limited to the Components owned by the NestedScope object's tracks.

6.7 CompositionMob class

The CompositionMob class specifies how to combine essence elements into a sequence, how to modify essence elements, and how to synchronize essence elements.

The CompositionMob class is a sub-class of the Mob class.



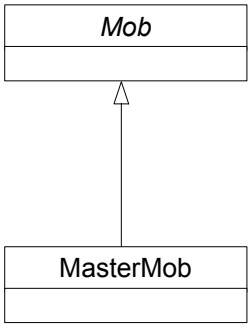
Property Name	Type	Req ?	Meaning
DefaultFadeLength	Length	Opt – see conditional rule 1	Specifies the default length of the audio fade-in and fade-out to be applied to all audio SourceClips that do not specify the audio fade properties
DefaultFadeType	FadeType	Opt – see conditional rule 1	Specifies the default type of audio fade
DefaultFadeEditUnit	Rational	Opt – see conditional rule 1	Specifies the edit units in which the default fade length is specified

Conditional rule 1 The properties DefaultFadeLength, DefaultFadeType and DefaultFadeEditUnit specify default audio fade behaviour for a CompositionMob. If any of these properties are specified, then all three shall be specified.

6.8 MasterMob class

The MasterMob class provides access to the SourceMobs and EssenceData objects

The MasterMob class is a sub-class of the Mob class.



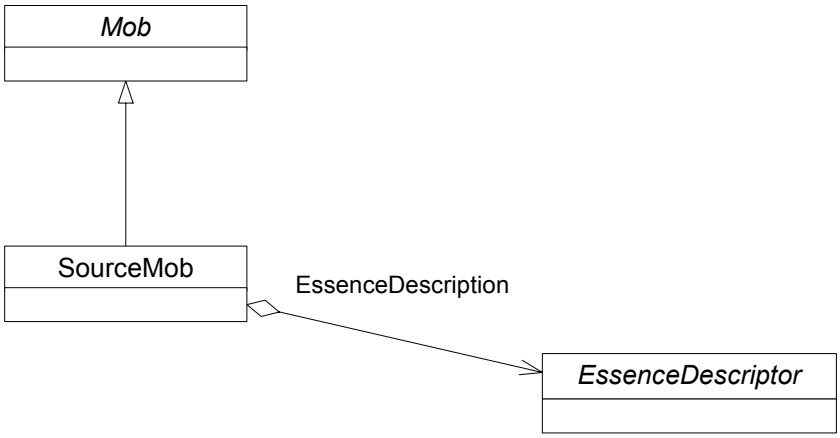
The MasterMob class does not define any additional properties.

6.9 SourceMob class

The SourceMob class describes essence that is either stored in a digital form in a file or stored on a physical media, such as tape or film.

The SourceMob class is a sub-class of the Mob class.

The essence represented by a SourceMob is immutable. If the essence changes, such as if a videotape is redigitized, you must create a new SourceMob with a new Mob ID.



Property Name	Type	Req ?	Meaning
EssenceDescription	StrongReference to EssenceDescriptor	Req	Describes the format of the essence associated with the SourceMob.

A SourceMob object shall either be a file SourceMob or a physical SourceMob. If a SourceMob has an EssenceDescriptor that belongs to the FileDescriptor class, then the SourceMob is a file SourceMob. If a SourceMob has an EssenceDescriptor that does not belong to the FileDescriptor class, then the SourceMob is a physical SourceMob.

If the digital essence is a stream of interleaved essence, then the file SourceMob should have one MobSlot for each channel of interleaved essence.

If the physical media contains more than one track of essence, then the physical SourceMob should have one MobSlot for each physical track. In addition, the physical SourceMob may have a MobSlot for timecode data and may have a MobSlot for edgecode data.

The MobSlots in a file SourceMob should have a Segment that is a SourceClip. If there is a Mob that describes the previous generation of essence, the SourceClip should specify the MobID of that Mob. The previous generation can be a physical SourceMob or another file SourceMob. If there is no previous generation of essence or there is no Mob describing it, the SourceClip should specify a zero-value SourceID property.

The MobSlot in a physical SourceMob should have a Segment that is a SourceClip, Timecode or Edgecode. If there is a Mob that describes the previous generation of essence, the SourceClip should specify the MobID of that Mob. The previous generation should be a physical SourceMob. If there is no previous generation of essence or there is no Mob describing it, the SourceClip should specify a zero-value SourceID property.

Informative note: The length of the Segment in the Mob MobSlot indicates the duration of the essence. If you create a SourceMob for a physical media source and you do not know the duration of the essence, specify a length of 24 hours.

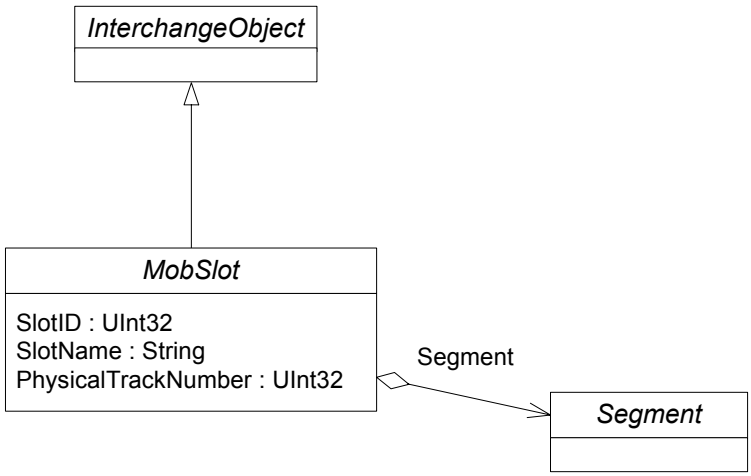
6.10 MobSlot class

The MobSlot class represents an individual track in a Mob.

The MobSlot class is a sub-class of the InterchangeObject class.

The MobSlot class is an abstract class.

All MobSlot objects shall be members of the set of MobSlots of a Mob object.



Property Name	Type	Req ?	Meaning
SlotID	UInt32	Req	Specifies an integer that is used to reference the MobSlot
SlotName	String	Opt	Specifies a text name for the MobSlot
PhysicalTrackNumber	UInt32	Opt	Specifies the physical channel
Segment	StrongReference to Segment	Req	Specifies the value of the MobSlot

6.11 TimelineMobSlot class

The TimelineMobSlot class describes time-varying timeline essence.

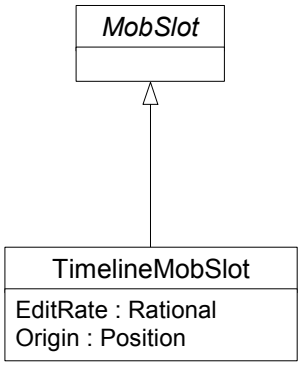
The TimelineMobSlot class is a sub-class of the MobSlot class.

If a TimelineMobSlot has a Sequence containing an Event, then the Sequence shall conform to the following rules:

All Segments in the Sequence shall be Events or Fillers.

All Events in the Sequence shall belong to the same concrete sub-class of Event.

All Events and Fillers in the Sequence shall have the same data definition as the Sequence.



Property Name	Type	Req ?	Meaning
EditRate	Rational	Req	Specifies the units of time for the TimelineMobSlot
Origin	Position	Req	Specifies the offset used to resolve SourceClip references to this TimelineMobSlot. A positive value of Origin means that the first sample of the essence is earlier than the zero position. A negative value of Origin means that the zero position is earlier than the first sample of the essence.

The TimelineMobSlot specifies the edit rate for the Segment it has. The Segment specifies its length in the edit rate set by the TimelineMobSlot. The Segment also specifies its own data kind.

6.12 EventMobSlot class

The EventMobSlot class has a Sequence of Events.

The EventMobSlot class is a sub-class of the MobSlot class.

An EventMobSlot shall have a concrete Segment that is either a concrete sub-class of Event or a Sequence.

If an EventMobSlot has a Sequence, then the Sequence shall conform to the following rules:

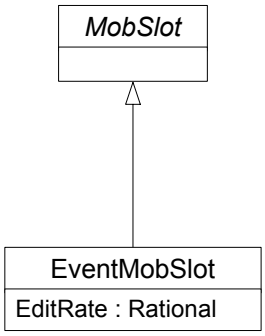
All Segments in the Sequence shall be Events.

All Events in the Sequence shall belong to the same concrete sub-class of Event.

All Events in the Sequence shall have the same data definition as the Sequence.

In a Sequence, the Position of each Event shall be greater than or equal to the Position of the Event preceding it in the Sequence.

Informative note: To correlate the Events in an EventMobSlot with a TimelineMobSlot to which they may refer, the EventMobSlot may be given the same data definition and the same PhysicalTrackNumber as the target TimelineMobSlot.

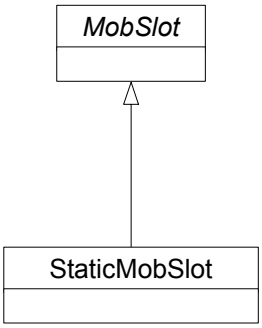


Property Name	Type	Req ?	Meaning
EditRate	Rational	Req	Specifies the units in which the events specify their starting time and duration

6.13 StaticMobSlot class

The StaticMobSlot describes essence data that has no relationship to time, such as a static image.

The StaticMobSlot class is a sub-class of the MobSlot class.

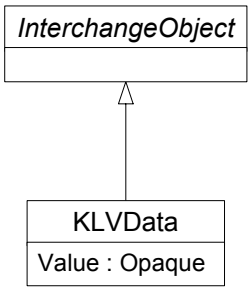


The StaticMobSlot class does not define any additional properties. StaticMobSlot objects have Segments that do not have any relationship with time; consequently, a StaticMobSlot does not define an edit rate.

6.14 KLVData class

The KLVData class contains user data specified with a Key (SMPTE label), Length, and Value.

The KLVData class is a sub-class of the InterchangeObject class.

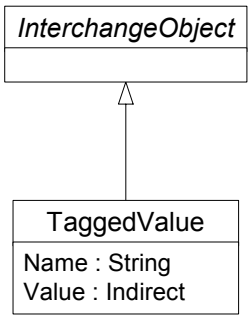


Property Name	Type	Req ?	Meaning
Value	Opaque	Req	Specifies the key, length, and value

6.15 TaggedValue class

The TaggedValue class specifies a user-defined tag and value.

The TaggedValue class is a sub-class of the InterchangeObject class.



Property Name	Type	Req ?	Meaning
Name	String	Req	User-specified tag
Value	Indirect	Req	User-specified value

6.16 Parameter class

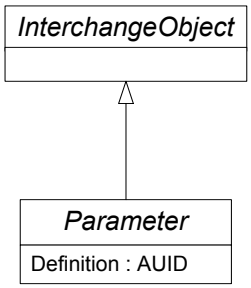
The Parameter class specifies an effect control value.

The Parameter class is a sub-class of the InterchangeObject class.

The Parameter class is an abstract class.

A Parameter object shall be owned by an OperationGroup object.

A Parameter object is an effect control, which specifies values for adjustments in the way the effect should be performed. An effect can have constant control parameters or have control parameters 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 parameters, 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 ConstantValue object. A time-varying value is specified with a VaryingValue object.

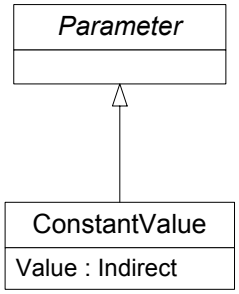


Property Name	Type	Req ?	Meaning
Definition	AUID	Req	Identifies the Parameter

6.17 ConstantValue class

The ConstantValue class specifies a constant data value for an effect control value.

The ConstantValue class is a sub-class of the Parameter class.

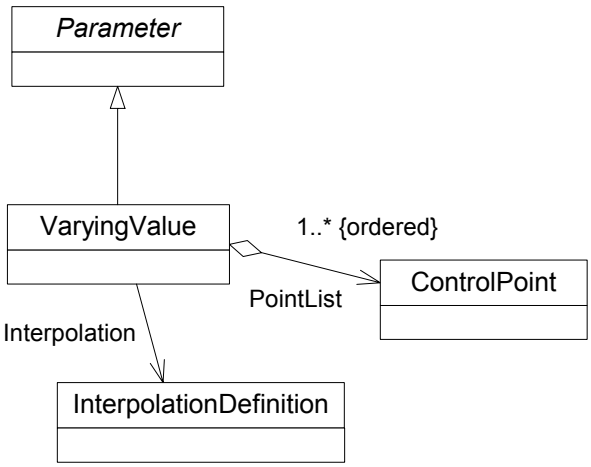


Property Name	Type	Req ?	Meaning
Value	Indirect	Req	Specifies the value

6.18 VaryingValue class

The VaryingValue class specifies a changing data value for an effect control value.

The VaryingValue class is a sub-class of the Parameter class.



Property Name	Type	Req ?	Meaning
Interpolation	WeakReference to InterpolationDefinition	Req	Specifies the kind of interpolation to be used to find the value between ControlPoints
PointList	StrongReferenceVector of ControlPoint	Req	Has an array of ControlPoints, each of which specifies a value and a time point at which the value is defined

A VaryingValue object shall have at least one ControlPoint. A VaryingValue object should have at least two Control Points, one should specify a value at the time 0.0 and another should specify a value at the time 1.0.

ControlPoints shall be ordered by their time value.

A VaryingValue object is a Parameter that returns time-varying values that are determined by an ordered set of ControlPoints. Each ControlPoint specifies the value for a specific time point within the Segment. The values for time points between two ControlPoints are calculated by interpolating between the two values.

A ControlPoint that has a Time value equal to 0.0 represents the time at the beginning of the VaryingValue object; one with a time equal to 1.0 represents the time at the end of the VaryingValue object. ControlPoints 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 VaryingValue object—they do not affect values outside of the duration of the VaryingValue 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.

If more than two ControlPoint objects specify the same value, the last ControlPoint determines the value for the time point specified and is used to interpolate values after this time point.

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

$$Value_X = (Time_X – Time_A) / (Time_B - Time_A) \times (Value_B – Value_A) + Value_A$$

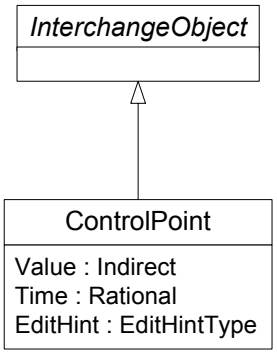
If the first ControlPoint in a VaryingValue object specifies a time value greater than 0, this value is extrapolated to the 0.0 time point by holding the value constant. If the last ControlPoint in a VaryingValue object specifies a time value less than 1.0, this value is extrapolated to the 1.0 time point by holding the value constant. This extrapolation method of holding values is used if the interpolation method specified for the Varying alue object is constant or linear interpolation.

The VaryingValue object specifies a value for each time point within the VaryingValue object; however if you are generating a stream of essence from the CompositionMob owning the VaryingValue object, it may be important to adjust values produced by the VaryingValue object based on sample-rate quantization. Within an essence sample unit, there can only be a single value of the VaryingValue object when generating that sample.

6.19 ControlPoint class

The ControlPoint class specifies a value and a time point and is used to specify an effect control value.

The ControlPoint class is a sub-class of the InterchangeObject class.



Property Name	Type	Req ?	Meaning
Value	Indirect	Req	Specifies the value

Property Name	Type	Req ?	Meaning
Time	Rational	Req	Specifies the time within the VaryingValue segment for which the value is defined
EditHint	EditHintType	Opt	Specifies a hint to be used if the effect starting time or length is changed during editing

A ControlPoint object specifies the value at a specific time in a VaryingValue object. The values of all ControlPoint objects owned by a VaryingValue must have the same type.

A Time equal to 0.0 represents the time at the beginning the VaryingValue Object; a Time equal to 1.0 represents the time at the end of the VaryingValue object

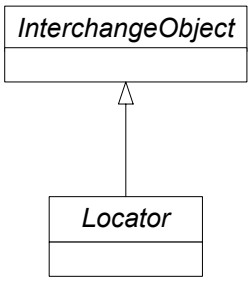
6.20 Locator class

The Locator class provides information to help find a file that contains the essence or to help find the physical media.

The Locator class is a sub-class of the InterchangeObject class.

The Locator class is an abstract class.

A Locator object shall be a member of the array of Locators in an EssenceDescriptor.

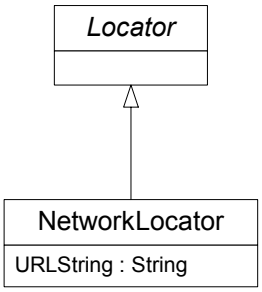


The Locator class does not define any additional properties.

6.21 NetworkLocator class

The NetworkLocator class provides information to help find a file containing essence.

The NetworkLocator class is a sub-class of the Locator class.



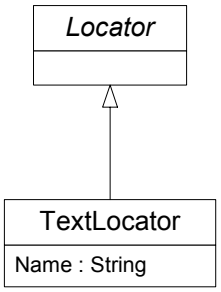
Property Name	Type	Req ?	Meaning
URLString	String	Req	Universal Resource Locator (URL) for file containing the essence

The NetworkLocator has a URL that provides a hint to help an application find a file containing the essence data.

6.22 TextLocator class

The TextLocator class provides information to help find a file containing the essence or to help find the physical media.

The TextLocator class is a sub-class of the Locator class.



Property Name	Type	Req ?	Meaning
Name	String	Req	Text string containing information to help find the file containing the essence or the physical media

A TextLocator object provides information to the user to help locate the file containing the essence or to locate the physical media. The TextLocator is not intended for applications to use without user intervention.

7 Component Classes

This chapter includes the class specifications for classes in the Component package. Figure 15 shows the class hierarchy for the Component package.

The class specification pages are presented in order according to the class hierarchy, reading the Figure left to right, depth-first.

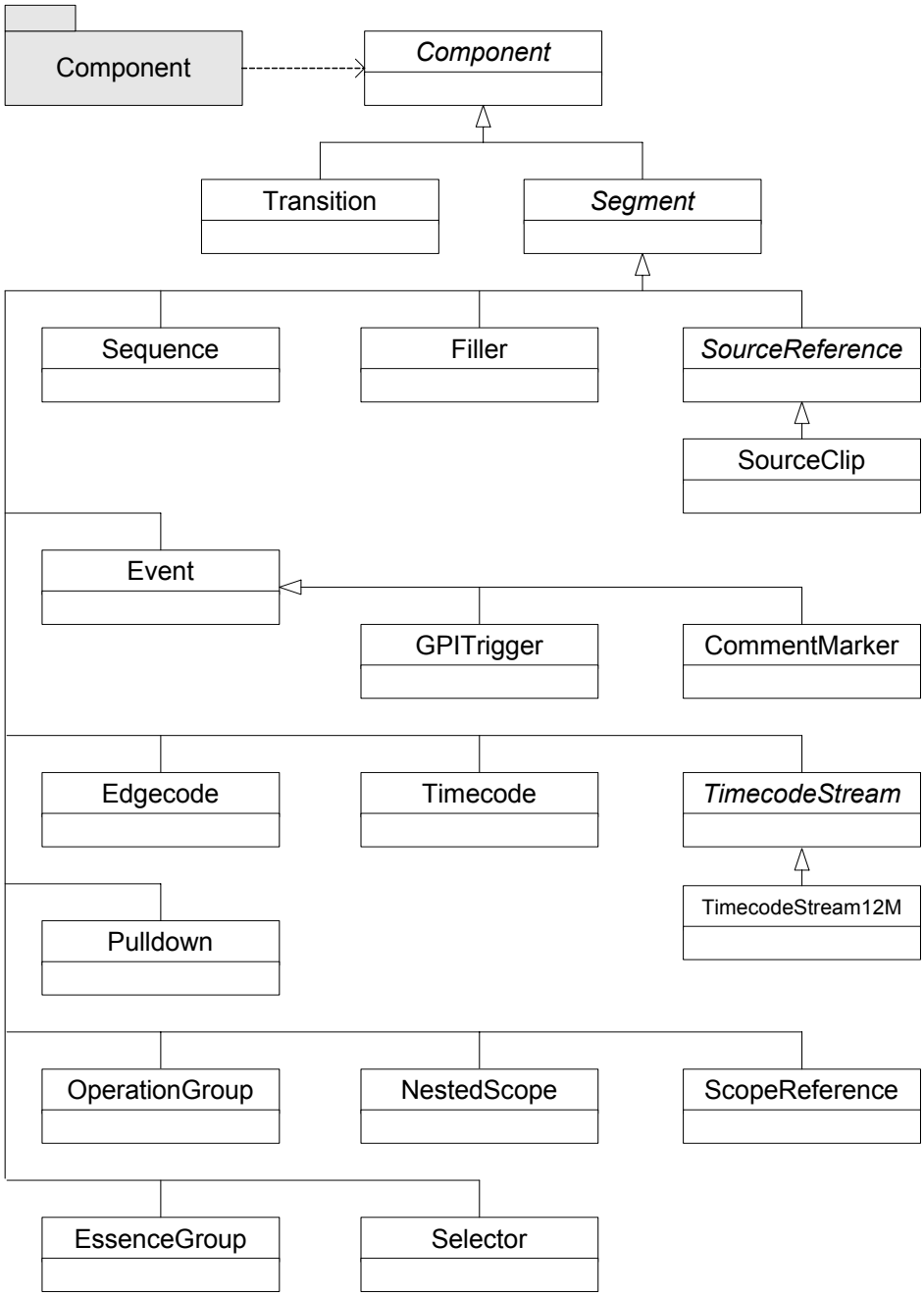


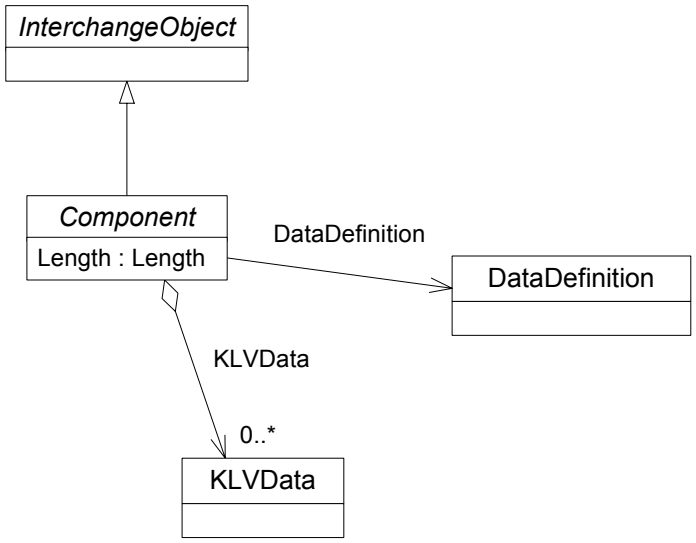
Figure 15 Component Package

7.1 Component class

The Component class represents an essence element.

The Component class is a sub-class of InterchangeObject.

The Component class is an abstract class.



Property Name	Type	Req ?	Meaning
DataDefinition	WeakReference to DataDefinition	Req	Specifies the DataDefinition object that specifies the kind of data described by the component
Length	Length	Opt – see conditional rule 1	Specifies the duration in edit units of the component
KLVData	StrongReferenceVector of KLVData	Opt	Contains a set of user KLV data consisting of a key (a SMPTE label), a length, and a value.

