

# XML BASED DICTIONARIES FOR MXF/AAF APPLICATIONS

D. Beenham, P. Schmidt and G. Sylvester-Bradley

Sony Broadcast & Professional Research Laboratories, UK

## ABSTRACT

Both the Advanced Authoring Format (AAF) and the proposed Material eXchange Format (MXF) standard define a large set of highly structured metadata. This paper discusses a representation of the MXF/AAF metadata models using a set of XML Schema based dictionaries. MXF/AAF implementers can use these schemas for three purposes: a.) to standardise metadata handling in a human readable format, b.) to assist MXF standard maintenance and development, and c.) to automate the creation and maintenance of MXF/AAF encoding/decoding software. Examples of successful implementations of each application are given.

## INTRODUCTION

The Advanced Authoring Format (AAF) (1) is an open binary file format optimised for the post-production environment. It has a rich metadata model capable of describing video, audio, data and instructions for rendering this material into a finished programme. AAF applications are able to support extensions to the common metadata model because each AAF file carries the metadata model definitions as well as the metadata and essence.

The Material eXchange Format (MXF) (2) is a binary stream format targeted at the interchange of finished audio-visual material, data and metadata, primarily between servers, archives and content creation devices. It shares a common metadata model with AAF. MXF essence structure is described using a sub-set of AAF metadata.

The MXF effort has led to the development of a default extension to the common AAF metadata model, which adds comprehensive editorial Descriptive Metadata (DM). This default extension is known as the Geneva Scheme.

Many manufacturers and other organisations in the broadcast industry are developing implementations of MXF. The core of any implementation is an encoder/decoder (codec) that is able to encode metadata and audio-visual essence from separate sources into an MXF file and similarly decode an MXF file into its constituent essence and metadata.

MXF provides an envelope for existing audio-visual essence standards, but there is no common standalone encoding of the metadata model outside an MXF (or AAF) file. Several implementers have chosen to develop human-readable representations of the metadata model using Extensible Mark-up Language (XML) (3). Agreement on a common XML format is necessary for metadata interoperability between MXF systems and between other devices such as content archive systems and search engines.

## A Common XML Metadata Format

Key requirements have been identified for a common XML format:

- It should reflect the metadata model directly and not model the existing binary encodings.
- It should use the inherent capabilities of XML for structured content, follow current best practice and take advantage of latest technologies – yet allow for lightweight processing.
- It should provide the means to validate the metadata represented by an XML document against the metadata model, in order to simplify subsequent processing. The mechanism should be modular to allow autonomous specification of the valid XML format for metadata extensions such as the Geneva Scheme.

Two other goals for this work have been identified, as follows.

### **Standards Process Support**

The MXF specification documents capture complex interrelated data structures. The specification has several internal validity constraints to maintain, including consistent use of common definitions and uniqueness of the binary identifiers used to encode MXF in its 'native' format. Automated support should therefore assist maintenance of these constraints.

Developers have constructed alternative representations of the specifications to aid their implementation efforts. Automated support should also be applied to this task.

### **Harmonisation of Existing MXF Codecs**

Throughout the MXF standardisation process, implementers have been constructing their codecs by reference to the MXF specification documents. This has resulted in significant manual effort being expended on tracking changes to the specifications, particularly to ensure interoperability after each revision. Accelerated harmonisation of MXF codec implementations, in step with the standardisation process, should be an important priority.

### **XML SCHEMA BASED DICTIONARIES FOR MXF/AAF**

A recent XML technology, XML Schema (4), enables validation of metadata represented in XML against most of the constraints in the MXF/AAF specifications. It is also well matched to the additional requirements, which other schema technologies such as Document Type Definition (DTD) defined in (3) and Schematron (5) are not. In brief:

- The ability to create modular schemas provides an open mechanism for the definition of additional metadata. XML namespaces (6) can be used to avoid conflicts between schemas created by different organisations.
- XML Schema has a mechanism to add structured human and machine targeted annotations, using the `documentation` and `appinfo` elements (4). This feature can be used to record the equivalent MXF/AAF 'native' binary encodings and additional information for display to users and developers.
- The key fact that XML Schema is itself written in XML syntax exposes it to other XML technologies, such as Extensible Stylesheet Language Transformations (XSLT) (7). It is possible to apply this technology to re-use modular schemas and their annotations to achieve all stated goals.

A set of modular schemas reflecting the MXF/AAF metadata model has been created. The remainder of this paper discusses the details and use of these 'golden' files, for a number of different applications.

