-OSTA.org-
Optical Storage
Technology Association

MultiPhoto/Video

Manifest, Metadata and Practices for Digital Photo-Video Collections

Core Specification

Revision 0.90
Working Draft

June 28th, 2002

IMPORTANT NOTICE

This document is a working draft for review by OSTA 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 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/MPVCore-Spec-0.90WD.PDF

POINTS OF CONTACT

OSTA	*Technical Content*
David Bunzel OSTA President Tel: +1 (408) 253-3695 Email: dbunzel@osta.org http://www.osta.org *MultiPhoto/Video Website* http://www.osta.org/mpv/	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

(a) This document is an authorized and approved publication of the Optical Storage Technology Association (OSTA). The specifications are the exclusive property of OSTA but may be referred to and utilized by the general public for any legitimate purpose, particularly in the design and development of writable optical 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 and appropriate reference is made to the origin of the material. 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 unauthorized and specifically prohibited.

(b) 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, copyright, mask work right or trade secret right). By publication of this document, no position is taken by OSTA with respect to the validity or infringement of any patent or other intellectual property right, whether owned by a Member or Associate of OSTA or otherwise. OSTA hereby expressly disclaims any liability for infringement of intellectual property rights of others by virtue of this OSTA document, nor does OSTA undertake a duty to advise users or potential users of OSTA documents of such notices or allegations. OSTA 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 expressly disclaims any intent to promote infringement of any intellectual property right by virtue of the evolution, adoption, or publication of this OSTA document.

(c) This document is a specification adopted by Optical Storage Technology Association (OSTA). This document may be revised by OSTA and users are advised to obtain the latest version. It is intended solely as a guide for companies interested in developing products which can be compatible with other products developed using this document. This document is provided "AS IS". OSTA makes no representation or warranty regarding this document, and anyone using this document shall do so at their 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 direct, indirect, exemplary, incidental, proximate or consequential damages or expenses arising from the use of this document for any reason whatsoever, even if OSTA is advised of a particular use of this document. This document defines only one approach to compatibility, and other approaches may be available in the industry.

(d) 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 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 representation, exchange, 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 [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, such 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 supports use of established specifications and standards.

The development and promotion of MultiPhoto/Video is sponsored by the Optical Storage Technology Assocation (OSTA). The specification development and promotion process is open to all members; all organizations and individuals are welcomed as members. The association includes over 50 member companies from all over the world that produce products that collectively represent a majority marketshare in mainstream recordable optical storage categories.

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

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
- **Disc Archive Profile**: key task: interoperability of photo archives on recordable optical discs
- **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

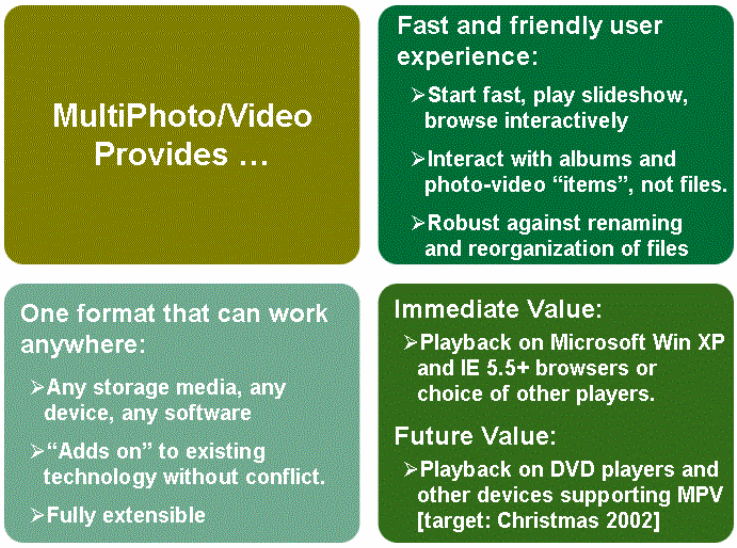
Underlying all profiles is the "Core", which defines the overall framework of all MPV profiles. The Basic and Presentation Profiles, for example, build on the Core and, when implemented in consumer electronics devices like DVD players or in application software, can provide compelling playback of photo-video slideshows and interactive browsing of photo-video content. It can also facilitate interchange of photo-video content between applications.

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 may be augmented by information from various profiles. For example, the Presentation profile provides information that may be used by player applications and devices to provide an attractive playback 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 without royalty from the Optical Storage Technology Assocation (OSTA). OSTA is an industry association promoting the use and interoperability of recordable CD and DVD discs in computer and consumer electronics devices.

