MultiPhoto/Video Specification

Manifest, Metadata and Practices for Digital Photo-Video Collections

DRAFT
LOGO

Basic Profile and Presentation Profile

Revision 0.35
Working Draft

March 25, 2002

IMPORTANT NOTICE

This document is a working draft for review by OSTA and I3A members and approved parties. It is a draft document and will be updated, replaced, or obsoleted by other documents at any time and without notice. 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".

NOT FOR DISBTRIBUTION ON A PUBLIC WEBSITE

This document is available at http://www.osta.org/mpv/mpvmbrs/specs/MPV-Spec-Basic-Pres-0.35WD.PDF

POINTS OF CONTACT

OSTA David Bunzel OSTA President Tel: +1 (408) 253-3695 Email: dbunzel@osta.org http://www.osta.org *I3A* Lisa Walker I3A Co-Executive Director and Chief Marketing Officer Tel: +1 949-481-7645 Email: lisaw@i3a.org http://www.i3a.org	*MultiPhoto/Video Website* http://www.osta.org/mpv/index.htm *Technical Content* 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

The MultiPhoto/Video specification defines a manifest and metadata format and practices for processing and playback of collections of digital photo, video, and related audio and file content stored on an optical disc and other storage media such as memory cards and computer harddrives or exchanged via internet protocols.

COPYRIGHT NOTICE

LICENSING
IMPORTANT NOTICES

This document was developed by the Optical Storage Technology Association (OSTA) and International Imaging Industry Association (I3A). This document may be revised by OSTA. It is intended solely as a guide for companies interested in developing products which can be compatible with other products developed using this and closely-related documents. OSTA makes no representation or warranty regarding this document, and any company using this document shall do so at its sole risk, including specifically the risks that a product developed will not be compatible with any other product or that any particular performance will not be achieved. OSTA shall not be liable for any exemplary, incidental, proximate or consequential damages or expenses arising from the use of this document. This document defines only one approach to compatibility, and other approaches may be available in the industry.

This document is an authorized and approved publication of OSTA and I3A. The underlying information and materials contained herein are the exclusive property of OSTA and of the I3A as defined by a separate license but may be referred to and utilized by the general public for any legitimate purpose, particularly in the design and development of optical recording and reading systems and subsystems. This document may be copied in whole or in part provided that no revisions, alterations, or changes of any kind are made to the materials contained herein. Only OSTA has the right and authority to revise or change the material contained in this document, and any revisions by any party other than OSTA are totally unauthorized and specifically prohibited, except as specifically allowed by a license from the OSTA.

Compliance with this document may require use of one or more features covered by proprietary rights (such as features which are the subject of a patent, patent application, copyright, mask work right or trade secret right). By publication of this document, no position is taken by OSTA or I3A with respect to the validity or infringement of any patent or other proprietary right, whether owned by a Member or Associate of OSTA or otherwise. OSTA and I3A hereby expressly disclaim any liability for infringement of intellectual property rights of others by virtue of the use of this document. OSTA and I3A have not and do not investigate any notices or allegations of infringement prompted by publication of any OSTA or I3A document, nor does OSTA or I3A undertake a duty to advise users or potential users of OSTA and I3A documents of such notices or allegations. OSTA and I3A hereby expressly advises all users or potential users of this document to investigate and analyze any potential infringement situation, seek the advice of intellectual property counsel, and, if indicated, obtain a license under any applicable intellectual property right or take the necessary steps to avoid infringement of any intellectual property right. OSTA and I3A expressly disclaim any intent to promote infringement of any intellectual property right by virtue of the evolution, adoption, or publication of this document.

The information in this document is believed to be accurate as of the date of publication.
THIS DOCUMENT IS PROVIDED "AS IS." THE OPTICAL STORAGE TECHNOLOGY ASSOCIATION AND INTERNATIONAL IMAGING INDUSTRY ASSOCIATION MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO: WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THIS DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
THE OPTICAL STORAGE TECHNOLOGY ASSOCIATION AND INTERNATION IMAGING INDUSTRY ASSOCIATION WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

MultiPhoto/Video is a trademark of Optical Storage Technology Assocation, Inc. All other trademarks are the property of their respective owners. The names and/or trademarks of OSTA and I3A members may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission.

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.

Applications and devices and users that use MultiPhoto/Video benefit even when they only interact with still images in basic ways; when content like video clips and still sequences are added, as can be captured by a majority of the digital cameras introduced recently, the benefits expand.

MultiPhoto/Video 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:

- **Basic Profile**: key tasks: defining content collections, renditions, identifiers, and access to other metadata
- **Presentation 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

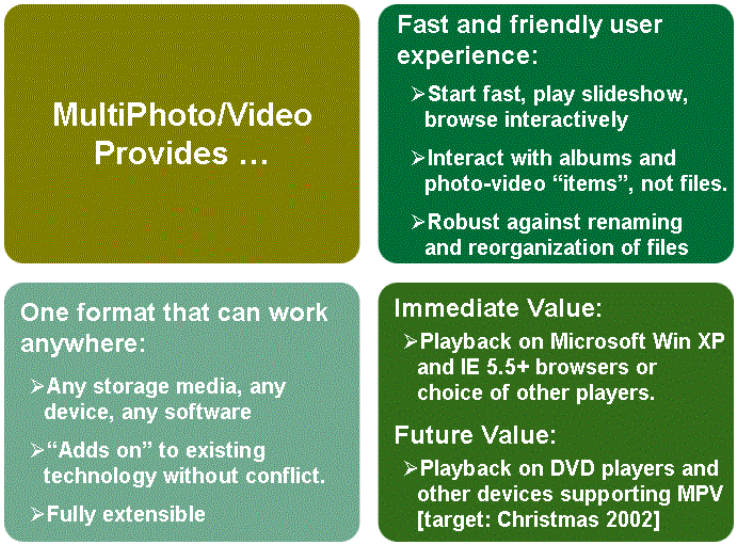
This document defines the Basic and Presentation Profiles of the MultiPhoto/Video specification, which may be implemented in consumer electronics devices, like DVD players, or application software to provide a firmware application that implements compelling playback of photo-video slideshows and interactive browsing of photo-video content.

MultiPhoto/Video technology has three central components: Collections, Metadata, and Identification. Each of these make reference in various ways to data files containing the photo-video content. This information is augmented with Presentation information that may be used by player applications and devices to provide an attractive user experience.

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 "ALBUM.MPV" and zero or more dependent files.

1.3 Terms of Use

This section of the specification is descriptive and not intended to be complete nor definitive. Please refer to the definitive statement of licensing terms at the beginning of the MultiPhoto/Video specification document for a precise and legal description.

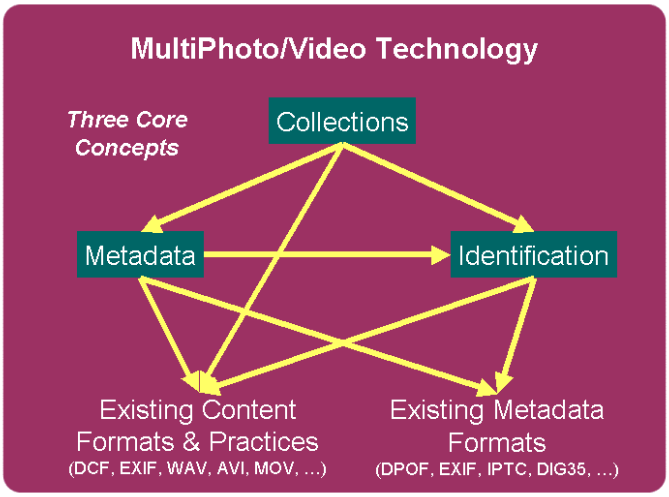
The MultiPhoto/Video specification is developed using an open process. The resulting specification is available at no or modest cost from OSTA and I3A. No royalty is charged by OSTA or I3A for use of the specification. The overall desire is to develop a specification that is not subject to separate licensing requirements or royalty. During the development process, the expectation is that all participants contribute their efforts and intellectual property without any expectation or requirement for compensation. However, OSTA and I3A does not warrant that the specification is not or will not be subject to such claims by other parties.

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.

Chapter 2: Key Concepts

MultiPhoto/Video has some key concepts anapproaches.

The Basic Profile has three core concepts centered on Collections, Metadata, and Identification. The Presentation Profile adds in Presentation information.



2.1 Collections

Collections are assembled using a few basic concepts.

ALBUM

A collection is an ordered group of objects called an Album. Albums can reference other albums. Multiple albums can be grouped together in one file or isolated in separate files. Album references use URIs, allowing reference to local or remote albums. Albums may have renditions and related documents.

FOREGROUND, BACKGROUND

Users interact with Album-level Foreground and Background objects; they and the Album's Related Documents are conceptually the primary objects in a collection. Typically, users interact most with foreground objects while background objects are secondary and fewer. Foreground and background objects may also contain additional content, including renditions and related documents. Additional content may enhance the performance, scope, presentation, and other characteristics of an album but do not fundamentally change it from a user's perspective.

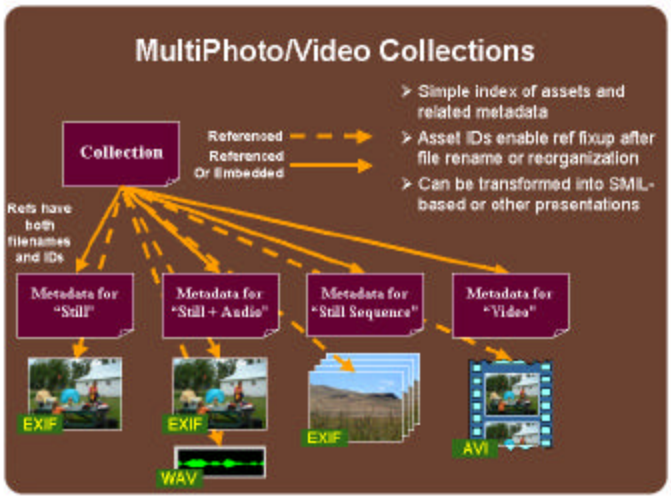
SIMPLE MEDIA OBJECTS

An album may contain the following types of media objects. MPV does not constrain which formats of these media objects may be in a collection. Simple media objects correspond to physical storage entities, i.e. files.

- AlbumRef
- Audio
- Document
- Still
- Print

- Text
- Video

Any media object may contain renditions and related documents.



COMPOSITE MEDIA OBJECTS

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

- StillWithAudio
- StillMultishotSequence
- StillPanoramaSequence
- Par
- Seq

Composite media objects may be album items, renditions, or related documents. The Seq and Par objects allow for arbitrary expression of other media objects but lack the direct association with the user's capture mode.

RENDITIONS

Any simple or composite media object and the album itself may have one or more renditions. Typically, original object is the master rendition and is usually defined implicitly. Renditions other than the master rendition are derived versions of the original media object. The relationship between the original rendition and the derived renditions 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.

RELATED DOCUMENTS

All simple and composite media objects and the album itself may have one or more related documents. Such documents may have any relation to the media object, including other objects used in constructing the object or additional metadata related to the object.

2.2 Metadata

MPV IS METADATA, NOT DATA

MPV provides metadata to describe photo/video object collections. It does not contain the actual asset data files themselves. The set of MPV metadata defines collections, identifiers, simple and composite objects, 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. This type of metadata may be embedded within an MPV document, even when it is not part of the MPV schema.

NAMESPACES AND NAMING EQUIVALENCE

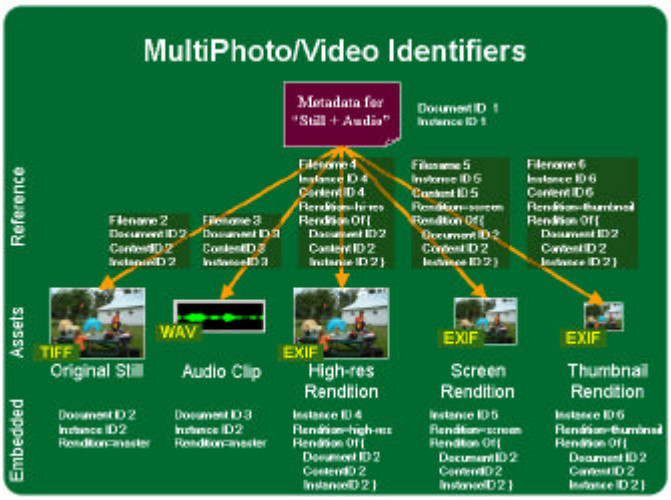
XML namespaces are a means to allow elements of the same name that exist in different schema to co-exist within the same document.

MPV requires that the MPV namespace prefixes on all elements and attributes be used in all XML encodings.

Some older existing XML-based applications and schema do not support namespaces. MPV can be encoded using a pseudo namespace by prefixing all elements and attributes with a defined namespace prefix separated with the underscore ("_"). Such an encoding is defined to enable the MPV specification to be used when namespaces are not supported; however, documents of this type are NOT well-formed MPV documents and need to be translated to use namespaces before they can be expected to interoperate with other MPV processors.

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:

- lastURL – last known location
- instanceID – unique identifier for a object that can also be used to reference to elements in an XML document
- documentID – the same for all renditions
- contentID – computed using the concent as input; statistically unique for each 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.

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.

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. The ability to fixup broken references is a key contribution that MPV makes to industry practices for representing collections.

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.

2.4 Presentation

The MPV Basic Profile defines how to represent collections. The MPV Presentation Profile defines how to present them.

USER TASKS

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 Presentation Profile extends the spec with very basic presentation information to enhance the user's experience.

PRESENTATION CONTROL

The overall approach for representing presentation information derives from SMIL, a powerful XML format for representing presentations from the World Wide Web Consortium (W3C). MPV Presentation 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 representation and also suggests a MPV playback strategy on platforms that also have SMIL players.

Because MPV also allows arbitrary metadata to be embedded or referenced, it is possible to embed additional presentation information in SMIL or other presentation languages. These may be used by players aware of these formats and practices.

XML LEVERAGE

MPV is well-formed XML. This allows the MPV collection document to be used with standard XML processing environments. For example, when opened in the Microsoft Internet Explorer 5.5 and above web browser, an MPV document with associated style sheet can present an attractive user interface for playback of MPV photo-video collections. Similarly, straight forward XSLT translation can convert an MPV document into a SMIL-based presentation for playback with an appropriate player.

2.5 Profiles and Modules, Schema and Practices

The MultiPhoto/Video specification is organized in the following ways.

Schema define the structure of MPV content, proviing a precise grammar and vocabulary of expression. MPV uses XML-Schema [XSCHEMA], a well-known schema definition language, to define this grammar and vocabulary in combination with prose descriptions to clarify usage and behaviour. A wide variety of commercial and open source tools support the use of XML Schema, including for schema design and schema and content validation.

In MPV, all schema are available in machine-readable form in addition to inclusion on a fragmentary basis within the specification document. The machine-readable schema in the normative definition; in the case of discrepancy, it supercedes the fragmentary descriptions in the specification document.

Practices define required and recommended behaviours in prose or pseudo code. Practices are a critical component to interoperability because they establish expectations and processes for how MPV content is handled.

Modules are a grouping of Schema and Practices and are the unit of design that provides a coherent set of capabilities. Modules are indivisible; they cannot be subdivided. Modules may be combined if designed to be compatible.

Profiles are a set of Modules and are the unit of formal specification, of specification implementation and of specification compliance. Products can implement or not implement profiles. Each profile in MultiPhoto/Video defines only those modules that are necessary for the key tasks targeted by the profile.

Chapter 3: Overall Required and Best Practices

The following required and best practices apply to all MPV content in all profiles unless explicitly stated otherwise.

3.1 Processing a MPV Document

A MPV document may be processed in any manner that complies with XML processing conventions and is consistent with the XML specification and the MPV XML Schema specifications. XML processing instructions shall be permitted; if the MPV processor cannot honor the processing instructions, they may be ignored.

3.2 Processing Unknown Elements and Attributes

Elements and attributes unknown to the MPV processor are allowed in MPV content; they may be any namespace. The MPV processor may choose how to handle them so long as general processing of the MPV document is not aborted.

The recommended practice is to ignore unknown attributes and to further decompose unknown elements. It is likely that unknown elements may contain content with known elements. In this case, it may be possible to provide for fallback processing or presentation in which the known elements are presented without the context of the containing and unknown element.

For example, a new composite type may be introduced, such as "AudioSequence". While this container is unknown, it contains Audio objects, which can be processed separately.

3.3 Namespace Usage

MPV requires that the namespace prefix is used on all elements and attributes in all XML encodings. Default namespace usage is not permitted. By convention, the namespace "mpv:" is used for the Basic Profile schema.

3.4 Pseudo-Namespace Prefixes

Some older existing XML-based applications and schema are incompatible with namespaces. MPV can be used in these cases by applying a pseudo namespace by replacing the ":" with an underscore "_" and using a well-specified prefix.

For example, all occurances of the "mpv:" and "mpvp:" namespace string fragment are replaced with the string "mpv_" and "mpvp_".

Such documents, however, are not MPV-compliant documents and must be translated to use namespaces before interoperability with other MPV-aware applications or devices can be expected. Nonetheless, better implementations of MPV content processors will accept pseudo-namespace MPV content.

3.5 Naming Conventions

MPV element names use UpperCamelCase, in which the leading character is uppercase. MPV attribute names use lowerCamelCase, in which the leading letter is lowercase.

3.6 Character Set

All MPV content shall use the UTF-8 character set [UTF-8]. Content is further constrained by XML allowable characters.

3.7 Allowable Characters

XML documents are encoded in text format and parsed; binary offsets are not used. This places constraints on the allowable characters of element and attribute names and values. In particular, string values need to be transformed on writing and reading to encode and decode disallowed characters.

3.8 Embedded Within an XML Packet

In most profiles, MPV content is anticipated to be held external to the photo-video assets that it references. The most common container for MPV content is expected to be the manifest. The manifest is a pure XML document.

MPV content may be embedded within other XML content. In this case, it may be embedded directly according to the design of the containing schema. Optionally, if allowed, it may be wrapped in an XML packet within the containing schema.

In the event that MPV content needs to be embedded in an arbitrary file, it should be wrapped in an XML packet. The following section was excerpted from [XMP-FW]. 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.

