AAF Specification

Version 1.0 DR4

Notice

Product specifications are subject to change without Notice. The software described in this document is furnished under a license agreement, and may be used or copied only in accordance with the terms of the license agreement.

THE ADVANCED AUTHORING FORMAT SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR OR INTENDED PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION, OR SAMPLE. IN NO EVENT WILL THE PROMOTERS OR ANY OF THEM BE LIABLE FOR ANY DAMAGES, INCLUDING BUT NOT LIMITED TO LOSS OF PROFITS, LOSS OF USE, INCIDENTAL, CONSEQUENTIAL, INDIRECT, OR SPECIAL DAMAGES ARISING OUT OF USE OF THIS ADVANCED AUTHORING FORMAT SPECIFICATION WHETHER OR NOT SUCH PARTY OR PARTIES HAD ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

Trademarks

Avid, Avid Cinema, Digidesign, Media Composer, OMF, Open Media Framework, OMF Interchange, Pro Tools, and Softimage are registered trademarks and Sound Designer II is a trademark of Avid Technology, Inc., or its subsidiaries or divisions.

Adobe, After Effects, Photoshop, and Premiere are registered trademarks of Adobe Systems Incorporated. Apple and Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries. DirectX, Microsoft, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and other countries. Matrox is a registered trademark of Matrox Electronic Systems Ltd. Pinnacle is a trademark of Pinnacle Systems, Inc. Sound Forge is a registered trademark of Sonic Foundry, Inc. UNIX is a registered trademark of The Open Group. All other trademarks contained herein are the property of their respective owners.

Table of Contents

1. Introduction

The Advanced Authoring Format, or AAF, is an industry-driven, cross-platform, multimedia file format that will allow interchange of essence and compositional information between AAF-compliant applications.

Background

High-end, rich content authoring is a delicate struggle, wrestling together highly disparate source media, and arranging all of these elements to form a coherent whole.

Consider the scenario of putting together all of the audio elements for a film soundtrack: this involves transferring all of the music tracks, the ambient sound tracks, the performer's dialogue, and the Foley effects from their original source, remixing or editing all of them, and doing split-second synchronizations to the motion picture elements. This process requires a lot of information about each audio source element, as well as information about other essence associated with it at the moment of playback.

The media industry uses a wide range of source materials, as well as a set of highly varied capture tools with very different constraints (cameras, keyboards, audio input sources, scanners). This wide variety leads to a great deal of time and effort spent converting data into formats that can be used by the wide variety of authoring applications. Other issues include synchronization accuracy for time-based data (film, video, audio, animation); operating system and hardware dependencies for interactive media titles; and download, streaming and playback performance in Internet media applications.

AAF is an industry-driven, cross-platform, multimedia file format that allows interchange of data between AAF-compliant applications. There are two kinds of data that can be interchanged using AAF:

- Audio, video, still image, graphics, text, animation, music, and other forms of multimedia data. In AAF these kinds of data are called ***essence*** data, because they are the essential data within a multimedia program that can be perceived directly by the audience
- Data that provides information on how to combine or modify individual sections of essence data or that provides supplementary information about essence data. In AAF these kinds of data are called ***metadata***, which is defined as data about other data. The metadata in an AAF file can provide the information needed to combine and modify the sections of essence data in the AAF file to produce a complete multimedia program.

The Society of Motion Picture and Television Engineers (SMPTE) has addressed these problems in the dedicated hardware world by creating a set of standards that has worked very well through its history. Computer-based media tool vendors have come up with many varied, mostly proprietary approaches that all have many strengths as well as weaknesses. As digital technology for essence capture, editing, compositing, authoring, and distribution approaches ubiquity, the industry demands better interoperability and standard practices. This document is a specification for a new media industry standard file format, designed to meet information interchange needs.

Incorporated in January 2000, the AAF Association Inc. is a broadly-based trade association intended to promote the development and adoption of AAF technology throughout the media industry. With representatives from many major players in the industry, the AAF Association intends to help deliver the full benefits of digital media to content creators including film, television, and post-production professionals.

Digital Essence File Formats and Issues

Rich media authoring often involves manipulating several types of digital essence files concurrently and managing interactions and relationships between them. These types of essence generally fall into the following categories:

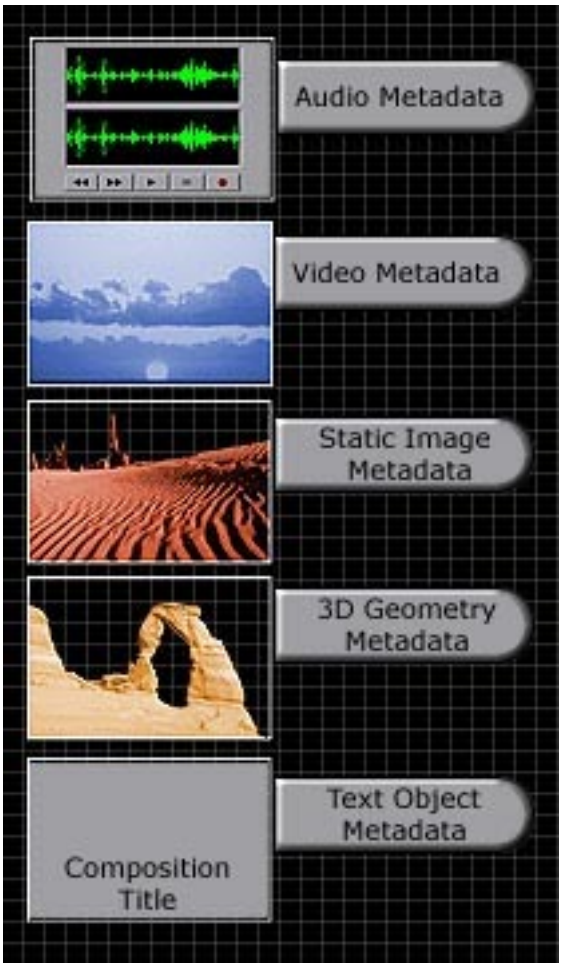
- Motion Picture Film/Video
- Audio
- Still Images
- Animation
- 3-D Geometry
- Text

Despite the relatively small number of categories, the sheer number of available digital essence file formats, each with its own strength or specific quality (i.e. preferred compression codec, optimized file size, preferred color resolution, support for transparency, support for sequential display, analog-to-digital fidelity, or operating system platform), results in many file format-to-file format conversions to produce a high-quality end product.

The following formats are just a few of the many in use today:

- AVI and WAV files are widely used essence containers for video and audio, but they do not support the storage of compositional information or ancillary data such as SMPTE timecode.
- Apple®'s QuickTime® is a technology that incorporates a file format standard optimized for play back and streaming media, with software for handling a variety of media formats, invoking effects, and playing QuickTime files. The metadata support in QuickTime is focused on information needed to play or stream the file.
- Microsoft® DirectX® files are optimized for 3-D images, but do not support other time-varying essence formats as well as AVI and WAV.

- Open Media Framework® (OMF®) Interchange file format, developed by Avid Technology, Inc., is a good step in the direction of interchange, but it has not been widely adopted by other imaging tools vendors.
- The Advanced Streaming Format, or ASF, is a new file format developed by Microsoft for the delivery of streaming essence programs over limited-bandwidth connections. While it meets many of the needs of this market, content creation file formats have differing needs.
- The Adobe Photoshop (PSD) file format is used for storage of still image compositions and related metadata information. While being recognized by many content-creation applications for images, it has limited capabilities for other essence data types.

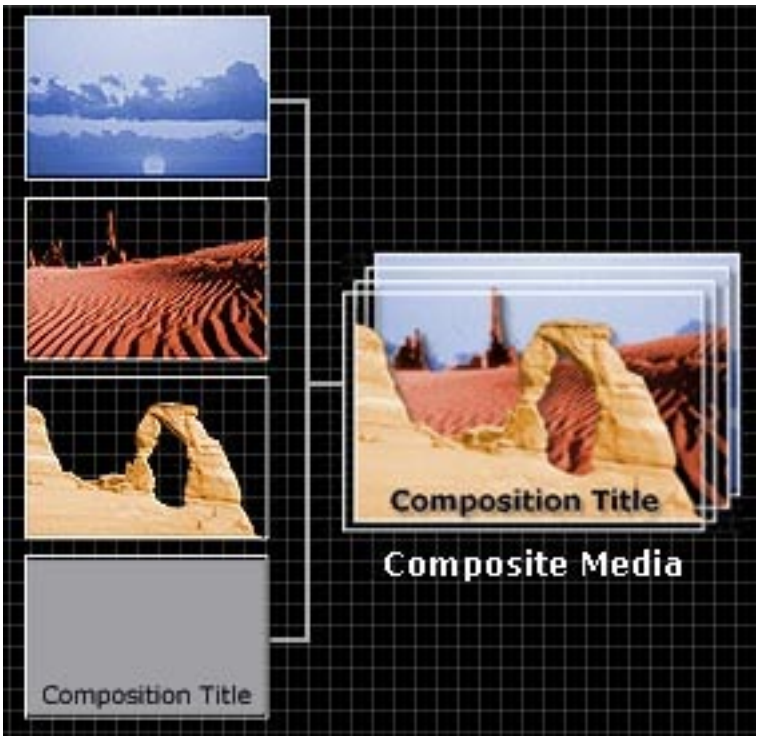


The Advanced Authoring Format helps the content creation and authoring process by addressing the shortcomings of these and other formats. In this way, AAF will allow creative energies to be more focused on the quality of the compositions rather than dealing with unnecessary and painful interchange issues, and allows software development to focus on improvements to the authoring application's feature set.

Digital Essence Authoring

The multimedia content authoring process generally involves 1) opening one or more source essence files, 2) manipulating or editing the essence, and 3) saving the results. Multimedia authoring applications read and manipulate certain types of essence and save the resulting file to their own proprietary format, which is usually specific to a particular hardware platform or operating system. This closed approach generally makes the reuse or repurposing of essence extremely difficult. In particular, the compositional metadata (the data that describes the construction of the composition and not the actual essence data itself) is not transferable between authoring applications.

The Advanced Authoring Format defines authoring as the creation of multimedia content including related metadata. In the authoring process, it is important to record not only the editing and scripting decisions that have been made, but also the steps used to reach the final output, the sources used to create the output, the equipment configuration, intermediate data, and any alternative choices that may be selected during a later stage of the process.



For example, an audio engineer might be recording, editing and mixing the sound for a video. She could record or load the source media tracks, do gain normalization, and then mix the tracks while applying pan, volume, and time compression transforms to the individual tracks. When the work is complete, she can save the files.

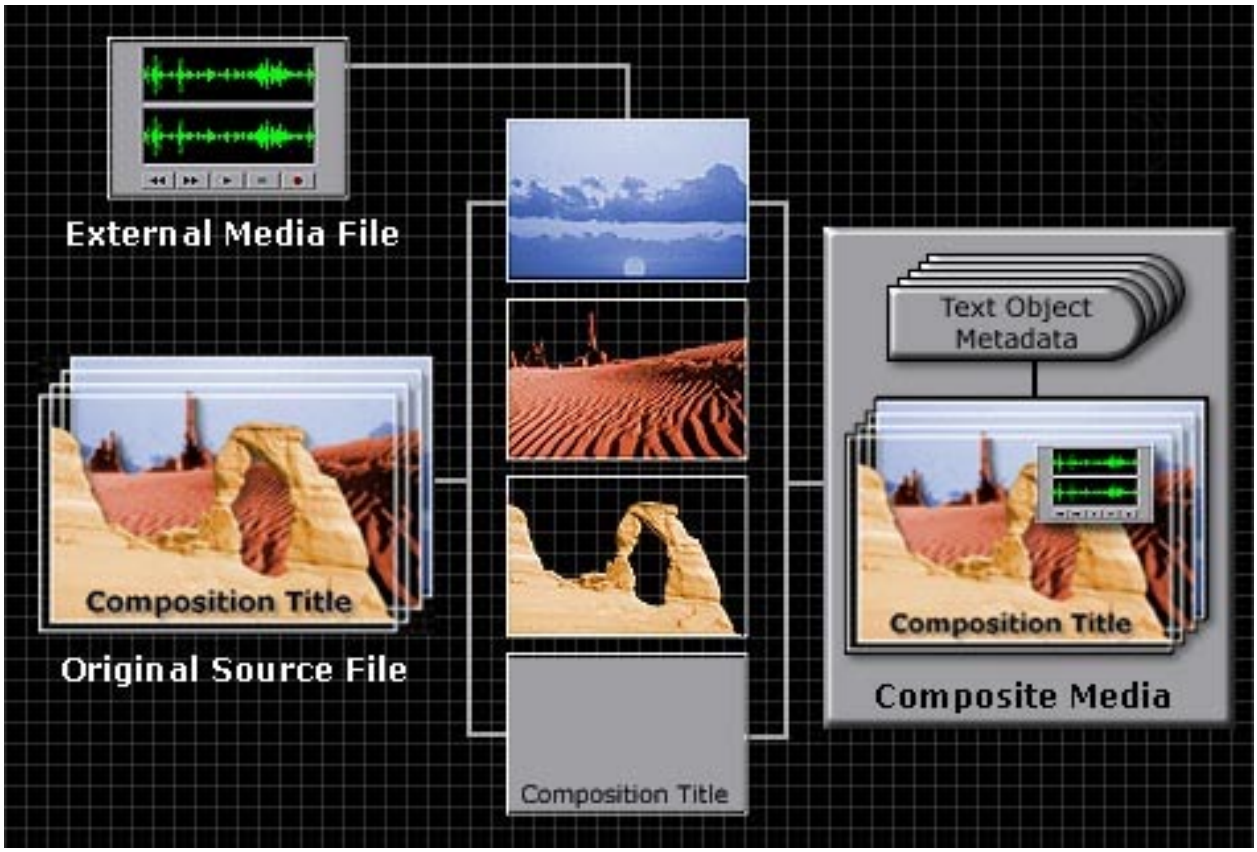
If the authoring application saves the resulting essence information as a single, "flattened" file, then changes cannot be made without going through all of the steps and processes involved. Users may spend much time and energy reconverting and transferring information and reentering instructions, and ultimately rewriting the entire file.

If the authoring application saves the editing and transform data separately from the essence data, then the essence can be changed directly by a sound-editing application without having to open the authoring application. However, the metadata (data used to describe any compositional positioning, layering, playback behavior, editing cut lists, essence mixing, or manipulation) is not accessible unless the authoring application is opened.

In an ideal environment a user would be able to use many different applications and not be concerned with interchange. The essence data and the decisions made in one application would be visible to a user in another application.

The Advanced Authoring Format's unified interchange model enables interoperability between applications. This offers distinct advantages over the current model of separate formats and authoring tools for each essence type:

- The authoring process requires a wide range of applications that can combine and modify essence. Although applications may have very different domains, such as an audio editing application and a 3D graphics animation application, the authoring process requires both applications to work together to produce the final presentation.
- Applications can extract valuable information about the essence data in an AAF file even when it does not understand the essence data format. It can display this information, which allows the user to better coordinate the authoring process.



By enabling interoperability between authoring applications, AAF enables the user to focus on the creative production processes rather than struggling with conversions during the authoring and production phases of the project. Although there are many other issues related to completely transparent interoperability, the significant benefit that AAF provides to end users is assurance that compositions output by AAF-compliant applications will be accessible by the right tool for the job, without risk of being "stranded" by proprietary file format restrictions.

The authoring applications that can use AAF for interchange include:

- Television studio systems, including picture and sound editors, servers, effects processors, archiving, and broadcast automation systems
- Post-production systems, including digitization, offline editing, graphics, compositing, and rendering systems
- Image manipulation applications, including palettizing tools
- Audio production/engineering systems, including multitrack mixers and samplers
- 3D rendering systems
- Multimedia content creation systems, including scripting, cataloging, titling, logging, and content repackaging and repurposing applications

- Image and sound recording equipment, including cameras and camcorders, scanners, telecines, sound dubbers, disk recorders, and data recorders

Digital Essence Interchange

The Advanced Authoring Format provides applications with a mechanism to interchange a broad range of essence formats and metadata, but applications may have interchange restrictions due to other considerations. For this reason, it is important to understand the different kinds of interchange possible and to describe the various levels of interchange between authoring applications.

The following is a general description of the levels of AAF interchange that applications can adopt. For detailed information on a specific product's AAF support level, see that product's documentation.

- Interchange of limited set of essence data
- Interchange of broad set of essence data with some related metadata
- Interchange of essence data and rich set of metadata including compositions but having limited support for some essence types
- Full interchange of all essence types and all metadata described in this specification and preserving any additional private information stored in the AAF file

The Advanced Authoring Format is designed to be a universal file format for interchange between systems and applications. It incorporates existing multimedia data types such as video, audio, still image, text, and graphics. Applications can store application-specific data in an AAF file and can use AAF as the application's native file format. AAF does not impose a universal format for storing essence content data. It has some commonly used formats built in, such as CDCI and RGBA images, WAV and AIFC audio, but also provides an extension framework for new formats or proprietary formats. As standard formats for essence are adopted by groups such as the SMPTE and the Audio Engineering Society (AES), AAF will provide built-in support for these formats.

Data Encapsulation

At its most basic level, AAF encapsulates and identifies essence data to allow applications to identify the format used to store essence data. This makes it unnecessary to provide a separate mechanism to identify the format of the data. For example, AAF can encapsulate and label WAV audio data and RGB video data.

Compositional Information

The actual audio, video, still, and other essence data makes up only part of the information involved in authoring. There is also compositional information, which describes how sections of audio, video or still images are combined and modified. Given the many creative decisions involved in composing the separate elements into a final presentation, interchanging compositional information as well as essence data is extremely desirable, especially when using a diverse set of authoring tools. AAF includes a rich base set of essence effects (such as transitions or chroma-key effects), which can be used to modify or transform the essence in a composition. These effects use the same binary plug-in model used to support codecs, essence handlers, or other digital processes, used to process the essence to create the desired impact.

Media Derivation

One of AAF's strengths is its ability to describe the process by which one kind of media was derived from another. AAF files contain the information needed to return to an original media source in case it needs to be used in a different way. For example, when an AAF file contains digital audio and video data whose original source was film, the AAF file may contain descriptive information about the film source, including edgecode and in- and out-point information from the intermediate videotape. This type of information is useful if the content creator needs to repurpose material, for instance, for countries with different television standards. Derivation information can also describe the creation of computer-generated essence: if a visual composition was generated from compositing 3D animation and still images, the AAF file can contain the information to go back to the original animation sources and make changes without having to regenerate the entire composition.

Flexibility and Efficiency

The Advanced Authoring Format is not designed to be a streaming essence format, but it is designed to be suitable for native capture and playback of essence, and to have flexible storage of large data objects. For example, AAF allows sections of data to be broken into pieces for storage efficiency, as well as including external references to essence data. AAF also allows in-place editing; it is not necessary to rewrite the entire file to make changes to the metadata.

Extensibility

The Advanced Authoring Format defines extensible mechanisms for storing metadata and essence data. This ensures that AAF will be able to include new essence types and essence data formats as they become commonly used. The extensibility of the effects model allows ISVs or tool vendors to develop a rich library of new and engaging effects or processes to be utilized with AAF files. The binary plug-in model gives AAF-compliant applications the flexibility to determine when a given effect or codec has been referenced inside of the AAF file, to determine if that effect or codec is available, and if not, to find it and load it on demand.

Digital Essence Delivery

In contrast to authoring systems, delivery systems and mechanisms are primarily used to transport and deliver a complete multimedia program. Although it would be ideal to use a single format for both authoring and delivery, these processes have different requirements. With authoring as its primary focus, AAF's metadata persistence enables optimal interchange during the authoring process. By allowing the content files to be saved without the metadata (that is by stripping out the metadata or flattening the file), AAF optimizes completed compositions for delivery, without restricting features needed for authoring.

From a technical standpoint, digital media content delivery has at least two major considerations: 1) target playback hardware (TV, audio equipment, PC) and 2) distribution vehicle (Film, Broadcast TV, DVD and other digital media, and network). When content is delivered, the delivery format is usually optimized for the particular delivery vehicle (DVD, DTV, and others), and the essence data is often compressed to conserve space or enable fast download.

We expect that the content created using AAF in the authoring process will be delivered by many different vehicles, including broadcast television, packaged media, film, and networks. These delivery vehicles will use data formats such as baseband video, MPEG-2 Transport Stream, QuickTime 4, and the Advanced

Streaming Format (ASF). These formats do not need the rich set of metadata used during the authoring process, and can be optimized for delivery by stripping out this metadata or flattening the file.

AAF File Format

The Advanced Authoring Format is a structured container for essence and metadata that provides a single object-oriented model to interchange a broad variety of essence types including video, audio, still images, graphics, text, MIDI files, animation, compositional information and event triggers. The AAF format contains the essence assets and preserves their file-specific intrinsic information, as well as the authoring information (in- and out-point, volume, pan, time and frame markers, and so on) involving those essence assets and any interactions between them.

To meet the rich content authoring and interchange needs, AAF must be a robust, extensible, platform-independent structured storage file format, able to store a variety of raw essence file formats and the complex metadata that describes the usage of the essence data, and must be capable of efficient playback and incremental updates. As the evolution of digital media technology brings the high-end and low-end creation processes into convergence, AAF must also be thoroughly scalable and usable by the very high-end professional applications as well as consumer-level applications.

Structured storage, one of the technical underpinnings of AAF, refers to a data storage architecture that uses a "file system within a file" architecture. This container format is to be a public domain format, allowing interested parties to add future developments or enhancements in a due process environment. Microsoft is specifically upgrading the core technology compound file format on all platforms (Microsoft Windows®, Apple® Macintosh®, UNIX®) to address the needs of AAF, for instance, files larger than 2 gigabytes and large data block sizes.

Other important features of AAF include:

- Information about the original sources retained by AAF, so that the resulting edited essence can be traced back to its original source
- References to external essence files, with files located on remote computers in heterogeneous networks
- An extensible video and audio effects architecture with a rich set of built-in base effects
- Support for a cross-platform binary plug-in model

AAF Specification Development

The AAF Association will ensure that AAF remains an open public standard without bias or prejudice. The AAF Association will also drive the continued advancement of the AAF technology in the future.

2. Introduction to Objects, Packages, and Essence Data

Advantages of 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 labeling different kinds of information
- Objects make it possible to treat different items in the same way for attributes they share. With an AAF file:
    - one can find out the duration of video data, audio data, MIDI file data, or animation data, without having to deal with their differences.
    - one can play audio or video data either contained within the AAF object, or stored in an external file and referenced by the AAF object.
- When the information becomes very complex, objects provide a mechanism to describe it in a structured way. Some simple summary information can be easily obtained.

Although simple interchange is easily done without using an object model, the object model provides a framework to handle more complex interchanges. The structured approach of the object model makes it easier to describe complex data.

Object Model

This interchange format provides an object-oriented mechanism to interchange multimedia information.

Object-oriented interchange has the following advantages:

- Objects provide a framework for containing and labeling different kinds of information
- Objects make it possible to treat different items in the same way for attributes they share.
- One can find out the duration of video data, audio data, MIDI file data, or animation data, without having to deal with their differences.
- One can play audio or video data either contained within an object, or stored in an external file and referenced by an object.
- When the information becomes very complex, objects provide a mechanism to describe it in a structured way. Some simple summary information can be easily obtained.

Although simple interchange is easily done without using an object model, the object model provides a framework to handle more complex interchanges. The structured approach of the object model makes it easier to describe complex data.

Header Object

An interchange file contains:

- Packages and the objects they have
- Essence data
- One Header object and its related objects

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

Each object in an interchange file belongs to a class. The class defines the 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 standard defines classes using a class hierarchy, in which a subclass inherits the properties of its superclass. All classes are subclasses of the InterchangeObject class.

Using an object-oriented mechanism makes it easier to extend this interchange standard, and it provides a flexible framework that will work for interchange between applications with disparate data models.

The storage wrapper format has exactly one Header object in an AAF file. The Header object owns all other objects in the file This ownership relationship is specified by the StrongRef, StrongReferenceVector, and StrongReferenceSet property types.

There shall be exactly one Header object. The Header object shall specify the following

a) Byte order used to store data in the file

b) Date and time that the file was last modified; if the file has not been modified the date and time that the file was first created shall be specified as the modification date and time

c) ContentStorage object that has all Packages (Packages) and EssenceData objects

d) Dictionary object that has all definitions

e) Version number; files conforming to this document shall specify a version number 1.0; future revisions of this document may specify a higher version number

f) An ordered set of identification objects that provide information about the applications that created or modified the file

Figure 2-1 illustrates a typical AAF file.

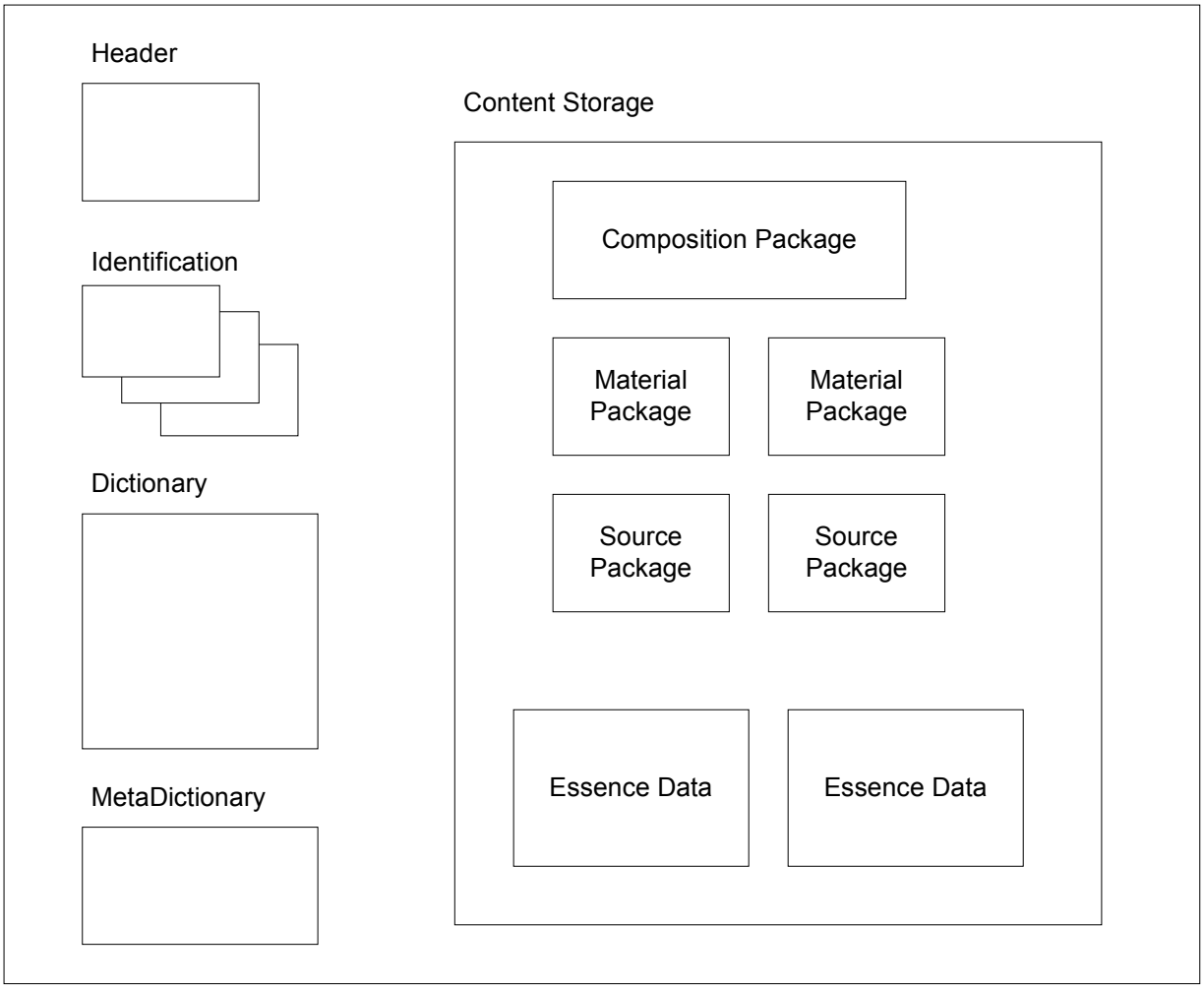


Figure 2-1 — Typical AAF file with Packages and EssenceData objects

Dictionary

The Dictionary has ClassDefinitions, PropertyDefinitions, TypeDefinitions, DataDefinitions, ParameterDefinitions, and EffectDefinitions. If an AAF file contains any classes, properties, types, data definitions, parameter definitions, or effects that are not defined by this document, the Dictionary shall have the definition for these extensions to this document.

Essence Data and Metadata

Essence data is picture, sound, and other forms of data that can be directly perceived. *Metadata* is data that describes essence data, performs some operation on essence data, or provides supplementary information about the essence data. For example, digitized sound data is essence data, but the data that describes its format, specifies its duration, and gives it a descriptive name is metadata.

Much of the creative effort that goes into a multimedia presentation is represented by metadata. How one section transitions into another, how special effects modify the data we perceive, and how all the different kinds of primary data are related to each other (such as synchronizing picture and sound) are all represented as metadata. This interchange format provides a way to interchange this rich set of metadata.

Packages

A Package (Package) is an object that has a SMPTE universal label and describes essence. A Package is

A Package (Package) is an object that has a universal identifer and consists of metadata. Packages describe how essence data is formatted or how separate pieces of essence data are combined or composed. Packages 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.

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

A Package can describe more than one kind of essence. For example, a Package can have audio, video, still image, and timecode data. A Package has one or more Slots. Each Slot can describe only one kind of essence data. A Slot can be referenced from outside of the Package. For example, a Package can have two Slots with audio, one Slot with video, three Slots with still images, and two Slots with timecode. Each Slot in a Package has a SlotID that is unique within the Package. To reference the essence data in a Slot, the PackageID and the SlotID is used.

The following sections describe:

- Kinds of Packages

- How a Package is associated with essence data
- Kinds of Slots
- How one Package references another
- How time-varying essence data, such as audio and video, is described in Packages and slots
- How other kinds of essence data, such as still images, are described in Packages and slots

Kinds of Packages

There are different kinds of Packages, which have metadata that is used in different ways.

Physical Source Packages, which describe media that was used to generate the digital essence data. For example a Physical Source Package can describe a videotape that was digitized to create digital video data and digital audio data

File Source Packages, which describe the digital essence data and provide a mechanism to locate the digital essence data

Material Packages, which provide information that helps locate the File Source Packages

Composition Packages, which contain the creative decisions about how essence data is to be presented

Physical Source Packages and Other Kinds of Source Packages

Physical Source Packages 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.

File Source Packages

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

- The format used to store the digital essence data, such as WAVE and AIFC for audio and RGBA, MPEG, and JPEG 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

Material Packages

Material Packages provide an association between Composition Packages, which describe the creative decisions, and File Source Packages, which describe and identify the essence data. Material Packages

insulate the Composition Package from the detailed information about how the essence data is stored. Material Packages 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 used to or in the kind of format used to store it
- Effect descriptions, such as color correction, that do not represent a creative decision but instead correct an error in essence acquisition or conversion

Composition Packages

Composition Packages describe the creative editing and composing decisions that combine individual bits of essence data into a presentation. A Composition Package 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

Package Kind Summary

Table 2-1 summarizes the different kinds of Packages.

Table 2-1 – Kinds of Packages

Kind of Package	Function
Composition Package	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
Material Package	Collect and possibly synchronize related essence data; provides indirect access to essence data, which is independent of storage details
File Source Package	Provides direct access to and describes format of digital essence data that is (or can be) stored in a computer file
Physical Source Package	Describes physical media such as a videotape or film

A CompositionPackage describes how to combine individual essence elements to produce a program.