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 provides for the basic definition of collections of photo-video content. A Presentation Profile extends the specification for an enhanced presentation experience of interactive browsing and slideshow playback provided by DVD players and other devices and media player software. Additional profiles will be developed subsequently.

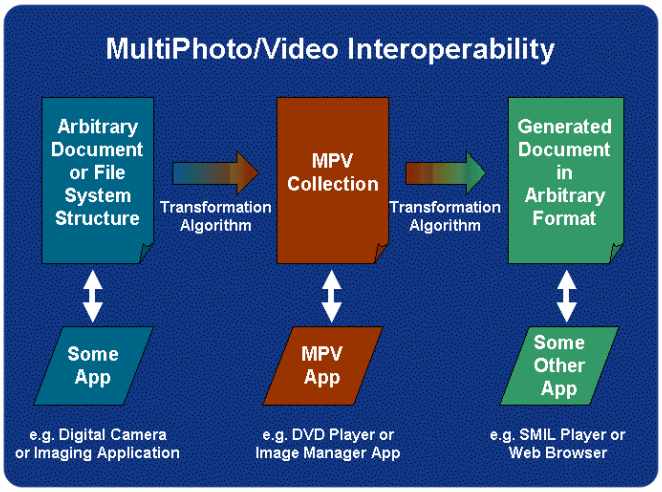
The MPV specification will further promote adoption of current and new categories of digital imaging products by enhancing ease-of-use and interoperability of photo/video content collections and applications. The format enables an end-user experience that starts fast, is highly interactive, provides for playing and editing collections of photo/video content, never requires the device or application to reveal 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 [SMIL20] and I3A's DIG35 [DIG35-2001]. MPV is compatible with and supports the DCF [DCF-1999]and Exif [Exif2002] specifications that are widely used in digital cameras.

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 as "ALBUM.PVM" and zero or more dependent files.



MPV is well suited as an intermediate format for exchange of photo-video content collections across applications, devices, and services. Some applications may also choose to use it as the primary format for storing their own data. MPV is structured such that it may be used with reasonable efficiency as a lightweight textual database to maintain metadata and related information for hundreds to the low thousands of photo-video content 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 from OSTA. No royalty is charged by OSTA 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 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, a logo program for compliant products, and a website. These materials and procedures are made available and administered by OSTA at a modest cost. OSTA charges no royalty for use of the specification or logo. In addition, some sample open-source code implementations of key steps in processing MPV content may be contributed by interested parties.

Chapter 2: Key Concepts of the MPV and Related Specifications

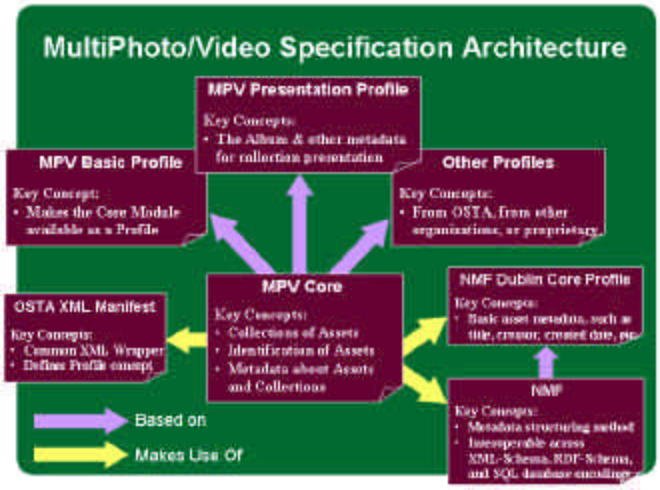
2.1 MPV Specification Architecture

MPV is not just one specification, it is a family of related specifications. This architecture results from several principle objectives:

- MPV should be highly extensible, allowing anyone to create proprietary or open extensions to MPV without modifying the MPV specficiation itself.
- Adding extensions should not damage interoperability of the basic collection information.
- Specifications that are fundamentally separable concepts should be separated. This allows each specification to be used and evolve independently of each other.
- MPV should not define alternate representations where mainstream representations exist.

These principles result in the following characteristics of the MPV and related specifications.

- The **MPV Core** is the essence of the MPV specification. However, it cannot be used by itself; it must always be incorporated into a Profile, which is the basic unit of extension in MPV.
- The **MPV Basic and Presentation Profiles** are extensions that utilize the MPV Core. **Other Profiles** are extensions organized in exactly the same way.
- MPV makes use of the **OSTA XML Manifest**, which defines the Profile concept.
- MPV makes use of the **NMF Specification** for structured representation of arbitrary metadata. NMF is a wholly separate concept.
- MPV recommends use of the NMF-encoding of **Dublin Core**, a separate and widely adopted specification for representing basic metadata about assets of all kinds.