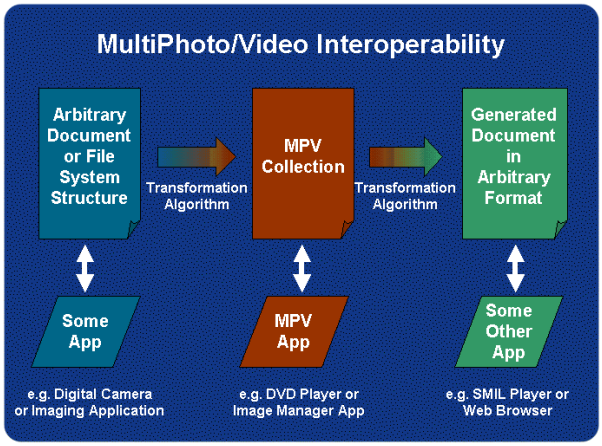
The XML Packet format was designed to accomplish the following:

- 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.

3.9 MPV Interoperability

MPV collections and documents exist within a complex and dynamic ecosystem of existing and new applications, devices, services, and formats. While MPV could be the primary format in which an application could store and represent not only photo-video collections but also to host its own data, this is not required. MPV can and very often will be used as an intermediate or derived format that provides for richer interoperability of applications, devices, services, and formats than is currently possible.



Through careful design, it is possible to overlay MPV collections onto existing arbitrary document and file system structures. This "add-on" behaviour does NOT require ANY changes to the original content to achieve useful and valuable results. A primary result is improved interoperability with other products that can accept MPV format as input representation, either natively or through a transformation step.

One advantage of MPV's use of industry-standard XML is that a diverse collection of commercial and open-source tools are available for use. A trend in the industry to better separate underlying data from the presentation of that data also reinforces the value and use of MPV and XML. For example, standard processing languages and tools such as XSLT can readily process and transform MPV content into arbitrary other formats.

For example, this approach underlies MPV's ability to be presented in existing applications, such as Microsoft's Internet Explorer 5.5 and above browser, which is also built into Windows XP. Using a straightforward style sheet, MPV can be transformed on the fly and rendered as an attractive slideshow within the IE browser.

3.10 MPV Extensions

Extensions can be made to MPV content in several forms. In each case, the extensions are gathered together and defined as a MPV Profile, which represents a set of schema and practices.

CUSTOM METADATA

Custom metadata is the preferred form of extension. At design time, a new set of metadata is defined that will be placed in the mpv:Metadata or mpv:VXMP container elements. The products that can produce and consume this metadata can now communicate using an in-place context-aware private communication channel hosted by the MPV document.

MPV SCHEMA ALTERATIONS

Using the power of XML Schema, alterations can be made to the MPV Schema definition. These alterations are specific to a profile. Because MPV processors are required to be tolerant of unknown attributes and elements, the altered content is able to interoperate with standard MPV-aware procesors, although at basic levels, while providing enhanced functionality with products aware of the extensions.

Chapter 4: MPV Basic Profile 1.0

The MPV Basic Profile 1.0 is designed to accomplish the following key tasks: defining collections of photo-video objects and related types of content including other media types and renditions, identifiers of those objects, and access to metadata.

The MPV Basic Profile 1.0 consists of the following modules and practices, which are specified in detail separately in this document.

- MPV Manifest Module Schema 1.0
- MPV Manifest Module Practices 1.0
- MPV VXMP Module Schema 1.0
- MPV VXMP Module Practices 1.0

The MPV Basic Profile is expected to be supported by most MPV-aware applications and devices and provides the basis for interoperability of collections across all range of storage media, devices, applications, and services.

Chapter 5: MPV Presentation Profile 1.0

The MPV Presentation Profile 1.0 is designed to accomplish the following key tasks: presentation of collections of photo-video objects defined using the Basic Profile, especially viewing a slideshow and interactively browsing the collection.

The MPV Presentation Profile 1.0 consists of the following modules and practices, which are specified in detail separately in this document.

- MPV Manifest Module Schema 1.0
- MPV Manifest Module Practices 1.0
- MPV VXMP Module Schema 1.0
- MPV VXMP Module Practices 1.0
- MPV Presentation Module Schema 1.0
- MPV Presentation Module Practices 1.0

The MPV Presentation Profile is expected to be supported by most MPV-aware applications and devices that present collections to users and provides the basis for interoperability of collections across all range of storage media, devices, applications, and services.

Note that the Presentation Profile is defined to be compatible with the Basic Profile by including the same modules in addition to the Presentation Module.

Chapter 6: MPV Manifest Module Schema

6.1 Module Introduction

An MPV manifest is a packaging mechanism enabling the grouping of MPV albums. In typical usage, a MPV manifest may contain only one album; when it contains multiple albums, the first is the default.

In typical usage, a MPV manifest is stored in a stand-alone file. Any appliation that produces or consumes MPV content stored in stand-alone files in a storage filesystem shall be compliant with the Manifest Module specification.

By implication of terminology, an MPV manifest contains reference to all the content that is relevant to a collection – it makes manifest the collection; it is a manifest of the collection.

6.2 Schema Information

The Manifest Module uses the MPV namespace. The introductory schema information is expressed as follows.

Schema group	Namespace Identifier	Conventional Namespace Prefix	Specified Prefix for Namespace-Incompatible Environments
Manifest	urn:osta-org:mpv:1.0	mpv:	mpv_

The XML Schema definition of the manifest begins by importing the MPV Album schema module. Thus the manifest schema explicitly includes also the Album schema.

Schema declaration:

```
<xs:schema targetNamespace="urn:osta-org:mpv:1.0" xmlns:mpv="urn:osta-org:mpv:1.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="qualified">

  <xs:import namespace="urn:osta-org:mpv:1.0" schemaLocation="album.xsd"/>
```


6.3 <mpv:mpv>

The top-level element of an MPV manifest is <mpv:mpv>. It may contain zero or more albums. The first album in the sequence is processed first and is the default album. A typical implementation will use this album to reference other albums, either in the same document or in other documents.

Note that no metadata or other information can be supplied at this outermost level; any such information should be embedded within an album. By implication, the manifest should be seen as a packaging mechanism without semantic value. The general assumption in MPV is that an album is self-standing and independent and that any context it requires can be extracted from content within the album. This content may, of course, make reference to data and other albums outside the album.

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


Chapter 7: MPV Manifest Module Practices

7.1 Finding an MPV Manifest File

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

When searching a file system for an MPV document, they can be located by name or by extension. When requested by name, the document is either found or not found. If not found, the algorithm defined elsewhere for lastURL fixup should be applied.

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. In this case, the document containing the MPV content must be specified by name, as it isn't practical to consider that all documents in a filesystem are candidate hosts of MPV content.

The MPV Manifest Module defines the following algorithm that describes how to locate an MPV document when no name of one is known.

```
If dealing with a removable storage unit, e.g. an optical disc inserted, the starting
current working directory is the root directory.

If dealing with a user's personal computer "login" account, there may be a set of
directories to be considered in sequence that will lead to the "root" MPV manifest for the
account.  Best Practices for which directories to consider are defined elsewhere.

If browsing a filesystem, the current working directory is decided by the application
conducting the search.

The scan algorithm to find a MPV manifest from a given current working directory is:

     In the current working directory, look for a file with one of the following
     case-insensitive names according to the order given.
          ALBUM.MPV
          MPVALBUM.XML
          INDEX.MPV
          <any name>.MPV, in an undefined order when more than one is present

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



```
     If no matching file is found, the parent and parent sibling directories of the
     current directory are scanned in an alphabetical breadth-first traversal to a
     height of two parent directories.

Files matching the pattern are processed in the order encountered.  When a MPV manifest
encountered, it is opened and scanned for an MPV Albu.  The first MPV Album encountered is
used.
```

The rationale behind this search algorithm is to first locate any top-level manifest containing MPV information, with a fallback of then finding named MPV manifests. 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 of 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.

N.B. By allowing the MPV manifest to carry the .XML extension or type, general purpose XML processors can operate on the MPV document and apply XML processing capabilities. For example, with Microsoft Internet Explorer 5.5 and above, an XML processing instruction in the MPV_TOC.XML file can invoke a style sheet that can transform the MPV document into an attractive browser-based presentation.

The search algorithm covers all of the following directories, where CWD is the current working directory. Naturally, when the path cannot be reached, it stops.

```
/P1
/P1/P2
/P1/P2b
/P1/P2/CWD
/P1/P2/CWD/C1
/P1/P2/CWD/C1b
/P1/P2/CWD/C1/C2
/P1/P2/CWD/C1b/C2
/P1/P2/CWD/C1b/C2b
/P1/P2/CWD/C1/C2/C3
/P1/P2/CWD/C1b/C2b/C3
```

But not these:

```
/P1b
/P1b/P2
/P1/P2b/D1
/P1/P2/CWD/C1/C2/C3/C4
```

In each of the directories scanned, the application shall search for all of the possible MPV manifest file names.

7.2 Manifest File Types

For systems in which file type is carried by the file name extension, such as Microsoft Windows and Unix, the MPV Manifest file will utilize an extension. The MPV Manifest Module defines two extensions a manifest may carry.

.mpv

This extension identifies a file to be a MPV manifest. Usage is case insensitive. This extension may be registered by an application to provide default and alternate processors of MPV manifests.

.xml

This extension identifies a file as containing XML content. Usage is case insensitive. A MPV manifest should only use this extension if it expects to be processed by a general-purpose XML processor such as Microsoft Internet Explorer. It is recommended that the manifest include an XML processing instruction specifying a stylesheet to use for presentation.

This extension may be registered by an application to provide general purpose XML content processing. An application should register this extension with care, as many types of content may carry the .xml extension and an application should do its best to handle this content in a general fashion.

For example, Microsoft Internet Explorer 5.5 and above registers this extension; when it processes the file, it looks for a stylesheet processing instruction. IE renders the results of applying the stylesheet to the XML content. This separation of content and presentation allows IE to be a general purpose XML processing engine and suitable for handling the .xml extension.

The Apple Macintosh operating system uses an internal file type stored as a resource value of the data fork of a file. The following file type may be used for MPV manifests on Macintosh systems. This file type has been registered with Apple, Inc., in accordance with recommended practices.

.mpv

This Apple Macintosh file type identifies the file to contain a MPV manifest. Usage is case sensitive. This extension may be registered by an application to provide a default processor of MPV manifests.

Some applications examine leading characters of a file in an attempt to determine its file type. No byte sequences can be counted on to always be present, generally all XML documents in the UTF-8 charsets begin with hexadecimal 3C 3F 78 6D 6C, ("<?xml"). While this will identify the document as an XML document, it does NOT identify it as an MPV manifest. This requires parsing the document to locate the outer element defined by the manifest schema.

7.3 Manifest MIME Media Type

MIME media types are widely used in internet applications to indicate the type of a file or content in a manner external of the file and independent of the name of the file or any information embedded in the file [MIME-2]. IANA maintains a registry of MIME media types and the set of MIME media types IANA thinks is registered at any time can be found at [MIMETYPES-REG].

The MIME media types that can be used for a MPV manfest are:

application/vnd.osta-org.mpv+xml

This MIME media type identifies the content to be a MPV manifest. Usage is case sensitive. This media type may be registered with internet browsers by an application to provide the default processor of a MPV manifest.

application/xml

This MIME media type identifies the content as containing XML content. Usage is case sensitive. A MPV manifest should only use this MIME type if it expects to be processed by a general-purpose XML processor such as Microsoft Internet Explorer. It is recommended that the manifest include an XML processing instruction specifying a stylesheet to use for presentation.

This MIME media type may be registered by an application to provide general purpose XML content processing. An application should register this media type with care, as many types of content may carry the application/xml media type and an application should do its best to handle this content in a general fashion.

For example, Microsoft Internet Explorer 5.5 and above registers this media type; when it processes the file, it looks for a stylesheet processing instruction. IE renders the results of applying the stylesheet to the XML

content. This separation of content and presentation allows IE to be a general purpose XML processing engine and suitable for handling the .xml extension.

7.4 Choosing Which File Type and MIME Media Type to Use

For products authoring MPV manifests, the choice of file extension and MIME media type is important. The product should consider the contexts in which it expects the manifest to be used. The primary decision factor is whether the product expects the manifest to be used in an environment that is explicitly MPV-aware or one that is not.

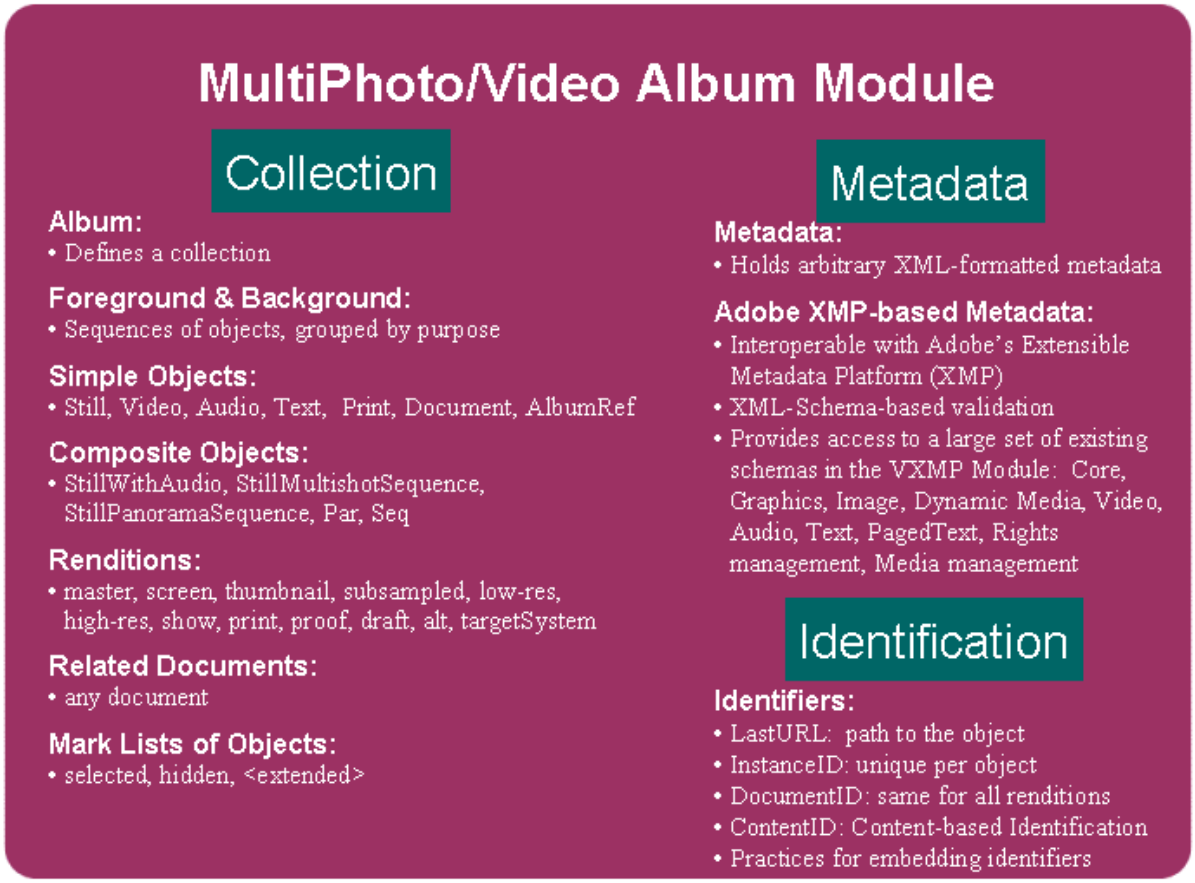
A MPV-aware environment will have the **.mpv** file extension and **application/vnd.osta-org.mpv+xml** media type registered to an application. A MPV-unaware environment will not.

Generally speaking, it is preferable to use a MPV manifest in an MPV-aware environment because the MPV-aware application is better able to utilize fully the MPV capabilities. In particular, an MPV-aware environment will likely handle better the situation in which the default lastURL reference in invalid; it should use other available lastURL values or the identifiers available on an object to fixup the lastURL value.

Chapter 8: MPV Album Module Schema

8.1 Module Introduction

The MultiPhoto/Video Album Module provides for the definition of collections of media objects. It is the essential core of the MPV specification. The Album modules has the following core components:



8.2 Schema Information

The XML Schema specification **[XSCHEMA]** defines the object-oriented grammar and basic types used here to define the MPV schema. Commercial and open source tools are available that can operate on schema defined using XML schema.

Schema group	Namespace Identifier	Conventional Namespace Prefix	Specified Prefix for Namespace-Incompatible Environments
Album	urn:osta-org:mpv:1.0	mpv:	mpv_

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 xmlns:mpv="urn:osta-org:mpv:1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="qualified">
```

8.3 Resource Identification

MPV collections are built using media objects. All media objects are based on these attributes and elements, which provide for four types of identifiers which can be used to reference other objects that compose the media object.

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:

- instanceID – unique identifier for every object. Also is an XML-style identifier for reference to elements in an XML document.
- documentID – the same for all renditions
- contentID – different for each rendition
- lastURL – last known location

More than one of most kinds of identifiers 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 contentIDs may be provided that utilize different computation algorithms with various tradeoffs of speed and robustness.

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 available widely 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.

8.3.1 Attributes: mpv:instanceID, mpv:documentID, mpv:contentID; Elements: <mpv:DocumentID>, <mpv:ContentID>

MPV objects have three types of computed identifiers: instanceIDs, documentIDs, and contentIDs. They may be specified via attributes or elements.

INSTANCE IDENTIFIERS

MPV uses this value to identify any referenced object, such as an image file. When practical and possible, the instanceID value used in an MPV document should be extracted from the referenced object as specified in the Appendix or according to the practices of metadata formats commonly used by that type of object. If not already present, the instanceID should be embedded in the referenced object in accordance with the identifier insertion conventions specified in the Appendix and by industry practice and other specifications. Whenever possible, the instanceID value should be the same value used to uniquely identify an obect using other metadata schema, such as Exif 2.2, XMP, or DIG35. An MPV element also uses this attribute to identify an XML fragment beginning with itself, allowing for easy reference in an XML document.

