-OSTA.org-
Optical Storage
Technology Association

MultiPhoto/Video Specification
--
Basic Profile and Playback Profile

Metadata and Practices for Digital Photo-Video Collections

Revision 0.21
Working Draft

March 3, 2002

This document is a working draft for revie w by OSTA and I3A members and invited guests. It is a draft document and will be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use OSTA and I3A Working Draft documents as reference materials, to cite them in other publications, or to refer to them as anything other than a “work in progress”.

THIS DOCUMET IS NOT FOR GENERAL DISTRIBUTION OR PUBLICATION AND SHOULD NOT BE POSTED ON INTERNAL OR EXTERNAL WEBSITES.

POINTS OF CONTACT

OSTA	*Technical Content*
David Bunzel OSTA President Tel: +1 (650) 938-6945 Fax: +1 xxxx Email: dbunzel@osta.org http://www.osta.org	Pieter van Zee Editor, MultiPhoto/Video Specification Tel: +1 541-715-8658 Email: pieter_van_zee@hp.com Felix Nemirovsky Chairman, MultiRead Subcommittee Tel: +1 415 643 0944 Email: felixn@oaktech.com

ABSTRACT

This specification defines a format for processing and playback of collections of digital photo, video, and related audio and file content on an optical disc and other storage media.

LICENSING
IMPORTANT NOTICES

DIG 35 Specification Copyright Notice

Contents

Chapter 1: Introduction

1.1 Executive Summary

MultiPhoto/Video (MPV) is an open specification that makes easier the processing and playback of collections of photo-video content, including stills, stills with audio, still sequences, video clips, and audio clips. By analogy, MPV is added to the original data to enable slideshow and browsing tasks of photo-video content just as DPOF is added to the original data to enable printing of photo content.

It uses a simple text-based format that is easily understood and also easy to produce and consume programmatically in firmware or computer software. MultiPhoto/Video does *not* tackle a large number of problems at once – instead, it focuses on a few key problems that it solves with simple but robust approaches. Where possible and practical, it makes use of established specifications and standards.

The development and promotion of MultiPhoto/Video is sponsored jointly by two industry-leading trade associations, the Optical Storage Technology Assocation (OSTA) and the International Imaging Industry Association (I3A). The specification development and promotion process is open to all members; all organizations and individuals are welcomed as members. These associations include over 100 member companies from all over the world that produce products that collectively represent a majority marketshare in mainstream consumer digital imaging and recordable optical storage categories.

MultiPhoto/Video is not only a specification. It also includes a compliance test suite and processes, compliance testing materials, and a logo program for compliant products. In addition, some sample open-source code implementations of key steps in processing MPV content are available. These materials and procedures are made available and administered by OSTA at a modest cost. OSTA and I3A charge no royalty for use of the specification or logo.

The specification is being developed in phases and results in "profiles". Each profile in MultiPhoto/Video defines only those formats and practices that are necessary for the key tasks targeted by the profile. A number of candidate profiles for development have been identified, including:

- **Playback Profile**: two key tasks: viewing a slideshow and interactively browsing content collections
- **Internet Profile**: key task: interacting with and sending collections of photo-video content over the web and email
- **Capture Profile**: key task: writing new content to storage media and updating the collection info
- **Editing Profile**: key task: modifying existing collections of photo-video content.
- **Printing Profile:** key task: printing collections of photo-video content
- **Container Profile**: key task: storing photo-video content collections in containers

This document defines the Playback Profile of the MultiPhoto/Video specification, and targets consumer electronics devices, like DVD players, that wish to provide a firmware application to playback photo-video slideshows and browse photo-video content.

MultiPhoto/Video technology is founded on three essential and central elements: Collections, "Extended Assets", and Identification. Each of these make reference in various ways to "Primary Assets", which are the data files containing the primary photo-video content, like EXIF, JFIF, TIFF, WAV, MP3, MPG, etc.

1.2 Overview

MultiPhoto/Video (MPV) is an open specification to enhance interoperability, ease-of-use, and abilities to play and manipulate collections of photo/video content, including still images, still with audio, still sequences, video clips, audio-only clips, and related files. MPV is made available at low cost and without royalty from the Optical Storage

Technology Assocation (OSTA) and the International Imaging Industry Association (I3A). OSTA is an industry association promoting the use and interoperability of recordable CD and DVD discs in computer and consumer electronics devices. I3A is an industry association promoting digital and film imaging technologies.

MPV enables PC software and consumer electronics devices like DVD players to playback and manipulate collections of digital photo/video content including still images, still with audio, still sequences, video clips, audio-only clips, and related files. The emphasis is on personal content originating from many sources including digital cameras, film, scanners and video digitizer and stored on a range of media including memory cards, recordable or stamped CDs and DVDs, and even computer harddisks or internet services.

Development of the specification will be in multiple stages. A basic profile for use by DVD players and media player software to provide slideshows and interactive browsing of digital photo/video content will be completed first – that is this document. Another basic profile for photo-video capture products like digital cameras, scanners, and imaging software will be developed subsequently. Both profiles will be fairly simple and easy to support.

The MPV specification will further promote adoption of current and new categories of digital imaging products by enhancing ease-of-use and interoperability of photo/video content collections and applications. The format enables an end-user experience that starts fast, is highly interactive, provides for playing and editing collections of photo/video content, never reveals the underlying storage file system, and can be implemented in firmware of consumer electronics devices like DVD players as well as by PC software. MPV can be produced automatically or interactively by digital cameras, scanners, imaging software, internet services and other devices.

MPV provides specific manifest and metadata formats and implementation practices that support existing industry specifications such as the World Wide Web Consortium's SMIL, I3A's DIG35, and Adobe's eXtensible Metadata Platform XMP. MPV is compatible with and supports the DCF and EXIF specifications from the JEITA and JCIA that are widely used in digital cameras. New metadata elements will be developed as necessary. The work is oriented to deliver tangible and useful results in the near-term.

Support for MPV can be "added on" to existing applications and conventions because it is non-invasive and can co-exist with existing file system structures and formats. The format is designed for longevity and extensibility through the use of industry-standard XML. The manifest format will support write-only media, high-performance update, and use in low-memory, low-performance devices.

Key technical advances provided by the MPV specification specifically enable or enhance interoperability and end-user experience. Collections of photo-video content can be specified with optional presentation information. Practices for how to represent, compute, insert, and compare identifiers of digital assets enable collections to be more robust when assets are renamed or moved. Metadata for compound assets like still image sequences and primary and dependent assets (e.g. thumbnails, low-res renditions) allow manipulation of higher level constructs than the individual primary assets.

The MPV format does not contain the content itself -- MPV is an aggregation of information about the content, including references to the content. It provides essentially a Table of Contents and metadata repository; a typical implementation is a stand-alone file such has "TOC.MPV" and zero or more dependent files.

Chapter 2: Key Concepts

MultiPhoto/Video has a few key concepts and approaches.

2.1 Collections

Collections are assembled using a few basic concepts.

Album

A collection is an ordered group of items called an Album. Albums can reference other albums. Albums can be grouped together in one file or isolated in separate files. Album references use URIs, allowing reference to local or remote albums.

Primary and Secondary Objects

Primary objects are objects are at the top-level of a collection. Secondary objects may be contained by primary objets. Typically, users interact conceptually with primary objects, while secondary objects enhance performance or represent additional information about the object.

Basic Media Objects

A MPV collection may contain the following types of media objects. MPV does not constrain which formats of these media objects may be in a collection. Basic media objects may be either primary or secondary assets.

- AlbumRef
- Audio
- File
- Image
- Metadata
- PrintDoc
- Text
- Video

Any media object may contain other assets as secondary assets.

Composite Media Objects

In addition to the basic media objects, MPV also defines composite assets, which are semantically meaningful groups of media objects. These correspond to typical capture modes of digital cameras.

- ImageWithAudio
- ImageMultishotSequence
- ImagePanoramaSequence
- ObjectsBag
- ObjectsSequence

Composite media objects may be either primary or secondary assets. The ObjectsSequence and ObjectsBag object is an arbitrary mix of any of the types of available media objects; while they allow for arbitrary expression of a composite collection of media objects, they also lack the power direct association with the user's capture mode.

Renditions

Any basic or composite media object may have multiple renditions. Renditions are derived versions of the original media object. The relationship between the original and the rendition is captured in metadata. The derived version

may be direct, as in a screen resolution image of a hi-res image, or indirect, as in a video stream or print rendition of a collection.

Presentation

The MPV Specification defines how to represent collections. Primary user tasks for collections are to allow the user to play a slideshow of or interactively browse the primary objects in the collection. The MPV Playback Profile extends the spec with very basic presentation information to enhance the user's experience.

The overall approach derived from SMIL, a powerful XML format for representing presentations from the World Wide Web Consortium (W3C). MPV Playback Profile is a very constrained derivative of SMIL that provides just a basic level of presentation control. A MPV document can be mechanically translated into any of the common SMIL profiles. This makes MPV a good intermediate represenation and also suggests a MPV implementation strategy on platforms that also have SMIL players.

MPV Extraction

MPV can be embedded in SMIL-based presentations and mechanically extracted using an efficient single-pass streaming data algorithm. This allows the MPV collection document to serve other meaningful purposes; a typical implementation will be playable using applications bundled with major operating systems, including Windows XP [HTML+Time profile in Internet Explorer] and Macintosh OS X.[SMIL 1.0 profile in QuickTime player].

Firmware or computer-based MPV players extract the MPV collection from the document. The extraction process normalizes the input data to just the set of capabilities supported by the firmware. This provides forward compatibility with future changes to the MPV specification with existing MPV players.

MPV cannot adopt nor recommend a single SMIL profile to implement because numerous profiles exist and are mutually incompatible. These compatibility decisions should be made by MPV implementers.

2.2 Metadata

MPV is Metadata, not Data

MPV provides metadata to describe asset collections. It does not contain the actual asset data files themselves. The set of MPV metadata defines collections, identifiers, composite assets, and a basic set of presentation information.

XML Packets

MPV uses XML packets to provide for embedding and extracting MPV metadata in arbitrary files. The XML packet format is defined by Adobe's XMP specification.

Other Metadata

Generally speaking, MPV recommends that metadata about basic media objects be embedded in the asset. Recommended practices are provided for using existing metadata formats in typical media file formats, such as EXIF, JFIF, TIFF, WAV, MP3, MPG, AVI, and MOV. Metadata for composite media objects often cannot reside only in the basic media objects because it spans multiple asset files. This information is often stored in various established metadata formats such as I3A's DIG35 and Adobe's XMP.

2.3 Identifiers

Types of Identifiers

Identifiers are the means by which references are made between a collection and the assets it references. All basic and composite media objects in a collection are identified by two or more identifiers. There are four kinds of identifiers:

- documentID – the same for all renditions
- instanceID – different for each rendition
- lastURL – last known location
- id – XML-style identifier for reference to elements in an XML document

More than one of each kind of identifier may be used. For example, multiple lastURLs may be provided to allow for different filenames in different file systems, such as on a CD. Multiple instanceIDs may be provided that utilize different computation algorithms with various tradeoffs of speed and robustness.

The lastURL can be a local filename or remote URL. Significantly, lastURL is not a robust reference; it is broken easily by the user renaming or rearranging the referenced assets. Equally, the lastURL can be broken easily when a collection and assets are transferred across devices, storage formats and file systems.