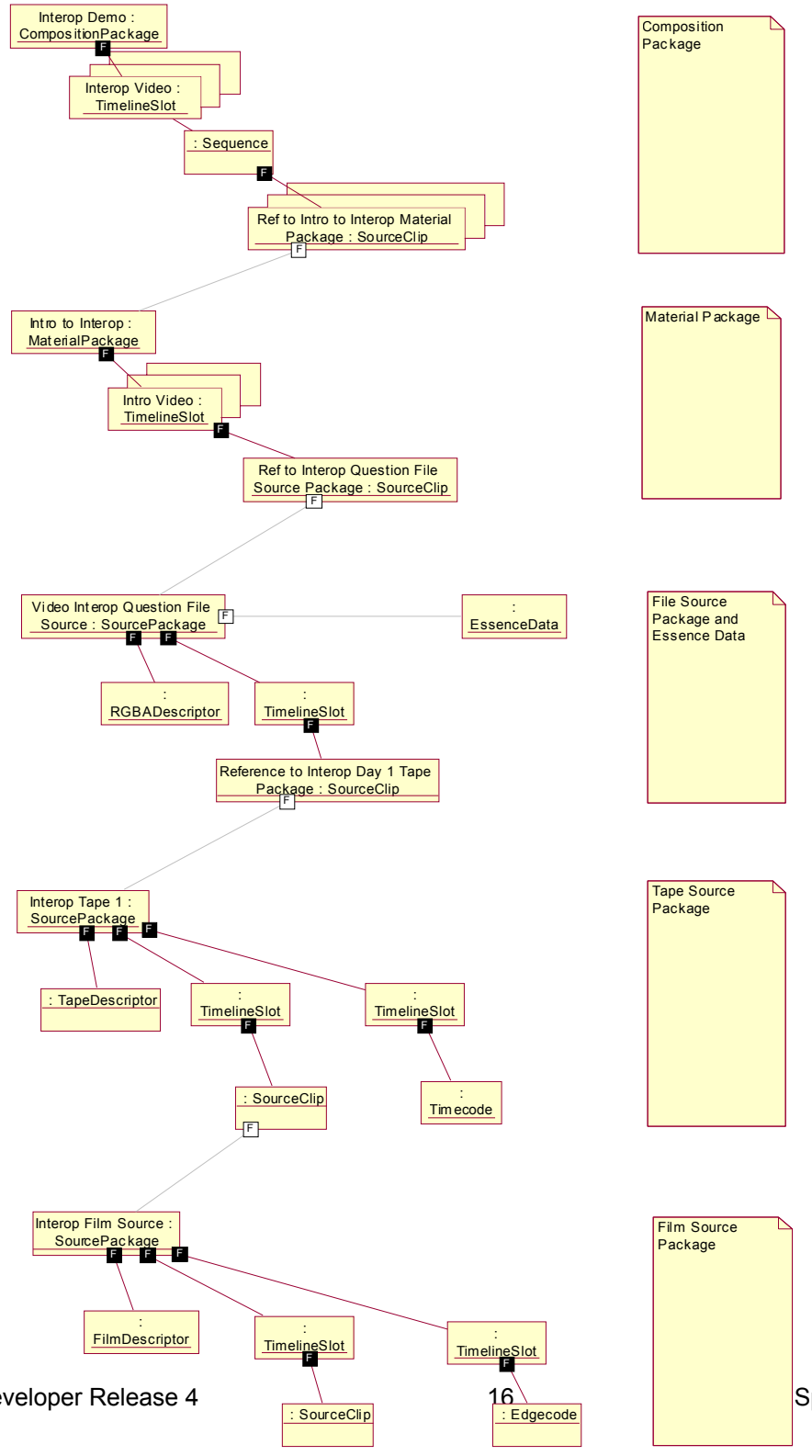

Interop Demo : CompositionPackage
Interop Video : TimelineSlot
: Sequence
Ref to Intro to Interop Material Package : SourceClip
Composition Package
Intro to Interop : MaterialPackage
Intro Video : TimelineSlot
Ref to Interop Question File Source Package : SourceClip
Material Package
Video Interop Question File Source : SourcePackage
: EssenceData
: RGBADescriptor
: TimelineSlot
Reference to Interop Day 1 Tape Package : SourceClip
File Source Package and Essence Data
Interop Tape 1 : SourcePackage
: TapeDescriptor
: TimelineSlot
: TimelineSlot
: SourceClip
: Timecode
Tape Source Package
Interop Film Source : SourcePackage
: FilmDescriptor
: TimelineSlot
: TimelineSlot
: SourceClip
: Edgecode
Film Source Package

Figure x-x: Relationships Betweeen Packages Describe Essence Derivation

A SourcePackage describes the format and derivation of essence stored in digital forms or stored on physical media, such as videotape or film. A SourcePackage shall have a EssenceDescriptor object that specifies the format of the essence. If the SourcePackage describes essence stored in a computer file, it shall have a FileDescriptor and is described as a File SourcePackage. If the SourcePackage describes essence stored on physical media, such as videotape or film, it is described as a Physical SourcePackage.

A MaterialPackage describes how individual essence elements are to be synchronized or interleaved. MaterialPackages provide a level of indirection between CompositionPackages and SourcePackages.

A Package shall have a unique identification, which is called a PackageID. A PackageID has an globally unique value and is specified using the PackageID property type, which holds a SMPTE UMID, a 32-byte unique identifier.

A Package shall have one or more Slots Each Slot describes an element of essence that can be referenced. A Slot shall specify an integer identifier, which is called a SlotID. Each Slot shall have a Segment object. A Slot can be a TimelineSlot, a StaticSlot, or an EventSlot. A TimelineSlot describesessence that varies over time, such as audio or video data. A StaticSlot describes essence that has a value that has no relationship to time, such as a static image. An EventSlot describes essence that has values at specified points in time, such as instructions to trigger devices at specified times or instructions to display an interactive region during a specified time in a program.

A Package can reference another Package to describe the source or derivation of the essence. A Package refers to another Package by having a SourceClip object. An SourceClip object has a weak reference to a Package using its identifying PackageID value; shall identify a Slot within the referenced Package with a SlotID value; and when referencing a TimelineSlot shall specify an offset in time within the referenced TimelineSlot.

If a Package owned by a ContentStorage object has a reference to a second Package, then the ContentStorage shall also own the second Package.

A SourcePackage may desribe derivation metadata, which describes a physical source that was used to generate the essence component. If the SourcePackage describes derivation metadata, then the SourcePackage shall a SourceClip that specifies the PackageID of the Physical SourcePackage that describes the physical media source. If there is no previous generation of physical media source, then the File SourcePackage shall has a SourceClip that specifies a SourceID value of 0, a SourceSlotID value of 0, and, in TimelineSlots, a StartTime value of 0.

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

a) One Package is a duplicate of the other

b) One Package is a modified version of the other subject to the restrictions on modifications described in this clause

The type of Package determines the kind of modifications that can be made to it and still retain its identity.

The information in a file source Package describing the format of essence is immutable and cannot be modified. Modifications to a file source Package 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 SourcePackage 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 SourcePackage describing physical media may describe timecode information and may describe edgecode information. This timecode and edgecode information describes the timecode and edgecode that is present in the physical media.

A MaterialPackage may be modified by replacing a reference to a SourcePackage with a reference to another SourcePackage or it may be modified by inserting a reference to an alternate SourcePackage. The modifications are subject to the restriction that the replacement or alternate SourcePackages should be representations of the same physical media as the originally referenced SourcePackage.

A CompositionPackage may be modified in any way.

Components

Components are essence elements. A component in a TimelineSlot has a duration expressed in edit units. The relation between edit units and clock time is determined by the edit rate of the TimelineSlot 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:

a) SourceClip which references a section of a Slot in another Package; for example a SourceClip in a TimelineSlot can describe video data

b) Sequence which specifies that its set components are arranged in a sequential order; in a TimelineSlot, the components are arranged in sequential time order

d) Effect 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

e) Filler which defines an unspecified value for its duration

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. A Transition causes the preceding and following Segments to be overlapped in time and to be combined by the Transition's effect.

File SourcePackages and EssenceData objects

A File SourcePackage 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:

a) 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 SourcePackage separate from the essence provides more flexible storage options while still allowing the composition to use the same access mechanism.

b) Audio and video data or other essence may be used in more than one CompositionPackage and these CompositionPackages can be in different files. Separating the File SourcePackage from the essence means that only the information in the File SourcePackage needs to be duplicated.

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

1. In a EssenceData object in the same file as the SourcePackage
2. In a EssenceData object in a different file, which must contain a duplicate of the SourcePackage
3. In a data file that is not a wrapper file

The PackageID connects the File SourcePackage to the essence if it is stored in a EssenceData object.. The File SourcePackage and its corresponding EssenceData object have the same PackageID value.

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. However, since there is no PackageID stored with this essence, it is difficult to identify a data file if the file has been moved or renamed.

How File Source Packages are Associated with Digital Essence Data

File Source Packages and Digital essence data are associated by having the same PackageID. Since the ContentStorage object that has all the Packages and Essence data objects in the file, applications can find the Essence data object associated with a File Source Package by searching for the appropriate PackageID.

Digital essence data may also be stored in a non-container file. In some cases, the digital essence file format has a mechanism to store a PackageID. In these cases, applications can still use the PackageID to associate the File Source Package with the digital essence data. When there is no mechanism to store a PackageID with the digital essence data, interchange files use specialized locator objects to associate a File Source Package with the digital essence data.

Kinds of Slots in Packages

Each kind of Slot defines a specific relationship between essence data and time. This standard currently defines the following kinds of Slots:

- Static Slot
- Timeline Slot
- Event Slot

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

A Timeline Slot 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 Event Slot describes essence that has an unpredictable relationship with respect to time. GPI (General Purpose Interface) events and MIDI are examples of irregularly timed events.

Table 2-2 summarizes the different kinds of Slots.

Table 2-2: Kinds of Slots

Kind of Slot	Function
Static Slot	Describes essence data that has no specific relationship to time, such as static images or static text.
Timeline Slot	Describes essence data that has a fixed or continuous relationship with time, such as audio, film, video, timecode, and edgecode
Event Slot	Describes essence data that has an irregular relationship with respect to time, such as GPI events, MIDI, interactive events, and user comments associated with specific times

How and Why One Package Refers to Another Package

A Package can have a reference to a Slot in another Package. Packages may reference other Packages, for example, to identify a original and unmodified essence data. A Source Reference identifies the referenced Package by specifying its PackageID and specifies the associated Slot by specifying its PackageID.

If a Package describes the original essence data, it has Source Clips that do not have a SourceID property.

Source Clips in Physical Source Packages identify the PackageID of a previous physical source of physical media. For example, a videotape Source Package has a Source Clip that specifies the Physical Source Package describing the film that was used to generate the videotape.

Source Clips in File Source Packages specify the PackageID of a Physical Source Package.. For example, a video File Source Package has a Source Clip that specifies the Physical Source Package describing a videotape used to generate the digital video data.

Source Clips in Composition Packages specify the PackageID of the Material Package, and are used to represent pieces of digital essence data. The Material Package provides a level of indirection between the digital essence data and the objects that refer to them.

A Composition Package is the result of the creative editing decision-making process. The original and unmodified essence data is collection of digital essence data combined and modified as described by the Composition Package.

In summary, Packages 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 2-1 illustrates how a Source Clip in a Composition Package references a Material Package. The Material Package references the File Source Package, which references the tape Package. Finally, the tape Package references the film Package.

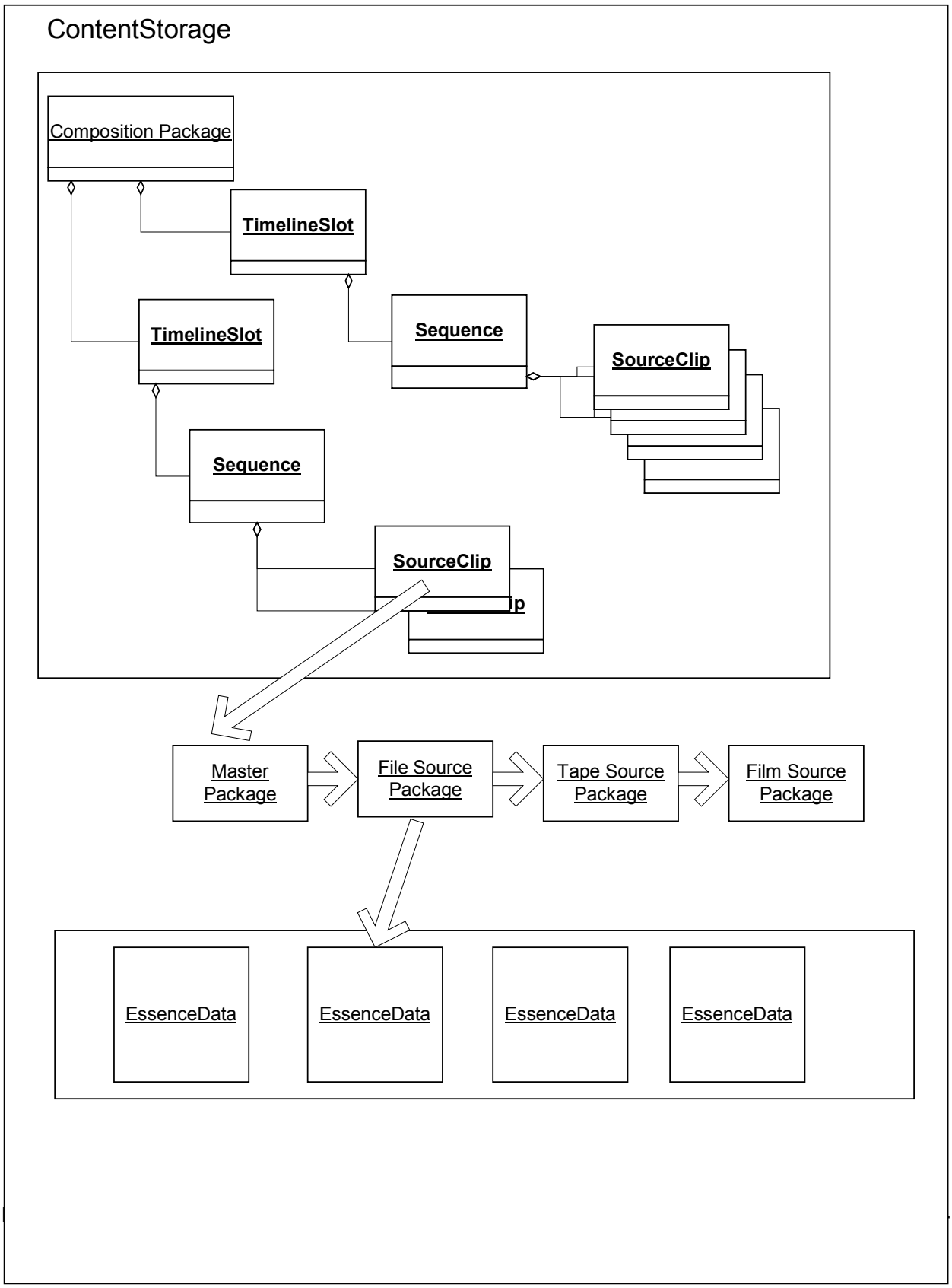

ContentStorage
Composition Package
TimelineSlot
TimelineSlot
Sequence
SourceClip
Sequence
SourceClip
Master Package
File Source Package
Tape Source Package
Film Source Package
EssenceData
EssenceData
EssenceData
EssenceData

Figure 2-1 Package references define the derivation of essence

Static Image Essence in Packages

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

In a Static Slot, a Source Clip refers to another Static Slot by specifying its PackageID and SlotID but does not specify an offset in time or a duration as in a Timeline Slot.

Composition Packages that have only Static Slots specify the editing decisions involved in composing static images. Figure 2-2 illustrates a typical Composition Package 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.

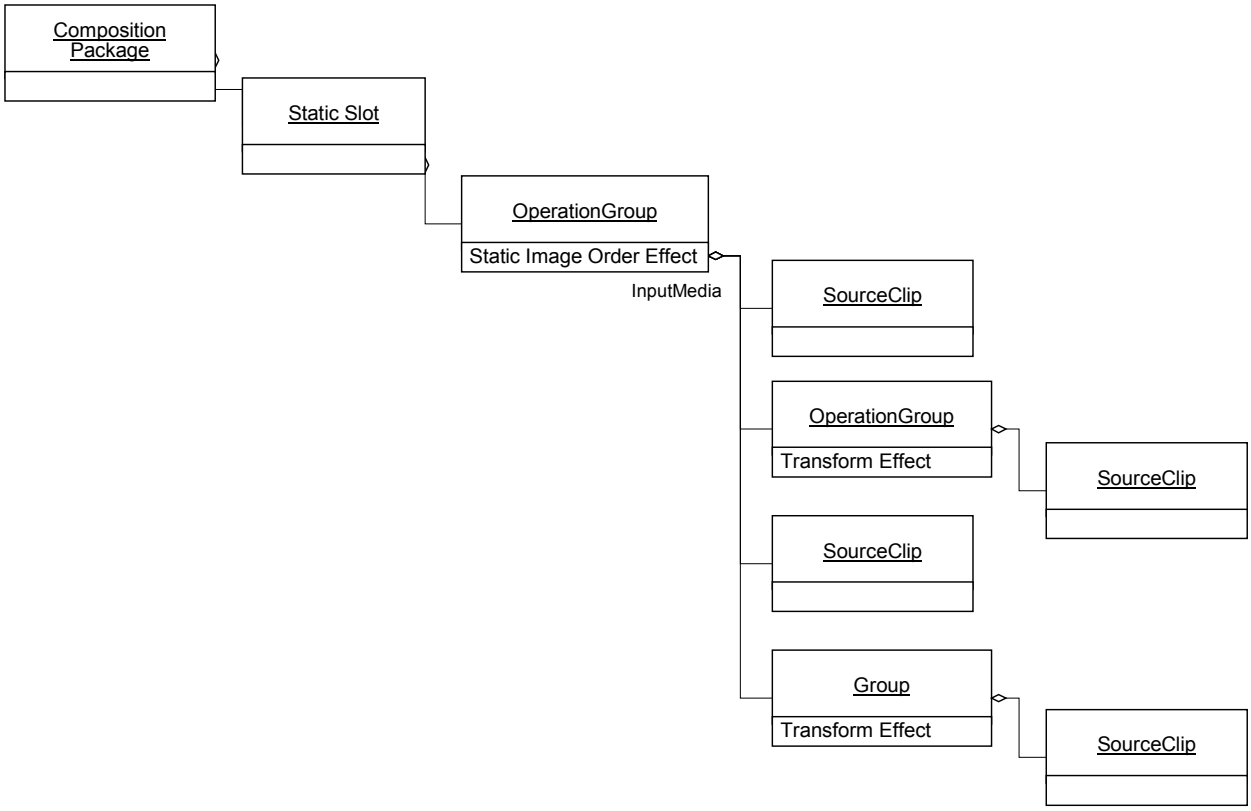


Figure 2-2 Composition Package with Static Essence

Time-varying Video and Audio Essence in Packages

Audio and video essence data is represented in Timeline Slots. 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 framerate of approximately 29.97 frames per second. Each Timeline Slot specifies an edit rate which defines the unit of time for objects referred to by that particular Timeline Slot. 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 Timeline Slots, a source clip references a Timeline Slot in another Package by specifying its PackageID and Timeline Slot ID number and by specifying a subsection of the Timeline Slot with an offset in time and a duration. For example, a source clip in a composition Package can reference a subsection of audio or video data by referencing a section of that essence data's Material Package.

A simple Composition Package has audio and video Timeline Slots where each Timeline Slot has a sequence of source clips. The sequence specifies that the source clips should be played consecutively, one after another. Each Timeline Slot in the Composition Package is to be played simultaneously with other co-timed Timeline Slots.

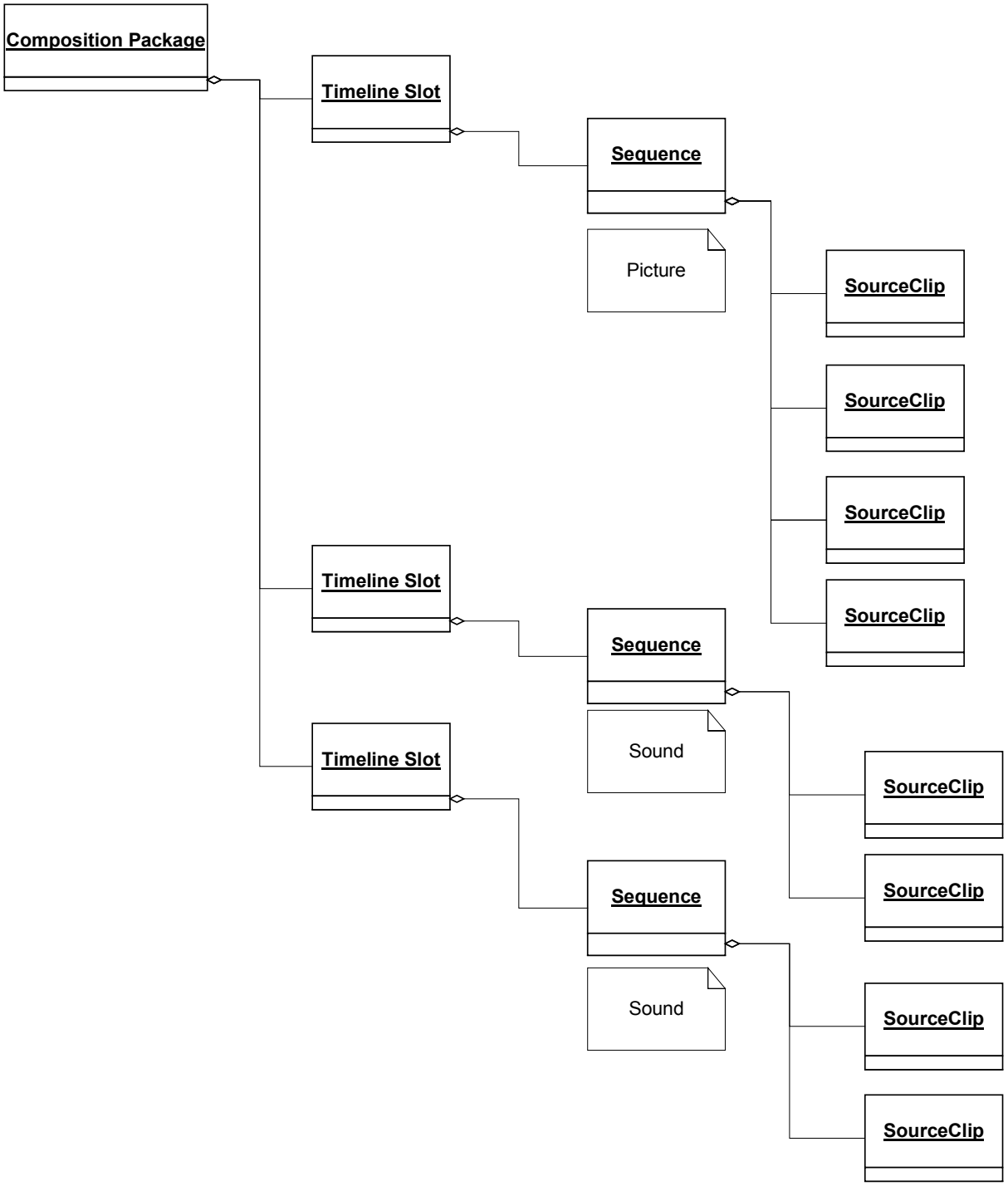


Figure 2-3: Structure of Composition Package with Timeline Slots

Each source clip 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 source clip can be thought of as a section of videotape or film to be spliced with other sections. By examining the section itself, may listen to its audio or view its frames, but one can not tell where it will appear in the finished piece until the preceding sections in the sequence are examined.

Figure 2-4 illustrates how the source clips in a sequence appear in a timeline view of a composition.

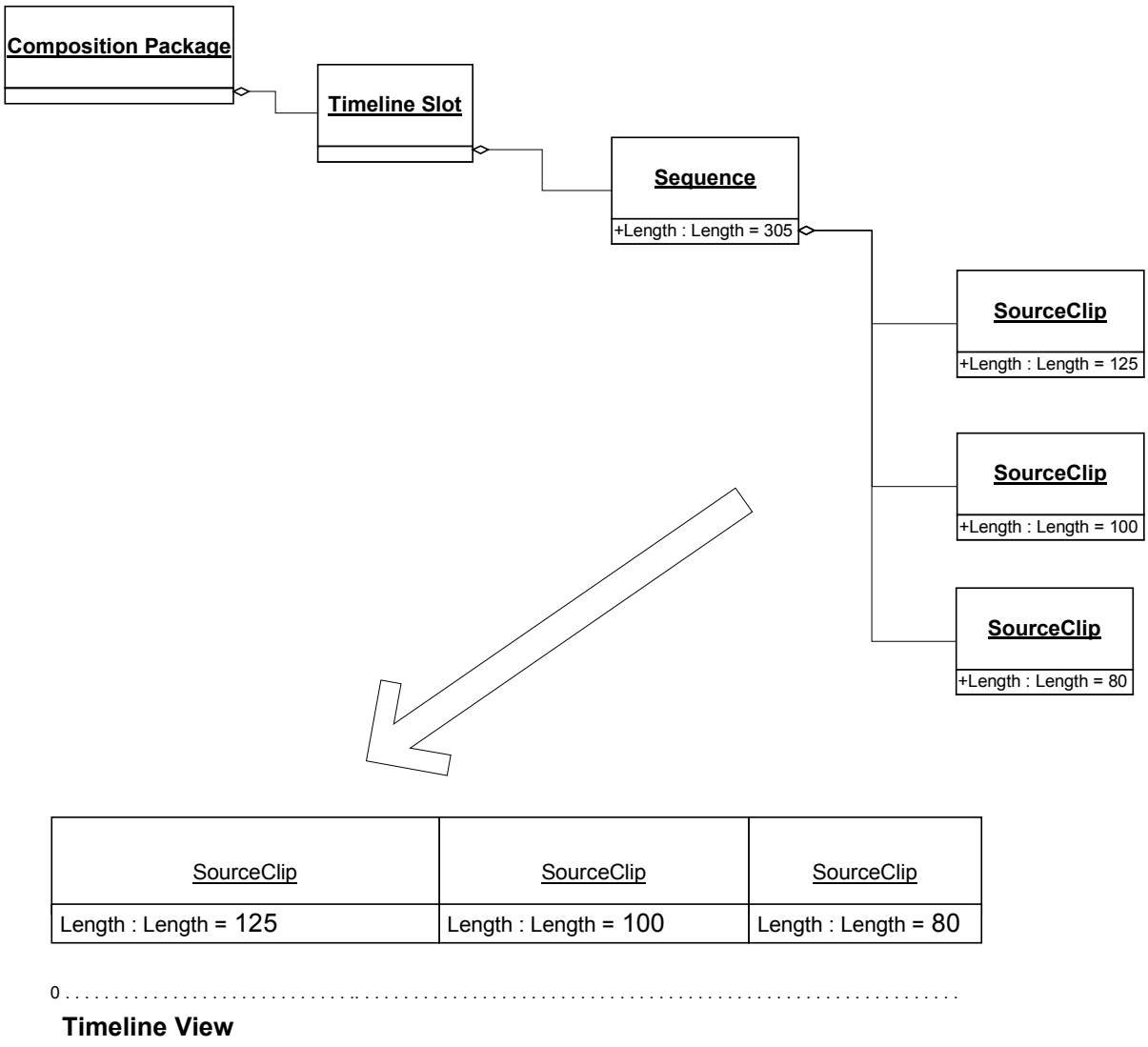


Figure 2-4: Sequence in Composition Package

Event Data in Packages

Events typically specify an action or define a behavior that takes place at a specified time. Typically, Event Slots specify events that are associated with the time-varying essence in a parallel Timeline Slot. Each Event Slot describes one kind of event. In each Event Slot, no two events should occur at the same time. Figure 2-5 illustrates a Composition Package that has a Timeline Slot with video essence data, an Event Slot that has comments defined for specific points in time, and an Event Slot that defines interactive areas in the resultant image.

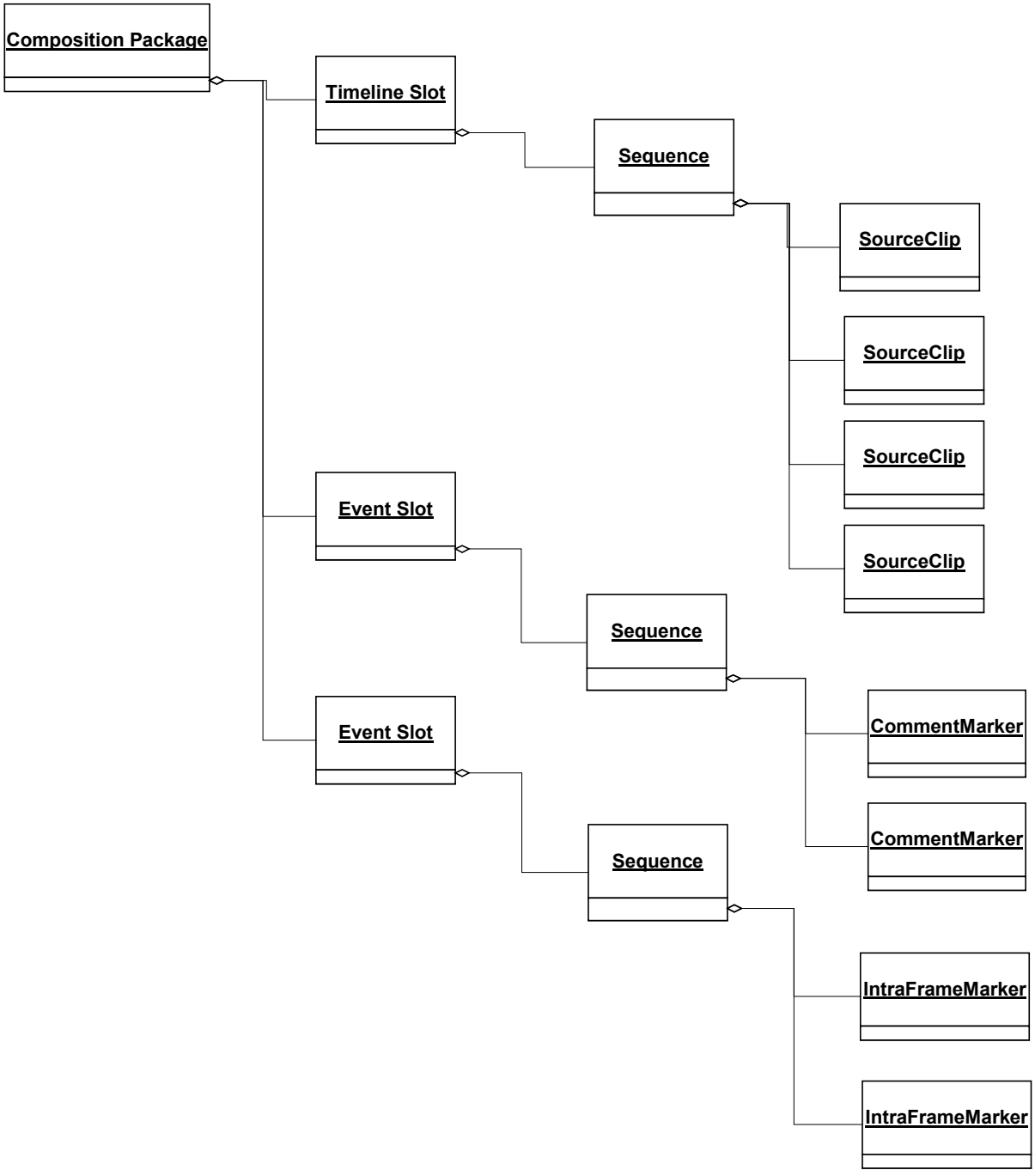


Figure 2-5 Composition Package with Video Essence and Events

3. Composition Packages

This chapter describes the AAF Composition Package (Package), which is the AAF object that describes editing information.

Composition Package Basics

Composition Packages (Packages) describe the creative editing and composing decisions that combine the individual pieces of essence data into a unified program. A Composition Package 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 Packages are designed to be capable of describing creative human decisions; consequently, their complexity is limited only by the limits of our imagination.

Composition Packages do not directly reference the essence data that they combine to form a program. Composition Packages reference the basic essence data with Source Clips that identify the Material Package and File Source Packages that describe the essence data. The Material Packages and File Source Packages have the information that is used to read and write the essence data.

In addition to Source Clips, Composition Packages 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 Package.

Composition Packages consist of Slots that describe a set of editing decisions that can be referenced from outside of the Package. Slots in a Composition Package 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 Package 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 Package could have two Timeline Slots describing audio data and one Timeline Slot describing video data. The edited program produced by this Composition Package would consist of three synchronized tracks: two audio and one video.

Another simple Composition Package could have one Static Slot , describing a single static image composed by combining a set of static images.

A complex Composition Package could have Timeline Slots, Static Slots, and Event Slots. The edited program produced by this Composition Package could have elements from each of these Slots combined in some form by the objects in the Composition Package.

Timeline Slots

Timeline Slots typically have a Sequence of audio or video segments. Each segment can consist of a simple Source Clip or a complex hierarchy of Effects. Figure 3-1 is a containment diagram of a Composition Package that has only Timeline Slots with audio and video data.

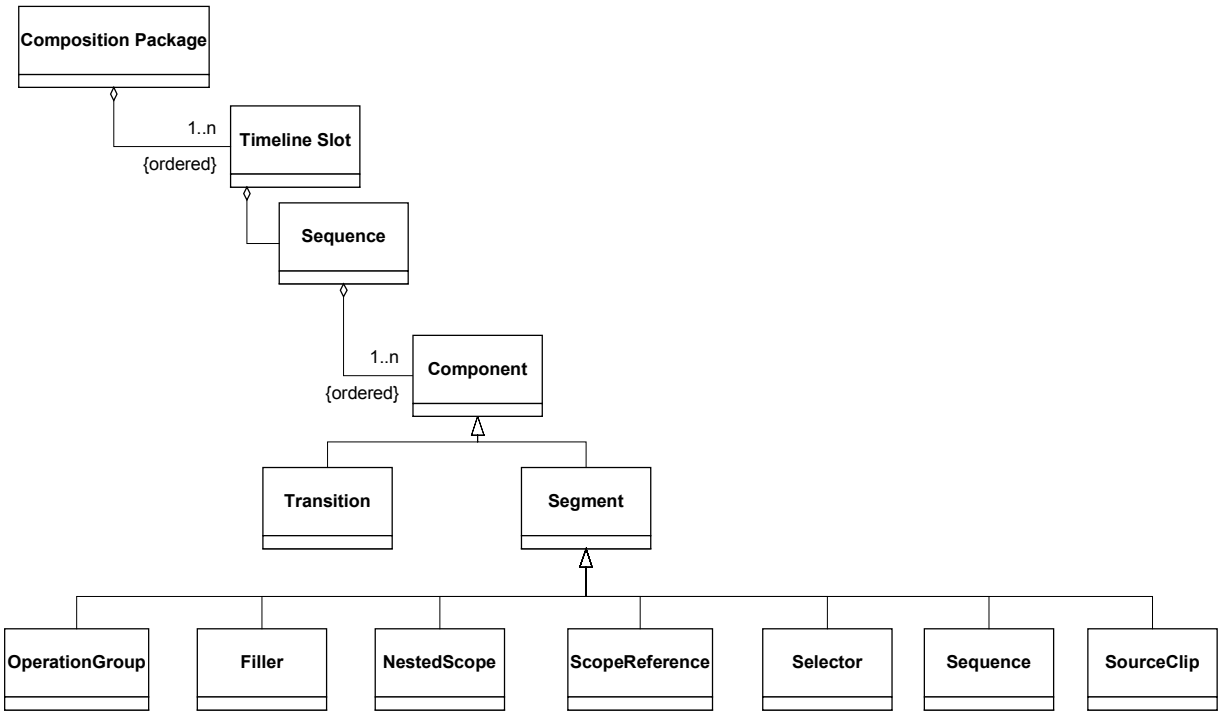


Figure 3-1: Containment Diagram for Composition Package with Timeline Slots

Sequences

A Sequence can have the following components:

- Source Clip: Specifies a section of essence or other time-varying data and identifies the Slot in another Package or within the same Package 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.
- EffectDefinition property of an Effect: Specifies an effect to be used in a Composition Package; 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 Package 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 Package 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 Package that a Segment starts is determined by the Components that precede it in the Sequence.