DOCUMENT IDENTIFIERS

DocumentIDs are an abstract concept: they remain constant across many versions and renditions of a given document. They are used to correlate relationship among separate things. MPV recommends that a DocumentID be a UUID, a 128-bit identifier which is readily generated by most modern operating systems. Sample source code for computing an arbitrary identifer is widely available and can be used for firmware implementations.

CONTENT IDENTIFIERS

Multiple ContentIDs may be provided that utilize different computation algorithms with various tradeoffs of speed and robustness. MPV recommends that two content identifiers be provided for robustness.

urn:osta-org:mpv:dsig:all:md5

urn:osta-org:mpv:dsig:body:md5

In particular, the body MD5 signature is recommended for JPEG images; this allows for JPEG metadata blocks to be edited without damaging the digital signature.

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. Note that the string value of MD5 identifiers does not use "-" separators, while the string value of UUID separators does.

Also note that composite objects also have an contentID. The algorithm to generate the signature is to concatentation the ordered list of sub-objects. For a StillWithAudio object, this would be done by streaming first the still image and then the audio through the algorithm. This allows one to have an contentID for both leaf and composite assets with almost no additional computational overhead if done right.

IDENTIFIER ATTRIBUTES SCHEMA

These are the basic attributes used in constructing resource identifiers and resource references.

```
<xs:attribute name="instanceID" type="xs:ID"/>
<xs:attribute name="documentID" type="xs:anyURI"/>
```



```
<xs:attribute name="contentID" type="xs:anyURI"/>
```

instanceID

MPV uses this value to identify any referenced object, such as an image file. An MPV element also uses this attribute to identify an XML fragment beginning with itself, allowing for easy reference in an XML document. The value syntax and practices for arriving at the instanceID are specified below.

documentID

An identifier that is the same for all renditions including the original, using the value syntax defined below.

contentID

An identifier that is different for each rendition, allowing the rendition to be uniquely identified, using the value syntax defined below.

None of the attributes is required.

IDENTIFIER ELEMENTS SCHEMA

DocumentID and ContentID may be specifed as zero or more elements. They have the same syntax as elements that they have as attributes.

```
<xs:element name="DocumentID" type="xs:string"/>
<xs:element name="ContentID" type="xs:string"/>
```

<mpv:DocumentID>

An identifier that is the same for all renditions including the original. Value syntax is the same as for the mpv:documentID attribute.

<mpv:ContentID>

An identifier that is different for each rendition, allowing the rendition to be uniquely identified. Value syntax is the same as for the mpv:contentID attribute.

INSTANCEID VALUE SYNTAX SPECIFICATION

MPV requires that all instanceID values be UUID-style unique ids encoded as 128-bit UUIDs in 32-hexcharacter string format, without hyphens ("-"). The value string syntax is in the form of a pure UUID value and is not URN qualified.

By requiring all instanceIDs to be unique, it is possible to merge MPV elements from multiple sources without fear of a collision of instanceIDs, making the process much simpler. It also enables references to the instanceID value to be sustained regardless of which collection a given item is in. UUIDs can be generated and converted to and from string format using established APIs in most operating systems or using available source code. MPV requires a specific string format; see the Appendix for further definition of the UUID string format.

DOCUMENTID AND CONTENTID VALUE SYNTAX SPECIFICATION

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.

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

urn:osta-org:mpv:uuid

The uuid is computed based on an algorithm said to generate close to unique numbers but not based on file content. This type of identifier can be used as a documentID. Example: `"urn:osta-org:mpv:uuid:EF886AEFA3B340da971BAF09B17DBC12"`

urn:osta-org:mpv:dsig:all:<algorithm>

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

urn:osta-org: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 contentID. Example: `"urn:osta-org:mpv:dsig:body:md5:EF886AEFA3B340da971BAF09B17DBC122"`

urn:osta-org: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. If unspecified, the default byte count is 8192. Example: `"urn:osta-org:mpv:dsig:head:30000:md5:EF886AEFA3B340da971BAF09B17DBC122"`

urn:osta-org: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. If unspecified, the default byte count is 8192. Example: `"urn:osta-org:mpv:dsig:tail:30000:md5:EF886AEFA3B340da971BAF09B17DBC122"`

8.3.2 Attributes: mpv:lastURL, mpv:byteOffset, mpv:xmlPacket, mpv:leaseID, mpv:leaseDur, mpv:leaseExpiresDate; Element: <mpv:LastURL>

Objects can be qualified according to a path to last known location. Hints can be given as to the byte offset within a file to find the specific data or whether the data is encapsulated in an XML packet. Use of lastURL, byteOffset, and xmlPacket enable applications to achieve rapid access to the data. The lastURL can be a local filename or remote URL. Multiple lastURLs may be provided to allow for a variety of possible locations or for different filenames in different file systems, such as on a CD.

However, lastURL, byteOffset,and xmlPacket are NOT robust references; they should be treated as useful hints. They may be broken by the user or an application renaming, reorganizing, or editing a file. The lastURL can be broken easily when a collection and assets are transferred across devices, storage formats and file systems. Applications that use lastURL, byteOffset, and xmlPacket should always have fallback schemes for the occasion when these hints fail to produce the desired data.

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. The ability to fixup broken references is a key contribution that MPV makes to industry practices for representing collections.

The concept of leases applies to URLs that have temporary lifetimes. This frequently occurs when MPV collections are constructed during dynamic processing operations and data exchanges. Leases are a separate concept from sessions; a session provides an authentication boundary to access, whereas a lease provides a temporal boundary to access that may span many sessions. For example, when a collection is created as part of an website shopping cart and cached on a client, the lifetime of the collection's URLs may be longer than a particular web session.

MOTIVATION

A particular need for multiple lastURLs is found when MPV collections are used on data CDs. Data CDs typically have several co-existing file systems with differing abilities to represent long filenames and filenames with international characters. Each device and operating system chooses 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 or international characters that is placed on a CD can be broken if the player device uses a different file system that doesn't support these 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.

SCHEMA

These are the basic attributes used in constructing resource identifiers and resource references.

```
<xs:attribute name="lastURL" type="xs:anyURI"/>
<xs:attribute name="byteOffset" type="xs:integer"/>
<xs:attribute name="xmlPacket" type="xs:integer"/>
<xs:attribute name="leaseID" type="xs:string"/>
<xs:attribute name="leaseDur" type="xs:integer"/>
<xs:attribute name="leaseExpiresDate" type="xs:date"/>
```

lastURL

The last known location can be a local filename or remote URL. The mpv:lastURL attribute is optional. In addition, zero or more <mpv:LastURL> elements may be specified. The recommended use of all lastURL attribute and elements present is to try them in the order specified. More information on the syntax of the lastURL attribute value is in the specification for the <mpv:LastURL>element.

byteOffset

Indicates a byte offset into the referenced file. 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 client receives a byte stream beginning at the offset.

xmlPacket

Indicates the information in the referenced file is encapsulated in the Nth XML packet in the file. This argument may be used even when lastURL refers to a local file. 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 client receives a byte stream beginning at the packet. The value "0" means the referenced information is contained in an XML packet but it isn't known which one.

leaseID

Identifies the lease associated with this URL

leaseDur

Identifies the duration in seconds since the time the lease was created that the URL will remain valid. This is the recommended value to be used with short-duration collections. When this attribute is unspecified, the assumption is that the lastURL is valid indefinitely.

leaseExpiresDate

Identifies the approximate date and time that the URL will expire. The value is approximate because it is unspecified whether this date was provided by the URL server or URL client and it is also unknown whether the system times of the client or server was correct when the expiration date was determined. When this attribute is unspecified, the assumption is that the lastURL is valid indefinitely.

Multiple lastURLs may be provided to allow for different filenames in different file systems, such as on a CD.

Several identifiers can also be specified as elements. <mpv:DocumentID>, <mpv:ContentID>, and <mpv:LastURL> has the same syntax as elements that they have as attributes.

<mpv:LastURL>

The last known location can be a local filename or remote URL. Zero or more <mpv:LastURL> elements may be specified in addition to an optional lastURL attribute on an element. The recommended use of all lastURL attribute and elements present is to try them in the order specified.

```
<xs:element name="LastURL">
 <xs:complexType>
  <xs:simpleContent>
   <xs:extension base="xs:anyURI">
    <xs:attribute name="hint" type="xs:anyURI"/>
    <xs:attribute name="filesystem" type="mpv:FilesystemType"/>
    <xs:attribute ref="mpv:byteOffset"/>
    <xs:attribute ref="mpv:xmlPacket"/>
    <xs:attribute ref="mpv:leaseID"/>
    <xs:attribute ref="mpv:leaseDur"/>
    <xs:attribute ref="mpv:leaseExpiresDate"/>
   </xs:extension>
  </xs:simpleContent>
 </xs:complexType>
</xs:element>

<xs:simpleType name="FilesystemBaseType">
 <xs:restriction base="xs:string">
  <xs:enumeration value="URI"/>
  <xs:enumeration value="ISO9660-1"/>
  <xs:enumeration value="ISO9660-2"/>
  <xs:enumeration value="ISO9660-3"/>
  <xs:enumeration value="HFS"/>
  <xs:enumeration value="Joliet"/>
  <xs:enumeration value="UDF15"/>
  <xs:enumeration value="RockRidge"/>
  <xs:enumeration value="FAT16"/>
  <xs:enumeration value="FAT32"/>
  <xs:enumeration value="NTFS"/>
  <xs:enumeration value="Windows"/>
  <xs:enumeration value="Unix"/>
 </xs:restriction>
</xs:simpleType>
<xs:simpleType name="FilesystemType">
 <xs:union memberTypes="mpv:FilesystemBaseType xs:string"/>
</xs:simpleType>
```

filesystem

A hint about the intended use or origin of the lastURL value. The following basic vocabulary is defined and refers to file systems.

- "URI" – compliant with URI naming conventions
- "ISO9660-1" – 8.3 file and directory names compliant with ISO 9660-1 CD file system
- "ISO9660-2" – 32 char file and directory names compliant with ISO 9660-2 CD file system
- "ISO9660-3" – file and directory names compliant with ISO 9660-3 CD file system
- "HFS" – 32 char file and directory names compliant with Macintosh HFS CD file system
- "Joliet" – UTF-8 encoding of the 64 character Unicode UCS-2 file and directory names compliant with Joliet CD file system

- "UDF15" – file and directory names compliant with UDF1.5 file system
- "RockRidge" – file and directory names compliant with RockRidge CD file system
- "FAT16" – 8.3 file and directory names compliant with Microsoft Windows FAT16 conventions. When a FAT16 file or directory has a dual long file name, it should be encoded as a separate LastURL value with the FAT32 filesystem type.
- "FAT32" – UTF-8 encoding of Unicode UCS-2 file and directory names compliant with Microsoft Windows FAT32 conventions. When a FAT16 file or directory has a dual long file name, it should be encoded as a separate LastURL value with the FAT32 filesystem type.
- "NTFS" – UTF-8 encoding of Unicode UCS-2 file and directory names compliant with Microsoft Windows NTFS conventions
- "Windows" – UTF-8 encoding of file and directory names compliant with an unspecified type of Microsoft Windows-based file system. Should only be used when FAT16, FAT32, and NTFS cannot be determined.
- "Unix" – UTF-8 encoding of file and directory names compliant with Unix conventions

hint

Indicates the hint associated with the intended use or origins of the lastURL. It is recommended that hints use URN-style qualified names to avoid possible name collisions, such as "urn:osta-org:mpv:original".

LASTURL SYNTAX AND ARGUMENTS DEFINITION

The value of <mpv:LastURL> element or the mpv:lastURL attribute 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 cse-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 using LastURL to open a local file may need to remove the arguments before using the lastURL, depending on the operatng system and APIs used.

Arguments may be placed on the lastURL value; argument names are case sensitive. When placed on the lastURL, the syntax is as follows:

lastURL#<mpv_instanceID value>?mpv_documentID=<value>&mpv_contentID=<value>
&mpv_filesystem=<value>&mpv_hint=<value>
&mpv_byteOffset=<value>&mpv_xmlPacket=1
&mpv_leaseID=<value>&mpv_leaseDur=<value>&mpv_leaseExpiresDate=<value>

The following arguments are defined by MPV for lastURL:

<mpv_instanceID value>

Indicates the fragment id in the referenced document. This will be a 32-character UUID string, without hyphens ("-").

mpv_documentID, mpv_contentID

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.

mpv_filesystem

Indicates the file system associated with the lastURL.

mpv_hint

Indicates the hint associated with the intended use or origins of the lastURL. It is recommended that hints use URN-style qualified names to avoid possible name collisions, such as "urn:osta-org:mpv:original".

mpv_byteOffset

Indicates a byte offset into the referenced file, such as "`lastURL?mpv_byteOffset=3342`".

mpv_leaseID

The leaseID of the URL

mpv_leaseDur

The lease duration of the URL

mpv_leaseExpiresDate

The lease expiration date of the URL

mpv_xmlPacket

Indicates the information in the referenced file is encapsulated in the Nth XML packet., such as "`lastURL?mpv_xmlPacket=3`".

8.4 Base Types

MPV is defined using a small set of base groups and types. They are as follows:

- **ElemIdAttrGroup, ElemIdElemGroup**: Provides for the basis for associating identification and metadata with most MPV elements in the collection.
- **ResourceIdAttrGroup, ResourceIdElemGroup**: Provides for the basis for associating robust identification and metadata with composite objects in the collection that do not have equivalents in discrete files.
- **ResourceFileAttrGroup, ResourceFileElemGroup**: Provides for the basis for associating robust identification and metadata with objects in the collection that operate as proxies for discrete files with associated data.
- **SimpleObjectBaseType, CompositeObjectBaseType**: Provides the base types for all MPV media objects to inherit from.
- **ObjectsListType**: Provides the set of known objects to choose among.

8.4.1 Groups: ElemIdAttrGroup, ElemIdElemGroup

MPV defines a number of XML schema groups which are used throughout the specification's grammar. These important groups define the set of available subelements and attributes of many of the elements defined by MPV.

An element that can be identified and referenced as the id value in the URI syntax of "transport:path#id?arguments" must include the mpv:id attribute. This group is the attribute group for elements to include.

```
<xs:attributeGroup name="ElemIdAttrGroup">
  <xs:attribute ref="mpv:instanceID"/>
</xs:attributeGroup>
```

An element in MPV can always be described using either XMP-compliant or any arbitrary kind of metadata. This element group provides for these subelements.

```
<xs:modelGroup name="ElemIdElemGroup"/>
```



```
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="mpv:VXMP" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element ref="mpv:Metadata" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:modelGroup>
```

8.4.2 Groups: ResourceIdAttrGroup, ResourceIdElemGroup

An element that has its own identity in an MPV sense always has at least three attributes: instanceID, documentID, and contentID. A resource uses these attributes to describe computed identifiers for itself and associated content. This group provides these attributes.

```
 <xs:attributeGroup name="ResourceIdAttrGroup">
  <xs:attributeGroup ref="mpv:ElemIdAttrGroup"/>
  <xs:attribute ref="mpv:documentID"/>
  <xs:attribute ref="mpv:contentID"/>
 </xs:attributeGroup>
```

An element that has its own identity can specify that identity via attributes or subelements. This group defines the subelements. Note that the "instanceID" attribute can only be specified as an attribute, unlike documentID and contentID, which can be specified as either or both attributes or subelements. Only one mpv:documentID or mpv:contentID attribute may be specified whereas many mpv:DocumentID and mpv:ContentID elements may occur. There is no significance to the order or location of appearance.

```
 <xs:group name="ResourceIdElemGroup">
        <xs:sequence>
            <xs:element ref="mpv:DocumentID" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="mpv:ContentID" minOccurs="0" maxOccurs="unbounded"/>
            <xs:group ref="mpv:ElemIdElemGroup"/>
        </xs:sequence>
</xs:group>
```

8.4.3 Groups: ResourceFileAttrGroup, ResourceFileElemGroup

An element that is a proxy for an external resource can identify it not only with identifiers but also a lastURL address. This group defines these attributes.

```
<xs:attributeGroup name="ResourceFileAttrGroup">
        <xs:attributeGroup ref="mpv:ResourceIdAttrGroup"/>
        <xs:attribute ref="mpv:lastURL"/>
        <xs:attribute ref="mpv:byteOffset"/>
        <xs:attribute ref="mpv:xmlPacket"/>
        <xs:attribute ref="mpv:leaseID"/>
        <xs:attribute ref="mpv:leaseDur"/>
        <xs:attribute ref="mpv:leaseExpiresDate"/>
</xs:attributeGroup>
```

An element that is a proxy for an external resource can identify itself not only with identifiers but also a LastURL address. This group defines these subelements. Note that the "instanceID" attribute can only be specified as an attribute, unlike documentID and contentID, which can be specified as either or both attributes or subelements. Only one mpv:documentID or mpv:contentID attribute may be specified whereas many mpv:DocumentID and mpv:ContentID elements may occur. There is no significance to the order or location of appearance. Note that xmlPacket and byteOffset are not allowed as subelements directly. Instead, if they are to be specified, they must be placed as attributes or arguments on the value of LastURL.


```
<xs:group name="ResourceFileElemGroup">
        <xs:sequence>
            <xs:element ref="mpv:DocumentID" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="mpv:ContentID" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="mpv:LastURL" minOccurs="0" maxOccurs="unbounded"/>
            <xs:elementGroup ref="mpv:ElemIdElemGroup"/>
        </xs:sequence>
</xs:group>
```

A resource uses these attributes to describe itself and associated content in XML format has the additional attribute and subelement that specifies that the data may be found in the Nth xml packet contained by the resource. This is a hint – the data may be present in the Mth xml packet – it should still be locatable using the standard XML packet scanning algorithm.

8.4.4 Types: SimpleObjectBaseType, CompositeObjectBaseType

These types are the base type of all media objects. They are not used directly as elements.