2.2 Profiles, Schema and Practices

The MultiPhoto/Video specifications contain the following kinds of content.

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 informative definition; in the case of discrepancy, the specification document supercedes the machine-readable schema.

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.

Profiles are a set of Schema and Practices and additional content and are the principle 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 schema and practices that are necessary for the key tasks targeted by the profile.

Referenced Specifications are other specifications used by the MPV specifications. These specifications may be from OSTA or other organizations.

2.3 MPV Core and the MPV Basic and Presentation Profiles

Profiles represent the basic unit of extension within MPV. Profiles define schema and practices that are available for addressing a given task. Typically, profiles define one or more top-level elements in a manifest plus various additional metadata.

The MPV Core specification [MPV-Core] and the MPV Basic [MPVBasic] and Presentation Profiles [MPV-Basic, MPV-Pres] were all developed at one time, and the Core documentation makes reference to the Basic and Presentation Profiles. However, nothing about the Basic and Presentation Profile specifications and implementation is treated specially in the Core. In other words, the Basic and Presentation Profiles play by the same rules that new, as yet undefined additional profiles must play by.

2.3.1 MPV Core

An overview of the MPV Core is described in detail in a following chapter. It provides the three core concepts of Collections, Metadata, and Identification. No profile is considered an MPV-related Profile unless it makes use of the MPV Core, either directly or indirectly.

MPV Core establishes the use of the OSTA XML Manifest for as the XML document file format for MPV content.

2.3.2 MPV Basic Profile

The MPV Basic Profile simply makes available the MPV Core as an MPV Profile. This is necessary since Profiles are the only unit of incorporation into an OSTA XML Manifest. The MPV Basic Profile provides two top-level elements in a manifest: AssetList and MarkedAssets.

Primary tasks that users of the MPV Basic Profile can accomplish include definition of collections of assets by reference and grouping of those assets into distinguished sets using MarkLists. These basic tasks are the essence of what MPV provides.

ASSETLIST

The AssetList is the basic unit of collection representation in MPV. Assets themselves may only be defined in an AssetList. Only one AssetList is allowed in a manifest.

MARKEDASSETS

The MarkedAssets element may contain MarkLists that make reference to assets. Standard marklist types like "primary" and "selected" provide for interchange of lists of distinguished assets. Multiple MarkLists may be present in the MarkedAssets element.

2.3.3 MPV Presentation Profile

Primary tasks for the MPV Presentation Profile are to provide albums that allow the user apply presentation information to a collection of assets. The primary usages are to play a slideshow, interactively browse the primary assets, or selectively print the album's contents.

The MPV Presentation Profile provides very basic presentation information that emphasizes use by a devices and applications with a broad range of presentation capabilities and significant amounts of application-level control over presentation behaviour. Additional profiles may define much richer or more tightly scoped presentation behaviours.

ALBUM, ALBUMREF

An album is a presentation-oriented view of the asset list and the most common representation of an MPV collection exposed to users . It is an ordered set of references to assets in asset lists. Albums can link to other albums. Multiple albums can be grouped together in one file or isolated in separate files. AlbumRefs can be to albums in the same or different manifests and local or remote. Albums may have renditions, related documents and mark lists of their own.

FOREGROUND, BACKGROUND

Users interact with Album-level Foreground and Background assets; they and the Album's Related Documents are conceptually the primary assets in a collection. Typically, users interact most with foreground assets while background assets are secondary and fewer. Foreground and background assets 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.

PRESENTATION CONTROL

The overall approach for representing presentation information is compatible with SMIL, a powerful XML format for representing presentations from the World Wide Web Consortium (W3C). MPV Presentation Profile provides a very constrained set of properties compatible with 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 the Presentation Profile is not extensive however, many other implementations can be contemplated. For example, compelling playback of MPV documents in modern web browsers is readily accomplished.

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

2.4 XML Usage

XML LEVERAGE

MPV content is well-formed XML. This allows the MPV document to be processed using standard XML processing tools and environments. For example, when opened in the Microsoft Internet Explorer 5.5 and above web browser, a 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. MPV can also be easily utilized within other XML specifications.

