Brad Gilmer
AAF Association

The Advanced Authoring Format (AAF) enables content creators to easily exchange digital media – essence – and metadata across platforms, and between applications. It simplifies project management, saves time and preserves valuable metadata that was often lost in the past when transferring essence between applications.

AAF is an industry-driven, cross-platform, file format that allows the interchange of data between multimedia authoring tools. AAF can be used to interchange **essence data** and **metadata**.

Essence data consists of picture, sound and other forms of data that can be directly perceived. *Metadata* is data that describes essence data, performs some operations on essence data, or provides supplementary information about the essence data. For example, digitized sound data is essence data. However, 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 programme 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 the picture and sound) ... are all represented as metadata. AAF provides a way to interchange this rich set of metadata.

AAF is being developed and promoted by the AAF Association [1].

Improving the workflows

The increasing capability of multimedia authoring tools to work in a networked environment is enabling changes to take place in production workflows. The traditional workflow – based around tape interchange,

Abbreviations

AAF	Advanced Authoring Format
API	Application Programming Interface
COM	(Microsoft) Component Object Model
CVS	Concurrent Versions System
DMM	(AAF) Data Model Manager
GPI	General Purpose Interface
IDL	Interface Definition Language
KLV	(SMPTE) Key Length Value
MPEG	Moving Picture Experts Group
MSS	Microsoft Structured Storage
MXF	(Pro-MPEG) Material eXchange Format
OM	(AAF) Object Manager
OS	Operating System
SDK	Software Development Kit
SMPTE	Society of Motion Picture and Television Engineers (USA)
UMID	(SMPTE) Unique Material Identifier

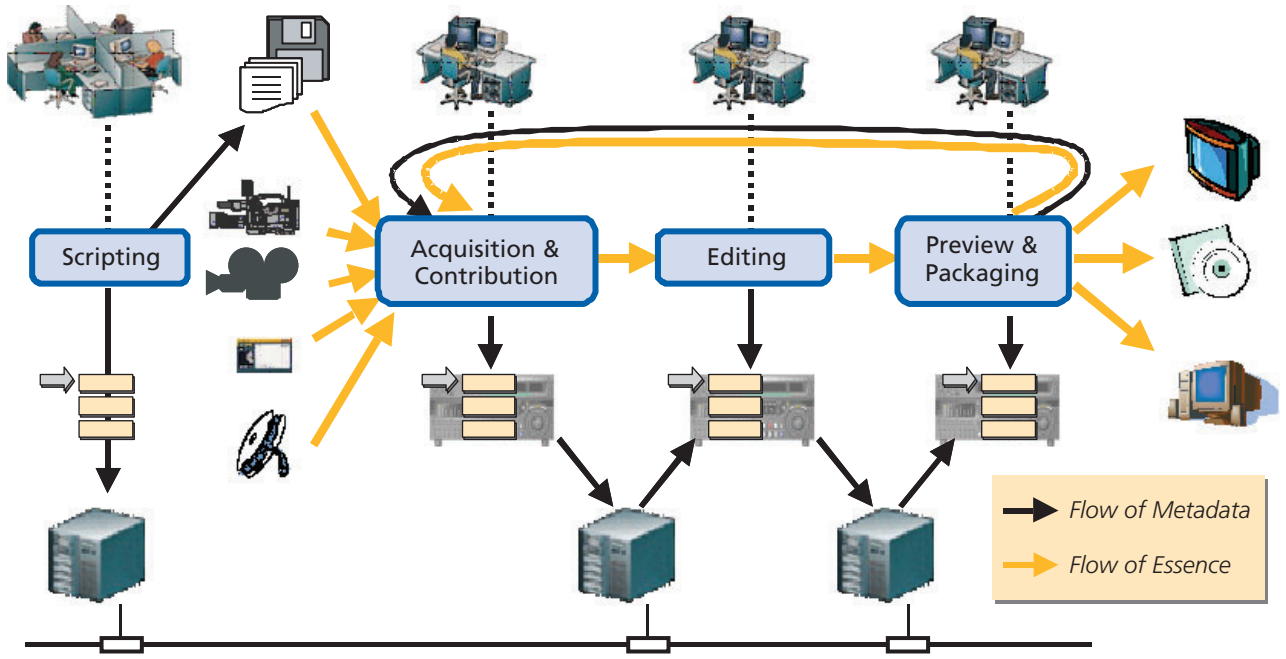


Figure 1
Content production flow

isolated non-linear editing and authoring tools, and ad-hoc metadata systems – is being recast as a more integrated networked system with a consistent approach to the format and interchange of essence and metadata.