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 3-2 illustrates the Sequence containment showing Transition, which itself has an Effect.

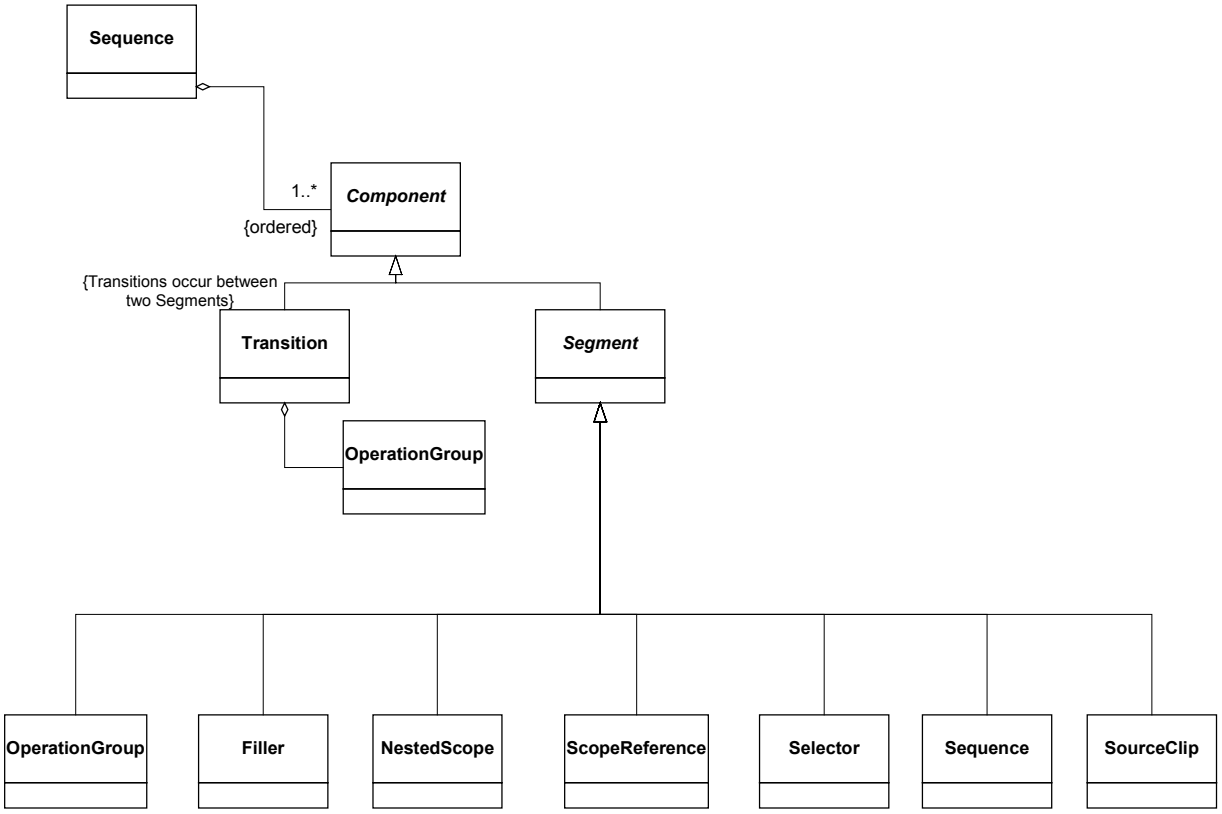
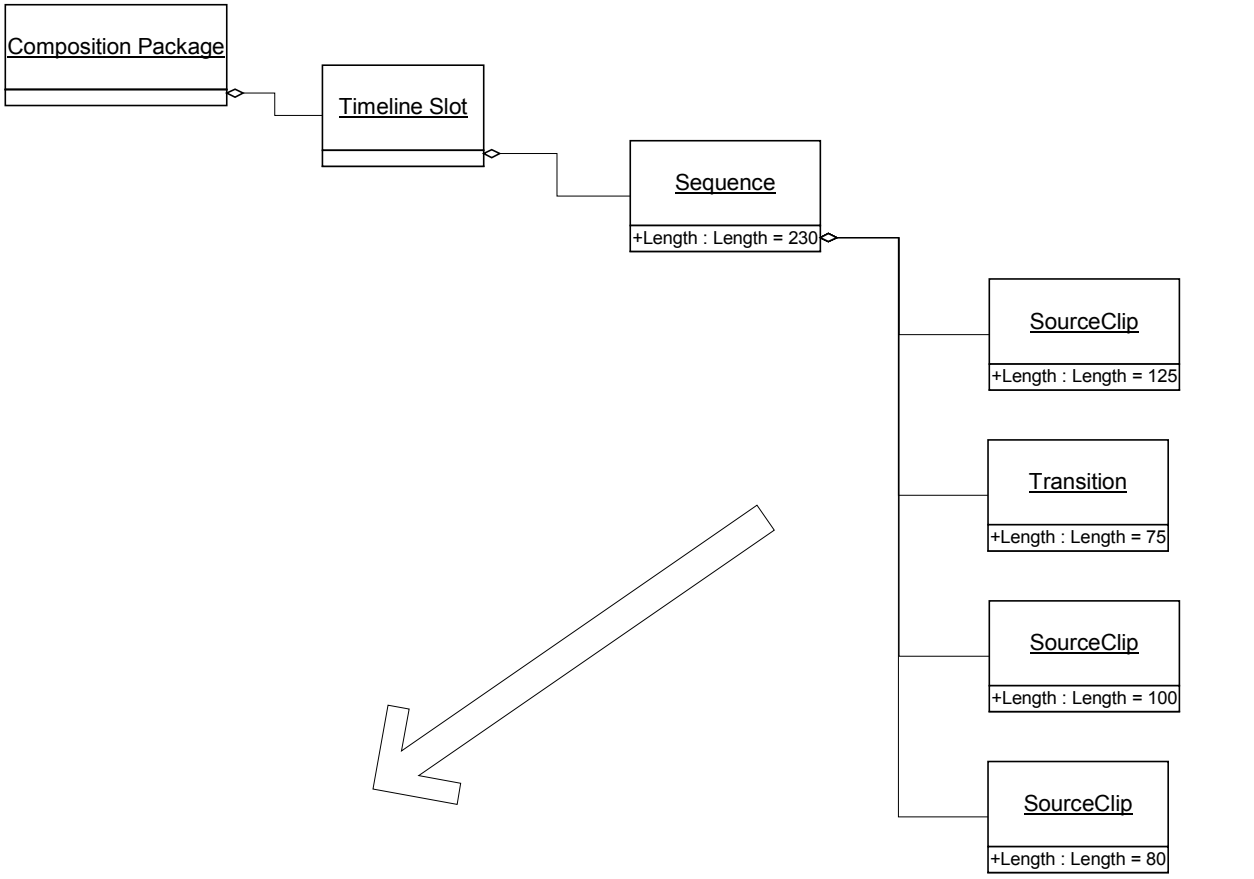


Figure 3-2: Containment Diagram of Sequence with Transition

Figure 3-3 shows an instance diagram of a Sequence containing Source Clips and a Transition. It shows the timeline view of the Sequence, in which the Transitions cause the two Source Clips to overlap


Composition Package
Timeline Slot
Sequence
+Length : Length = 230
SourceClip
+Length : Length = 125
Transition
+Length : Length = 75
SourceClip
+Length : Length = 100
SourceClip
+Length : Length = 80

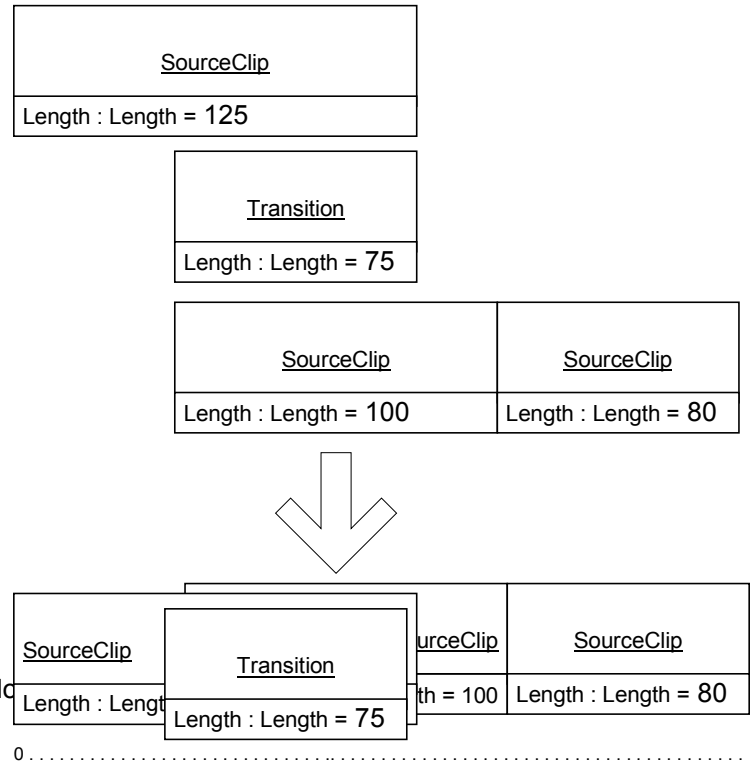

SourceClip
Length : Length = 125
Transition
Length : Length = 75
SourceClip
Length : Length = 100
SourceClip
Length : Length = 80
SourceClip
Length : Length = 75
SourceClip
Length : Length = 80
0
Timeline View

Figure 3-3: Transition Cause Segments to Overlap

To calculate the duration of a Sequence with Transitions, you add the durations of the Segments and then subtract the duration of the Transitions. In the example in Figure 3-3, the duration of the Sequence is 125 + 100 + 80 — 75, which equals 230.

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

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.

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.

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.

Static Slots

Static Slots describe Essence data that has no relationship to time. Consequently, Static Slots do not specify an edit rate and Segments in Static Slots do not have a duration. Figure 3-4 is a containment diagram for a Composition Package that has only Static Slots.

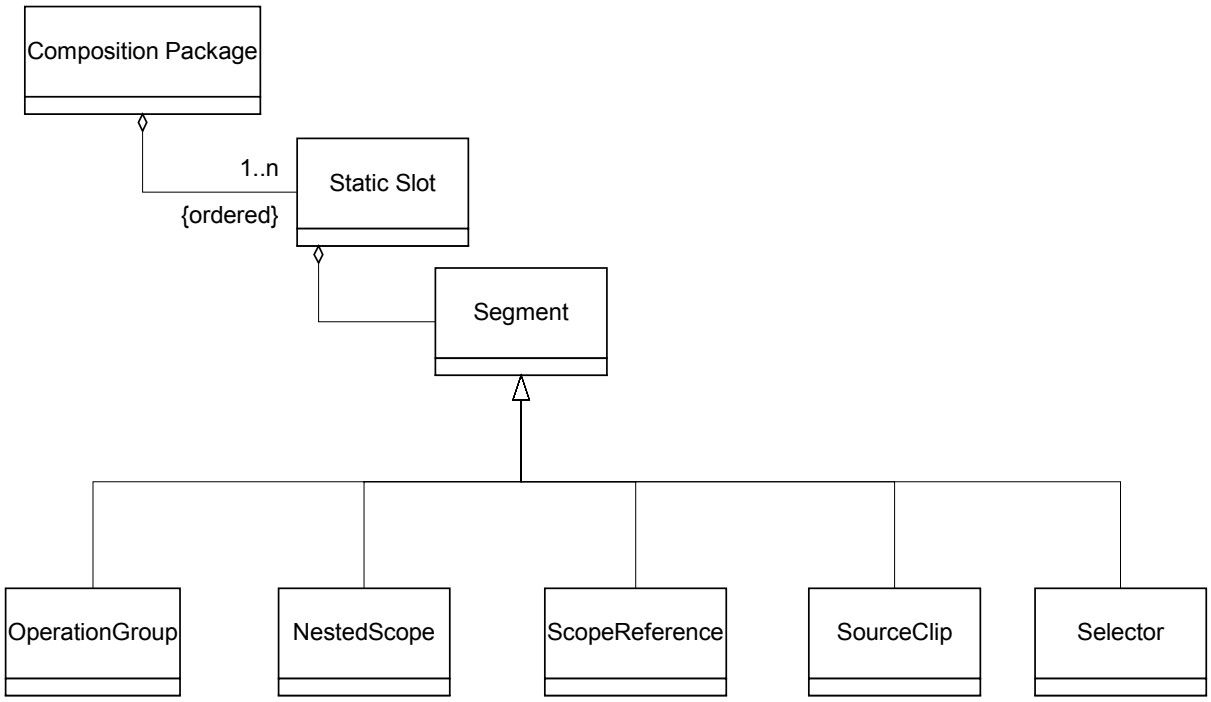


Figure 3-4: Containment Diagram for Composition Packages with Static Slots

Combining Different Types of Slots

A Composition Package can have Timeline Slots, Static Slots, and Event Slots. Although each kind of slot can only have Segments with the corresponding relationship to time, it is possible for a Slot to have a reference to another kind of Slot. For example, a video Timeline Slot can have a reference to an image in a Static Slot.

A Slot can reference a different kind of Slot by containing a Source Clip referencing the other Slot or by containing an Effect with a Source Clip referencing the other Slot. The Source Clip can reference Slots in other Packages or can reference other Slots in the same Package.

Conversion Operations

The Source Clip 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 3-1.

Table 3-1: Static, Timeline, and Event Conversions

Convert to:	Static	Event	Timeline
Convert from:			
Static		Source Clip plus Operation	Source Clip (Start Time ignored)
Event	Source Clip plus Operation		Source Clip plus Operation
Timeline	Source Clip (Length ignored)	Source Clip plus Operation	

Operations

This interchange standard includes a set of essence operation effects (such as transitions or chroma-key effects), which 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 an 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.

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.

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.

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.

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

For more information on defining effects, see the Defining Effect section.

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 an 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.

An 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.

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 Source Clip that identifies the Material Package and File Source Package that describe the rendered essence. If there is more than one implementation of a rendering, the Material Package could have a Essence Effect object.

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.

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 Package, 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 Source Clip than does not specify a PackageID parameter.

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 Source Clip for the island. shot
2. Effect for picture-in-picture effect.
3. Another Source Clip for the island shot.

Within each of the Effects, you would specify one of the input segments to have a Source Clip of the island shot. The problem with this way of implementing the Sequence is that there are three Source Clips 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 Source Clips 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.

How to Specify Scope References

The Package 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 Package 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 Package 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.

Other Composition Package Features

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

- Preserve editing choices
- Use audio fades

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 Package, 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 Package, an application treats the Selector object as if it were the selected Segment. However, when a user wants to edit the Composition Package, the application can present the alternative Segments as well as the selected one.

Using Audio Fade In and Fade Out

The Source Clip FadeInLength, FadeInType, FadeOutLength, and FadeOutType properties allow you to specify audio fades without an Effect object. Audio fades use these Source Clip 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 Source Clip properties rather than Effect properties simplifies the Composition Package 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 source clips to cause them to overlap. If the FadeOutLength of the preceding Source Clip is not equal to the FadeInLength of the following Source Clip, the crossfade will be asymmetric.

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

4. Describing and Storing Essence

This chapter shows how AAF files describe essence.

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, MPEG)
- Sampled audio essence in various formats (AIFC, Broadcast WAVE)
- Static image essence
- MIDI music essence
- Text essence in various formats
- Compound essence formats (DV, MPEG transport streams, ASF)

In addition to the essence formats described in this document, this interchange standard 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 standard defines the metadata in structures that are independent of the storage details of the essence format. This independence enables Composition Packages to reference essence data independently of its format. A Composition Package 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 standard 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 standard uses the following mechanisms to describe essence:

- Material Packages provide a level of indirection between Composition Packages and File Source Packages and can synchronize File Source Packages.
- Source Packages describe digital essence data stored in files or a physical media source such as videotape, audio tape, and film. The Source Package 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 Source Package 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 Source Packages 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.

This chapter contains the following sections:

- Describing Essence with Material Packages
- Describing Essence with Source Packages
- Describing Timecode
- Describing Essence with Pulldown

Describing Essence with Material Packages

A Material Package provides a level of indirection for accessing Source Packages from Composition Packages. The essence associated with a Source Package is immutable. Consequently, if you must make any changes to the essence data, you must create a new Source Package with a new unique PackageID. 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 Package may have many Source Clip objects that reference essence data updating every Source Clip in the Composition Package each time the essence is redigitized would be inefficient. By having the Composition Package access a Source Package only through a Material Package, this interchange standard ensures that you have to change only a single Material Package when you make changes to the essence data.

In addition, a Material Package can synchronize essence data in different Source Packages. For example, when an application digitizes a videotape, it creates separate Source Packages for the video and audio data. By having a single Material Package with one Slot for each Source Package, the Composition Package 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 Material Package can have a Essence Group object. The Essence Group object has a set of Source Clip objects, each of which identifies a Source Package 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 a essence data file is redigitized, it has to be broken into multiple files, this can be represented by a Sequence object in the Material Package that has a series of Source Clip objects, each identifying the Source Package associated with one of the files.

Typically, Material Packages 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 Source Clip object that identifies the Source Package. Material Packages cannot have Operation Groups, Nested Scopes, Selectors, Edit Rate Converters, or Transitions.

The following lists the reasons for having a Slot in a Material Package have an object other than a Source Clip:

- If there are multiple implementations of the same essence, the Slot can have a Essence Group instead of a Source Clip object.
- If the essence source has been broken into several Source Packages, the Slot can have a Sequence object. The Sequence object cannot have any component other than a Source Clip object or a Essence Group object.
- If one of a limited set of correction effects is applied to the essence data

Figure 4-1 illustrates the containment diagram for a Material Package describing timeline essence data, such as audio or video.

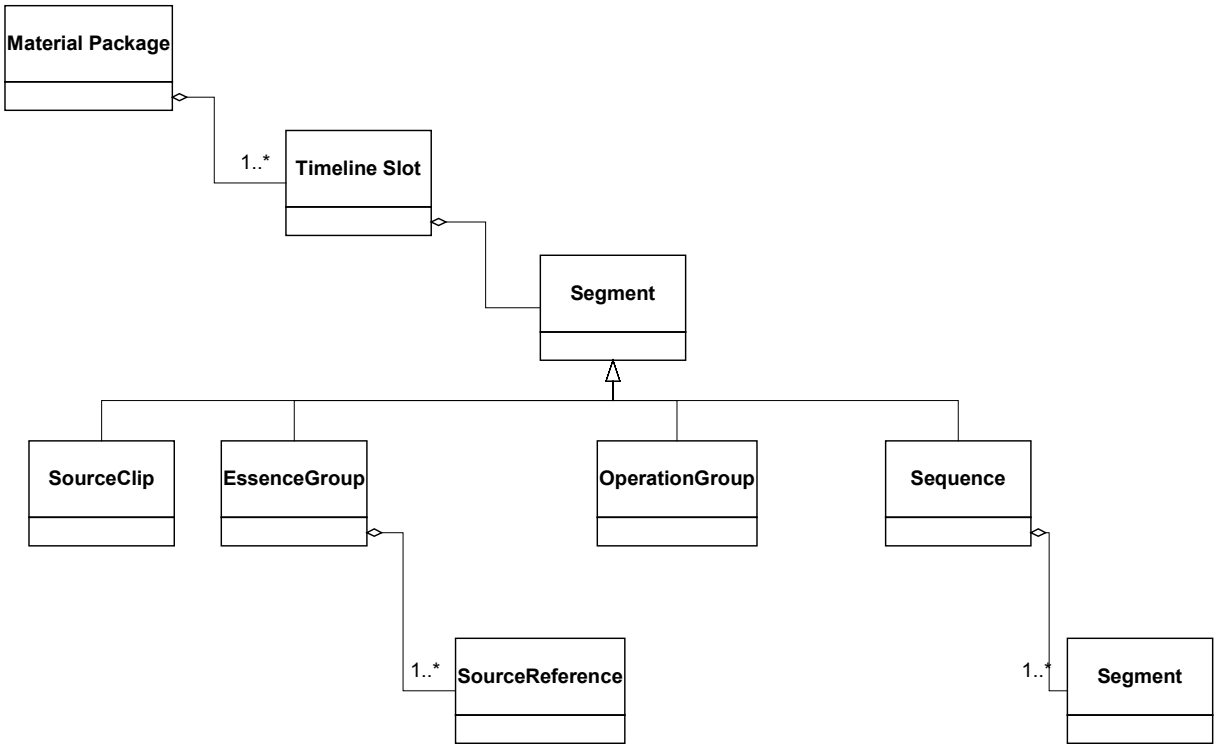


Figure 4-1: Material Package Containment Diagram

Describing Essence with Source Packages

A Source Package 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 Source Package has been derived from a previous generation of essence, the Slots should have Source Clips that identify the Package that describes the previous generation. If the Source Package describes essence that is not derived from a previous generation, the Slots should have Source Clips that specify the null Package.

Sample Rate and Edit Rate in Timeline Essence

In many cases the sample rate and edit rate in a file Source Package will be the same. However, it is possible to use different edit rates and sample rates in a Source Package. For example, you can create a Source Package 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.

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 Source Package indicates this position for each Timeline Slot 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 Source Clip 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 Source Clips in Composition Packages remain valid by changing the value of the Origin property in the Material Package. By setting the Origin to the current offset of the original starting point, the application ensures that existing Composition Packages remain valid.

Converting Edit Units to Sample Units

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

To play the digital essence data referenced by a Composition Package, the application uses the StartTime and Length values of the Composition Package's Source Clip, 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 EUs to sample durations, adds the file Slot's Origin to the Source Clip's StartTime, then converts the resulting sample time offset to a sample byte offset. Performing the final calculation for some 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 Source Package 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 Packages 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.
- Material Packages maintain a reference point in the digitized essence data called the source origin. Composition Packages reference positions in the essence data relative to the origin.

Describing Essence Format with Essence Descriptors

Source Packages describe the details of the essence format with a 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 a Essence Descriptor object belongs to a subclass of File Descriptor, it describes digital essence data. If a 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 standard to access it or to avoid duplicating already existing digital essence data. However, since there is no PackageID 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 Source Clip object that references it.

Figure 4-2 illustrates the containment diagram for File Source Packages and Figure 4-3 illustrates the containment diagram for Physical Source Packages.

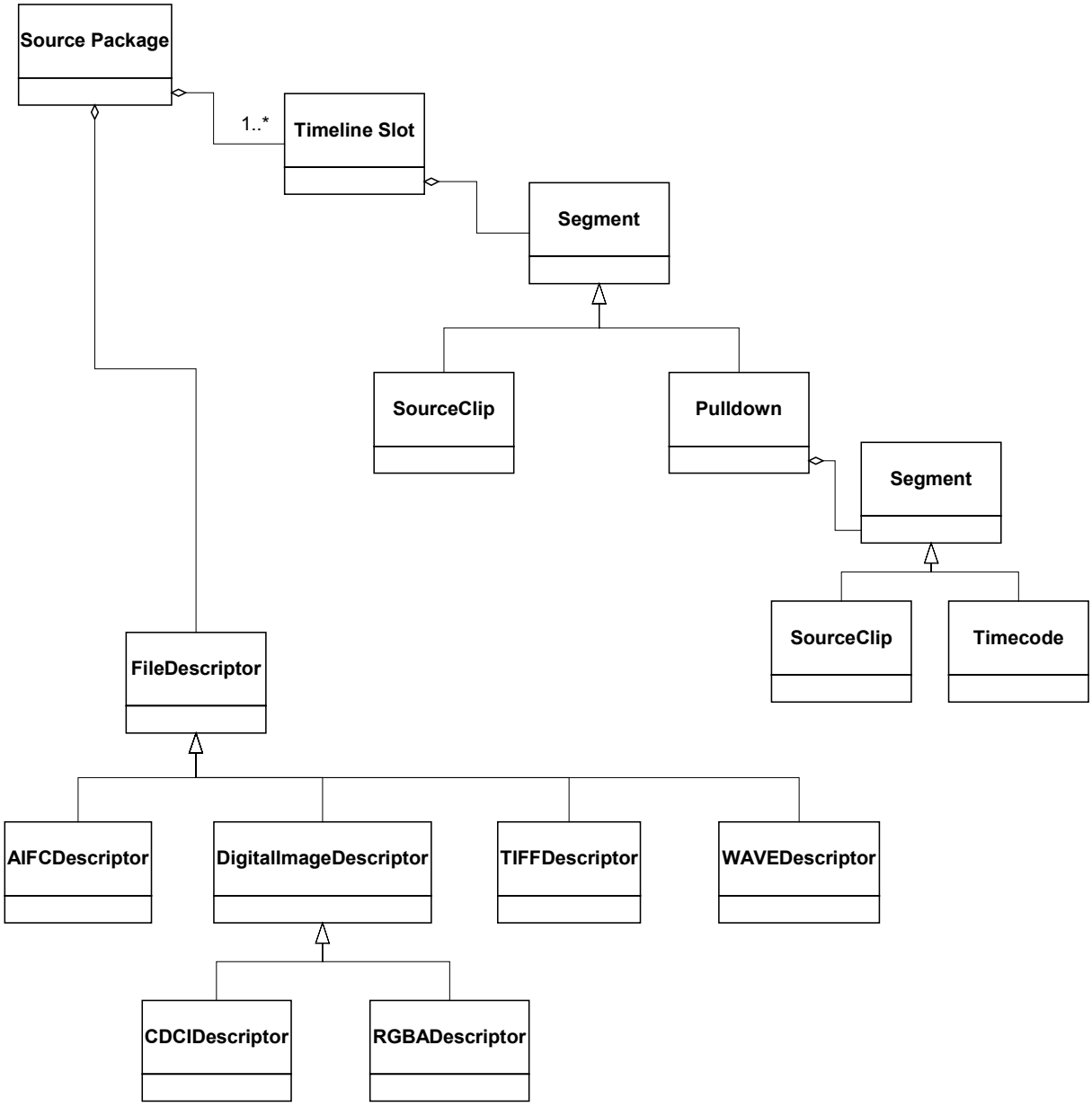


Figure 4-2: File Source Package Containment Diagram

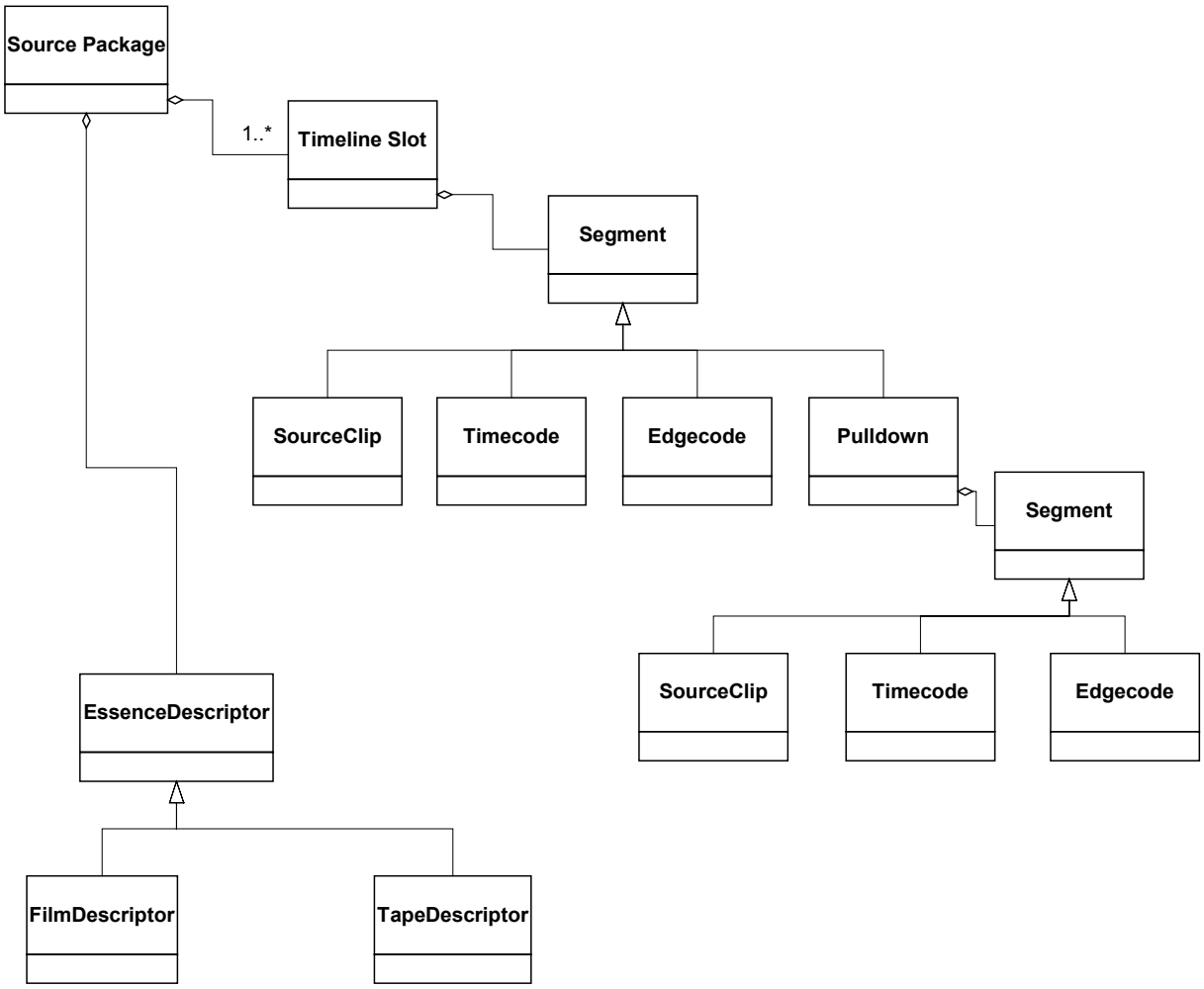


Figure 4-3: Physical Source Package Containment Diagram

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

Properties Describing Interleaving

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

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

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

Properties Describing Geometry

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

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

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

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

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

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

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

Properties Describing Sampling

The sampling properties describe the parameters used during the analog-to-digital digitization process. The properties detail the mapping between the signals as well as the format of the source analog signal. If the 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).

Properties Describing Alpha Transparency

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

Properties Describing Compression

The Compression property specifies that the image is compressed and the kind of compression used. Applications are required to support JPEG and no compression. 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.

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.

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.

Describing TIFF Image Essence

A TIFF Image Descriptor object describes the TIFF image data associated with the Source Package. 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.

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.

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.

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 standard 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 Material Package or Source Package 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.

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 Source Package, 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.

Describing Timecode

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

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

If a tape has a contiguous timecode, the Source Package can have:

- A Slot for each track of essence on the tape; the Slot should have a single Source Clip 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 Source Package in which the duration of the Source Clip does not necessarily match the duration of the videotape.

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

The starting timecode for digital essence data is specified by the Source Clip in the File Source Package and by the timecode track in the videotape Source Package. The Source Clip specifies the PackageID of the videotape Source Package, 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 Source Package.

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 Source Package has more than one timecode Slot, the PhysicalChannelNumber property indicates the purpose of each as described in Table 4-1.

Physical Channel	Usage
1	default TC
2	Sound TC
3	Aux. TC
4	AuxTC2
5	Aux TC3
6	Aux TC4
7	Aux TC5

Table 4-1: Physical Channel Number and Timecode Usage

Describing Edgecode

Film edgecode is described in Film Packages. Edgecode is specified with a Timeline Slot 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 PhysicalChanneNumber property as described in Table 4-2.

Physical Channel	Usage
1	Keycode #
2	Ink Number
3	Aux. Ink #

Table 4-2: Physical Channel Number and Timecode Usage

Describing Essence with Pulldown Objects

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

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.

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 4-4 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.