Simple objects are proxies to external files or resources, and this base type consists of attributes and elements that identify an external file. The identity values of a simple object are the same as identity values of the referenced file or resource.

```
  <xs:complexType name="SimpleObjectBaseType">
        <xs:sequence>
            <xs:group ref="mpv:ResourceFileElemGroup" minOccurs="0" maxOccurs="1"/>
            <xs:element ref="mpv:Related" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="mpv:Rendition" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attributeGroup ref="mpv:ResourceFileAttrGroup"/>
    </xs:complexType>
```

Composite objects define a new object that has it own identity distinct from the identity of its components; it is composed of other objects.

```
   <xs:complexType name="CompositeObjectBaseType">
        <xs:sequence>
            <xs:group ref="mpv:ResourceIdElemGroup" minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
        <xs:attributeGroup ref="mpv:ResourceIdAttrGroup"/>
    </xs:complexType>
```

8.4.5 Group: ObjectsListType

The MPV specification defines a collection of media objects The ObjectsList defines the set of available media objects in MPV. This is not used directly as an element.

```
<xs:group name="ObjectsList" type="mpv:ObjectsListType"/>
<xs:complexType name="ObjectsListType">
   <xs:choice>
     <xs:element ref="mpv:AlbumRef"/>
     <xs:element ref="mpv:Audio"/>
     <xs:element ref="mpv:Document"/>
     <xs:element ref="mpv:Still"/>
     <xs:element ref="mpv:StillWithAudio"/>
     <xs:element ref="mpv:StillMultishotSequence"/>
     <xs:element ref="mpv:StillPanoramaSequence"/>
```



```
      <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>
```

8.5 <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.

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

Arbitrary data can be attached to a collection, but it carries no explicit semantics with respect to MPV collection processing other than the data is associated with the collection. The Data element can make reference to data files associated with the Album but which are not media objects or are not Foreground or Background album items. For example, a file such as the DPOF AUTPRINT.MRK datafile placed in the DCF /MISC directory could be referenced using the Data element.

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.

MarkLists of an album represent lists of album items that are distinguished in some manner. Two marklist types are defined for selected and hidden album items.

```
<xs:element name="Album" type="mpv:AlbumType"/>
  <xs:complexType name="AlbumType">
      <xs:complexContent>
          <xs:extension base="mpv:CompositeObjectBaseType">
              <xs:sequence>
                  <xs:element ref="mpv:Background" minOccurs="0" maxOccurs="1"/>
                  <xs:element ref="mpv:Foreground" minOccurs="0" maxOccurs="1"/>
                  <xs:element ref="mpv:Related" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="mpv:Rendition" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="mpv:MarkList" minOccurs="0" maxOccurs="unbounded"/>
              </xs:sequence>
          </xs:extension>
      </xs:complexContent>
  </xs:complexType>
```

8.6 <mpv:VXMP>

The <mpv:VXMP> element wraps a syntactic subset of the metadata schemas defined by the Adobe Extensible Metadata Platform (XMP) Framework ([XMP-FW]. Using the <mpv:VXMP> element along with the appropriate VXMP schema is the preferred means in MPV to specify a wide variety of useful properties about albums and album items. MPV processors supporting the MPV Album Module are required to process the set of XMP-supplied

schemas (see below) that apply to their domains, allowing for a rich exchange of data between MPV-aware applications.

VXMP stands for Validating XMP, an encoding of XMP that uses XML Schema and can be validated using standard XML Schema validation tools. Validation is an important part of the practices and compliance tests that to facilitate MPV content interoperability. The <mpv:VXMP> element also utilizes a shortcut notation that makes it less verbose to use XMP schemas while still being completely compatible with the semantics of XMP. The <mpv:VXMP> element constrains the RDF-based syntax of XMP in such a way that it can be fully specified using an XML Schema. The rules for mapping the XMP syntax to and from MPV are described in an Appendix. This mapping can be applied mechanically, allowing for interoperability with XMP-aware applications and formats.

Complete XMP metadata in native format can be encoded in the MPV Album Module using the mpv:Metadata element; however, MPV processors are not required to support extraction and processing of any data contained by mpv:Metadata, so XMP data provided in this manner is less interoperabile with MPV applications.

```
<xs:element name="XMP" type="mpv:VXMPType"/>
 <xs:complexType name="XMPType" mixed="true">
  <xs:sequence>
   <xs:Any namespace="##any" processContents="lax" minOccurs="0"/>
  </xs:sequence>
  <xs:attributeGroup ref="mpv:ElemIdAttrGroup"/>
 </xs:complexType>
```

Example of VXMP metadata:

```
<mpv:VXMP>
  <xap:xap>
    <xap:Author>Pieter van Zee</xap:Author>
    <xap:CreateDate>2002-03-25T21:07:00Z</xap:CreateDate>
  </xap:xap>
</mpv:VXMP>
```

MPV ALBUM MODULE MPV:VXMP PRACTICES

For broadest interoperability with consumer and professional photo-video applications, this specification recommends that MPV applications preferentially produce and process metadata conforming to the XMP specification **[XMP-FW]**. This open and royalty-free specification, developed by Adobe Inc., defines schema and practices for metadata that are being broadly implemented across Adobe's product lines. Support for XMP also is appearing in many products that interoperate with Adobe products and data formats.

XMP provides a simple, modular schema for the following basic sets of useful metadata:

- Core properties: Author, Authors, CreateDate, CreatorTool, Description, Format, Keywords, Locale, ModifyDate, Nickname, Title
- Graphics: ColorSpace, Compression, GraphicsType, NaturalDimensions, NumberOfColors, NumberOfInks
- Image: Dimensions, Resolution
- Dynamic Media: Duration, NTracks, Tracks
- Video: BitRate, Dimensions, Interleaved, NaturalRate, Compression, Encoding
- Audio: ChannelCount, Compression, Rate, SampleSize, Volume
- Text: Encoding, FontList
- PagedText: MaxPageSize, NPages
- Rights management: Certificate, Copyright, Owner, UsageTerms
- Media management: ContainedResources, ContributorResources, DocumentID, History, LastURL, Manager, ManageTo, RenditionClass, RenditionOf, SaveID, VersionID, Versions

XMP does not itself provide great detail in any given area, but it provides sufficient information for many applications to make useful processing and presentation decisions. For detailed metadata, MPV and XMP refer to specialized metadata formats, such as Exif, DIG35, and IPTC.

MPV expects that metadata specified within an MPV file may be duplicated by metadata embedded in files that are refererenced. For example, the XMP Core schema element xap:Title is a duplicate of the title field stored in an Exif file. By convention, the properties stored in the MPV file are considered to take precedence over any similar properties stored in a separate data object. This approach provides unambiguous definition of which property to use in case of overlap, and allows a single referenced file to be utilized and represented in a variety of ways.

8.7 <mpv:Metadata>

The <mpv:Metadata> element "tunnels" well-formed XML content into a MPV document. The metadata element provides an open-ended low-cost means to specify additional metadata that is embedded within the MPV document for ready reference. To reference metadata that is external to the MPV document, use the <mpv:Document> element. A typical occurance of such data is to embed useful metadata, such as that defined by DIG35 **[DIG35]** or XMP data in native form **[XMP-FW]**, in the MPV document.

```
<xs:element name="Metadata" type="mpv:MetadataType"/>
<xs:complexType name="MetadataType" mixed="true">
  <xs:sequence>
    <xs:Any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
  <xs:attributeGroup ref="mpv:ElemIdAttrGroup"/>
  <xs:attribute name="schemaURI" type="xs:anyURI"/>
</xs:complexType>
```

Example of embedded native XMP metadata:

```
<mpv:Metadata mpv:schemaURI="adobe:ns:meta/"
              xmlns:x="adobe:ns:meta/">
  <x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="XMPTk 2.8">
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
      <rdf:Description about="" xmlns:xmp="http://ns.adobe.com/xap/1.0/"
              xmp:Author="Pieter van Zee" xmp:CreateDate="2002-03-25T21:07:00Z">
      </rdf:Description>
    </rdf:RDF>
  </x:xmpmeta>
</mpv:Metadata>
```

Example of embedded DIG35 metadata:

```
<mpv:Metadata mpv:schemaURI="http://www.digitalimaging.org/dig35/1.1/xml"
              xmlns="http://www.digitalimaging.org/dig35/1.1/xml">
<METADATA>
  <GENERAL_CREATION_INFO>
   <CREATION_TIME>2002-03-25T21:07:00</CREATION_TIME>
    <IMAGE_CREATOR>
      <PERSON_NAME>
        <NAME_COMP TYPE="Given">Pieter</NAME_COMP>
        <NAME_COMP TYPE="Family">van Zee</NAME_COMP>
      </PERSON_NAME>
    </IMAGE_CREATOR>
  </GENERAL_CREATION_INFO>
</METADATA>
</mpv:Metadata>
```


Example of embedded Dublin Core metadata:

```
<mpv:Metadata mpv:schemaURI="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
              xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description about="" xmlns:dc="http://purl.org/dc/elements/1.1/">
    <dc:creator>
      <rdf:Bag>
        <rdf:li>Pieter van Zee</rdf:li>
     </rdf:Bag>
     </dc:creator>
    <dc:date>
      <rdf:Seq>
        <rdf:li>3/25/2002 21:07:00</rdf:li>
     </rdf:Seq>
     </dc:date>
   </rdf:Description>
</rdf:RDF>
</mpv:Metadata>
```

8.8 <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. Additionally, objects may be organized into the generic RelatedDocuments group, which carries no specific semantics.

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

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

8.9 <mpv:Related>

Related is a generic container which carries no specific semantics other than being related to the object that contains it. It may contain any number of media objects.

```
<xs:element name="Related" type="mpv:RelatedType"/>
  <xs:complexType name="RelatedType">
        <xs:complexContent>
            <xs:extension base="mpv:CompositeObjectBaseType">
                <xs:sequence>
                    <xs:group ref="mpv:ObjectsList" minOccurs="0" maxOccurs="unbounded"/>
                    <xs:element ref="mpv:Rendition" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="hint" type="xs:string"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
```


hint

The hint applies to the related item and has an open vocabulary. Hints values must use URN-qualified names to avoid the possibility of name collisions, such as "urn:mycompany-com:mpv:somerelation".

8.10 <mpv:Rendition>

A rendition is a derivative of a media object. Renditions should have the same "documentID" as the parent media object but difference contentIDs. 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:CompositeObjectBaseType">
            <xs:sequence>
                <xs:group ref="mpv:ObjectsList" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="mpv:Related" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="renditionUsage" type="xs:string"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

renditionUsage

The vocabulary is an open vocabulary with the following initial values derived from the first field of the renditionClass attribute in Adobe's XMP specification and extended for MPV. Additional fields of the XMP renditionClass are specified as properties or metadata of the Rendition itself. Additional vocabulary values must use URN-qualified names to avoid the possibility of name collisions, such as "urn:mycompany-com:mpv:somerendition".

master

Indicates the master document(s). This is the equivalent of XMP's **default** class.

thumbnail

For a simplified and/or reduced preview of a version.

screen

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

subsampled

A subsampled resolution rendition. Has different resolution than default. Thumbnail and screen are subsampled renditionClasses with a specific purpose, typically for still images.

low-res

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

high-res

For a high quality, full size stand-in, but not the master. Typically compressed with respect to the master. Has the same resolution as default/master.

proof

For a review proof

draft

For a review rendition

print

Indicates a rendition of the content formatted for the printed page and ready for printing.

show

Indicates a rendition of the content formatted for presentation. This is most useful for composite objects (including: Album, StillWithAudio, StillMultishotSequence, StillPanoramaSequence, Par, Seq) that may offer a presentation-oriened rendition.

targetSystem

A rendition targetting a specific set of system characteristics. The value of "targetSystem" is qualified using colon (:) separated attribute=value pairs. The attribute and value vocabulary is provided by the System Test attributes and values of the SMIL 2.0 BasicContentControl specification [SMIL20].

Example:

"targetSystem:systemBitrate=28800"

"targetSystem:systemScreenSize=768X1024"

alt

An alternate rendition of the master document or documents that represents a rendering or version that is distinguished in some manner that cannot be described with other renditionUsage vocabulary. Further qualifications of the 'alt' name are reserved for use by all MPV profiles that may define additional particular renditions, such as "alt:subtitles".

8.11 <mpv:MarkList>

There are many situations where a subset of the Album items needs to be identified. Examples include a subset that is marked for downstream action like printing or e-mail. In addition, interactive profiles need to provide the user with the ability to add and remove album items from a selected set. MPV provides the MarkList as a general facility for dealing with these types of requirements. Marked items can be in any of the Background, Foreground, RelatedDocuments or Rendition containers.

```
<xs:element name="MarkList" type="mpv:MarkListType"/>
 <xs:complexType name="MarkListType">
    <xs:sequence>
      <xs:element name="idref" type="xs:ID"  minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attributeGroup ref="mpv:ElemIdAttrGroup"/>
    <xs:attribute name="markType" type="xs:string"/>
 </xs:complexType>
```

markType

The type of mark to apply to all the referenced items. The markType has an open vocabulary with the following initial values. It is recommended that new types use URN-qualified names to avoid the possibility of name collisions.

"urn:osta-org:mpv:selected"

The list of items in the album that the user has selected. The concept of selected is that the user knows the item is selected and will expect certain types of processing operations to operate on the set of selected items.

"urn:osta-org:mpv:hidden"

The list of items in the album that the user has hidden. The concept of hidden is that the user knows the item is in the collection and wants it to be, but generally doesn't want it displayed or otherwise processed. Hidden items can be unhidden, and so are maintained in the collection in the order given and over time. When inserting a new item, the insertion point is always in front of any hidden items that exist between the previous and next visible items. Hidden items are processed when necessary to preserve the state of the collection, such as a Save operation.

idref

The identifier of the album item that is marked. MPV specifies a best practice for the identifier to be a UUID value so that album items can be merged from various albums without fear of id collision.

By convention, if a processing application discovers that the marklist references an item that is not contained in the album, the item reference may be removed. This is considered a means of informal garbage collection.

8.12 <mpv:AlbumRef>

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

```
<xs:element name="AlbumRef" type="mpv:SimpleObjectBaseType"/>
```

8.13 <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:SimpleObjectBaseType"/>
```

8.14 <mpv:Still>

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

```
<xs:element name="Still" type="mpv:SimpleObjectBaseType"/>
```

8.15 <mpv:StillMultishotSequence>

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

```
<xs:element name="StillMultishotSequence" type="mpv:StillMultishotSequenceType"/>
<xs:complexType name="StillMultishotSequenceType">
        <xs:complexContent>
            <xs:extension base="mpv:CompositeObjectBaseType">
```



```
            <xs:sequence>
                <xs:element ref="mpv:Still" minOccurs="1" maxOccurs="unbounded"/>
                <xs:element name="mpv:CaptureDur" type="xs:string" minOccurs="0" maxOccurs="1"/>
                <xs:element ref="mpv:Related" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="mpv:Rendition" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

CaptureDur

The value of captureDur is a sequence of still-to-still 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 is always in relative time to the previous frame.

There are as many as N-1 clock values for N images. The last value provided is reused for all subsequent durations.

Example:
"FrameToFrame:0.3": any number of still images, each 0.3 seconds after the previous.

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

"FrameToFrame:120;210;70": 4 images, the second taken 120 seconds after the first, the third taken 210 seconds after the second, the fourth taken 70 seconds after the third.

8.16 <mpv:StillPanoramaSequence>

The StillPanoramaSequence 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="StillPanoramaSequence" type="mpv:StillPanoramaSequenceType"/>
<xs:complexType name="StillPanoramaSequenceType">
    <xs:complexContent>
        <xs:extension base="mpv:CompositeObjectBaseType">
            <xs:sequence>
                <xs:element ref="mpv:Still" minOccurs="1" maxOccurs="unbounded"/>
                <xs:element name="mpv:CapturePath" type="xs:string" minOccurs="0" maxOccurs="1"/>
                <xs:element ref="mpv:Related" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="mpv:Rendition" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```


CapturePath

The value of capturePath is a sequence of still image-to-image motions that indicate the path. The semicolon 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)*`
" There are as many as N-1 motions for N images. The last value provided is reused for all subsequent durations.

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:
"`FixedPt:270Y0P0R`": any number of still images, each one rotating to the left of the previous.

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

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

8.17 <mpv:StillWithAudio>

The StillWithAudio element groups a still image object with one or more audio objects. Typical renditions of the image object would be thumbnail and screen resolutions of the image.

```
<xs:element name="StillWithAudio" type="mpv:StillWithAudioType"/>
<xs:complexType name="StillWithAudioType">
    <xs:complexContent>
        <xs:extension base="mpv:CompositeObjectBaseType">
            <xs:sequence>
                <xs:element ref="mpv:Still" minOccurs="1" maxOccurs="1"/>
                <xs:element ref="mpv:Audio" minOccurs="1" maxOccurs="unbounded"/>
                <xs:element name="mpv:CaptureDur" type="xs:string" minOccurs="0" maxOccurs="1"/>
                <xs:element ref="mpv:Related" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="mpv:Rendition" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

CaptureDur

The value of captureDur is a <clock value> that indicates the duration of the first audio. Clock value is always in relative time.

Example:
"4": 4 seconds audio duration

8.18 <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:SimpleObjectBaseType"/>
```

8.19 <mpv:Document>

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

```
<xs:element name="Document" type="mpv:SimpleObjectBaseType"/>
```

8.20 <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:SimpleObjectBaseType"/>
```

8.21 <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:SimpleObjectBaseType"/>
```

8.22 <mpv:Par>

The Par element defines a composite object in which a set of media objects occur synchronously with each other.

```
<xs:element name="Par" type="mpv:ParType"/>
<xs:complexType name="ParType">
    <xs:complexContent>
        <xs:extension base="mpv:CompositeObjectBaseType">
            <xs:sequence>
                <xs:group ref="mpv:ObjectsList" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="mpv:Related" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="mpv:Rendition" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="hint" type="xs:string"/>
        </xs:extension>
```



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

hint

The hint applies to the Par object and has an open vocabulary. Hints values must use URN-qualified names to avoid the possibility of name collisions, such as "urn:mycompany-com:mpv:someobject".