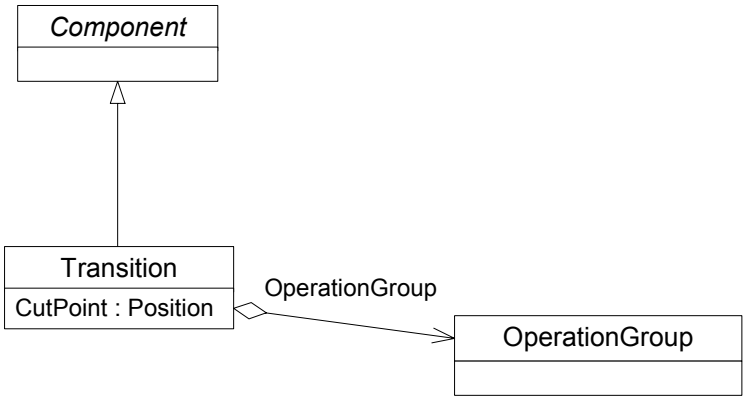
Conditional rule 1 If a Component is in a TimelineMobSlot, then it shall have a Length property. If a Component is in a StaticMobSlot, then it shall not have a Length property. If a Component is in an EventMobSlot, then it may have a Length property. If a Component in an EventMobSlot does not have a Length property, then the Component describes an instantaneous event that does not have a duration.

7.2 Transition class

The Transition class specifies that the two adjacent Segments should be overlapped when they are played and the overlapped sections should be combined using the specified effect.

The Transition class is a sub-class of the InterchangeObject class.

A Transition object shall be in a Sequence within a CompositionMob.



Property Name	Type	Req ?	Meaning
OperationGroup	StrongReference to OperationGroup	Req	Has an OperationGroup that specifies the effect to be performed during the Transition
CutPoint	Position	Req	Specifies a cutpoint to use if replacing the Transition with a cut.

The OperationGroup in a Transition shall specify an OperationDefinition with two input Segments. The OperationGroup in a Transition shall not have the InputSegments specified. The input segments are implicitly provided by the Segments preceding and following the Transition.

A Transition object specifies that sections of the preceding and following segments overlap for the duration of the Transition. The effect combines the essence from the overlapping sections in some way.

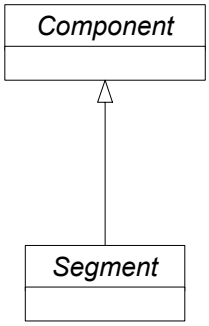
The Transition cut point has no direct effect on the results produced by a Transition. However, the cut point provides information that is useful if an application wishes to remove the Transition or substitute a cut when playing the Transition. The cut point is represented as an offset from the beginning of the Transition. When removing the Transition, an application would change the CompositionMob 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.

7.3 Segment class

The Segment class represents a Component that is independent of any surrounding object.

The Segment class is a sub-class of the Component class.

The Segment class is an abstract class.

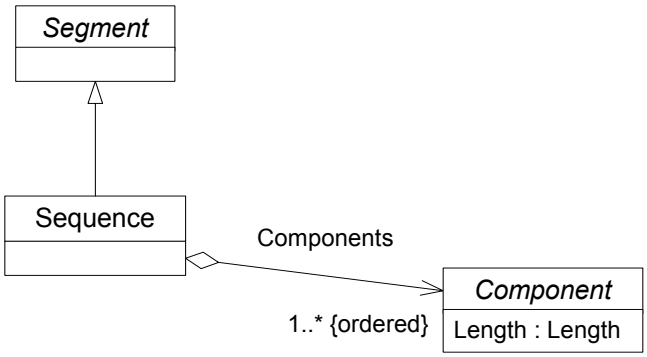


The Segment class does not define any additional properties.

7.4 Sequence class

The Sequence class combines an ordered list of Segments and Transitions.

The Sequence class is a sub-class of the Segment class.



Property Name	Type	Req ?	Meaning
Components	StrongReferenceVector of Component	Req	Has an ordered set of Component objects

The first and last Component in the ordered set shall be Segment objects

A Transition object shall only appear in a Sequence between two Segment objects. The length of each of these Segments shall be greater than or equal to the length of the Transition.

If a Segment object has a Transition before it and after it, the sum of the lengths of the surrounding Transitions shall be less than or equal to the length of the Segment that they surround.

The length of the Sequence shall be equal to the sum of the length of all Segments directly owned by the Sequence minus the sum of the lengths of all Transitions directly owned by the Sequence.

The data definition of each Component in the Sequence object shall be the same as the data definition of the Sequence.

The Sequence object is the mechanism for combining sections of essence to be played in a sequential manner.

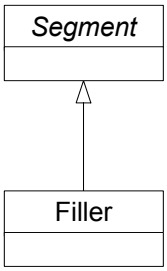
If a Sequence object has a Segment followed by another Segment, after the first Segment is played, the following one begins immediately

If a Sequence object has 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.

7.5 Filler class

The Filler class represents an unspecified value for the duration of the object.

The Filler class is a sub-class of the Segment class.



The Filler class does not define any additional properties.

Informative note: 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 MobSlots and NestedScope tracks that are not referenced or played.

If a Filler object is played, applications can choose any appropriate blank essence 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.

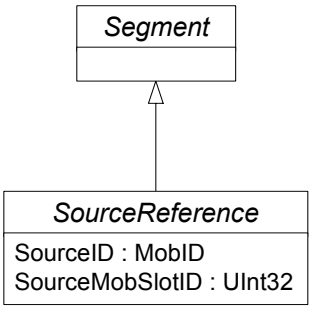
7.6 SourceReference class

The SourceReference class represents the essence or other data described by a MobSlot in a Mob.

The SourceReference class is a sub-class of the Segment class.

The SourceReference class is an abstract class.

A SourceReference object in a Mob refers to a MobSlot in another Mob by specifying the second Mob's MobID and the SlotID of the MobSlot owned by it.



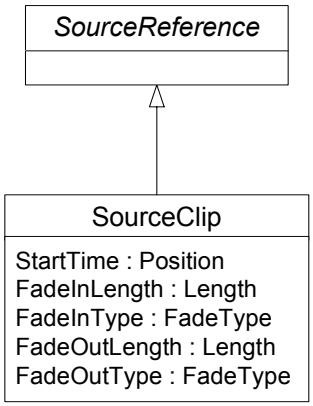
Property Name	Type	Req ?	Meaning
SourceID	MobID	Opt	Identifies the Mob being referenced. If the property has a value 0, it means that the Mob owning the SourceReference describes the original source
SourceMobSlotID	UInt32	Req	Specifies the SlotID of a Mob Slot within the specified Mob. If the SourceID has a value 0, then SourceMobSlotID shall also have a 0 value

To create a SourceReference that refers to a MobSlot within the same Mob as the SourceReference, omit the SourceID property.

7.7 SourceClip class

The SourceClip class represents the essence and identifies the source of the essence.

The SourceClip class is a sub-class of the SourceReference class.



Property Name	Type	Req ?	Meaning
StartTime	Position	Opt – see conditional rule 1	Specifies the offset from the origin of the referenced Mob MobSlot in edit units determined by the SourceClip object's context. If the SourceID has a value 0, then StartTime shall also have a 0 value
FadeInLength	Length	Opt – see conditional rule 2	Specifies the length of an audio fade in to be applied to the SourceClip
FadeInType	FadeType	Opt – see conditional rule 2	Specifies the type of the audio fade in
FadeOutLength	Length	Opt – see conditional rule 2	Specifies the length of an audio fade out to be applied to the SourceClip
FadeOutType	FadeType	Opt – see conditional rule 2	Specifies the type of the audio fade out

Conditional rule 1 If the SourceClip references a TimelineMobSlot or an EventMobSlot, then the StartTime property shall be specified. If the SourceClip references a StaticMobSlot, then the StartTime property shall not be specified.

Conditional rule 2 If the SourceClip data definition is not a Sound, then the fade properties should not be present.

If a SourceMob represents the original essence source and there is no previous generation, then its SourceClips shall specify a value 0 for its SourceID and 0 values for SourceMobSlotID and StartTime.

The data definition of the Segment owned by the referenced MobSlot shall be the same as the data definition of the SourceClip object.

A SourceClip's StartTime and Length values are in edit units determined by the slot owning the SourceClip.

Informative note: If the SourceClip references a MobSlot that specifies a different edit rate than the MobSlot owning the SourceClip, the StartTime and Length are in edit units of the slot owning the SourceClip, and not edit units of the referenced slot.

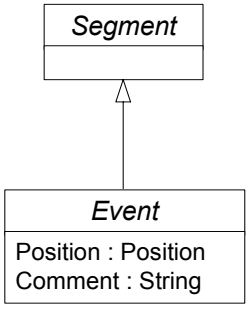
In a CompositionMob, SourceClips reference a section of essence by specifying the MasterMob that describes the essence. In a MasterMob, SourceClips reference the essence by specifying the file SourceMob that is associated with the essence. In a file SourceMob, SourceClips reference the essence stored on a physical media, such as tape or film, by specifying the physical SourceMob that describes the media. In a physical

SourceMob, SourceClips reference the essence stored on a previous generation of physical media by specifying the physical SourceMob that describes the media.

7.8 Event class

The Event class defines a text comment, a trigger, or an area in the image that has an associated interactive action.

The Event class is a sub-class of the Segment class.



Property Name	Type	Req ?	Meaning
Position	Position	Req – see conditional rule 1	Specifies the starting time of the event in an EventMobSlot
Comment	String	Opt	Specifies the purpose of the event

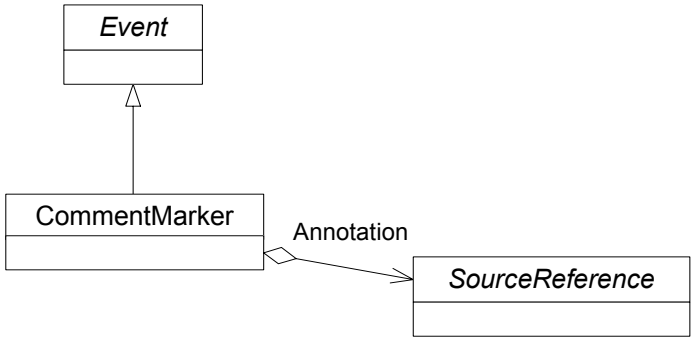
Conditional rule 1 If an Event is in a TimelineMobSlot or a StaticMobSlot, it shall not have a Position property. If an Event is in a EventMobSlot, it shall have a Position property.

An Event's Position is an absolute time expressed in edit units of the MobSlot owning the Event.

7.9 CommentMarker class

The CommentMarker class specifies a user comment that is associated with a point in time.

The CommentMarker class is a sub-class of the Event class.



Property Name	Type	Req ?	Meaning

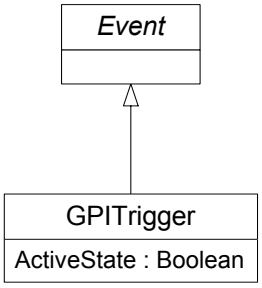
Property Name	Type	Req ?	Meaning
Annotation	StrongReference to SourceReference	Opt	Specifies text or audio annotation

Informative note: To correlate CommentMarkers with a MobSlot to which they may refer, the EventMobSlot may be given the same data definition and the same PhysicalTrackNumber as the target MobSlot.

7.10 GPITrigger class

The GPITrigger class specifies a trigger action that should be taken when the GPITrigger is reached.

The GPITrigger class is a sub-class of the Event class.



Property Name	Type	Req ?	Meaning
ActiveState	Boolean	Req	Specifies whether the event is turned on or off

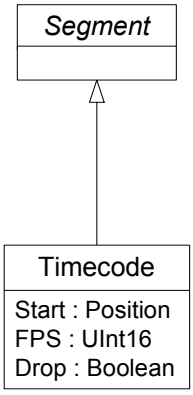
A GPITrigger object specifies a trigger action that should be taken when its position in time is reached. The ActiveState property specifies whether the trigger should be set on or off.

7.11 Timecode class

The Timecode class stores videotape or audio tape timecode information.

The Timecode class is a sub-class of the Segment class.

A Timecode object shall have a Timecode data definition.



Property Name	Type	Req ?	Meaning
Start	Position	Req	Specifies the timecode at the beginning of the segment
FPS	UInt16	Req	Frames per second of the videotape or audio tape
Drop	Boolean	Req	Indicates whether the timecode is drop (True value) or nondrop (False value)

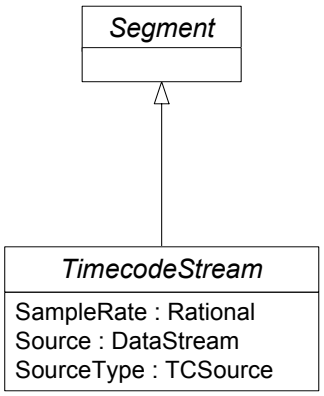
Informative note: A Timecode object can typically appear in either a SourceMob or in a Composition Mob. In a SourceMob, it typically appears in a MobSlot in a SourceMob that describes a videotape or audio tape. In this context, it describes the timecode that exists on the tape. In a Composition Mob, 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.

7.12 TimecodeStream class

The TimecodeStream class specifies a stream of timecode data.

The TimecodeStream class is a sub-class of the Segment class.

The TimecodeStream class is an abstract class.



Property Name	Type	Req ?	Meaning
SampleRate	Rational	Req	Specifies the sample rate of the timecode data contained in the Source property
Source	DataStream	Req	Contains the timecode data
SourceType	TCSource	Req	Specifies the kind of timecode

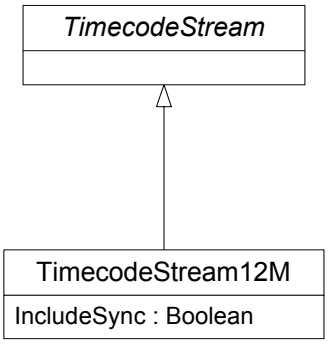
Informative note: A Timecode object can typically appear in either a SourceMob or in a Composition Mob. In a SourceMob, it typically appears in a MobSlot in a SourceMob that describes a videotape or audio tape. In this context, it describes the timecode that exists on the tape. In a Composition Mob, 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.

7.13 TimecodeStream12M class

The TimecodeStream12M class specifies a stream of timecode data in the SMPTE 12M format.

The TimecodeStream12M class is a sub-class of the TimecodeStream class.

A TimecodeStream12M object shall be a Segment in a TimelineMobSlot and have a Timecode data definition.



Property Name	Type	Req ?	Meaning	Default
IncludeSync	Boolean	Opt	Specifies whether the synchronization data is included in the timecode stream	False

Informative note: In the AAF reference implementation, the IncludeSync property is currently only accessible using the property direct interface

TimecodeStream and TimecodeStream12M specify a stream of timecode data. TimecodeStream12M conforms to the SMPTE 12M format. If the IncludeSync property has a true value, the synchronization data is included for each frame. If the IncludeSync property is false, the synchronization data, which has a fixed value, is omitted from the timecode stream.

In contrast to TimecodeStream, Timecode specifies a timecode by specifying the starting timecode value; other timecode values are calculated from the starting timecode and the time offset.

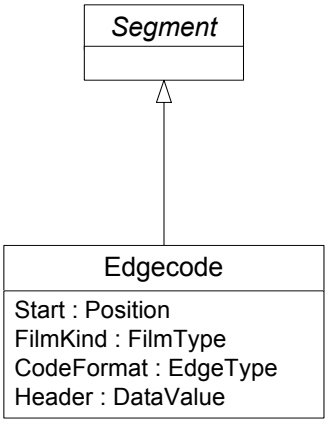
Informative note: TimecodeStream is useful to store user bits that were specified in the timecode on the videotape.

7.14 Edgecode class

The Edgecode class stores film edge code information.

The Edgecode class is a sub-class of the Segment class.

An Edgecode object shall have an Edgecode data definition.



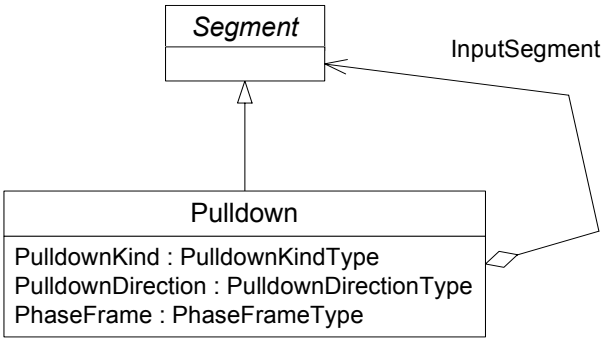
Property Name	Type	Req ?	Meaning
Start	Position	Req	Specifies the edge code at the beginning of the segment
FilmKind	FilmType	Req	Specifies the type of film
CodeFormat	EdgeType	Req	Specifies the edge code format
Header	DataValue	Opt	Specifies the text prefix that identifies the film. Typically, this is a text string of no more than 8 7-bit ISO characters

7.15 Pulldown class

The Pulldown class converts between film frame rates and videotape frame rates.

The Pulldown class is a sub-class of the Segment class.

Informative note: Pulldown objects are typically used in file SourceMobs and physical SourceMobs.



Property Name	Type	Req ?	Meaning
InputSegment	StrongReference to Segment	Req	Has a Segment that is either a SourceClip or Timecode. The length of the SourceClip or Timecode object is in the edit units determined by the PulldownKind and PulldownDirection
PulldownKind	PulldownKindType	Req	Specifies whether the Pulldown object is converting from nominally 30 Hz or 25 Hz video frame rate and whether frames are dropped or the video is played at another speed
PulldownDirection	PulldownDirectionType	Req	Specifies whether the Pulldown object is converting from tape to film speed or from film to tape speed
PhaseFrame	PhaseFrameType	Req	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

A Pulldown object provides a mechanism to convert from essence between video and film rates and describes the mechanism that was used to convert the essence.

Informative note: Pulldown objects are typically used in three ways:

In a tape SourceMob to describe how the videotape was created from film.

In a file SourceMob that has digital essence at film speed to describe how the digital essence was created from the videotape.

In a Mob to create Timecode tracks at different edit rates.

The object owned by the Pulldown object has an edit time specified by the essence speed that the Pulldown object is converting from.

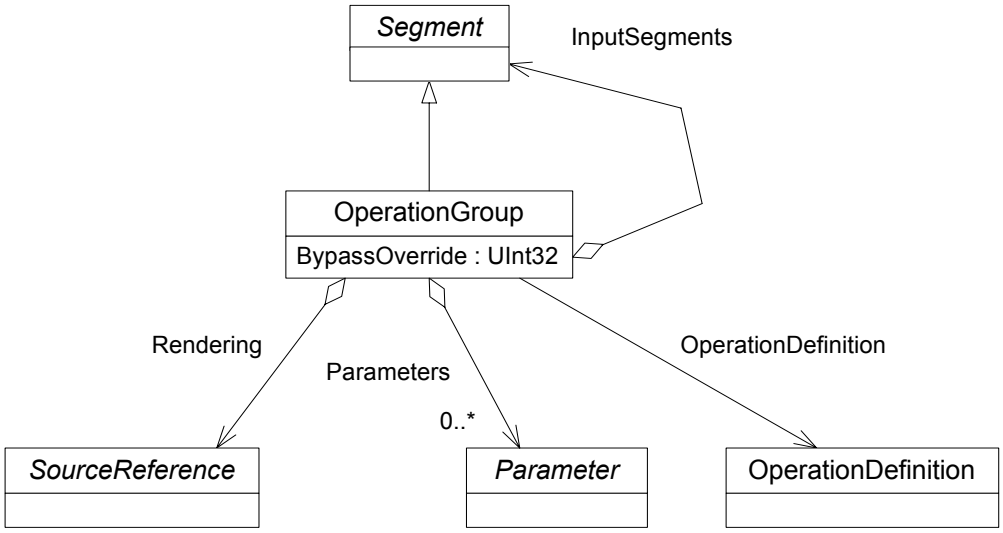
Each kind of pulldown identifies the speed of the tape. If two SourceMobs have a pulldown relationship, the edit rates of the video tracks should correspond to the frame rate of the essence.

7.16 OperationGroup class

The OperationGroup class contains an ordered set of Segments and an operation that is performed on these Segments.

The OperationGroup class is a sub-class of the Segment class.

An OperationGroup object shall be part of a CompositionMob.



Property Name	Type	Req ?	Meaning
OperationDefinition	WeakReference to OperationDefinition	Req	Has a weak reference to an Operation Definition that identifies the kind of operation
InputSegments	StrongReferenceVector of Segment	Opt	Has an array of input segments for the operation
Parameters	StrongReferenceVector of Parameter	Opt	Has an array of control Parameters. The order is not meaningful
Rendering	StrongReference to SourceReference	Opt	Specifies a rendered or precomputed version of the operation
BypassOverride	UInt32	Opt	Specifies the array index (1-based) of the input segment which is the primary input. This overrides any bypass specified by the OperationDefinition

Informative note: In the AAF reference implementation, the Parameters property is implemented as a set, not as a vector as specified here and in the MetaDictionary.

The length of the Rendering SourceClip shall equal the length of the OperationGroup.

In OperationGroup objects whose Operation Definition object does not specify a time warp the length of each input Segment shall equal the length of the OperationGroup.

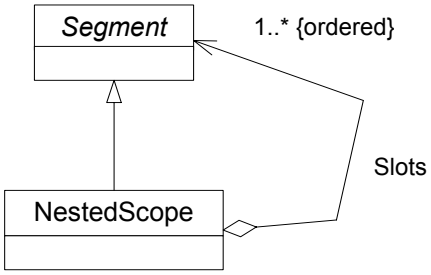
In an OperationGroup object that is in a Transition, the input segments are provided by the Transition and the InputSegments property shall be omitted.

7.17 NestedScope class

The NestedScope class defines a scope and has an ordered set of Segments.

The NestedScope class is a sub-class of the Segment class.

A NestedScope object shall be part of a CompositionMob.



Property Name	Type	Req ?	Meaning
Slots	StrongReferenceVector of Segment	Req	Has an ordered set of Segments; the last segment provides the value for the Nested Scope object.

The length of each Segment object in the Slots vector shall be equal to the length of the NestedScope object.

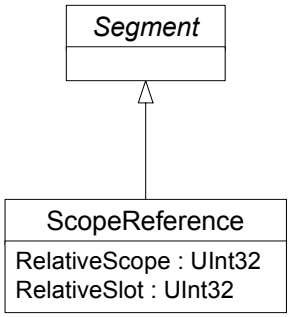
The data kind of the last Segment in the Slots vector shall be the same as the data kind of the NestedScope object.

Informative note: A Nested Scope object defines a scope and has 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.

7.18 ScopeReference class

The ScopeReference class refers to a section in the specified MobSlot or NestedScope slot.

The ScopeReference class is a sub-class of the Segment class.



Property Name	Type	Req ?	Meaning
RelativeScope	UInt32	Req	Specifies the number of Nested Scopes to pass to find the Nested Scope or Mob owning the slot.
RelativeSlot	UInt32	Req	Specifies the number of slots that precede the slot owning the Scope Reference to pass to find the slot referenced

The data kind of the Segment in the referenced slot shall be the same as the data kind of the Scope Reference object.

The value of RelativeScope shall be less than or equal to the number of Nested Scope objects that has the Scope Reference. If the Scope Reference is not owned by a Nested Scope object, then it can only refer to a slot defined by the Mob's scope and the RelativeScope shall have a value of 0.

Informative note: A RelativeScope value of 0 specifies the current scope, that is the innermost Nested Scope object that has the Scope Reference or the Mob scope if no Nested Scope object has it. A value of 1 specifies the scope level that has the Nested Scope object that has the Scope Reference.

The value of RelativeSlot shall be greater than 0 and less than or equal to the number of slots that precede it within the scope specified by RelativeScope.

Informative note: A RelativeSlot value of 1 specifies the immediately preceding slot.

A ScopeReference's Length value and the offset in the slot owning the ScopeReference are in edit units determined by the slot owning the ScopeReference.