NAMESPACES

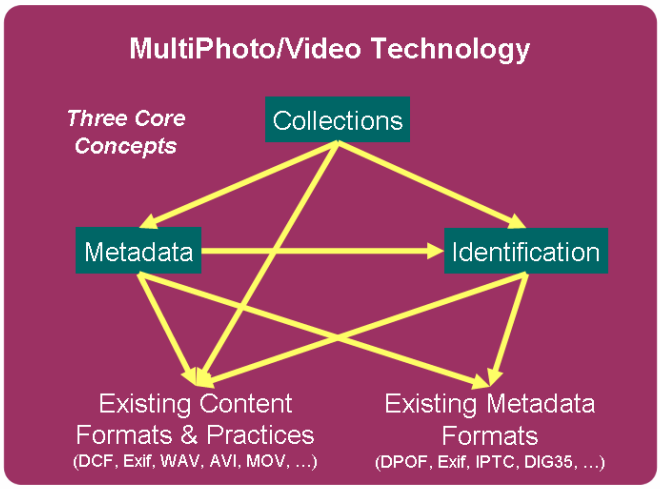
XML namespaces are a means to allow XML elements of the same name that exist in different schema to co-exist within the same document.

MPV requires the use of namespaces. By convention, all elements and attributes in MPV are used with their prefixes in all XML encodings. MPV does not support namespace-unaware processing. Most modern XML tools support namespace-aware processing.

Chapter 3: Key Concepts of the MPV Core

MultiPhoto/Video has some key concepts and approaches.

- The Core provides the three core concepts of Collections, Metadata, and Identification.
- Profiles are the unit of use and extension of MPV technology and are focused on addressing a specific set of tasks. Profiles include XML schema that specify content and best practices that guide its use. Over time, multiple profiles will be developed.

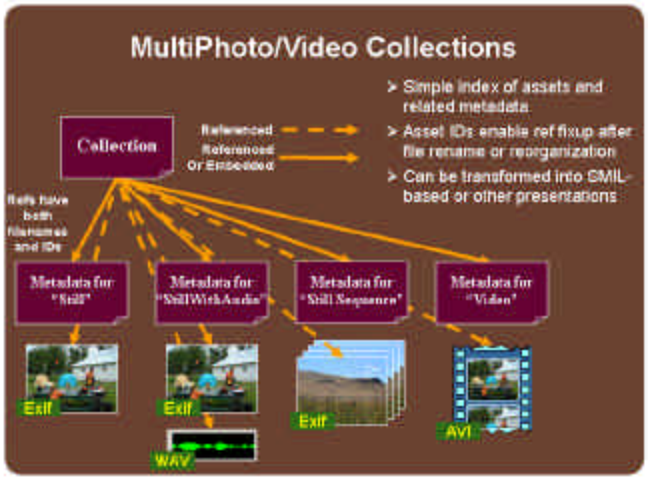


3.1 Collections

Collections are assembled using a few core concepts.

MANIFEST

The Photo/Video Manifest groups all the MPV components into a single XML document. A Photo/Video Manifest contains a least one asset list or manifest links. It may contain zero or more albums and mark lists, which provide views onto those assets. In typical usage, a Photo/Video Manifest is stored in a stand-alone file.



ASSET LIST

An asset list is an unordered set of assets that each have a unique local identifier in the MPV collection. It is the only place photo-video assets may be defined as part of the collection – everything else in MPV is

metadata and references to assets. A MPV collection contains at least one asset list or link to a manifest in another file. By analogy, an asset list may be considered a table of assets in a database and the id is the foreign key. Another analogy would be to the entries in a Unix file system inode.

MARK LIST

A mark list is an ordered set of asset references and associated metadata and mark type. It is provided as a basic mechanism for MPV profiles to reference assets. The special "primary" mark type identifies which assets in the asset list are considered to be top-level assets in a collection and gives them an order. Other predefined mark types are "selected" and "hidden"; the mark type is fully extensible.

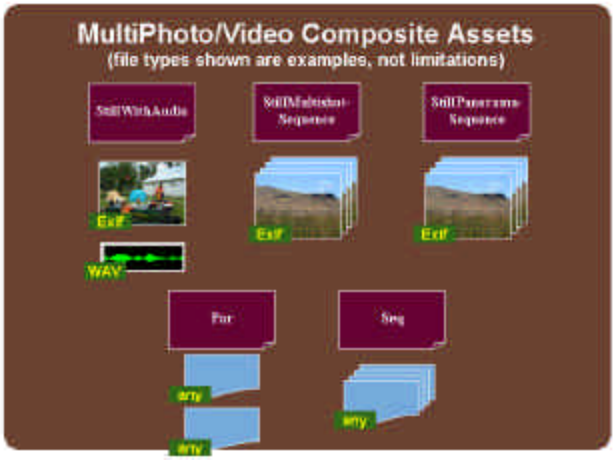
SIMPLE MEDIA ASSETS

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

- Still
- Video
- Audio
- Text
- Print
- Document
- ManifestLink

Any media asset may contain renditions and related documents.