8.23 <mpv:Seq>

The Seq element defines a composite object in which the set of media objects occur in an ordered sequence.

```
<xs:element name="Seq" type="mpv:SeqType"/>
<xs:complexType name="SeqType">
    <xs:complexContent>
        <xs:extension base="mpv:CompositeObjectBaseType">
            <xs:sequence>
                <xs:group ref="mpv:ObjectsList" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="mpv:Related" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="mpv:Rendition" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="hint" type="xs:string"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

hint

The hint applies to the Seq object and has an open vocabulary. Hints values must use URN-qualified names to avoid the possibility of name collisions, such as "urn:mycompany-com:mpv:someobject".

Chapter 9: MPV Album Module Practices

The MPV specification defines a manifest file and metadata formats for photo-video collections. It also specifies best practices. These practices are encouraged to allow for consistent user experience and interoperability of MPV collections and referenced data files.

The MPV logo certification test plan tests for implementation of many of these practices. Products seeking to use the MPV logo or simply to ensure compatibility with other MPV applications will need to comply.

9.1 Identification Practices

9.1.1 Types of Identifiers

MPV applications are highly recommended to use two types of computed identifiers widely – the UUID and the MD5. Identifiers need to be computed and compared at two times: when adding a new object to a collection and when resolving an object in collection into a file with data.

UUID – UNIVERSALLY UNIQUE IDENTIFIER

The UUID algorithm is widely deployed in commercial operating systems and source code is available. It is valuable when an identifier is needed quickly. The UUID algorithm generates 128-bit statistically unique values that have no relation to the content they identify; typically, they are inserted into the content as metadata to make the association more robust. However, requiring the identifier to be embedded also makes it fragile, because the identifier cannot be regenerated from the content data if the identifier gets lost or separated from the content. Identifiers can be lost even when inserted into a file because another application may edit the file and discard or damage the identifier as unknown metadata.

In MPV, UUIDs are representd as 32-byte hexidecimal strings.

MD5

The MD5 algorithm is widely deployed in commercial applications and source code is available. It is valuable when an identifier is needed fairly quickly. The MD5 algorithm generates 128-bit statistically unique values that are entirely dependent on the content they identify; this makes them fragile to changes in the content, but they do not need to be embedded in the content.

In MPV, MD5 values are representd as 32-byte hexidecimal strings.

9.1.2 Identifier Insertion and Extraction

MPV defines practices for inserting and extracting identifiers in a variety of popular file types. MPV applications shall follow these techniques precisely so that every MPV application can extract previously inserted identifiers. These practices are specified in detail in an appendix.

9.1.3 Identifier Computation and Naming

MPV defines practices for computing and naming identifiers. MPV applications shall follow these techniques precisely so that every MPV application can recompute previously computed identifiers. These UUID and MD5 computation algorithms are specified in detail in an appendix. In particular, the following computations are defined:

- urn:osta-org:mpv:uuid
- urn:osta-org:mpv:dsig:all:md5
- urn:osta-org:mpv:dsig:body:md5
  - Body identification is defined for popular file types.
- urn:osta-org:mpv:dsig:head:<byte count>:md5
- urn:osta-org:mpv:dsig:tail:<byte count>:md5

9.1.4 Best Practices for Identifiers

The best practices for identifiers use are to do the following when creating a new reference:

1. In all cases, at least one contentID should be computed. This will not require any modification to the referenced file itself. Two contentID signatures are recommended to be computed, when possible. The body:md5 signature is based on the media content of the file and is more robust than an all signature because it is less subject to damage by metadata editing. In addition to the body signature, an all:md5 signature is also recommended, as this value can be used to determine if a file has been changed since it was referenced in addition to being used for identification purposes.
2. When the referenced file already has an instanceID that is a UUID [see an Appendix for practices on determining this], retrieve it and use it. This may require reformatting the string to comply with MPV practices of 32-character UUID strings.
3. If the referenced file does not have an instanceID, one may be provided by computing a new UUID. If a new instanceID is generated, also should also be inserted into the referenced file. If there is no intent to insert the instanceID into the referenced file, it is not recommended to create one.
4. When the referenced file already has a documentID [see an Appendix for practices on determining this], retrieve it and use it. This may require reformatting the string to comply with MPV practices of 32-character UUID strings.
5. If the referenced file does not have a documentID, one may be provided. If a new documentID is generated, also should also be inserted into the referenced file. If there is no intent to insert the documentID into the referenced file, it is not recommended to create one.

9.1.5 Comparing Identifiers

The best form of comparison is for the complete ID value string to match. However, in some cases, only the 128-bit unique identifier value may be available, such as when the embedded metadata of a file only supports this format. In such cases, a sufficient match is for only the unique identifier values strings to match.

Whenever comparing identifer strings from other sources, all dash (-) characters which may be integrated into the identifier string value should be ignored to ensure a proper comparison.

9.2 Best Practices for LastURL Values

The lastURL is the easiest and fastest way to resolve a linkage between an MPV collection item and its assocatiated data file. Avoiding breakage of the lastURL value should be an objective of any application authoring MPV documents. The following best practices are recommended.

9.2.1 MPV Producers

RELATIVE PATHNAMES, NOT ABSOLUTE PATHNAMES

It is recommended for all lastURL values that represent paths to local files utilize relative pathnames, not absolute pathnames. This allows an MPV collection and its related files to be moved around within a filesystem without breaking the lastURL references.

MULTIPLE LASTURL ELEMENTS

A defensive maneuver that is low-cost and practical is to specify multiple alternate lastURL elements for any given object. Use the fileSystem attribute to hint to the processing application which lastURL it might try first if it knows the active file system.

PLACE IDENTIFIERS ON THE PATH AS ARGUMENTS

In addition to specifying identifier values as attributes of an element, also places the identifiers as arguments on the lastURL value. This allows for MPV-aware file handling APIs to emerge that can use the identifiers to do the fixup "under the covers". In particular, identifier values should be specified as attributes when the lastURL is a reference to any kind of server-mediated storage, including local file servers or remote webservers. Providing the identifier allows the server to do backend processing to access the datafile even if the pathname is incorrect.

9.2.2 MPV Consumers

MULTIPLE LASTURL MATCHING

Try all the lastURL values specified for an object before initiating fixup. Finding a working lastURL value is the fastest path to resolving the reference. If the fileSystem is known, check for an element with a matching fileSystem value. This may be of particular benefit when playing collections off of CDs.

STRIP OFF ARGUMENTS FOR LOCAL FILENAME REFERENCES

LastURL values will often include identifiers added on as arguments. Some file handling APIs may not support this syntax. Try stripping off the arguments and trying again.

PARTIAL PATH MATCHING

When a lastURL breaks because it uses a long name not supported by the current file system, try following the path while matching only the first five or six characters of each path segment. This may be successful in some cases, especially to locate a candidate directory that may contain the desired file.

9.3 LastURL Fixup Behaviour

When using an MPV collection, the objective is for the user always to have the illusion that the collection has reliably and robustly maintained the references to all objects in the collection. When the lastURL value fails to resolve, the objective is for the user never to be aware that the LastURL value required fixup. The fixup should be rapid and silent whenever possible.

In practice, MPV implementers know that lastURL references will break regularly and require frequent fixing. This can occur do to the user renaming or relocating referenced files, changing the location of the MPV collection document, or simplying using a storage media with multiple filesystems that are unable to reliably represent file and directory names.

Poor MPV applications will do no fixup; diligent MPV applications will make extensive efforts to fixup values. A variety of approaches and techniques for fixing up references is foreseen with a range of performance and robustness tradeoffs; fixup capabilities may become a point of differentiation among MPV applications.

The basic approach to fixup is to utilize the contentID values that are available in the MPV collection to re-establish connection to the referenced object. The significant advance that MPV makes in industry practices is to establish appropriate metadata formats and practices such that this becomes widely possible and implemented. Advanced implementations may also use documentID values and other renditions to regenerate needed objects on demand.

The basic fixup algorithm will scan for candidate files in limited locations, such as the current working directory. The advanced MPV implementation will scan a wide variety of locations, possibly conducting background scans and cached identifier values so that fixup can be immediate.

A basic fixup algorithm is as follows for local file references. This algorithm is a baseline for fixup implementations; in particular, it only scans files in one directory. This is not a recommended algorithm for remote references – in that case, it is assumed the server handling the request has performed its own search before reporting the reference unresolvable.

```
Strip the filename off the lastURL path
If the resulting directory is reachable
     use this path for the scan
else
     // the lastURL directory path is probably also broken
     use the current working directory of the processing application
Scan for all files in the directory with the same filetype or extension
If the MPV item needing fixup has a UUID-based ContentID
     If the filetype is conducive to fast lookup of embedded ContentIDs
          Do UUID-based identification test first
If the MPV item needing fixup has a MD5-based ContentID
     Do MD5-based identification testnext
If the MPV item needing fixup has other id algorithms known to the processing app
     Do those id tests last
For each candidate file
     If doing UUID-based id test
          look for UUID-based contentID in the target file
          if found and matches
               done
     If doing MD5-based id test
          compute MD5 value of target file [honoring all/body/head/tail qualifiers]
          if matches
               done
     If doing other id test
          compute id value of target file
          if matches
               done
If found match
     Fixup lastURL base path with newly located file; retain all arguments
```


As a performance optimization, it is recommended that the results of lookup or computation of all contentID values for target files be cached. This will allow subsequent fixup of other lastURL values to be very fast.

9.4 Best Practices With Storage Media

The MPV collection can be used with any type of storage media, including stamped and recordable CDs and DVDs, memory cards, and harddisks.

9.4.1 CD Best Practices

MPV collections are stored in datafiles which may be placed on a stamped or recordable CD. An important best practice is to ensure that at least one MPV collection manifest file will be found by a processing application that begins its scan to locate a MPV document at the root of the disc file system.

SUPPORTED FORMATS

The CD format must provide a filesystem and lossless reading of data. This includes the following well-known and commonly used formats.

- Yellow Book for CD-ROM and CD-ROM XA
- Green Book for CD-Interactive (CD-I)
- Orange Book for recordable CDs (CD-R, CD-RW)
- White Book for Video CD
- Blue Book for Enhanced Music CD (CD EXTRA)
- CD-I Bridge
- Multisession CD
- Photo CD

Also, it is assumed that all future CD formats, such as being produced by the Mt. Rainier Initiative, will be compatible with MPV because they will all support storage and retrieval of data files in a filesystem.

Of particular signifcance for MPV are three formats:
- Orange Book for recordable CDs (CD-R, CD-RW)
- White Book for Video CD
- Multisession CD

That is because MPV is ideally suited for use on recordable CDs created by applications used by end-users as well as by commercial applications and users. The following significant use cases are called out:

ORANGE BOOK DATACD WITH MPV CONTENT

At the present, the most common file systems used by consumers will be:

- ISO 9660-1: provides widespread compatibility, but only has 8.3 filenames and other significant limitations.
- Joliet: provides 64 character Unicode filenames and is usable on computers running Microsoft Windows 95 and above OS releases.
- HFS: provides 32 character filenames and is usable on computers running the Apple Macintosh OS.
- UDF 1.5: provides 255 character Unicode filenames and usable on computers running xxx. Typically used only on discs written in packet-writing mode.

Many or even all of these filesystems can co-exist on the same disc.

An MPV file references other files that are placed on the disc. There are two typical points at which problems can arise with MPV collections placed on the disc.

- lastURL references are broken as the disc contents are structured. For example, the user may specify the files representing the raw datafiles plus the collection in a disc authoring program. If those files are reorganized relative to their position on the harddisk, the lastURL references may break.
- lastURL references are broken as the disc contents are accessed. The active filesystem on the disc does not support the reference names embedded the lastURL document.

WHITE BOOK VIDEOCD WITH MPV CONTENT

It is possible to place a photo-video collection with MPV manifest in the data track of a VideoCD while the VideoCD portion of the disc provides an alternate presentation of the photo-video content.

When this disc is played in an MPV-aware device, the device should provide the user the choice whether to access the disc in VideoCD mode or in MPV mode.

In MPV mode, the MPV document may reference VideoCD-formatted content that is also on the disc, such as a video stream conforming to VideoCD specifications. Accessing VideoCD-formatted content referenced by the MPV collection should be possible. A typical use would be to playback a VideoCD-formatted video steam representing the MPV slideshow experience that is also used for the same purpose when the disc is played in VideoCD mode.

MULTISESSION CD WITH MPV CONTENT

Obviously, the MPV document should be rewritten with each additional CD session on the disc if the content it references has changed. An advanced MPV-aware application would check that all references were valid before burning another session that contained an MPV collection.

9.4.2 DVD Best Practices

DVDs are fundamentally data discs that use the UDF 1.02 file system. This provides a robust storage platform for use by MPV collections and because of their enormous storage capacity, users will benefit greatly when MPV collections are provided to facilitate access to their content. An important best practice is to ensure that at least one MPV collection manifest file will be found by a processing application that begins its scan to locate a MPV document at the root of the disc file system.

MPV can co-exist with the datafiles required by the DVD-Video format, allowing for a photo-video collection with MPV manifest to also be placed on a DVD-Video disc with renditions of the same content in the DVD-Video format.

9.4.3 Memory Card Best Practices

An important best practice is to ensure that at least one MPV collection manifest file will be found by a processing application that begins its scan to locate a MPV document at the root of the card file system.

9.4.4 Computer Harddisks

An important best practice is to ensure that at least one MPV collection manifest file will be found by a processing applicationThis can be used to provide access to one or more or even all of the collections on the harddisk.

The best practices scanning algorithm that for harddisk-based collections is somewhat different than described in the section on locating and extracting MPV documents. The following practices are recommended.

The user may expect that many different MPV-aware applications should be able to access the same set of albums. This requires a convention for locating a root MPV collection. The following directories are recommended for storing the root MPV collection, in order of preference:

- `/Desktop/My Documents/My Pictures`
- `C:/Documents and Settings/<user>/My Documents/My Pictures`
- `C:/Documents and Settings/All users/Application Data/MPV/<user>.MPV`
- `Breath-first alphabetical scan of all directories up to three levels below the root directory.`

When the intent is to access an MPV collection with local scope, the algorithm should be:

- `Current working directory`
- `Breath-first alphabetical scan of all directories up to two levels above the current location.`
- `Breath-first alphabetical scan of all directories up to three levels below the current location.`

This algorithm will find MPV collections produced by cameras conforming to the DCF specification.

9.5 Metadata Storage and Precedence Guidelines

MPV has the objective of capturing, storing, and exchanging metadata about digital assets. MPV is highly focused on management of collections photo-video assets and related media objects. MPV metadata is preferentially maintained apart from the assets themselves, making it non-invasive and easy to deploy, process, and update without significant or even any changes to existing implementations.

When metadata about an object is believed best handled by storing it in the object itself, the recommended practice is to use XMP [XMP-FW] in native RDF-style grammar or alternately to use well-known metadata formats such as Exif.

Both approaches are extremely useful. By practice, MPV metadata external to the object shall take precedence over metadata internal to the object. This allows, for example, a given object to have and display multiple titles.

Chapter 10: MPV VXMP Module

10.1 Introduction to XMP

[This information is adapted from information on Adobe's XMP website and informational literature available from http://www.adobe.com/products/xmp/main.html and does not represent an endorsement by OSTA or I3A or its affiliated members for any of the products mentioned.]

The eXtensible Metadata Platform (XMP) provides a common XML framework that standardizes the creation, processing, and interchange of document metadata across all manner of content workflows.

XMP encompasses the following: framework, schema, XMP packet technology, and the XMP Software Development Kit, which is available as an open-source license. XMP is based on a W3C's open standard for metadata, known as the Resource Description Framework (RDF). XMP is available at no cost for integration with any application or system.

XMP embeds metadata inside application files. Because the metadata is enclosed within the file, documents retain their context when they exit their original system or environment. The embedded metadata can include any XML schema, provided it is described in RDF syntax. Extensible, embedded metadata in application files provides significant potential for repurposing, archiving, and automation in publishing workflows.

Available as an open-source license and royalty free, XMP can be integrated into any system or application. Adobe has integrated the XMP framework into Adobe® Acrobat® 5.0, Adobe InDesign® 2.0, Adobe Illustrator® 10, and Adobe Photoshop® 7, and in forthcoming new releases of other Adobe products.

Industry support for the XMP framework comes from companies such as Documentum, IBM, Kodak, KPMG, North Plains Systems, and many others.

Key benefits

- XMP embeds metadata inside application files, which allows the files to retain context when passed outside of a database or publishing system.
- Because XMP is extensible, users can include any XML schema inside the XMP packet, provided it is described in RDF syntax.
- XMP is built on W3C standards, which takes a lot of the guesswork out of integration. Standardizing the metadata framework allows for the interchange of metadata across many different systems and applications.
- XMP is shared as an open-source license. Users and integrators get access to the source code via the SDK.
- Acrobat 5.0, InDesign 2.0, Illustrator 10, Photoshop 7, and eventually all other Adobe applications will support XMP, as will those of other vendors supporting XMP in their products. This facilitates easy metadata exchange between applications supporting the XMP specification.

- XMP is available for integration with any application or system. Because the framework is completely extensible, it can be extended to include any XML schema.

10.2 Module Introduction

COMPATIBLE APPROACHES

XMP and MPV have similar objectives – capturing, storing, and exchanging metadata about digital assets. MPV is highly focused on management of collections photo-video assets and related media objects. MPV metadata is preferentially maintained apart from the assets themselves, making it non-invasive and easy to deploy, process, and update without significant or even any changes to existing implementations.

XMP provides detailed metadata schema and practices that especially represent metadata about each of the assets in a collection. XMP metadata is preferentially embedded in the asset, providing a mechanism that tightly binds metadata to the data.

Both approaches are extremely useful, and to enhance the interoperability of these approaches and foster industry efforts to achieve a single common set of metadata representations for media objects, MPV adopts wholly the XMP family of schema.

XMP AND VXMP MAPPING

MPV and XMP both use XML, but MPV use the XML-Schema framework for defining its structure while XMP uses the RDF framework. To use the XMP schema in MPV, a XML-Schema mapping of the XMP RDF schema is used. This mapping is fully mechanical and can be fully automated to allow for easy interoperability with XMP-aware applications and formats and is described in an Appendix; it is expected to be available from Adobe in the future. Through this mapping, all the XMP RDF schema are available as VXMP XML-Schema representations.

The availability of an XML-Schema representation of XMP is of particular benefit to MPV because it allows for robust validation of MPV content; VXMP stands for Validating XMP and the schema and compliant content can be validated using standard XML Schema validation tools. Because many firmware applications in consumer electronics devices will be producers and consumers MPV content, robust validation techniques are highly valuable to help ensure proper implementation. Validation is an important part of the MPV practices and compliance tests that facilitate MPV content interoperability.

The VXMP representation also utilizes a shortcut notation that makes it less verbose to use XMP schemas while still being completely compatible with the semantics of XMP. The VXMP mapping constrains the RDF-based syntax of XMP in such a way that it can be fully specified using an XML Schema.

Using the <mpv:VXMP> Album Module element along with the appropriate VXMP schema is the preferred means in MPV to specify a wide variety of useful properties about albums and album items. MPV processors supporting the MPV Album Module are required to process the set of XMP-supplied schemas (see below) that apply to their domains, allowing for a rich exchange of data between MPV-aware applications.

Example of VXMP metadata (same content as in XMP example below):