Some of the processes in a typical content production workflow are shown in *Fig. 1*. Even in this apparently simple example, some complex requirements arise:

- pre-production metadata is required during acquisition;
- the flow of essence and metadata from acquisition devices into multiple editing and authoring tools, and possibly preview and packaging, are required;
- the metadata must track the essence as it is copied through a succession of physical and file-based media;
- packaged content may be further re-used;
- different versions of the content are required for different types of distribution.

To enable this kind of workflow, a systematic and open approach is required to the organization and interchange of essence and metadata. The Advanced Authoring Format is one such solution, with particular strengths in the film and television post-production industries.

AAF specifications and software

The major parts of AAF are:

- The AAF Object Specification;
- The AAF Low-Level Container Specification;
- The AAF Software Development Kit (SDK) Reference Implementation.

The **AAF Object Specification** defines a structured container for storing essence data and metadata, using an object-oriented model. The AAF Object Specification defines the logical contents of the objects and the rules for how the objects relate to each other.

The **AAF Low-Level Container Specification** describes how each object is stored on disk. It uses Structured Storage, a file storage system developed by Microsoft, to store the objects on disk.

The **AAF SDK Reference Implementation** [2] is an object-oriented programming toolkit and documentation which allow client applications to access the data stored in an AAF file. The AAF SDK Reference Implementation is a platform-independent toolkit provided in source form. The AAF SDK has been built and tested on several reference platforms by the AAF Association. The reference platforms are currently Windows 2000, MacOS, Irix and Linux.

The AAF SDK is held on SourceForge.net [2], a large Open Source development website. The SDK can be freely downloaded from there, using a web browser or a CVS tool.

AAF object model

Advantages of object-oriented interchange

AAF uses an object-oriented mechanism to structure the metadata and essence. Object-oriented interchange has the following advantages:

- Objects provide a framework for containing and labelling the different kinds of information.
- Objects make it possible to treat different items in the same way for attributes that they share. For example, 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. Similarly, 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.

AAF object model capabilities

The AAF object model has the following capabilities:

- Provides a mechanism to encapsulate essence and metadata. The object model 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 in order to process the essence.
- 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.
- Provides a mechanism to describe the derivation of essence from the original media sources. For example, 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.
- 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 on the essence.
- 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.

Fundamental AAF objects

A **Package** is an object that has a universal (globally unique) identifier and consists of metadata. Packages describe composition, essence or physical media. Packages have names and descriptions, but are primarily identified by a unique identifier, which is called a PackageID (may also be a basic SMPTE UMID). *Table 1* list four kinds of Package that are commonly used in the AAF object model.

Table 1: Different kinds of AAF Package

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 the 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.

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

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. *Table 2* lists three kinds of Slot commonly used in the AAF object model.

Table 2: Different kinds of AAF Slot

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 annotation associated with specific times.

A Slot has a **Segment** describing an essence element. The Segment subclasses include the following:

- **SourceClip** which references a section of a Slot in another Package; for example a SourceClip in a TimelineSlot can describe video data.
- **Sequence** which specifies that its set components are arranged in a sequential order; in a TimelineSlot, the components are arranged in sequential time order.

- **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.
- **Filler** which defines an unspecified value for its duration.

Some other common AAF classes are summarized in *Table 3*.