## DATA BINDING BETWEEN MXF/AAF AND XML

The main advantage of using XML Schema is its ability to verify and validate instance XML documents. This feature has been utilised to create a common XML instance document format, which contains rich metadata as defined by MXF and AAF.

This section explains the encoding of metadata in MXF and how this has been mapped into XML instance documents. The mapping for AAF is similar.

### MXF Data Structure

All MXF metadata are defined and encoded as Key-Length-Value (KLV) entities (8). Each key, or Universal Label (UL), is a binary identifier for a specific metadata item. The keys are defined in a global registry maintained by the Society of Motion Picture and Television Engineers (SMPTE). These registry entries are unique. The length is the number of bytes necessary to represent the value. The value is the actual content of a given metadata item.

MXF organises the large number of metadata items by defining data groups of KLV items (2). These data groups are also encoded in KLV. The value of the group is the defined set or ordered list of individual KLV items. The main structure of a KLV group is illustrated in Figure 1.

KLV groups are related to each other. Relationships between groups are achieved by assigning unique identifiers (an Instance UID) to each KLV group. A KLV item refers to another group by including the Instance UID in its value.

This feature is used to build two types of relationships in MXF:

- Aggregation: Each KLV group must be referenced from one other KLV group, creating a tree-like ownership hierarchy. These relationships are called “strong references”.
- Cross-referencing: Some KLV groups may additionally be referenced from elsewhere in the tree. These relationships are called “weak references”.

### XML Data Binding

A consistent mapping between KLV encoding and XML concepts has been developed.

The keys of KLV groups and items are matched to a human readable name in the MXF technical documents. This name is taken as the XML element name.

The item data types supported in MXF are defined in the standard. Most have an equivalent string representation. For those not covered by a built-in XML Schema type, user-derived types have been defined in a separate XML Schema document.

Table 1 summarises the principal features of the mapping. The examples section contains a sample of the XML format with an excerpt from the XML Schema used to validate it (Example 1).

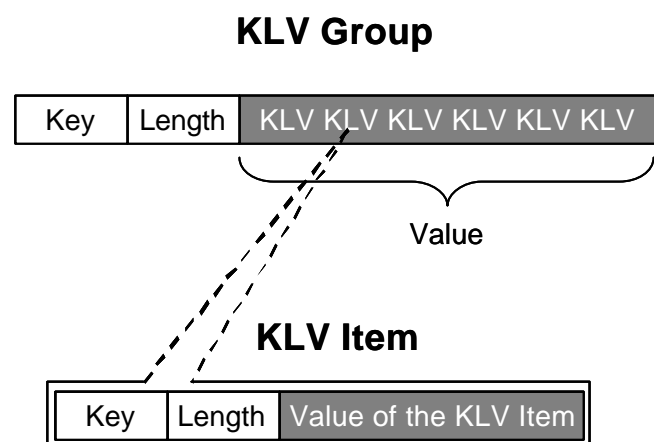


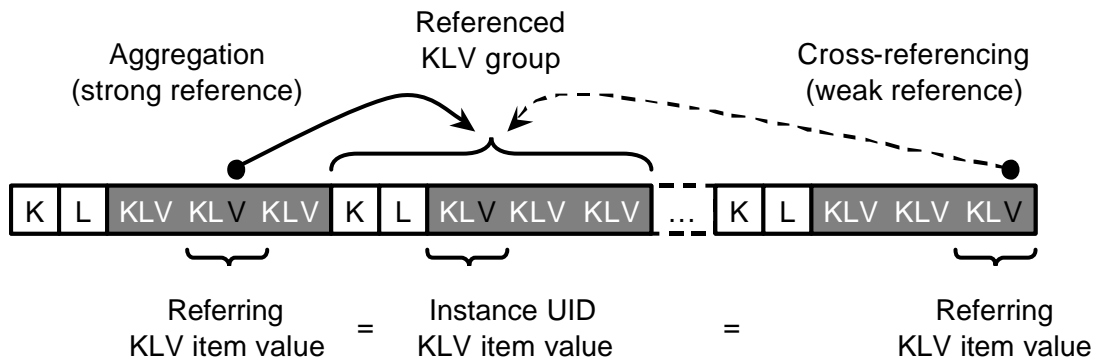
Figure 1 – Structure of KLV groups. The value part of a group is a number of KLV items.