For example, data CDs typically have several file systems with differing abilities to represent long filenames. Each device and operating system choose one file system to be active at a given time. The lastURL values of any collection referring to datafiles with long file or directory names that is placed on a CD can be broken if the player device uses a different file system that doesn't support long names. Thus the file named "Trip to the beach with Mom and Dad and the kids on Memorial Day 2001.JPG" can be stored on a data CD, but due to its length, the file name is different in each of the four common file systems the CD may have: ISO 9660, Joliet, HFS, and UDF.

To be robust against broken lastURL names, MPV provides identifier mechanisms and practices that allow the lastURL values to be fixed up when broken by searching for files with identifiers that match those contained in the collection.

Computing Identifiers

Identifiers can be computed and inserted in media objects in a variety of ways.

- arbitrary identifiers – computed in some manner independent of the asset data and assigned to the asset. Arbitrary identifiers are typically quick to generate and compare but are fragile because if they are damaged or lost, they cannot be reconstructed.
- content-based identifiers – computed in some manner dependent on the asset data. Content-based identifiers are typically slower to generate and compare, but are more robust and also less invasive because they can be regenerated based on the content itself.

Arbitrary identifiers are computing using a variety of algorithms typically available in the operating system. MPV uses the UUID 128-bit identifier which is readily generated by most modern operating systems. Sample source code for computing an assigned identifer is provided and can be used for firmware implementations.

Many content-based identifier computation methods exist. MPV specifies the MD5 algorithm as the basic algorithm that should always be supported. MD5 computes a 128-bit hash of the byte values in an arbitrary set of content.

Chapter 3: Locating and Extracting MPV Data

3.1 Design Approach

The MPV collection document is the essential document to be managed and manipulated for collections of photo-video content. MPV collections define a structured association of primary assets and provide access to metadata about those assets.

MPV documents can be located by name or by extension.

MPV collections have a very powerful capability: they can be *embedded* in XML documents that use a different and arbitrary XML schema and then extracted on-the-fly using an extremely efficient operation while parsing.

The power of this technique is extremely significant – it allows an MPV collection to be defined within another XML document. This allows the MPV collection document to serve other meaningful purposes; a typical implementation will be a HTML or SMIL document that provides access to the collection using other popular applications, such as a web browser.

MPV documents can also be embedded in binary documents and located using a simple byte stream scanner without further understanding of the binary format. This ability utilizes the definition of XML packets, as defined by Adobe for their XMP Extensible Metadata Platform.

3.2 Finding an MPV Document

The following algorithm describes how an MPV document is located.

```
When a removable storage unit is mounted, e.g. an optical disc inserted, the starting
current directory is the root directory.  Otherwise, the current directory is as decided by
the application conducting the search.

In the current directory, look for a file with one of the following case-insensitive names
according to the order given.
     TOC.MPV
     INDEX.MPV
     INDEX.HTML
     INDEX.HTM
     DEFAULT.HTML
     DEFAULT.HTM
     HOMEPAGE.HTML
     HOMEPAGE.HTM
     INDEX.SMIL
     INDEX.SMI
     INDEX.<any extension>
     <any name>.MPV

If no matching file is found, the subdirectories of the current directory are scanned in an
alphabetical breadth-first traversal to a depth of three subdirectories.

The first file matching the pattern is then processed according to the MPV processing
algorithm described below to extract the MPV document.
```

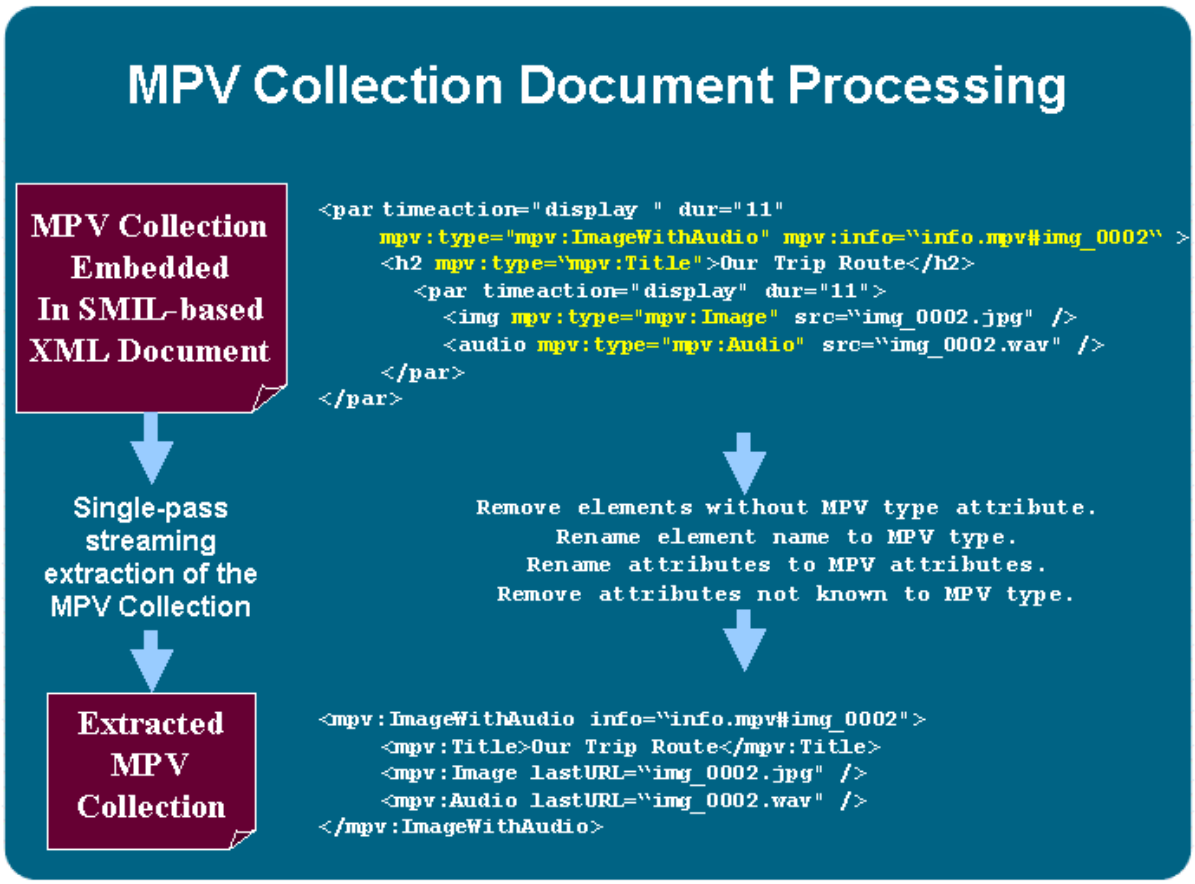

The rationale behind this algorithm is to first locate any top-level document containing MPV information, with a fallback of then finding named MPV collections. It is allowed for the MPV document to be located several directories down from the top, such as when stored in the same directory containing media objects structured according to the DCF specification, such as /DCIM/100DSCAM. One advantage placing the MPV document in the /DCIM/100DSCAM directory is that it can be merged with other DCF-structured assets without collision because the camera maker provides a unique directory name under /DCIM.

3.3 MPV Document Extraction

MPV documents can be embedded in SMIL-based presentations and mechanically extracted using an efficient single-pass streaming data algorithm. This allows the MPV collection document to serve other meaningful purposes; a typical implementation will be playable using applications bundled with major operating systems, including Windows XP [HTML+Time profile in Internet Explorer] and Macintosh OS X.[SMIL 1.0 profile in QuickTime player]. Significantly, it also isolates firmware implementations from evolving specifications of input formats.

3.3.1 Algorithm

This algorithm allows a set of one or more html+time or SMIL2 input documents to be processed in a single pass to create a single MPV output document. Additional input documents may be processed as a result of encountering external references in the initial input document set. Note that processing can be implemented using a standard XSLT processor. However, because MPV will often be processed in firmware, the extraction algorithm is extremely simple and does not require an XSLT processor. Sample implementations should be available.



The algorithm is conceptually simple


```
Emit MPV document header
WHILE another XML element to process
{
      if an MPV processing instruction found
            execute the PI
            exit WHILE loop
      if element is not in MPV namespace
            if the mpv:type attribute exists
                  replace element name with mpv:type name
                  remove mpv:type attribute
            else if the element is known to the player
                  handle in player-specific manner
            else
                  next element
      for all attributes
            if attribute name is in MPV namespace
                  next attribute
            if attribute name is known for this element type
                  rename attribute to MPV attribute name
            if attribute name is not in MPV namespace, discard
}
Emit MPV document footer
```

The resulting algorithm efficiently generates an MPV document that contains only elements and attributes that are in the MPV namespace or are known to the player application. This algorithm can be implemented as an integral part of a custom XML parser or inserted upstream of an existing parser.

NOTE: we may want to support a feature like rdf's parseType="Literal" for raw transfer of XML content. This is not currently supported in XMP

Processing Instructions

The MPV processor is configured via processing instructions to know about the rewrite rules for attributes.

```
<?mpv type="AttrRewrite" mpvElement="element-name"
      srcAttr="source-attribute" mpvAttr="mpv-attribute" ?>
```

When the MPV processor encounters a "AttrRewrite" MPV processing instruction in an XML document, the MPV processor immediately adds the rule to the processing algorithm.

```
<?mpv type="RewriteRulesRef" ref="url-ref-to-Rewrite-Rules" ?>
```

When it encounters a "RewriteRulesRef" instruction, the processor stops processing the current document, locates the referenced document, and adds its configuration information to the processing algorithm.

3.3.2 Album References

A MPV document contains one or more Albums. The Album is made up of one or more AlbumRef entries. The top-level entries may be made available to the system in a variety of ways a few of which are described in this document.

There are two types of AlbumRef entries. The Processing Instruction entry and the element entry. They are intended to be equivalent although one is oriented for transparent processing whereas the other is oriented to presentation.

Processing Instruction

```
<?mpv type="AlbumRef" ref="url-ref-to-Album" ?>
```


When the MPV processors encounters the "AlbumRef" MPV processing instruction in an XML document, the MPV processor immediately stops processing the current document and seeks to locate the referenced document. Processing continues with this document.

This powerful technique allows the MPV document to be in separate document from the document that matches the naming conventions, allowing for easy integration into existing systems.

Element

The element style of AlbumRef is intended to serve as both a presentation element in the HTML+TIME or SMIL show and as a representation of the information needed for the mpv-toc. Since the possible approaches for presenting a hierarchical listing are open-ended, there is no constraint on the possible element type that the mpv attributes will be attached to. The only constraint is that the element occur before any of the Album elements and that it contain both the mpv:type attribute and the mpv:lastURL attribute.
<base-element mpv:type="mpv:AlbumRef" />

3.3.3 MPV Type to SMIL Type Mapping

It is readily possible to author a SMIL document coTntaining MPV markup which can be properly extracted using the MPV processing algorithm. Alternately, it is readily possible to transform an MPV document into any of several kinds of SMIL documents.

This section gives a element-level mapping between MPV and the corresponding SMIL elements.

Table: MPV to SMIL element mappings

mpv:type of src	html+time	smil2
mpv:Album	**t:par**	**smil20:par**
mpv:Background	**t:seq**	**smil20:seq**
mpv:Foreground	**t:seq**	**smil20:seq**
mpv:AlbumRef	**t:ref**	**smil20:ref**
mpv:Audio	**t:audio**	**smil20:audio**
mpv:File	**t:ref**	**smil20:ref**
mpv:Image	**t:img or img**	**smil20:img**
mpv:ImageMultishotSequence	**t:seq**	**smil20:seq**
mpv:ImagePanoramaSequence	**t:seq**	**smil20:seq**
mpv:ImageWithAudio	**t:par**	**smil20:par**
mpv:Metadata	**t:ref**	**smil20:ref**
mpv:Par	**t:par**	**smil20:par**
mpv:Print	**t:ref**	**smil20:ref**
mpv:Seq	**t:seq**	**smil20:seq**
mpv:Text	**t:text**	**smil20:text**
mpv:TransitionFilter	**t:transitionFilter**	**smil20:transitionFilter**
mpv:Video	**t:video**	**smil20:video**

3.4 XML Packets

MPV collections may be embedded in arbitrary files when wrapped in an XML packet. The following section was excerpted from "*XMP – Extensible Metadata Platform 14 Sept 01"* , Copyright 2001 Adobe Inc. The objective is to justify and specify the use of XML packets in a manner wholly identical to that used by Adobe.