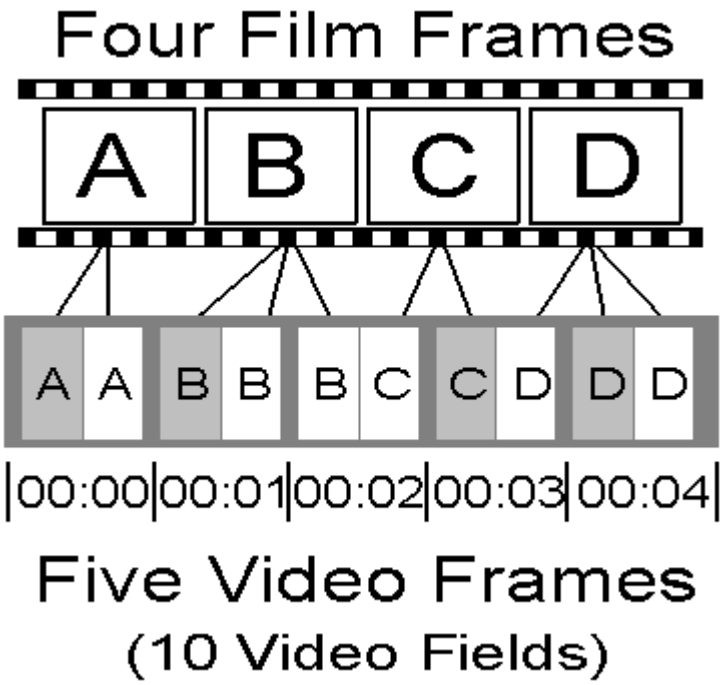


Figure 4-4: Telecine Three-Two Pulldown

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

If the final distribution format is being generating from film, there are advantages to digitizing the videotape to digital video 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.

Other Forms of Pulldown

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

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

Pulldown Objects in Source Packages

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

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

You need to use the phase information to convert an offset in the Package track containing the Pulldown object to an offset in the previous generation Package. To convert a film-rate offset, you multiply it by 5/4 to get a video rate offset, but if the result is not an integer, you use the phase information to determine whether you round up or down to get an integer value.

Typically a videotape is generated from more than one piece of film. In this case, the picture track in the videotape Source Package 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 Source Package 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.

5. Extending AAF

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 standard and may later progress to be included in the dynamic document that describes this standard.

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 standard. This standard defines the properties required to specify a locator to find a plug-in.

Extensions are specified in the Header `Dictionary` property.

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.

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

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 Header object. If the property has a property type defined in this document, you can omit the property type definition.

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 Packages

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 Packages 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 Source Packages that are not File Source Packages
- 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 Source Package, but you do have to define a new Essence Descriptor.

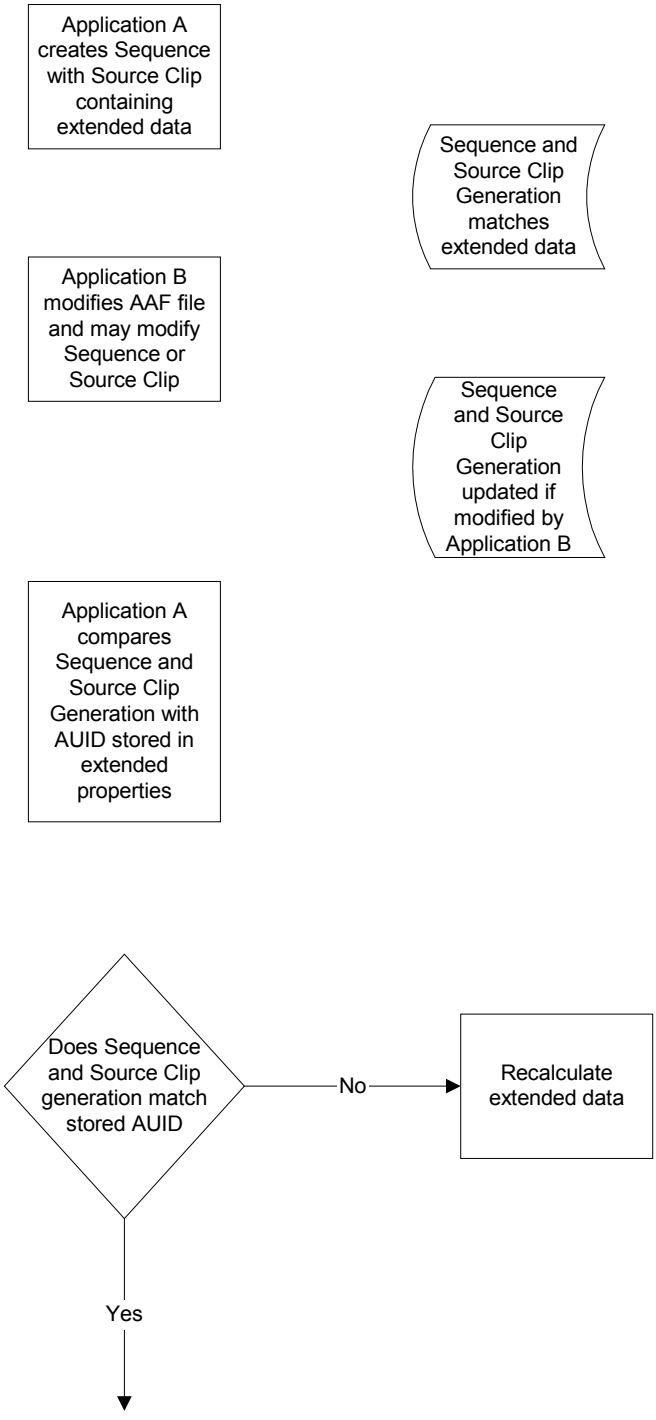
If the new essence type consists of a single kind of essence data, such as a single video stream or a static image, the Source Package should have a single Slot. If the essence type is a compound format that has multiple tracks of essence data, the File Source Package should have a separate Slot for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

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


Application A creates Sequence with Source Clip containing extended data
Sequence and Source Clip Generation matches extended data
Application B modifies AAF file and may modify Sequence or Source Clip
Sequence and Source Clip Generation updated if modified by Application B
Application A compares Sequence and Source Clip Generation with AUID stored in extended properties
Does Sequence and Source Clip generation match stored AUID
No
Recalculate extended data
Yes

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.

6. AAF Class Model and Class Hierarchy

This specification 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 and the MetaDefinition classes.

The InterchangeObject class is the root for most of the classes in AAF including those for Mobs and Essence Data. The AAFObject 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 AAFMetaDefinition 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.

If an AAF file contains extensions to the base AAF classes, these extensions will be defined in the file's AAFHeader object's `ClassDictionary` and `Definitions` properties. An AAF file will have one and only one Header object.

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 Appendix tbs.

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 to specifics such as extended property set management.

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.

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. Annex A lists the classes in the class hierarchy and specifies the properties that are required and the properties that are optional for each class. Annex A 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 ClassID property of the InterchangeObject class.

Class Hierarchy

Figures 6-1 through 6-4 illustrates the InterchangeObject class hierarchy. Figure 6-5 illustrates the MetaDefinition class hierarchy.

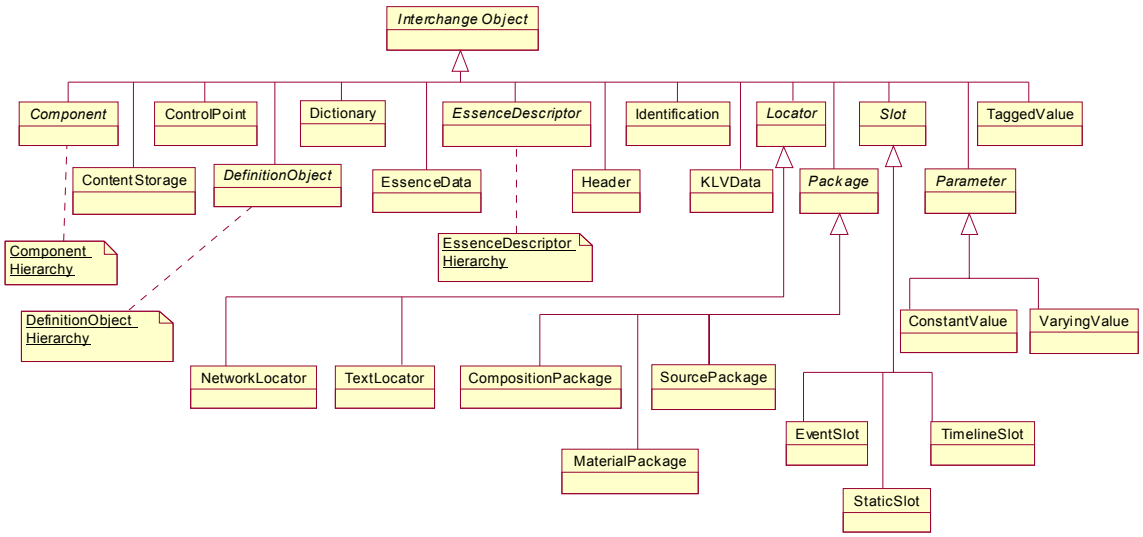


Figure 6-1 Class Hierarchy: InterchangeObject

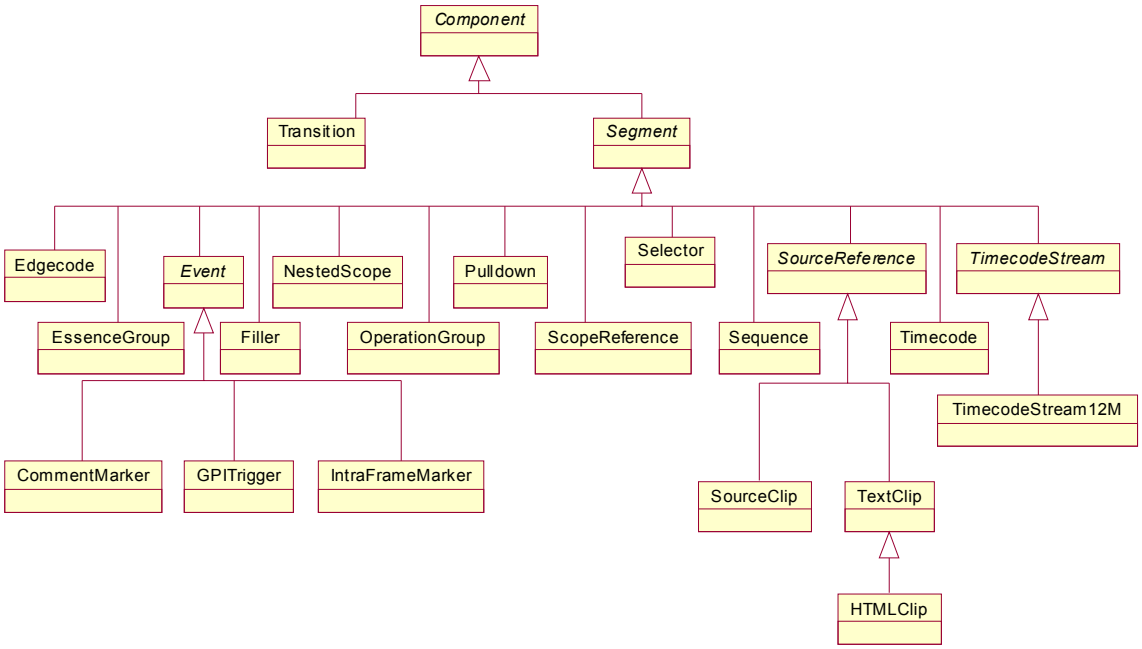


Figure 6-2 Class Hierarchy: Component

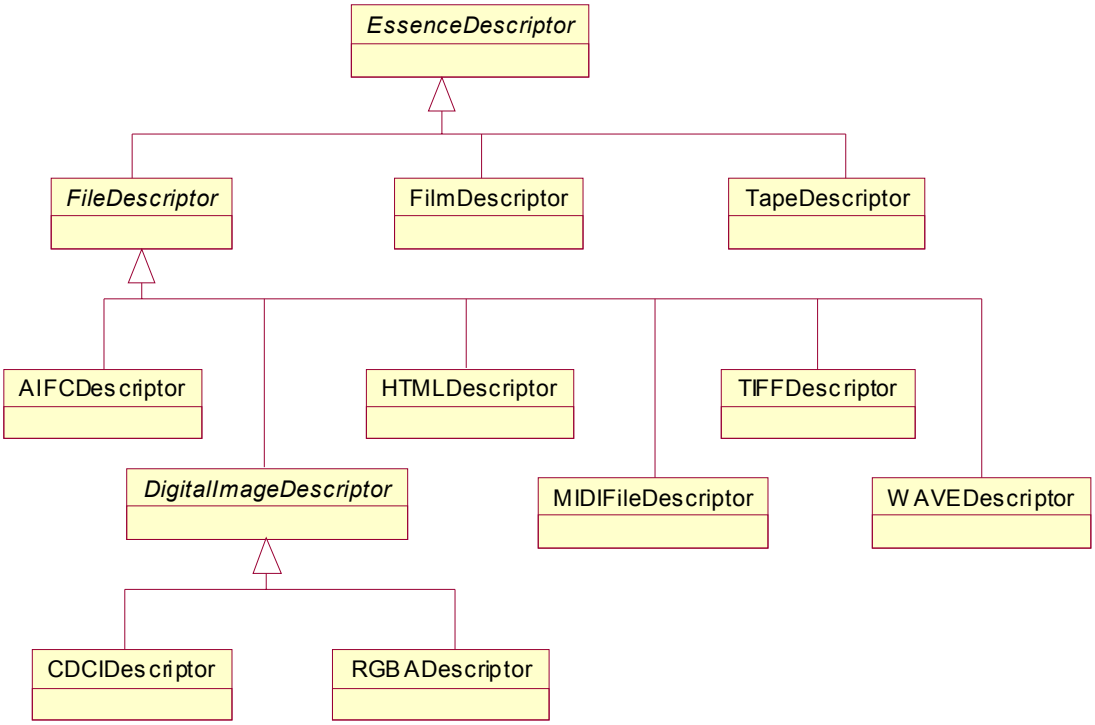


Figure 6-3 Class Hierarchy: EssenceDescriptor

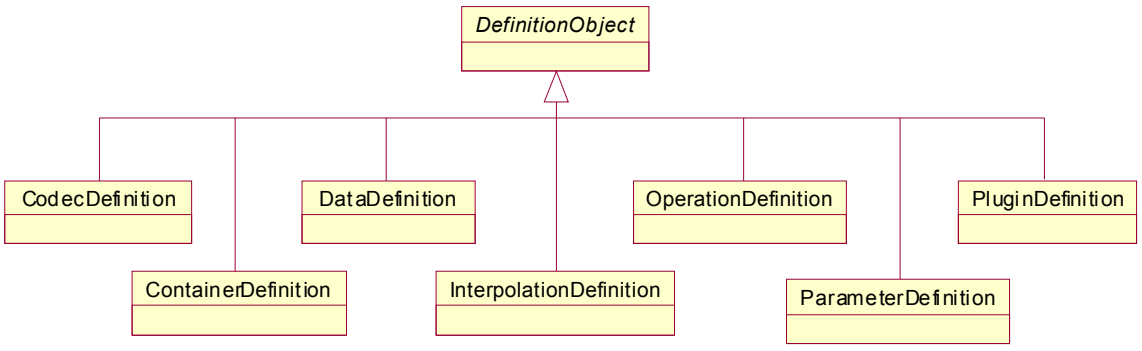


Figure 6-4 Class Hierarchy: DefinitionObject

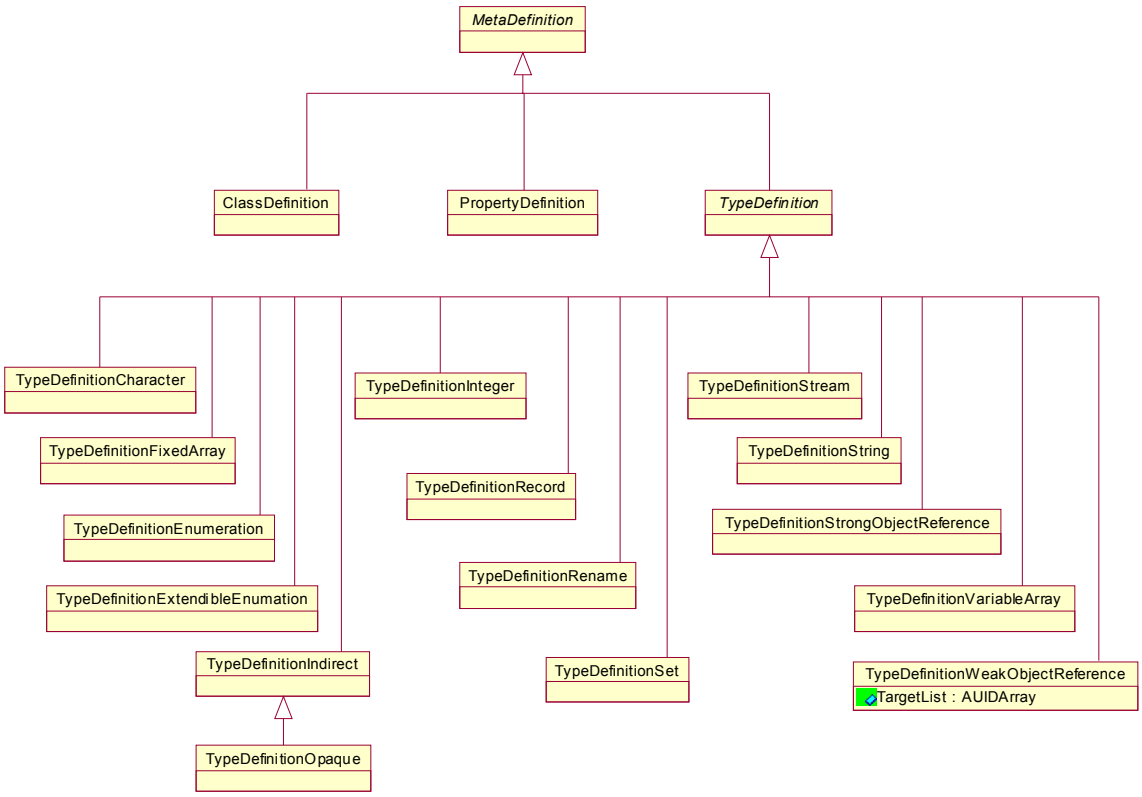


Figure 6-5 Class Hierarchy: MetaDefinition

Appendix A 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 of one of its superclasses is allowed subject to the restrictions listed in Appendix A.

Appendix A: AAF Object Classes for Essence and Metadata Interchange

This document contains the reference descriptions of the AAF classes. The reference pages are arranged alphabetically.

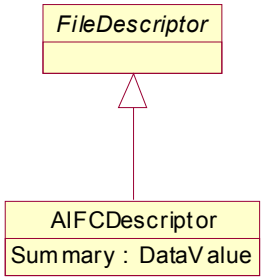
AIFCDescriptor Class

The AIFCDescriptor class specifies that a File Source Package is associated with audio content data formatted according to the Audio Interchange File Format with Compression (AIFC).

The AIFCDescriptor class is a subclass of the FileDescriptor class.

The AIFC audio format is a recommended audio format, but the AIFC format is not required for compliance with this document.

An AIFCDescriptor object shall be owned by a File Source Package.



An AIFCDescriptor object shall have the required properties and may have the optional properties listed in the following table.

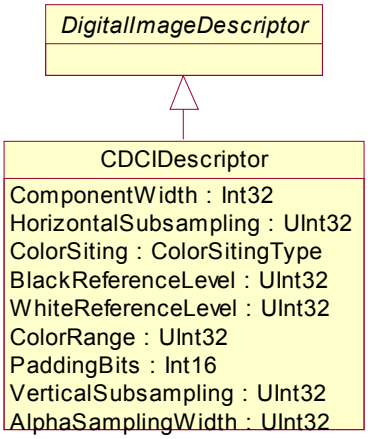
Property Name	Type	Explanation
`Pref:Summary`	`PrefT:DataValue`	A copy of the descriptive information in the associated AIFC Audio Data value. Required.

CDCIDescriptor Class

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

The CDCIDescriptor class is a subclass of the DigitalImageDescriptor class.

A CDCIDescriptor object shall be the EssenceDescription in a File Source Package.



A CDCIDescriptor object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation

Property Name	Type	Explanation
`Pref:ComponentWidth`	`PrefT:UInt32`	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. Required.
`Pref:HorizontalSubsampling`	`PrefT:UInt32`	Specifies the ratio of luminance sampling to chrominance sampling in the horizontal direction. For 4:2:2 video, the value is 2, which means that there are twice as many luminance values as there are color-difference values. The other legal value is 1. Required.
`Pref:ColorSiting`	`PrefT:ColorSitingType` Specifies how to compute subsampled chrominance component values. Values are: `0` coSiting — To calculate subsampled pixels, take the preceding pixel's color value, discard the other color values, and cosite the color with the first luminance value. 1 averaging — To calculate subsampled pixels, take the average of the two adjacent pixels' color values, and site the color in the center of the luminance pixels. 2 threeTap — To calculate subsampled pixels, take 25 percent of the previous pixel's color value, 50 percent of the first value, and 25 percent of the second value. For the first value in a row, use 75 percent of that value since there is no previous value. The threeTap value is only meaningful when the HorizontalSubsampling property has a value of 2. Optional; when omitted, treat as coSiting.	

Property Name	Type	Explanation
`Pref:BlackReferenceLevel`	`PrefT:UInt32`	Specifies the digital luminance component value associated with black. For CCIR-601/2, the value is 16; for YUV, the value is 0. The same value is used in CDCI and RGBA when the standard CCIR colorspace conversion is used. Optional; if omitted the default value is 0.
`Pref:WhiteReferenceLevel`	`PrefT:UInt32`	Specifies the digital luminance component value associated with white. For CCIR-601/2, 8-bit video, the value is 235; for YUV 8-bit video, the value is 255. Optional; if omitted, the default value is maximum unsigned integer value for component size.
`Pref:ColorRange`	`PrefT:UInt32`	Specifies the range of allowable digital chrominance component values. Chrominance values are signed and the range specified is centered on 0. For CCIR-601/2, the value is 225; for YUV the value is 255. This value is used for both chrominance components. Optional; the default value is the maximum unsigned integer value for the component size.
`Pref:PaddingBits`	`PrefT:Int16`	Specifies the number of bits padded to each pixel. Optional; default is 0.
`Pref:VerticalSubsampling`	`PrefT:UInt32`	Specifies the ratio of luminance sampling to chrominance sampling in the vertical direction. Optional; default value is 1.
`Pref:AlphaSamplingWidth`	`PrefT:UInt32`	Specifies the number of bits used to store the Alpha component. Optional; default value is 0.

Note 1 This format is commonly known as YCbCr.

Note 2 Chrominance subsampling reduces storage requirements by omitting the color difference information for some pixels. When reading the image, the color difference

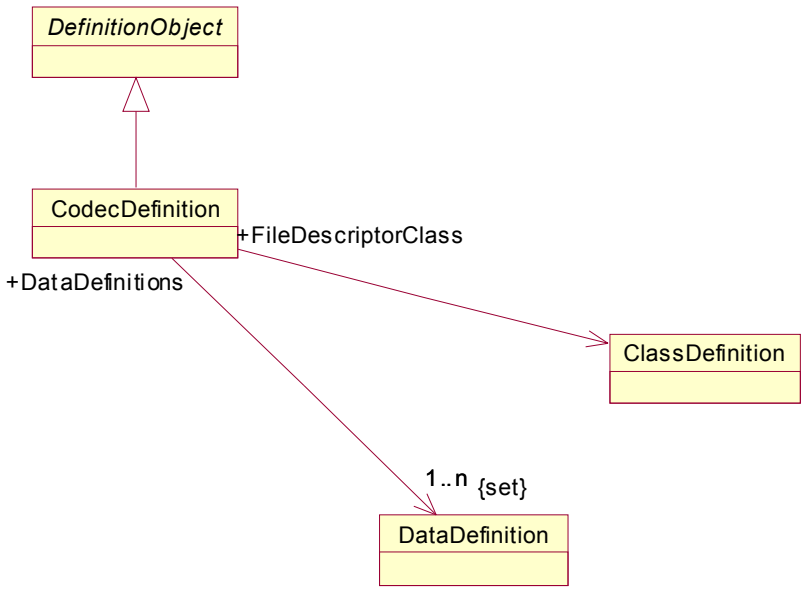
information for these pixels is calculated from the color difference information of the adjacent pixels. Color siting specifies how to calculate the color difference information when the two pixels have unequal color difference information.

CodecDefinition Class

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

The CodecDefinition class is a subclass of the DefinitionObject class.

All CodecDefinition objects are owned by a Dictionary object.



All CodecDefinition objects are owned by a Dictionary object. A CodecDefinition object shall have the required classes listed in the following table

Property Name	Type	Explanation
`Pref: FileDescriptorClass`	`PrefT: WeakReference` to ClassDefinition	Specifies the ClassDefinition of the subclass of FileDescriptor that identifies the essence format that this codec processes. Required.
`Pref:DataDefinition`	`PrefT: WeakReferenceSet` of DataDefinition	Specifies the DataDefinitions of the essence formats that this codec processes. Required.

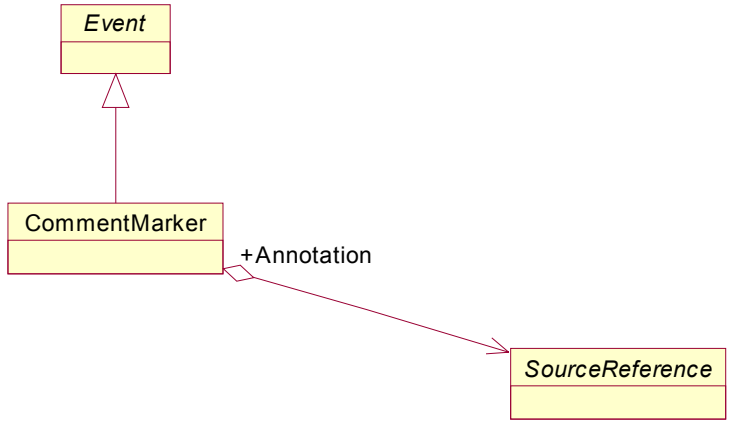
The FileDescriptorClass property identifies the essence format that the Codec can process. For example, a Codec that processes CDCI video data has a FileDescriptorClass that is a weak reference to the ClassDefinition object defining the CDCIDescriptor class. Note that a Codec may not be able to process 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.

CommentMarker Class

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

CommentMarker is a subclass of Event. A CommentMarker object may have a SourceReference that specifies a text or audio annotation.



A CommentMarker object shall have the required properties and may have the optional properties listed in the following table.

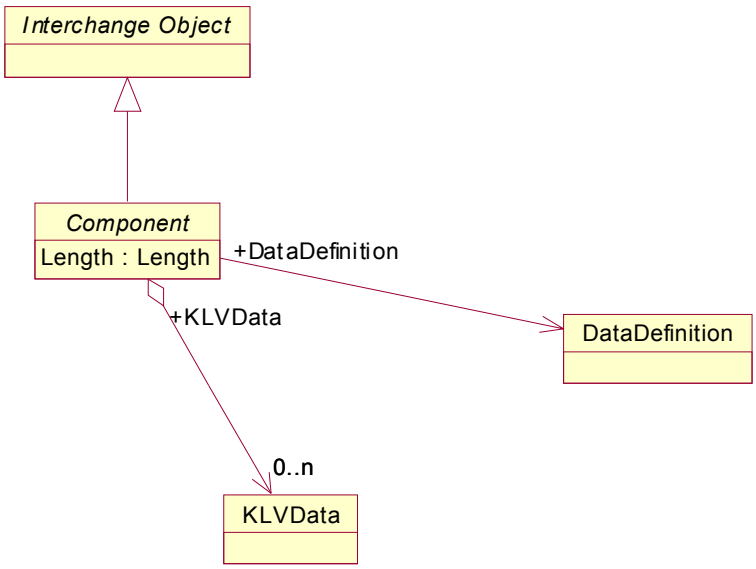
Property Name	Type	Explanation
`Pref:Annotation`	`PrefT:StrongReference` to SourceReference	Specifies text or audio annotation. Optional.

Component Class

The Component class represents a essence element.

The Component class is a subclass of InterchangeObject.

The Component class is an abstract class; consequently an object that belongs to the Component class shall also belong to a subclass of the Component class.



A Component object shall have the required properties and may have the optional properties listed in the following table. A Component object shall have or shall not have the Length property according to the rule in the list entry 1 following the table

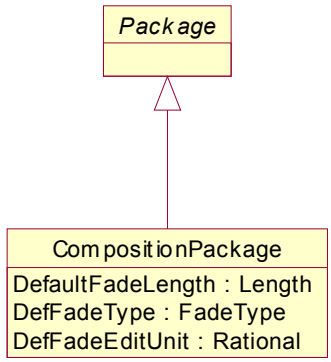
Property Name	Type	Explanation
`Pref:DataDefinition`	`PrefT:WeakReference` to DataDefinition	Specifies the DataDefinition object that specifies the kind of data described by the component. Required.
`Pref:Length`	`PrefT:Length`	Specifies the duration in edit units of the component. Optional; see rule 1.
`Pref:KLVData`	`PrefT: StrongReferenceVector of KLVData`	Contains a set of user KLV data consisting of a key (a SMPTE label), a length, and a value. Optional.

1. If a Component is in a TimelineSlot, then it shall have a Length property. If a Component is in a StaticSlot, then it shall not have a Length property. If a Component is in an EventSlot, then it may have a Length property. If a Component in an EventSlot does not have a Length property, then the Component describes an instantaneous event that does not have a duration.

CompositionPackage Class

The CompositionPackage class specifies how to combine content data elements into a sequence, how to modify content data elements, and how to synchronize content data elements.

The CompositionPackage class is a subclass of the Package class.



A CompositionPackage object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:DefaultFadeLength`	`PrefT:Length`	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. Optional; if specified, then the default fade type and the default fade edit units must also be specified.
`Pref:DefaultFadeType`	`PrefT:FadeType`	Specifies the default type of audio fade. Optional; if specified, then the default length and default edit units must also be specified. Specifies the type of the audio fade in; may have one of the following values: 0 No fade 1 Linear amplitude fade 2 Linear power fade 3 Linear dB fade Additional registered and private fade in types may be defined. Optional.

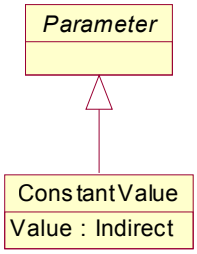
`Pref:DefaultFadeEditUnit`	`PrefT:Rational`	Specifies the edit units in which the default fade length is specified. Optional; if specified, then the default fade length and default fade type must also be specified.

1. A CompositionPackage object shall have one or more Slots
2. A ContentStorage may have any number of composition Packages.

ConstantValue Class

Specifies a constant data value and a duration and is used to specify an effect control value.

The ConstantValue class is a subclass of the Parameter class.



A ConstantValue object shall have the required properties and may have the optional properties listed in the following table.

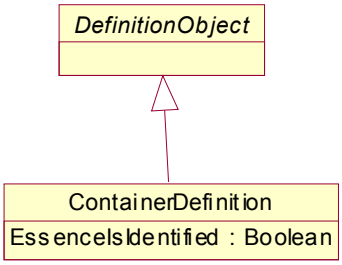
Property Name	Type	Explanation
`Pref:Value`	`PrefT:Indirect`	Specifies the value. Required.

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 subclass of the DefinitionObject class.

All ContainerDefinition objects shall be owned by the Dictionary object.



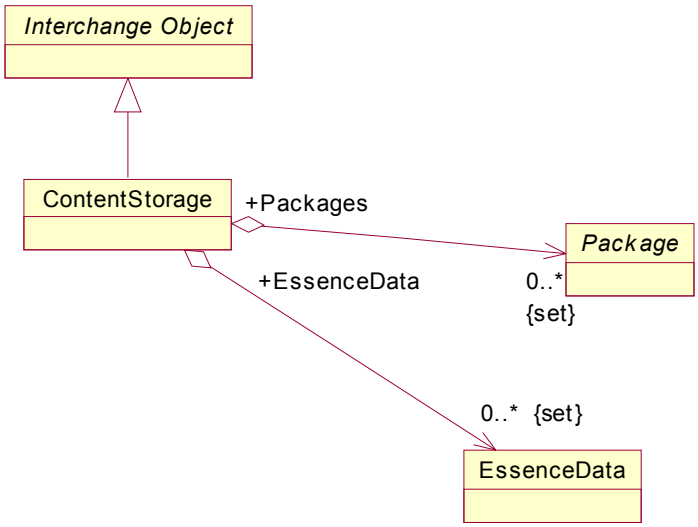
All ContainerDefinition objects shall be owned by the Dictionary object. A ContainerDefinition object may have the optional classes listed in the following table

Property Name	Type	Explanation
`Pref: EssenceIsIdentified`	`PrefT:Boolean`	Specifies that the container uses the PackageID to identify the essence data and that the container may contain multiple essence data objects, each identified by a PackageID. Optional.

ContentStorage Class

The ContentStorage class has the Packages and EssenceData objects in a file. A AAF file shall have one and only one ContentStorage object.

The ContentStorage class is a subclass of the InterchangeObject class.



A ContentStorage object shall have the required properties and may have the optional properties listed in the following table.

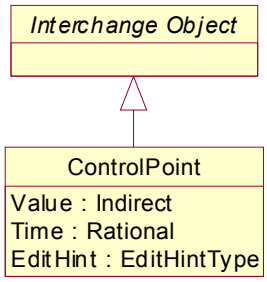
Property Name	Type	Explanation
`Pref:Packages`	`PrefT: StrongReferenceSet` of Package	Has a set of all Packages in the file. Required.
`Pref:EssenceData`	`PrefT: StrongReferenceSet` of EssenceData	Has a set of all EssenceData objects in the file. Optional.