KLV encoding	XML encoding
KLV group/item key	Element name and namespace
Binary representation of item value	String representation using XML Schema built-in types or user-derived types
Strong reference	Element containment
Weak reference	Referencing with ID and IDREF attributes

Table 1 – The mapping between KLV and XML encoding.

Figure 2 demonstrates how references are implemented in KLV and their mapping to XML. In KLV, both types of reference use the Instance UID mechanism. In XML, aggregation is represented by element containment and cross-referencing uses the ID and IDREF types defined in (3).

### KLV aggregation and cross-referencing



### XML aggregation and cross-referencing

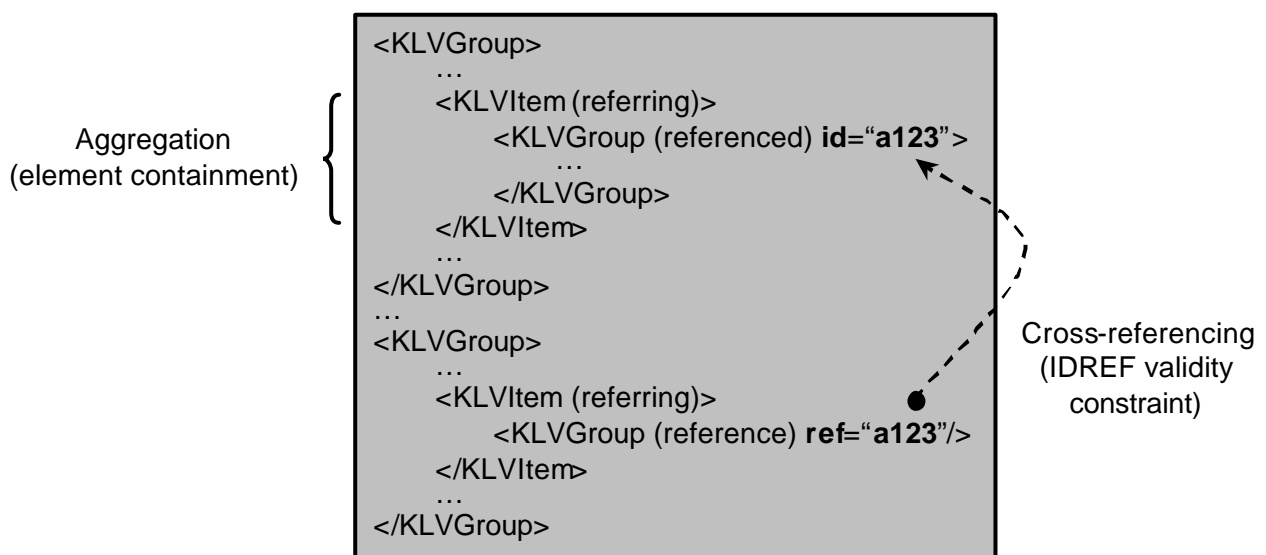


Figure 2 – Aggregation and cross-referencing in KLV and XML.

## Summary of Fundamental XML Usage

Figure 3 provides a summary of the basic interactions for which the XML format has been utilised. The next section discusses automation of the identified development processes.