Informative note: If the ScopeReference references a MobSlot that specifies a different edit rate than the MobSlot owning the ScopeReference, the Length value and the offset in the slot owning the ScopeReference are in edit units of the slot owning the ScopeReference, and not edit units of the referenced slot.

A Scope Reference object has the same time-varying values as the section of the Nested Scope slot or MobSlot that it references. Scope Reference objects allow one or more objects to share the values produced by a section of a slot.

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 MobSlot has a different edit rate than the Mob MobSlot owning the Scope Reference, the starting position and duration are converted to the specified Mob MobSlots 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.

If a ScopeReference refers to a MobSlot, the MobSlot shall belong to the same sub-class of MobSlot as the MobSlot owning the ScopeReference object.

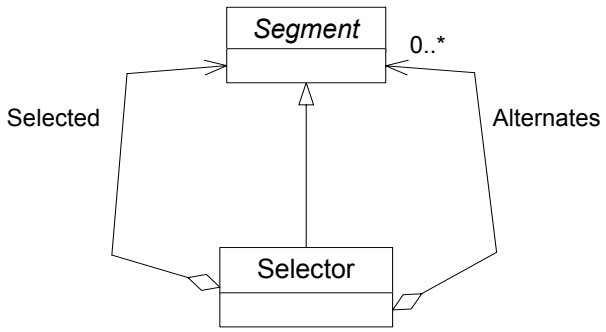
Informative note: This means that ScopeReferences should not be used to convert between timeline, static, and event data; use SourceClips or SourceClips in conjunction with OperationGroups to perform these conversions.

Informative note: Applications may encounter some Scope References referring to non-existent slots within a relative scope, therefore they should check that the referenced slot actually exists.

7.19 Selector class

The Selector class provides the value of a single Segment while preserving references to unused alternatives.

The Selector class is a sub-class of the Segment class.



Property Name	Type	Req ?	Meaning
Selected	StrongReference to Segment	Req	Has the selected Segment
Alternates	StrongReferenceVector of Segment	Opt	Has a set of unused alternative Segments

The duration of the selected Segment and of each alternative Segment shall equal the duration of the Selector object.

The data kind of the selected Segment and of each alternative Segment shall be the same as the data kind of the Selector object.

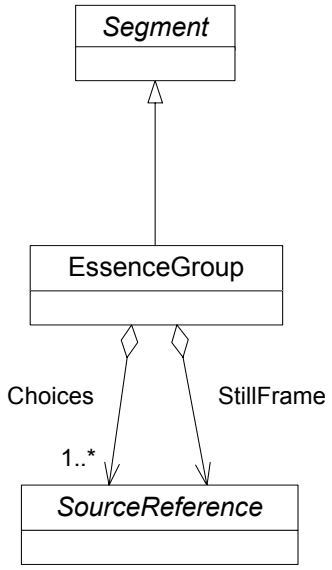
A Selector object represents an editing decision. This is in contrast with an Essence Group object which presents a group of alternative implementations of the same essence that the application can choose from based on the most appropriate or efficient essence format among the alternatives.

7.20 EssenceGroup class

The EssenceGroup class describes multiple digital representations of the same original content source.

The EssenceGroup class is a sub-class of the Segment class.

An EssenceGroup object shall be a Segment in a MasterMob MobSlot. An EssenceGroup shall only be the direct child of a MobSlot, i.e., a MobSlot can have an EssenceGroup which represents a choice between a Sequence of resolution A or a Sequence of resolution B or a Sequence of resolution C, but all Components in each Sequence shall be of the same resolution (the Components may also be Filler).



Property Name	Type	Req ?	Meaning
Choices	StrongReferenceVector of Segment	Req	Has a collection of Segments that identify the alternate representations that may be chosen. The order of the items in the collection is not meaningful
StillFrame	StrongReference to SourceReference	Opt	Has a SourceReference that identifies the essence for a single-frame image representation of the essence

The Segment shall either be a SourceClip or a Sequence. If the Segment is a Sequence, it shall contain only SourceClip and Filler objects.

The length of each Segment in the Choices set shall be the same as the length of the EssenceGroup object.

The length of the StillFrame SourceClip shall be 1.

Informative note: Typically the different representations vary in essence format, compression, or frame size. The application is responsible for choosing the appropriate implementation of the essence.

8 DefinitionObject Classes

This chapter includes the class specifications for classes in the DefinitionObject package. Figure 16 shows the class hierarchy for the DefinitionObject package.

The class specification pages are presented in order according to the class hierarchy, reading the Figure left to right, depth-first.

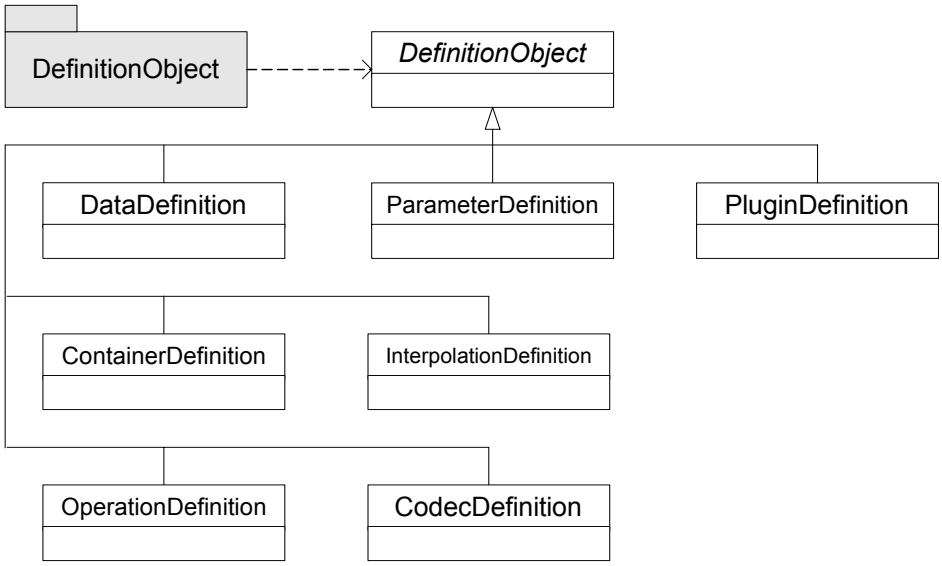


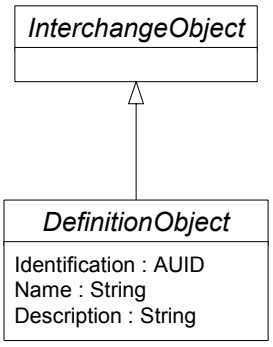
Figure 16 DefinitionObject Package

8.1 DefinitionObject class

The DefinitionObject class defines an item to be referenced.

The DefinitionObject class is a sub-class of the InterchangeObject class.

The DefinitionObject class is an abstract class.



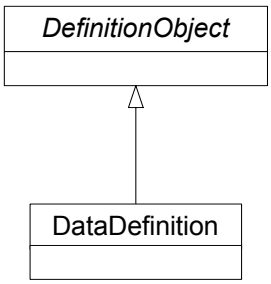
Property Name	Type	Req ?	Meaning
Identification	AUID	Req	Specifies the unique identifier for the item being defined
Name	String	Req	Specifies the display name of the item being defined
Description	String	Opt	Provides an explanation of the use of the item being defined

8.2 DataDefinition class

The DataDefinition class specifies the kind of data that can be stored in a Component.

The DataDefinition class is a sub-class of the DefinitionObject class.

All DataDefinition objects shall be owned by the Dictionary object.



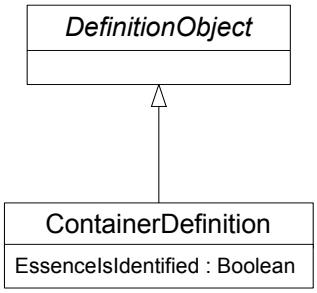
The DataDefinition class does not define any additional properties.

8.3 ContainerDefinition class

The ContainerDefinition class specifies the mechanism used to store essence data. A container can be either a kind of file, such as an AAF file or it can be another mechanism for storing essence data.

The ContainerDefinition class is a sub-class of DefinitionObject.

All ContainerDefinition objects shall be owned by the Dictionary object.



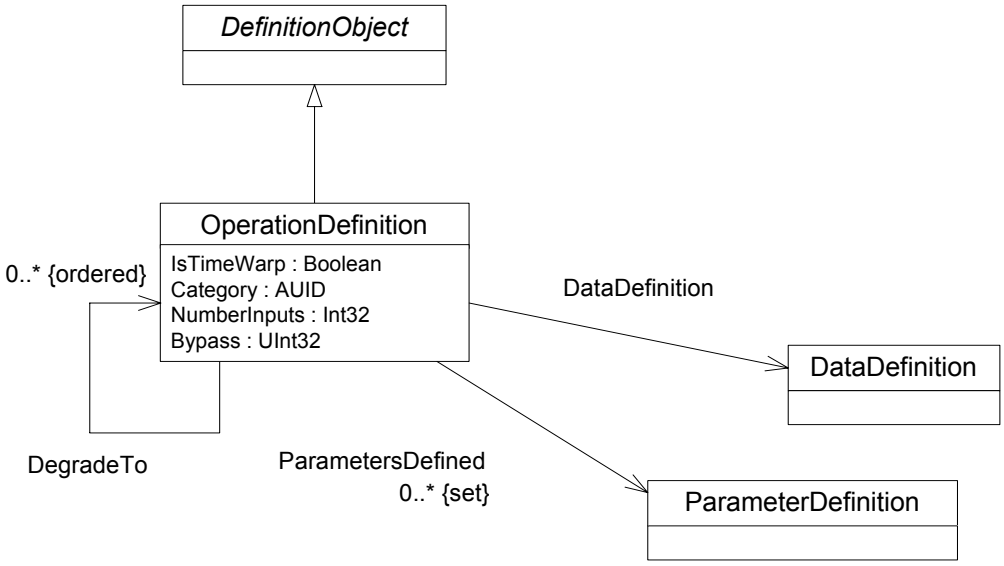
Property Name	Type	Req ?	Meaning
EssenceIsIdentified	Boolean	Opt	Specifies that the container uses the MobID to identify the essence data and that the container may contain multiple essence data objects, each identified by a MobID.

8.4 OperationDefinition class

The OperationDefinition class identifies an operation that is performed on an array of Segments.

The OperationDefinition class is a sub-class of the DefinitionObject class.

All OperationDefinition objects shall be owned by the Dictionary object.



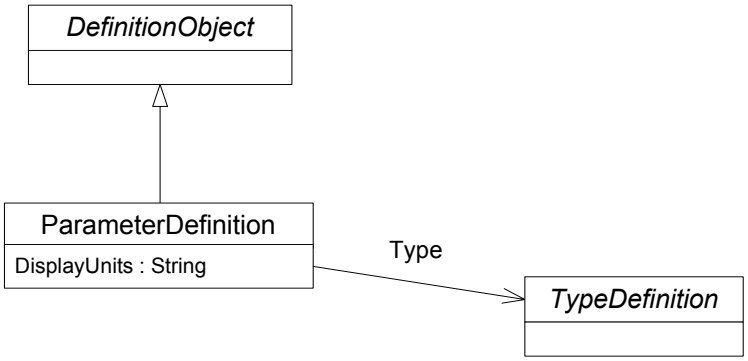
Property Name	Type	Req ?	Meaning	Default
DataDefinition	WeakReference to DataDefinition	Req	Identifies the kind of data that is produced by the operation	
IsTimeWarp	Boolean	Opt	If true, specifies that the duration of the input segments can be different from the duration of the Operation	False
DegradeTo	WeakReferenceVector to OperationDefinition	Opt	Specify simpler operations that an application can substitute for the defined operation if it cannot process it.	
Category	AUID	Opt	Specifies the kind of operation, such as Video Effect, Audio Effect, or 3D operation.	CATEGORY_Effect.
NumberInputs	Int32	Req	Specifies the number of input segments. A value of -1 indicates that the effect can have any number of input segments.	
Bypass	UInt32	Opt	Specifies the array index (1-based) of the input segment which is the primary input	
ParametersDefined	WeakReferenceSet of ParameterDefinition	Opt	Specify the Parameters that can be used as controls for the operation.	

8.5 ParameterDefinition class

The ParameterDefinition class defines a kind of Parameter for an effect.

The ParameterDefinition class is a sub-class of the DefinitionObject class.

All ParameterDefinition objects shall be owned by the Dictionary object.



Property Name	Type	Req ?	Meaning
Type	WeakReference to TypeDefinition	Req	Specifies the data type of the Parameter
DisplayUnits	String	Opt	A displayable string identifying the units in which the parameter is measured. For example: L"% of picture width"

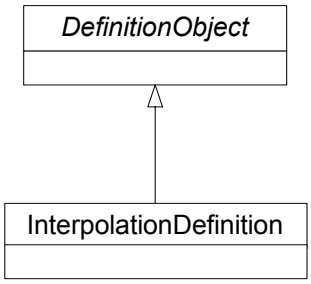
The value of a Parameter is specified in a property with an Indirect type. The Indirect type shall specify the same TypeDefinition as its corresponding ParameterDefinition. The values of Parameters are specified in the ConstantValue Value property and in the ControlPoint Value property. ControlPoints are contained in the VaryingValues sub-class of Parameter.

8.6 InterpolationDefinition class

The InterpolationDefinition class specifies the mechanism used to calculate the values produced by a VaryingValue using the specified ControlPoints.

The InterpolationDefinition class is a sub-class of the DefinitionObject class.

All InterpolationDefinition objects shall be owned by the Dictionary object.



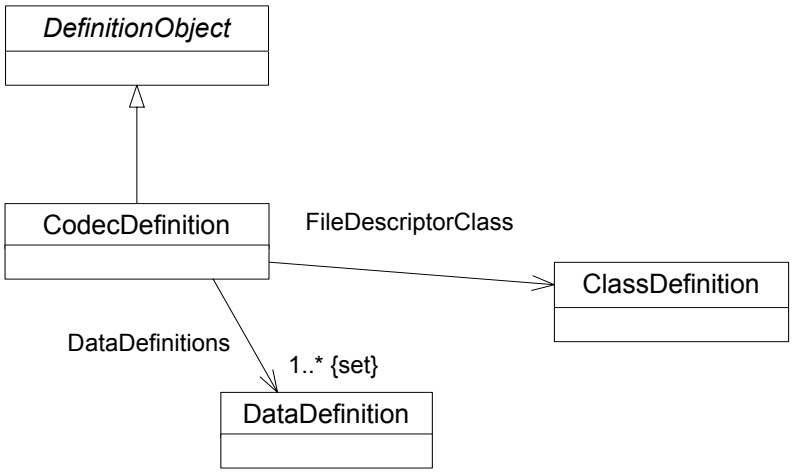
The InterpolationDefinition class does not define any additional properties.

8.7 CodecDefinition class

The CodecDefinition specifies an essence codec.

The CodecDefinition class is a sub-class of the DefinitionObject class.

All CodecDefinition objects shall be owned by the Dictionary object.



Property Name	Type	Req ?	Meaning
FileDescriptorClass	WeakReference to ClassDefinition	Req	Specifies the ClassDefinition of the sub-class of FileDescriptor that identifies the essence format that this codec processes
DataDefinitions	WeakReferenceSet of DataDefinition	Req	Specifies the DataDefinitions of the essence formats that this codec processes

The FileDescriptorClass property identifies the essence format that the Codec can process. For example, a Codec that processes CDCI video data has a FileDescriptorClass that is a weak reference to the ClassDefinition object defining the CDCIDescriptor class. Note that a Codec may not be able to process all variants of essence formats. For example, a hardware accelerated Codec may only be able to process some compressions within CDCI.

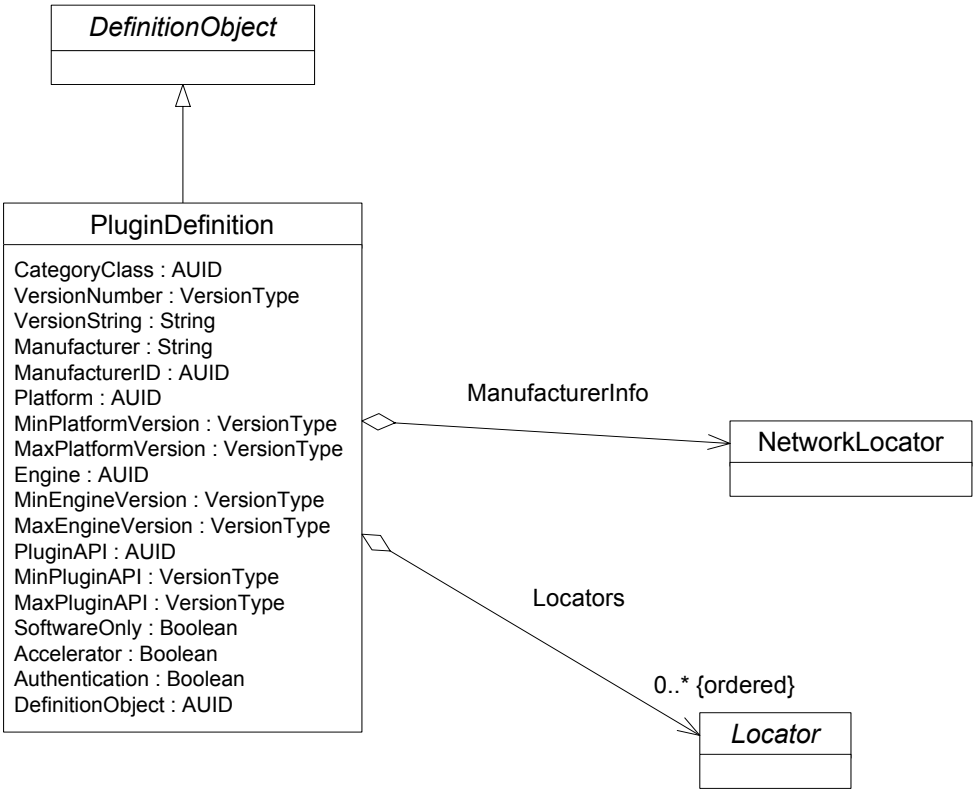
In most cases, a codec processes only one kind of DataDefinition. But some Codecs that process interleaved essence data may be able to handle more than one. For example a Codec that processes MPEG or DV essence may be able to handle both the picture and sound data.

8.8 PluginDefinition class

The PluginDefinition class identifies code objects that provide an implementation for a DefinitionObject, such as CodecDefinition or for a MetaDefinition, such as a ClassDefinition.

The PluginDefinition class is a sub-class of the DefinitionObject class.

All PluginDefinition objects shall be owned by the Dictionary object.



Property Name	Type	Req ?	Meaning	Default
CategoryClass	AUID	Req	Specifies the kind of plugin	
VersionNumber	VersionType	Req	Specifies the version of the plugin	
VersionString	String	Opt	Specifies a string that can be used to identify the plugin version to the user	
Manufacturer	String	Opt	Specifies a string that can be used to identify the plugin manufacturer to the user	
ManufacturerInfo	StrongReference to NetworkLocator	Opt	Specifies a NetworkLocator that identifies a web page containing information about the manufacturer	
ManufacturerID	AUID	Opt	Specifies a SMPTE label or other unique identifier that is assigned to identify the manufacturer	
Platform	AUID	Opt	Identifies the platform environment, which consists of the hardware platform and the operating system, required by the plugin	
MinPlatformVersion	VersionType	Opt	Specifies the minimum version number of the specified platform that the plugin requires	
MaxPlatformVersion	VersionType	Opt	Specifies the maximum version number of the specified platform that the plugin requires	
Engine	AUID	Opt	Identifies the software subsystem used for essence management and playback used by the plugin	
MinEngineVersion	VersionType	Opt	Specifies the minimum version number of the specified engine that the plugin requires	
MaxEngineVersion	VersionType	Opt	Specifies the maximum version number of the specified engine that the plugin requires	

Property Name	Type	Req ?	Meaning	Default
PluginAPI	AUID	Opt	Identifies the plugin interfaces supported by the plugin	
MinPluginAPI	VersionType	Opt	Specifies the minimum version number of the specified plugin interfaces that the plugin supports	
MaxPluginAPI	VersionType	Opt	Specifies the maximum version number of the specified plugin interfaces that the plugin supports	
SoftwareOnly	Boolean	Opt	Specifies if the plugin is capable of executing in a software-only environment	False
Accelerator	Boolean	Opt	Specifies if the plugin is capable of using hardware to accelerate essence processing	False
Locators	StrongReferenceVector of Locator	Opt	Specifies an ordered list of locators that identify locations that provide access to the plugin implementation	
Authentication	Boolean	Opt	Specifies that the plugin implementation supports authentication.	False
DefinitionObject	AUID	Opt – see conditional rule 1	Specifies the AUID of the ClassDefinition for the DefinitionObject or MetaDefinition that it provides an implementation of	

Conditional rule 1 DefinitionObject shall be specified.

Informative note: The Req/Opt status of DefinitionObject is the result of maintaining compatibility with the MetaDictionary used by the AAF reference implementation v1.0.1.

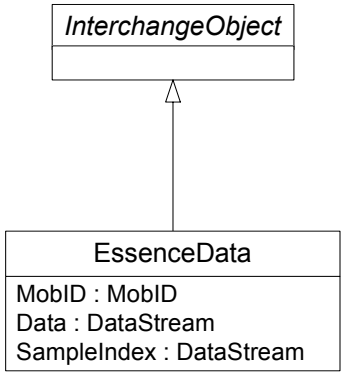
9 EssenceData Classes

9.1 EssenceData class

The EssenceData class contains essence.

The EssenceData class is a sub-class of the InterchangeObject class.

All EssenceData objects shall be owned by the ContentStorage object.



Property Name	Type	Req ?	Meaning
MobID	MobID	Req	Identifies the SourceMob that describes the essence
Data	DataStream	Req	Contains the essence data
SampleIndex	DataStream	Opt	Contains an index to the samples or frames. The format of the index is determined by the Codec

10 EssenceDescriptor Classes

This chapter includes the class specifications for classes in the EssenceDescriptor package. Figure 17 shows the class hierarchy for the EssenceDescriptor package.

The class specification pages are presented in order according to the class hierarchy, reading the Figure left to right, depth-first.

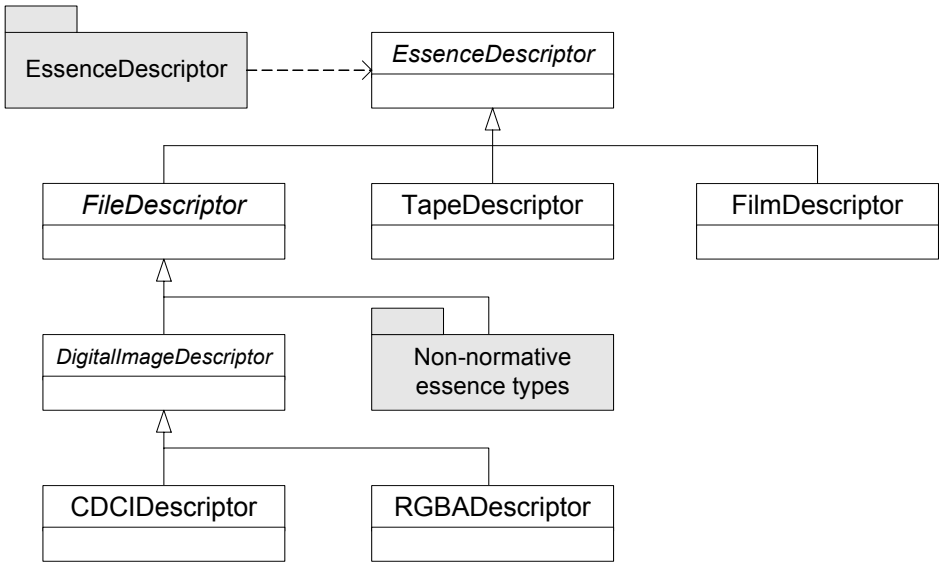


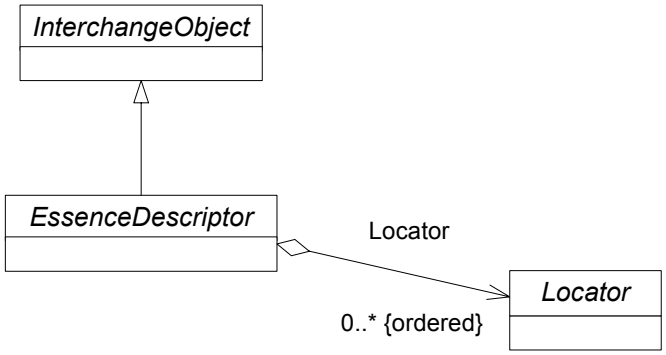
Figure 17 EssenceDescriptor Package

10.1 EssenceDescriptor class

The EssenceDescriptor class describes the format of the essence associated with a file SourceMob or of the media associated with a physical SourceMob.