The XML Packet format was developed by Adobe to enable simple scanners to find XML data embedded in files with formats that a simple scanner may not understand, such as Photoshop® or PDF files. The format uses a syntax that is as close to XML as possible to minimize the filtering burden on the simple scanner.

The XML Packet format was designed to accomplish the following:

- Support embedding in binary and text formats, including the various Unicode encodings.
- Deal with arbitrary positioning within a byte stream (so as not to rely on machine word boundaries, etc.)
- Enable multiple XML packets to be embedded in a single data .le.
- Provide easy-to-scan markers for delimiting the XML packet. Such markers should be XML syntax-compatible to allow transmission to an XML parser without additional filtering.
- Enable in-place editing, including expansion, of metadata embedded in XML packets. The procedure for creating a XML Packet is described in this section. The packet includes a header and trailer (see Figure 3.3). The header provides byte ordering information, and optional encoding information.

The full specification for XML Packets is found in the Appendix.

Chapter 4: MPV Basic Profile Schema

The basic profile for the MultiPhoto/Video specification provides for the definition of collections of media objects. The following schema elements and attributes make up the MPV Basic Profile.

4.1 Schema Information

Almost all of the MPV schema uses the MPV namespace. Basic XML schema types are defined in the XS namespace. The introductory schema information is expressed as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:osta-com:mpv:1.0" xmlns:mpv ="urn:osta-com:mpv:1.0"
xmlns:xs="http://www.w 3.org/2001/XMLSchema" elementFormDefault ="qualified" attributeFormDefault="qualified">
```

4.2 Reference Attributes Base Type

MPV collections are built using references to media objects. All references are based on this type, which provide for four types of identifiers which can be used to reference other objects.

```
<xs:complexType name =" RefAttrBaseType">
  <xs:attribute name ="documentID" type="xs:string"/>
  <xs:attribute name ="instanceID" type ="xs:string"/>
  <xs:attribute name ="lastURL" type="xs:string"/>
  <xs:attribute name ="id" type ="xs:ID"/>
</xs:complexType>
```

None of the attributes is required.

documentID

An identifier that is the same for all renditions including the original.

instanceID

An identifier that is different for each rendition, allowing the rendition to be uniquely identified.

lastURL

The last known location can be a local filename or remote URL. The value of lastURL may carry arguments using standard URL syntax, "`lastURL?arg1=<value>&arg2=<value>…`". This allows the lastURL reference to carry information useful to accessing the target object. The order of the arguments is not relevant and argument names are case-insensitive. All MPV arguments carry the "mpv" prefix in the argument name. The "<value>" string uses the syntax appropriate to the argument. Any URN-illegal characters are translated in the usual way.

Significantly, lastURL is not a robust reference; it is broken easily by the user renaming or rearranging the referenced assets. Equally, the lastURL can be broken easily when a collection and assets are transferred across devices, storage formats and file systems. However, any arguments are still valid even if the basename is broken.

The application processing lastURL to open a local file may need to remove the arguments before using the lastURL, depending on the operatng system and APIs used.

id

XML-style identifier for reference to elements in an XML document

The type of identifier and its value are combined in the attribute's value string. This allows a variety of identification algorithms to be applied. A player application must be able to interpret the algorithm string in order to accurately regenerate or extract the identifier from a candidate object.

More than one of each kind of identifier may be used. For example, multiple lastURLs may be provided to allow for different filenames in different file systems, such as on a CD. Multiple instanceIDs may be provided that utilize different computation algorithms with various tradeoffs of speed and robustness.

Arbitrary identifiers are computing using a variety of algorithms typically available in the operating system. MPV uses the UUID 128-bit identifier which is readily generated by most modern operating systems. Sample source code for computing an arbitrary identifer is provided and can be used for firmware implementations.

Many content-based identifier computation methods exist; MPV considers these to be "digital signatures". MPV specifies the MD5 algorithm as the basic algorithm that should always be supported. MD5 computes a 128-bit hash of the byte values in an arbitrary set of content.

Identifier Syntax Definition

The following types of identifiers are defined by MPV at this time and may be used as the values of documentID, instanceID and even id.

urn:mpv:uuid

The id is computed based on an algorithm said to generate close to unique numbers but not based on file content. This type of identifier is well suited for use as a documentID or as an instanceID. Example:
`"urn:mpv:uuid:EF886AEFA3B340da971BAF09B17DBC122"`

urn:mpv:dsig:all:<algorithm>

Every byte in the entire file is processed. Example:
`"urn:mpv:dsig:all:md5:EF886AEFA3B340da971BAF09B17DBC122"`

urn:mpv:dsig:body:<algorithm>

Only the primary "body" of the file is processed. For example, in an EXIF file, only the primary JPEG-compressed data is processed. While more robust, this approach requires the processor to be able to interpret the file format sufficiently to isolate the body for processing. However, this may be common for many datatypes. This type of identifier is well suited for use as an instanceID. Example:
`"urn:mpv:dsig:body:md5:EF886AEFA3B340da971BAF09B17DBC122"`

urn:mpv:dsig:head:<byte count>:<algorithm>

Only the <byte count> integer number of bytes from the start of the file is processed. This is attractive to robustly refer to very large files or to files that are frequently edited or appended and for which the head can generate an approximately unique signature. Example:
`"urn:mpv:dsig:head:30000:md5:EF886AEFA3B340da971BAF09B17DBC122"`

urn:mpv:dsig:tail:<byte count>:<algorithm>

Only the <byte count> integer number of bytes from the end of the file is processed. This is attractive to quickly detect changes in files that are frequently edited or appended. Example:
`"urn:mpv:dsig:tail:30000:md5:EF886AEFA3B340da971BAF09B17DBC122"`

LastURL Syntax and Arguments Definition

Arguments may be placed on the lastURL value. When placed on the lastURL, the syntax is as follows:

lastURL?mpvByteOffset=<value>&mpvXMLPacket&mpvDocumentId=<value>&
mpvInstanceId=<value>&mpvId=<value>

The following arguments are defined by MPV for lastURL:

mpvByteOffset

Indicates a byte offset into the referenced file, such as "`lastURL?mpvByteOffset=3342`". This argument may be used even when lastURL refers to a local file. The processing application must detect the argument and seek to the specified byte offset in the file before reading any data. When the value of lastURL is resolved by a web server, the web server is the processing application and the MPV player receives a byte stream beginning at the offset.

mpvXMLPacket

Indicates the information in the referenced file is encapsulated in an XML packet. This argument may be used even when lastURL refers to a local file; it does not have a value. The processing application must detect the argument and seek to the XML packet in the file before reading any data. When the value of lastURL is resolved by a web server, the web server is the processing application and the MPV player receives a byte stream beginning at the packet.

mpvDocumentId, mpvInstanceId, mpvId

Putting identifiers on the URL can aide in resolving a broken reference when the lastURL value is used with a media management system. These arguments carry values that conform to their definition.

4.3 Discrete Media Attributes Base Type

All media objects support these basic attributes, in addition to the identifier attributes. Discrete media objects have no timeline component.

```
<xs:complexType name="MediaAttrBaseType">
  <xs:complexContent>
    <xs:extension base="mpv:RefAttrBaseType">
      <xs:attribute name="title" type="xs:string"/>
      <xs:attribute name="alt" type="xs:string"/>
      <xs:attribute name="mediaType" type="xs:string"/>
      <xs:attribute name="xml:lang" type="xs:string"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

mediaType

Content type of the media object referenced. This value takes precedence over other possible sources of the media type (for instance, the "Content-type" field in an HTTP or RTSP exchange, or the file extension). When the media object is available in many data formats, implementations MAY use the "type" value to influence which of the multiple formats is used. An extensive set of mediaTypes is defined by MPV. Please refer to the appendix.

tit

The title attribute offers advisory information about the object. Values of the title attribute may be rendered by players in a variety of ways.

alt

For MPV players that cannot display a particular media object, this attribute specifies alternate text. "alt" may be displayed in addition to the media or instead of media.

xml:lang

Used to identify the natural or formal language for the element's content.

4.4 Continuous Media Attributes Base Type

Media objects with time-based character support these attributes and act to modify how the object is perceived without actually editing the object data itself. This is not just presentation related – all acts on the object should utilize the clip times.

```
<xs:complexType name="ContMediaAttrBaseType">
  <xs:complexContent>
    <xs:extension base="mpv:MediaAttrBaseType">
      <xs:attribute name="clipBegin" type="xs:string"/>
      <xs:attribute name="clipEnd" type="xs:string"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

clipBegin

The clip-begin attribute specifies the beginning of a sub-clip of a continuous media object as offset from the start of the media object.
Values in the clip-begin attribute have the following syntax:

```
Clip-time-value   ::= Metric "=" ( Clock-val | Smpte-val )
Metric            ::= Smpte-type | "npt"
Smpte-type        ::= "smpte" | "smpte-30-drop" | "smpte-25"
Smpte-val         ::= Hours ":" Minutes ":" Seconds
                      [ ":" Frames [ "." Subframes ]]
Hours             ::= 2DIGIT
Minutes           ::= 2DIGIT
Seconds           ::= 2DIGIT
Frames            ::= 2DIGIT
Subframes         ::= 2DIGIT
```

The value of this attribute consists of a metric specifier, followed by a time value whose syntax and semantics depend on the metric specifier. The following formats are allowed:

SMPTE Timestamp

SMPTE time codes [SMPTE] can be used for frame-level access accuracy. The metric specifier can have the following values:

smpte

smpte-30-drop

These values indicate the use of the "SMPTE 30 drop" format with 29.97 frames per second. The "frames" field in the time value can assume the values 0 through 29. The difference between 30 and 29.97 frames per second is handled by dropping the first two frame indices (values 00 and 01) of every minute, except every tenth minute.

smpte-25

The "frames" field in the time specification can assume the values 0 through 24.

The time value has the format hours:minutes:seconds:frames.subframes. If the frame value is zero, it may be omitted. Subframes are measured in one-hundredth of a frame.
Examples:

```
clip-begin="smpte=10:12:33:20"
```

Normal Play Time

Normal Play Time expresses time in terms of SMIL clock values. The metric specifier is "npt", and the syntax of the time value is identical to the syntax of SMIL clock values.
Examples:


```
clip-begin="npt=123.45s"
clip-begin="npt=12:05:35.3"
```

clipEnd

The clip-begin attribute specifies the end of a sub-clip of a continuous media object (such as audio, video or another presentation) that should be played. It uses the same attribute value syntax as the clip-begin attribute. If the value of the "clip-end" attribute exceeds the duration of the media object, the value is ignored, and the clip end is set equal to the effective end of the media object.

4.5 Media Base Type

These two types are the base type of all media objects. Discrete objects (no timeline component) use the MediaBaseType, while continuous objects with a timeline component use ContMediaBaseType, giving access to the continuous media attributes.