```
<?xml version="1.0" encoding="UTF-8" ?>
<mpv:mpv xmlns:mpv="urn:osta-org:mpv:1.0"
         xmlns:xap="http://ns.adobe.com/xap/1.0/">
...
  <mpv:VXMP>
    <xap:xap>
      <xap:Author>Pieter van Zee</xap:Author>
      <xap:CreateDate>2002-03-25T21:07:00Z</xap:CreateDate>
    </xap:xap>
```

```
  </mpv:VXMP>
...
</mpv:mpv>
```

NATIVE XMP IN MPV IS ALSO POSSIBLE

XMP metadata in native RDF format can be encoded in the MPV Album Module using the mpv:Metadata element; however, MPV processors are not required to support extraction and processing of any data contained by mpv:Metadata, so XMP data provided in this manner is less interoperabile with MPV applications. This content cannot be validated using readily-available commercial tools.

Example of XMP metadata (same content as in VXMP example above):

```
<?xml version="1.0" encoding="UTF-8" ?>
<mpv:mpv xmlns:mpv="urn:osta-org:mpv:1.0"
         xmlns:xap="http://ns.adobe.com/xap/1.0/"
         xmlns:x="adobe:ns:meta/"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:xmp="http://ns.adobe.com/xap/1.0/">
...
  <mpv:Metadata>
    <x:xmpmeta>
      <rdf:RDF>
        <rdf:Description about=''>
          <xmp:Author>Pieter van Zee</xmp:Author>
          <xmp:CreateDate>2002-03-25T21:07:00Z</xmp:CreateDate>
        </rdf:Description>
      </rdf:RDF>
    </x:xmpmeta>
  </mpv:Metadata>
...
</mpv:mpv>
```

10.3 VXMP Schema

XMP provides a simple, modular schema for the following basic sets of useful metadata:

- Core properties: Author, Authors, CreateDate, CreatorTool, Description, Format, Keywords, Locale, ModifyDate, Nickname, Title
- Graphics: ColorSpace, Compression, GraphicsType, NaturalDimensions, NumberOfColors, NumberOfInks
- Image: Dimensions, Resolution
- Dynamic Media: Duration, NTracks, Tracks
- Video: BitRate, Dimensions, Interleaved, NaturalRate, Compression, Encoding
- Audio: ChannelCount, Compression, Rate, SampleSize, Volume
- Text: Encoding, FontList
- PagedText: MaxPageSize, NPages
- Rights management: Certificate, Copyright, Owner, UsageTerms
- Media management: ContainedResources, ContributorResources, DocumentID, History, LastURL, Manager, ManageTo, RenditionClass, RenditionOf, SaveID, VersionID, Versions

XMP is also developing XMP-compatible specifications for existing metadata formats, such as Exif and IPTC.

These schema are provided in an Appendix. The value syntax and semantics is wholly defined by the XMP specification provided by Adobe [XMP-FW].

Schema group	Namespace Identifier	Conventional Namespace Prefix	Specified Prefix for Namespace-Incompatible Environments
Dublin Core	*http://purl.org/dc/elements/1.1/*	dc:	dc_
PDF Core	*http://ns.adobe.com/pdf/1.3/*	pdf:	pdf_
XAP Core	*http://ns.adobe.com/xap/1.0/*	xap:	xap_
Graphics	*http://ns.adobe.com/xap/1.0/g/*	xapG:	xapG_
Image	*http://ns.adobe.com/xap/1.0/g/img/*	xapGimg:	xapGimg_
Dynamic Media	*http://ns.adobe.com/xap/1.0/dyn/*	xapDyn:	xapDyn_
Video	*http://ns.adobe.com/xap/1.0/dyn/v/*	xapDynV:	xapDynV_
Audio	*http://ns.adobe.com/xap/1.0/dyn/a/*	xapDynA:	xapDynA_
Text	*http://ns.adobe.com/xap/1.0/t/*	xapT:	xapT_
PagedText	*http://ns.adobe.com/xap/1.0/t/pg/*	xapTPg:	xapTPg_
Rights management	*http://ns.adobe.com/xap/1.0/rights/*	xapRights:	xapRights_
Media management	*http://ns.adobe.com/xap/1.0/mm/*	xapMM:	xapMM_
Support	*http://ns.adobe.com/xap/1.0/s/*	xapS:	xapS_
Basic Job Ticket	*http://ns.adobe.com/xap/1.0/bj/*	xapBJ:	xapBJ_

Of the three "Core" schemas, the Dublin Core schema should be used preferentially when the same property xists in multiple Core schema. The PDF core is deprecated.

In addition to the primary schema groups defined above, XMP also defines the following property types.

Schema	Namespace Identifier	Conventional Namespace Prefix	Specified Prefix for Namespace-Incompatible Environments
Choice, XChoice	*http://ns.adobe.com/xap/1.0/ValueQualifier#*	vQual:	vQual_
ResourceEvent	*http://ns.adobe.com/xap/1.0/sType/ResourceEvent#*	stEvt:	stEvt_
ResourceRef	*http://ns.adobe.com/xap/1.0/sType/ResourceRef#*	stRef:	stRef_
Version	*http://ns.adobe.com/xap/1.0/sType/Version#*	stVer:	stVer_
File Disposition	*http://ns.adobe.com/xap/1.0/sType/FileDisposition#*	stDsp:	stDsp_
Job	*http://ns.adobe.com/xap/1.0/sType/Job#*	stJob:	stJob_
Dimensions	*http:ns.adobe.com/xap/1.0/sType/Dimensions#*	stDim:	stDim_
Duration	*http:ns.adobe.com/xap/1.0/sType/Duration#*	stDuration:	stDuration_
Font	*http://ns.adobe.com/xap/1.0/sType/Font#*	stFnt:	stFnt_
Resolution	*http:ns.adobe.com/xap/1.0/sType/Resolution#* NOTE: in the XMP documentation, the namespace identifier did not end in a #. This is believed to be an error.	stRes:	stRes_
Track Description	*http://ns.adobe.com/xap/1.0/sType/TrackDesc#*	stTrk:	stTrk_

10.4 VXMP Practices

10.4.1 Embed XMP in Asset Files, not VXMP

For highest interoperability with other mainstream applications, applications that embed metadata in asset files should use pure XMP notation, not VXMP. This approach achieves the highest interoperability with existing XMP applications, of which there will be many.

Typically, however, only ID information is embedded in or extracted from asset files by MPV-aware applications. Metadata, even asset-specific metadata, that is developed for use in an MPV document, will use the VXMP notation.

10.4.2 Precedence Rules for Duplicate Data

By convention, the properties stored in the MPV file are considered to take precedence over any similar properties stored in a separate data object. This approach provides unambiguous definition of which property to use in case of overlap, and allows a single referenced file to be utilized and represented in a variety of ways.

Chapter 11: MPV Presentation Module Schema

11.1 Module Introduction

The MPV Presentation Module define a few elements and attributes. These are focused on enhancing the presentation experience. The key components 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.

11.2 Schema Information

The MPV presentation module uses the mpvp: and mpvpTrans: namespace. The introductory schema information is expressed as follows.

Schema group	Namespace Identifier	Conventional Namespace Prefix	Specified Prefix for Namespace-Incompatible Environments
Presentation	urn:osta-org:mpv:presentation:1.0	mpvp:	mpvp_
Presentation TransitionFilter	urn:osta-org:mpv:presentation:1.0:TransitionFilter	mpvpTrans:	mpvpTrans_
Presentation Defaults	urn:osta-org:mpv:presentation:1.0:Default	mpvpDflt:	mpvpDflt_

The presentation schema is defined using the following namespaces.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:mpv="urn:osta-org:mpv:1.0"
xmlns:mpvp="urn:osta-org:mpv:presentation:1.0"
xmlns:mpvpTrans="urn:osta-org:mpv:presentation:1.0:TransitionFilter"
xmlns:mpvpDflt="urn:osta-org:mpv:presentation:1.0:Dflt"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="qualified">
```


11.3 <mpvp:mpvp> Media Object Presentation Schema

The Presentation Profile defines a schema for presentation properties. This schema can be used on all media objects by specying the root element of the mpvp schema as the only child of the mpv:VXMP element.

This schema is derived from the SMIL specifications [SMIL10] and [SMIL20]. The SMIL language and SMIL players are seen as an attractive tools for presenting MPV documents. A carefully constrained set of SMIL elements and attributes are chosen for this basic presentation schema. The emphasis was on selecting just those features essential to deliver a basic user experience that is easy to use and compelling and that can also be implementated across many platforms and embedded devices.

The guiding practice for applications and devices that process and present MPV content based on this schema is that presentation properties inherit to lower areas of scope. For example, a <mpvp:Fit> value specified as metadata of the <mpv:Foreground> element itself will apply to all its children.

SCHEMA

```
<xs:element name="mpvp">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="BackgroundColor" type="xs:string" minOccurs="0"/>
      <xs:element name="TextColor" type="xs:string" minOccurs="0"/>
      <xs:element name="Dur" type="mpvp:DurationType" minOccurs="0"/>
      <xs:element name="Fit" type="mpvp:FitType" minOccurs="0"/>
      <xs:element name="RepeatCount" type="xs:string" minOccurs="0"/>
      <xs:element name="RepeatDur" type="xs:string" minOccurs="0"/>
      <xs:element name="ShowRotated" type="xs:integer" minOccurs="0"/>
      <xs:element name="TransitionFilter" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="mpvpTrans:mpvpTrans"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:simpleType name="DurationType">
  <xs:restriction base="xs:string"/>
</xs:simpleType>

<xs:simpleType name="FitType">
  <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>
```


PROPERTY DESCRIPTIONS

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         ::= Timecount-value
Timecount-value     ::= Timecount ("." Fraction)?
Fraction            ::= DIGIT+
Timecount           ::= 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).

ShowRotated

Specifies the total degrees from 0 to 360 for rotation of the object when presented. The default value is 0.

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.

BackgroundColor

Specifies the background color of the element and all subelements. This is used to fill any visual region not occluded by the object's rendition. The default background color is "transparent", which implies that, by default, the implementation dependent window background will be shown. The vocabulary for BackgroundColor is defined by CSS2 system colors **[CSS2]**.

TextColor

Specifies the text color of the element and all subelements. The default text color is implemenation dependent, which implies that, by default, the implementation dependent window background color will be shown under the implementation dependent text color. The vocabulary for TextColor is defined by CSS2 system colors **[CSS2]**.

TransitionFilter

Specifies the transitionFilter that should be applied to the media object. The value of the property is an mpvpTrans:mpvpTrans element (see below)

EXAMPLE OF SYNTAX

The mpvp schema has a root element, mpvp:mpvp that appears the only child of the xmp:XMP metadata element. In the following example, only two of the optional properties of the schema are specified for the Still, the backgroundColor and the Fit properties.

```
<mpv:mpv
        xmlns:mpv=”urn:osta-org:mpv:1.0”
        xmlns:mpvp=”urn:osta-org:mpv:presentation:1.0”
        xmlns:mpvpTrans=”urn:osta-org:mpv:presentation:1.0:mpvp:Trans”
>
  . . .
  <mpv:Still>
        . . .
        <mpv:VXMP>
                <mpvp:mpvp>
```



```
                        <mpvp:BackgroundColor>Blue</mpvp:BackgroundColor>
                        <mpvp:TextColor>White</mpvp:TextColor>
                        <mpvp:Fit>meet</mpvp:Fit>
                        <mpvp:TransitionFilter>
                                <mpvpTrans:mpvpTrans>
                                        <mpvpTrans:Type>barWipe</mpvpTrans:Type>
                                </mpvpTrans:mpvpTrans>
                        </mpvp:TransitionFilter>
                </mpvp:mpvp>
        </mpv:VXMP>
  </mpv:Still>
  . . .
</mpv:mpv>
```

11.4 <mpvpTrans:mpvpTrans> Transition Filter

A transition filter that implements a transition from the object before to the object after. It is applied at the completion of presenting the object on which it is defined and transitioning into the next object that is defined. This element is strictly presentation oriented. It is specified as the value of the TransitionFilter property in the mpvp schema.

SCHEMA

```
<xs:element name="mpvpTrans">
    <xs:complexType>
        <xs:all>
            <xs:element name="Dur" type="xs:string" minOccurs="0"/>
            <xs:element name="Subtype" type="xs:string" minOccurs="0"/>
            <xs:element name="Type" type="xs:string"/>
        </xs:all>
    </xs:complexType>
</xs:element>
```

PROPERTY DESCRIPTIONS

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.

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.

11.5 <mpvp:Default>

The Default element specifies default presentation parameters for presentation. Typical usage will be on the top-level Album or on Foreground or Background elements.

SCHEMA

```
<xs:element name="mpvpDflt">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="vxmp:TopResourceBase">
        <xs:sequence>
          <xs:element name="StillDur" type="xs:float" minOccurs="0"/>
          <xs:element ref="mpvp:mpvp"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

PROPERTY DESCRIPTIONS

StillDur

Sets the duration for presenting a still image.

mpvp

Sets default values for presenting an object.

Chapter 12: MPV Presentation Module Practices

The Presentation Module practices are largely recommended because there is an on-going growth of basic and innovative presentation techniques and attributes. Rather than strive for completeness or overspecification of behaviour, MPV strives to provide just enough presentation schema and practices to enable applications to provide the user two basic capabilities: browsing and watching. Close behind in user value for photo-video content is printing. Additional presentation capabilities may be expressed as custom metadata and processed by specialized players.

12.1 Best Practices for Presenting a Manifest

A manifest may contain more than one Album. For presentation purposes, the first album is the default Album; other content may be ignored. If additional Albums in a manifest are to be presented, they must be referred to explicitly from another Album, such as the default Album; alternately, the presentation application may choose to present the user with the list of albums defined in the manifest.

A key presentation characteristic is perceived startup time from storage media insertion or presentation invocation to begin of the presentation. This performance can be accelerated in various ways:

- place the manifest in a location that is quickly found by the scanning algorithm for manifests
- provide the user positive feedback that the Manifest is being loaded or processed
- load only one album at a time
- provide Screen and Thumbnail Renditions for images and video
- present using placeholders for objects whose lastURL values require fixup; then slowly fixup broken lastURLs

12.2 Best Practices for Watching

The basic watch experience is to play back the content in the MPV album as follows. The Background and Foreground parts of an Album have the same expressive power. Background objects and metadata are played under foreground objects and metadata, both in z-order and in audio mixing.

Using this behavior, a typical watch experience of a slideshow can be provided:

- Foreground seqence of Still, Video, and Audio content
- Background music track of Audio content

For better performance, use of "Screen" Renditions is recommended when present.

More advanced watching applications will provide the user to pause, rewind, and fast forward through the content. Additional operations may be available directly on the content as watched, such as "Mark" or "Print".

UNSUPPORTED TYPES

When an object has a media type that isn't supported for playback (e.g. a GIF image), the recommended behaviour for the watch application is to skip silently over the unsupported object.

ALBUMREF

When an object is an AlbumRef, the recommended behaviour for the watch application is to skip silently over the AlbumRef.

PROVIDED SHOWS

Watch applications should study the Renditions available on the Album or Foreground for a "Show" Rendition video. This represents a pre-computed rendition of the watch experience and should be used in preference to the watch experience the application can produce itself or at least presented as an option to the user. The "Show" rendition provides a means for sophisticated authoring and production applications to separately produce an advanced watch experience that can be accessed simply by playing a video. There is a user expectation that the Show content and Album content be approximately the same.

12.3 Best Practices for Browsing

The basic browsing experience should provide two basic capabilities:

- browsing of thumbnails of photo-video content
- browsing of full-screen views of the photo-video content

For better performance in thumbnail mode, use of "Thumbnail" Renditions for both stills and video is recommended. For better performance in full-screen mode, use of "Screen" Renditions for stills is recommended.

Advanced browsing applications will treat StillsMultishotSequence and StillsPanoramaSequence specially. For example, in thumbnail browse mode, they may show just one of the stills and iconically represent that the item is a sequence of stills.

UNSUPPORTED TYPES

When an object has a media type that isn't supported for playback (e.g. a GIF image), the recommended behaviour for the browsing application is to show an empty thumbnail with a message that the media type is not supported by this player.

ALBUMREFS

When the Album contains AlbumRef, these are presented to the user as choices to link to another Album. A "Thumbnail" or "Screen" Rendition on the AlbumRef can offer a visual depiction of the target Album.

When browsing, it is recommended that AlbumRefs be presented just as another type of object, albeit one that when selected opens another album.

RENDITIONS

Other renditions at the album and object levels may be of interest. In particular, Print objects may offer useful content to print, such as thumbnail index pages or CD labels. Browsing applications are recommended to provide the user access to renditions and objects of the type "Print" and "Text".

12.4 Best Practices for Supported Formats

The following formats are recommended to be supported for playback by players that desire to play a large percentage of the photo-video content they may encounter.

Stills:

- JPEG, in both Exif and JFIF variations
- TIFF
- GIF
- PNG

Video:

- AVI MJPEG
- MOV PJPEG
- MPEG1
- MPEG1 VideoCD
- MPEG2

Audio:

- WAV
- MPEG1 Layer 3 (MP3)
- MPEG1 Layer 2

Print:

- PDF

12.5 Examples

[TODO]

12.5.1 Startup List of Albums with Background Image

12.5.2 Representing a Title Image

12.5.3 Album Renditions of a Video Slideshow and Printed Content

12.5.4 Building Up a StillSequenceWithAudio Type

Appendix I: Practices for Embedding Identifiers and Rendition Information in Media Files

XMP defines conventions for embedding metadata in document types. MPV makes use of these conventions and defines a minimal xpacket that should be embedded by compliant MPV applications.

The minimal information that should be embedded is based on the XMP guidelines. MPV requires that two schemas be used in the embedded information. These are the mpvMM schema and the xapMM schema.

I.1 MPV media management schema

[TODO – more to do here]

The mpv media management schema provides some additional properties that aren't currently defined by XMP that the MPV framework requires in order to guarantee a reasonable level of base capability in envisaged operating environments.

The schema is specified using both an XMP/MPV with the mechanical transformation of Appendix IV: being used to provide the native XMP syntax that is actually used in embedding.

SCHEMA

mpvMM schema allows the specification of an ordered list of contentID that are associated with this document instance.

Schema namespace: "urn:osta-org:mpv:1.0:mpv:MM
Schema namespace prefix "mpvMM"

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:osta-org:mpv:1.0:MM" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xapMPV="urn:osta-org:mpv:1.0:xapMPV" xmlns:mpvMM="urn:osta-org:mpv:1.0:MM"
elementFormDefault="qualified" attributeFormDefault="qualified">
    <xs:annotation>
```



```
        <xs:documentation>
        The mpv media management schema that is used in embedded descriptors
    </xs:documentation>
    </xs:annotation>
    <!--
    imports
    -->
    <xs:import namespace="urn:osta-org:mpv:1.0:xapMPV" schemaLocation="../xap/xapMPV.xsd"/>
    <!--
    xap:xap
    -->
    <xs:element name="mpvMM">
        <xs:annotation>
            <xs:documentation>
    The mpvMM element is the root element in the mpv media management schema.
    </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="ContentIDs">
                    <xs:complexType>
                        <xs:complexContent>
                            <xs:extension base="xapMPV:Bag">
                                <xs:sequence>
                                    <xs:element ref="mpvMM:ContentIDs_li"/>
                                </xs:sequence>
                            </xs:extension>
                        </xs:complexContent>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <!--
    xap:ContentIDs_li
    -->
    <xs:element name="ContentIDs_li"/>
</xs:schema>
```