COMPOSITE MEDIA ASSETS



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

- StillWithAudio
- StillMultishotSequence
- StillPanoramaSequence
- Par
- Seq

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

RENDITIONS

Any simple or composite media asset and even an album may have one or more renditions. The asset itself is the master rendition and is defined implicitly by the asset definition. Renditions other than the master rendition are derived versions of the original media asset. The relationship between the master rendition and the derived renditions is captured as an attribute and arbitrary additional metadata. The derived version may be direct, as in a screen resolution image rendition of a high-res image, or indirect, as in a video stream or print rendition of a collection. However, generally speaking, Renditions do not represented edited versions of a master; use the Related asset to express an editing relationship such as cropped or color adjusted.

RELATED DOCUMENTS

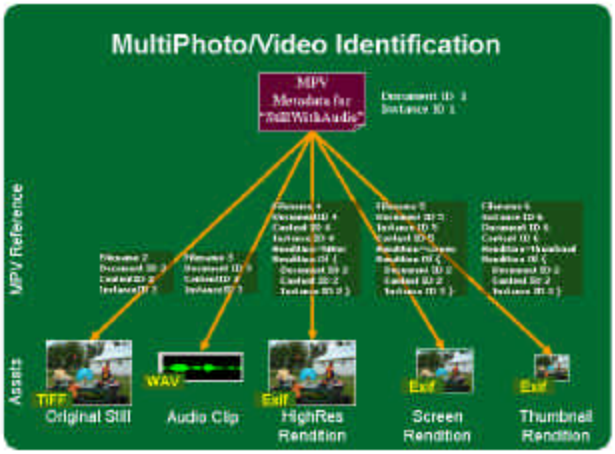
All simple and composite media assets and an album may have one or more related documents. Such documents may have any relation to the media asset, including other assets used in constructing the asset or additional metadata related to the asset. A common usage is to express an editing relationship, such as the original asset that was cropped to get the master asset.

3.2 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 assets in a collection are identified by two or more identifiers. There are five kinds of identifiers:

- id – a unique identifer local to the MPV XML document in which it is used when the element is referenced by another element.
- lastURL – last known location
- instanceID – unique identifier for an asset
- documentID – the same for all renditions
- contentID – computed using the content as input; statistically unique for each asset.



More than one of the lastURL, docmentID, and contentID 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.

The lastURL can be a local filename or remote URL and is the primary means for referencing assets. 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. The name “lastURL” highlights this tenuous relationship, as in “last known to be at this URL.”

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 assets 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 readily available 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. Sample source code is readily available.

3.3 Metadata

MPV IS METADATA, NOT DATA

Metadata is data about data and MPV defines metadata that is used to describe photo/video asset collections and related information. MPV does not contain the actual asset data files themselves. MPV is "add on" metadata that is located outside the asset files themselves and does not require the asset files to change.

COLLECTION METADATA

As described earlier, MPV provides the schema to represent collections and identifiers for photo/video asset collections. For every aspect of the collection, MPV provides mechanisms for associating additional metadata as described in this section. This provides a straightforward extension mechanism for almost every aspect of MPV.

NMF-STRUCTURED METADATA

MPV makes use of a format called Normalized Metadata Format [NMF]. NMF is an approach to structuring metadata that has the advantage of being mechanically interchangeable across several important metadata encodings: XML Schema-based, RDF-Schema-based, and SQL database tables. NMF can be used to structure any kind of metadata and this is the preferred mechanism for representing metadata in MPV because it provides for ready interchange across supported encodings. NMF metadata schema and content are validatable using commonly available XML-Schema-based tools.

MPV recommends that new metadata schema be designed using this format. In addition, existing schema may be encoded in this format as well. One such schema is Dublin Core [DC-NMF], a widely adopted schema for describing asset properties such as title, creator, created date, etc. MPV recommends use of DC for representing this information in MPV documents.

ARBITRARY METADATA

MPV provides a mechanism to embed any well-structured XML metadata within an MPV document. This provides for ready use of existing schema from any source. This approach is simple and entirely appropriate for many situations, but it does not provide the structured approach and ready interchange of NMF-structured metadata.

EMBEDDED METADATA

MPV metadata is external to assets themselves. Most assets also may have embedded metadata. In this case, the question of precedence arises. This cannot be dictated uniformly. However, generally speaking, MPV recommends that metadata in MPV takes precedence over metadata embedded in assets when an application is reading metadata about an asset; this cannot apply when the external metadata is in conflict with intrinsic asset properties, such as height, width, or colorspace. When creating new asset metadata to be stored in an MPV document, there is no recommended behaviour. Some applications may choose to write-through metadata to the asset's existing metadata

format(s) when a given data item is not yet present in the asset file; others may write-through the metadata in a new way; others may not write-through at all.

Metadata for composite media assets often cannot reside in the basic media assets because it spans multiple asset files. This type of metadata may be stored in the MPV document.