The EssenceDescriptor class is a sub-class of the InterchangeObject class.

The EssenceDescriptor class is an abstract class.



Property Name	Type	Req ?	Meaning
Locator	StrongReferenceVector of Locator	Opt	Has an array of Locator objects that provide operating-system-dependent data or text information that provide hints for finding files or physical media

Locator objects provide information either for finding files or for finding physical media according to the following rules:

If the object owning the Locators belongs to the FileDescriptor class as well as the EssenceDescriptor class, then the Locators are owned by a file SourceMob and provide information for finding files. A file SourceMob can have any number of Locators and the Locators may belong to any sub-class of Locator.

If the object owning the Locators belongs to the EssenceDescriptor class but not to the FileDescriptor class, then the Locators are owned by a physical SourceMob and provide information for finding physical media. A physical SourceMob can have any number of locators and Locators may belong to any sub-class of Locator.

An EssenceDescriptor may have more than one Locator objects and an EssenceDescriptor may have more than one Locator object of the same sub-class of Locator. For example, a file SourceMob may have more than one Locator to provide hints to find the file on more than one operating system or to provide more than one hint on the same operating system.

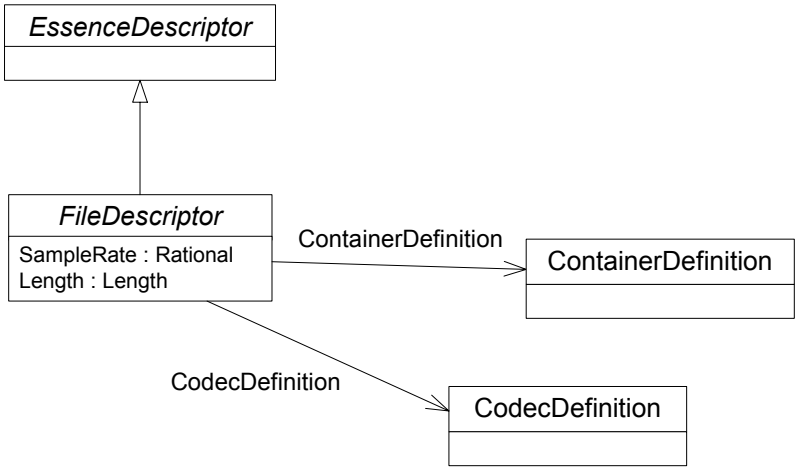
Informative note: Locators in a file SourceMobs provide hints to help find files associated with the file SourceMob, but they are only hints because their correctness cannot be guaranteed, since users may rename or delete files. Typically, this can happen if the AAF file is renamed after being created. If an application cannot find a file by using the hint, it can search through all accessible AAF files to find the EssenceData object with the MobID value.

10.2 FileDescriptor class

The FileDescriptor class describes the essence associated with a file SourceMob.

The FileDescriptor class is a sub-class of the EssenceDescriptor class.

The FileDescriptor class is an abstract class.



Property Name	Type	Req ?	Meaning
SampleRate	Rational	Req – see conditional rule 1	The sample rate of the essence
Length	Length	Req – see	Duration of the essence in sample units

Property Name	Type	Req ?	Meaning
		conditional rule 1	
ContainerFormat	WeakReference to ContainerDefinition	Opt – see conditional rule 2	Identifies the container mechanism used to store the essence
CodecDefinition	WeakReference to CodecDefinition	Opt	Identifies the mechanism used to compress and uncompress samples of essence or used to convert samples of essence from one format to another

Conditional rule 1 FileDescriptors describing static essence shall omit the SampleRate and Length properties. FileDescriptors describing time-varying essence shall specify the SampleRate and Length properties.

Conditional rule 2 ContainerFormat shall be specified.

Informative note: The Req/Opt status of SampleRate, Length and ContainerFormat is the result of maintaining compatibility with the MetaDictionary used by the AAF reference implementation v1.0.1.

The File Descriptor specifies the sample rate and the length in the sample rate of the essence. The sample rate of the data can be different from the edit rate of the Timeline MobSlot in the File SourceMob.

Informative note: In the case of picture essence, the Sample Rate is usually the frame rate. The value should be numerically exact, for example {25,1} or {30000, 1001}.

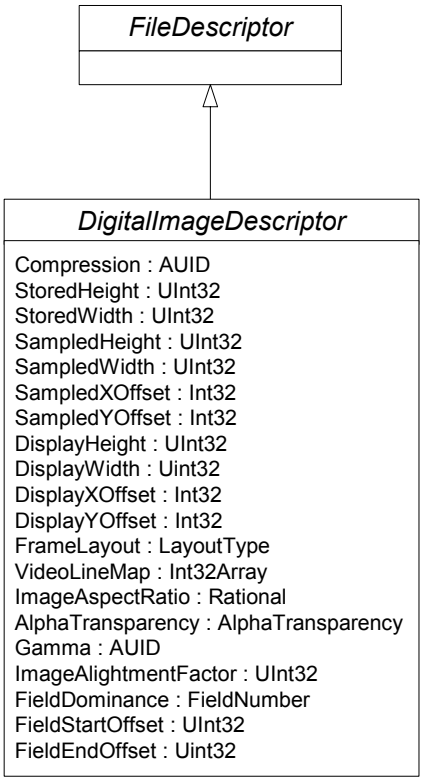
Informative note: Care should be taken if a sample rate of {2997,100} is encountered, since this may have been intended as a (mistaken) approximation to the exact value.

10.3 DigitalImageDescriptor class

The DigitalImageDescriptor class specifies that a File SourceMob is associated with video essence that is formatted either using RGBA or luminance/chrominance formatting.

The DigitalImageDescriptor class is a sub-class of the FileDescriptor class.

The DigitalImageDescriptor class is an abstract class.



Property Name	Type	Req ?	Meaning	Default
Compression	AUID	Opt – see conditional rule 1	Kind of compression and format of compression information.	Un-compressed
StoredHeight	UInt32	Req	Number of pixels in vertical dimension of stored view	
StoredWidth	UInt32	Req	Number of pixels in horizontal dimension of stored view	
SampledHeight	UInt32	Opt – see conditional rule 2	Number of pixels in vertical dimension of sampled view	StoredHeight
SampledWidth	UInt32	Opt – see conditional rule 2	Number of pixels in horizontal dimension of sampled view	StoredWidth
SampledXOffset	Int32	Opt – see conditional rule 2	X offset, in pixels, from top-left corner of stored view	0
SampledYOffset	Int32	Opt – see conditional rule 2	Y offset, in pixels from top-left corner of stored view	0
DisplayHeight	UInt32	Opt – see conditional rule 3	Number of pixels in vertical dimension of display view	StoredHeight
DisplayWidth	UInt32	Opt – see conditional rule 3	Number of pixels in vertical dimension of display view	StoredWidth
DisplayXOffset	Int32	Opt – see conditional	X offset, in pixels, from top-left corner of sampled view	0

Property Name	Type	Req ?	Meaning	Default
		rule 3		
DisplayYOffset	Int32	Opt – see conditional rule 3	Y offset, in pixels, from top-left corner of sampled view	0
FrameLayout	LayoutType	Req	Describes whether all data for a complete sample is in one frame or is split into more than one field	
VideoLineMap	Int32Array	Req	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 interlaced video, there are 2 values in the array.	
ImageAspectRatio	Rational	Req	Describes the ratio between the horizontal size and the vertical size in the intended final image	
AlphaTransparency	AlphaTransparency	Opt	Specifies whether the minimum Alpha value or the maximum Alpha value indicates transparency	minimum
Gamma	AUID	Opt	Specifies the expected output gamma setting on the video display device	
ImageAlignmentFactor	UInt32	Opt	Specifies the alignment when storing the digital essence. 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.	0
FieldDominance	FieldNumber	Opt	Specifies whether field 1 or field 2 is dominant in images composed of two interlaced fields	
FieldStartOffset	UInt32	Opt	Specifies unused bytes at the start of each video field	0
FieldEndOffset	UInt32	Opt	Specifies unused bytes at the end of each video field	0

Conditional rule 1 A DigitalImageDescriptor describing uncompressed essence shall omit the Compression property. A DigitalImageDescriptor describing compressed essence shall specify the Compression property. The Compression property specifies that the image is compressed and the kind of compression used.

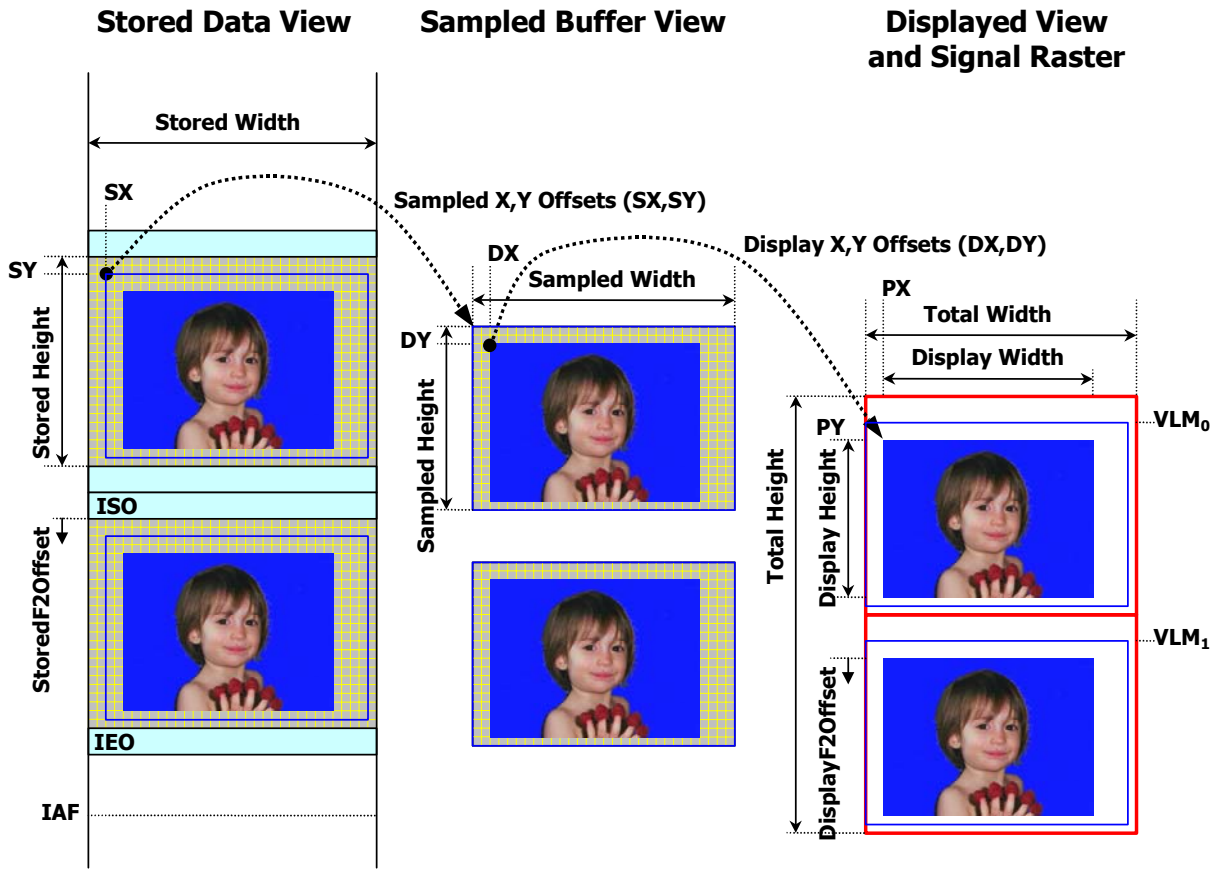
Conditional rule 2 If a DigitalImageDescriptor has any of the sampled geometry properties, SampledHeight, SampledWidth, SampledXOffset, and SampledYOffset, it shall have all of them.

Conditional rule 2 If a DigitalImageDescriptor has any of the display geometry properties, DisplayHeight, DisplayWidth, DisplayXOffset, and DisplayYOffset, it shall have all of them.

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.

The relationships among the views are described below.



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 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 baseband signal and the beginning of the digitized fields. The baseband lines are expressed in scan line numbers that are appropriate for the signal format. For example, a typical 625-line 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 essence has to be treated differently. If the digital essence was computer generated (RGB), the values may be either {0,1} (even field first) or {1,0} (odd field first).

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.

Informative note: In the AAF reference implementation, the FieldDominance, FieldStartOffset and FieldEndOffset properties are implemented by the CDCIDescriptorHelper API, not by the DigitalImageDescriptor API.

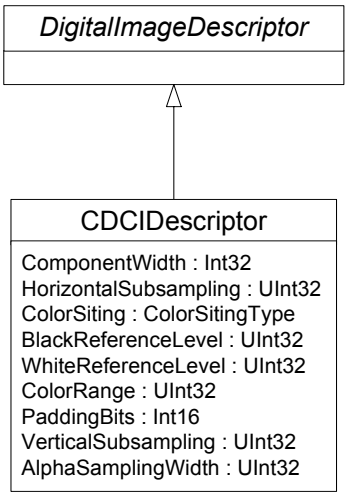
10.4 CDCIDescriptor class

The CDCIDescriptor class specifies that a file SourceMob is associated with video essence formatted with one luminance component and two color-difference components as specified in this document.

Informative note: This format is commonly known as YCbCr.

The CDCIDescriptor class is a sub-class of the DigitalImageDescriptor class.

A CDCIDescriptor object shall be owned by a file SourceMob.



Property Name	Type	Req ?	Meaning	Default
ComponentWidth	UInt32	Req	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 PaddingBits property. If the PaddingBits property is omitted, samples are packed contiguously.	Un-compressed
HorizontalSubsampling	UInt32	Req	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. Legal	

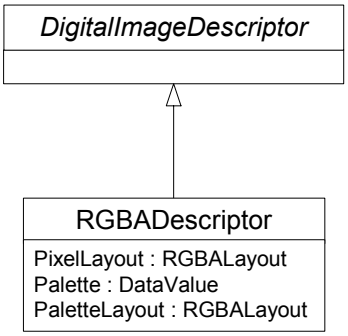
Property Name	Type	Req ?	Meaning	Default
			values are 1, 2 and 4.	
ColorSiting	ColorSitingType	Opt	Specifies color siting	Co-sited
BlackReferenceLevel	UInt32	Opt	Specifies the digital luminance component value associated with black e.g. 16 or 64 (8 or 10-bits)	0
WhiteReferenceLevel	UInt32	Opt	Specifies the digital luminance component value associated with white e.g. 235 or 940 (8 or 10 bits)	Max unsigned integer value for component size
ColorRange	UInt32	Opt	Specifies the range of allowable digital chrominance component values e.g. 225 or 897 (8 or 10 bits)	Max unsigned integer value for component size
PaddingBits	Int16	Opt	Specifies the number of bits padded to each pixel	0
VerticalSubsampling	UInt32	Opt	Specifies the ratio of luminance sampling to chrominance sampling in the vertical direction	1
AlphaSamplingWidth	UInt32	Opt	Specifies the number of bits used to store the Alpha component	0

10.5 RGBADescriptor class

The RGBADescriptor class specifies that a File SourceMob is associated with video essence formatted with three color component or with three color components and an alpha component as specified in this document.

The RGBADescriptor class is a sub-class of the DigitalImageDescriptor class.

An RGBADescriptor object shall be owned by a file SourceMob.



Property Name	Type	Req ?	Meaning
PixelLayout	RGBALayout	Req	Specifies the order and size of the components within the pixel
Palette	DataValue	Opt – see conditional rule 1	An array of color values that are used to specify an image
PaletteLayout	RGBALayout	Opt – see conditional rule 1	An array of RGBAComponent that specifies the order and size of the color components as they are stored in the palette

Conditional rule 1 If the PixelLayout property includes a ‘P’, then the RGBADescriptor object shall have the Palette and PaletteLayout properties.

If the PixelLayout property includes an ‘R’, ‘G’, or ‘B’, then it shall not include a ‘P’. If the PixelLayout property includes a ‘P’, then it shall not include an ‘R’, ‘G’, or ‘B’.

An RGBADescriptor object describes essence that contains component-based images where each pixel is made up of a red, a green and a blue value. Optionally, an alpha value can be included in each pixel. The alpha value determines the transparency of the color. Each pixel can be described directly with a component value or a by an index into a pixel palette.

Properties in the RGBADescriptor 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.

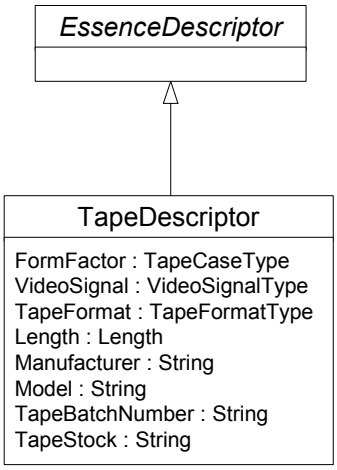
If a color palette is used, the descriptor allows you to specify the color palette and the structure used to store each color in the palette.

10.6 TapeDescriptor class

The TapeDescriptor class describes audio tape or video tape media.

The TapeDescriptor class is a sub-class of the EssenceDescriptor class.

A TapeDescriptor object shall be owned by a physical SourceMob.



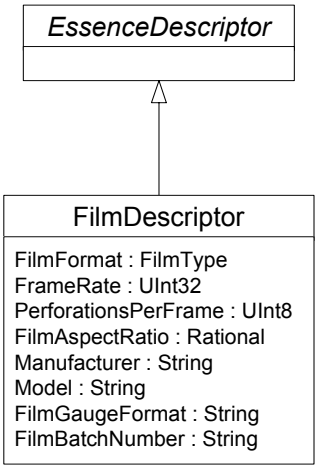
Property Name	Type	Req ?	Meaning
FormFactor	TapeCaseType	Opt	Describes the physical size of the tape
VideoSignal	VideoSignalType	Opt	Describes the video signal type
TapeFormat	TapeFormatType	Opt	Describes the format of the tape
Length	UInt32	Opt	Tape capacity in minutes
Manufacturer	String	Opt	Text string to display to end users, identifying the manufacturer of the tape
Model	String	Opt	Text string to display to end users, identifying the manufacturer’s brand designation of the tape
TapeBatchNumber	String	Opt	Specifies the batch number of the tape
TapeStock	String	Opt	Specifies the string identifying the tape stock

10.7 FilmDescriptor class

The FilmDescriptor class describes film media.

The FilmDescriptor class is a sub-class of the EssenceDescriptor class.

A FilmDescriptor object shall be owned by a physical SourceMob.



Property Name	Type	Req ?	Meaning
FilmFormat	FilmType	Opt	Identifies the format of the film
FrameRate	UInt32	Opt	Specifies the frame rate in frames per second
PerforationsPerFrame	UInt8	Opt	Specifies the number of perforations per frame on the film stock
FilmAspectRatio	Rational	Opt	Ratio between the horizontal size of the frame image and the vertical size of the frame image
Manufacturer	String	Opt	A string to display to end users, indicating the manufacturer of the film
Model	String	Opt	A string to display to end users, indicating the manufacturer's brand designation for the film, such as "5247"
FilmGaugeFormat	String	Opt	Specifies the film gauge format
FilmBatchNumber	String	Opt	Specifies the batch number of the tape

11 Non-normative Essence Types

This chapter includes the class specifications for classes in the Non-normative Essence Types package. (Non-normative essence types are defined by industry specifications, and are not described by international standards). Figure 18 shows the class hierarchy for the Non-normative Essence Types package.

The class specification pages are presented in order according to the class hierarchy, reading the Figure left to right, depth-first.

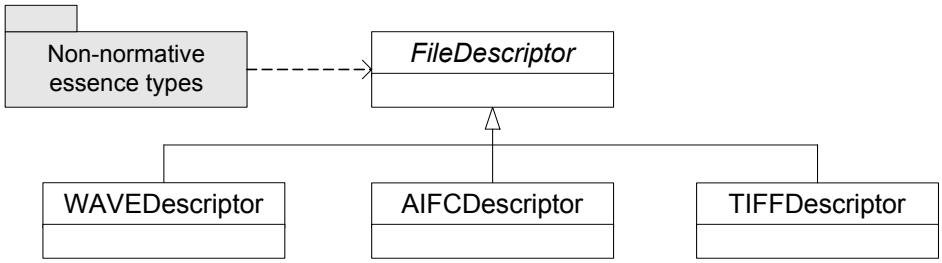


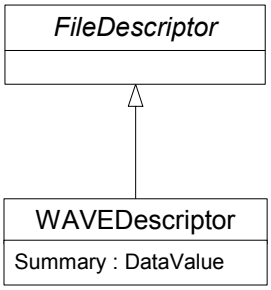
Figure 18 Non-normative essence types Package

11.1 WAVEDescriptor class

The WAVEDescriptor class specifies that a File SourceMob is associated with audio essence formatted according to the RIFF Waveform Audio File Format (WAVE).

The WAVEDescriptor class is a sub-class of the FileDescriptor class.

A WAVEDescriptor object shall be owned by a file SourceMob.



Property Name	Type	Req ?	Meaning
Summary	DataValue	Req	A copy of the WAVE file information without the sample data

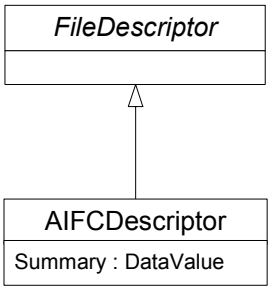
A WAVEDescriptor describes a WAVE file containing digitized audio data in little-endian byte order. It contains data formatted according to the Microsoft/IBM Multimedia Programming Interface and Data Specifications, Version 1.0, but limited to the section describing the RIFF Waveform Audio File Format 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.

11.2 AIFCDescriptor class

The WAVEDescriptor class specifies that a File SourceMob is associated with audio essence formatted according to the Audio Interchange File Format with Compression (AIFC).

The AIFCDescriptor class is a sub-class of the FileDescriptor class.

An AIFCDescriptor object shall be owned by a file SourceMob.



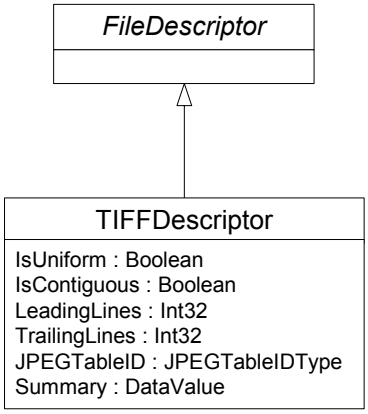
Property Name	Type	Req ?	Meaning
Summary	DataValue	Req	A copy of the descriptive information in the associated AIFC Audio Data value

11.3 TIFFDescriptor class (optional)

The TIFFDescriptor class specifies that a File SourceMob is associated with video essence formatted according to the TIFF specification.