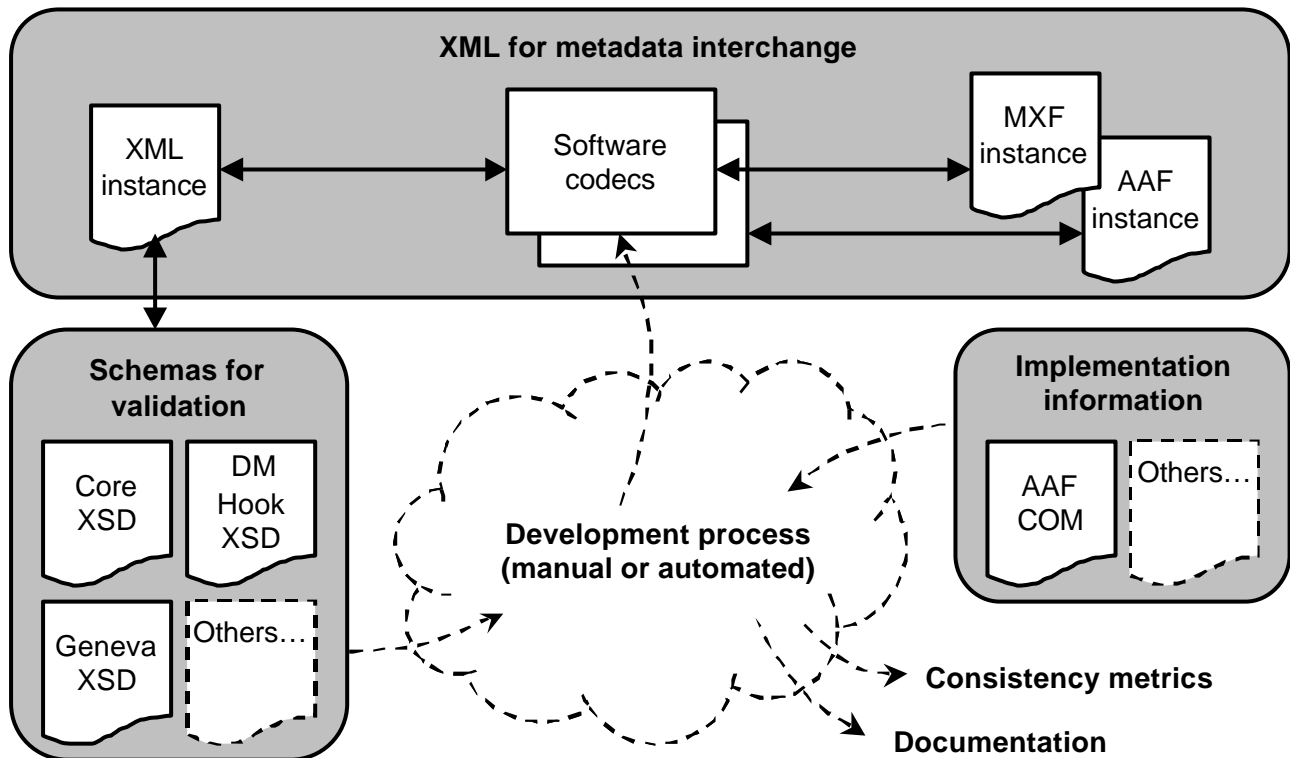


Figure 3 – Interactions between XML metadata, schemas and software codecs.

## APPLICATIONS

### MXF Standard Maintenance and Development

The primary application of the XML Schema documents is to validate XML instance documents. However, using XSLT, the schemas have been employed in other roles. The first of these has been to publish web-browser-based editions of the MXF/AAF metadata models (Figure 4). This has efficiently exposed changes to the semantics of the MXF metadata model as they have been introduced by revisions to the specification documents.

The HTML pages and links reflect the structure and relationships described by the schemas.

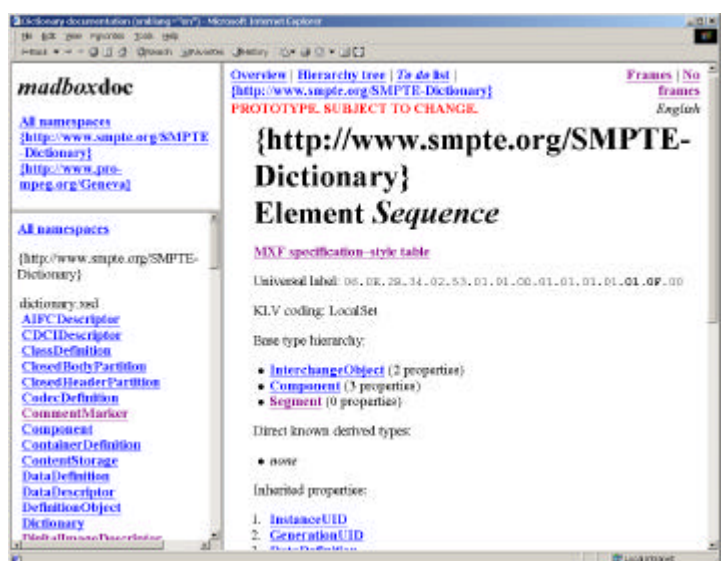


Figure 4 – XSLT transformation to HTML.

XSLT templates copy important information such as the UL from schema annotations (for which, see Example 1) using XML Path Language (XPath) (9) notation:

```
xs:annotation/xs:appinfo/klv:group/@ul
```

Human targeted annotations in the schemas specify their language with the `xml:lang` special attribute (3) so that subsequent localisation of these output files is possible with XSLT using the `lang` function defined for this purpose in the XPath recommendation.