Chapter 4: Overall Required and Best Practices

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

4.1 Namespace Usage

MPV requires namespace-aware processing. This is available with most modern XML tools. There is no namespace-unaware encoding.

MPV requires that the namespace prefix is used on all elements and attributes of MPV-specific content. Default namespace usage is not permitted. This does not apply to embedded NMF metadata, which does use default namespaces.

4.2 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.

4.3 Character Set

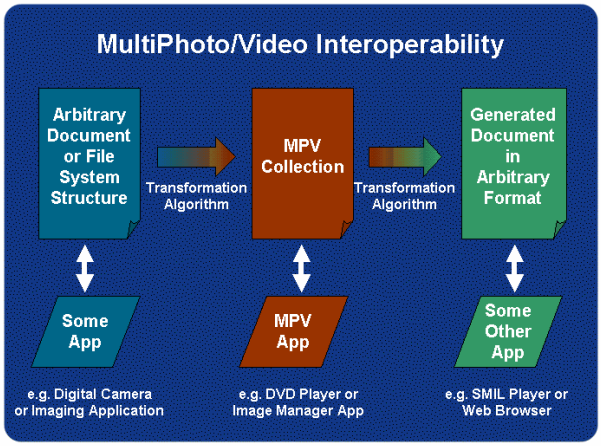
All MPV content shall use the UTF-8 character set [UTF-8]. Content is further constrained by XML allowable characters.

4.4 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.

4.5 MPV Interoperability

MPV collections and documents exist within a complex and dynamic ecosystem of existing and new applications, devices, services, and formats. MPV can be used as the primary format in which an application stores its state and represents not only photo-video collections but also hosts application-specific data. However, this is not required.



MPV is simple enough and has enough flexibility and extensibility that it is often straightforward to transform to and from the MPV representation. A primary result is improved interoperability with other products. 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.

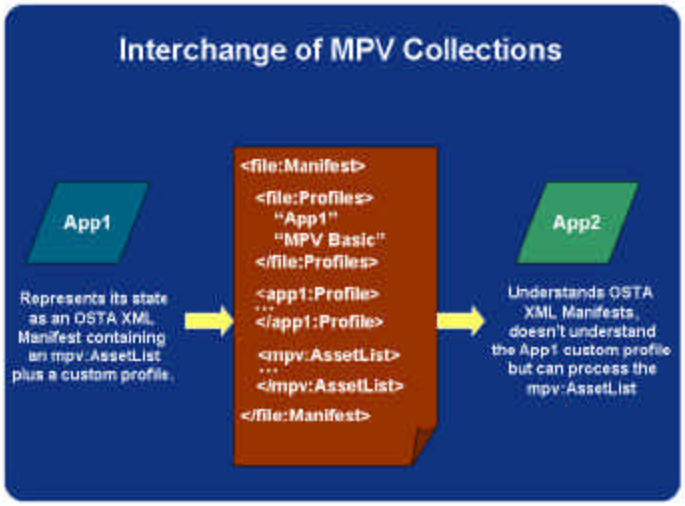
As an interchange format, MPV's AssetList concept provide a means for reliably exchanging basic collections and associated metadata. This simple ability is very valuable, as no well established interchange format for photo/video collections exists.

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. Alternately, IE's ability to load and process XML documents via a DOM (Document Object Model) provides browser-based scripts complete access to the MPV manifest's content and allows it to be rendered for playback.

4.6 OSTA XML Manifest Usage

The OSTA XML Manifest provides a common XML document wrapper element, defines the concept of Profiles, and defines the mechanism for embedding content from multiple Profiles within the same XML document without collision.



All MPV profiles are profiles within an OSTA XML Manifest. Because all MPV-based profiles have the concept of the AssetList top-level element in a manifest, all MPV-aware processors of an OSTA XML Manifest can process the AssetList, even if the additional content provided by profiles in the manifest cannot be understood.

This ability provides the essential mechanism for interchange of MPV collections across a diverse set of applications – all applications will understand the AssetList, even if they do not understand anything else.

4.7 MPV Extensions

Extensions can be made to MPV content in several forms.

4.7.1 Profiles

Profiles represent the basic unit of extension within MPV. All the related extensions are gathered together and defined as a MPV Profile, which represents a set of schema and practices. Profiles define schema and practices that are available for addressing a given task. Typically, profiles define one or more top-level elements in a manifest plus various additional metadata.