The TIFFDescriptor class is a sub-class of the FileDescriptor class.

A TIFFDescriptor object shall be owned by a file SourceMob.



Property Name	Type	Req ?	Meaning	Default
IsUniform	Boolean	Req	True for data having the same number of rows per strip throughout	
IsContiguous	Boolean	Req	True for data stored in contiguous bytes	
LeadingLines	Int32	Opt	Number of leading lines to be thrown away	0
TrailingLines	Int32	Opt	Number of trailing lines to be thrown away	0
JPEGTableID	JPEGTableIDType	Opt	Registered JPEG table code or JT_NULL	
Summary	DataValue	Req	A copy of the TIFF IFD (without the sample data)	

12 Compressed Picture Essence Types

This chapter is reserved for future Essence Descriptors.

13 Sound Essence Types

This chapter is reserved for future Essence Descriptors.

14 Multiplexed Essence Types

This chapter is reserved for future Essence Descriptors.

15 Reserved

This chapter is reserved for future Class definitions.

16 Reserved

This chapter is reserved for future Class definitions.

17 Reserved

This chapter is reserved for future Class definitions.

18 Reserved

This chapter is reserved for future Class definitions.

19 Reserved

This chapter is reserved for future Class definitions.

20 Reserved

This chapter is reserved for future Class definitions.

21 Built-In Types

This chapter specifies the Types which are built in to the AAF reference implementation, including the controlled sets of enumerated values.

The Type specifies the data type of property values and of parameters.

The Type is identified by a globally unique integer. The following table lists the data types by name.

21.1 Basic and Structured Types

Data Type	Meaning
AUID	A 16-byte unique identifier whose value is a SMPTE 298M Universal Label or a UUID or GUID
AUIDArray	Array of 16-byte unique identifiers
Boolean	Specifies either True or False.
Char	Specifies a single character value.
DataValue	Specifies essence or a block of data whose type is specified by a data kind.
Int8	Specifies an 8-bit 2's complement integer value.
Int8Array	Specifies an array of Int8 values.
Int16	Specifies a 16-bit 2's complement integer value.
Int16Array	Specifies an array of Int16 values.
Int32	Specifies a 32-bit 2's complement integer value.
Int32Array	Specifies an array of Int32 values.
Int64	Specifies a 64-bit 2's complement integer value.
Int64Array	Specifies an array Int64 values.
JPEGTableIDType	Specifies the JPEG tables used in compressing TIFF data.
Length	Specifies the length of a Component with an Int64.
MobID	Specifies a 32-byte unique identifier that can hold a SMPTE UMID.
PixelRectangle	Specifies of Rectangle in pixels. Is Record with the following elements: Horizontal: UInt16 Vertical: UInt16
Position	Specifies an offset into a Component with an Int64.
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.
Rational	Specifies a rational number by means of an Int32 numerator and an Int32 denominator.
RationalRectangle	Specifies an area within an image with 4 rationals, where the first two rationals specify the horizontal and vertical position of the upper-left corner of the rectangle and the last two rationals specify the horizontal and vertical position of the lower-right corner. The position of the center of the image is defined as (0/1, 0/1) (rounding up and to the left); the upper left pixel of the image is (-1/1, -1/1); and the lower-right pixel of the image is (1/1, 1/1).
RGBAComponent	Specifies a component within an RGBA pixel. Is a record with the following fields: Field Type Explanation Code RGBAComponentKind Enumerated value specifying component Size UInt8 Integer specifying the number of bits
String	Specifies a string of Unicode characters.
StrongReference	Specifies an owned object, which is logically contained by the owning object.

Data Type	Meaning
StrongReferenceVector	Specifies an ordered set of owned objects.
StrongReferenceSet	Specifies an unordered set of owned uniquely identified objects.

TimeStamp	Specifies a date and Universal Time Code using the following structure:
	Type TimeStamp Record {
	Type Date Record {
	Type Year Int16
	Type Month UInt8
	Type Day UInt8
	}
	Type Time Record {
	Type Hour UInt8
	Type Minute UInt8
	Type Second UInt8
	Type Fraction UInt8
	}
	}
	Where Fraction is expressed in 1/100 of a second.

UInt8	Specifies an unsigned 8-bit integer value.
UInt8Array	Specifies an array of unsigned 8-bit integer value.
UInt16	Specifies an unsigned 16-bit integer value.
UInt16Array	Specifies an array of unsigned 16-bit integer value.
UInt32	Specifies an unsigned 32-bit integer value.
UInt32Array	Specifies an array of unsigned 64-bit integer values.
UInt64	Specifies an unsigned 64-bit integer value.
UInt64Array	Specifies an array of 32-bit integer values.
VersionType	Specifies a 2-byte unsigned version number.
WeakReference	Reference to an object that defines a unique identifier
WeakReferenceVector	Reference to an ordered set of uniquely identified objects
WeakReferenceSet	Reference to an unordered set of uniquely identified objects

21.2 Enumerated Types

Data Type	Meaning
ColorSitingType	Specifies how to compute subsampled values as an enumerated UInt8. Values are 0 coSiting The first luminance value of the image is co-sited with the first chrominance value. 1 averaging The color pixel is sited at the point horizontally midway between the luminance pixels on each line. 2 threeTap Reserved.
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

Data Type	Meaning
	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.
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).
EdgeType	Specifies the kind of film edge code as an enumerated UInt8. Values are: 0 ET_NULL Invalid edge code 1 ET_KEYCODE Eastman Kodak KEYKODE TM format. 2 ET_EDGENUM4 edge code format: nnnn+nn. 3 ET_EDGENUM5 edge code format: nnnnn+nn.
EditHintType	Specifies hints to be used when editing Control Points as an enumerated UInt8. Values are: 0 EH_NoEditHint 1 EH_Proportional 2 EH_RelativeLeft 3 EH_RelativeRight 4 EH_RelativeFixed
FadeType	Specifies the type of the audio fade as an enumerated UInt8; may have one of the following values: 0 No fade 1 Linear amplitude fade 2 Linear power fade Additional registered and private fade types may be defined.
FilmType	Specifies the format of the film as an enumerated UInt8. 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
LayoutType	Describes whether all data for a complete sample is in one frame or is split into more than one field as an enumerated UInt8. Values are: 0 FULL_FRAME: frame consists of a full sample in progressive scan lines 1 SEPARATE_FIELDS: sample consists of two fields, which when interlaced produce a full sample 2 ONE_FIELD: sample consists of two interlaced fields, but only one field is stored in the data stream 3 MIXED_FIELDS
ProductVersion	Specifies the version number of an application. Consists of 4 UInt16 integer values followed by a UInt8. The first four integers specify the major, minor, tertiary and patch level version numbers. The fifth integer has the following values: 0 kVersionUnknown No additional version information 1 kVersionReleased Released product 2 kVersionDebug Development version 3 kVersionPathched Released version with patches 4 kVersionBeta Prerelease beta test version 5 kVersionPrivateBuild Version not intended for general release
PulldownKindType	Specifies whether the Pulldown object is converting from nominally 30 Hz or 25 Hz video frame rate and whether frames are dropped or the video is played at another speed as an enumerated UInt8. Values are: 0 kTwoThreePD Converting between nominally 30 Hz video frame rate and film frame rate by dropping or adding frames 1 kPalPD Converting between 25 Hz video frame rate and film frame rate by dropping or adding frames 2 kOneToOneNTSC Converting between nominally 30 Hz video frame rate and film frame rate by speeding up or slowing down the frame rate. 3 kOneToOnePAL Converting between 25 Hz video frame rate and film frame rate by speeding up or slowing

Data Type	Meaning
	down the frame rate. 4 kVideoTapNTSC Converting between video frame rate and film frame rate by recording original film and video sources simultaneously. **Informative note:** 3 kOneToOnePAL and 4 kVideoTapNTSC are not implemented in the AAF reference implementation MetaDictionary
PulldownDirectionType	Specifies whether the Pulldown object is converting from tape to film speed or from film to tape speed as an enumerated UInt8. Values are: 0 kTapeToFilmSpeed The InputSegment is at video frame rate and the Mob track owning the Pulldown object is at film frame rate. 1 kFilmToTapeSpeed The InputSegment is at film frame rate and the Mob track owning the Pulldown object is at video frame rate.
RGBAComponentKind	Enumerated type that specifies the color or function of a component within a pixel as an enumerated UInt8. May have the following values: 0x52 'R' Red component 0x47 'G' Green component 0x42 'B' Blue component 0x41 'A' Alpha component 0x46 'F' Fill component 0x50 'P' Palette code 0x00' Terminates list of components
RGBALayout	Specifies the order and size of the components within the pixel. The RGBALayout type is a fixed-size 8 element array, where each element consists of the RGBAComponent type with the following fields: Code Enumerated value specifying component Size UInt8 integer specifying the number of bits A Fill component indicates unused bits. After the components have been specified, the remaining Code and Size fields should be set to 0.
TapeCaseType	Describes the physical size of the tape as an enumerated UInt8; may have one of the following values: 0 None 1 3/4 inch videotape 2 VHS video tape 3 8mm videotape 4 Betacam videotape 5 Compact cassette 6 DAT cartridge 7 Professional audio tape (e.g. Nagra)
TapeFormatType	Describes the format of the tape as an enumerated UInt8; may have one of the following values: 0 None 1 Betacam 2 BetacamSP 3 VHS 4 S-VHS 5 8mm 6 Hi8
TCSource	Specifies the kind of timecode as an enumerated UInt8; may have one of the following values: 0 LTC timecode 1 VITC timecode
VideoSignalType	Specifies the type of video signal on the videotape as an enumerated UInt8. Values are: 0 None 1 NTSC

Data Type	Meaning
	2 PAL 3 SECAM **Informative note:** 3 SECAM is not implemented in the AAF reference implementation MetaDictionary

22 Built-In Data Definitions

This chapter specifies the Data Definitions which are built in to the AAF reference implementation.

22.1 Built-In Data Definitions

Data Kind	Meaning
Edgecode	Specifies a stream of film edge code values.
Matte	Specifies a stream of essence that contains an image of alpha values.
Picture	Specifies a stream of essence that contains image data.
PictureWithMatte	Specifies a stream of essence that contains image data and a matte.
Sound	Specifies a stream of essence that contains a single channel of sound.
Timecode	Specifies a stream of tape timecode values.

23 Built-In Extensible Enumerations

This chapter is reserved for Extensible Enumerations.

24 Built-In OperationDefinitions

This chapter is reserved for OperationDefinitions.

25 Tutorial on Compositions

This chapter describes the AAF Composition Mob, which is the AAF class that describes editing information.

25.1 Composition Mob Basics

Composition Mobs describe the creative editing and composing decisions that combine the individual pieces of essence data into a unified program. A Composition Mob can describe editing decisions that vary in complexity from very simple compositions, which combine a few pieces of essence in order, to very complex compositions that have complex, layered effects and thousands of individual pieces of essence that are combined in various ways. Composition Mobs are designed to be capable of describing creative human decisions; consequently, their complexity is limited only by the limits of our imagination.

Composition Mobs do not directly reference the essence data that they combine to form a program. Composition Mobs reference the basic essence data with SourceClips that identify the MasterMob and File SourceMobs that describe the essence data. The MasterMobs and File SourceMobs have the information that is used to read and write the essence data.

In addition to SourceClips, Composition Mobs can have Sequences, Effects, Transitions, and other objects that combine or modify the basic essence data to produce the elements of essence data that go into the final program. The essence data that results from these transformations can be stored in the file, but typically is generated by the application from the basic essence data and is not stored until the distribution media is generated from the Composition Mob.

Composition Mobs consist of Slots that describe a set of editing decisions that can be referenced from outside of the Mob. Slots in a Composition Mob typically describe sets of editing decisions that are combined in some form to produce the final program.

Slots can describe timeline essence data, such as audio and video, static essence data, such as static images, and other kinds of data, such as text or events.

A Composition Mob can have Slots that all describe timeline essence data, that all describe static essence data, or that describe different kinds of essence data.

A simple Composition Mob could have two TimelineMobSlots describing audio data and one TimelineMobSlot describing video data. The edited program produced by this Composition Mob would consist of three synchronized tracks: two audio and one video.

Another simple Composition Mob could have one StaticMobSlot , describing a single static image composed by combining a set of static images.

A complex Composition Mob could have TimelineMobSlots, StaticMobSlots, and EventMobSlots. The edited program produced by this Composition Mob could have elements from each of these Slots combined in some form by the objects in the Composition Mob.

25.2 TimelineMobSlots

TimelineMobSlots typically have a Sequence of audio or video segments. Each segment can consist of a simple SourceClip or a complex hierarchy of Effects. Figure 19 below is a containment diagram of a Composition Mob that has only TimelineMobSlots with audio and video data.

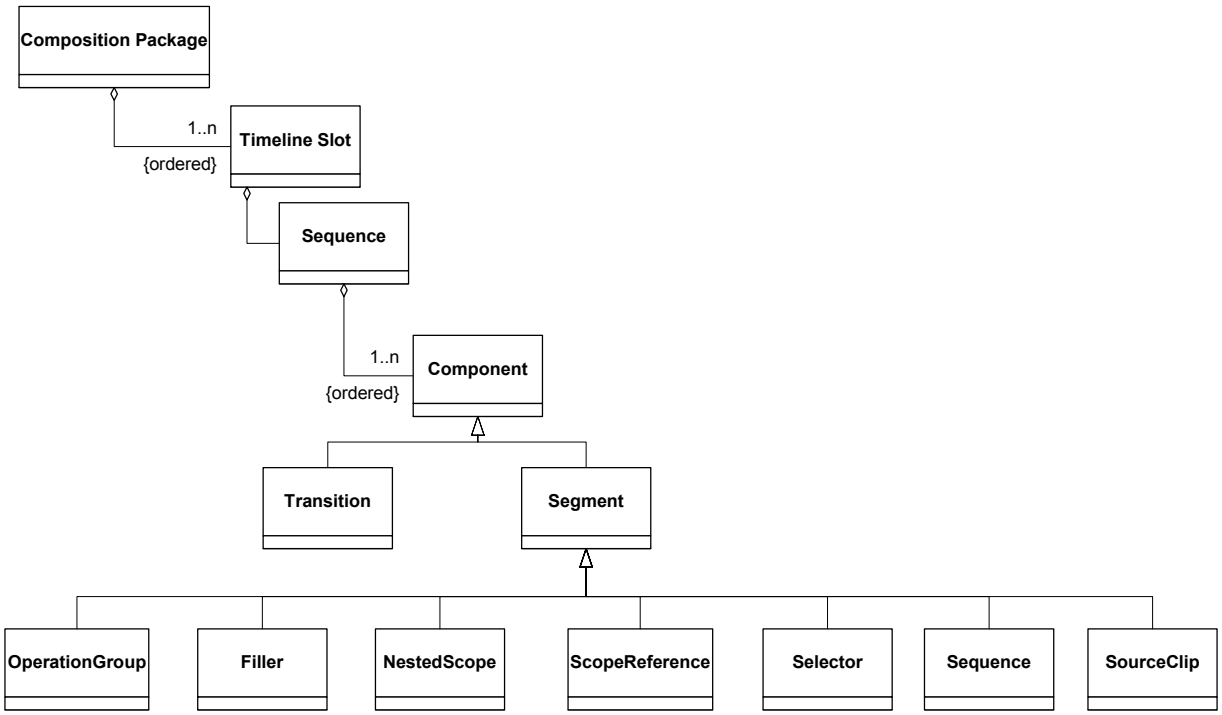


Figure 19 Containment Diagram for a CompositionMob with TimelineMobSlots

25.3 Sequences

A Sequence can have the following components:

- SourceClip: Specifies a section of essence or other time-varying data and identifies the Slot in another Mob or within the same Mob that describes the essence.
- 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 Slots and Nested Scope Segments that are not referenced or played.
- Transition: Causes two adjacent Segments to overlap in time and to be combined by an effect.
- Operation Group: Specifies an effect to be used in a Composition Mob; specifies kind of effect, input essence segments, and control arguments.
- Sequence: A Sequence within a Sequence combines a set of Components into a single segment, which is then treated as a unit in the outer Sequence.
- 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 a 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 effect 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.

The Sequence object combines a series of timeline Components in sequential order. If the Sequence has only Segments, each Segment is played sequentially after the Segment that precedes it. The time in the Composition Mob that a Segment starts is determined by the Components that precede it in the Sequence.

25.4 Transitions

A Transition can occur in a Sequence between two Segments. The Transition causes the preceding and following Segments to overlap in time. The Transition specifies an effect that is used to combine the overlapping Segments. Figure 20 below illustrates the Sequence containment showing Transition, which itself has an Effect.

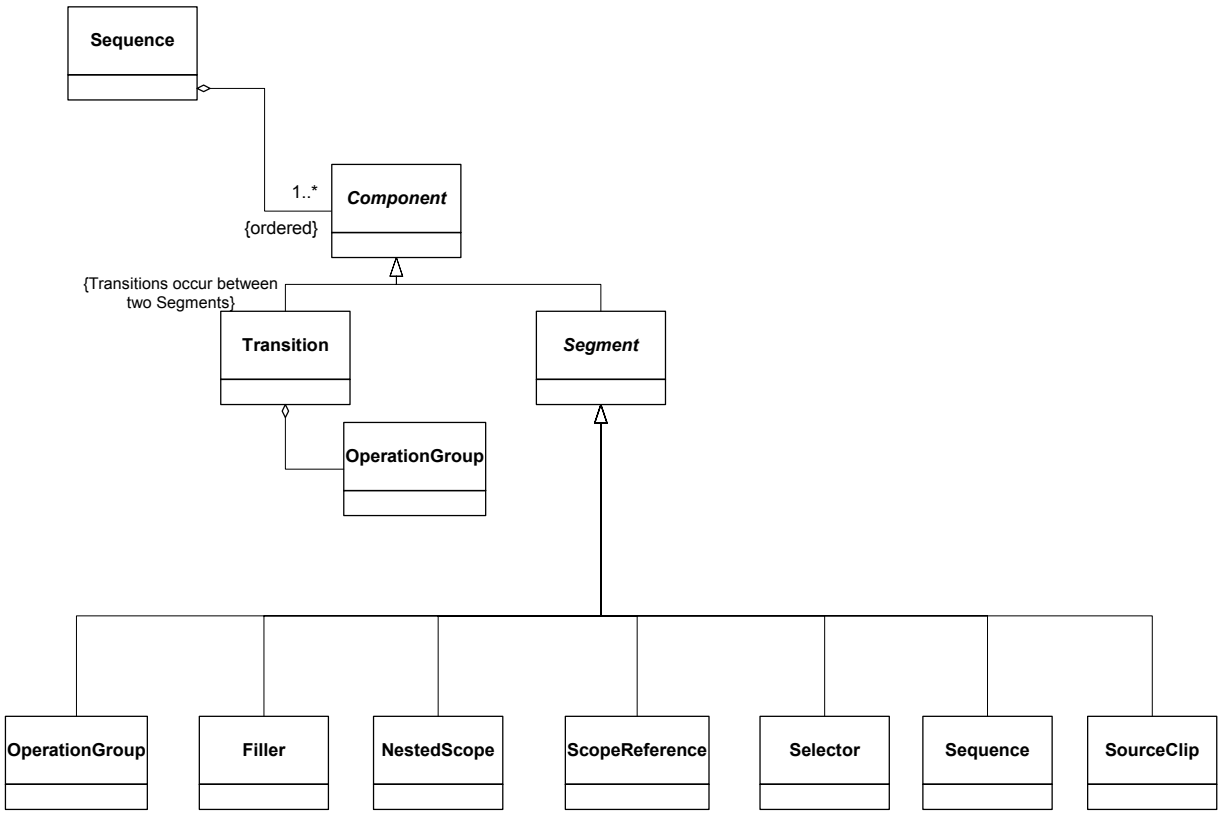
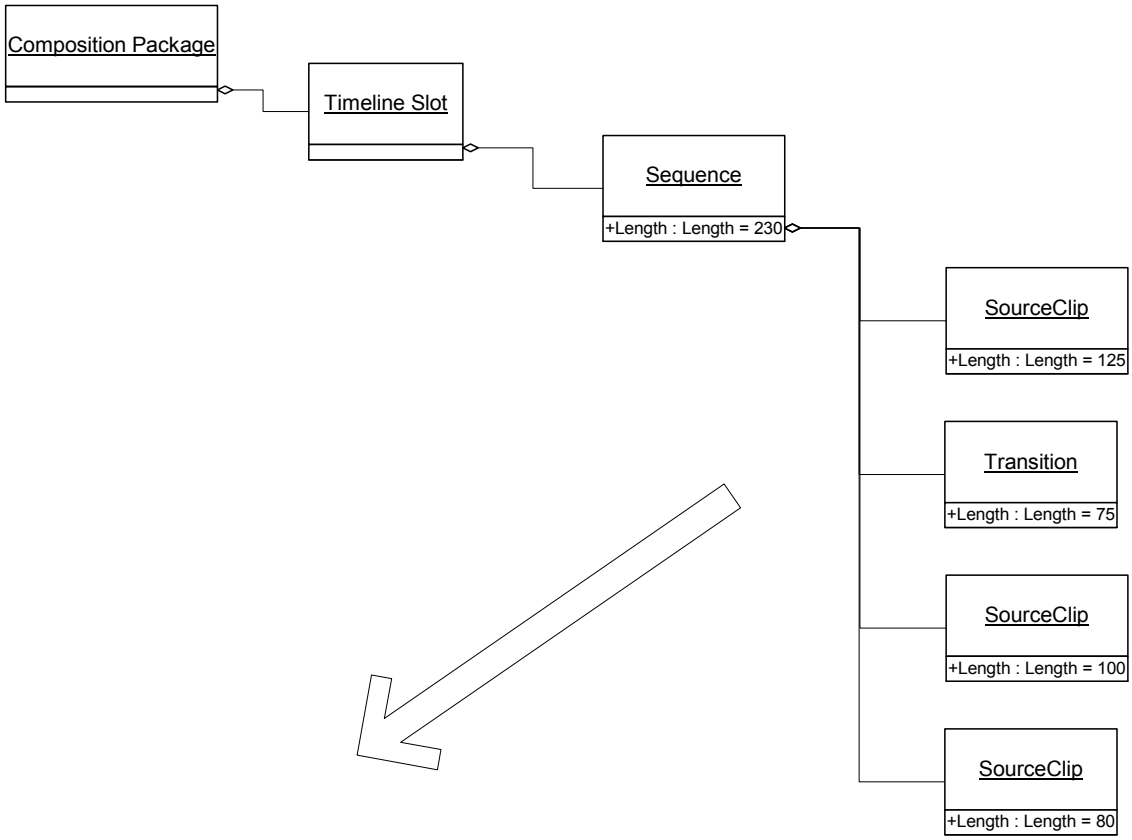
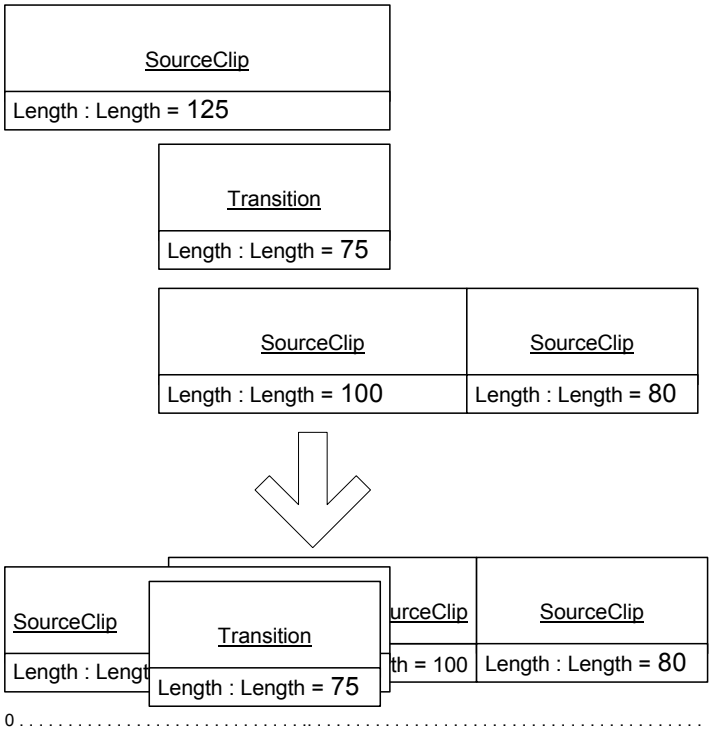


Figure 20 Containment Diagram for a Sequence with a Transition

Figure 21 below shows an instance diagram of a Sequence containing SourceClips and a Transition. It shows the timeline view of the Sequence, in which the Transitions cause the two SourceClips to overlap.