```
<xs:complexType name="MediaBaseType">
  <xs:complexContent>
    <xs:extension base="mpv:MediaAttrBaseType">
      <xs:sequence>
        <xs:element ref="mpv:Title" minOccurs="0"/>
        <xs:element ref="mpv:Metadata" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ContMediaBaseType">
  <xs:complexContent>
    <xs:extension base="mpv:ContMediaAttrBaseType">
      <xs:sequence>
        <xs:element ref="mpv:Title" minOccurs="0"/>
        <xs:element ref="mpv:Metadata" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

4.6 <mpv>

The top-level element of an MPV document is <mpv>. It may contain one or more albums. The first album in the sequence is played first by a player. A typical implementation will use this album to reference other albums, either in the same document or in other documents.

```
<xs:element name="mpv" type="mpv:mpvType"/>
<xs:complexType name="mpvType">
  <xs:sequence>
    <xs:element name="mpv:Album" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

4.7 <mpv:Album>

An album defines a collection of media objects. They are organized in foreground and background collections. During playback, foreground and background are rendered in parallel for the slideshow. For interactive browsing, only foreground objects are used.

Renditions of an album represent derivatives of the collection. Typical renditions include a thumbnail representation of the album, a rendered video of the slideshow and print-formatted pages of the collection.

A typical use of the Background element is to specify a backdrop image to underly the thumbnails during interactive browsing and a sequence of audios to provide music during the slideshow.

```
<xs:element name="Album" type="mpv:AlbumType"/>
<xs:complexType name="AlbumType">
  <xs:extension base="mpv:RefAttrBaseType">
    <xs:sequence>
      <xs:element ref="mpv:Title" minOccurs="0"/>
      <xs:element ref="mpv:Meta" minOccurs="0"/>
      <xs:element ref="mpv:Background" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="mpv:Foreground" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="mpv:Rendition" minOccurs="0"/>
    </xs:sequence>
  </xs:extension>
  <xs:attribute name="mpv:defaultDur" type="xs:integer"/>
</xs:complexType>
```

4.8 <mpv:Title>

The title element is an alternative to using the title attribute to specify title content. When both are present, the title element's content takes precedence.

```
<xs:element name="Title" type="mpv:TitleType"/>
<xs:complexType name="TitleType" mixed="true"/>
```

4.9 <mpv:Meta>

The meta element provides an open-ended low-cost means to specify additional metadata via name-value paired attributes. The only vocabulary defined for 'name' is the Dublin Core set of names, which are in the dc: namespace. Items that are members of bag and seq values are identified by surrounding them in … elements.

```
<xs:element name="Meta" type="mpv:MetaType"/>
<xs:complexType name="MetaType">
  <xs:attribute name="name" type="xs:string"/>
  <xs:attribute name="value" type="xs:string"/>
</xs:complexType>
```

Table: Dublin Core Schema

The namespace prefix is dc. The namespace is http://purl.org/dc/elements/1.1/.

Property	Value Type	Description
dc:contributor	bag ProperName	Other contributors to document
dc:coverage	Text	The extent or scope of the resource
dc:creator	seq ProperName	Authors who created document
dc:date	seq Date	Date(s) that something interesting happened to the resource
dc:description	alt Text	A textual description of the resource, selected by language
dc:format	MIMEType	Data format
dc:identifier	Text	Unique identi.er of the resource
dc:language	bag Locale	Languages used in the content of the document
dc:publisher	bag ProperName	Publishers
dc:relation	bag Text	Relationships to other documents
dc:rights	alt Text	Informal rights statement, selected by language
dc:source	Text	Unique identifier of the work from which this resource was derived

dc:subject	bag Text	The topic of the resource
dc:title	alt Text	Document title
dc:type	bag XChoice (open)	novel, poem, working paper, etc.

4.10 <mpv:Foreground>, <mpv:Background>

The MPV specification allows collections of objects to be organized conceptually into foreground and background content. The player decides how best to mix and present these contents.

A typical use will be audio objects defined in the background to underly foreground content during slideshow playback. Because MPV Playback Profile includes only a limited amount of presentation information, only a modest level of playback control is possible.

```
<xs:element name="Background" type="mpv:SeqType"/>

<xs:element name="Foreground" type="mpv:SeqType"/>
```

***What about background color??

4.11 <mpv:Object>

The MPV specification defines a collection of media objects The Object element defines the set of primary media objects in MPV.

```
<xs:group name="Object" type="mpv:ObjectType"/>
<xs:complexType name="ObjectType">
  <xs:choice>
    <xs:element ref="mpv:AlbumRef"/>
    <xs:element ref="mpv:Audio"/>
    <xs:element ref="mpv:File"/>
    <xs:element ref="mpv:Image"/>
    <xs:element ref="mpv:ImageWithAudio"/>
    <xs:element ref="mpv:ImageMultishotSequence"/>
    <xs:element ref="mpv:ImagePanoramaSequence"/>
    <xs:element ref="mpv:Metadata"/>
    <xs:element ref="mpv:Par"/>
    <xs:element ref="mpv:Print"/>
    <xs:element ref="mpv:Seq"/>
    <xs:element ref="mpv:Text"/>
    <xs:element ref="mpv:Video"/>
  </xs:choice>
</xs:complexType>
```

4.12 <mpv:AlbumRef>

An AlbumRef element references another album using the same identifiers as all other media objects. A title and renditions may be supplied. A typical rendition would be a thumbnail representing the album.

```
<xs:element name="AlbumRef" type="mpv:AlbumRefType"/>
<xs:complexType name="AlbumRefType">
  <xs:complexContent>
    <xs:extension base="mpv:MediaBaseType">
      <!—The album is referenced using the attributes on the media base type -->
      <xs:element ref="Rendition" type="mpv:RenditionType" minOccurs="0"/>
    </xs:sequence>
  </xs:extension>
</xs:complexContent>
```



```
</xs:complexType>
```

4.13 <mpv:Audio>

The audio element specifies an audio object. A typical rendition would be a thumbnail trailer representing the audio track or a "subsampled" resolution representing a lower sampling rate rendition.

```
<xs:element name ="Audio" type="mpv:AudioType"/>
<xs:complexType name ="AudioType">
  <xs:complexContent>
    <xs:extension base="mpv:ContMediaBaseType">
      <xs:sequence>
        <!—The audio is referenced using the attributes on the media base type -->
        <xs:element ref="Rendition" type="mpv:RenditionType" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

4.14 <mpv:File>

The file element specifies an arbitrary file. If of a known type, mediatype attribute may specify the type of the file. A typical rendition would be a thumbnail representing the file.

```
<xs:element name ="File" type="mpv:FileType"/>
<xs:complexType name ="FileType">
  <xs:complexContent>
    <xs:extension base="mpv:MediaBaseType">
      <xs:sequence>
        <!—The file is referenced using the attributes on the media base type -->
        <xs:element ref="Rendition" type="mpv:RenditionType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

4.15 <mpv:Image>

The image element specifies an image object. Typical renditions would be thumbnail and screen resolution images.

```
<xs:element name ="Image" type="mpv:ImageType"/>
<xs:complexType name ="ImageType">
  <xs:complexContent>
    <xs:extension base="mpv:MediaBaseType">
      <xs:sequence>
        <!—The image is referenced using the attributes on the media base type -->
        <xs:element ref="Rendition" type="mpv:RenditionType" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

4.16 <mpv:ImageMultishotSequence>

The ImageMultishotSequence element groups a sequence of images and specifies the capture rate. A typical rendition would be a thumbnail representing the image sequence.

```
<xs:element name ="ImageMultishotSequence" type="mpv:ImageMultishotSequenceType"/>
<xs:complexType name ="ImageMultishotSequenceType">
```



```
<xs:complexContent>
  <xs:extension base="mpv:MediaBaseType">
    <xs:sequence>
      <xs:element ref="mpv:Title" minOccurs="0"/>
      <xs:element ref="mpv:Image" minOccurs="1"/>
      <xs:element ref="Rendition" type="mpv:RenditionType" minOccurs="0"/>
      <xs:element name="mpv:captureDur" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:extension>
</xs:complexContent>
</xs:complexType>
```

captureDur

The value of captureDur is a sequence of image-to-image durations that indicate the capture rate. The semicolon character ";" is used as a delimiter and the path begins with an algorithm declaration. The only rate algorithm defined by MPV is "FrameToFrame".

The frame to frame algorithm uses the following captureDur syntax: "`FrameToFrame:<clock value>;<clock value>;<clock value>`…". Clock value is always in relative time to the previous frame.

There are N-1 clock values for N images.

Example:
"FrameToFrame:0.4s;0.4s;0.4s": 4 images, each 0.4 seconds after the previous.

"FrameToFrame:2m0s;2m0s;2m0s": 4 images, each two minutes after the previous.

4.17 <mpv:ImagePanoramaSequence>

The ImagePanoramaSequence element groups a sequence of images taken to create a panorama and specifies the capture path. The degenerate case of one image in a sequence allows a user to capture only one image in this mode before changing capture modes without requiring the MPV data be rewritten. A typical rendition would be a thumbnail representing the sequence or an image representing the composite image formed by stitching together the image sequence.

```
<xs:element name="ImageMultishotSequence" type="mpv:ImageMultishotSequenceType"/>
<xs:complexType name="ImageMultishotSequenceType">
  <xs:complexContent>
    <xs:extension base="mpv:MediaBaseType">
      <xs:sequence>
        <xs:element ref="mpv:Image" minOccurs="1"/>
        <xs:element ref="Rendition" type="mpv:RenditionType" minOccurs="0"/>
        <xs:element name="mpv:capturePath" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

capturePath

The value of capturePath is a sequence of image-to-image motions that indicate the path. The colon character ":" is used as a delimiter and the path begins with an algorithm declaration. The only path algorithm defined by MPV is "FixedPt".

The fixed point algorithm uses the following capture path syntax:
"`fixedpt:<degrees>Y<degreees>P<degrees>R:<degrees>Y<degreees>P<degrees>R:`…"

Yaw-Pitch-Roll motions are in positive decimal degrees in 3D space assuming a fixed reference point, as follows: "<degrees>Y<degreees>P<degrees>R". There are N-1 motions for N images.

Yaw: 0 is no movement, 90 is rotation to the right, 270 is rotation to the left
Pitch: 0 is no movement, 90 is rotation upwards, 270 is rotation downwards
Roll: 0 is no movement, 90 is rotation clockwise, 270 is rotation counterclockwise.

Example:
"90Y0P0R:90Y0P0R:90Y0P0R": 4 images, each one rotating to the right of the previous.

"0Y90P0R:90Y0P0R:0Y270P0R": 4 images whose capture path describes a box in space

4.18 <mpv:ImageWithAudio>

The image element references an image object of some kind. Typical renditions would be thumbnail and screen resolutions of the image.

```
<xs:element name ="ImageWithAudio" type="mpv:ImageWithAudioType"/>
<xs:complexType name ="ImageWithAudioType">
  <xs:complexContent>
    <xs:extension base="mpv:MediaBaseType">
      <xs:sequence>
        <xs:element ref="mpv:Image" minOccurs="1" maxOccurs="1"/>
        <xs:element ref="mpv:Audio" minOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

4.19 <mpv:Metadata>

The metadata element specifies the location of metadata about the object. The metadata language may be specified by the media type. A typical rendition would the same metadata in a different media type.

```
<xs:element name ="Metadata" type="mpv:MetadataType"/>
<xs:complexType name ="MetadataType">
  <xs:complexContent>
    <xs:extension base="mpv:MediaBaseType">
      <xs:sequence>
        <!—The file is referenced using the attributes on the media base type -->
        <xs:element ref="Rendition" type="mpv:RenditionType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

4.20 <mpv:Seq>, <mpv:Par>

The Seq element defines a set of media objects in MPV that occur in an ordered sequence, without overlap. The Par element defines a set of media objects in MPV that occur synchronously with each other.