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 subclass of InterchangeObject.

A ControlPoint shall be one of the set of ControlPoint objects in the VaryingValue PointList property.



A ControlPoint object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:Value`	`PrefT:Indirect`	Specifies the value. Required.
`Pref:Time`	`PrefT:Rational`	Specifies the time within the Varying Value segment for which the value is defined. Required.

`Pref:EditHint`	`PrefT:EditHintType`	Specifies a hint to be used if the Effect starting time or length is changed during editing. Can be EH_Proportional, EH_RelativeLeft, or EH_RelativeRight. Optional.

1. A Control Point object specifies the value at a specific time in a Varying Value object. The Control Point object must have the same type as the Varying Value object owning it.

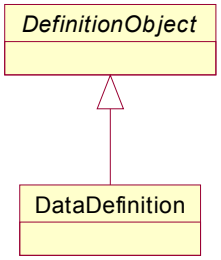
2. A Time equal to 0.0 represents the time at the beginning the Varying Value Object; a Time equal to 1.0 represents the time at the end of the Varying Value object

DataDefinition Class

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

The DataDefinition class is a subclass of the DefinitionObject class.

All DataDefinition objects shall be owned by a Dictionary object.



The DataDefinition class does not define any additional properties.

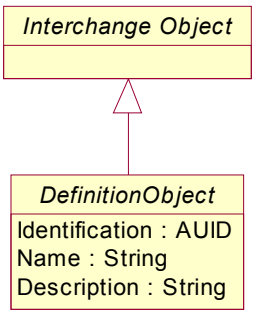
Note 1 A Data Definition object identifies the kind of the data produced by a Component object.

DefinitionObject Class

The DefinitionObject is an abstract class that defines an item to be referenced.

The DefinitionObject class is a subclass of the InterchangeObject class.

The DefinitionObject class is an abstract class; consequently an object that belongs to the DefinitionObject class shall also belong to a subclass of the DefinitionObject class.



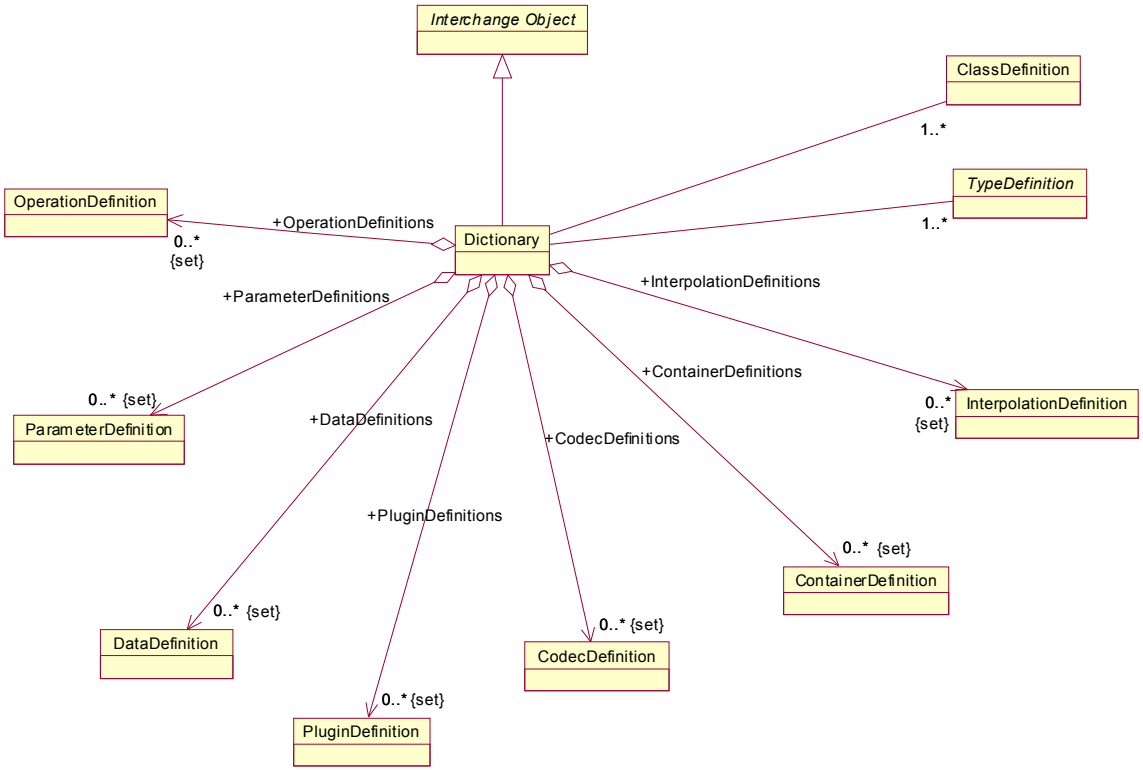
A DefinitionObject object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:Identification`	`PrefT:AUID`	Specifies the unique identifier for the item being defined. Required.
`Pref:Name`	`PrefT:String`	Specifies the display name of the item being defined. Required.
`Pref:Description`	`PrefT:String`	Provides an explanation of the use of the item being defined. Optional.

Dictionary Class

The Dictionary class has Definition objects.

The Dictionary class is a subclass of the InterchangeObject class.



A Dictionary object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:ClassDefinitions`	`PrefT: StrongReferenceSet` of ClassDefinition	Specifies the ClassDefinitions that are used in the file. Optional.
`Pref:TypeDefinitions`	`PrefT: StrongReferenceSet` of TypeDefinition	Specifies the Types that are used in the file. Optional.
`Pref:OperationDefinitions`	`PrefT: StrongReferenceSet` of OperationDefinition	Specifies the OperationDefinitions that are used in the file. Optional.
`Pref:ParameterDefinitions`	`PrefT: StrongReferenceSet` of ParameterDefinition	Specifies the ParameterDefinitions that are used in the file. Optional.
`Pref:DataDefinitions`	`PrefT: StrongReferenceSet` of DataDefintion	Specifies the DataDefinitions that are used in the file. Optional.

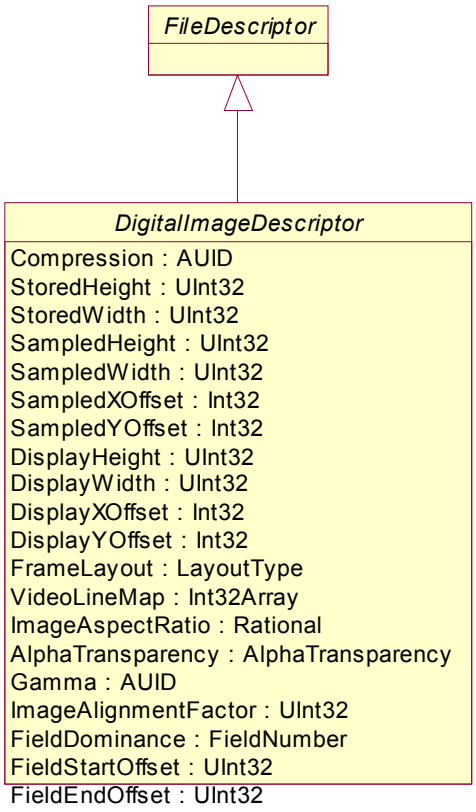
Property Name	Type	Explanation
`Pref:PluginDefinitions`	`PrefT: StrongReferenceSet` of PluginDefinition	Identifies code objects that provide an implementation for a DefinitionObject, such as a CodecDefinition or an InterpolationDefinition. Optional.
`Pref:CodecDefinitions`	`PrefT: StrongReferenceSet` of CodecDefinition	Specifies CodecDefinitions that describe code that can compress or uncompress samples of EssenceData or that can convert samples to another format.
`Pref: ContainerDefinitions`	`PrefT: StrongReferenceSet` of ContainerDefinition	Specifies ContainerDefinitions that describe container mechanisms used to store EssenceData. Optional.
`Pref: InterpolationDefinitions`	`PrefT: StrongReferenceSet` of InterpolationDefinition	Specifies InterpolationDefinitions that can calculate values in a VaryingValue based on the values specified by the ControlPoints. Optional.

DigitalImageDescriptor Class

The DigitalImageDescriptor class specifies that a File Source Package is associated with video content data that is formatted either using RGBA or luminance/chrominance formatting.

The DigitalImageDescriptor class is a subclass of the FileDescriptor class.

The DigitalImageDescriptor class is an abstract class; consequently an object that belongs to the DigitalImageDescriptor class shall also belong to a subclass of DigitalImageDescriptor.



A DigitalImageDescriptor object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:Compression`	`PrefT:AUID`	Kind of compression and format of compression information. Optional; if there is no compression, the property is omitted.
`Pref:StoredHeight`	`PrefT:UInt32`	Number of pixels in vertical dimension of stored view. Required.
`Pref:StoredWidth`	`PrefT:UInt32`	Number of pixels in horizontal dimension of stored view. Required.
`Pref:SampledHeight`	`PrefT:UInt32`	Number of pixels in vertical dimension of sampled view. Optional; the default value is StoredHeight.

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

Property Name	Type	Explanation
Pref: AlphaTransparency	PrefT: AlphaTransparency	Specifies whether the minimum Alpha value or the maximum Alpha value indicates transparency. Values are 0 kAAFMinValueTransparent: a 0 value indicates transparency 1 kAAFMaxValueTransparent: the maximum Alpha value indicates transparency Optional.
Pref:Gamma	PrefT:AUID	Specifies the expected output gamma setting on the video display device. Optional.
Pref: ImageAlignmentFactor	PrefT:UInt32	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. Optional; default value is 0.
Pref:FieldDominance	PrefT:FieldNumber	Specifies whether field 1 or field 2 is dominant in images composed of two interlaced fields. Optional.
Pref: FieldStartOffset	PrefT:UInt32	Specifies unused bytes at the start of each video field. Optional; default value is 0.
Pref:FieldEndOffset	PrefT:UInt32	Specifies unused bytes at the end of each video field. Optional; default value is 0.

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

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

3. The Compression property specifies that the image is compressed and the kind of compression used.

4. 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 geometry's—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 in Figure A-1.

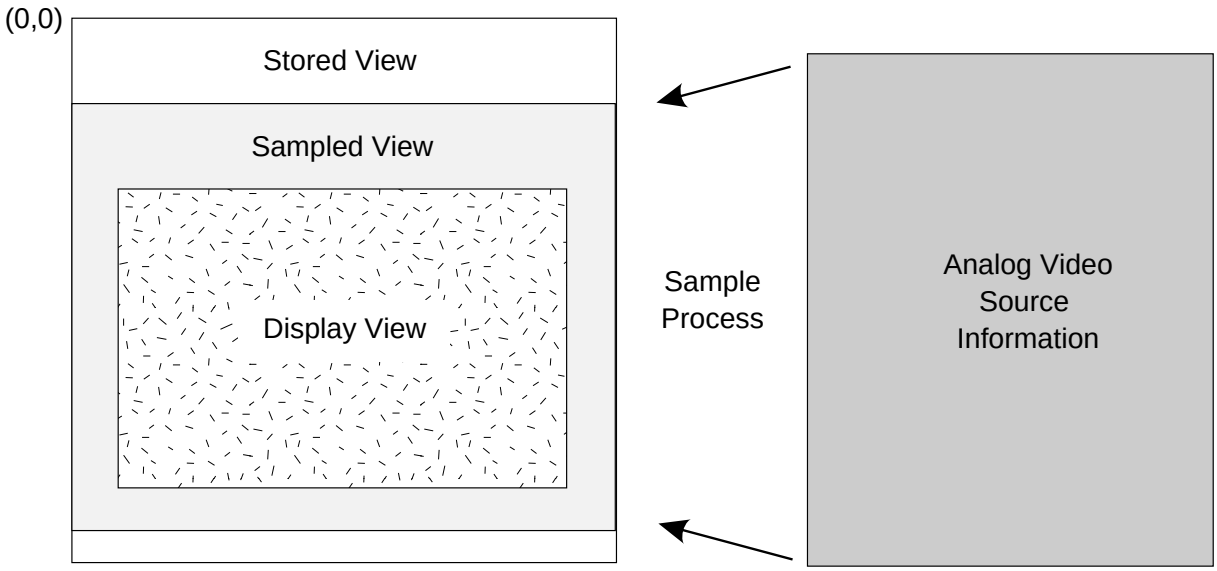


Figure A-1 – Stored, Sampled, and Displayed View

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

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

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

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

Each of these data views 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.

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

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

7. 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}

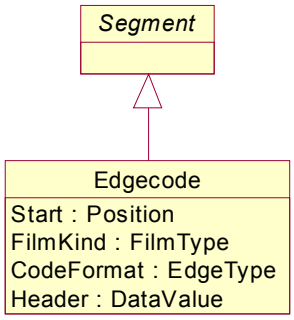
   A value of 0 is allowed only when computer-generated essence has to be treated differently. If the digital essence was computer generated (RGB), the values may be either {0,1} (even field first) or {1,0} (odd field first).

8. 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.

Edgecode Class

The Edgecode class stores film edge code information.

The Edgecode class is a subclass of the Segment class.



An Edgecode object shall have the required properties listed in the following table.

Property Name	Type	Explanation
`Pref:Start`	`PrefT:Position`	Specifies the edge code at the beginning of the segment. Required.

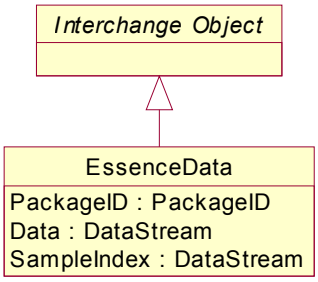
Pref:FilmKind	PrefT:FilmType	Specifies the type of film; one of these: 0 FT_NULL 1 FT_35MM 2 FT_16MM 3 FT_8MM 4 FT_65MM Required.
Pref:CodeFormat	PrefT:EdgeType	Specifies the edge code format; one of these: 0 ET_NULL 1 ET_KEYCODE 2 ET_EDGENUM4 3 ET_EDGENUM5 Required.
Pref:Header	PrefT:DataValue	Specifies the text prefix that identifies the film. Typically, this is a text string of no more than 8 7-bit ISO characters. Optional.

EssenceData Class

The EssenceData class contains essence.

The EssenceData class is a subclass of the InterchangeObject class.

The EssenceData class is an abstract class; consequently an object that belongs to the EssenceData class shall also belong to a subclass of EssenceData.



A EssenceData object shall have the required properties listed in the following table.

Property Name	Type	Explanation
Pref:PackageID	PrefT:PackageID	Identifies the source Package that describes the essence. Required.

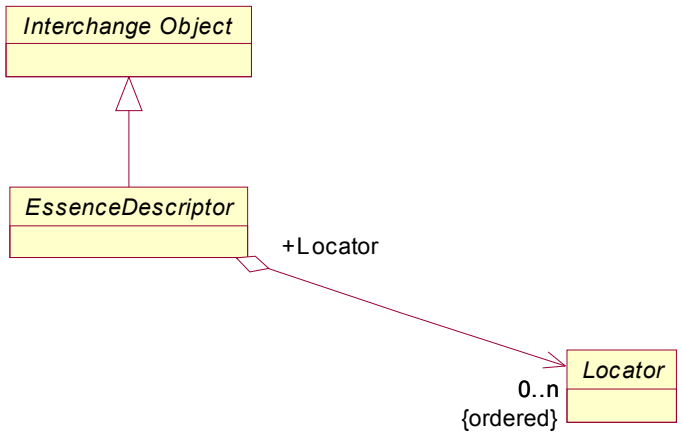
`Pref:Data`	`Pref:DataStream`	Contains the essence data. Required.
`Pref:SampleIndex`	`Pref:DataStream`	Contains an index to the samples or frames. The format of the index is determined by the Codec. Optional.

EssenceDescriptor Class

The EssenceDescriptor class describes the format of the content data associated with a File Source Package or of the media associated with a Physical Source Package.

The EssenceDescriptor class is a subclass of the InterchangeObject class.

The EssenceDescriptor class is an abstract class; an object that belongs to the EssenceDescriptor class shall belong to a subclass of EssenceDescriptor.



A EssenceDescriptor object may have the optional properties described in the following table.

Property Name	Type	Explanation
`Pref:Locator`	`PrefT: StrongReferenceVector` of Locator	Has an array of Locator objects that provide operating-system-dependent data or text information that provide hints for finding files or physical media. Optional.

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

1) 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 source Package and provide information for finding files. A file source Package can have any number of locators and the locators may belong to any subclass of Locator.

2) 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 source Package and provide information for finding physical media. A physical source Package can have any number of locators; the locators shall belong to the TextLocator subclass of Locator.

Note 1 Locators in a file source Packages provide hints to help find files associated with the file source Package, 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 your application cannot find a file by using the hint, it can search through all accessible AAF files to find the EssenceData object with the PackageID value.

Note 2 A essence descriptor may have more than one locator objects and a essence descriptor may have more than one locator object of the same subclass of Locator. For example, a file source Package may have more than one locator for any of the following reasons:

– locators that provide hints to find the file on more than one operating system

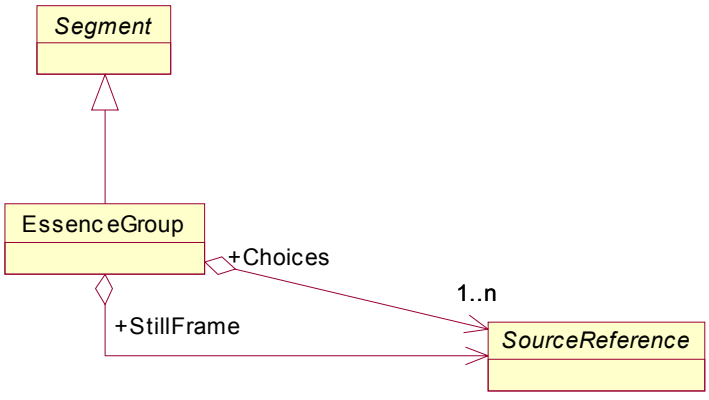
– locators that provide more than one hint on the same operating system

EssenceGroup Class

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

The EssenceGroup class is a subclass of the Segment class.

An EssenceGroup object shall be a Segment in a Material Package Slot.



An EssenceGroup object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:Choices`	`PrefT: StrongReferenceVector` of Segment	Has a collection of Segment that identify the alternate representations that may be chosen. The order of the items in the collection is not meaningful. Required.
`Pref:StillFrame`	`PrefT: StrongReference` to SourceReference	Has a Source Reference that identifies the essence for a single-frame image representation of the essence. Optional.

1. The Segment shall either be a Source Clip or a Sequence. If the Segment is a Sequence, it shall contain only Source Clip and Fill objects.
2. The length of each Segment in the Choices set must be the same as the length of the Essence Group object.
3. The length of the StillFrame Source Clip must be 1.

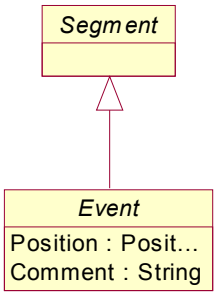
Note Typically the different representations vary in essence format, compression, or frame size. The application is responsible for choosing the appropriate implementation of the essence.

Event Class

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

Event is a subclass of `Segment`. Typically an Event is owned by a Sequence in an EventSlot.

The Event class is an abstract class; consequently an object that belongs to the Event class shall also belong to a subclass of the Event class.



An Event object shall have the required properties and may have the optional properties listed in the following table.

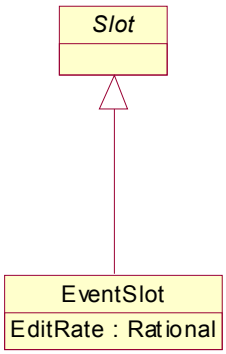
Property Name	Type	Explanation
`Pref:Position`	`PrefT:Position`	Specifies the starting time of the event in the EventSlot. Required.
`Pref:Comment`	`PrefT:String`	Specifies the purpose of the event. Optional.

An Event specifies its position as an absolute time expressed in the edit rates of the EventSlot that has it.

EventSlot Class

EventSlot has a Sequence of Events.

`EventSlot` is a subclass of `Slot`. An EventSlot object, as all Slots, has a concrete Segment, which is typically a Sequence.



An EventSlot object shall have the required properties listed in the following table.

Property Name	Type	Explanation
`Pref:EditRate`	`PrefT:Rational`	Specifies the units in which the events specify their starting time and duration. Required.

1. An EventSlot shall have a concrete Segment that is either an Event or a Sequence.
2. If an EventSlot has a Sequence, then the Sequence shall conform to the following rules:
    A. All Segments in the Sequence shall be Events.
    B. All Events in the Sequence shall belong to the same concrete subclass of Event.

C. All Events in the Sequence shall have the same DataDefinition as does the Sequence.

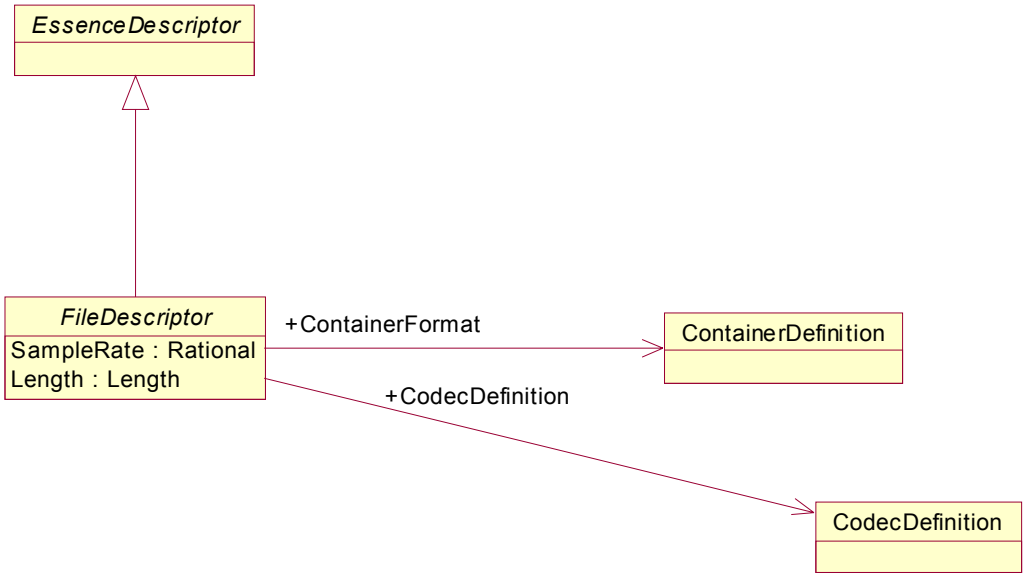
D. In a Sequence, the Position of each Event shall be greater than or equal to the Position of the Event preceding it in the Sequence.

FileDescriptor Class

The FileDescriptor class describes essence associated with a File Source Package.

The FileDescriptor class is a subclass of the EssenceDescriptor class.

The FileDescriptor class is an abstract class; consequently an object that belongs to the FileDescriptor class shall also belong to a subclass of the FileDescriptor class.



A FileDescriptor object shall have the required properties listed in the following table.

Property Name	Type	Explanation
`Pref:SampleRate`	`PrefT:Rational`	The sample rate of the essence. Optional.
`Pref:Length`	`PrefT:Length`	Duration of the essence in sample units. Optional.
`Pref:ContainerFormat`	`PrefT:WeakReference` to ContainerDefinition	Identifies the container mechanism used to store the EssenceData. Required.

`Pref:CodecDefinition`	`PrefT:WeakReference` to CodecDefinition	Identifies the mechanism used to compress and uncompress samples of EssenceData or used to convert samples of EssenceData from one format to another. Required.

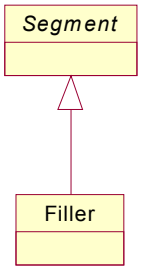
1. FileDescriptors describing static essence shall omit the SampleRate and Length properties. FileDescriptors describing time-varying essence shall specify the SampleRate and Length properties.

2. The Essence 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 Source Clip in the File Source Package.

Filler Class

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

The Filler class is a subclass of the Segment class.



The Filler class does not define any additional properties.

Note 1 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 Package Slots and Nested scope Slots that are not referenced or played.

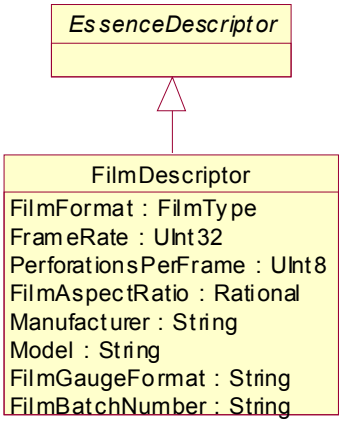
Note 2 If a Filler object is played, applications can choose any appropriate blank essence to play. Typically, a video Filler object would be played as a black section, and an audio Filler object would be played as a silent section.

FilmDescriptor Class

The FilmDescriptor class describes film media.

The FilmDescriptor class is a subclass of the EssenceDescriptor class.

An FilmDescriptor object shall be the EssenceDescription of a Physical Source Package.



An FilmDescriptor object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation

`Pref:FilmFormat`	`PrefT:FilmType`	Identifies the format of the film; one of the following:
		`0 FT_NULL`
		`1 FT_35MM`
		`2 FT_16MM`
		`3 FT_8MM`
		`4 FT_65MM`
		Optional.

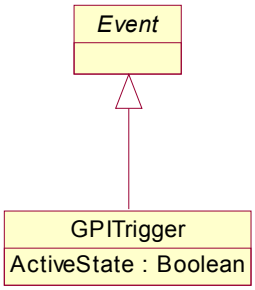
`Pref:FrameRate`	`PrefT:UInt32`	Specifies the frame rate in frames per second. Optional.
`Pref:PerforationsPerFrame`	`PrefT:UInt8`	Specifies the number of perforations per frame on the film stock. Optional.
`Pref:FilmAspectRatio`	`PrefT:Rational`	Ratio between the horizontal size of the frame image and the vertical size of the frame image. Optional.
`Pref:Manufacturer`	`PrefT:String`	A string to display to end users, indicating the manufacturer of the film. Optional.
`Pref:Model`	`PrefT:String`	A string to display to end users, indicating the manufacturer's brand designation for the film, such as "5247". Optional.

Property Name	Type	Explanation
Pref:FilmGaugeFormat	PrefT:String	Specifies the gauge and film format, such as “Blair Viventoscope”. Optional
Pref:FilmBatchNumber	PrefT:String	Specifies the batch number identifying the film. Optional.

GPITrigger Class

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

GPITrigger is a subclass of Event. GPITrigger objects are owned by a Sequence in an EventSlot object.



A GPITrigger object shall have the required properties listed in the following table.

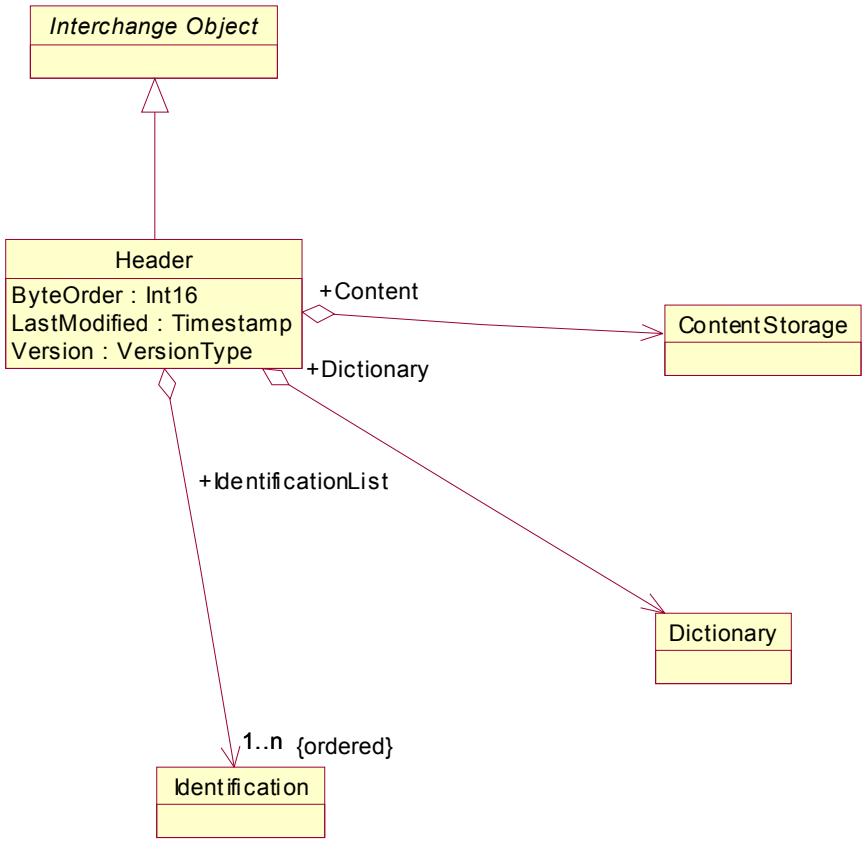
Property Name	Type	Explanation
Pref:ActiveState	PrefT:Boolean	Specifies whether the event is turned on or off. Required.

An 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.

Header Class

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

The Header class is a subclass of the InterchangeObject class.



The Header object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation

`Pref:ByteOrder`	`PrefT:Int16`	Specifies the byte order for the AAF file. One of the following:
		`'II'` `Little-endian byte order`
		`'MM'` `Big-endian byte order`
		Required.

`Pref:LastModified`	`PrefT:TimeStamp`	Time and Date the file was last modified. Required.
`Pref:Content`	`PrefT:StrongReference` to ContentStorage	Has the ContentStorage object that has all Packages and Essence Data in the file. Required.

Property Name	Type	Explanation
`Pref:Dictionary`	`PrefT:StrongReference` to Dictionary	Has a Dictionary object that has the DefinitionObjects that define the classes, control codes, data definitions, effects, properties, and types, Data Definition and Effect Definition objects defined in the AAF file. Required.
`Pref:Version`	`PrefT:VersionType`	Version number of this document that the file conforms to; shall be 1.0 or higher. Required.
`Pref:IdentificationList`	`PrefT: StrongReferenceVector` of Identification	Has an ordered set of Identification objects, which identify the application that created or modified the AAF file. Required.

1. Each Edit Interchange file shall have exactly one Header object.

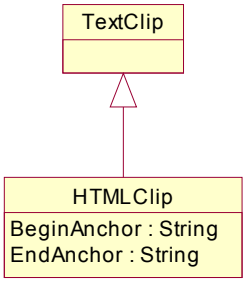
Note 1 The value of the ByteOrder property is either 'MM' (hexadecimal 0x4d4d) for big-endian byte order, which is used in some architectures such as the Motorola 680x0 architecture, or 'II' (hexadecimal 0x4949) for little-endian byte order, which is used in some architectures such as the Intel x86 architecture. Big-endian and little-endian refer to whether the most- or least-significant byte is stored first. In the big-endian byte order, the most-significant byte is stored first (at the address specified, which is the lowest address of the series of bytes that constitute the value). In the little-endian byte order, the least-significant byte is stored first. In both cases, bytes are stored with the most-significant bit first.

Note 2 The value of LastModified represents the last time the file was modified.

HTMLClip Class

HTMLClip is a reference to HTML text essence.

The HTMLClip class is a subclass of the TextClip class.



An HTMLClip object may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:BeginAnchor`	`PrefT:String`	Specifies the HTML tag that defines the start of the text. Optional.
`Pref:EndAnchor`	`PrefT:String`	Specifies the HTML tag that defines the end of the text. Optional.

Typically an HTMLClip is in a StaticSlot and defines a section of HTML text that is associated with the essence data in a parallel TimelineSlot. The duration of the HTMLClip defines the extent of the association with the parallel Package Slot.

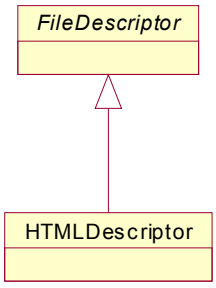
The BeginAnchor and EndAnchor properties specify the HTML tags that delineate the start and end of the referenced text. The BeginAnchor tag shall precede the EndAnchor tag. If the BeginAnchor and EndAnchor properties are omitted, the HTMLClip references all the HTML text in the essence data object.

An HTMLClip object has an association with a Slot object describing HTML essence data. .

HTMLDescriptor Class

HTMLDescriptor specifies that the essence data is in HTML text format.

HTMLDescriptor is a subclass of FileDescriptor. An HTMLDescriptor object is owned by a File SourcePackage object.



HTMLDescriptor does not specify any properties.

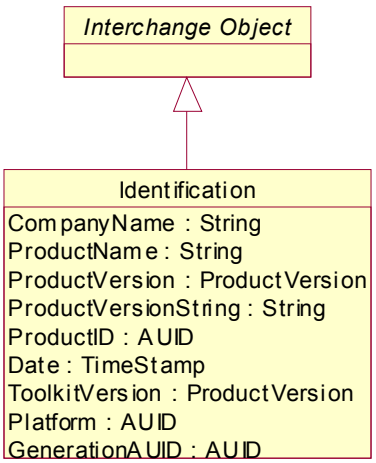
An HTMLDescriptor object specifies that the File SourcePackage describes an HTML object, which contains text, formatted according to the HTML standard.

Identification Class

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

Identification is a subclass of InterchangeObject.