Timeline View

Figure 21 Timeline View of Overlapping Segments of a Transition

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 the above Figure, the duration of the Sequence is 125 + 100 + 80 — 75, which equals 230.

If you are inserting a Transition between two SourceClips, and you want to preserve the overall duration of the two Segments, you must adjust the SourceClip's Length and StartTime values.

25.4.1 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 the following one begins, 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.

25.4.2 Treating Transitions As Cuts

If you cannot play a Transition's effect, you should treat it as a cut. Treating it 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.

25.4.3 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 to avoid ambiguous constructions.

It is possible to create Sequences that appear to start or end with Transitions or that appear to have 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.

25.5 StaticMobSlots

StaticMobSlots describe Essence data that has no relationship to time. Consequently, StaticMobSlots do not specify an edit rate and Segments in StaticMobSlots do not have a duration. Figure 22 below is a containment diagram for a Composition Mob that has only StaticMobSlots.

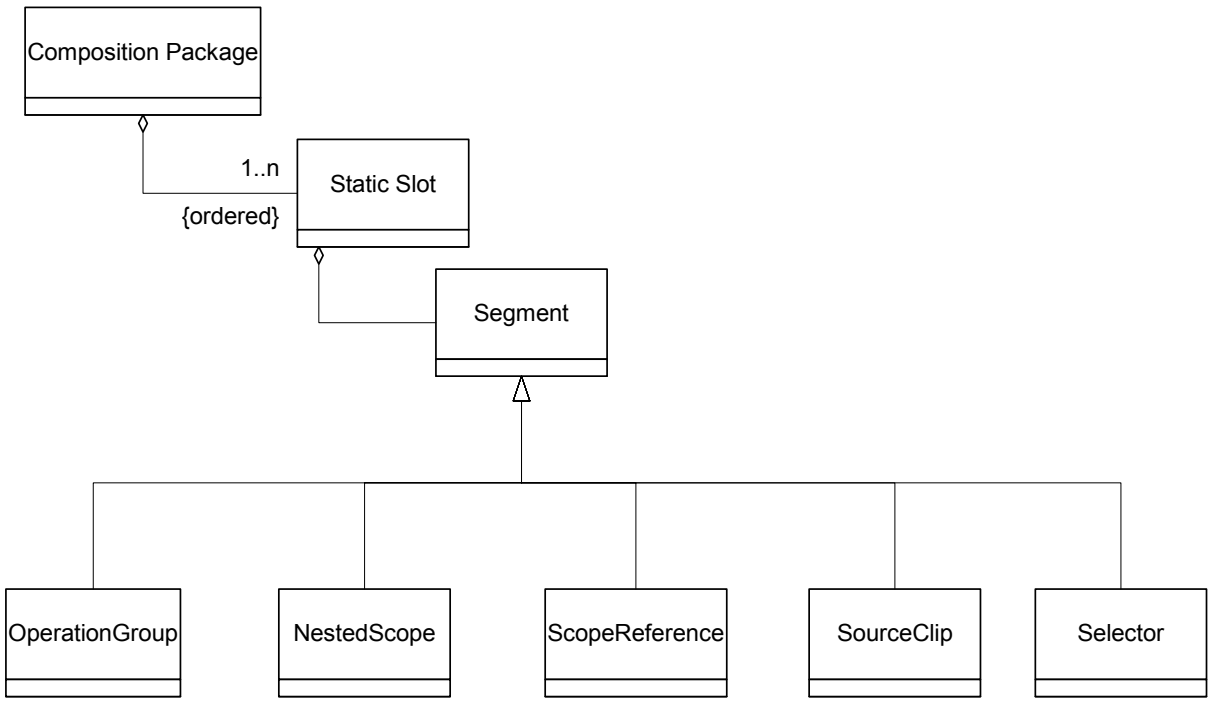


Figure 22 Containment Diagram of a CompositionMob with StaticMobSlots

25.6 Combining Different Types of Slots

A Composition Mob can have TimelineMobSlots, StaticMobSlots, and EventMobSlots. Although each kind of slot can only have Segments with the corresponding relationship to time (which is enforced by the presence or absence of Length and Position properties), it is possible for a Slot to have a reference to another kind of Slot. For example, a video TimelineMobSlot can have a reference to an image in a StaticMobSlot.

A Slot can reference a different kind of Slot by containing a SourceClip referencing the other Slot or by containing an Effect with a SourceClip referencing the other Slot. The SourceClip can reference Slots in other Mobs or can reference other Slots in the same Mob.

25.6.1 Conversion Operations

The SourceClip provides the conversion operation in some simple cases:

- Taking an instantaneous value (such as a still frame) from a Timeline Component.
- Repeating a Static Segment to create a Timeline Segment.

In these cases, the Data Kind of the two Segments must be the same. In all other cases, an explicit operation is required. The Operation Definition must explicitly allow inputs of the appropriate temporal nature and produce a result of the required temporal nature. Conversion operations are summarized in Table below.

Convert to:	Static	Event	Timeline

Convert from:			
Static		SourceClip plus Operation	SourceClip (Start Time ignored)
Event	SourceClip plus Operation		SourceClip plus Operation
Timeline	SourceClip (Length ignored)	SourceClip plus Operation	

25.7 Operations

Essence operation effects (such as transitions or chroma-key effects) can be used to modify or transform the essence to produce a Segment of essence. Operations can act on and produce any kind of essence: timeline, static, or event. The essence that an effect acts on is called its input essence. These effects use the same binary plug-in model used to support codecs, essence handlers, or other digital processes to be used to process the essence to create the desired impact. The binary plug-in model gives applications the flexibility to determine when a given effect or codec has been referenced inside of the file and to determine if that effect or codec is available, and if not, to find it and load it on demand.

Many common effects act on timeline or static essence and produce the same kind of essence as they act on. For example, a picture-in-picture effect can act on either timeline video or static image essence. It combines two input essence Segments to produce a resulting Segment. A picture-in-picture effect with timeline video input essence Segments produces a timeline video result. A picture-in-picture effect with static image input essence Segments produces a static image result. There are also effects than convert from one kind of essence to another.

A specific usage of an effect in a file is described by an OperationGroup object. The OperationGroup that produces a segment is made up of the following:

- Has an ordered set of input essence Segments.
- Is associated with an OperationDefinition object.
- Has a set of effect control parameters.
- May optionally have a rendered version of the Operation.

25.7.1 Effect Input Essence Segments

Most common effects have either one or two input essence Segments. Some effects can have more than two input essence Segments, and there are effects that have no input essence Segments.

25.7.2 Filter Effects with One Input Essence Segment

An effect that has one input essence Segment is often called a filter effect because it takes its input essence Segment, modifies it by passing it through some kind of filter, and then produces a resulting essence Segment. Some example picture filter effects are a blur effect or a color correction effect. An example audio filter effect is a gain effect.

If an application cannot generate a filter effect, it can usually substitute the input essence Segment for the effect result and have a meaningful, if not completely accurate, output. You cannot substitute the input essence for time-warp effects. Time-warp effects are timeline essence effects where the duration of the input essence Segment is different from the duration of the effect result. Slow motion and freeze-frame effects are examples of time-warp effects.

25.7.3 Effects with Two Input Essence Segments

Effects with two input essence Segments combine the Segments to produce a single resulting Segment. For example, a picture-in-picture or a superimpose effect takes two images and combines them to produce a single image.

A transition effect is a timeline effect with two input essence Segments that are intended to change from one input essence Segment to another. Examples of transition effects are wipes and audio crossfades. For more information about effects in transitions, see the Transition Effects section in this chapter.

Some effects can have any number (greater than zero) of Segments. These effects typically are used to add together a number of essence Segments. For example, the audio mixdown effect takes any number of audio Segments and adds them to produce a combined audio Segment. Another example is the image ordering effect that takes a set of pictures (static or timeline) and combines them by placing one in front of another in the order in which they are specified.

25.7.4 Effect Definitions

Effects are identified by a AUID, a unique identifier. The file also contains an EffectDefinition object that provides additional information about the effect. It identifies the effect it is defining with a AUID and includes the following additional information:

- Effect name and description for display purposes
- Number of essence input segments
- Control code definitions that define the effect's parameters
- Information to find plug-in code to process the effect

25.7.5 Effect Control Parameters

Effect controls are contained in a set of Parameters. Each Parameter identifies the purpose of the parameter by specifying a parameter AUID and specifies either a single constant or a varying value. The Effect Definition lists the parameters that can be specified for the Effect.

A constant value is specified by an ConstantValue object, which has a single value. For timeline effects, this means the value is constant over time.

For timeline effects, a varying value is specified by a VaryingValue object, which specifies a value that varies over time. Note that it is possible to define parameters whose value varies over a domain other than time. For example, a color-correction effect can have a parameter whose value varies depending on the color space of a pixel in an image.

A VaryingValue object specifies its values by containing an ordered set of Control Points. Each Control Point specifies a value for a specific point in time. The Control Point identifies the point in time by specifying a rational number where zero represents the time at the beginning of the effect and 1/1 represents the time at the end of the effect.

A Varying Value specifies how to interpolate the effect between the time points whose value is specified by a Control Point. A Varying Value can have linear, constant, B-Spline, logarithmic, or Bezier interpolation.

- Linear interpolation means that the parameter varies in a straight line between two values.
- Constant interpolation means that the parameter holds a constant value until the next Control Point is reached.
- B-spline, logarithmic, and Bezier interpolations are mathematical formulas to create a curve between two points.

If two Control Points specify the same time, the second defines the value at that time. The first is used only to interpolate for times before the specified time.

If the first Control Point has a time greater than zero, its value is extrapolated as a constant backward to zero. If the last Control Point has a time less than 1/1, its value is extrapolated as a constant forward to 1/1.

25.7.6 Rendered Effect Essence

Sometimes it is desirable to compute the results of an Effect once and store them. When the Effect 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 essence data that can be played to produce the effect. The Effect identifies a rendered effect by containing a SourceClip that identifies the MasterMob and File SourceMob that describe the rendered essence. If there is more than one implementation of a rendering, the MasterMob could have an Essence Effect object.

25.7.7 Effects in Transitions

The Effect that is used in a Transition does not have any explicitly specified input essence Segments. Instead, the Effect gets its input essence Segments from the Segment that precedes it and the Segment that follows the Transition object in the Sequence.

In most cases, effects used in Transitions are defined to have two input essence Segments and a special-level parameter. When an effect is used in a Transition, the following specify its behavior:

The outgoing essence is the first, or A, input essence Segment.

The incoming essence is the second, or B, input essence Segment.

If the level parameter is not explicitly specified, its default value is a Varying Value with two Control Points: a value of zero at time zero, and a value of 1/1 at time 1/1.

Note that when an effect is used in a transition, it should not have any explicit input essence Segments. But an effect in a Transition can override the default values for the level parameter.

25.8 Scope and References

Scope Reference objects enable you to reference from within one slot the values produced by another slot. A Scope Reference can reference a Segment in a Nested Scope, or it can reference a Segment in another Slot. It can refer to a Segment in the same Nested Scope that it is defined in or in an outer Nested Scope that has it.

Although Scope References can be used to reference other Slots in the same Mob, they should only be used to reference Slots with the same data kind and the same relationship to time. If you need to reference a Slot with another relationship with time, you should use a SourceClip than does not specify a MobID parameter.

25.8.1 Why Use Scope References

Two reasons to use Scope References are:

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

Although you can layer overlapping sections of essence 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. A title superimposed on a long shot of a Mediterranean island.
2. A shot of the star inserted in a picture-in-picture effect over the island shot.
3. Ending with the island shot.

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

1. Effect for title effect with the SourceClip for the island shot.
2. Effect for picture-in-picture effect.
3. Another SourceClip for the island shot.

Within each of the Effects, you would specify one of the input segments to have a SourceClip of the island shot. The problem with this way of implementing the Sequence is that there are three SourceClips that refer to adjacent sections of the same scene with no linkage indicated in the file. If you change the length of one of the SourceClips or Effects, you need to change the other Segments in the Sequence to ensure continuity.

Alternatively, you could specify this with Nested Scope and Scope Reference objects where the Nested Scope would contain:

- One slot that has the full island shot.
- One slot that had a Sequence containing the two Effects and a Scope Reference to the other slot. Each of the Effects specifies one of its input essence Segments with a Scope Reference to the other 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.

25.8.2 How to Specify Scope References

The Mob defines a scope consisting of the ordered set of Slots. A Scope Reference object in a Slot can specify any 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 zero specifies the current scope, which is the innermost Nested Scope object that has the Scope Reference or the Mob scope if no Nested Scope object has it. A relative scope value of one specifies that you must pass through the Nested Scope object containing the Scope Reference to find the Nested Scope or Mob scope that has 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 one specifies the immediately previous slot.

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 Slot, the corresponding section of the slot is the time span that has the equivalent starting position from the beginning of the Slot and the equivalent length as the Scope Reference object has within its Slot. If the specified Slot has a different edit rate from the Slot containing the Scope Reference, the starting position and duration are converted to the specified 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 Nested Scope segments and the same duration as the Scope Reference object has in the specified scope.

25.9 Other Composition Mob Features

25.9.1 Preserving Editing Choices with 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.

25.9.2 Using Audio Fade In and Fade Out

The SourceClip FadeInLength, FadeInType, FadeOutLength, and FadeOutType properties allow you to specify audio fades without an Effect object. Audio fades use these SourceClip properties instead of Effect properties of the Effect 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 SourceClip properties rather than Effect properties 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 Effects.

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

- Insert a Transition object with the MonoAudioMixdown effect between the two audio SourceClips to cause them to overlap. If the FadeOutLength of the preceding SourceClip is not equal to the FadeInLength of the following SourceClip, the crossfade will be asymmetric.

Specify the overlapping audio SourceClips as different input essence Segments in a MonoAudioMixdown of an Effect.

26 Tutorial on Describing Essence

This chapter explains how AAF files describe essence.

26.1 Overview of Essence

AAF files can describe and contain a broad range of essence types and formats. These essence types include the following:

- Video essence in various formats (RGBA, YCbCr)
- Sampled audio essence in various formats (AIFC, WAVE)
- Static image essence

In addition to the essence formats described in this document, this interchange specification provides a general mechanism for describing essence formats and defines a plug-in mechanism that allows applications to import and export new types of essence data.

This specification defines the metadata in structures that are independent of the storage details of the essence format. This independence enables Composition Mobs to reference essence data independently of its format. A Composition Mob describes editing decisions in a manner that is independent of the following:

- Byte order of the essence (AIFC and WAVE)
- Whether the essence data is contained within the file or is in another container file
- Whether the digital essence data is accessible
- Format or compression used to store the digital essence data

This interchange specification makes it easier for applications to handle different formats by providing a layer that is common to all.

Essence source information describes the format of audio and video digital data, how the digital data was derived from tape or film, and the format of the tape and film. Source information can also include tape timecode, film edgecode data, and pulldown information.

This interchange specification uses the following mechanisms to describe essence:

- MasterMobs provide a level of indirection between Composition Mobs and File SourceMobs and can synchronize File SourceMobs.

- SourceMobs describe digital essence data stored in files or a physical media source such as videotape, audio tape, and film. The SourceMob has the following objects that provide information about the essence:
    - Slots specify the number of tracks in the essence source, the duration of each track, the edit rate, and the SourceMob that describes the previous generation of essence. In addition, Slots can have timecode and edge code information.
    - Essence Descriptors describe the kind of essence and the format of the essence and specify whether the SourceMobs describe digital essence data stored in files or a physical media source.
    - Pulldown objects describe how essence is converted between a film speed and a video speed.
- Essence data objects contain the digital essence data and provide supplementary information such as frame indexes for compressed digital essence data.

26.2 Describing Essence with MasterMobs

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

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

The same essence data can exist in more than one digital essence data implementation. Different implementations represent the same original essence data but can differ in essence format, compression, or byte order. If there are multiple implementations of digitized essence, the MasterMob can have an Essence Group object. The Essence Group object has a set of SourceClip objects, each of which identifies a SourceMob associated with a different implementation of the essence data. An application can examine these implementations to find the one that it is able to play or that it can play most efficiently. Essence Groups may be needed if you have systems with different architectures or compression hardware accessing a single interchange file.

If, when an essence data file is redigitized, it has to be broken into multiple files, this can be represented by a Sequence object in the MasterMob that has a series of SourceClip objects, each identifying the SourceMob associated with one of the files.

Typically, MasterMobs have a very simple structure. They have an externally visible Slot for each track of essence and do not have any other slots. Typically, each Slot has a single SourceClip object that identifies the SourceMob. MasterMobs cannot have Operation Groups, Nested Scopes, Selectors, Edit Rate Converters, or Transitions.

The following lists the reasons for having a Slot in a MasterMob have an object other than a SourceClip:

- If there are multiple implementations of the same essence, the Slot can have an Essence Group instead of a SourceClip object.

- If the essence source has been broken into several SourceMobs, the Slot can have a Sequence object. The Sequence object cannot have any component other than a SourceClip object or an Essence Group object.
- If one of a limited set of correction effects is applied to the essence data

Figure 23 below illustrates the containment diagram for a MasterMob describing timeline essence data, such as audio or video.

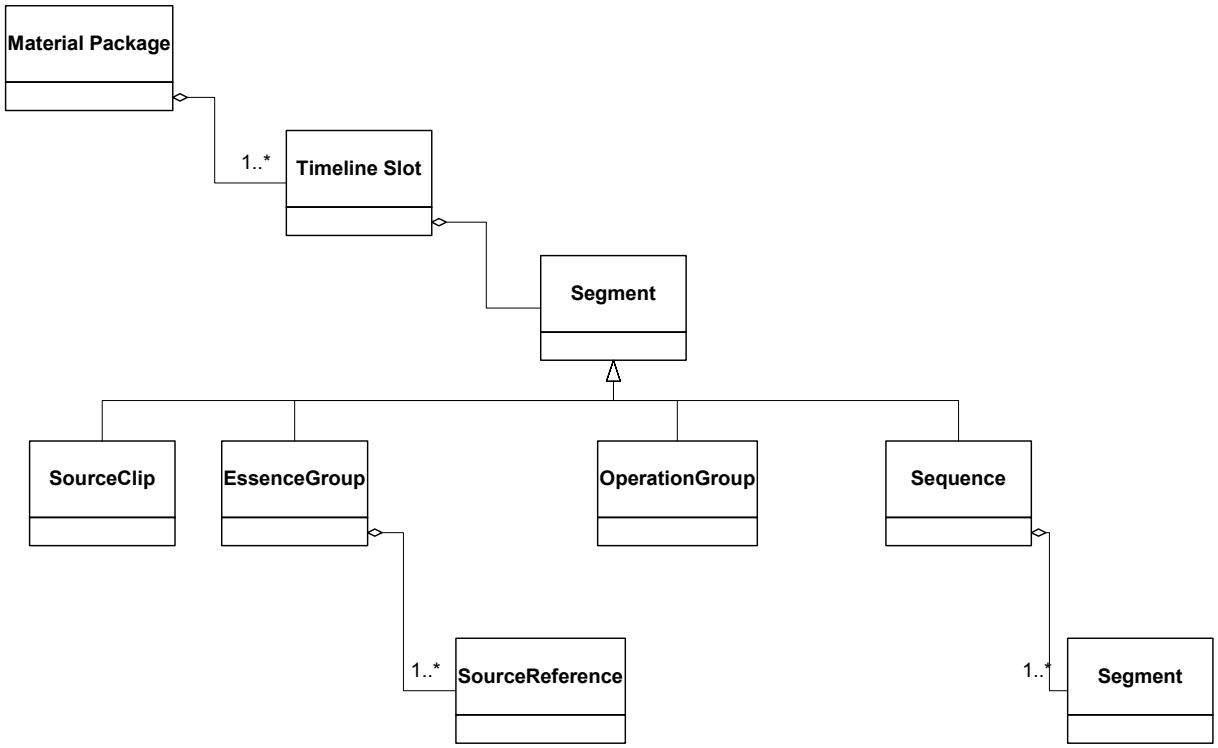


Figure 23 Containment Diagram for a MasterMob with TimelineMobSlots

26.3 Describing Essence with SourceMobs

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

If the essence described by the SourceMob has been derived from a previous generation of essence, the Slots should have SourceClips that identify the Mob that describes the previous generation. If the SourceMob describes essence that is not derived from a previous generation, the Slots should have zero-value SourceClips.

26.3.1 Sample Rate and Edit Rate in Timeline Essence

In many cases the sample rate and edit rate in a file SourceMob will be the same. However, it is possible to use different edit rates and sample rates in a SourceMob. For example, you can create a SourceMob 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 File Descriptor . When accessing the digital essence data, your application must convert from the edit rate to the sample rate.

26.3.2 The Source Origin in Timeline Essence

When an application accesses the digital essence data, it locates the starting position by measuring from a position known as the source origin. Each file SourceMob indicates this position for each TimelineMobSlot in order to provide a reference point for measurements of its essence data.

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

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 Essence data object starts at a different point. However, the application will ensure that existing SourceClips in Composition Mobs remain valid by changing the value of the Origin property in the MasterMob. By setting the Origin to the current offset of the original starting point, the application ensures that existing Composition Mobs remain valid.

26.3.3 Converting Edit Units to Sample Units

A TimelineMobSlot uses its own edit rate. So, a SourceClip 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 essence data.

To play the digital essence data referenced by a Composition Mob, the application uses the StartTime and Length values of the Composition Mob's SourceClip, which are specified in edit units, along with the edit rate to determine the samples to be taken from the essence data. The application converts edit units to sample durations, adds the file Slot's Origin to the SourceClip's StartTime, then converts the resulting sample time offset to a sample byte offset. Performing the final calculation for some essence 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 has a frame index.

An application would not need to reference the original physical SourceMob 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 essence 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 essence data by converting edit units to sample units and then to byte offsets.
- MasterMobs maintain a reference point in the digitized essence data called the source origin. Composition Mobs reference positions in the essence data relative to the origin.

26.4 Describing Essence Format with Essence Descriptors

SourceMobs describe the details of the essence format with an Essence Descriptor object. Essence Descriptor is an abstract class that describes the format of the essence data. The essence data can be digitized essence data stored in a file or it can be essence data on audio tape, film, videotape, or some other form of essence storage.