EXAMPLE

Example for a thumbnail:

```
<?xpacket ?>
<rdf:RDF>

 <mpvMM:mpvMM rdf:about="">
     <mpvMM:contentIDs>
        <rdf:Seq>
            <rdf:li>urn:osta-org:mpv:dsig:body:md5:__sig_value__</rdf:li>
            <rdf:li>urn:osta-org:mpv:uuid:__uuid_value__</rdf:li>
        </rdf:Seq>
     </mpvMM:contentIDs>
 </mpvMM:mpvMM>

 <xapMM:xapMM rdf:about="">
     <xapMM:documentID>urn:osta-org:mpv:uuid:__uuid_value__</xapMM:documentID >
     <xapMM:RenditionOf>
            <stRef:stRef>
```

```
                    <stRef:documentID>urn:osta-org:mpv:uuid:9705C6F1-8db6-11d5-a7dd-
  cdf3d5439594</stRef:documentID>
                     <stRef:contentID>urn:osta-
  org:mpv:dsig:all:md5:___dsigvalue__</stRef:contentID>
                    <stRef:versionID>3</stRef:versionID>
                    <stRef:RenditionClass>thumbnail</stRef:RenditionClass>
              </stRef:stRef>
      </xapMM:RenditionOf>
  </xapMM:xmpMM>

</rdf:RDF>
<?xpacket ?>
```

I.2 Exif 2.1, TIFF, GIF, PNG, HTML, PDF, SVG/XML, Adobe Illustrator .ai, EPS

For purposes of embedding identifiers in to the named filetypes, the XMP Embedding specification defines the approach.

I.3 Exif 2.2

[todo]

I.4 Video

[todo]
MPEG1
MPEG2
AVI
MOV
WMV

I.5 Audio

[todo]
MP3
WAV
WMA

Appendix II: Media Types Reference

[TODO] – this Appendix still needs a lot of work, primarily just cleanup. A key major change is to move towards 100% compatibility with MIMETypes. This will support interoperability with Adobe XMP. We will define a vendor-supplied prefix to make all these in to MIME types.

II.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".

II.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	

II.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	Description

Major Type = "MPEG2_PES"	
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

II.4 Image Types

*** More do do here ***

II.5 Metadata Types

*** More do do here ***

binary/Exif
text/Exif
text/XMP
text/DIG35

II.6 Print Types

*** More do do here ***

application/pdf
text/XHTML

II.7 Text Types

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

II.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

Appendix III: 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), "fanInVertical" (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", "fadeFromColor"

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.

Appendix IV: XMP-VXMP Mapping

MPV bases many of its core technical approaches on the conceptual framework defined by Adobe's Extensible Metadata Platform (XMP) [XMP-FW]. MPV defines a syntactic mapping of the XMP framework that allows the specification of XMP using XML schema that can be mapped mechanically and easily to the XML/RDF encoding used by XMP. The MPV toolkit will provide efficient streaming implementations of conversions from the MPV encoding of XMP to the native RDF encoding.
Below, the native XMP representation in RDF/XML is referred to as XMP while the representation of XMP in XML Schema is referred to as VXMP (validating XMP).
XMP 1.0 uses a three level syntactic organization of metadata:

Resource -> Schema -> Property

Resource

The XMP 1.0 resource level is represented by an rdf:RDF element that will only contain metadata describing a single resource. This level will contain one or more schema instances describing the resource.

Schema

The XMP 1.0 schema instance is represented by a rdf:Description element or typed node element. Each XMP schema is directly encoded in an XML schema that uses the same namespace as the XMP schema.

Property

The top-level properties contained in a schema instance will all have the same namespace as the schema. Each XMP property will have the same naming conventions and data type in its XML Schema representation as in its XMP schema representation.
The VXMP representation constrains the XMP syntax using the following rules:

- The top-level syntactic wrappers of the XMP up to the rdf:Description element are elided in the VXMP representation.
- The XMP convention (see each top-level schema) is described in a separate rdf:Description which is mapped to an wrapper element in the host document. In the case of MPV the host element is named mpv:VXMP.
- The resource identification information such as the instanceID which is the value of rdf:about, xmp:documentID and xmp:version is implicit and provided by the host document. The MPV host document has a complete mapping of identifiers for maintaining traceability between the XMP representation and the VXMP representation.
- Properties are always encoded as elements and the value of the property is encoded as the content of the element. Attributes are not used.
- The XML schema is defined based on the ordering in the XMP 1.0 framework document. The properties are listed in alphabetical order.

IV.1 LocalName and Namespace mapping

- The names and namespaces from the XMP specification are directly used in the VXMP representation. All of the property valuetypes and vocabularies (see sections 4.3 and 4.4 of the XMP specification) are specified in a single support schema with the namespace of:
  - "http://ns.adobe.com/vxmp/1.0/"
- This schema is the base namespace for the any schema constructs that aren't specified as being part of a specific schema in the XMP specification.

IV.2 TypedNode mapping

- All top-level and nested RDF resources are encoded as the equivalent of RDF typed nodes.
- The top-level typednode namespace is that of the schema and the localname is the XMP namespace prefix used in the XMP specification.
- The nested RDF resources (complex property values) are encoded as typed nodes where the namespace is that of the schema and the local-name is the name specified in the XMP specification.

IV.3 Container mapping

- All containers elements (rdf:Bag, rdf:Seq and rdf:Alt) are collapsed out and replaced by a naming convention on the property which is the parent of the container element.
- The naming convention is to append an underscore character to the property name followed by the container localname, i.e. either "Seq", "Bag" or "Alt". .
- The container list items are named using the name of the parent element, i.e. the name before having the suffix appended.
- VXMP also directly supports the simplified syntax that allows a container valued property with only a single list item to omit the container and list syntax and specify the list item value as the value of the property.
  - Note that while this allows round tripping between XMP and VXMP, it does not work with RDF Schema that doesn't support this syntax simplification.

IV.4 Examples

Below are two complete representations of both the XMP and VXMP versions of a metadata set. The metadata contains two schema descriptions about the same resource. VXMP assumes the same convention as XMP 1.0 where the properties from a particular top-level metadata schema are grouped in a separate element. In the case of XMP, this element is the rdf:Description element. The base VXMP mapping doesn't specify the top-level element used to group a schema instance. In the case of MPV, the wrapper element is the mpv:VXMP element.
Native XMP Representation

XMP EXAMPLE 1

```
<x:xmpmeta xmlns:x='adobe:ns:meta/' x:xmptk='XMPTk 2.8'>

<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
 xmlns:iX='http://ns.adobe.com/iX/1.0/'>
```



```
<rdf:Description about=''
 xmlns:xmp='http://ns.adobe.com/xap/1.0/'>

  <xmp:Advisory>
    <rdf:Bag>
      <rdf:li>http://ns.adobe.com/xap/1.0/ BaseURL</rdf:li>
      <rdf:li>http://ns.adobe.com/xap/1.0/ Description/*[@xml:lang='en']</rdf:li>
      <rdf:li>http://ns.adobe.com/xap/1.0/ Keywords/*[2]</rdf:li>
    </rdf:Bag>
  </xmp:Advisory>

  <xmp:Author>Jane Doe</xmp:Author>  <!-- an alias of xmp:Authors/*[1] -->

  <xmp:Authors>  <!-- an alias of dc:creator -->
    <rdf:Seq>
      <rdf:li>Jane Doe</rdf:li>
      <rdf:li>John Doe</rdf:li>
      <rdf:li>Jack Doe</rdf:li>
    </rdf:Seq>
  </xmp:Authors>

  <xmp:BaseURL>http://mydoc</xmp:BaseURL>

  <xmp:CreateDate>2001-08-13T10:42:24Z</xmp:CreateDate>

  <xmp:CreatorTool>Microsoft Visual C++ for Windows</xmp:CreatorTool>

  <xmp:Description>  <!-- an alias of dc:description -->
    <rdf:Alt>
      <!-- The following text contains Unicode characters. Note that    -->
      <!-- the XMPDumper is not able to display all of them.            -->
      <rdf:li xml:lang='en'>This document is a sample XML file</rdf:li>
      <rdf:li xml:lang='fr'>Ce document est un fichier d'exemple XML</rdf:li>
      <rdf:li xml:lang='de'>Dieses Dokument ist eine XML Beispieldatei</rdf:li>
    </rdf:Alt>
  </xmp:Description>

  <xmp:Format>text/xml</xmp:Format>  <!-- an alias of dc:format -->

  <xmp:Keywords>  <!-- an alias of dc:subject -->
    <rdf:Bag>
      <rdf:li>XMP</rdf:li>
      <rdf:li>Schema</rdf:li>
      <rdf:li>sample</rdf:li>
    </rdf:Bag>
  </xmp:Keywords>

  <xmp:Locale>  <!-- an alias of dc:language -->
    <rdf:Bag>
        <rdf:li>en</rdf:li>
        <rdf:li>fr</rdf:li>
        <rdf:li>de</rdf:li>
    </rdf:Bag>
  </xmp:Locale>

  <xmp:MetadataDate>2001-08-13T10:42:24Z</xmp:MetadataDate>
```



```
    <xmp:ModifyDate>2001-08-13T11:02:13Z</xmp:ModifyDate>

    <xmp:Nickname>sample</xmp:Nickname>

    <xmp:Title>  <!-- an alias of dc:title -->
      <rdf:Alt>
        <rdf:li xml:lang='en'>XMP Core Schema Example</rdf:li>
        <rdf:li xml:lang='fr'>XMP Core Schema Exemple</rdf:li>
        <rdf:li xml:lang='de'>XMP Core Schema Beispiel</rdf:li>
      </rdf:Alt>
    </xmp:Title>

  </rdf:Description>

</rdf:RDF>
```

VXMP EXAMPLE 1

```
<?xml version="1.0"?>
<mpv:VXMP xmlns:xap="http://ns.adobe.com/xap/1.0/" xmlns:mpv="urn:osta-org:mpv:1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:osta-
 org:mpv:1.0
..\schemas\mpv_basic.xsd">
 <xap:xap>
 <xap:Advisory_Bag>
        <xap:Advisory>http://ns.adobe.com/xap/1.0/BaseURL"</xap:Advisory>

 <xap:Advisory>http://ns.adobe.com/xap/1.0/Description/*[@xml:lang='en']</xap:Advisory>
        <xap:Advisory>http://ns.adobe.com/xap/1.0/Keywords/*[2]</xap:Advisory>
 </xap:Advisory_Bag>
 <xap:Author>Jane Doe</xap:Author>
 <!-- an alias of xap:Authors/*[1] -->
 <xap:Authors_Seq>
        <!-- an alias of dc:creator -->
        <xap:Authors>Jane Doe</xap:Authors>
        <xap:Authors>John Doe</xap:Authors>
        <xap:Authors>Jack Doe</xap:Authors>
 </xap:Authors_Seq>
 <xap:BaseURL>http://mydoc</xap:BaseURL>
 <xap:CreateDate>2001-08-13T10:42:24Z</xap:CreateDate>
 <xap:CreatorTool>Microsoft Visual C++ for Windows</xap:CreatorTool>
 <xap:Description_Alt>
        <!-- an alias of dc:description -->
        <!-- The following text contains Unicode characters. Note that    -->
        <!-- the XMPDumper is not able to display all of them.            -->
        <xap:Description xml:lang="en">This document is a sample XML
 file</xap:Description>
        <xap:Description xml:lang="fr">Ce document est un fichier d'exemple
 XML</xap:Description>
        <xap:Description xml:lang="de">Dieses Dokument ist eine XML
 Beispieldatei</xap:Description>
 </xap:Description_Alt>
 <xap:Format>text/xml</xap:Format>
 <!-- an alias of dc:format -->
 <xap:Keywords_Bag>
```



```
        <!-- an alias of dc:subject -->
        <xap:Keywords>XMP</xap:Keywords>
        <xap:Keywords>Schema</xap:Keywords>
        <xap:Keywords>sample</xap:Keywords>
  </xap:Keywords_Bag>
  <xap:Locales_Bag>
        <!-- an alias of dc:language -->
        <xap:Locales>en</xap:Locales>
        <xap:Locales>fr</xap:Locales>
        <xap:Locales>de</xap:Locales>
  </xap:Locales_Bag>
  <xap:MetadataDate>2001-08-13T10:42:24Z</xap:MetadataDate>
  <xap:ModifyDate>2001-08-13T11:02:13Z</xap:ModifyDate>
  <xap:Nickname>sample</xap:Nickname>
  <xap:Title_Alt>
        <!-- an alias of dc:title -->
        <xap:Title xml:lang="en">XMP Core Schema Example</xap:Title>
        <xap:Title xml:lang="fr">XMP Core Schema Exemple</xap:Title>
        <xap:Title xml:lang="de">XMP Core Schema Beispiel</xap:Title>
  </xap:Title_Alt>
  </xap:xap>
</mpv:VXMP>
```

XMP EXAMPLE 2

```
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:iX="http://ns.adobe.com/iX/1.0/">
        <rdf:Description about=""
  xmlns:stDsp="http://ns.adobe.com/xap/1.0/sType/FileDisposition#"
  xmlns:xmpS="http://ns.adobe.com/xap/1.0/s" xmpS:EntityTag="W/&quot;A weak ETag&quot;"
  xmpS:ResourceID="unique timestamp2001-08-14T11:02:13Z" xmpS:Size="10000">
        <xmpS:FileDisposition>
              <rdf:Alt>
                    <rdf:li rdf:parseType="Resource">
                          <stDsp:filename>aFile</stDsp:filename>
                          <stDsp:OS>Windows</stDsp:OS>
                          <stDsp:directoryPath>c:\mydir\path\
</stDsp:directoryPath>
                    </rdf:li>
                    <rdf:li rdf:parseType="Resource">
                          <stDsp:filename>aFile</stDsp:filename>
                          <stDsp:OS>MacOS</stDsp:OS>
                          <stDsp:directoryPath>hd:mydir:path:
                           </stDsp:directoryPath>
                    </rdf:li>
                    <rdf:li rdf:parseType="Resource">
                          <stDsp:filename>aFile</stDsp:filename>
                          <stDsp:OS>Unix</stDsp:OS>
                          <stDsp:directoryPath>/mydir/path/
                           </stDsp:directoryPath>
                    </rdf:li>
                    <rdf:li rdf:parseType="Resource">
                          <stDsp:filename>aPage</stDsp:filename>
                          <stDsp:OS>URL</stDsp:OS>
```



```
                                <stDsp:directoryPath>http://www.foo.bar/
                                 </stDsp:directoryPath>
                            </rdf:li>
                    </rdf:Alt>
            </xmpS:FileDisposition>
            </rdf:Description>
    </rdf:RDF>
    </x:xmpmeta>
```

VXMP EXAMPLE 2

```
<?xml version="1.0"?>
<mpv:VXMP xmlns:mpv="urn:osta-org:mpv:1.0"
  xmlns:stDsp="http://ns.adobe.com/xap/1.0/sType/FileDisposition#"
  xmlns:xmpS="http://ns.adobe.com/xap/1.0/s" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance" xsi:schemaLocation="urn:osta-org:mpv:1.0
..\schemas\mpv_basic.xsd">
  <xmpS:xmpS>
  <xmpS:EntityTag>W/"A weak ETag"</xmpS:EntityTag>
  <xmpS:ResourceID>unique timestamp2001-08-14T11:02:13Z</xmpS:ResourceID>
  <xmpS:Size>10000</xmpS:Size>
  <xmpS:FileDisposition_Alt>
          <stDsp:FileDisposition>
                  <stDsp:filename>aFile</stDsp:filename>
                  <stDsp:OS>Windows</stDsp:OS>
                  <stDsp:directoryPath>c:\mydir\path\</stDsp:directoryPath>
          </stDsp:FileDisposition>
          <stDsp:FileDisposition>
                  <stDsp:filename>aFile</stDsp:filename>
                  <stDsp:OS>MacOS</stDsp:OS>
                  <stDsp:directoryPath>hd:mydir:path:</stDsp:directoryPath>
          </stDsp:FileDisposition>
          <stDsp:FileDisposition>
                  <stDsp:filename>aFile</stDsp:filename>
                  <stDsp:OS>Unix</stDsp:OS>
                  <stDsp:directoryPath>/mydir/path/</stDsp:directoryPath>
          </stDsp:FileDisposition>
          <stDsp:FileDisposition>
                  <stDsp:filename>aPage</stDsp:filename>
                  <stDsp:OS>URL</stDsp:OS>
                  <stDsp:directoryPath>http://www.foo.bar/</stDsp:directoryPath>
          </stDsp:FileDisposition>
  </xmpS:FileDisposition_Alt>
  </xmpS:xmpS>
</mpv:VXMP>
```


Appendix V: 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.