4.7.2 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 nmf:Metadata 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 uses an open content model (one with no constraints on the content that can occur there) for the content in the following elements:

- mpv:Metadata element
- nmf:Metadata element
- the top-level Manifest element (see the OSTA XML Manifest specification [MANIFEST])

4.7.3 MPV Schema Extensions

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

4.7.3.1 Elements

New assets (along with the references to those assets) can be defined for use in profiles. Once these assets are defined (and their schemas made available to the MPV processor), they can be used wherever the existing assets (and the references to those assets) can occur in MPV content

These new asset types must be defined as follows:

- define a complexType that derives from mpv:AssetRefBaseType.
- define an element that is an instance of the derived type and is substitutable for mpv:AssetRefBase.

If the asset is a simple asset, it must:

- define a complexType that derives from mpv:SimpleAssetBaseType
- define an element that is an instance of the derived complexType and is substitutable for mpv:SimpleAssetBase.

If the asset is a composite asset, it must:

- define a complexType that derives from mpv:ComplexAssetBaseType
- define an element that is an instance of the derived complexType and is substitutable for mpv:ComplexAssetBase.

4.7.3.2 Attributes

Attributes unknown to the MPV processor are allowed in MPV content; they may be in any namespace including mpv defined namespaces (see mpv:AnyAttrGroup attributeGroup). The MPV processor may choose how to handle them so long as general processing of the MPV document is not aborted. This requirements allows content from unknown extensions and future versions to be part of the MPV document without disturbing general processing by processors unaware of them.

4.8 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.

Significantly, MPV processors MUST support the DOCTYPE and IMPORT constructs that allow XML content to be inserted inline from one file into another. This is supported by most commercially available and open source parsers.

A variety of commercial and open source tools are available for processing XML content. For example, many firmware and application software implementations utilize expat [EXPAT], a C language open source XML parser that is namespace aware.

4.9 Processing Unknown Elements and Attributes

4.9.1 Schema-Unaware Processors

Schema-unaware processors will be the typical type of processor performed by most or all firmware implementations and many software application implementations. They do not have access to the XML-Schema defining MPV.

The recommended practice for schema-unaware processors 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 assets, which can be processed separately.

4.9.2 Schema-Aware Processors

Schema-aware processors exist and are practical to deploy using currently available XML-Schema processing tools. They do have access to the XML-Schema defining MPV.

One of the most important aspects of schema-aware processing is how unknown content is handled. There are two types of unknown content that can be encountered by an MPV processor. One is content that occurs in one of the locations where MPV has specified that any kind of content can be specified. The other is content that occurs in locations that require that the content conform to specific requirements such as being derived from an abstract base type.

There are three levels of validation that schema-aware processors may apply to MPV content. They are:

None

No validation is applied to the content, in which case all content will appear to be valid according to the schemas.

Lax

Any content for which schemas are available will be validated. Content for which schemas aren't available will be ignored.

Strict

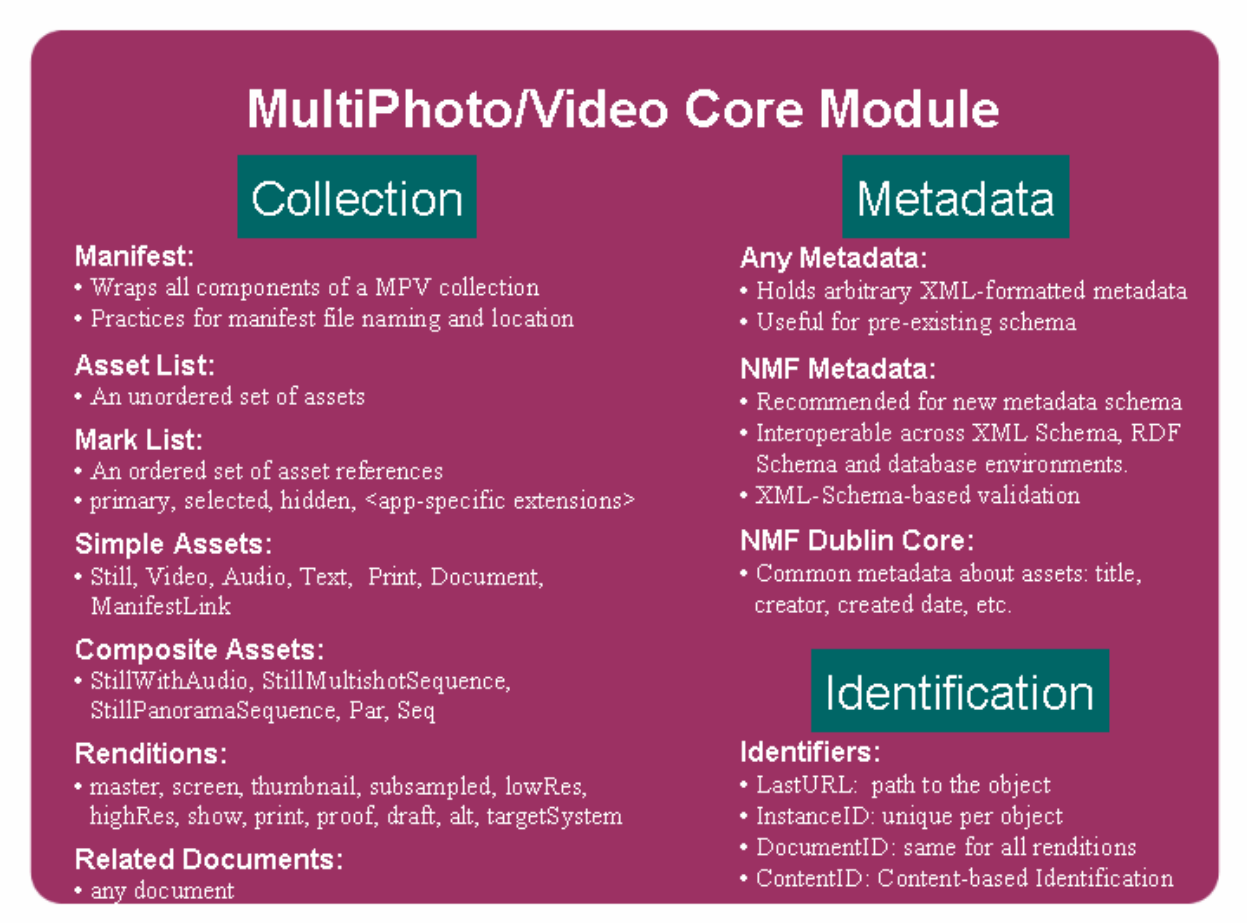
Any content for which schemas are not available will be treated as invalid.