The second important use of XSLT has been to represent programmatically the MXF specification authors' intended validity constraints. After modification to the schemas, XSLT has been used to check that these constraints have not been violated. For example, the core of the template representing the requirement for unique ULs is an XSLT key (7):

```
<xsl:key name="group-ul" match="/xs:schema/xs:complexType"
use="xs:annotation/xs:appinfo/klv:group/@ul" />
```

This has been the basis for generating pass/fail metrics and informative diagnostic output.

## Mapping XML Schemas To High-Level Programming Languages

The common set of XML Schemas and XSLT stylesheet transforms have been applied to generate class hierarchies representing the MXF metadata model in both C++ and Java.

### C++ XML transform implementation.

The C++ coding effort has been directed towards the introduction of MXF descriptive metadata into the AAF Software Development Kit (SDK). This MXF metadata divides into a number of base 'hook' metadata items and abstractions necessary for all DM schemes and the schemes themselves that define larger numbers of concrete metadata models (the only currently defined scheme being Geneva).

The logical divisions between the DM hook and Geneva Scheme metadata models have been captured in the modularisation of the XML Schemas. It has thus proved to be a relatively simple task to maintain the low coupling between the various modules during source code generation, and reflect this in the creation of independent libraries that can be dynamically loaded as needed by an AAF application.

The XSLT transform can be directed to produce one or other of the libraries simply by specifying a different input schema. Example 2 shows an excerpt from the generated C++. Another XSLT transform has been developed to provide unit tests to exercise the code generated for each metadata class and its linkage to underlying hand-written code.

### Java XML transform implementation.

A reference MXF SDK implementation has been produced in Java. A large part of its code base has also been generated by XSLT transformation of the XML Schemas.

In the course of the XSLT transformation process, a Java source file is generated for each XML element (KLV group) defined by a schema module. The class details are generated from the XML Schema documents using the techniques previously described. The Java XSLT stylesheets map XML Schema data types into native Java types. Java classes generated from different XML Schema modules are created in different Java packages (see Java documentation (10) on Java packaging). This approach has been taken to assist implementation of future MXF specification extensions and modifications, in particular to facilitate the addition of future descriptive metadata schemes.

The resulting Java SDK supports a three way data binding between in-memory Java objects and persistent storage as both XML and KLV.

## CONCLUSIONS

A method of relating MXF/AAF metadata models to an XML element structure has been developed using XML Schema. This and the use of the annotation mechanism have formed the basis for a range of applications: validation of XML instance documents, creation of MXF/AAF documentation, monitoring of changes to the MXF/AAF specifications and generation of C++ and Java source code.

The use of a common set of XML Schema dictionaries has proved very efficient for both MXF/AAF standardisation and software development. The technique has proved especially valuable in a period during which the MXF specification has undergone significant changes. It has facilitated the rapid and accurate tracking of these changes to the standard into applications and significantly reduced the potential for isolated errors and inconsistencies. In view of on-going changes and extensions to the MXF specification, the modularity of the XML Schema design has been recognised as an essential feature.

Although the many benefits of this XML to MXF/AAF mapping have been demonstrated, it is important to discuss the limitations. For instance, the modelling of file structure metadata (e.g. audio/video indexing) depends strongly on the particular codec implementation and therefore a common XML format is not appropriate.

However, most metadata definitions in MXF/AAF are accessible to an XML data model. As XML is widely used for handling MXF/AAF metadata by developers from different organisations, the XML Schema dictionaries may be used as a first step towards harmonisation of XML formats. Agreement on a common format of an XML model for MXF/AAF will benefit all developers. Discussions on this issue have already taken place with several other interested organisations.

## REFERENCES

1. See AAF specification and supporting documentation at <http://www.aafassociation.org/>
2. See MXF information at <http://www.pro-mpeg.org/>
3. Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., eds., World Wide Web Consortium (W3C), 2000. Extensible Mark-up Language (XML) 1.0 (Second Edition). 6 October 2000. See <http://www.w3.org/TR/REC-xml>
4. Thompson, H. S., Beech, D., Maloney, M., Mendelsohn, N., eds., W3C, 2001. XML Schema Part 1: Structures. 2 May 2001. See <http://www.w3.org/TR/xmlschema-1>
5. Jelliffe, R., 2001. Schematron: An XML Structure Validation Language using Patterns in Trees. See <http://www.ascc.net/xml/schematron>
6. Bray, T., Hollander, D., Layman, A., eds., W3C, 1999. Namespaces in XML. 14 January 1999. See <http://www.w3.org/TR/REC-xml-names>
7. Clark, J., ed., W3C, 1999. Extensible Stylesheet Language Transformations (XSLT) Version 1.0. 16 November 1999. See <http://www.w3.org/TR/xslt>
8. The Society of Motion Picture and Television Engineers (SMPTE), 2001. SMPTE 336M-2001: Data Encoding Protocol using Key-Length-Value (KLV).
9. Clark, J., DeRose, S., eds., W3C, 1999. XML Path Language (XPath) Version 1.0. 16 November 1999. See <http://www.w3.org/TR/xpath>