Header
XML Data
Trailer

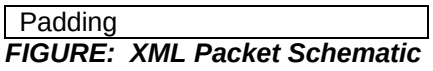


FIGURE: XML Packet Schematic

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:

```
V.1.1 <?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:

```
V.1.2 Attribute ::= Name '=' AttValue
V.1.3 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:

```
V.1.4 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:

```
V.1.5 [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:

```
V.1.6 <?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.

```
V.1.7 <?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”.

Appendix VI: MPV Basic Profile Schema Definition

This appendix contains the complete schema defined by MPV Basic Profile.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <!--
Global resource identification Attributes
-->
  <xs:attribute name="instanceID" type="xs:ID"/>
  <xs:attribute name="documentID" type="xs:anyURI"/>
  <xs:attribute name="contentID" type="xs:anyURI"/>
  <xs:attribute name="lastURL" type="xs:anyURI"/>
  <xs:attribute name="byteOffset" type="xs:integer"/>
  <xs:attribute name="xmlPacket" type="xs:integer"/>
  <!--
     global resource identification elements
     -->
  <xs:element name="DocumentID" type="xs:anyURI"/>
  <xs:element name="ContentID" type="xs:anyURI"/>
  <!--
     mpv:FilesystemType
     -->
  <xs:simpleType name="FilesystemBaseType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="URI"/>
      <xs:enumeration value="ISO9660-1"/>
      <xs:enumeration value="ISO9660-2"/>
      <xs:enumeration value="ISO9660-3"/>
      <xs:enumeration value="HFS"/>
      <xs:enumeration value="Joliet"/>
      <xs:enumeration value="UDF15"/>
      <xs:enumeration value="RockRidge"/>
      <xs:enumeration value="FAT16"/>
      <xs:enumeration value="FAT32"/>
      <xs:enumeration value="NTFS"/>
      <xs:enumeration value="Windows"/>
      <xs:enumeration value="Unix"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="FilesystemType">
    <xs:union memberTypes="mpv:FilesystemBaseType xs:string"/>
  </xs:simpleType>
  <!--
```



```
    mpv:LastURL
    -->
<xs:element name="LastURL">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:anyURI">
        <xs:attribute name="hint" type="xs:anyURI"/>
        <xs:attribute name="filesystem" type="mpv:FilesystemType"/>
        <xs:attribute ref="mpv:byteOffset"/>
        <xs:attribute ref="mpv:xmlPacket"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<!--
    mpv:ElemIdAttrGroup
    -->
<xs:attributeGroup name="ElemIdAttrGroup">
  <xs:attribute ref="mpv:id"/>
</xs:attributeGroup>
<!--
    mpv:ElemIdElemGroup
     -->
<xs:group name="ElemIdElemGroup">
  <xs:sequence>
    <xs:element ref="mpv:VXMP" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="mpv:Metadata" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:group>
<xs:attributeGroup name="ResourceIdAttrGroup">
  <xs:attributeGroup ref="mpv:ElemIdAttrGroup"/>
  <xs:attribute ref="mpv:documentID"/>
  <xs:attribute ref="mpv:instanceID"/>
</xs:attributeGroup>
<!--
    mpv:ResourceIdElemGroup
    -->
<xs:group name="ResourceIdElemGroup">
  <xs:sequence>
    <xs:element ref="mpv:DocumentID" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="mpv:InstanceID" minOccurs="0" maxOccurs="unbounded"/>
    <xs:group ref="mpv:ElemIdElemGroup"/>
  </xs:sequence>
</xs:group>
<!--
    mpv:ResourceFileAttrGroup
    -->
<xs:attributeGroup name="ResourceFileAttrGroup">
  <xs:attributeGroup ref="mpv:ResourceIdAttrGroup"/>
  <xs:attribute ref="mpv:lastURL"/>
  <xs:attribute ref="mpv:byteOffset"/>
  <xs:attribute ref="mpv:xmlPacket"/>
</xs:attributeGroup>
<!--
    mpv:RsourceFileElemGroup
    -->
<xs:group name="ResourceFileElemGroup">
  <xs:sequence>
    <xs:element ref="mpv:DocumentID" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="mpv:InstanceID" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="mpv:LastURL" minOccurs="0" maxOccurs="unbounded"/>
    <xs:group ref="mpv:ElemIdElemGroup"/>
  </xs:sequence>
</xs:group>
<!--
    mpv:SimpleObjectBaseType
```

```
    -->
<xs:complexType name="SimpleObjectBaseType">
  <xs:sequence>
    <xs:group ref="mpv:ResourceFileElemGroup" minOccurs="0"/>
    <xs:element ref="mpv:Related" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="mpv:Rendition" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attributeGroup ref="mpv:ResourceFileAttrGroup"/>
</xs:complexType>
<!--
mpv:CompositeObjectBaseType
-->
<xs:complexType name="CompositeObjectBaseType">
  <xs:sequence>
    <xs:group ref="mpv:ResourceIdElemGroup" minOccurs="0"/>
  </xs:sequence>
  <xs:attributeGroup ref="mpv:ResourceIdAttrGroup"/>
</xs:complexType>
<!--
    ============================================================
    ============================================================

        Third level Object Defintions

    ============================================================
    ============================================================
    -->
<!--
    mpv:Metadata
     -->
<xs:element name="Metadata">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:any namespace="##any" processContents="lax" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="schemaURI" type="xs:anyURI"/>
  </xs:complexType>
</xs:element>
<!--
    mpv:Related
    -->
<xs:element name="Related" type="mpv:RelatedType"/>
<xs:complexType name="RelatedType">
  <xs:complexContent>
    <xs:extension base="mpv:CompositeObjectBaseType">
      <xs:sequence>
        <xs:group ref="mpv:ObjectsList" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="mpv:Rendition" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="hint" type="xs:string"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<!--
    mpv:Rendition
    -->
<xs:simpleType name="RenditionUsageBaseType">
  <xs:restriction base="xs:anyURI">
    <xs:enumeration value="master"/>
    <xs:enumeration value="thumbnail"/>
    <xs:enumeration value="screen"/>
    <xs:enumeration value="subsampled"/>
    <xs:enumeration value="low-res"/>
    <xs:enumeration value="proof"/>
    <xs:enumeration value="draft"/>
    <xs:enumeration value="print"/>
```



```
    <xs:enumeration value="show"/>
    <xs:enumeration value="targetSystem"/>
    <xs:enumeration value="alt"/>
   </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="RenditionUsageType">
   <xs:union memberTypes="mpv:RenditionUsageBaseType xs:anyURI"/>
  </xs:simpleType>
  <xs:element name="Rendition" type="mpv:RenditionType"/>
  <xs:complexType name="RenditionType">
   <xs:complexContent>
    <xs:extension base="mpv:CompositeObjectBaseType">
     <xs:sequence>
      <xs:group ref="mpv:ObjectsList" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="mpv:Related" minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:attribute name="renditionUsage" type="xs:string"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
  <!--
     mpv:VXMP
     -->
  <xs:element name="XMP">
   <xs:complexType mixed="true">
    <xs:sequence>
     <xs:any namespace="##any" processContents="lax" minOccurs="0"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <!--
     ============================================================
     ============================================================

         Top level Object Defintions

     ============================================================
     ============================================================
     -->
  <!--
     mpv:mpv
     -->
  <xs:element name="mpv">
   <xs:complexType>
    <xs:sequence>
     <xs:element ref="mpv:Album" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <!--
     mpv:Album
     -->
  <xs:element name="Album" type="mpv:AlbumType"/>
  <xs:complexType name="AlbumType">
   <xs:complexContent>
    <xs:extension base="mpv:CompositeObjectBaseType">
     <xs:sequence>
      <xs:element ref="mpv:Background" minOccurs="0"/>
      <xs:element ref="mpv:Foreground" minOccurs="0"/>
      <xs:element ref="mpv:Related" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="mpv:Rendition" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="mpv:MarkList" minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
```



```
<!--
   mpv:MarkList
   -->
<xs:element name="MarkList" type="mpv:MarkListType"/>
<xs:complexType name="MarkListType">
 <xs:sequence>
  <xs:element name="idref" type="xs:ID" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attributeGroup ref="mpv:ElemIdAttrGroup"/>
 <xs:attribute name="markType" type="xs:string"/>
</xs:complexType>
<!--
   mpv:Background
   -->
<xs:element name="Background" type="mpv:SeqType"/>
<!--
   mpv:Foreground
   -->
<xs:element name="Foreground" type="mpv:SeqType"/>
<!--
   ============================================================
   ============================================================

       Second level Object Defintions

   ============================================================
   ============================================================
   -->
<!--
   mpv:ObjectsListType
   -->
<xs:group name="ObjectsList">
 <xs:choice>
  <xs:element ref="mpv:StillWithAudio"/>
  <xs:element ref="mpv:AlbumRef"/>
  <xs:element ref="mpv:Audio"/>
  <xs:element ref="mpv:Document"/>
  <xs:element ref="mpv:Still"/>
  <xs:element ref="mpv:StillMultishotSequence"/>
  <xs:element ref="mpv:StillPanoramaSequence"/>
  <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:group>
<!--
   mpv:AlbumRef
   -->
<xs:element name="AlbumRef" type="mpv:SimpleObjectBaseType"/>
<!--
   mpv:Audio
   -->
<xs:element name="Audio" type="mpv:SimpleObjectBaseType"/>
<!--
   mpv:Document
   -->
<xs:element name="Document" type="mpv:SimpleObjectBaseType"/>
<!--
   mpv:Still
   -->
<xs:element name="Still" type="mpv:SimpleObjectBaseType"/>
<!--
   mpv:StillWithAudio
   -->
```

```
<xs:element name="StillWithAudio" type="mpv:StillWithAudioType"/>
<xs:complexType name="StillWithAudioType">
 <xs:complexContent>
  <xs:extension base="mpv:CompositeObjectBaseType">
   <xs:sequence>
    <xs:element ref="mpv:Still"/>
    <xs:element ref="mpv:Audio" maxOccurs="unbounded"/>
    <xs:element ref="mpv:Related" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="mpv:Rendition" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<!--
   mpv:StillMultishotSequence
   -->
<xs:element name="StillMultishotSequence" type="mpv:StillMultishotSequenceType"/>
<xs:complexType name="StillMultishotSequenceType">
 <xs:complexContent>
  <xs:extension base="mpv:CompositeObjectBaseType">
   <xs:sequence>
    <xs:element ref="mpv:Still" maxOccurs="unbounded"/>
    <xs:element name="captureDur" type="xs:string" minOccurs="0"/>
    <xs:element ref="mpv:Related" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="mpv:Rendition" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<!--
   mpv:StillPanoramaSequence
   -->
<xs:element name="StillPanoramaSequence" type="mpv:StillPanoramaSequenceType"/>
<xs:complexType name="StillPanoramaSequenceType">
 <xs:complexContent>
  <xs:extension base="mpv:CompositeObjectBaseType">
   <xs:sequence>
    <xs:element ref="mpv:Still" maxOccurs="unbounded"/>
    <xs:element name="capturePath" type="xs:string" minOccurs="0"/>
    <xs:element ref="mpv:Related" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="mpv:Rendition" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<!--
   mpv:Par
   -->
<xs:element name="Par" type="mpv:ParType"/>
<xs:complexType name="ParType">
 <xs:complexContent>
  <xs:extension base="mpv:CompositeObjectBaseType">
   <xs:sequence>
    <xs:group ref="mpv:ObjectsList" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="mpv:Related" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="mpv:Rendition" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="hint" type="xs:string"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<!--
   mpv:Print
   -->
<xs:element name="Print" type="mpv:SimpleObjectBaseType"/>
<!--
```



```
    mpv:Seq
    -->
  <xs:element name="Seq" type="mpv:SeqType"/>
  <xs:complexType name="SeqType">
   <xs:complexContent>
    <xs:extension base="mpv:CompositeObjectBaseType">
     <xs:sequence>
      <xs:group ref="mpv:ObjectsList" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="mpv:Related" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="mpv:Rendition" minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:attribute name="hint" type="xs:string"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
  <!--
    mpv:Text
    -->
  <xs:element name="Text" type="mpv:SimpleObjectBaseType"/>
  <!--
    mpv:Video
    -->
  <xs:element name="Video" type="mpv:SimpleObjectBaseType"/>
</xs:schema>
```


Appendix VII: MD5 Computation and String Representation

MPV utilizes MD5 as a well-defined high-performance technique for "fingerprinting" content. It plays a central role in the MPV practices for identifying files and content and fixing up broken references.

MD5 is a technique for computing a 128-bit statistically unique identifier based on processing of a byte stream. MD5 was defined in "The MD5 Message-Digest Algorithm", RFC 1321, April 1992, available at http://www.ietf.org/rfc/rfc1321.txt.

VII.1 MD5 Computation

Please refer to the referenced standard [MD5] for a sample implementation. Further information and source code is available at http://userpages.umbc.edu/~mabzug1/cs/md5/md5.html and other web locations. Performance-optimized implementations exist and some CPUs even have instructions tuned to compute MD5 values.

Of most interest to MPV are the definitions of the "body" semantic for MD5-based identifiers. This semantic is file-type specific and defined in more detail in this section.

VII.2 String Representation of a MD5 Identifier in MPV

RFC 1321 [MD5] provides a sample MDPrint() algorithm that prints the identifier as a 32-byte Hexidecimal string. This representation is the only accepted representation in MPV.

The formal definition of the MPV representation of MD5 string values is provided by the following extended BNF:

```
UUID                    = 16*<hexOctet>
hexOctet                = <hexDigit> <hexDigit>
hexDigit =
      "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
      | "a" | "b" | "c" | "d" | "e" | "f"
```



```
        | "A" | "B" | "C" | "D" | "E" | "F"
```

The following is an example of the string representation of a UUID:

```
f81d4fae7dec11d0a76500a0c91e6bf6
```

The following is an example of the printing the string representation of a UUID:

```
static void PrintId (id)
unsigned char id[16];
{
  unsigned int i;

  for (i = 0; i < 16; i++)
      printf ("%02x", id[i]);
}
```

VII.3 Definitions of MD5 "body" Identifiers for Various Media and File Types

[TODO]

Appendix VIII: UUID Computation and String Representation

A universally unique identifier (UUID) format was defined in the Open Software Foundation's Distributed Computing Environment RPC standard also available as ISO-11578, which defines UUIDs in an appendix.

A internet draft was proposed that specifically defines UUIDs. This expired in 1998 and was removed from the standard location at http://search.ietf.org/internet-drafts/draft-leach-uuids-guids-01.txt. Various copies still exist on the internet and are useful defacto standards. The following webpage http://www.ics.uci.edu/pub/ietf/webdav/uuid-guid/draft-leach-uuids-guids-01.txt is an archive of the draft standard and also includes source code for UUID generation both with and without the use of ethernet MAC addresses.

VIII.1 UUID Computation

Please refer to the archive of the draft standard for the sample implementation.

VIII.2 String Representation of a UUID in MPV

The draft specification for UUIDs includes a standard representation of UUID as a string value. This representation uses "-" values to segment the UUID value. Various operating systems and programming tools variously produce and consume UUID string values.

Within MPV, UUID values are represented as 32-byte Hexidecimal strings, as described the sample algorithm. This representation is the only accepted representation in MPV. All comparison of UUID values between MPV elements and UUIDs originating from other souces must process the external UUID to remove all non-Hexidecimal characters prior to comparison.

The formal definition of the MPV representation of UUID string values is provided by the following extended BNF:

```
UUID                    = 16*<hexOctet>
hexOctet                = <hexDigit> <hexDigit>
hexDigit =
      "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```



```
    | "a" | "b" | "c" | "d" | "e" | "f"
    | "A" | "B" | "C" | "D" | "E" | "F"
```

The following is an example of the string representation of a UUID:

```
f81d4fae7dec11d0a76500a0c91e6bf6
```

The following is an example of the printing the string representation of a UUID:

```
static void PrintId (id)
unsigned char id[16];
{
  unsigned int i;

  for (i = 0; i < 16; i++)
      printf ("%02x", id[i]);
}
```


Appendix IX: Typographic Conventions

[TODO] – this is out of date and laxly followed.

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>
```


Appendix X: References

[CSS2]

"Cascading Style Sheets, level 2", Bert Bos, Håkon Wium Lie, Chris Lilley, Ian Jacobs. W3C Recommendation 12 May 1998.
Available at http://www.w3.org/TR/REC-CSS2

[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/

[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”, June 18, 2001, International Imaging Industry Association (I3A) [recently formed by combining the Digital Imaging Group and PIMA].
http://www.i3a.org

[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

[MD5]

"The MD5 Message-Digest Algorithm", RFC 1321, April 1992.
Available at http://www.ietf.org/rfc/rfc1321.txt. Further information and source code available at http://userpages.umbc.edu/~mabzug1/cs/md5/md5.html

[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

[MIMETYPES-REG]

IANA official registry of MIME media types
Available at http://www.isi.edu/in-notes/iana/assignments/media-types/media-types

[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.

[SMIL20]

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

[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

[URI]

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

[UCS-2]

16-bit encoding of ISO 10646, commonly known as the Unicode character set.

[UTF-8]

Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

[W3C-NSURI]

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

[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

[XMP-FW]

"*XMP – Extensible Metadata Platform 14 Sept 01"* , Copyright 2001 Adobe Inc,
Available at http://xml.coverpages.org/XMP-MetadataFramework.pdf. Also at
http://partners.adobe.com/asn/developer/xmp/download/docs/MetadataFramework.pdf

[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/

Appendix XI: ToDo and Things to remember & discuss

- OSTA needs to register the "osta-org" naming authoring with IANA for use with its URN qualified names, such as "urn:osta-org:mpv:dsig:all:md5:342603EC-D93E-DE34-93CD-98B9A6C98DDC".
- There are lots of [TODO] sections and areas left in the document.