```
<xs:element name ="Seq" type="mpv:SeqType"/>
<xs:complexType name ="SeqType">
  <xs:extension base="mpv:MediaAttrBaseType">
    <xs:sequence>
      <xs:element ref="mpv:Title" minOccurs="0"/>
      <xs:group ref="mpv:Object" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:extension>
</xs:complexType>

<xs:element name ="Par" type="mpv:ParType"/>
```



```
<xs:complexType name ="ParType">
  <xs:extension base="mpv:RefAttrBaseType">
    <xs:sequence>
      <xs:group ref ="mpv:Object"  minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:extension>
</xs:complexType>
```

4.21 <mpv:Print>

The print element specifies a document containing print-formatted content. The formatting language may be specified by the media type. A typical rendition would be a thumbnail representing the file.

```
<xs:element name ="Print" type="mpv:PrintType"/>
<xs:complexType name ="PrintType">
  <xs:complexContent>
    <xs:extension base="mpv:MediaBaseType">
      <xs:sequence>
        <!—The file is referenced using the attributes on the media base type -->
        <xs:element ref="Rendition" type="mpv:RenditionType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

4.22 <mpv:Text>

The text element specifies a document containing text content. The formatting language may be specified by the media type. A typical rendition would be a thumbnail representing the file.

```
<xs:element name ="Text" type="mpv:TextType"/>
<xs:complexType name ="TextType">
  <xs:complexContent>
    <xs:extension base="mpv:MediaBaseType">
      <xs:sequence>
        <!—The file is referenced using the attributes on the media base type -->
        <xs:element ref="Rendition" type="mpv:RenditionType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

4.23 <mpv:Video>

The video element references a video stream of some kind. A typical rendition would be a thumbnail image representing the video.

```
<xs:element name ="Video" type="mpv:VideoType"/>
<xs:complexType name ="VideoType">
  <xs:complexContent>
    <xs:extension base="mpv:ContMediaBaseType">
      <xs:sequence>
        <!—The video is referenced using the attributes on the media base type -->
        <xs:element ref="Rendition" type="mpv:RenditionType" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```


4.24 <mpv:Rendition>

A rendition is a derivative of the primary object. Renditions should have the same "documentID" as the primary, but difference instanceIDs. A rendition can contain any number of media objects.

```
<xs:element name="Rendition" type="mpv:RenditionType"/>
<xs:complexType name="RenditionType">
  <xs:complexContent>
    <xs:extension base="mpv:MediaBaseType">
      <xs:attribute name="renditionClass" type="xs:string"/>
        <xs:group ref="mpv:Object"  minOccurs="0"  maxOccurs="unbounded"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

The vocabulary for renditionClass is based on Adobe's XMP specification and extended for MPV.

default

Indicates the master document; no additional tokens allowed

thumbnail

For a simplified and/or reduced preview of a version. Additional tokens, if any, provide more characteristics of the thumbnail. The colon character ":" is used as a delimiter. The recommended order is: thumbnail:<format>:<size>:<colorspace>.

Examples:

thumbnail:jpeg

thumbnail:160x120

thumbnail:16x16

thumbnail:gif:8x8:bw

screen

For a screen resolution/Web rendition. Has different resolution than default.

Examples:

screen:jpeg

screen:640x480

low-res

For a low quality, full size stand-in. Has the same resolution as default.

proof

For a review proof

draft

For a review rendition

page

Indicates a rendition of the content formatted for the printed page and ready for printing. Additional tokens, if any, provide more characteristics of the page. The colon character ":" is used as a delimiter. The recommended order is: page:<size>:<layout>:<colorspace>. Use the mediaType attribute to indicate media type.

size ::= letter|4x6|A4|A6
layout::= portrait|landscape

Examples:

page:letter:portrait

page:4x6:landscape

subsampled

A subsampled resolution rendition. Has different resolution than default. Additional tokens, if any, provide more characteristics of the subsampled rendition. The colon character ":" is used as a delimiter. The recommended order is: subsampled:<media type format>:<sample rate or size>:<colorspace>.

Examples:

subsampled:audio/mp3:64kbit

subsampled:stream/asf:256kbit

subsampled:image/jpeg:160x120

Chapter 5: MPV Playback Profile Schema

5.1 MPV Playback Profile

The MPV Playback Profile uses the MPV Basic Profile in total and adds a few elements and attributes. These enhance the possible playback experience. The key additions to the MPV specification are:

- Media Presentation Attributes – presentation-related information, such as duration
- Transition Filter – transition effects between objects

These changes are made by altering and extending the MPV Basic Profile schema.

5.2 Media Presentation Attributes Base Type

The playback profile adds these presentation attributes to all media objects by extending the MediaAttrBaseType.

New Attributes:

```
<xs:attribute name="dur" type="xs:string"/>
<xs:attribute name="repeatCount" type="xs:string"/>
<xs:attribute name="repeatDur" type="xs:string"/>
<xs:attribute name="fit" type="mpv:fitClassType"/>

<xs:simpleType name="fitClassType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="hidden"/>
    <xs:enumeration value="fill"/>
    <xs:enumeration value="meet"/>
    <xs:enumeration value="scroll"/>
    <xs:enumeration value="slide"/>
  </xs:restriction>
</xs:simpleType>
```

***ISSUE: image – rotate??

dur

Specifies the simple duration.
The attribute value can be any of the following:

Clock-value

Specifies the length of the simple duration, measured in element active time.
Value must be greater than 0.

Clock values have the following syntax:

```
Clock-value          ::= ( Full-clock-value | Partial-clock-value |
Timecount-value )
Full-clock-value     ::= Hours ":" Minutes ":" Seconds ("." Fraction)?
Partial-clock-value  ::= Minutes ":" Seconds ("." Fraction)?
Timecount-value      ::= Timecount ("." Fraction)? (Metric)?
Metric               ::= "h" | "min" | "s" | "ms"
Hours                ::= DIGIT+; any positive number
Minutes              ::= 2DIGIT; range from 00 to 59
Seconds              ::= 2DIGIT; range from 00 to 59
Fraction                 ::= DIGIT+
Timecount            ::= DIGIT+
2DIGIT                     ::= DIGIT DIGIT
DIGIT                         ::= [0-9]
```


"media"

Specifies the simple duration as the intrinsic media duration. This is only valid for elements that define media.

"indefinite"

Specifies the simple duration as indefinite.

repeatCount

Specifies the number of iterations of the simple duration. It can have the following attribute values:

numeric value

This is a (base 10) "floating point" numeric value that specifies the number of iterations. It can include partial iterations expressed as fraction values. A fractional value describes a portion of the simple duration. Values must be greater than 0.

"indefinite"

The element is defined to repeat indefinitely (subject to the constraints of the parent time container).

repeatDur

Specifies the total duration for repeat. It can have the following attribute values:

Clock-value

Specifies the duration in element active time to repeat the simple duration.

"indefinite"

The element is defined to repeat indefinitely (subject to the constraints of the parent time container).

fit

This attribute specifies the behavior if the intrinsic height and width of a visual media object differ from the values specified by the height and width attributes in the "region" element. This attribute replaces the behavior defined in CSS2.
This attribute can have the following values:

fill

Scale the object's height and width independently so that the content just touches all edges of the box.

hidden (default)

- If the intrinsic height (width) of the media object element is smaller than the height (width) defined in the "region" element, render the object starting from the top (left) edge and fill up the remaining height (width) with the background color.
- If the intrinsic height (width) of the media object element is greater than the height (width) defined in the "region" element, render the object starting from the top (left) edge until the height (width) defined in the "region" element is reached, and clip the parts of the object below (right of) the height (width).

meet

Scale the visual media object while preserving its aspect ratio until its height or width is equal to the value specified by the height or width attributes, while none of the content is clipped. The object's left top corner is positioned at the top-left coordinates of the box, and empty space at the left or bottom is filled up with the background color.

scroll

A scrolling mechanism should be invoked when the element's rendered contents exceed its bounds.

slice

Scale the visual media object while preserving its aspect ratio so that its height or width are equal to the value specified by the height and width attributes while some of the content may get clipped. Depending on

the exact situation, either a horizontal or a vertical slice of the visual media object is displayed. Overflow width is clipped from the right of the media object. Overflow height is clipped from the bottom of the media object.

5.3 <mpv:Transition Filter>

A transition filter that implements a transition from the object before to the object after. This element is strictly presentation oriented.

```
<xs:element name="transitionFilter" type="mpv:transitionFilterType"/>
<xs:complexType name="TransitionFilterType">
  <xs:attribute name="type" type="xs:string" use="required"/>
  <xs:attribute name="subType" type="xs:string"/>
  <xs:attribute name="dur" type="xs:string"/>
  <xs:attribute name="mode" type="mpv:transModeClassType"/>
</xs:complexType>

<xs:simpleType name="transModeClassType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="in"/>
    <xs:enumeration value="out"/>
  </xs:restriction>
</xs:simpleType>
```

type

This is the type or family of transition. This attribute is required and must be one of the transition families listed.

subtype

This is the subtype of the transition. This parameter is optional and if specified, must be one of the transition subtypes appropriate for the specified type as listed. If this parameter is not specified then the transition reverts to the default subtype for the specified transition type.

mode

Indicates whether the transitionFilter's parent element will transition in or out. Legal values are "in" indicating that the parent media will become more visible as the transition progress increases and "out" indicating that the parent media will become less visible as the transition progress increases. The default value is "in".

dur

The default duration is the intrinsic duration built into the transition. All of the transitions have a default duration of 1 second.

MPV Player implementations are not required to implement any transitions. If they do implement transitions, the following transitions types are recommended to be implemented first.

Transition type	**Default Transition subtype**
barWipe	leftToRight
irisWipe	rectangle
clockWipe	clockwiseTwelve
snakeWipe	topLeftHorizontal

Please refer to the appendix for a complete set of defined transition types and subtypes.

5.4 Presentation-Enhanced MediaAttrBaseType

*** more here ***

```
<xs:complexType name="MediaAttrBaseType">
  <xs:complexContent>
    <xs:extension base="mpv:RefAttrBaseType">
      <xs:attribute name="title" type="xs:string"/>
      <xs:attribute name="alt" type="xs:string"/>
      <xs:attribute name="mediaType" type="xs:string"/>
      <xs:attribute name="xml:lang" type="xs:string"/>
      <xs:attribute name="dur" type="xs:string"/>
      <xs:attribute name="repeatCount" type="xs:string"/>
      <xs:attribute name="repeatDur" type="xs:string"/>
      <xs:attribute name="fit" type="mpv:fitClassType"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