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



An Identification object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:GenerationAUID`	`PrefT:AUID`	AUID generated at the time the application created or opened for modification the file. Required.
`Pref:CompanyName`	`PrefT:String`	Specifies the name of the company or organization that created the application. Required.
`Pref:ProductName`	`PrefT:String`	Specifies the name of the application. Required.
`Pref:ProductVersion`	`PrefT:ProductVersion`	Specifies the version number of the application. Consists of 5 16-bit integer values that specify the version of an application. The first four integers specify the major, minor, tertiary, and patch version numbers. The fifth integer has the following values:

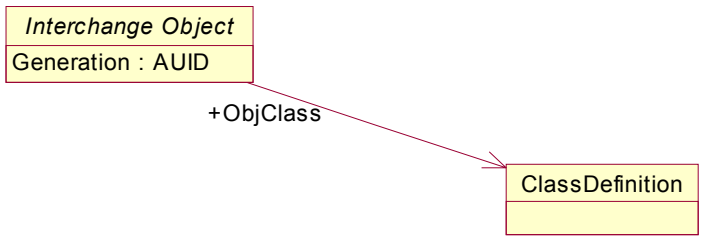
Property Name	Type	Explanation
		0 kVersionUnknown: No additional version information 1 kVersionReleased: Released product 2 kVersionDebug: Development version 3 kVersionPatched: Released product with patches 4 kVersionBeta: Prerelease beta test version 5 kVersionPrivateBuild: Private build Optional.
`Pref: ProductVersionString`	`PrefT:String`	Specifies the version number of the application in string form. Required.
`Pref:ProductID`	`PrefT:AUID`	Identifies the application. Required.
`Pref:Date`	`PrefT:TimeStamp`	Time and date the application created or modified the AAF file. Required.
`Pref:SDKVersion`	`PrefT:ProductVersion`	Specifies the version number of the SDK library. Optional.
`Pref:Platform`	`PrefT:String`	Specifies the platform on which the application is running. Optional.

The Identification class identifies the application that created or modified the file.

InterchangeObject Class

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

The InterchangeObject class is an abstract class; consequently an object that belongs to the InterchangeObject class shall also belong to a subclass of the InterchangeObject class.



An InterchangeObject shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:ObjClass`	`PrefT:WeakReference to ClassDefinition`	Identifies the class of the object. Required.
`Pref:Generation`	`PrefT:AUID`	Identifies when the object was created or last modified. Optional. If omitted, the object was created or last modified in the first generation of the file.

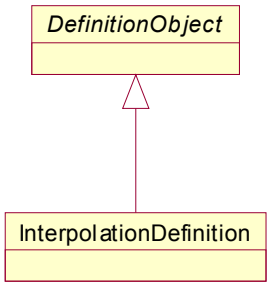
.

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 subclass of the DefinitionObject class.

All InterpolationDefinition objects shall be owned by a Dictionary object.

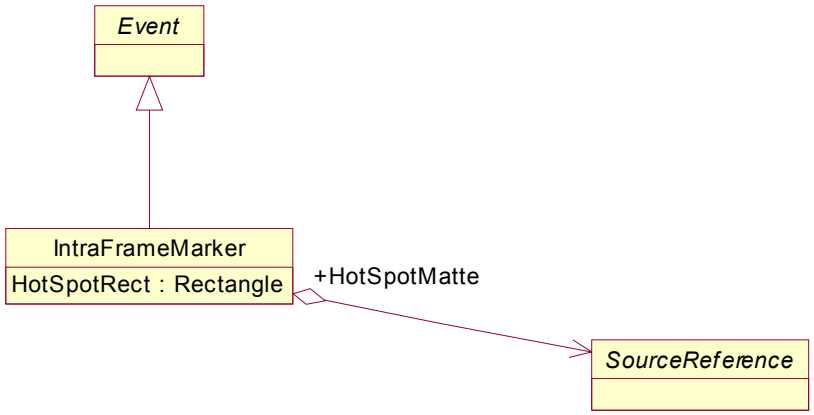


The InterpolationDefinition class does not define any additional properties.

IntraFrameMarker Class

IntraFrameMarker specifies an area in an image that can cause an action if a user specifies it during composition playback (typically by clicking in the specified area).

IntraFrameMarker is a subclass of Event. IntraFrameMarker objects are owned by a Sequence in an EventSlot.



An IntraframeMarker object shall have the properties listed in the following table as specified by the rules following the table.

Property Name	Type	Explanation
`Pref:HotSpotRect`	`PrefT:Rectangle`	Specifies an area on the image that can cause an interactive action. Optional.
`Pref:HotSpotMatte`	`PrefT:StrongReference` to SourceClip	Specifies a SourceClip with a datadefinition of Matte. Optional

1. An IntraFrameMarker object shall have at least one of the following properties: HotSpotRect and HotSpotMatte.

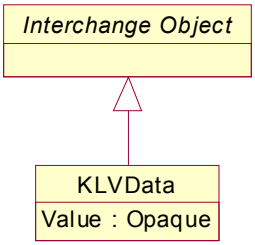
2. If an IntraFrameMarker object has both a HotSpotRect and a HotSpotMatte property, the HotSpotMatte property defines the interactive region. In this case, the HotSpotRect provides supplementary information.

An IntraframeMarker defines a region of an image that has an interactive event associated with it.

KLVData Class

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

KLVData is a subclass of InterchangeObject.



A KLVData object shall have the required properties listed in the following table.

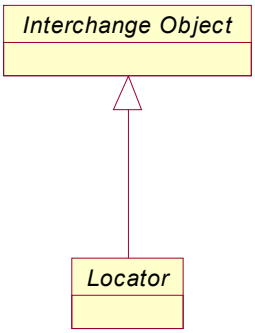
Property Name	Type	Explanation
`Pref:Value`	`PrefT:Opaque`	Specifies the key, length, and value. Required.

Locator Class

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

The Locator class is a subclass of the InterchangeObject class.

The Locator class is an abstract class; consequently object that belongs to the Locator class shall also belong to a subclass of Locator.



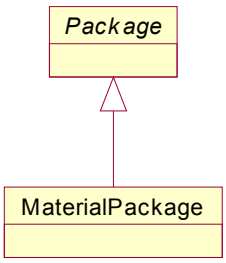
The Locator class does not define any additional properties.

MaterialPackage Class

The MaterialPackage class provides access to the File Source Packages and EssenceData Objects.

The MaterialPackage class is a subclass of the Package class.

All MaterialPackage objects in a file shall be owned by the ContentStorage object.



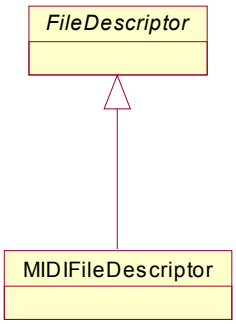
The MaterialPackage class does not define any additional properties.

An MaterialPackage object shall have one or more Slots.

MIDIFileDescriptor class

MIDIFileDescriptor specifies that essence data is in the MIDI music file format.

MIDIFileDescriptor is a subclass of FileDescriptor. MIDIFileDescriptor objects are owned by a File Source Package object.



MIDIFileDescriptor does not define any properties.

An MIDIFileDescriptor describes a MIDI object that contains multi-track MIDI stream data stored in the Standard MIDI File 1.0 format. The MIDI file data may be stored with either timecode-based times or metrical times. (Metrical times are stored as subdivisions of a quarter note, and related to real time by a tempo map).

If the MIDI file data is stored with timecode-based times, the SampleRate property of the MIDIFileDescriptor Class shall specify the MIDI file's delta-time interval expressed as an edit rate. Referring to the MIDI file specification, this information is specified in the <division> word of the header

chunk. The SampleRate is the sample rate corresponding to the MIDI format specified in the <division> word, multiplied by the "ticks per frame" value specified in the <division> word.

If the MIDI file data is stored with metrical times, the tempo map stored in the MIDI file data provides a conversion from each metrical time to a real time (in seconds). The MIDI file will be referenced from Packages using only real times. The application must make an implicit conversion from the metrical times to real time to determine which section of the MIDI file data is being referenced. As such, the SampleRate property may be any arbitrary value. It merely provides a reference rate for specifying the length of the MIDI file, and a resolution for identifying locations within the MIDI file by real time. We recommend using a rate that matches the edit rate of tracks in CompositionPackages that reference MIDI data.

As the MIDI file specification states, tempo meta-events describing the tempo map must be stored in the first track of the MIDI file. If there is no tempo meta-event at the beginning of first track, a default tempo of 120 quarter notes per minute is assumed.

The MIDIFileDescriptor Length property should be the maximum of the times of the End of Track meta-events for all tracks in the MIDI file, expressed in the SampleRate.

Tracks in MIDI files

A MIDI file may contain multiple tracks. The associated SourcePackage must have a Slot for each track stored in the MIDI file. The Slot PhysicalTrack property identifies the corresponding track within the MIDI file, beginning with 1 for the first track in the MIDI file.

The Slot SlotName property provides the name of the track. The names stored in Track Name meta-events within the MIDI file are ignored. It is recommended that the Track Name meta-events, if present, match the TrackName properties of the Slots.

Each Slot of the MIDIFileSourcePackage should have as its Segment a single SourceClip that does not have a SourceID property, a DataDefinition of MIDI, and a length corresponding to the position of the End of Track meta-event stored in the corresponding track of the MIDI file data.

Each Slot of the MIDIFileSourcePackage should have an EditRate that matches the SampleRate of the MIDIFileDescriptor (MIDD) attached to that SourcePackage.

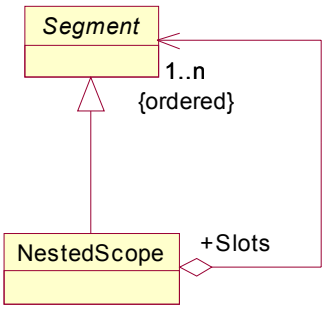
SMPTE timecode

The SMPTE Offset meta-event in the MIDI file data is ignored. It is recommended that this timecode information be represented by a timecode track within the SourcePackage associated with the MIDI file data.

NestedScope Class

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

The NestedScope class is a subclass of the Segment class.



A NestedScope object shall the required properties listed in the following table.

Property Name	Type	Explanation
`Pref:Slots`	`PrefT: StrongReferenceVector` of Segment	Has an ordered set of Segments; the last segment provides the value for the Nested Scope object. Required.

1. The length of each Segment object in the set must be equal to the length of the Nested Scope object.

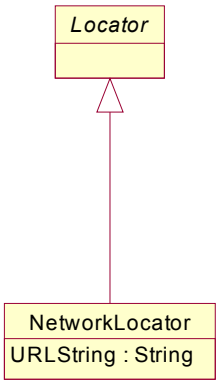
2. The data kind of the last Segment in the set must be the same as the data kind of the Nested Scope object.

Note 1 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 Packages 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 Package.

NetworkLocator Class

NetworkLocator provides information to help find a file containing essence data.

NetworkLocator is a subclass of Locator. Locators can be used in FileDescriptors, which are owned by FileSourcePackages.



Property Name	Type	Explanation
`Pref:URLString`	`PrefT:String`	Universal Resource Locator (URL) for file containing the essence data. Required.

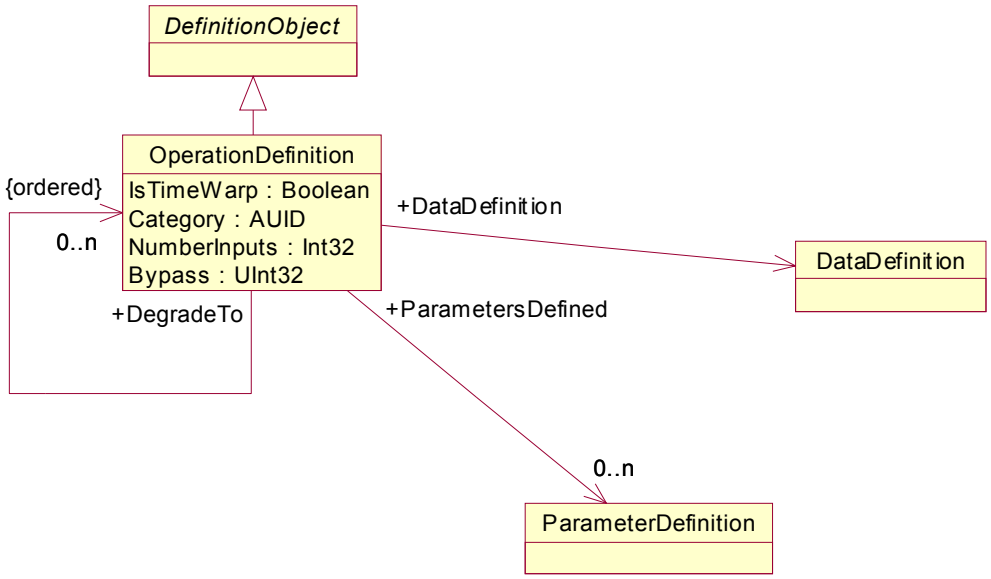
The NetworkLocator has a URL that provides a hint to help an application find a file containing the essence data.

OperationDefinition Class

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

The OperationDefinition class is a subclass of the DefinitionObject class.

All OperationDefinition objects shall be owned by a Dictionary object.



An OperationDefinition object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:DataDefinition`	`PrefT:WeakReference` to DataDefinition	Identifies the kind of data that is produced by the operation. Required.
`Pref:IsTimeWarp`	`PrefT:Boolean`	If true, specifies that the duration of the input segments can be different from the duration of the Operation. Optional; default value is false.
`Pref:DegradeTo`	`PrefT:WeakReferenceVector` to OperationDefinition	Specify simpler operations that an application can substitute for the defined operation if it cannot process it. Optional
`Pref:Category`	`PrefT:AUID`	Specifies the kind of operation, such as Video Effect, Audio Effect, or 3D operation. Required.
`Pref:NumberInputs`	`PrefT:Int32`	Specifies the number of input segments. A value of -1 indicates that the effect can have any number of input segments. Required.
`Pref:Bypass`	`PrefT:Uint32`	Specifies then array index (1-based) of the input segment which is the primary input. Optional.

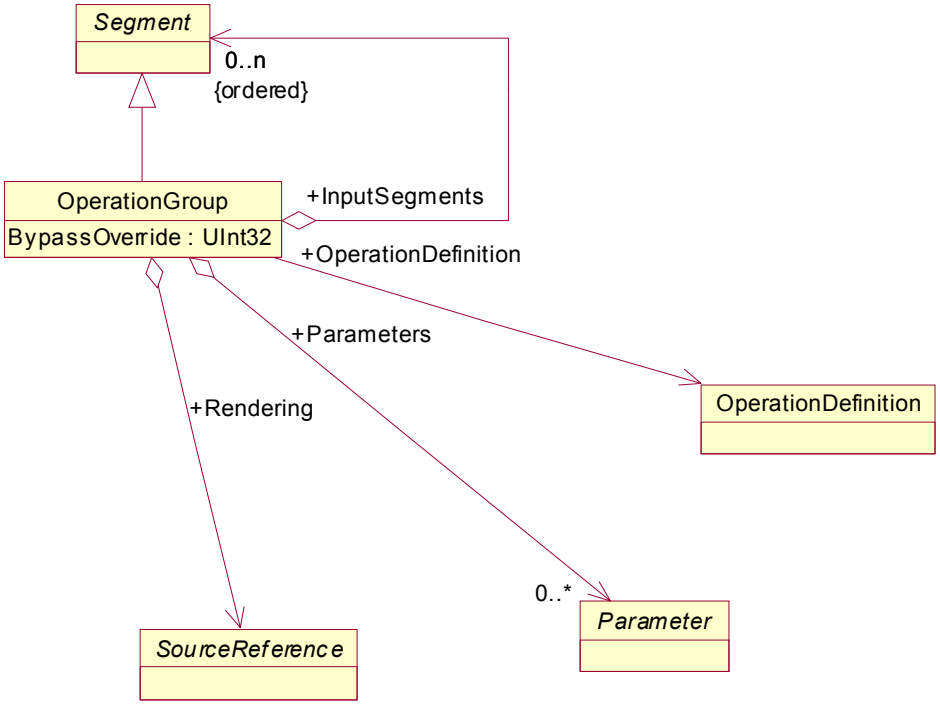
Property Name	Type	Explanation
`Pref:ParametersDefined`	`PrefT: WeakReferenceSet` of ParameterDefinition	Specify the Parameters that can be used as controls for the operation. Optional

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 subclass of the Segment class

An OperationGroup object can only be part of a Composition Package.



An Effect object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation

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

1. In all OperationGroup objects, the length of the Rendering Source Clip must each equal the length of the OperationGroup.

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

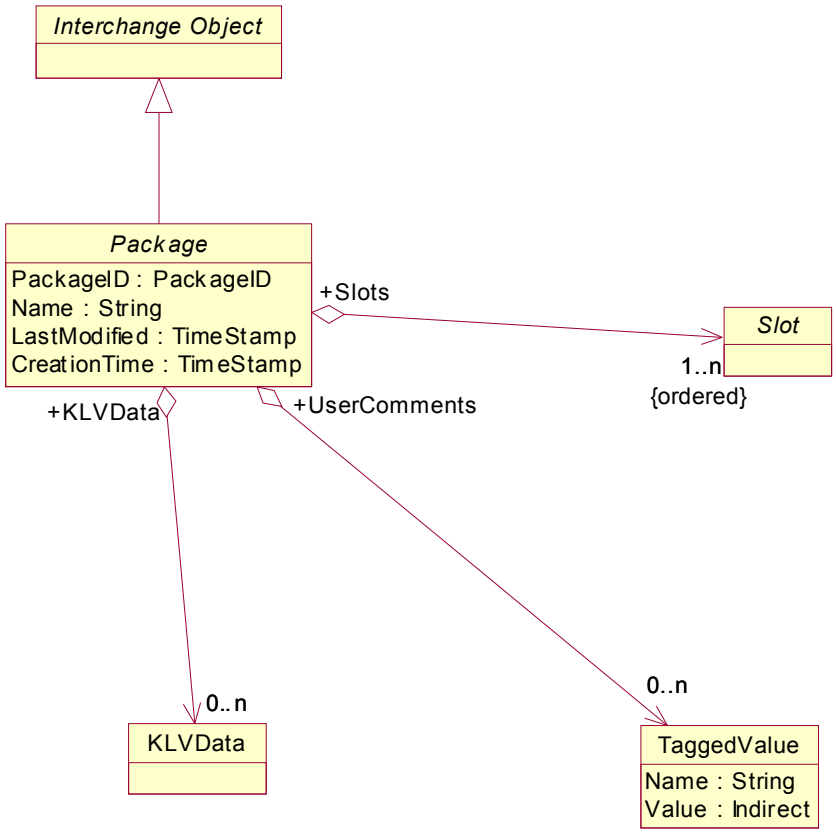
3. In OperationGroup that is in a Transition, the input segments are provided by the Transition and the InputSegments property should be omitted.

Package Class

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

The Package class is a subclass of the InterchangeObject class.

The Package class is an abstract class; consequently an object that belongs to the Package class shall also belong to a subclass of Package.



A Package object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:PackageID`	`PrefT:PackageID`	Unique Package Identification. Required.
`Pref:Name`	`PrefT:String`	Name of Package for display to end user. Optional.
`Pref:Slots`	`PrefT:StrongReferenceVector` of Slots	Has an ordered set of Slots. Required.
`Pref:LastModified`	`PrefT:TimeStamp`	Date and time when the Package was last modified. Required.
`Pref:CreationTime`	`PrefT:TimeStamp`	Date and time when the Package was originally created. Required.

Property Name	Type	Explanation
`Pref:UserComments`	`PrefT: StrongReferenceVector` of TaggedValues	Has a set of TaggedValues that specify user comments. Optional.
`Pref:KLVData`	`PrefT: StrongReferenceVector of KLVData`	Contains a set of user KLV data consisting of a key (a SMPTE label), a length, and a value. Optional.

1. A Package object shall have one or more Slots.

Note 1 Packages have a globally unique ID, and they are the only elements of an AAF file that can be referenced from outside the file.

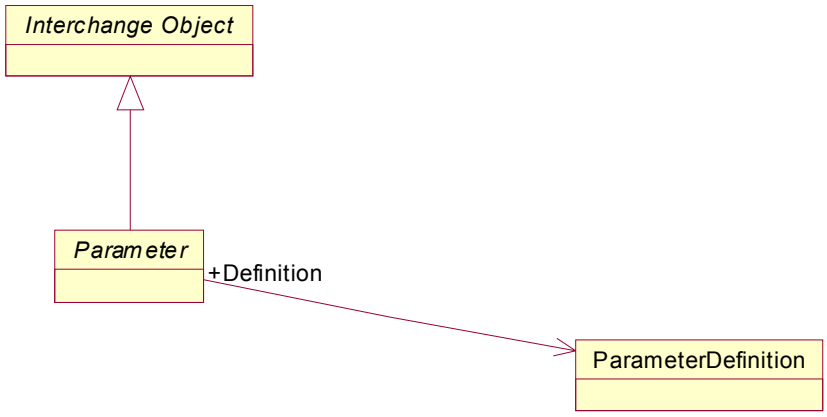
Note 2 Slots are ordered to allow ScopeReferences within one slot to reference another slot. The Package defines a scope consisting of the ordered set of Slots. A ScopeReference object in a Slot can refer to any Slot that precedes it. A ScopeReference returns the same time-varying values as the section in the specified Package Slot that corresponds to the starting point of the ScopeReference in the Slot and the duration of the ScopeReference. In addition to Packages, NestedScope objects define scopes; however, their scope is limited to the Components owned by the Nested scope object's slots.

Parameter Class

Parameter class specifies a control argument for an effect.

Parameter class is a subclass of Object. Parameter objects are owned by Effect objects.

Parameter is an abstract class; consequently, any object that belongs to the Parameter class shall also belong to a subclass of Parameter.



A Parameter object shall have the required properties listed in the following table.

Property Name	Type	Explanation
`Pref:Definition`	`PrefT:WeakReference` to ParameterDefinition	Identifies the Parameter. Required.

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.

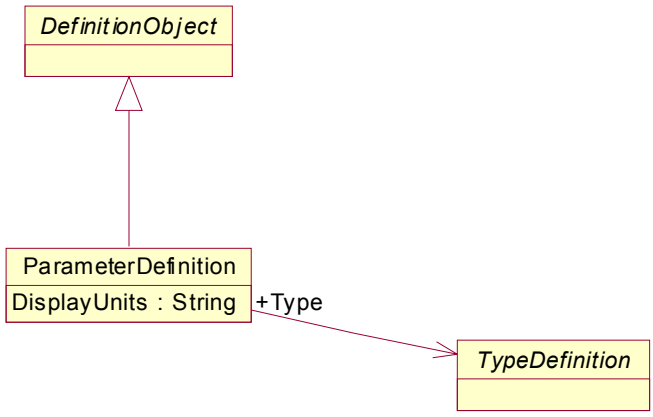
A constant control argument can be specified with a Constant Value object. A time-varying value is specified with a Varying Value object.

ParameterDefinition Class

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

The ParameterDefinition class is a subclass of the DefinitionObject class.

All ParameterDefinition objects in a file shall be owned by the Dictionary object.



A ParameterDefinition object shall have the required properties and may have the optional properties listed in the following table.

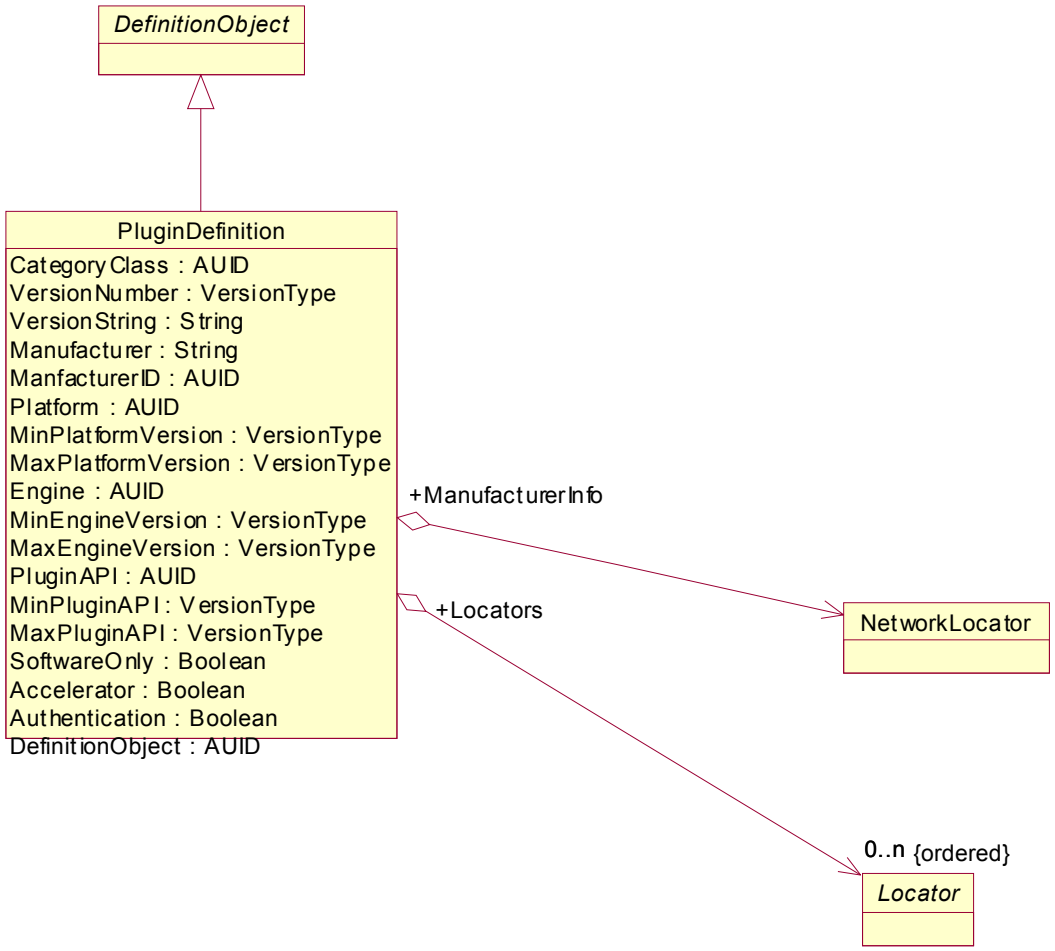
Property Name	Type	Explanation
`Pref:Type`	`PrefT:WeakReference` to TypeDefinition	Specifies the data type of the Parameter. Required.

The value of a Parameter is specified in a property with an Indirect type. The Indirect type must 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 subclass of Parameter.

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 subclass of the DefinitionObject class.



All PluginDefinition objects shall be owned by a Dictionary object. A PluginDefinition object shall have the required properties and may have the optional classes listed in the following table

Property Name	Type	Explanation
`Pref:CategoryClass`	`PrefT: AUID`	Specifies the kind of plugin. Required.
`Pref:VersionNumber`	`PrefT: VersionType`	Specifies the version of the plugin. Required.
`Pref:VersionString`	`PrefT:String`	Specifies a string that can be used to identify the plugin version to the user. Optional.
`Pref:Manufacturer`	`PrefT:String`	Specifies a string that can be used to identify the plugin manufacturer to the user. Optional.

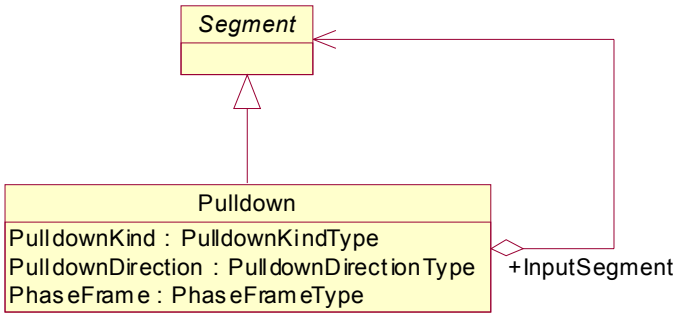
Property Name	Type	Explanation
Pref: ManufacturerInfo	PrefT: StrongReference to NetworkLocator	Specifies a NetworkLocator that identifies a web page containing information about the manufacturer. Optional.
Pref:ManufacturerID	PrefT:AUID	Specifies a SMPTE label or other unique identifier that is assigned to identify the manufacturer. Optional.
Pref:Platform	PrefT:AUID	Identifies the platform environment, which consists of the hardware platform and the operating system, required by the plugin. Optional.
Pref: MinPlatformVersion	PrefT: VersionType	Specifies the minimum version number of the specified platform that the plugin requires. Optional.
Pref: MaxPlatformVersion	PrefT: VersionType	Specifies the maximum version number of the specified platform that the plugin requires. Optional.
Pref:Engine	PrefT:AUID	Identifies the software subsystem used for essence management and playback used by the plugin. Optional.
Pref: MinEngineVersion	PrefT: VersionType	Specifies the minimum version number of the specified engine that the plugin requires. Optional.
Pref: MaxEngineVersion	PrefT: VersionType	Specifies the maximum version number of the specified engine that the plugin requires. Optional.
Pref:PluginAPI	PrefT:AUID	Identifies the plugin interfaces supported by the plugin. Optional.
Pref:MinPluginAPI	PrefT: VersionType	Specifies the minimum version number of the specified plugin interfaces that the plugin supports. Optional.
Pref:MaxPluginAPT	PrefT: VersionType	Specifies the maximum version number of the specified plugin interfaces that the plugin supports. Optional.
Pref:SoftwareOnly	PrefT:Boolean	Specifies if the plugin is capable of executing in a software-only environment. Optional; default value is False.
Pref:Accelerator	PrefT:Boolean	Specifies if the plugin is capable of using hardware to accelerate essence processing. Optional; default value is False.
Pref:Locators	PrefT: StrongReferenceVector of Locator	Specifies an ordered list of locators that identify locations that provide access to the plugin implementation. Optional.
Pref:Authentication	PrefT:Boolean	Specifies that the plugin implementation supports authentication. Optional; default value is False.

Property Name	Type	Explanation
`Pref: DefinitionObject`	`AUID`	Specifies the AUID of the ClassDefinition for the DefinitionObject or MetaDefinition that it provides an implementation of. Required.

Pulldown Class

Pulldown converts between film frame rates and videotape frame rates.

Pulldown is a subclass of Segment. Pulldown has either a SourceClip or a Timecode object. Pulldown objects are typically used in FileSourcePackages and Physical SourcePackages.



A Pulldown object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:InputSegment`	`PrefT:StrongReference` to Segment	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. Required

Property Name	Type	Explanation
Pref:PulldownKind	PrefT:PulldownKindType	Specifies whether the Pulldown object is converting from 525-line (30000/1001 frames per second) or 625-line (24 frames per second) and whether frames are dropped or the video is played at another speed. Values are: 0 kTwoThreePD Converting between 525-line and film by dropping or adding frames 1 kPalPD Converting between 625-line and film by dropping or adding frames 2 kOneToOneNTSC Converting between 525-line and film by speeding up or slowing down the frame rate. 3 kOneToOnePAL Converting between 625-line and film by speeding up or slowing down the frame rate. 4 kVideoTapNTSC Converting between 525-line and film by recording original film and video sources simultaneously. Required.
Pref: PulldownDirection	PrefT: PulldownDirectionType	Specifies whether the Pulldown object is converting from tape to film speed or from film to tape speed. Values are: 0 kVideoToFilmSpeed The InputSegment is at video speed and the Package track owning the Pulldown object is at film speed. 1 kFilmToVideoSpeed The InputSegment is at film speed and the Package track owning the Pulldown object is at video speed. Required.
Pref:PhaseFrame	PrefT: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. Required.

An 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. Pulldown objects are typically used in three ways:

1. In a tape SourcePackage to describe how the videotape was created from film
2. In a file SourcePackage that has digital essence at film speed to describe how the digital essence was created from the videotape

3. In a Package 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.

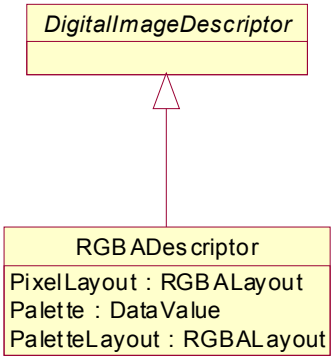
Each kind of pulldown identifies the speed of the tape. If two SourcePackages have a pulldown relationship, the edit rates of the video tracks should correspond to the frame rate of the essence.

RGBADescriptor Class

The RGBADescriptor class specifies that a File Source Package is associated with video content data formatted with three color component or with three color components and an alpha component.

The RGBADescriptor class is a subclass of the DigitalImageDescriptor class.

An RGBADescriptor object shall be the EssenceDescription in a File Source Package.



An RGBADescriptor object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:PixelLayout`	`PrefT: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 For each component in the pixel, the following codes should be specified: 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 A Fill component indicates unused bits. After the components have been specified, the remaining Code and Size fields should be set to 0. Required.
`Pref:Palette`	`PrefT:DataValue`	An array of color values that are used to specify an image. Optional.
`Pref:PaletteLayout`	`PrefT:RGBALayout`	An array of RGBAComponent that specifies the order and size of the color components as they are stored in the palette. Optional.

1. 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'.

2. If the PixelLayout property includes a 'P', then the RGBADescriptor object shall have the Palette, PaletteLayout, and PaletteStructure properties.

Note 1 An RGBADescriptor object describes content data 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.

Note 2 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.

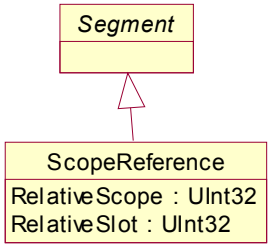
Note 3 If a color palette is used, the descriptor allows you to specify the color palette and the structure used to store each color in the palette.