There are two kinds of Essence Descriptors:

- File Descriptors that describe digital essence data stored in Essence data objects or in noncontainer data files. The Essence File Descriptor class is also an abstract class; its subclasses describe the various formats of digitized essence. If an Essence Descriptor object belongs to a subclass of File Descriptor, it describes digital essence data. If an Essence Descriptor object does not belong to a subclass of File Descriptor, it describes a physical media source.
- Essence Descriptors that describe a physical media source. This specification defines the Film Descriptor and Tape Descriptor, but additional private or registered subclasses of Essence Descriptors can be defined.

If the digital essence data is stored in an AAF file, the ContainerDefinition property in the File Descriptor shall reference the ContainerDefinition for the AAF file format. Digital essence data can be stored in a noncontainer

data file to allow an application that does not support this interchange specification to access it or to avoid duplicating already existing digital essence data. However, since there is no MobID stored with raw essence data, it is difficult to identify a raw essence data file if the Locator information is no longer valid. The format of the digital essence data in the raw file is the same as it would be if it were stored in an Essence data object.

The File Descriptor specifies the sample rate and length of the essence data. The sample rate of the data can be different from the edit rate of the SourceClip object that references it.

Figure 24 and Figure 25 below illustrate the containment diagram for File SourceMobs and the containment diagram for Physical SourceMobs.

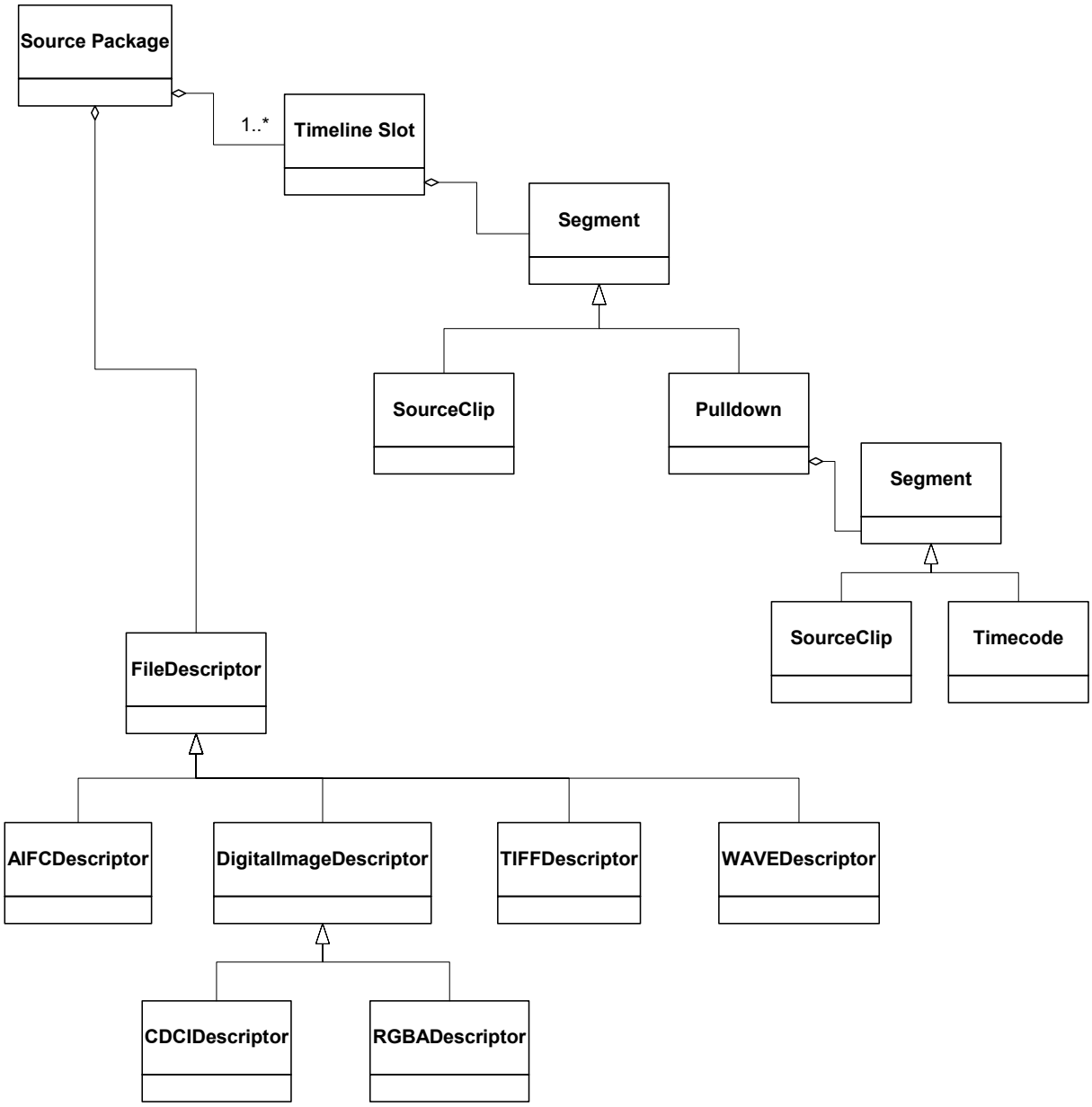


Figure 24 Containment Diagram for a File SourceMob

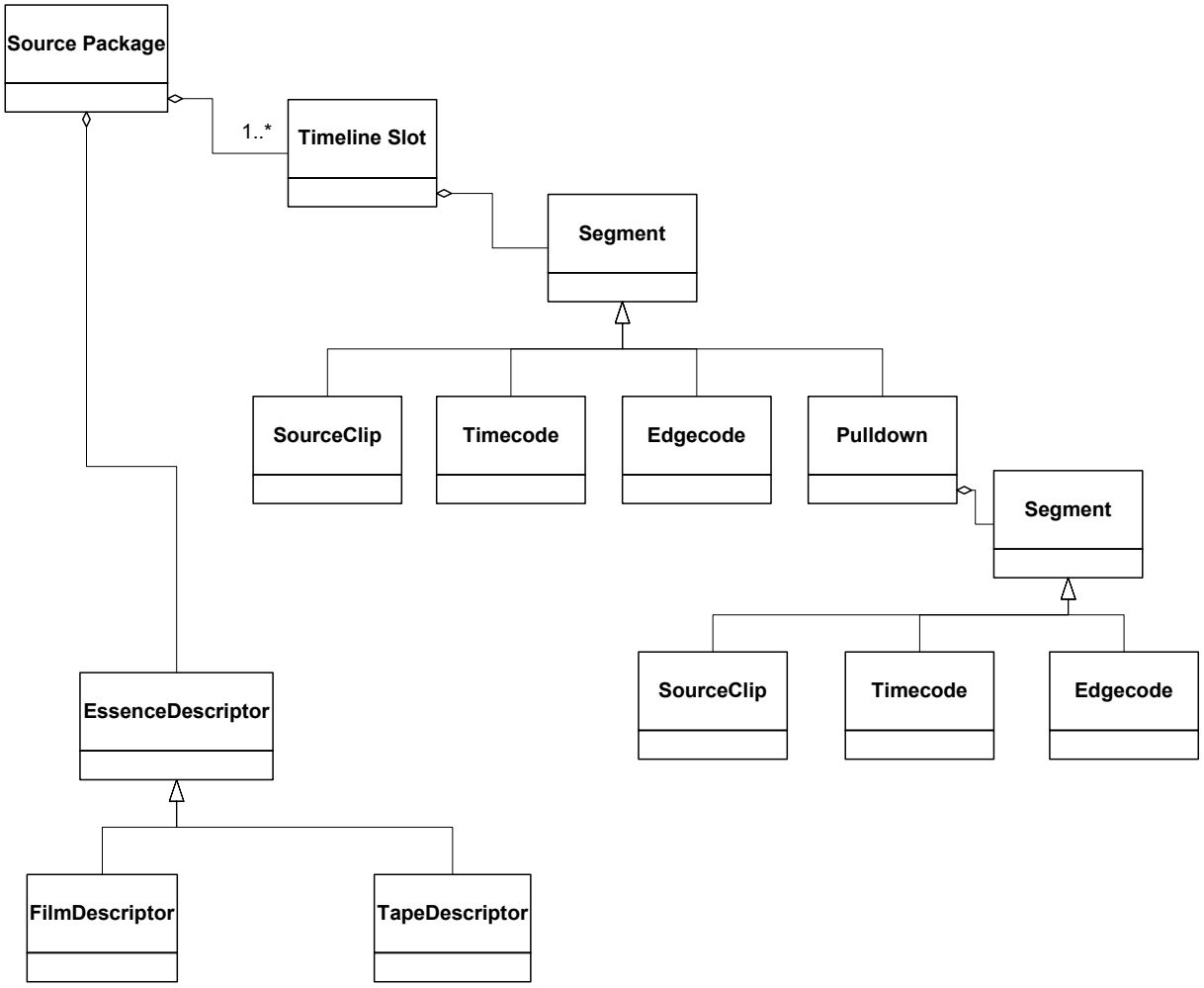


Figure 25 Containment Diagram for a Physical SourceMob

26.4.1 Describing Image Essence

The goal of the 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 essence from an analog source. This information allows applications to interpret the digital data to represent the original essence.

This section explains the image essence descriptions that are common to all image essence 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

26.4.2 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.

26.4.3 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 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.

26.4.4 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 essence 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 essence, 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 essence has to be treated differently. If the digital essence was computer generated (RGB), the values can be either {0,1} (even field first) or {1,0} (odd field first).

26.4.5 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.

26.4.6 Properties Describing Compression

The Compression property specifies that the image is compressed and the kind of compression used. 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 essence. While tables may vary per image, these tables will represent a starting point.

The JPEG Image Data object has 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.

Other values of the compression parameter will be defined for other schemes such as MPEG-2 Video, and these other schemes will have their own parametric metadata and frame tables, etc.

26.4.7 RGBA Component Image Descriptors

An RGBA Component Image object describes essence data that consists of 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.

26.4.8 Color Difference Component Image Descriptors

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

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.

26.4.9 Describing TIFF Image Essence

Informative note: The TIFF image format has been superseded by the Color Difference Component Image Descriptor format and the RGBA Component Image Descriptor format in the current version of the specification. The TIFF format is included in this specification for compatibility.

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

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.

26.4.10 Describing Audio Essence

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 AIFC essence format defined by this specification 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 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 includes all of the metadata needed for playback.

If a MasterMob or SourceMob has two stereo audio essence tracks, the PhysicalChannelNumber indicates the physical input channel according to the following convention: 1 indicates the left channel and 2 indicates the right channel.

26.4.11 Describing Tape and Film

The Tape Descriptor describes videotape and audio tape media sources. The Film Descriptor describes film sources. Their properties describe the physical storage format used for the essence. When you create a tape or film SourceMob, 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.

26.4.12 Describing Timecode

Timecode typically is described in a SourceMob or in a Composition Mob. Timecode can be described by specifying a starting timecode value or by including a stream of timecode data.

A Timecode object in a SourceMob typically appears in a Slot in a SourceMob 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 SourceMob can have:

- A Slot for each track of essence on the tape; the Slot should have a single SourceClip whose Length equals the duration of the tape.

- A Slot 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 Slot can have a Sequence of Timecode objects; each representing a contiguous section of timecode on the tape or can specify the timecode stream data.

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 SourceMob in which the duration of the SourceClip does not necessarily match the duration of the videotape.

The timecode information for digital essence data and file SourceMobs is contained in the videotape SourceMob that describes the videotape used to generate the digital essence data.

The starting timecode for digital essence data is specified by the SourceClip in the File SourceMob and by the timecode track in the videotape SourceMob. The SourceClip specifies the MobID of the videotape SourceMob, the SlotID for the Slot describing the essence data, and the offset in that track. To find the timecode value, you must find the value specified for that offset in the timecode Slot of the videotape SourceMob.

If a videotape has continuous timecode for the entire tape, it is specified by a single Timecode object. If a videotape has discontinuous timecode, interchange files typically describe it with a single Timecode object that encompasses all timecode values that are used on the videotape. Discontinuous timecode can also be described by the following

A timecode track that has a sequence of Timecode objects, each of which specifies the starting timecode and the duration of each section of continuous timecode on the videotape

A timecode stream that duplicates the timecode data stored 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.

If a SourceMob has more than one timecode Slot, the PhysicalTrackNumber property indicates the purpose of each as described in Table below.

Physical Track Number	Usage
1	default TC
2	Sound TC
3	Aux. TC
4	Aux. TC
5	Aux. TC
6	Aux. TC
7	Aux. TC

Physical Track Number and Timecode Usage

26.4.13 Describing Edgecode

Film edgecode is described in Film Mobs. Edgecode is specified with a TimelineMobSlot containing an Edgecode object. The Edgecode object specifies the starting edgecode value, the type of film, and the text egdecode header. If there is more than one edgecode Slot, the purpose of each is described by the PhysicalTrackNumber property as described in the Table below.

Physical Track Number	Usage
1	Keycode Number

Physical Track Number	Usage
2	Ink Number
3	Aux. Ink Number

Physical Track Number and Edgecode Usage

26.4.14 Describing Essence with Pulldown Objects

Pulldown is a process to convert essence with one frame rate to essence with another frame rate. This interchange specification describes how essence has been converted with Pulldown objects in File SourceMobs and videotape SourceMobs.

26.4.15 What is Pulldown?

Pulldown is a process to convert between essence at film speed of 24 frames per second (fps) and essence 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 essence 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 tapping the image by means of either 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 essence 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.

26.4.16 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 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 26 below 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.

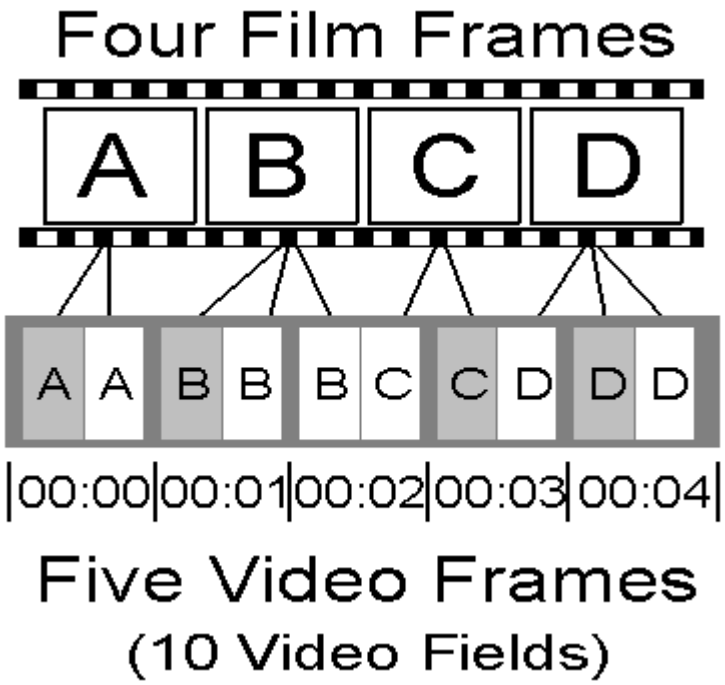


Figure 26 Telecine Three-Two Pulldown

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.

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 material 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 essence 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.

26.4.17 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.

26.4.18 Pulldown Objects in SourceMobs

If NTSC video is digitized to a 24-fps film rate using a reverse Three-Two pulldown, both the File SourceMob and the Videotape SourceMob have Pulldown objects.

The Pulldown object in the File SourceMob describes how the videotape was digitized. The track in the File SourceMob has an edit rate of 24/1 but the SourceClip in the Pulldown object has an edit rate of 30/1. The Pulldown object specifies the phase of the first frame of the digital essence 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 SourceMob describes how the video was generated from film. The track in the videotape SourceMob has an edit rate of 30/1 but the SourceClip 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 SourceMob has a Sequence object which has a Pulldown object for each section of film that has been telecined. If the videotape has discontinuous timecode and the videotape SourceMob timecode track has 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.

27 Meta-Classes

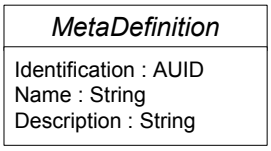
AAF Meta-Classes are the classes provided for defining Interchange Objects. This chapter contains the reference descriptions of the AAF Meta-Classes. The reference pages are arranged alphabetically, followed by the MetaDictionary class which contains the MetaDefinitions.

27.1 MetaDefinition class

The MetaDefinition class is an abstract class that defines a class, type, or property in an AAF file.

The MetaDefinition class is a root class.

The MetaDefinition class is an abstract class.



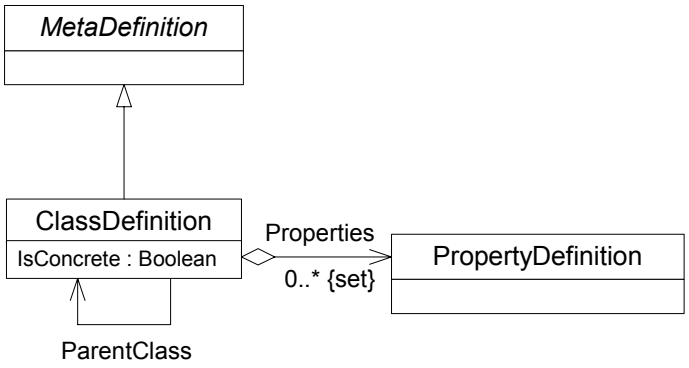
Property Name	Type	Req ?	Meaning
Identification	AUID	Req	Specifies the unique identifier for the item being defined
Name	String	Req	Specifies the display name of the item being defined
Description	String	Opt	Provides an explanation of the use of the item being defined

27.2 ClassDefinition class

The ClassDefinition class extends the class hierarchy defined in this document by specifying a new class or by defining additional optional properties for a class defined in this document.

The ClassDefinition class is a sub-class of the MetaDefinition class.

All ClassDefinition objects shall be owned by the MetaDictionary object.



Property Name	Type	Req ?	Meaning
ParentClass	WeakReference to ClassDefinition	Req	Specifies the parent of the class being defined
Name	StrongReferenceSet of PropertyDefinition	Opt	Specifies the set of PropertyDefinition objects that define the properties for a class
IsConcrete	Boolean	Req	Specifies if the class is concrete. If the class is not concrete, then it is abstract. Any object in an AAF file that belongs to an abstract class shall also belong to a concrete subclass of the abstract class

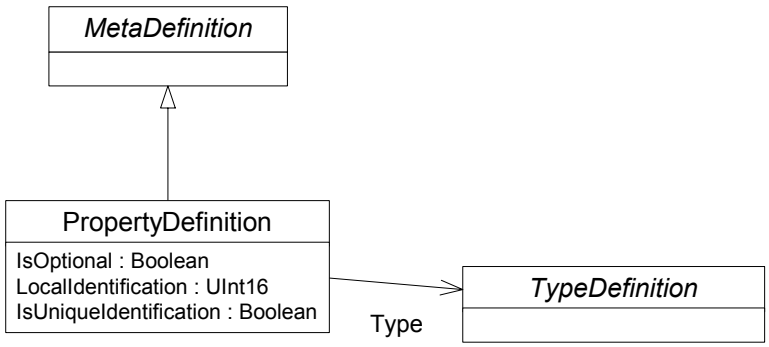
Any class extension shall be descended from the InterchangeObject class. A Class Definition object specifying the InterchangeObject class shall have a ParentClass property with a weak reference to itself.

27.3 PropertyDefinition class

The PropertyDefinition class describes properties allowed for a class.

The PropertyDefinition class is a sub-class of the MetaDefinition class.

A PropertyDefinition object shall be owned by a ClassDefinition.



Property Name	Type	Req ?	Meaning

Property Name	Type	Req ?	Meaning
Type	WeakReference to TypeDefinition	Req	Specifies the property type
IsOptional	Boolean	Req	Specifies whether objects instances can omit a value for the property
LocalIdentification	UInt16	Req	Specifies a local integer identification that is used to identify the property in the AAF file
IsUniqueIdentifier	Boolean	Opt	Specifies that this property provides a unique identification for this object

The PropertyDefinition object specifies that a property can be used in a class.

The LocalIdentification property is used internally within the AAF file for efficiency purposes but has no semantic meaning.

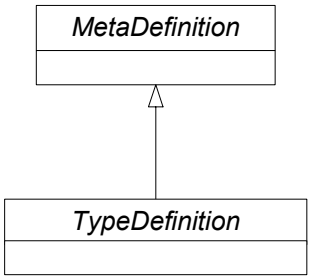
Informative note: In the AAF reference implementation, related interfaces are IAAFPropertyDef and IAAFClassDef.

27.4 TypeDefinition class

The TypeDefinition class defines a property type.

The TypeDefinition class is a sub-class of the MetaDefinition class.

The TypeDefinition class is an abstract class.



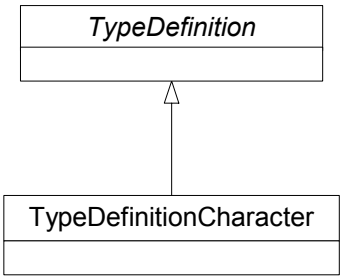
The TypeDefinition class does not define any additional properties.

27.5 TypeDefinitionCharacter class

The TypeDefinitionCharacter class defines a property type that has a value of a single 2-byte character.

The TypeDefinitionCharacter class is a sub-class of the TypeDefinition class.

All TypeDefinitionCharacter objects shall be owned by the MetaDictionary object.



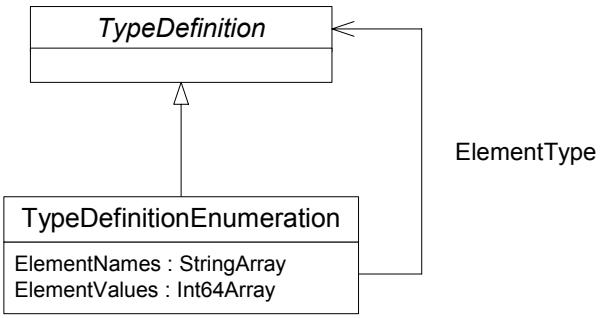
The TypeDefinitionCharacter class does not define any additional properties.

27.6 TypeDefinitionEnumeration class

The TypeDefinitionEnumeration class defines a property type that can have one of a set of integer values.

The TypeDefinitionEnumeration class is a sub-class of the TypeDefinition class.

All TypeDefinitionEnumeration objects shall be owned by the MetaDictionary object.



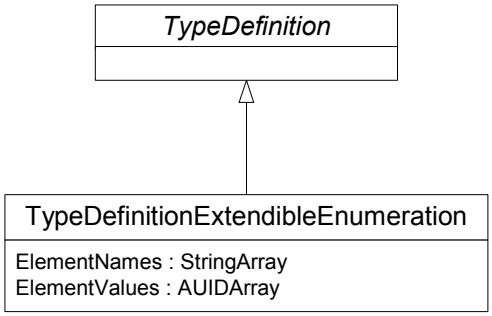
Property Name	Type	Req ?	Meaning
ElementType	WeakReference to TypeDefinition	Req	Specifies the TypeDefinition that defines the underlying integer type
ElementNames	StringArray	Req	Specifies the names associated with each enumerated value
ElementValues	Int64Array	Req	Specifies the valid enumerated values. The integer values shall be positive and each value in the array shall be unique

27.7 TypeDefinitionExtendibleEnumeration class

The TypeDefinitionExtendibleEnumeration class defines a property type that can have one of an extendible set of AUID values.

The TypeDefinitionExtendibleEnumeration class is a sub-class of the TypeDefinition class.

All TypeDefinitionExtendibleEnumeration objects shall be owned by the MetaDictionary object.