5.5 Presentation-Enhanced MediaBaseType and ContMediaBaseType

*** more here ***

```
<xs:complexType name="MediaBaseType">
  <xs:complexContent>
    <xs:extension base="mpv:MediaAttrBaseType">
      <xs:sequence>
        <xs:element ref="mpv:Title" minOccurs="0"/>
        <xs:element ref="mpv:Metadata" minOccurs="0"/>
        <xs:element ref="mpv:TransitionFilter" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ContMediaBaseType">
  <xs:complexContent>
    <xs:extension base="mpv:ContMediaAttrBaseType">
      <xs:sequence>
        <xs:element ref="mpv:Title" minOccurs="0"/>
        <xs:element ref="mpv:Metadata" minOccurs="0"/>
        <xs:element ref="mpv:TransitionFilter" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```


Chapter 6: MPV Playback Profile Practices & Behavior

*** lots to do here ***

Chapter 7: Media Types Reference

7.1 File Formats

File formats can be different from media types. Many file formats are containers that can contain a variety of media types. File format can frequently be determined by interrogating the extension of a file.

Table: Typical File Format Type file name extensions.

Format Types	Description
ASF	Microsoft Advanced Streaming Format
AVI	Microsoft Audio-Video Interleaved Format
MOV	QuickTime Format
MPG	MPEG Format
WMA	Windows Media Audio
WMV	Microsoft Windows Media Video

Table: Suggested File Format Type values

Format Types	Description
AIFF	AIFF format
MPA	MPEG 1 Layer 2 Format
MP3	MPEG 1 Layer 3 Format
WAV	WAV format
WMA	Windows Media Audio

While file formats are important, the type of media in the format is more critical for a player to understand because that is what will determine in detail the player's ability to process the content. This information is specified as a "media type".

7.2 Introduction to Media Types

MPV defines a rich set of media types that can be used describe in some detail the type of media stream referenced. Note that media type is quite distinct from file format; it describes the format of the media stream within the file. Many file formats are container formats that can contain a variety of media types.

Note that the MPV media types use the same format but have different values than MIME types. That is because the set of available approved MIME types is sparse and poorly defined. In contrast, MPV tightly defines a comprehensive set of media types.

Media Type: This field is specified as a string, "<**Major type**>/<**Minor type**>". The major and minor type of the media contained in the file is separated by a slash "/", such as "stream/MPEG1VideoCD". The values are case insensitive.

Table: Major Media Type values.

Major Media Types	Description
Audio	Audio.
AUXLine21Data	Line 21 data. Used by closed captions.

File	File. Used by closed captions.
Interleaved	Interleaved. Used by Digital Video (DV).
Midi	MIDI format.
MPEG2_PES	MPEG-2. Used by DVD.
ScriptCommand	Data is a script command, used by closed captions.
Stream	Byte stream with no time stamps.
Text	Text.
Timecode	Timecode data.
Video	Video.

Table: Common video media types.

Media Type	Description	Sample contents
stream/MPEG1System	MPEG-1 System Stream	BYTE stream, no alignment
stream/MPEG1Video	MPEG-1 Native Video Stream	Array of video stream bytes (no system layer)
stream/MPEG1VideoCD	MPEG-1 System Stream for Video CD	BYTE stream, no alignment
video/MJPG	Motion JPEG (MJPG) compressed video in an AVI file	
video/MPEG1Packet	MPEG-1 Video Packet	Single MPEG-1 packet including packet header
video/MPEG1Payload	MPEG-1 Video payload	Byte-aligned MPEG-1 video data
video/QTJpeg	QuickTime Photo JPEG compressed data.	
video/QTMovie	Apple® QuickTime® compression.	

Table: Common audio media types.

Media Type	Description	Sample contents
audio/MPEG1Packet	MPEG-1 Audio Packet	Single MPEG-1 packet including packet header
audio/MPEG1Payload	MPEG-1 Audio payload	Byte-aligned MPEG-1 audio data
stream/AIFF	Data from AIFF file	
stream/AU	Data from AU file	
stream/DssAudio	Dss Audio	
stream/DssVideo	Dss Video	
stream/MPEG1Audio	MPEG-1 Native Audio Stream	Array of audio stream bytes (no system layer)
stream/WAVE	Data from WAV file	

7.3 Audio Types

NOTE: The detailed information for major and minor media types is based directly on extensive information available in Microsoft DirectShow 8.0 SDK and is Copyright 1995-2001 Microsoft Corp, All Rights Reserved. As granted by the terms of use in Microsoft's documentation license, this information has been excerpted and revised and included in the MPV specification for its own use without explicit permission.

Table: Major Media Type values.

Major Media Types	Description
Audio	Audio.
Midi	MIDI format.
Stream	Byte stream with no time stamps.

Table: Media Type values when Major Media Type is "Stream".

Media Types for Major Type = "Stream"	Description
stream/AIFF	Data from AIFF file
stream/AU	Data from AU file
stream/DssAudio	Dss Audio
stream/DssVideo	Dss Video
stream/MPEG1Audio	MPEG audio
stream/WAVE	Data from WAV file

Table: Media Type values when Major Media Type is "MPEG2_PES".

Media Types for Major Type = "MPEG2_PES"	Description
MPEG2_PES/DOLBY_AC3	Dolby data
MPEG2_PES/MPEG2_AUDIO	MPEG-2 audio data
MPEG2_PES/DVD_LPCM_AUDIO	DVD audio data

Table: Media Type values when Major Media Type is "Audio".

Media Types for Major Type = "Audio"	Description
PCM	PCM audio.
MPEG1Packet	MPEG1 Audio packet.
MPEG1Payload	MPEG1 Audio Payload.

The suggested media types vary significantly according to the actual format of MPEG-1 data. The following information summarizes the media types for MPEG-1 data.

Table: Media types for various types of MPEG-1 audio data.

Media Type	Description	Sample contents
stream/MPEG1Audio	MPEG-1 Native Audio Stream	Array of audio stream bytes (no system layer)
audio/MPEG1Packet	MPEG-1 Audio Packet	Single MPEG-1 packet including packet header
audio/MPEG1Payload	MPEG-1 Audio payload	Byte-aligned MPEG-1 audio data

7.4 Image Types

*** More do do here ***

7.5 Metadata Types

*** More do do here ***

binary/EXIF
text/EXIF
text/XMP
text/DIG35

7.6 Print Types

*** More do do here ***
application/pdf
text/XHTML

7.7 Text Types

*** More do do here ***
text/plain
text/HTML
text/XHTML

7.8 Video Types

NOTE: The detailed information for major and minor media types is based directly on extensive information available in Microsoft DirectShow 8.0 SDK and is Copyright 1995-2001 Microsoft Corp, All Rights Reserved. As granted by the terms of use in Microsoft's documentation license, this information has been excerpted and revised and included in the MPV specification for its own use without explicit permission.

Table: Media Type values when Major Media Type is "Video".

Minor Media Types for Major Type = "Video"	**Description**
video/ARGB32	ARGB, 32 bits per pixel. Uncompressed RGB samples with valid alpha bits.
video/CFCC	MJPG format produced by some cards.
video/CLJR	Cirrus Logic Jr YUV 411 format with less than 8 bits per Y, U, and V sample. Cinepak can produce it and Cirrus 5440 can produce an overlay with it. A Y sample at every pixel, a U and V sample at every fourth pixel horizontally on each line; every vertical line sampled.
video/CPLA	Cinepak UYVY format.
video/dvhd	High Definition DV format.
video/dvsd	Standard DV format.
video/dvsl	Long Play DV format.
video/IF09	Indeo produced YVU9 format with additional information about differences from the last frame. 9.5 bits per pixel but reported as 9.
video/IJPG	Intergraph JPEG format.
video/MJPG	Motion JPEG (MJPG) compressed video.

video/MPEG1Packet	MPEG1 Video Packet.
video/MPEG1Payload	MPEG1 Video Payload.
video/Overlay	Video delivered using hardware overlay.
video/Plum	Plum MJPG format.
video/QTJpeg	QuickTime JPEG compressed data.
video/QTMovie	Apple® QuickTime® compression.
video/QTRle	QuickTime RLE compressed data.
video/QTRpza	QuickTime RPZA compressed data.
video/QTSmc	QuickTime SMC compressed data.
video/RGB1	RGB, 1 bit per pixel. Palettized.
video/RGB24	RGB, 24 bits per pixel. Uncompressed RGB samples.
video/RGB32	RGB, 32 bits per pixel. Uncompressed RGB samples. Do not use the alpha bits with this media type. (Compare MEDIASUBTYPE_ARGB32.)
video/RGB4	RGB, 4 bits per pixel. Palettized.
video/RGB555	555 format of RGB, 16 bits per pixel. Uncompressed RGB samples.
video/RGB565	565 format of RGB, 16 bits per pixel. Uncompressed RGB samples.
video/RGB8	RGB, 8 bits per pixel. Palettized.
video/TVMJ	TrueVision MJPG format.
video/UYVY	UYVY format data. A packed YUV format. A Y sample at every pixel, a U and V sample at every second pixel horizontally on each line; every vertical line sampled. Probably the most popular of the various YUV 4:2:2 formats. Byte ordering (lowest first) is U0, Y0, V0, Y1, U2, Y2, V2, Y3, U4, Y4, V4, Y5, where the suffix 0 is the leftmost pixel and increasing numbers are pixels increasing left to right. Each 4-byte block is 2 image pixels.
video/VideoPort	Video port data, used with DVD.
video/VPVBI	Video port vertical blanking interval (VBI) data.
video/VPVideo	Video port video data.
video/WAKE	MJPG format produced by some cards.
video/Y211	YUV 211 format data. A packed YUV format. A Y sample at every second pixel, a U and V sample at every fourth pixel horizontally on each line; every vertical line sampled. Byte ordering (lowest first) is Y0, U0, Y2, V0, Y4, U4, Y6, V4, Y8, U8, Y10, V8, where the suffix 0 is the leftmost pixel and increasing numbers are pixels increasing left to right. Each 4-byte block is 4 image pixels.
video/Y411	YUV 411 format data. Same as Y41P.
video/Y41P	Y41P format data. A packed YUV format. A Y sample at every pixel, a U and V sample at every fourth pixel horizontally on each line; every vertical line sampled. Byte ordering (lowest first) is U0, Y0, V0, Y1, U4, Y2, V4, Y3, Y4, Y5, Y6, Y7, where the suffix 0 is the leftmost pixel and increasing numbers are pixels increasing left to right. Each 12-byte block is 8 image pixels.
video/YUY2	YUY2 format data. Same as UYVY but with different pixel ordering. Byte ordering (lowest first) is Y0, U0, Y1, V0, Y2, U2, Y3, V2, Y4, U4, Y5, V4, where the

	suffix 0 is the leftmost pixel and increasing numbers are pixels increasing left to right. Each 4-byte block is 2 image pixels.
video/YVU9	Standard YVU9 format uncompressed data. A planar YUV format. A Y sample at every pixel, a U and V sample at every fourth pixel horizontally on each line; a Y sample on every vertical line, a U and V sample at every fourth vertical line. 9 bits per pixel.
video/YVYU	YVYU format data. A packed YUV format. Same as UYVY but with different pixel ordering. Byte ordering (lowest first) is Y0, V0, Y1, U0, Y2, V2, Y3, U2, Y4, V4, Y5, U4, where the suffix 0 is the leftmost pixel and increasing numbers are pixels increasing left to right. Each 4-byte block is 2 image pixels.

Table: Media Type values when Major Media Type is "Stream".

Minor Media Types for Major Type = "Stream"	**Description**
stream/Asf	Advanced Streaming Format (ASF)
stream/Avi	Data from AVI file
stream/DssVideo	Dss Video
stream/MPEG1System	MPEG system
stream/MPEG1Video	MPEG video
stream/MPEG1VideoCD	MPEG video CD

Table: Suggested Minor Media Type values when Major Media Type is "MPEG2_PES".

Minor Media Types for Major Type = "MPEG2_PES"	**Description**
MPEG2_PES/DVD_SUBPICTURE	Subpicture data
MPEG2_PES/MPEG2_TRANSPORT	MPEG-2 transport stream
MPEG2_PES/MPEG2_TRANSPORT_STRIDE	MPEG-2 multiplexed packets (a.k.a. stride packets)
MPEG2_PES/MPEG2_PROGRAM	MPEG-2 program stream

Table: Major and minor types corresponding to various MPEG-1 data.

The media types vary significantly according to the actual format of MPEG-1 data. The following information summarizes the media types for MPEG-1 data.

Media Type	Description	Sample contents
stream/MPEG1System	MPEG-1 System Stream	BYTE stream, no alignment
stream/MPEG1VideoCD	MPEG-1 System Stream for Video CD	BYTE stream, no alignment
stream/MPEG1Video	MPEG-1 Native Video Stream	Array of video stream bytes (no system layer)
video/MPEG1Packet	MPEG-1 Video Packet	Single MPEG-1 packet including packet header
video/MPEG1Payload	MPEG-1 Video payload	Byte-aligned MPEG-1 video data

Chapter 8: Transition Types Reference

The following table is excerpted from the SMIL 2.0 specification. It lists the vocabulary of defined transition types and subtypes. The SMPTE Wipe Codes (where appropriate) are provided in parentheses after the subtype name and are for reference only. The Wipe Codes are not part of the transition subtype name. The default transition subtype for each type is indicated by the word [default].

Transition type	**Transition subtypes** (SMPTE Wipe Codes in parentheses)
Edge Wipes - wipes occur along an edge	
"barWipe"	"leftToRight" (1) [default], "topToBottom" (2)
"boxWipe"	"topLeft" (3) [default], "topRight" (4), "bottomRight" (5), "bottomLeft" (6), "topCenter" (23), "rightCenter" (24), "bottomCenter" (25), "leftCenter" (26)
"fourBoxWipe"	"cornersIn" (7) [default], "cornersOut" (8)
"barnDoorWipe"	"vertical" (21) [default], "horizontal" (22), "diagonalBottomLeft" (45), "diagonalTopLeft" (46)
"diagonalWipe"	"topLeft" (41) [default], "topRight" (42)
"bowTieWipe"	"vertical" (43) [default], "horizontal" (44)
"miscDiagonalWipe"	"doubleBarnDoor" (47) [default], "doubleDiamond" (48)
"veeWipe"	"down" (61) [default], "left" (62), "up" (63), "right" (64)
"barnVeeWipe"	"down" (65) [default], "left" (66), "up" (67), "right" (68)
"zigZagWipe"	"leftToRight" (71) [default], "topToBottom" (72)
"barnZigZagWipe"	"vertical" (73) [default], "horizontal" (74)
Iris Wipes - shapes expand from the center of the media	
"irisWipe"	"rectangle" (101) [default], "diamond" (102)
"triangleWipe"	"up" (103) [default], "right" (104), "down" (105), "left" (106)
"arrowHeadWipe"	"up" (107) [default], "right" (108), "down" (109), "left" (110)
"pentagonWipe"	"up" (111) [default], "down" (112)
"hexagonWipe"	"horizontal" (113) [default], "vertical" (114)
"ellipseWipe"	"circle" (119) [default], "horizontal" (120), "vertical" (121)
"eyeWipe"	"horizontal" (122) [default], "vertical" (123)
"roundRectWipe"	"horizontal" (124) [default], "vertical" (125)
"starWipe"	"fourPoint" (127) [default], "fivePoint" (128), "sixPoint" (129)
"miscShapeWipe"	"heart" (130) [default], "keyhole" (131)

Clock Wipes - rotate around a center point	
"clockWipe"	"clockwiseTwelve" (201) [default], "clockwiseThree" (202), "clockwiseSix" (203), "clockwiseNine" (204)
"pinWheelWipe"	"twoBladeVertical" (205) [default], "twoBladeHorizontal" (206), "fourBlade" (207)
"singleSweepWipe"	"clockwiseTop" (221) [default], "clockwiseRight" (222), "clockwiseBottom" (223), "clockwiseLeft" (224), "clockwiseTopLeft" (241), "counterClockwiseBottomLeft" (242), "clockwiseBottomRight" (243), "counterClockwiseTopRight" (244)
"fanWipe"	"centerTop" (211) [default], "centerRight" (212), "top" (231), "right" (232), "bottom" (233), "left" (234)
"doubleFanWipe"	"fanOutVertical" (213) [default], "fanOutHorizontal" (214), "fanIn Vertical" (235), "fanInHorizontal" (236)
"doubleSweepWipe"	"parallelVertical" (225) [default], "parallelDiagonal" (226), "oppositeVertical" (227), "oppositeHorizontal" (228), "parallelDiagonalTopLeft" (245), "parallelDiagonalBottomLeft" (246)
"saloonDoorWipe"	"top" (251) [default], "left" (252), "bottom" (253), "right" (254)
"windshieldWipe"	"right" (261) [default], "up" (262), "vertical" (263), "horizontal" (264)
Matrix Wipes - media is revealed in squares following a pattern	
"snakeWipe"	"topLeftHorizontal" (301) [default], "topLeftVertical" (302), "topLeftDiagonal" (303), "topRightDiagonal" (304), "bottomRightDiagonal" (305), "bottomLeftDiagonal" (306)
"spiralWipe"	"topLeftClockwise" (310) [default], "topRightClockwise" (311), "bottomRightClockwise" (312), "bottomLeftClockwise" (313), "topLeftCounterClockwise" (314), "topRightCounterClockwise" (315), "bottomRightCounterClockwise" (316), "bottomLeftCounterClockwise" (317)
"parallelSnakesWipe"	"verticalTopSame" (320), [default] "verticalBottomSame" (321), "verticalTopLeftOpposite" (322), "verticalBottomLeftOpposite" (323), "horizontalLeftSame" (324), "horizontalRightSame" (325), "horizontalTopLeftOpposite" (326), "horizontalTopRightOpposite" (327), "diagonalBottomLeftOpposite" (328), "diagonalTopLeftOpposite" (329)
"boxSnakesWipe"	"twoBoxTop" (340) [default], "twoBoxBottom" (341), "twoBoxLeft" (342), "twoBoxRight" (343), "fourBoxVertical (344), "fourBoxHorizontal" (345)
"waterfallWipe"	"verticalLeft" (350) [default], "verticalRight" (351), "horizontalLeft" (352), "horizontalRight" (353)
Non-SMPTE Wipes	
"pushWipe"	"fromLeft" [default], "fromTop", "fromRight", "fromBottom"
"slideWipe"	"fromLeft" [default], "fromTop", "fromRight", "fromBottom"
"fade"	"crossfade" [default], "fadeToColor", "fadeFromColo r"

The "pushWipe" transitions looks as if the destination media "pushes" the background media away. In other words, both the background media and the destination media are moving.

In the "slideWipe" transitions, the destination media moves, but the background media does not. The visual effect of "slideWipe" transitions is that the destination media is "sliding" in across the background media.

The "fade" transitions are pixel-by-pixel blends between the destination media and either the background media or a specified color.

Chapter 9: XML Packet Reference

MPV collections may be embedded in arbitrary files when wrapped in an XML packet. The following section was excerpted from "*XMP – Extensible Metadata Platform 14 Sept 01"* , Copyright 2001 Adobe Inc. The objective is to justify and specify the use of XML packets in a manner wholly identical to that used by Adobe.

The XML Packet format was developed by Adobe to enable simple scanners to find XML data embedded in files with formats that a simple scanner may not understand, such as Photoshop® or PDF files. The format uses a syntax that is as close to XML as possible to minimize the filtering burden on the simple scanner.

The XML Packet format was designed to accomplish the following:

- Support embedding in binary and text formats, including the various Unicode encodings.
- Deal with arbitrary positioning within a byte stream (so as not to rely on machine word boundaries, etc.)
- Enable multiple XML packets to be embedded in a single data .le.
- Provide easy-to-scan markers for delimiting the XML packet. Such markers should be XML syntax-compatible to allow transmission to an XML parser without additional filtering.
- Enable in-place editing, including expansion, of metadata embedded in XML packets. The procedure for creating a XML Packet is described in this section. The packet includes a header and trailer (see Figure 3.3). The header provides byte ordering information, and optional encoding information.

NOTE: Be aware that an XML packet might contain valid XML that is not necessarily MPV or XMP compliant. It is desirable to preserve such non-MPV and XMP XML if possible.

Here is a sketch of an XML packet showing the text of the header and trailer:

```
<?xpacket begin='• ¡' id='W5M0MpCehiHzreSzNTczkc9d'?>
... 700 bytes of XML data text ...
... 500 bytes of XML whitespace as padding ...
<?xpacket end='w'?>
```

Where '• ¡' represents the Unicode "zero width non-breaking space character" (U+FEFF) used as a byte-order marker.

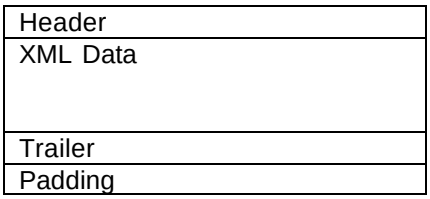


FIGURE: XML Packet Schematic

The entire packet must conform to the Well-Formedness requirements of the XML specification, except for the lack of an XML declaration at its start. Also, there are additional constraints:

- Different packets may be in different character encodings.
- Packets must not nest.
- Data attributes in the header and trailer processing instructions are separated by exactly one blank (U+0020) character.

The following sections describe the parts of the packet illustrated in Figure 3.3.

Header

The Header is an XML processing instruction:

```
<?xpacket ... ?>
```

The remainder of the processing instruction contains information about the packet. The syntax observes XML attribute syntax, which is production [41] Attribute, which is roughly:

```
Attribute ::= Name '=' AttValue
AttValue ::= '"' ([^<&"] | Reference)* '"' | ''' ([^<&'] | Reference)*
'''
```

Note the use of either matching single or double quotes. Otherwise, a common error would be the use of the wrong quote character.

The header processing instruction must have two or more attributes. The first attribute must be the *begin* attribute, the second must be the *id* attribute. Other attributes may appear in any order, and unrecognized attributes should be ignored. The description of each attribute follows.

Attribute: begin
This mandatory attribute is present only in the initial processing instruction, and indicates that it is the beginning of a new packet. The value of this attribute is the Unicode zero width nonbreaking space character U+FEFF in the appropriate encoding (UTF-8, UTF-16, or UTF-32). This serves as a byte order marker, where the character is written in the natural order of the authoring/generating application (consistent with the byte order of the XML data encoding). For backwards compatibility with earlier versions of the XML packet speci.cation the value of this attribute may be the empty string, indicating an 8-bit encoding.

As described in the *Usage Hints* below, an XML Packet processor should be reading a single byte at a time until it has successfully interpreted a valid packet header. While processing the value of the `begin` attribute, if the processor detects the byte value '0xFE' followed by '0xFF,' it knows that the packet is big-endian order. If the processor detects the byte value '0xFF' followed by '0xFE,' it knows that the packet is little-endian order. If the processor detects the byte value '0xEF,' followed by '0xBB,' followed by '0xBF,' it knows this is UTF-8. If the attribute has no value (quote or double quote followed immediate by another quote or double quote), the byte order is irrelevant and the overall character encoding *must not* be any 16- or 32-bit Unicode encoding (that is, it must be UTF-8, US-ASCII, etc.).

Attribute: id
Next, there is a mandatory `id`. For all packets de.ned by this version of the syntax, the value of the `id` is the following string of 7-bit ASCII characters:

```
W5M0MpCehiHzreSzNTczkc9d
```

The value of the attribute must be encoded in the character encoding of the overall packet (see below). Thus, if the overall encoding is big-endian UTF-16, the id value should be converted from 7-bit ASCII to UTF-16 by inserting nulls.

Attribute: bytes
An optional `bytes` attribute may be present, specifying the total length of the packet in bytes. If the length extends beyond the end of the trailer processing instruction, the additional bytes must be properly encoded Unicode whitespace and are considered padding.

NOTE: Earlier versions of this speci.cation recommended placement of the padding after the trailer processing instruction. This is now discouraged, the padding should come before the trailer. Placing the padding before the trailer and omitting the `bytes` attribute has always been valid, it is now the only recommended practice. Use of the `bytes` attribute is dangerous for XML packets embedded in text .les. For example, moving a text .le from a Macintosh or UNIX system to Windows typically causes all single byte line endings (CR or LF) to become 2 bytes (CRLF). This would invalidate the length given by the `bytes` attribute.

Attribute: encoding

The `id` attribute may be followed by an optional `encoding` attribute. It is identical to the `encoding` attribute in the XML declaration (see productions [23] and [80] in the XML specification). It speci.es the character encoding of the entire packet. If this attribute is omitted, the encoding of the packet must be UTF-8. The following is a simplified BNF syntax for the encoding attribute:

```
[A-Z a-z] ([A-Z a-z 0-9._] | -)*
```

XML Data

The bytes of the XML data are placed here. If the encoding is speci.ed in the Header, the encoding of the XML data must match. If the encoding was omitted from the Header, the encoding of the XML data must be UTF-8.

You should omit the XML declaration for the XML data when using this packet syntax for embedding. The XML specification requires that the XML declaration be "the first thing in the entity." This will never be the case for an embedded XML Packet, the somewhat ambiguous definition of "entity" with respect to embedding notwithstanding. You may preserve the information contained in your XML declaration by translating it into a comment or a processing instruction, such as:

```
<?was-xml version="1.0" standalone="yes"?>
```

Padding

In order to enable in-place edits and expansion of the embedded XML, padding should be added to the packet so that additions and edits may be easily made to the packet without overwriting existing application data. It is recommended that applications allocate 50% of the XML data size as padding, with a minimum of 4 KB. This padding must be XML compatible whitespace. The recommended practice is to use the blank character (U+0020) for padding, in the appropriate encoding, with a newline about every 100 characters.

Trailer

This mandatory processing instruction indicates the end of the XML packet.

```
<?xpacket ... ?>
```

This processing instruction has one mandatory attribute, described below. The `end` attribute must be the .rst attribute. Other unrecognized attributes may follow and should be ignored.

Attribute: end

This mandatory attribute indicates that this is the trailer. The value of the attribute is either "r" or "w". If "r", the packet is "read-only" and should not be updated in-place. If "w", the packet may be updated in-place if and only if there is available space through the padding. If the size of the Header+XML data+Trailer is less than it was before the update, the padding should be increased accordingly so that the overall packet size remains constant. Use the value "r" for file formats which compute invariants over all of their contents, such as checksums. If in doubt, use "r".

Chapter 10: Typographic Conventions

Schema are in yellow boxes in Courier font.

```
Plain text is schema.
```

Examples of MPV metadata structures are in Courier font.

```
Example.

<MPV>
   <ALBUM>
      ...
   </ALBUM>
</MPV>
```


Chapter 11: References

[DCF-1999]

"Design rule for Camera File system, Version 1.0", JEIDA standard, English Version 1999.1.7, Japanese Electronic Industry Development Association (JEIDA).

[DIG35-2001]

"DIG35 Specification – Metadata for Digital Images, Version 1.1 Working Draft", April 16, 2001, International Imaging Industry Association (I3A) [recently formed by combining the Digital Imaging Group and PIMA]. http://www.i3a.org

[C14N-2001]

"Canonical XML Version 1.0", March 15th, 2001, W3C, http://www.w3.org/TR/xml-c14n

[DATETIME]

"Date and Time Formats", M. Wolf, C. Wicksteed. W3C Note 27 August 1998,
Available at: http://www.w3.org/TR/NOTE-datetime

[DC]

"Dublin Core Metadata Initiative", a Simple Content Description Model for Electronic Resources.
Available at http://purl.org/DC/

[DOM1]

"Document Object Model (DOM) Level 1 Specification", W3C Recommendation 1 October 1998,
Available at http://www.w3.org/TR/REC-DOM-Level-1

[DOM2Events]

"W3C Document Object Model Level 2 Events", T. Pixley, W3C Candidate Recommendation,
Available at http://www.w3.org/TR/DOM-Level-2/events.html

[DOM2]

"W3C (World Wide Web Consortium) Document Object Model (DOM) Level 2 Specification. W3C Candidate Recommendation 10 May, 2000
Available at http://www.w3.org/TR/DOM-Level-2

[DOM2CSS]

"W3C (World Wide Web Consortium) Document Object Model (DOM) Level 2 Specification. W3C Candidate Recommendation 10 May, 2000. "Document Object Model CSS"
Available at http://www.w3.org/TR/DOM-Level-2/css.html

[HTML4]

"HTML 4.01 Specification" D. Raggett, A. Le Hors, I. Jacobs. W3C Recommendation 24 December 1999,
Available at http://www.w3.org/TR/html401/

[ISO8601]

"Data elements and interchange formats - Information interchange - Representation of dates and times", International Organization for Standardization, 1998.

[ISO10646]

""Information Technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane", ISO/IEC 10646-1:1993. This reference refers to a set of codepoints that may evolve as new characters are assigned to them. This reference therefore includes future amendments as long as they do not change character assignments up to and including the first five amendments to ISO/IEC 10646-1:1993. Also, this reference assumes that the character sets defined by ISO 10646 and Unicode remain character-by-

character equivalent. This reference also includes future publications of other parts of 10646 (i.e., other than Part 1) that define characters in planes 1-16. "

[JFIF]

"JPEG File Interchange Format, Version 1.02"; Eric Hamilton, September 1992.
Available at http://www.w3.org/Graphics/JPEG/jfif.txt

[MIME-2]

"RFC 2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types"; N. Freed, N. Borenstein, November 1996.
Available at ftp://ftp.isi.edu/in-notes/rfc2046.txt

[MOBILE-GUIDE]

"HTML4.0 Guidelines for Mobile Access", T. Kamada, Takuya Asada, Masayasu Ishikawa, Shin'ichi Matsui. W3C Note 15 March 1999
Available at http://www.w3.org/TR/NOTE-html40-mobile/

[PICS]

"PICS 1.1 Label Distribution - Label Syntax and Communication Protocols", T. Krauskopf, J. Miller, P. Resnick and W. Trees. W3C Recommendation 31 October 1996,
Available at http://www.w3.org/TR/REC-PICS-labels

[PNG-MIME]

"Registration of new Media Type image/png"; Glenn Randers-Pehrson, Thomas Boutell, 27 July 1996.
Available at ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/image/png

[PNG-REC]

"PNG (Portable Network Graphics) Specification Version 1.0"; Thomas Boutell (Ed.).
Available at http://www.w3.org/TR/REC-png

[QT]

"QuickTime Movie File Format Specification", May 1996.
Available at http://developer.apple.com/techpubs/quicktime/qtdevdocs/REF/refFileFormat96.htm

[QT-MIME]

"Registration of new MIME content-type/subtype"; Paul Lindner, 1993.
Available at http://www.isi.edu/in-notes/iana/assignments/media-types/video/quicktime

[RDFsyntax]

"Resource Description Framework (RDF) Model and Syntax Specification", Ora Lassila and Ralph R. Swick. W3C Recommendation 22 February 1999,
Available at http://www.w3.org/TR/REC-rdf-syntax/

[RDFschema]

"Resource Description Framework (RDF) Schema Specification", Dan Brickley and R.V. Guha.W3C Proposed Recommendation 03 March 1999,
Available at http://www.w3.org/TR/PR-rdf-schema/

[RFC1766]

"Tags for the Identification of Languages", H. Alvestrand, March 1995.
Available at ftp://ftp.isi.edu/in-notes/rfc1766.txt

[SMIL10]

"Synchronized Multimedia Integration Language (SMIL) 1.0" P. Hoschka. W3C Recommendation 15 June 1998,
Available at http://www.w3.org/TR/REC-smil.

[SMIL-ANIMATION]

"SMIL Animation" Patrick Schmitz and Aaron Cohen. W3C "Last Call" Working Draft, work in progress.
Available at http://www.w3.org/TR/http://www.w3.org/TR/smil-animation/

[SMIL20]

"Synchronized Multimedia Integration Language (SMIL 2.0) Specification". W3C Working Draft, work in progress.
Available at http://www.w3.org/TR/smil20/

[SMIL-CSS2]

"Displaying SMIL Basic Layout with a CSS2 Rendering Engine". W3C Note 20 July 1998,
Available at: http://www.w3.org/TR/NOTE-CSS-smil.html

[SMIL-DOM]

"Synchronized Multimedia Integration Language Document Object Model (DOM)". W3C Working Draft, work in progress.
Available at http://www.w3.org/TR/smil-boston-dom/

[SMIL-MOD]

"Synchronized Multimedia Modules based upon SMIL 1.0", Patrick Schmitz, Ted Wugofski and Warner ten Kate. W3C Note 23 February 1999,
Available at http://www.w3.org/TR/NOTE-SYMM-modules

[SMPTE-EDL]

"Transfer of Edit Decision Lists", ANSI/SMPTE 258M/1993

[UAAG]

"User Agent Accessibility Guidelines 1.0", Jon Gunderson, Ian Jacobs. W3C Proposed Recommendation 10 March 2000
Available at http://www.w3.org/TR/UAAG10/

[URI]

"Uniform Resource Identifiers (URI): Generic Syntax", T. Berners-Lee, R. Fielding, L. Masinter, August 1998. Note that RFC 2396 updates [RFC1738] and [RFC1808].

[W3C-NSURI]

"URIs for W3C namespaces". Policy and administrative issue for W3C, Oct. 1999.
Available at http://www.w3.org/1999/10/nsuri

[WAI-SMIL-ACCESS]

"Accessibility Features of SMIL " Marja-RiittaKoivunen, Ian Jacobs, W3C NOTE 21 September 1999.
Available at http://www.w3.org/TR/SMIL-access

[XML10]

"Extensible Markup Language (XML) 1.0" T. Bray, J. Paoli and C.M. Sperberg-McQueen. W3C Recommendation 10 February 1998 ,
Available at http://www.w3.org/TR/REC-xml

[XML-NS]

"Namespaces in XML", Tim Bray, Dave Hollander, Andrew Layman. W3C Recommendation 14 January 1999, ,
Available at http://www.w3.org/TR/REC-xml-names

[XSCHEMA]

"XML Schema, XML Schema Part 1: Structures". W3C Working Draft, work in progress.
Available at http://www.w3.org/TR/xmlschema-1/

[XSL]

"Extensible Stylesheet Language (XSL) Specification", Stephen Deach. W3C Working Draft, work in progress. Available at http://www.w3.org/TR/xsl/

Chapter 12: ToDo and Things to remember & discuss

- include a chapter on UUID insertion into JPEG EXIF and JFIF images, MPEG video, AVI, and QuickTime files for identification
- multiple lastURLs and identifiers on any ref – is this compatible with using attribute-base references?
- lastURL needs a hint to distinguish between filesystems
- discussed recommended practices for identifiers
- should the processing instruction be an "include" or an "exec"?
- should the "id" attribute be considered an identifier?? Should references to other albums and elements use the 'id', 'documentID', or 'instanceID' value?
- Is the order of the elements in MPV:Album important to high-perf processing?
- do we want to allow for multiple albums in an MPV file?? We could instead allow for multiple MPV documents in a single file. This seems likely anyway.