Note 4 RGBA content data can be converted to CDCI and then compressed with JPEG. Once the data has been converted and compressed, it is described by a CDCIDescriptor Essence Descriptor.

ScopeReference Class

The ScopeReference class refers to a section in the specified Package Slot or Nested Scope slot.

The ScopeReference class is a subclass of the Segment class.



An ScopeReference object shall have the required properties listed in the following table.

Property Name	Type	Explanation
`Pref:RelativeScope`	`PrefT:UInt32`	Specifies the number of Nested Scopes to pass to find the Nested Scope or Package owning the slot. Required.
`Pref:RelativeSlot`	`PrefT:UInt32`	Specifies the number of slots that precede the slot owning the Scope Reference to pass to find the slot referenced. Required.

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

2. The value of RelativeScope must be less than or equal to the number of Nested Scope objects that 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 Package's scope and the RelativeScope must have a value of 0.

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

4. If the scope reference references a Package slot that specifies a different edit rate than the Package slot owning the scope reference, the Length value and the offset in the track owning the scope reference must be converted from the edit rate of the track owning the scope reference to the edit rate of the referenced track.

Note 1 A Scope Reference object has the same time-varying values as the section of the Nested Scope slot or Package Slot that it references. Scope Reference objects allow one or more objects to share the values produced by a section of a slot.

Note 2 If a Scope Reference specifies a Package slot, the corresponding section of the slot is the one that has the equivalent starting position from the beginning of the Package slot and the equivalent length as the Scope Reference object has within its Package slot. If the specified Package Slot has a different edit rate than the Package Slot owning the Scope Reference, the starting position and duration are converted to the specified Package Slots edit units to find the corresponding section.

Note 3 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.

Note 4 Relative scope is specified as an unsigned integer. It specifies the number of nested scopes that you must pass through to find the referenced scope. A value of 0 specifies the current scope, that is the innermost Nested Scope object that has the Scope Reference or the Package 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.

Note 5 Relative slot is specified as a positive integer. It specifies the number of preceding slots that you must pass to find the referenced slot within the specified relative scope. A value of 1 specifies the immediately preceding slot.

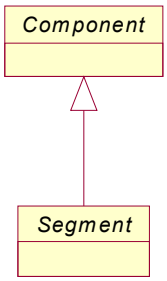
Note 6 If a ScopeReference refers to a Slot, the Slot shall belong to the same subclass of Slot as the Slot owning the ScopeReference object. This means that ScopeReferences should not be used to convert between timeline, static, and event data; use SourceClips or SourceClips in conjunction with Effect to perform these conversions.

Segment Class

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

The Segment class is a subclass of the Component class.

The Segment class is an abstract class; consequently an object that belongs to the Segment class shall also belong to one of the subclasses of Segment.

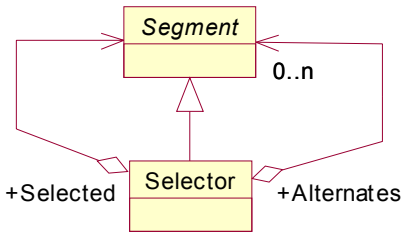


Segment does not define any additional properties.

Selector Class

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

The Selector class is a subclass of the Segment class.



The Selector class shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:Selected`	`PrefT: StrongReference` to Segment	Has the selected Segment. Required.
`Pref:Alternates`	`PrefT: StrongReferenceVector` of Segment	Has a set of unused alternative Segments. Optional.

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

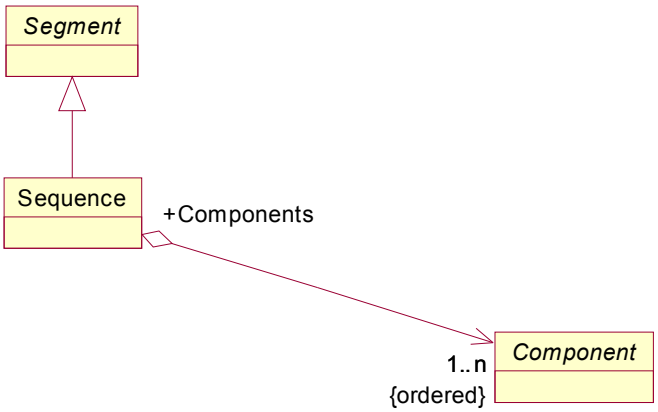
2. The data kind of the selected Segment and of each alternative Segment shall be the same as the data kind of the Selector object.

Note A Selector object represents an editing decision. This is in contrast with a 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.

Sequence Class

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

The Sequence class is a subclass of Segment.



A Sequence object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:Components`	`PrefT:StrongReferenceVector` of Component	Has an ordered set of Component objects. Required.

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

2. 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.

3. 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.

4. 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.

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

Note 1 The Sequence object is the mechanism for combining sections of essence to be played in a sequential manner.

Note 2 If a Sequence object has a Segment followed by another Segment, after the first Segment is played, the following one begins immediately

Note 3 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.

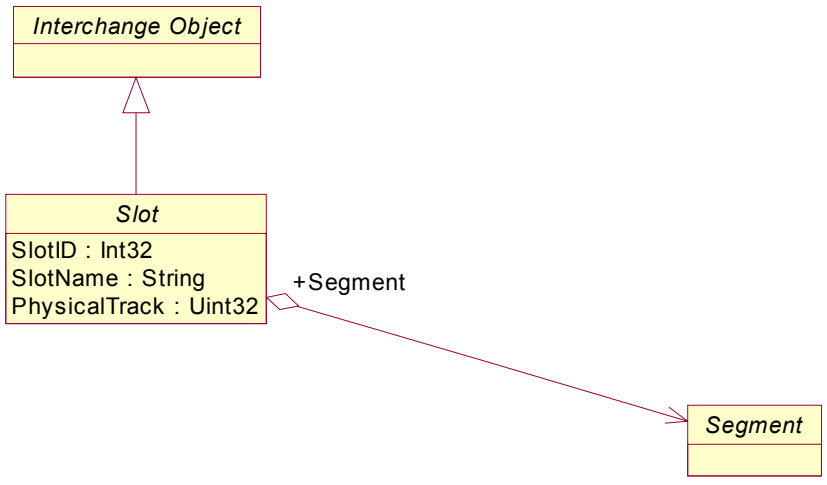
Slot Class

The Slot class represents a Segment of essence in a Package. Slot objects are owned by Packages. A Slot object has a Segment, which can be a timeline, static, or event Segment.

The Slot class is a subclass of the InterchangeObject class.

All Slot objects shall be members of the set of Slots of a Package object.

The Slot class is an abstract class; consequently, any object that belongs to the Slot class shall also belong to a subclass of Slot.



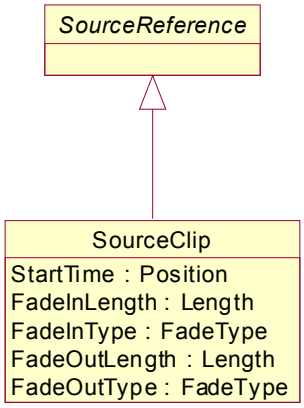
An Slot object shall have the required properties and may have the optional properties described in the following table.

Property Name	Type	Explanation
Pref:SlotID	PrefT:Int32	Specifies an integer that is used to reference the Package Slot. Required.
Pref:SlotName	PrefT:String	Specifies a text name for the Slot. Optional.
Pref:Segment	PrefT: StrongReference to Segment	Specifies the value of the Slot. Required.
Pref: PhysicalTrackNumber	PrefT:UInt32	Specifies the physical channel. Optional.

SourceClip Class

The SourceClip class represents the content data and identifies the source of the content data.

The SourceClip class is a subclass of the SourceReference class.



A SourceClip object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
Pref:StartTime	PrefT:Position	Specifies the offset from the origin of the referenced Package Slot in edit units determined by the Source Clip object's context. If the SourceID has a value 0, then StartTime must have a 0 value. Optional; see rule 1.

Property Name	Type	Explanation
`Pref:FadeInLength`	`PrefT:Length`	Specifies the length of an audio fade in to be applied to the Source Clip. Optional.
`Pref:FadeInType`	`PrefT:FadeType`	Specifies the type of the audio fade in; may have one of the following values: 0 No fade 1 Linear amplitude fade 2 Linear power fade 3 Linear dB fade Additional registered and private fade in types may be defined. Optional.
`Pref:FadeOutLength`	`PrefT:Length`	Specifies the length of an audio fade out to be applied to the Source Clip. Optional
`Pref:FadeOutType`	`PrefT:FadeType`	Specifies the type of the audio fade out; may have one of the following values: 0 No fade 1 Linear amplitude fade 2 Linear power fade 3 Linear dB fade Additional registered and private audio fade types may be defined. Optional.

1. If the SourceClip references a TimelineSlot or an EventSlot, then the StartTime property shall be specified. If the SourceClip references a StaticSlot, then the StartTime property shall not be specified.

2. The data definition of the Segment owned by the referenced Package Slot shall be the same the data definition of the Source Clip object.

3. The fade properties are only allowed when the Component DataDefinition specifies Sound.

4. If the source clip references a track that specifies a different edit rate than the track owning the source clip, the StartTime and Length values must be converted from the edit rate of the track owning the source clip to the edit rate of the referenced track.

Note 1 In a Composition Package, Source Clips reference a section of essence by specifying the Material Package that describes the essence.

Note 2 In a Material Package, Source Clips reference the essence by specifying the File Source Package that is associated with the essence.

Note 3 In a File Source Package, Source clips reference the content data stored on a physical media, such as tape or film, by specifying the Physical Source Package that describes the media.

Note 4 In a Physical Source Package, Source Clips reference the content data stored on a previous generation of physical media by specifying the Physical Source Package that describes the media.

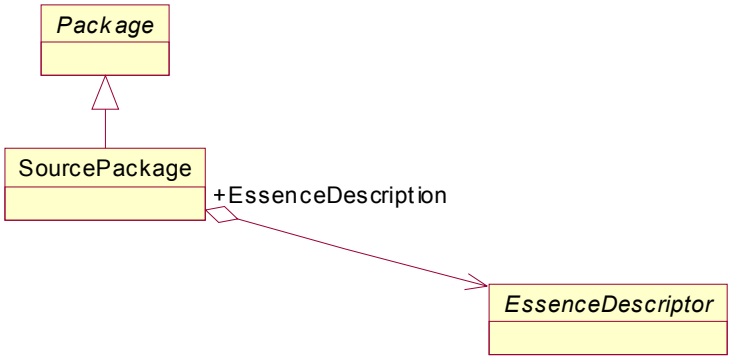
Note 5 If a Source Package represents the original essence source and there is no previous generation, then its Source Clips must specify a value 0-0-0 for its SourceID and 0 values for SourceTrackID and StartTime.

SourcePackage Class

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

The SourcePackage class is a subclass of the Package class.

All SourcePackage objects shall be owned by the ContentStorage object.



A SourcePackage object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:EssenceDescription`	`PrefT: StrongReference` to EssenceDescriptor	Describes the format of the essence associated with the Source Package. Required.

1. A SourcePackage object shall have one or more Slots.

2. An SourcePackage object shall either be a File Source Package or a Physical Source Package. If a SourcePackage has a Essence Descriptor that belongs to the FileDescriptor class, then the SourcePackage is a File Source Package. If a SourcePackage has a Essence Descriptor that does not belong to the FileDescriptor class, then the SourcePackage is a Physical Source Package.

3. A File Source Package shall have at least one Slot. If the digital essence is a stream of interleaved content data, then the File Source Package shall at least have one Slot for each channel of interleaved content data.

4. A Physical Source Package describes physical media, such as an audio tape, film, or videotape. A Physical Source Package shall have at least one Slot. If the physical media contains more than one track of content data, then the Physical Source Package should have one Slot for each physical track. In addition, the Physical Source Package may have a Slot for timecode data and may have a Slot for edgecode data.

5. The Slots in a File Source Package should have a Segment that is a Source Clip. If there is a Package that describes the previous generation of content data, the Source Clip should specify the PackageID of that Package. The previous generation can be a Physical Source Package or another File Source Package. If there is no previous generation of content data or there is no Package describing it, the Source Clip should not have a SourceID property.

6. The Slot in a Physical Source Package should have a Segment that is a Source Clip, Timecode, or Edgecode. If there is a Package that describes the previous generation of content data, the Source Clip should specify the PackageID of that Package. The previous generation should be a Physical Source Package. If there is no previous generation of content data or there is no Package describing it, the Source Clip should not have a SourceID property.

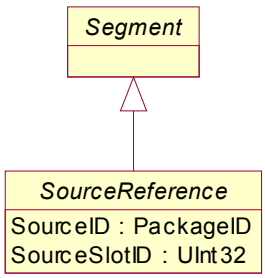
The length of the Segment in the Package Slot indicates the duration of the essence. If you create a Source Package for a physical media source and you do not know the duration of the essence, specify a length of 24 hours.

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

SourceReference Class

SourceReference is an abstract class that represents the essence or other data described by a Slot in a Package.

SourceReference is an abstract class; any object that belongs to the SourceReference class shall also belong to a subclass of SourceReference.



A SourceReference object shall have the required properties and may have the optional properties listed in the following table.

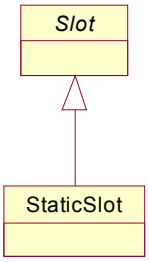
Property Name	Type	Explanation
`Pref:SourceID`	`PrefT:PackageID`	Identifies the Package being referenced. If the property is omitted, it means that the Package owning the SourceReference describes the original source. Optional.
`Pref:SourceSlotID`	`PrefT:UInt32`	Specifies the SlotID of a Package Slot within the specified Package. If the SourceID has a value 0, then SourceTrackID shall also have a 0 value. Required.

An SourceReference object in a Package refers to a Slot in another Package by specifying the second Package's Package ID and the Slot ID of the Slot owned by it. To create a SourceReference that refers to a Slot within the same Package as the SourceReference, specify a weak reference to the containing Package.

StaticSlot Class

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

StaticSlot is a subclass of Slot. Slot objects are owned by Packages.



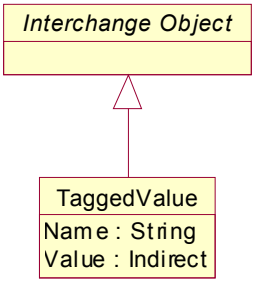
The StaticSlot class does not define any properties.

StaticSlot objects have Segments that do not have any relationship with time; consequently, a StaticSlot does not define an edit rate.

TaggedValue Class

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

The TaggedValue class is a subclass of the InterchangeObject class.



A TaggedValue object shall have the required properties listed in the following table.

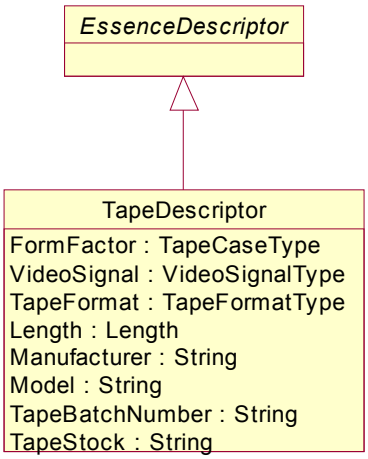
Property Name	Type	Explanation
`Pref:Name`	`PrefT:Sting`	User-specified tag. Optional.
`Pref:Value`	`PrefT:Indirect`	User-specified value. Required.

TapeDescriptor Class

The TapeDescriptor class describes audio tape or videotape media.

The TapeDescriptor class is a subclass of the EssenceDescriptor class.

An TapeDescriptor object shall be the EssenceDescription of a Physical Source Package.



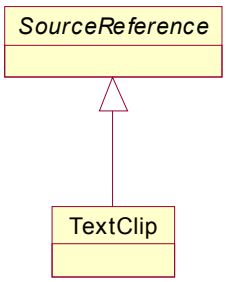
An TapeDescriptor object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
Pref:FormFactor	PrefT:TapeCaseType	Describes the physical size of the tape; may have one of the following values: 0 3/4 inch videotape 1 VHS video tape 2 8mm videotape 3 Betacam videotape 4 Compact cassette 5 DAT cartridge 6 Professional audio tape Optional.
Pref:VideoSignal	PrefT:VideoSignalType	Describes the video signal type; may have one of the following values: 0 NTSC 1 PAL 2 SECAM Optional.
Pref:TapeFormat	PrefT:TapeFormatType	Describes the format of the tape; may have one of the following values: 0 Betacam 1 BetacamSP 2 VHS 3 S-VHS 4 8mm 5 Hi8 Optional.
Pref:Length	PrefT:UInt32	Tape capacity in minutes. Optional.
Pref:Manufacturer	PrefT:String	Text string to display to end users, identifying the manufacturer of the tape. Optional.
Pref:Model	PrefT:String	Text string to display to end users, identifying the manufacturer's brand designation of the tape. Optional.
Pref:TapeBatchNumber	PrefT:String	Specifies the batch number of the tape. Optional
Pref:TapeStock	PrefT:String	Specifies the string identifying the tape stock. Optional.

TextClip Class

TextClip has a weak reference to a Slot describing text essence data.

TextClip is an abstract class and is a subclass of SourceReference.



The TextClip class does not define any properties.

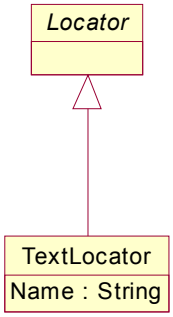
TextClip references a Package Slot containing text essence data.

TextLocator Class

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

The TextLocator class is a subclass of the Locator class.

A TextLocator object shall be the a member of the array of locators in a Essence Descriptor in either a File Source Package or a Physical Source Package.



A TextLocator object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation

Pref:Name	PrefT:String	Text string containing information to help find the file containing the essence or the physical media. Required.

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

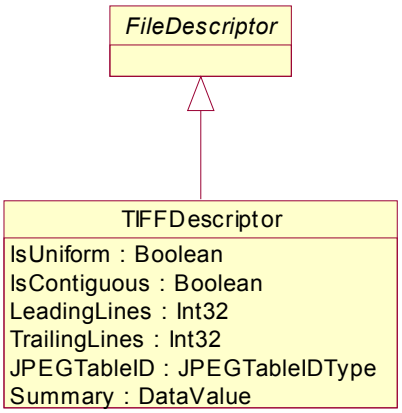
TIFFDescriptor Class

The TIFFDescriptor class specifies that a File Source Package is associated with video content data formatted according to the TIFF specification.

The TIFFDescriptor class is a subclass of the FileDescriptor.

The TIFF video format is a video format which may be used in AAF files, but the TIFF format is not required for compliance with this document. It is preferable to use either the RGBA or CDCI format for video content data.

A TIFFDescriptor object shall be owned by a File Source Package.



A TIFFDescriptor object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
Pref:IsUniform	PrefT:Boolean	True for data having the same number of rows per strip throughout. Required.
Pref:IsContiguous	PrefT:Boolean	True for data stored in contiguous bytes. Required.

Pref:LeadingLines	PrefT:Int32	Number of leading lines to be thrown away. Optional; default value is 0.
Pref:TrailingLines	PrefT:Int32	Number of trailing lines to be thrown away. Optional; default value is 0.
Pref:JPEGTableID	PrefT:JPEGTableIDType	Registered JPEG table code or JT_NULL. Optional.
Pref:Summary	PrefT:DataValue	A copy of the TIFF IFD (without the sample data). Required.

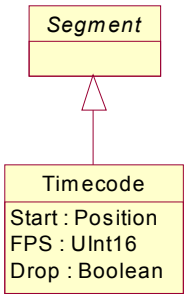
Note 1 A TIFF Image Descriptor object describes the TIFF image data associated with the Source Package. The image data is formatted according to the TIFF specification, Revision 6.0, available from Aldus Corporation. The TIFF object type supports only the subset of the full TIFF 6.0 specification defined as baseline TIFF in that document.

Note 2 The JPEGTableID is an assigned type for preset JPEG tables. The table data must also appear in the TIFFData 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.

Timecode Class

The Timecode class stores videotape or audio tape timecode information.

The Timecode class is a subclass of the Segment class.



A Timecode object shall have the required properties listed in the following table.

Property Name	Type	Explanation
Pref:Start	PrefT:Position	Specifies the timecode at the beginning of the segment. Required.

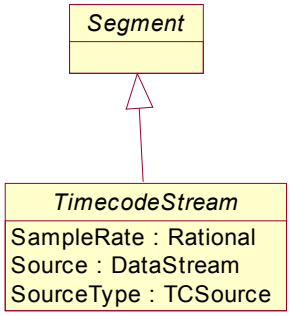
Pref:FPS	PrefT:UInt16	Frames per second of the videotape or audio tape. Required.
Pref:Drop	PrefT:Boolean	Indicates whether the timecode is drop (True value) or nondrop (False value). Required.

Note A Timecode object can typically appear in either a Source Package or in a Composition Package. In a Source Package, it typically appears in a Package Slot in a Source Package that describes a videotape or audio tape. In this context, it describes the timecode that exists on the tape. In a Composition Package, it represents the timecode associated with the virtual media represented by the Composition Package. If the Composition Package is rendered to a videotape, the Timecode should be used to generate the timecode on the videotape.

TimecodeStream Class

TimecodeStream specifies as stream of timecode data.

TimecodeStream is an abstract class and is a subclass of Segment. TimecodeStream always has a timecode DataDefinition. TimecodeStream has a subclass TimecodeStream12M.



A TimcodeStream object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
Pref:SampleRate	PrefT:Rational	Specifies the sample rate of the timecode data contained in the Source property. Required.
Pref:Source	PrefT:DataStream	Contains the timecode data. Required
Pref:SourceType	PrefT:TCSource	Specifies the kind of timecode: 1 LTC timecode 2 VITC timecode

	Required.

TimecodeStream specifies a stream of timecode data.

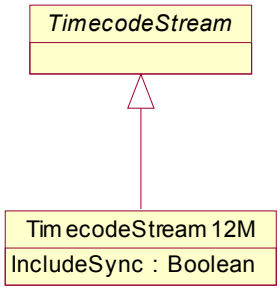
In contrast to TimecodeStream, Timecode specifies a timecode by specifying the starting timecode value; other timecode values are calculated from the starting timecode and the time offset.

TimecodeStream is useful to store user bits that were specified in the timecode on the videotape. It is also useful to store timecode when the timecode does not have a linear relationship with the tape, such as when the tape was accelerating while the essence data was recorded.

TimecodeStream12M Class

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

Timecode12M is a subclass of Timecode. Timecode objects always have a timecode DataDefinition and can be used in aTimelinePackage.



A TimecodeStreaPackageject shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:IncludeSync`	`PrefT:Boolean`	Specifies whether the synchronization data is included in the timecode stream. Required.

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.

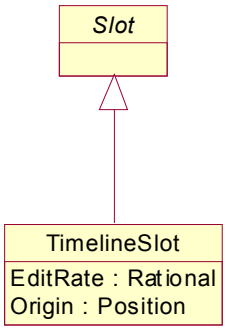
In contrast to TimecodeStream, Timecode specifies a timecode by specifying the starting timecode value; other timecode values are calculated from the starting timecode and the time offset.

TimecodeStream is useful to store user bits that were specified in the timecode on the videotape. It is also useful to store timecode when the timecode does not have a linear relationship with the tape, such as when the tape was accelerating while the essence data was recorded.

TimelineSlot Class

TimelineSlot describes time-varying timeline essence.

TimelineSlot is a subclass of Slot. Slot objects are owned by Package objects.



A TimelineSlot shall have the required properties listed in the following table.

Property Name	Type	Explanation
`Pref:EditRate`	`PrefT:Rational`	Specifies the units of time for the TimelineSlot. Required.
`Pref:Origin`	`PrefT:Position`	Specifies the offset used to resolve SourceClip references to this TimelineSlot. Required.

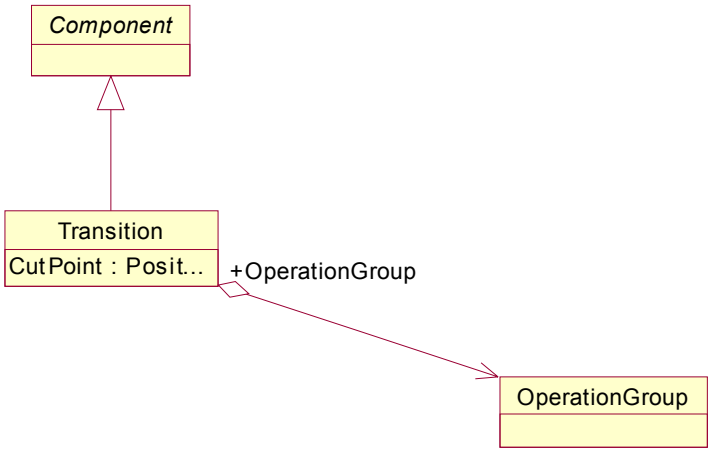
The TimelineSlot specifies the edit rate for the Segment it has. The Segment specifies its length in the edit rate set by the TimelineSlot. The Segment also specifies its own data kind.

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 subclass of the Component class.

A Transition object shall be in a Sequence within a Composition Package.



A Transition object shall have the required properties listed in the following table.

Property Name	Type	Explanation
`Pref: OperationGroup`	`StrongReference` to OperationGroup	Has an OperationGroup that specifies the effect to be performed during the Transition. Required.
`Pref:CutPoint`	`Position`	Specifies a cutpoint to use if replacing the Transition with a cut. Required.

Note 1 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.

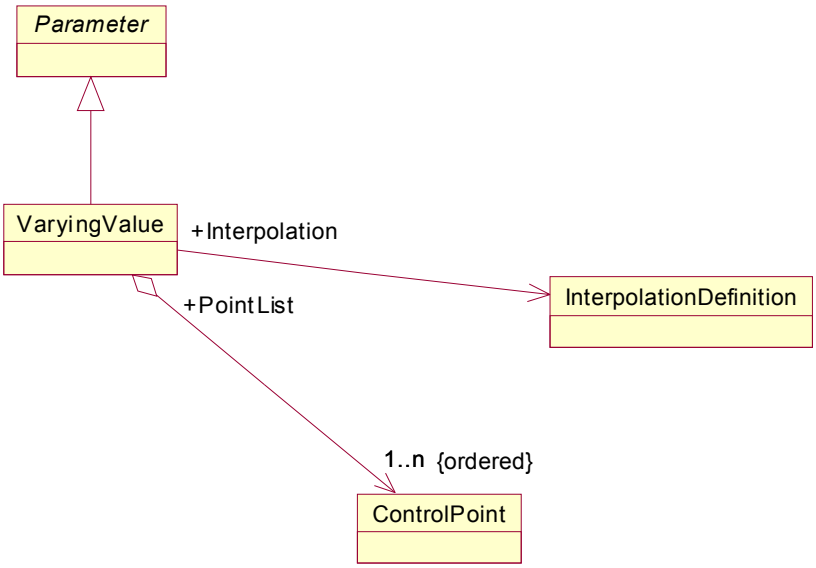
Note 2 The OperationGroup in a Transition shall specify an OperationDefinition with 2 input segments. The OperationGroup in a Transition shall not have the InputSegments specified. The input segments will be provided by the segments preceding and following the Transition.

Note 3 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 Composition Package so that the preceding Segment ends where the cutpoint 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.

VaryingValue Class

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

The VaryingValue class is a subclass of the Parameter class.



A VaryingValue object shall have the required properties listed in the following table.

Property Name	Type	Explanation
`Pref:Interpolation`	`PrefT:WeakReference` to InterpolationDefinition	Specifies the kind of interpolation to be used to find the value between Control Points. Required.
`Pref:PointList`	`PrefT: StrongReferenceVector` of ControlPoint	Has an array of Control Points, each of which specifies a value and a time point at which the value is defined. Required.

1. A VaryingValue object shall have at least one Control Point. 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.

2. Control Points shall be ordered by their time value.

3. A Varying Value object is a Parameter that returns time-varying values that are determined by an ordered set of Control Points. Each Control Point specifies the value for a specific time point

within the Segment. The values for time points between two Control Points are calculated by interpolating between the two values.

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

5. 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.

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

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

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

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

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

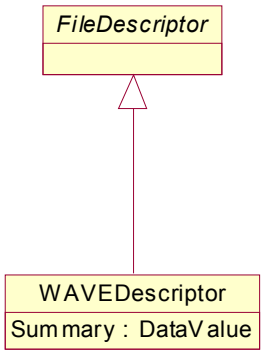
WAVEDescriptor Class

The WAVEDescriptor class specifies that a File Source Package is associated with audio content data formatted according to the RIFF Waveform Audio File Format (WAVE).

The WAVEDescriptor class is a subclass of the FileDescriptor class.

The WAVE audio format is a recommended audio format, but the WAVE format is not required for compliance with this document.

A WAVEDescriptor object shall be owned by a File Source Package.



An WAVEDescriptor object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:Summary`	`PrefT:DataValue`	A copy of the WAVE file information without the sample data. Required.

A WAVE Audio Descriptor describes a WAVE object contains digitized audio data in the little-endian byte ordering (used on the Intel architecture). It contains data formatted according to the Microsoft/IBM Multimedia Programming Interface and Data Specifications, Version 1.0, but limited to the section describing the RIFF Waveform Audio File Format audio data. The 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.

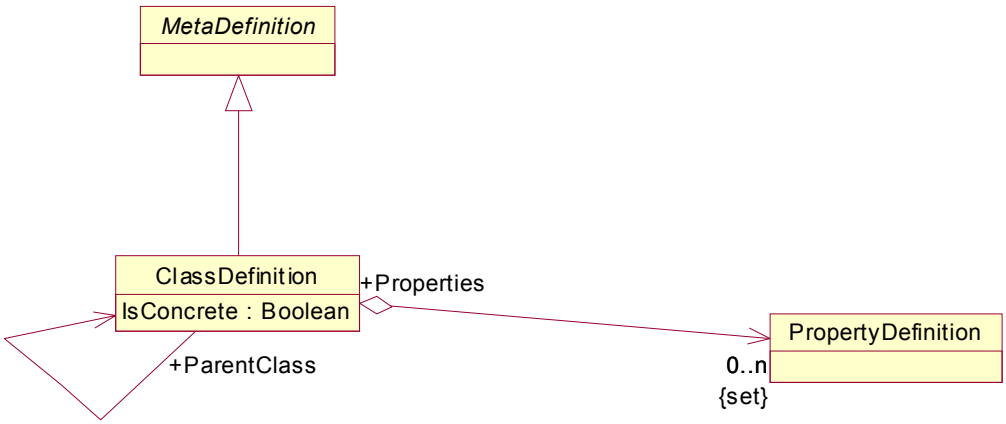
Appendix B: AAF Classes for Defining Interchange Objects

This document contains the reference descriptions of the AAF classes. The reference pages are arranged alphabetically.

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 subclass of the DefinitionObject class.



All ClassDefinition objects are owned by the Dictionary object. A ClassDefinition object shall have the required classes listed in the following table.

Property Name	Type	Explanation
`Pref:ParentClass`	`PrefT:WeakReference` to ClassDefinition	Specifies the parent of the class being defined. Required.
`Pref:Properties`	`PrefT:StrongReferenceSet` of PropertyDefinition	Specifies an ordered set of PropertyDefinition objects that define the properties for a class. Optional.
`Pref:IsConcrete`	`PrefT:Boolean`	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. Required.

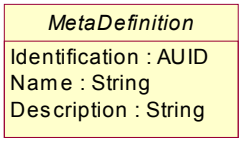
1. Any class extension must be descended from the InterchangeObject class. A Class Definition object specifying the InterchangeObject class shall have a ParentClass property with a weak reference to itself.

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; consequently an object that belongs to the MetaDefinition class shall also belong to a subclass of the MetaDefinition class.



A MetaDefinition object shall have the required properties and may have the optional properties listed in the following table.

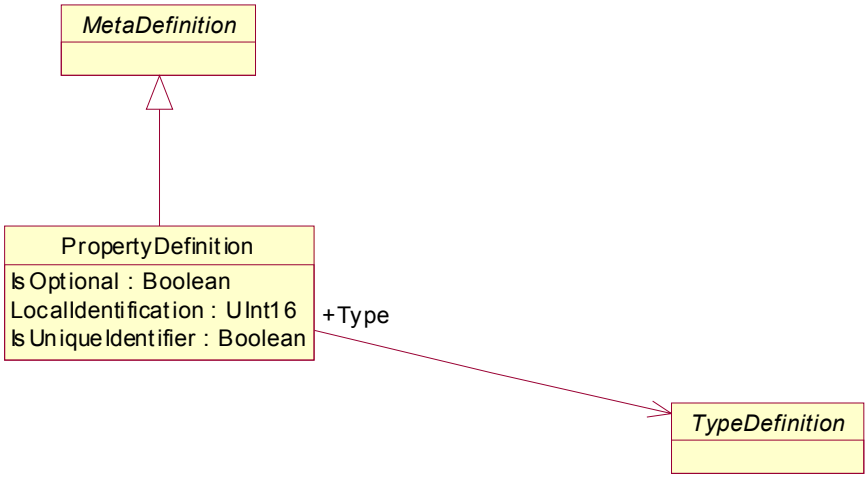
Property Name	Type	Explanation
`Pref:Identification`	`PrefT:AUID`	Specifies the unique identifier for the item being defined. Required.
`Pref:Name`	`PrefT:String`	Specifies the display name of the item being defined. Required.

Pref:Description	PrefT:String	Provides an explanation of the use of the item being defined. Optional.