Table 3: Other common AAF classes

AAF class	Function
Header	Provides file-wide information and contains the Identification(s), Dictionary and ContentStorage. There is one Header per file.
Identification	Provides information about the application(s) that created or modified the file.
Dictionary	Contains DefinitionObjects, that is definitions of the Classes, Types, Effects and Parameters used in the file.
PluginDefinition	Identifies code objects that provide an implementation for a DefinitionObject, e.g. a codec providing an implementation for a CodecDefinition.
ContentStorage	Contains the Packages and EssenceData objects in the file. There is one ContentStorage object per file.
EssenceData	Contains essence associated with a Package.
EssenceDescriptor	Describes the format of essence associated with a File Source Package or media associated with a Physical Source Package.
Locator	Provides information to help find a file that contains the essence.
TaggedValue	Specifies a user-defined tag, key and value.
KLVData	Specifies user data with a Key (SMPTE label), Length and Value.
Transition	Specifies that the two adjacent Segments should be overlapped when they are played and the overlapped sections should be combined using the specified Effect.
Parameter	Specifies a control argument for an effect.
ControlPoint	Specifies a value and a time point and is used to specify an effect control point.

Extending AAF

AAF defines a base set of built-in classes. These built-in classes can be used to interchange a broad range of data between applications, but applications may have additional forms of data that cannot be described by the basic set of built-in classes.

To provide for this, AAF is designed to allow extensions. Its files can include extensions that define new effects, new kinds of metadata and new kinds of essence data. Typically, new features appear in one application and gradually become common to many. Consequently, new features are first defined as private extensions to the AAF specification and may later progress to be included in a revised AAF specification and be directly supported by the AAF SDK Reference Implementation.

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.

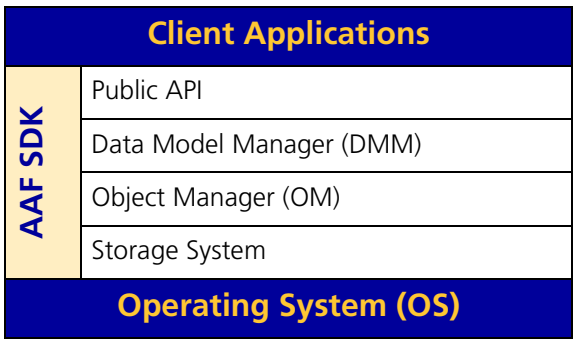
- To store internal application-specific information so that an application can use this interchange format as a native file format.
- To define new essence formats for use by plug-in codecs

The extra stored information can vary from a single private property to a complex structure of private objects. Extensions may define new effects, classes, properties, property types, essence types and plug-in code.

AAF SDK software architecture

The AAF SDK has a layered design, consisting of a public API, a reference implementation of the AAF object model (known as the data model manager), an object manager and a storage system *(see Table. 4)*. This design allows the possibility of alternative public APIs or storage systems in the future.

Table 4: Layered design of the AAF SDK



The role of each layer will now be briefly described.

Public Application Programming Interface (API)

The public API is the aspect of the Data Model Manager that all client applications see, and that treats all potential clients equally. It is written in Interface Definition Language (IDL), to permit bindings to different languages (e.g. C, C++) and object brokers (e.g. Component Object Model).

It provides basic services: persistence (save and restore), transaction (add, modify, delete), accessories (get, set), and navigation (traversal, iteration, query). It has a regular predictable structure, to encourage a consistent coding style and allow extension over time. It provides clear mechanisms for extension of the Data Model, so that new object types can be linked into the API without causing revision or recompilation of the kernel software.

Microsoft's Component Object Model (COM) is currently employed by the SDK as a programming interface for client applications. The SDK includes a minimal implementation of COM for use on non-Microsoft platforms.

Data Model Manager

Beneath the public API there are various interfaces and implementation helper functions that are not expected to be directly called by the client. It is here that much of the design value of the Data Manager is concentrated. One of the benefits of using IDL to define the public API is that the unpublished implementation details are defined separately, reducing the temptation for clients to use an internal function and risk less than full error checking.

Object Manager

The Object Manager (OM) provides the basic functions of Saving and Restoring objects and sub-objects and maintaining the relationships between them. The architecture separates Data Model Management from generic Object Management. The interface between these two subsystems is not public; the DMM interface exposes the OM interface polymorphically through the DMM API.