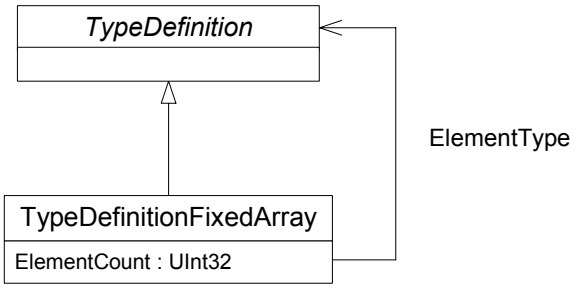
Property Name	Type	Req ?	Meaning
ElementNames	StringArray	Req	Specifies the names associated with each enumerated value
ElementValues	AUIDArray	Req	Specifies the known AUID values that can be used in this type

27.8 TypeDefinitionFixedArray class

The TypeDefinitionFixedArray class defines a property type that has a fixed number of values of the underlying type. The order of the values is meaningful.

The TypeDefinitionFixedArray class is a sub-class of the TypeDefinition class.

All TypeDefinitionFixedArray objects shall be owned by the MetaDictionary object.



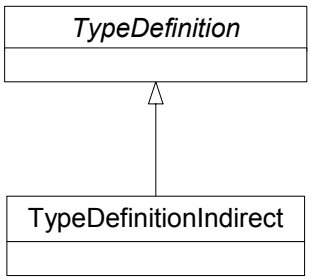
Property Name	Type	Req ?	Meaning
ElementType	WeakReference to TypeDefinition	Req	Specifies the TypeDefinition that defines the type of each element of the array
ElementCount	UInt32	Req	Specifies the number of elements in the array

27.9 TypeDefinitionIndirect class

The TypeDefinitionIndirect class defines a property type that has a value whose type is specified in each instance.

The TypeDefinitionIndirect class is a sub-class of the TypeDefinition class.

All TypeDefinitionIndirect objects shall be owned by the MetaDictionary object.



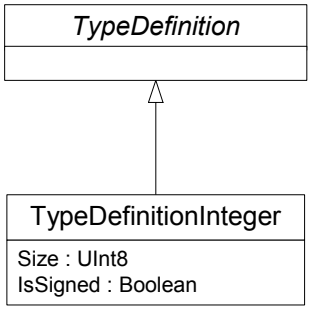
The TypeDefinitionIndirect class does not define any additional properties.

27.10 TypeDefinitionInteger class

The TypeDefinitionInteger class defines a property type that is an integer with the specified number of bytes.

The TypeDefinitionInteger class is a sub-class of the TypeDefinition class.

All TypeDefinitionInteger objects shall be owned by the MetaDictionary object.



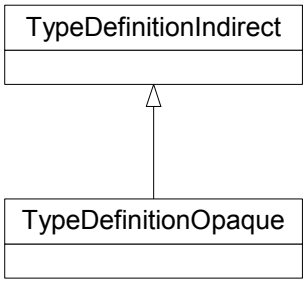
Property Name	Type	Req ?	Meaning
Size	UInt8	Req	Specifies the number of bytes to store the integer. Legal values are 1, 2, 4, and 8
IsSigned	Boolean	Req	Specifies if the integer is signed (True) or unsigned (False)

27.11 TypeDefinitionOpaque class

The TypeDefinitionOpaque class defines a property type that has a value whose type is specified in each instance.

The TypeDefinitionOpaque class is a sub-class of the TypeDefinitionIndirect class.

All TypeDefinitionOpaque objects shall be owned by the MetaDictionary object.



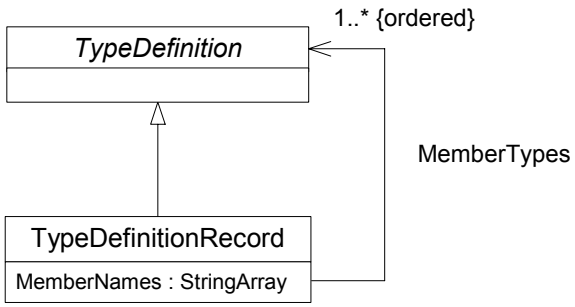
The TypeDefinitionOpaque class does not define any additional properties.

27.12 TypeDefinitionRecord class

The TypeDefinitionRecord class defines a property type that consists of an ordered set of fields, where each field has a name and type.

The TypeDefinitionRecord class is a sub-class of the TypeDefinition class.

All TypeDefinitionRecord objects shall be owned by the MetaDictionary object.



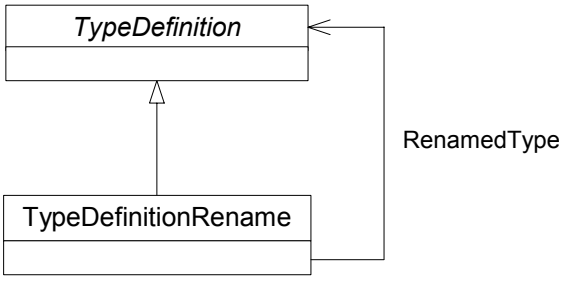
Property Name	Type	Req ?	Meaning
MemberTypes	WeakReferenceVector of TypeDefinition	Req	Specifies the type of each element of the record
MemberNames	StringArray	Req	Specifies the name of each element of the record

27.13 TypeDefinitionRename class

The TypeDefinitionRename class defines a property type that has the same structure and representation as its underlying type but has a different meaning.

The TypeDefinitionRename class is a sub-class of the TypeDefinition class.

All TypeDefinitionRename objects shall be owned by the MetaDictionary object.



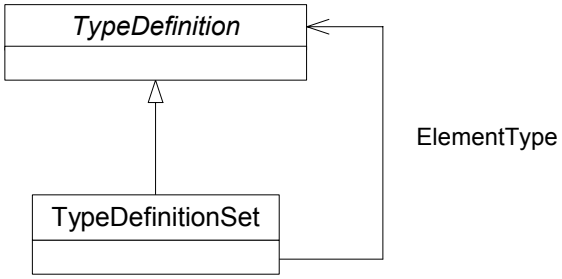
Property Name	Type	Req ?	Meaning
RenamedType	WeakReference to TypeDefinition	Req	Specifies the underlying type

27.14 TypeDefinitionSet class

The TypeDefinitionSet class defines a property type that has a collection of object references to uniquely identified objects. The order of the objects has no meaning.

The TypeDefinitionSet class is a sub-class of the TypeDefinition class.

All TypeDefinitionSet objects shall be owned by the MetaDictionary object.



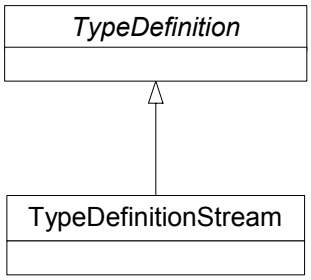
Property Name	Type	Req ?	Meaning
ElementType	WeakReference to TypeDefinition	Req	Specifies the TypeDefinition that identifies the kind of object reference. This TypeDefinition shall belong to either the TypeDefinitionStrongObjectReference or TypeDefinitionWeakObjectReference

27.15 TypeDefinitionStream class

The TypeDefinitionStream class defines a property type that is stored in a stream and has a value that consists of a varying number of the bytes. The order of the bytes is meaningful.

The TypeDefinitionStream class is a sub-class of the TypeDefinition class.

All TypeDefinitionStream objects shall be owned by the MetaDictionary object.



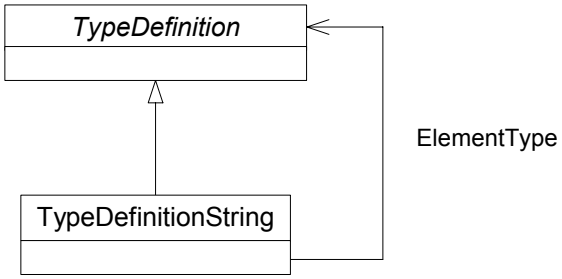
The TypeDefinitionStream class does not define any additional properties.

27.16 TypeDefinitionString class

The TypeDefinitionString class defines a property type that consists of a zero-terminated array of the underlying character or integer type.

The TypeDefinitionString class is a sub-class of the TypeDefinition class.

All TypeDefinitionString objects shall be owned by the MetaDictionary object.



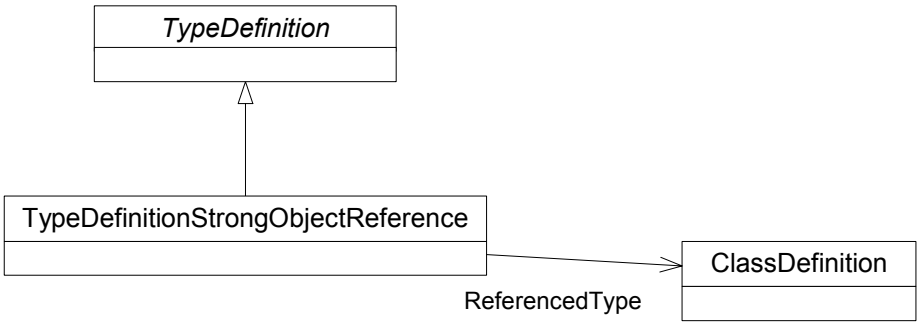
Property Name	Type	Req ?	Meaning
ElementType	WeakReference to TypeDefinition	Req	Specifies the string element, which may be a character (TypeDefinitionCharacter) or integer (TypeDefinitionInteger)

27.17 TypeDefinitionStrongObjectReference class

The TypeDefinitionStrongObjectReference class defines a property type that defines an object relationship where the target of the strong reference is owned by the object with the property with the TypeDefinitionStrongObjectReference type. An object can be the target of only one strong reference.

The TypeDefinitionStrongObjectReference class is a sub-class of the TypeDefinition class.

All TypeDefinitionStrongObjectReference objects shall be owned by the MetaDictionary object.



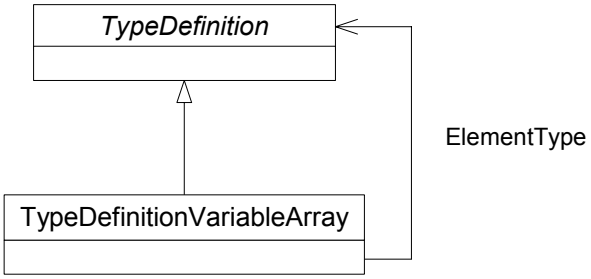
Property Name	Type	Req ?	Meaning
ReferencedType	WeakReference to ClassDefinition	Req	Specifies the class that the referenced object shall belong to (the referenced object may also belong to a subclass of the referenced class)

27.18 TypeDefinitionVariableArray class

The TypeDefinitionVariableArray class defines a property type that has a varying number of values of the underlying type. The order of the values is meaningful.

The TypeDefinitionVariableArray class is a sub-class of the TypeDefinition class.

All TypeDefinitionVariableArray objects shall be owned by the MetaDictionary object.



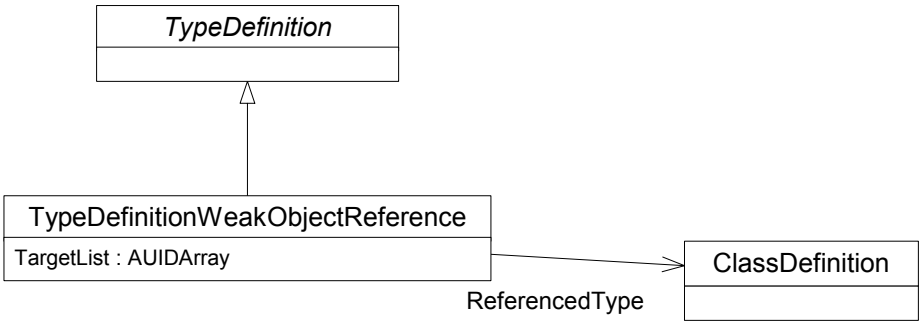
Property Name	Type	Req ?	Meaning
ElementType	WeakReference to TypeDefinition	Req	Specifies the type of the element of the array

27.19 TypeDefinitionWeakObjectReference class

The TypeDefinitionWeakObjectReference class defines a property type that defines an object relationship where the target of the weak reference is referenced by the object with the property with the TypeDefinitionWeakObjectReference type. Only objects that define a unique identification (AUID) can be the targets of weak object references. An object can be the target of one or more than one weak references.

The TypeDefinitionWeakObjectReference class is a sub-class of the TypeDefinition class.

All TypeDefinitionWeakObjectReference objects shall be owned by the MetaDictionary object.



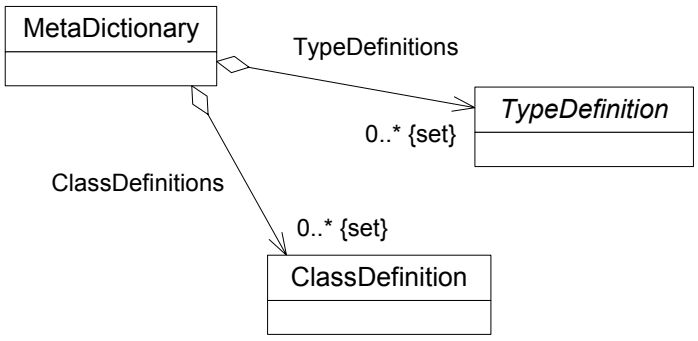
Property Name	Type	Req ?	Meaning
ReferencedType	WeakReference to ClassDefinition	Req	Specifies the class that the referenced object shall belong to (the referenced object may also belong to a subclass of the referenced class)
TargetList	AUIDArray	Req	Specifies the AUIDs that specify the properties from the root of the file to the property that has the

Property Name	Type	Req ?	Meaning
			StrongReferenceSet containing the uniquely identified objects that may be the target of the weak reference. The first AUID in the array identifies the object in the file's root storage. The last AUID in the array identifies the property containing the set of uniquely identified objects. The AUIDs between the first and the last identify properties that must have a TypeDefinitionStrongObjectReference and define the containing hierarchy from the object in the root storage to the object containing the StrongReferenceSet.

27.20 MetaDictionary class

The MetaDictionary class contains MetaDefinition objects. An AAF file shall have exactly one MetaDictionary object.

The MetaDictionary class is a root class.



Property Name	Type	Req ?	Meaning
ClassDefinitions	StrongReferenceSet of ClassDefinition	Opt	Specifies the ClassDefinitions that are used in the file
TypeDefinitions	StrongReferenceSet of TypeDefinition	Opt	Specifies the TypeDefinitions that are used in the file

Informative note: The Dictionary object API in the AAF reference implementation (IAAFDictionary) is used to access the MetaDictionary.

28 Extensions

28.1 Overview of Extending AAF

The Advanced Authoring Format is designed to allow extensions. AAF files can include extensions that define new effects, new kinds of metadata, and new kinds of essence data.

As the technologies of authoring applications advance, people can use the applications to do new things and will want to interchange this new kind of information between applications. Typically, these new features are added by one or a few applications, and gradually, as the technology matures, the features become common to many applications. Consequently, these features are first defined as private extensions to this specification and may later progress to be included in this specification.

Applications may want to store information in extensions for the following reasons:

- To store optional information which can be displayed to the user by other applications. For example an application can store user-specified comments about essence or compositions.
- To store information for targeted exchange. Two or more applications can be coded to understand private or registered information.
- To store internal application-specific information so that the application can use this interchange format as a native file format.
- To define new essence formats for use by plug-in codecs.

The extra information stored by an application can vary in scale from a single private property to a complex structure of private objects.

Extensions may define the following:

- New effects
- New classes
- New properties
- New property types
- New essence types
- Plug-in code

New effects and new essence types may require special code to process the effect or essence. This code can be supplied in a plug-in module. The plug-in mechanism is not defined as part of this specification. This specification defines the properties required to specify a locator to find a plug-in.

Extensions are specified in the Header Dictionary and the MetaDictionary objects.

28.2 Defining New Effects

The EffectsDefinition class defines new effects. Effect definitions include the following:

AUID that identifies the effect

Effect name and description for display purposes

- Plugin locators
- Number of essence input segments, specifies -1 for effects that can have any number of input essence segments
- Control code definitions that define the effect's parameters:
    - AUID identifying control code
    - Data kind of parameter
    - Range of allowed values
    - Text associated with enumerated values

When appropriate new Effect Definitions should use existing control codes and data kinds. If an Effect Definition specifies a previously defined control code, it must specify the same data kind.

If the data kind definition specifies a range of allowed values, an Effect Definition can limit the range of allowed values to a lesser range but cannot extend the range.

28.3 Defining New Classes

To define a new class, you need to generate a AUID for the class and then have your application create an ClassDefinition object in any interchange file that has the new class. The ClassDefinition object specifies the following:

- AUID that identifies the class
- Superclass of the class
- Class name for display purposes
- Properties that can be included in objects belonging to the class

28.4 Defining New Properties

You define new properties as part of a Class Definition. If you are defining a new class, you must specify all the properties that can be used for the class. If you are adding optional properties to a class defined by this document, you need only to specify the new properties in the class definition. You can omit the properties defined in this document from the class definition.

In a class definition, each property definition specifies the following:

- AUID that identifies the property
- Property name for display purposes
- AUID that identifies the property type
- Optionally, range of allowed values or text associated with enumerated values

If the property has a new property type, the property type definition shall be included in the definition objects defined in the MetaDictionary object. If the property has a property type defined in this document, you can omit the property type definition.

28.5 Defining New Essence Types

The scope of the task of defining new essence types varies greatly depending on how different the new essence type is from the existing ones. Defining a new essence type can consist of any of the following

- Defining a new compression method for an existing data kind, such as video
- Defining a new essence type that requires a new data kind for segments
- Defining a new essence type that requires a new kind of Slot and a new set of classes for Composition Mobs

This section contains a brief description of how to define a new essence type that uses an existing data kind. Describing the requirements of defining a new data kind, a new kind of Slot, or new classes for Composition Mobs is beyond the scope of this document.

To define a new essence type, you must:

- Define a new subclass of FileDescriptor or a new subclass of EssenceDescriptor for SourceMobs that are not File SourceMobs
- Define a new subclass of EssenceData or use an existing class
- Create a plug-in essence codec that can import and export the essence data based on the information in the File Descriptor

Typically, when defining a new essence format you can use the existing classes for the Slots and Segments in the SourceMob, but you do have to define a new Essence Descriptor.

If the new essence type consists of a single kind of essence data, such as a single video stream or a static image, the SourceMob should have a single Slot. If the essence type is a compound format that has multiple tracks of essence data, the File SourceMob should have a separate Slot for each separate track of essence data.

28.6 Tracking Changes with Generation

If your application stores extended data that is dependent on data stored in AAF's built-in classes and properties, your application may need to check if another application has modified the data in the built-in classes and properties.

The InterchangeObject Generation property allows you to track whether another application has modified data in an AAF file that may invalidate data that your application has stored in extensions. The Generation property is a weak reference to the Identification object created when an AAF file is created or modified. If your application creates extended data that is dependent on data stored in AAF built-in classes or properties, you can use the Generation property to check if another application has modified the AAF file since the time that your application set the extended data. To do this, your application stores the value of the GenerationAUID of the Identification object created when your application set the value of the extended data.

Consider the following example (Figure 27 below), an application creates a Sequence containing a SourceClip with extended properties that contain data that make it more efficient for the application to play the SourceClip. However, this data is dependent on the section of essence to be played and the position of the SourceClip in the Sequence. The section of essence to be played is specified by the SourceClip's SourceID and SourceSlotID properties and the position in the Sequence is specified by the Sequence Components property.

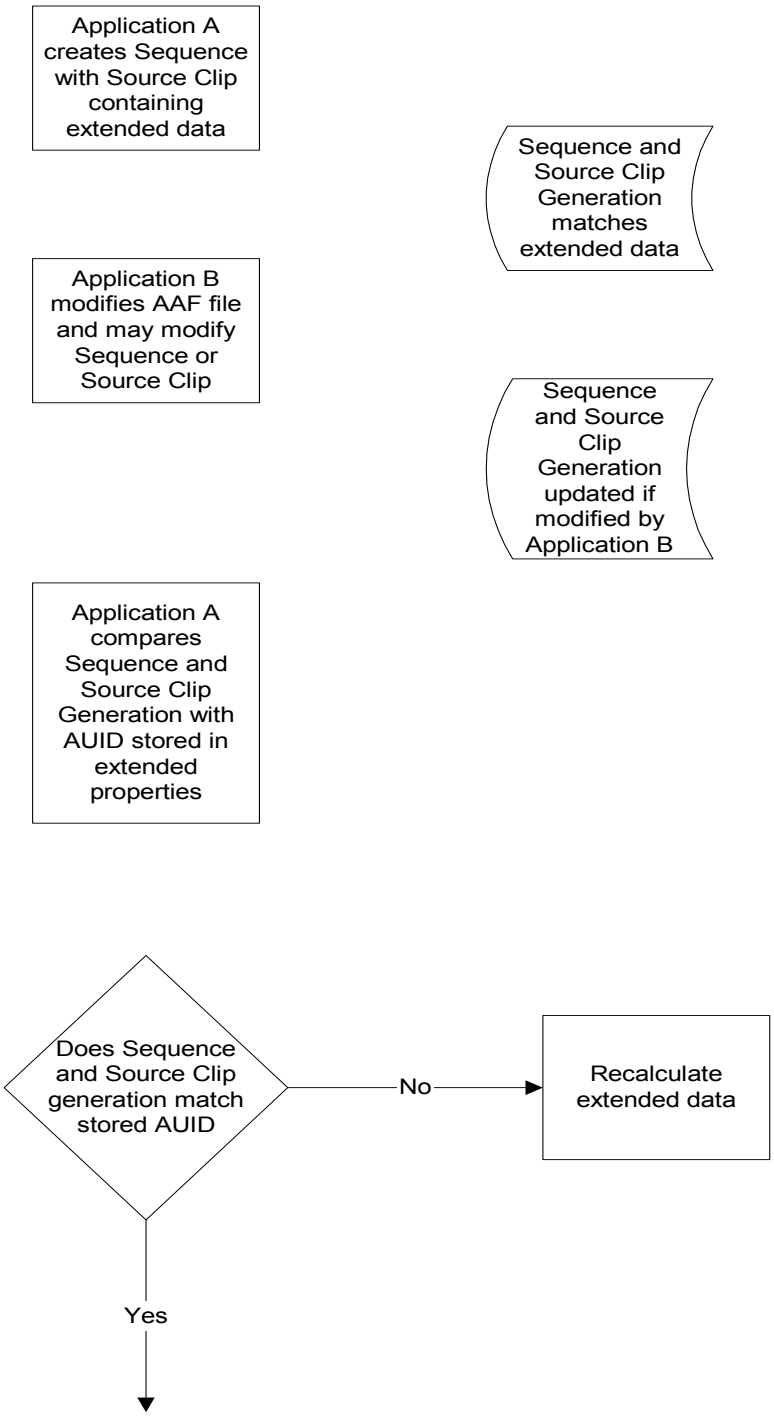


Figure 27 Using InterchangeObject::Generation to track generation

When an object is created or modified, the Generation property is set as a weak reference to the Identification object created when the AAF file was created or opened for modification. If the Generation property is not present in an object, that object was created or last modified when the file was first created.

29 Bibliography

1. Advanced Authoring Format, http://www.aafassociation.org

2. EBU / SMPTE Task Force for Harmonized Standards for the Exchange of Program Material as Bit-streams – 1998, http://www.smpte.org and http://www.ebu.ch
3. SMPTE 377M-2003 Television — Material Exchange Format (MXF) File Format Specification
4. SMPTE 330M-2000 for Television - Unique Material Identifier (UMID)
5. SMPTE 12M, 1995: For Television, Audio and Film – Time and Control Code
6. Apple Computer – Audio Interchange File Format with Compression (AIFC) v1
7. Microsoft Corporation – RIFF Waveform Audio File Format
8. Aldus Developers Desk – TIFF Revision 6.0 Final – June 3, 1992.