PropertyDefinition Class

PropertyDefinition describes properties allowed for a class.

PropertyDefinition is a subclass of DefinitionObject. An PropertyDefinition object is a owned by a ClassDefinition object. An PropertyDefinition object has an association with a TypeDefinition object that defines the property type.



A PropertyDefinition object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
Pref:Type	PrefT:WeakReference to TypeDefinition	Specifies the property type. Required.
Pref:IsOptional	PrefT:Boolean	Specifies whether objects instances can omit a value for the property. Required.
Pref: LocalIdentification	PrefT:UInt16	Specifies a local integer identification that is used to identify the property in the AAF file. Required.
Pref: IsUniqueIdentifier	PrefT:Boolean	Specifies that this property provides a unique identification for this object. Optional.

The PropertyDefinition object specifies that a property can be used in a class. For classes defined by this specification, the ClassDefinition object can omit any properties defined in this document.

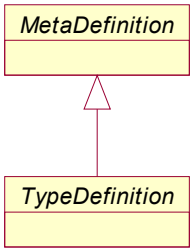
Note the LocalIdentification property is used internally within the AAF file for efficiency purposes but has no semantic meaning.

Related interfaces in the AAF Reference Implementation SDK are IAAFPropertyDef and IAAFClassDef.

TypeDefinition Class

TypeDefinition defines a property type.

TypeDefinition is a subclass of DefinitionObject. Type Definition is an abstract class. Any object in an AAF file that belongs to the TypeDefinition class shall also belong to a subclass of TypeDefinition. TypeDefinition objects are owned by the Dictionary object.

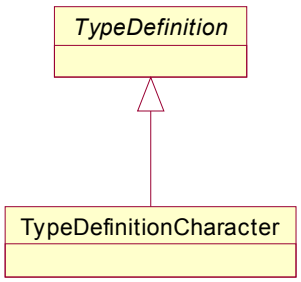


The TypeDefinition class does not define any additional properties.

TypeDefinitionCharacter Class

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

TypeDefinitionCharacter is a subclass of TypeDefinitionObject. TypeDefinitionEnumeration objects are owned by the Dictionary object.

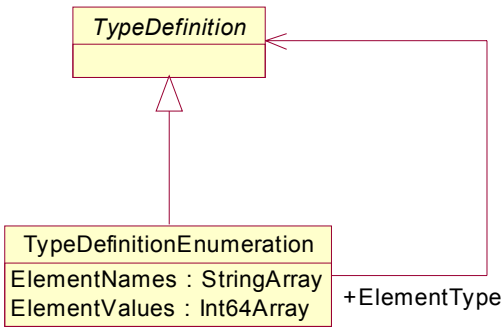


The TypeDefinitionCharacter class does not define any additional properties.

TypeDefinitionEnumeration Class

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

TypeDefinitionEnumeration is a subclass of TypeDefinitionObject. TypeDefinitionEnumeration objects are owned by the Dictionary object.

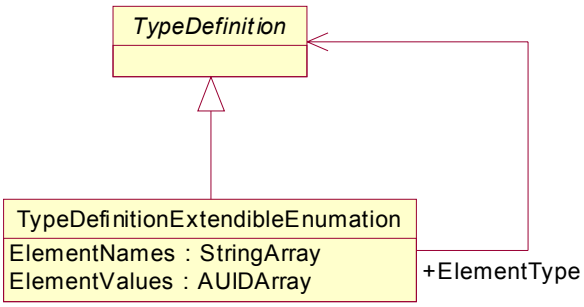


A TypeDefinitionEnumeration object shall have the required properties listed in the following table.

Property Name	Type	Explanation
`Pref:ElementType`	`PrefT:WeakReference` to TypeDefinition	Specifies the TypeDefinition that defines the underlying integer type. Required.
`Pref:ElementNames`	`PrefT:StringArray`	Specifies the names associated with each enumerated value. Required.
`Pref:ElementValues`	`PrefT:Int64Array`	Specifies the valid enumerated values. The integer values shall be positive and each value in the array shall be unique. Required.

TypeDefinitionExtendibleEnumeration

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



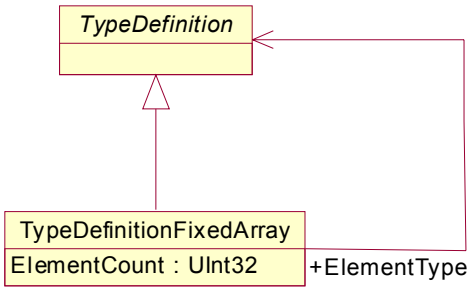
A TypeDefinitionExtendibleEnumeration object shall have the required properties listed in the following table.

Property Name	Type	Explanation
`Pref:ElementNames`	`PrefT:StringArray`	Specifies the names associated with each enumerated value. Required.
`Pref: ElementValues`	`PrefT:AUIDArray`	Specifies the known AUID values that can be used in this type. Required

TypeDefinitionFixedArray Class

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

TypeDefinitionFixedArray is a subclass of TypeDefinition. TypeDefinitionFixedArray objects are owned by the Dictionary object.



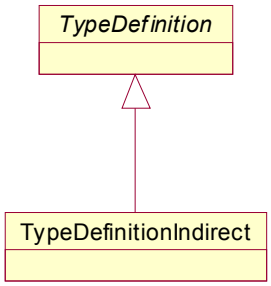
A TypeDefinitionFixedArray object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:ElementType`	`PrefT:WeakReference` to TypeDefinition	Specifies the TypeDefinition that defines the type of each element of the array. Required.
`Pref:ElementCount`	`PrefT:UInt32`	Specifies the number of elements in the array. Required.

TypeDefinitionIndirect Class

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

TypeDefinitionIndirect is a subclass of TypeDefinitionObject. TypeDefinitionIndirect objects are owned by the Dictionary object.

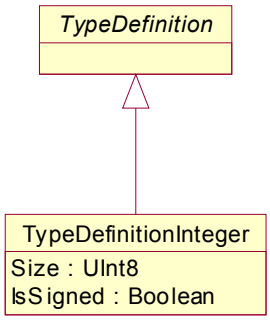


The TypeDefinitionIndirect class does not define any additional properties.

TypeDefinitionInteger Class

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

TypeDefinitionInteger is a subclass of TypeDefinition. TypeDefinitionInteger objects are owned by the Dictionary object.



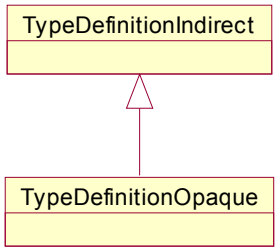
A TypeDefinitionInteger object shall have the required properties listed in the following table.

Property Name	Type	Explanation
`Pref:Size`	`PrefT:UInt8`	Specifies the number of bytes to store the integer. Legal values are 1, 2, 4, and 8. Required.
`Pref:IsSigned`	`PrefT:Boolean`	Specifies if the integer is signed (True) or unsigned (False). Required

TypeDefinitionOpaque Class

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

TypeDefinitionOpaque is a subclass of TypeDefinitionIndirect. TypeDefinitionOpaque objects are owned by the Dictionary object.

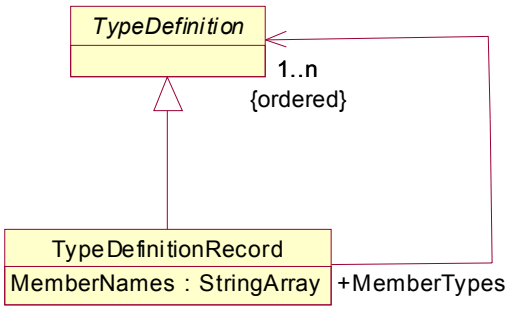


The TypeDefinitionOpaque class does not define any additional properties.

TypeDefinitionRecord Class

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

TypeDefinitionRecord is a subclass of TypeDefinition. TypeDefinitionRecord objects are owned by the Dictionary object.



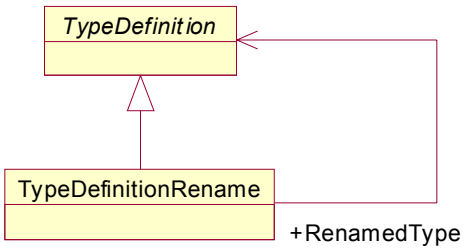
A TypeDefinitionRecord object shall have the required properties listed in the following table.

Property Name	Type	Explanation
`Pref:MemberTypes`	`PrefT: WeakReferenceVector of TypeDefinition`	Specifies the type of each element of the record. Required.
`Pref:MemberNames`	`PrefT:StringArray`	Specifies the name of each element of the record. Required.

TypeDefinitionRename Class

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

TypeDefinitionRename is a subclass of TypeDefinition. TypeDefinitionRename objects are owned by the Dictionary object.



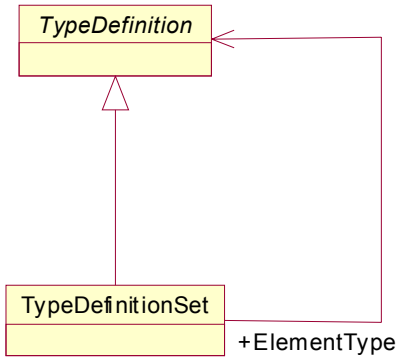
A TypeDefinitionRename object shall have the required property listed in the following table.

Property Name	Type	Explanation
`Pref:RenamedType`	`PrefT: WeakReference` to TypeDefinition	Specifies the underlying type. Required.

TypeDefinitionSet Class

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

TypeDefinitionSet is a subclass of TypeDefinition. TypeDefinitionSet objects are owned by the Dictionary object.



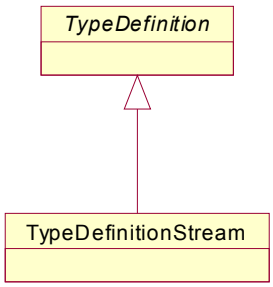
A TypeDefinition object shall have the required property listed in the following table.

Property Name	Type	Explanation
`Pref:ElementType`	`PrefT: WeakReference` to TypeDefinition	Specifies the TypeDefinition that identifies the kind of object reference. This TypeDefinition shall belong to either the TypeDefinitionStrongObjectReference or TypeDefinitionWeakObjectReference. Required.

TypeDefinitionStream Class

TypeDefinitionStream 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.

TypeDefinitionStream is a subclass of TypeDefinition. TypeDefinitionStream objects are owned by the Dictionary object.

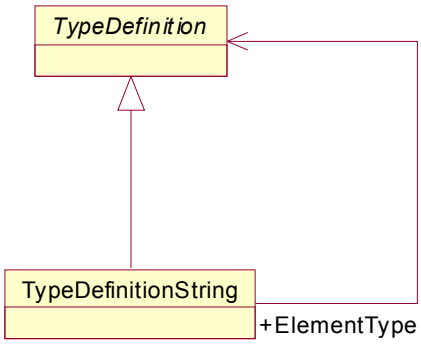


The TypeDefinitionStream class does not define any additional properties.

TypeDefinitionString Class

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

TypeDefinitionString is a subclass of TypeDefinition. TypeDefinitionString objects are owned by the Dictionary object.



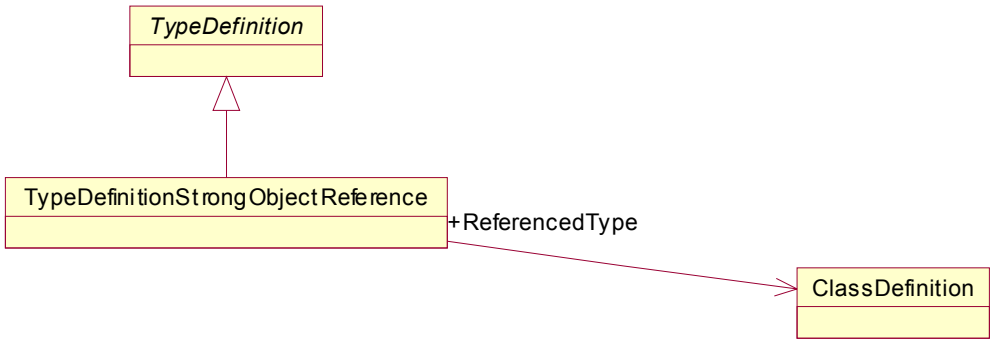
A TypeDefinitionString object shall have the required properties and may have the optional properties listed in the following table.

Property Name	Type	Explanation
`Pref:ElementType`	`PrefT: WeakReference` to TypeDefinition	Specifies the string element, which may be a character (TypeDefinitionCharacter) or integer (TypeDefinitionInteger). Required.

TypeDefinitionStrongObjectReference Class

TypeDefinitionStrongObjectReference 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.

TypeDefinitionStrongObjectReference is a subclass of TypeDefinition.
TypeDefinitionStrongObjectReference objects are owned by the Dictionary object.



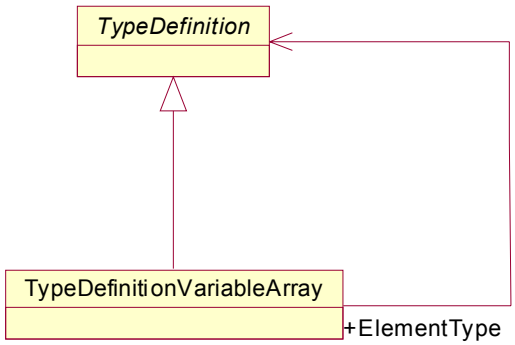
A TypeDefinitionStrongObjectReference object shall have the required property listed in the following table.

Property Name	Type	Explanation
`Pref: ReferencedType`	`PrefT: WeakReference` to ClassDefinition	Specifies the class that the referenced object shall belong to (the referenced object may also belong to a subclass of the referenced class). Required.

TypeDefinitionVariableArray Class

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

TypeDefinitionVariableArray is a subclass of TypeDefinition. TypeDefinitionVariableArray objects are owned by the Dictionary object.



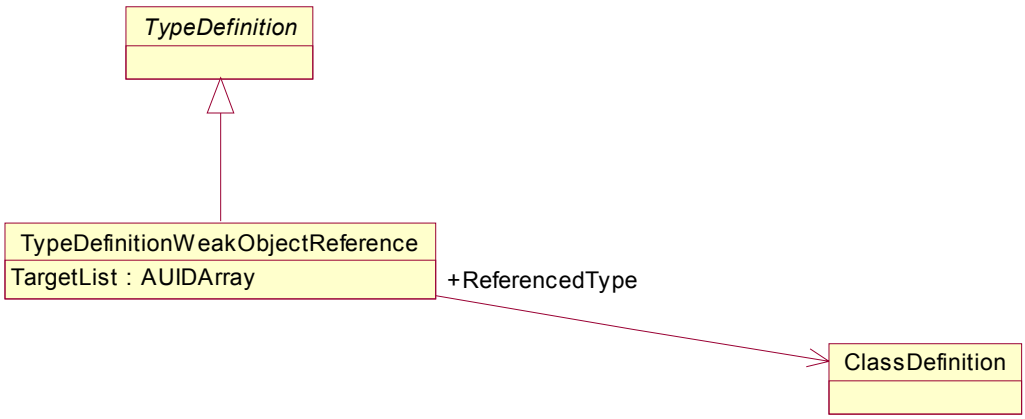
A TypeDefinition object shall have the required property listed in the following table.

Property Name	Type	Explanation
`Pref:ElementType`	`PrefT:WeakReference` to TypeDefinition	Specifies the type of the element of the array. Required.

TypeDefinitionWeakObjectReference Class

TypeDefinitionWeakObjectReference 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.

TypeDefinitionWeakObjectReference is a subclass of TypeDefinition.
TypeDefinitionWeakObjectReference objects are owned by the Dictionary object.



A TypeDefinitionWeakObjectReference object shall have the required property listed in the following table.

Property Name	Type	Explanation
`Pref: ReferencedType`	`PrefT: WeakReference` to ClassDefinition	Specifies the class that the referenced object shall belong to (the referenced object may also belong to a subclass of the referenced class). Required.
`Pref: TargetList`	`PrefT:AUIDArray`	Specifies the AUIDs that specify the properties from the root of the file to the property that has the 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. Required.

Appendix C Data types

This document defines two sets of types: the data type and the data definition. The data type specifies the type of property values and of parameters. The data definition specifies the type for objects in the Component class. Annex A lists the data type of each property in each class. Objects that belong either to the class Component have a property that identifies the data definition of the object. Data definition is used to identify the basic types of essence produced by Components.

The data type is identified by a globally unique integer. Table B-1 lists the data types identified by the name.

The data definition is identified by a globally unique integer. Table B-2 lists the data kinds identified by the name.

Table C-1 – Data Types

Data Type	Explanation
`PrefT:AUID`	128-bit unique integer identifier which is a SMPTE Universal Label conforming to SMPTE 298M-1997 or another 128-bit unique identifier
`PrefT:AUIDArray`	Array of 128-bit unique integer identifiers
`PrefT:Boolean`	Specifies either True or False.
`PrefT:Char`	Specifies a single character value.
`PrefT:ColorSitingType`	Specifies how to compute subsampled values as a 16-bit enumerated type. Values are 0 coSiting — To calculate subsampled pixels, take the preceding pixel's color value, discard the other color values, and cosite the color with the first luminance value.

Table C-1 – Data Types

Data Type	Explanation
	1 averaging To calculate subsampled pixels, take the average of the two adjacent pixel's color values, and site the color in the center of the luminance pixels. 2 threeTap To calculate subsampled pixels, take 25 percent of the previous pixel's color value, 50 percent of the first value, and 25 percent of the second value. For the first value in a row, use 75 percent of that value since there is no previous value. The threeTap value is only meaningful when the HorizontalSubsampling property has a value of 2.
`PrefT:CompCodeArray`	Specifies the order in which the RGBA components are stored as an array of character. Each element in the array represents a different color component. The array can contain the following characters: 'A' Alpha component 'B' Blue component 'F' Fill component 'G' Green component 'P' Palette code 'R' Red component '0' no component Each character except '0' can appear no more than one time in the array. The array is terminated by a 0 byte and has a maximum of 8 elements (including the terminating byte). Note that a byte with the ASCII '0' indicates no component and a byte with a 0 (ASCII NULL) terminates the string.
`PrefT: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).
`PrefT:DataValue`	Specifies essence or a block of data whose type is specified by a data kind.
`PrefT:EdgeType`	Specifies the kind of film edge code as an enumerated Int16. Values are: 0 ET_NULL Invalid edge code 1 ET_KEYCODE Eastman Kodak KEYKODE TM format. 2 ET_EDGENUM4 edge code format: nnnn+nn. 3 ET_EDGENUM5 edge code format: nnnnn+nn.
`PrefT:EditHintType`	Specifies hints to be used when editing Control Points. Values are: 0 EH_Proportional 1 EH_RelativeLeft

Table C-1 – Data Types

Data Type	Explanation
	2 EH_RelativeRight
PrefT:FadeType	Specifies the type of the audio fade; may have one of the following values: 0 No fade 1 Linear amplitude fade 2 Linear power fade 3 Linear dB fade Additional registered and private fade types may be defined.
PrefT:FilmType	Specifies the format of the film. as an Int16 enumerated value. Values are: 0 FT_NULL invalid film type 1 FT_35MM 35 millimeter film 2 FT_16MM 16 millimeter film 3 FT_8MM 8 millimeter film 4 FT_65MM 65 millimeter film
PrefT:Int8	Specifies an 8-bit 2's complement integer value.
PrefT:Int8Array	Specifies an array of Int8 values.
PrefT:Int16	Specifies a 16-bit 2's complement integer value.
PrefT:Int16Array	Specifies an array of Int16 values.
PrefT:Int32	Specifies a 32-bit 2's complement integer value.
PrefT:Int32Array	Specifies an array of Int32 values.
PrefT:Int64	Specifies a 64-bit 2's complement integer value.
PrefT:Int64Array	Specifies an array Int64 values.
PrefT:JPEGTableIDType	Specifies the JPEG tables used in compressing TIFF data.
PrefT:LayoutType	Describes whether all data for a complete sample is in one frame or is split into more than one field as an enumerated Int16. 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 SINGLE_FIELD: sample consists of two interlaced fields, but only one field is stored in the data stream 3 MIXED_FIELDS
PrefT:Length	Specifies the length of a Component with an UInt64.
PrefT:PackageID	Specifies a 32-byte unique identifier that can hold a SMPTE UMID.

Table C-1 – Data Types

Data Type	Explanation
PrefT:PixelRectangle	Specifies of Rectangle in pixels. Is Record with the following elements: Horizontal: UInt 16 Vertical: UInt16
PrefT:Position	Specifies an offset into a Component with an Int64.
PrefT:ProductVersion	Specifies the version number of an application. Consists of 5 Uint16 integer values. The first four integers specify the major, minor, tertiary, and patch version numbers. The fifth integer has the following values: 0 kVersionUnknown No additional version information 1 kVersionReleased Released product 2 kVersionDebug Development version 3 kVersionPathched Released version with patches 4 kVersionBeta Prerelease beta test version 5 kVersionPrivateBuild Version not intended for general release
PrefT:PulldownKindType	Specifies whether the Pulldown object is converting from NTSC or PAL video and whether frames are dropped or the video is played at another speed. Values are: 0 kTwoThreePD Converting between NTSC and film by dropping or adding frames 1 kPalPD Converting between PAL and film by dropping or adding frames 2 kOneToOneNTSC Converting between NTSC and film by speeding up or slowing down the frame rate. 3 kOneToOnePAL Converting between PAL and film by speeding up or slowing down the frame rate. 4 kVideoTapNTSC Converting between NTSC and film by recording original film and video sources simultaneously.
PrefT: PulldownDirectionType	Specifies whether the Pulldown object is converting from tape to film speed or from film to tape speed. Values are: 0 kVideoToFilmSpeed The InputSegment is at video speed and the Package track owning the Pulldown object is at film speed. 1 kFilmToVideoSpeed The InputSegment is at film speed and the Package track owning the Pulldown object is at video speed.
PrefT:PhaseFrameType	Specifies the phase within the repeating pulldown pattern of the first frame after the pulldown conversion. A value of 0 specifies that the

Table C-1 – Data Types

Data Type	Explanation
	Pulldown object starts at the beginning of the pulldown pattern.
`PrefT:Rational`	Specifies a rational number by means of an Int32 numerator and an Int32 denominator.
`PrefT: 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).
`PrefT: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
`PrefT: RGBAComponentKind`	Enumerated type that specifies the color or function of a component within a pixel. 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
`PrefT: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 For each component in the pixel, the following codes should be specified: 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 A Fill component indicates unused bits. After the components have

Table C-1 – Data Types

Data Type	Explanation
	been specified, the remaining Code and Size fields should be set to 0.
`PrefT:String`	Specifies a string of Unicode characters.
`PrefT:StrongRef`	Specifies an owned object, which is logically contained by the owning object..
`PrefT: StrongReferenceVector`	Specifies an ordered set of owned objects.
`PrefT: StrongReferenceSet`	Specifies an unordered set of owned uniquely identified objects.
`PrefT:TapeCaseType`	Describes the physical size of the tape; may have one of the following values: 0 3/4 inch videotape 1 VHS video tape 2 8mm videotape 3 Betacam videotape 4 Compact cassette 5 DAT cartridge 6 Professional audio tape
`PrefT:TapeFormatType`	Describes the format of the tape; may have one of the following values: 0 Betacam 1 BetacamSP 2 VHS 3 S-VHS 4 8mm 5 Hi8
`PrefT:TCSource`	Specifies the kind of timecode; may have one of the following values: 1 LTC timecode 2 VITC timecode

`PrefT: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

Table C-1 – Data Types

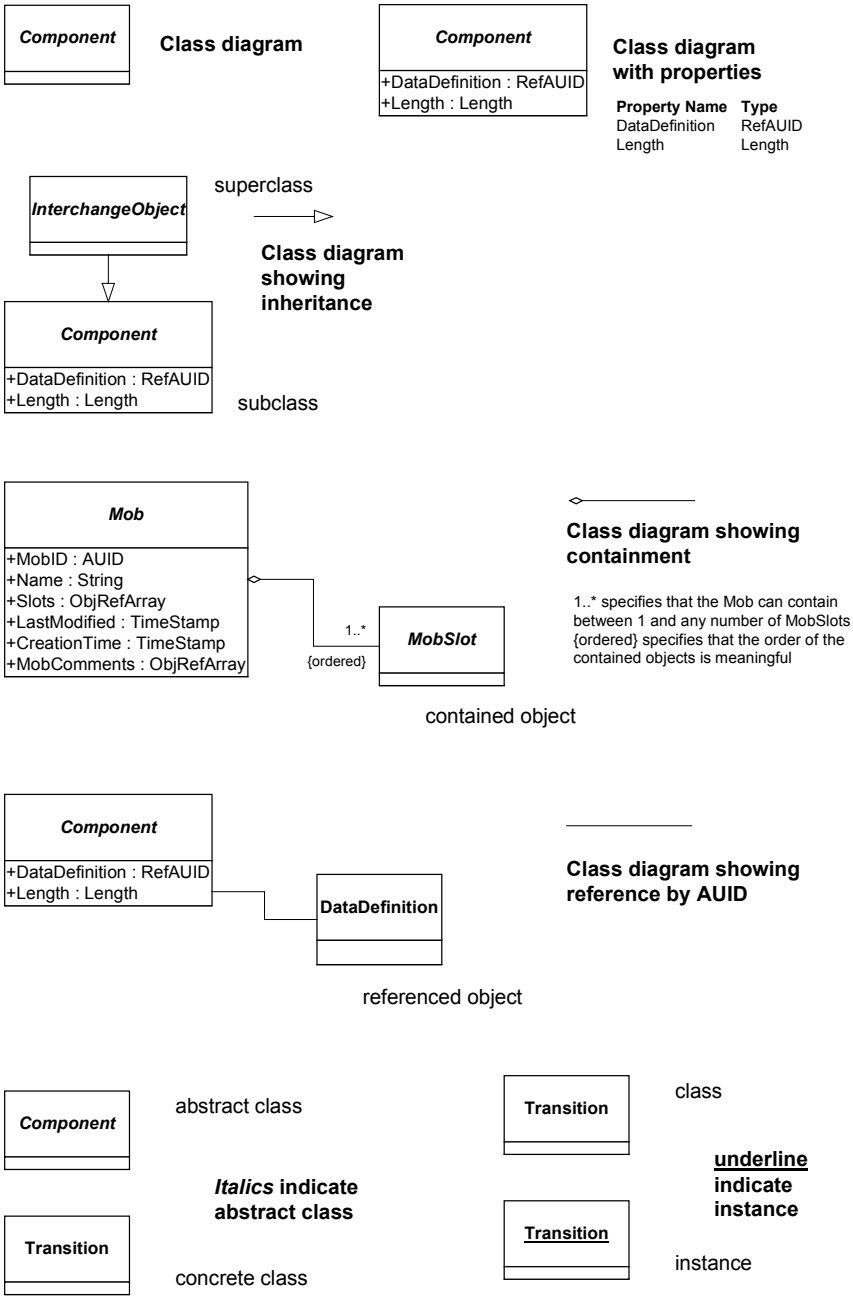
Data Type	Explanation
	Type Fraction UInt8 } } Where Fraction is expressed in 1/100 of a second.
`PrefT:UInt8`	Specifies an unsigned 8-bit integer value.
`PrefT:UInt8Array`	Specifies an array of unsigned 8-bit integer value.
`PrefT:UInt16`	Specifies an unsigned 16-bit integer value.
`PrefT:UInt16Array`	Specifies an array of unsigned 16-bit integer value.
`PrefT:UInt32`	Specifies an unsigned 32-bit integer value.
`PrefT:UInt32Array`	Specifies an array of unsigned 64-bit integer values.
`PrefT:UInt64`	Specifies an unsigned 64-bit integer value.
`PrefT:UInt64Array`	Specifies an array of 32-bit integer values.
`PrefT:VersionType`	Specifies a 2-byte unsigned version number.
`PrefT:VideoSignalType`	Specifies the type of video signal on the videotape. Values are: 0 NTSC 1 PAL 2 SECAM
`PrefT:WeakRef`	Reference to an object that defines a unique identifier
`PrefT:` `WeakReferenceVector`	Reference to an ordered set of uniquely identified objects
`PrefT:WeakReferenceSet`	Reference to an unordered set of uniquely identified objects

Table B-2 – Data Definitions

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

Appendix D Conventions

The following documentation conventions are used in the diagrams in this document.


Component
Class diagram
Component
+DataDefinition : RefAUID
+Length : Length
Class diagram
with properties
Property Name Type
DataDefinition RefAUID
Length Length
InterchangeObject
superclass
Class diagram
showing
inheritance
Component
+DataDefinition : RefAUID
+Length : Length
subclass
Mob
+MobID : AUID
+Name : String
+Slots : ObjRefArray
+LastModified : TimeStamp
+CreationTime : TimeStamp
+MobComments : ObjRefArray
1..*
{ordered}
MobSlot
Class diagram showing
containment
1..* specifies that the Mob can contain
between 1 and any number of MobSlots
{ordered} specifies that the order of the
contained objects is meaningful
contained object
Component
+DataDefinition : RefAUID
+Length : Length
DataDefinition
Class diagram showing
reference by AUID
referenced object
Component
abstract class
Italics indicate
abstract class
Transition
concrete class
Transition
class
underline
indicate
instance
Transition
instance

Appendix E: Terms and Definitions

abstract class

class that is not sufficient to define an object; an object must also belong to a subclass of the abstract class

audio

audio essence

sound in transmitted or stored in any form, including sound stored on analog tape, analog sound broadcast on radio waves, sound transmitted through air, and sound stored in a digital format on tape or disk

AUID

unique identifier which is a SMPTE Universal Label conforming to SMPTE 298M-1997 or another 128-bit unique identifier

big-endian

byte order in which 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); bytes are stored with the most-significant bit first

byte order

convention used to stored multibyte numeric values on a platform

class

category of objects, which have common properties, relationships, and semantics

class dictionary

structure in a file that defines the class hierarchy for classes not specified in this document

class hierarchy

specification to the subclass and superclass relationship among a set of classes

component

basic object that defines essence in a slot

composition Package

package that specifies association and composition metadata that describe how to combine and modify content elements and content items to produce a content package

composition metadata

metadata that describes how to combine essence in a sequence and to modify essence

content

program material and related information of any variety

data definition

structure that determines the basic kind of data produced by a component

data type

structure that determines the kind of value that can be stored in a property

derivation metadata

metadata that describes the source that provides the values of an object

descriptive metadata

metadata that provides additional information

edgecode

codes that are marked on film to facilitate the location of specific frames

edit rate

rational number that specifies the units used to specify the duration of components in a slot; the edit rate is the number of units that equal one second in clock time

edit unit

unit in which the integer length of components in a slot are specified

essence

parts of content that directly represent program material, such as audio, video, graphic, still-image, text, or other sensor data

essential metadata

metadata that describes that is required to decode the essence

file source Package

package that describes an essence component stored in a digital form in a file

Effect

segment that combines or modifies one or more input segments according to the specified effect definition and controls parameters

header

root object of the file that has the Packages and EssenceData objects in the file and defines extensions to the classes used to store objects in the file

inheritance

mechanism that defines a relationship between classes where a subclass inherits the properties, relationships, and semantics of its superclass

interleaved channels

storage format that combines two or more channels of audio data or video data into a single stream

little-endian

byte order in which the least-significant byte is stored first (at the address specified, which is the lowest address of the series of bytes that constitute the value); bytes are stored with the most-significant bit first

material Package

package that specifies association and derivation metadata; it provides a level of indirection between a composition Package and a file source Package and synchronizes file source Packages

metadata

parts of content which data that is used to describe essence or provide information on how to use the essence

package

structure that has a globally unique identity and describes essence

PackageID

value that defines the unique identification of a Package

slot

object in a package that describes essence and is externally accessible

object

collection of properties, each of which has a name, a type, and a value

ordered set

ordered collection of unique values

physical source package

package that describes physical media

property

element in a file that has a name, type, and value

property name

property type name

text name that identifies a data type

property value

data that stored in a property, which is in an object

rational number

numeric value expressed by an integer numerator and an integer denominator that specifies a numeric value that can have a fractional part

relational metadata

metadata that describes how to synchronize or interleave essence

sample rate

rational number that specify the number of samples of essence that are played in one second

segment

component that has well defined boundaries; a segment can be used without any other components in contrast to a transition, which can only be used in a sequence and need to be surrounded by segments

sequence

sequence that has an ordered set of components and causes them to be arranged in a sequential order

set

unordered collection of unique values

edit interchange file

storage wrapper data file that stores essence and metadata in objects that conforms to this document

source clip

segment that specifies essence by referencing a slot in a package

source package

package that describes an essence component stored either in a digital form or on a physical media source

static metadata

metadata that describes the edit interchange file as a whole

storage wrapper

persistent storage mechanism for the storage of complex content

NOTE This mechanism allows descriptive information to be stored with the data in such a way that it is possible to query the wrapper file to find out the format of the data and then to use that information to read and interpret the encapsulated data.

strong reference

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 logical contains the owned object.

subclass

class that is defined as having the properties, relationships and semantics as another class, which is called its superclass, and may have additional properties, relationships, and semantics that are not present in the superclass

superclass

class that has another class as its subclass

timecode

codes that are written on videotape and audiotape to facilitate the location of a point on the tape

transition

component which causes the segment that precedes it in the sequence to be overlapped in time with the segment that follows it in the sequence

variant metadata

metadata that describes an element or subsection within an edit interchange file

weak reference

relation 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.