Storage System.

The Storage System underlying the Object Manager is normally one of the file systems provided by the OS. In the AAF SDK, this function is provided by Microsoft Structured Storage (MSS). MSS refers to a data storage architecture that uses a "file system within a file" architecture. This container format is a public domain format, allowing interested parties to add future developments or enhancements in a due process environment. Microsoft has specifically upgraded the core technology compound file format on all platforms (Windows, MacOS, UNIX variants) to address the needs of AAF.

Operating System

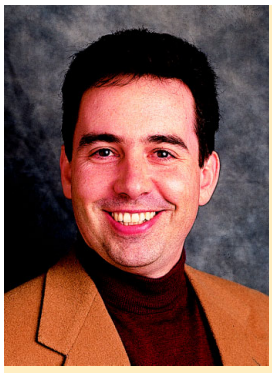
Underlying all the other subsystems is the Operating System. One of the challenges in designing the DMM and OM was to keep them separable from the Operating System, in order to serve the cross-platform interoperability requirements of the clients.

AAF and MXF

The capabilities of AAF come at the price of complexity within the AAF SDK reference implementation. Whilst this may be of little consequence within a software application running on a PC-class device, it can have a significant impact within embedded systems such as VTRs or cameras where processing and memory resources may be scarce. The chosen solution to this is a second related format – known as the Material eXchange Format (MXF) [3] – which is being developed jointly by the Professional MPEG Forum [4] and the AAF Association [1].

MXF reuses a subset of the AAF object model. The parts dealing with material (rushes or rendered finished programmes) are carried over into MXF while the parts dealing with compositions, effects and the in-file dictionary are removed.

MXF is streamable. By using SMPTE 336M KLV coding, instead of Structured Storage, and applying other rules on placement of data within the stream, MXF provides capabilities such as playing while recording, and operation with isolated sections of streams. By replacing Structured Storage however, the AAF feature of in-place editing of existing files is lost.



Brad Gilmer has extensive experience in broadcast operations and production. He is the Executive Director of the Advanced Authoring Format Association, a trade association charged with fostering interoperability in the rich media authoring environment. He serves as Executive Director of the Video Services Forum, an organization representing video transport service providers, manufacturers and users. He is also President of Gilmer & Associates, Inc, a management and technology consulting company. His company specializes in the strategic deployment of technology to meet the needs of the broadcast industry.

Mr Gilmer is active within the Society of Motion Picture and Television Engineers, chairing several committees and working with the SMPTE Registration Authority. He is also co-chair of the File Interchange group within the Pro-MPEG Forum. He writes a monthly column on computers and networks for Broadcast Engineering Magazine and World BE.

Brad Gilmer was formerly the Head of Engineering and Operations for Turner Broadcasting Systems, Inc.

It is appropriate that two storage mechanisms exist; they are each tailored to the requirements of the environment in which they will work. The important aspect is that they both use the same metadata object model, which allows direct mapping of data between AAF and MXF files. An on-going development of the AAF SDK Reference Implementation is the ability to access MXF files through its existing APIs. This will enable authoring systems to access MXF and AAF in combination. The Professional MPEG Forum and the AAF Association envisage authoring tools that will allow the interchange of AAF files – which hold composition metadata for referencing and accessing the source material stored in MXF or AAF. Rendered finished programmes for play-out or archive may be stored as MXF, while the AAF version can also support re-versioning.

AAF Association

Incorporated in January 2000, the AAF Association is a broadly-based trade association whose membership includes many major players in the film, television and post-production industries. For more information, please see [1] or contact info@aafassociation.org.

Acknowledgements

The author would like to acknowledge the contributions of AAF Association members to this article.

Bibliography

[1] The AAF Association: http://www.aafassociation.org.

[2] The AAF Software Development Kit: http://aaf.sourceforge.net.

[3] Bruce Devlin: MXF – the Material eXchange Format
EBU Technical Review No. 291, July 2002.

[4] The Pro-MPEG Forum: http://www.pro-mpeg.org/.