10. Gosling, J., Joy, B., Steele, G., 2000. Java Language Specification (Second Edition). See <http://java.sun.com/>

## ACKNOWLEDGEMENTS

The authors would like to thank their colleagues at Sony Broadcast & Professional Research Laboratories, Sony Broadband Solutions Network Company and members of the Pro-MPEG Forum and AAF Association for their contributions and support of this work.

## EXAMPLES

### XML Schema excerpt

```
<xs:element name="Sequence" type="sequence" substitutionGroup="Segment"/>
<xs:complexType name="sequence">
  <xs:annotation>
    <xs:appinfo>
      <klv:group ul="06.0E.2B.34.02.53.01.01.0D.01.01.01.01.0F.00" coding="LocalSet"/>
    </xs:appinfo>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="segment">
      <xs:all>
        <xs:element name="Components">
          <xs:annotation>
            <xs:documentation>List of Components contained in the Sequence</xs:documentation>
          <xs:appinfo>
            <klv:item localTag="10.01" ul="06.0E.2B.34.01.01.01.02.06.01.01.04.06.09.00.00"/>
          </xs:appinfo>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="Component" minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:all>
  </xs:extension>
</xs:complexContent>
</xs:complexType>
```

### XML instance excerpt

```
<Sequence>
  <DataDefinition><DataDefinition ref="ddef-picture"/></DataDefinition>
  <Components>
    <SourceClip>
      <DataDefinition><DataDefinition ref="ddef-picture"/></DataDefinition>
      <SourceTrackID>0</SourceTrackID>
      <StartPosition>0</StartPosition>
    </SourceClip>
    ...
  </Components>
</Sequence>
```

Example 1 – XML instance sample and part of the XML Schema used to validate it.

File: GenevaIDs.h

```
// Stored Object IDs (Set Universal Labels in MXF)
// One AUID for each class defined in this module.
const aafUID_t AUID_AAFGeneva_GenevaFramework =
{ 0x0D020101, 0x017F, 0x0100, { 0x06, 0x0E, 0x2B, 0x34, 0x02 } };
const aafUID_t AUID_AAFGeneva_ProductionClipFramework =
{ 0x0D020101, 0x017F, 0x0200, { 0x06, 0x0E, 0x2B, 0x34, 0x02 } };
const aafUID_t AUID_AAFGeneva_ProductionFramework =
{ 0x0D020101, 0x0101, 0x0100, { 0x06, 0x0E, 0x2B, 0x34, 0x02 } };
const aafUID_t AUID_AAFGeneva_SceneFramework =
{ 0x0D020101, 0x0101, 0x0200, { 0x06, 0x0E, 0x2B, 0x34, 0x02 } };
const aafUID_t AUID_AAFGeneva_ClipFramework =
{ 0x0D020101, 0x0101, 0x0300, { 0x06, 0x0E, 0x2B, 0x34, 0x02 } };
const aafUID_t AUID_AAFGeneva_GenevaObject =
{ 0x0D020101, 0x017F, 0x1000, { 0x06, 0x0E, 0x2B, 0x34, 0x02 } };
```

File: CAAFGeneva\_SceneFramework.h

```
class CAAFGeneva_SceneFramework :
public IAAF_Geneva_SceneFramework,
    IAAFPlugin,
    IAAFClassExtension,
    CAAF_Unknown
{
public:
    STDMETHOD( SetSceneNumber )
        ( aafCharacter_constptr pSceneNumber );
    STDMETHOD( GetSceneNumber )
        ( aafCharacter* pSceneNumber, aafUInt32 bufSize );
    STDMETHOD( GetSceneNumberBufLen )
        ( aafUInt32* bufSize );
    STDMETHOD( RemoveSceneNumber )( );
```

Example 2 – Generated C++ code for a portion of the Geneva Scheme library.