These differing levels of validation are controlled in part by the processing of the xs:any type, which is used in several areas of MPV. This is readily configurable.

Chapter 5: MPV Core Schema, Part 1: Identification

5.1 Module Introduction

The MultiPhoto/Video Core provides for the definition of collections of media assets. It is the essential core of the MPV specification. The Core has the following core components:



5.2 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	Schema Location	Conventional Namespace Prefix
Core	http://ns.osta.org/mpv/1.0/	core/impl/core.xsd	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 ="http://ns.osta.org/mpv/1.0/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="qualified">
```

USER-VISIBLE SCHEMA ELEMENTS

The following schema elements are the user-visible elements used when creating a document with core MPV content. When authoring or reading MPV documents, these are the elements that will be encountered. Note that attribute names are not listed.

Collection Mgmt
AssetList
MarkList

Assets
Audio
AudioRef
Document
DocumentRef
ListRef
ManifestLink
ManifestLinkRef
Par
ParRef
Print
PrintRef
Related
Rendition
Seq
SeqRef
Still
StillRef
StillMultishotSequence
StillMultishotSequenceRef
StillPanoramaSequence
StillPanoramaSequenceRef
StillWithAudio
StillWithAudioRef
Text
TextRef
Video
VideoRef

Identification
ContentID
DocumentID
LastURL

Metadata
mpv:Metadata
nmf:Metadata

UNDERLYING SCHEMA CONTENT

The following schema elements are secondary types used in the formal description of the MPV core schema to define the user-visible elements and attributes. They are not used directly by name in MPV documents.

Elements	Groups	Complex types	Simple types	Attr. groups
AssetRefBase	AssetChoiceGroup	AssetListType	FilesystemBaseType	ElemIdAttrGroup
CompositeAssetBase	AssetRefChoiceGroup	AssetRefBaseType	FilesystemType	ResourceFileAttrGroup
ListRefBase	ElemIdElemGroup	AssetRefListBaseType	MarkType	ResourceIdAttrGroup
ManifestChildBase	RelationsElemGroup	CompositeAssetBaseType	MarkTypeBaseType	
SimpleAssetBase	ResourceFileElemGroup	ListRefBaseType	RelationshipBaseType	
	ResourceIdElemGroup	ManifestChildBaseType	RelationshipType	
		ManifestChildType	RenditionUsageBaseType	
		MarkListType	RenditionUsageType	
		ParType		
		RelatedType		
		RenditionType		
		SeqType		
		SimpleAssetBaseType		
		StillMultishotSequenceType		
		StillPanoramaSequenceType		
		StillWithAudioType		

5.3 Group: mpv:AnyAttrGroup

MPV allows attributes above and beyond those defined in the element schema to be used for all mpv elements. This capability is implemented via an XML Schema mechanism called xs:anyAttribute. The following attribute group is defined by MPV which should be included as part of the content model of any element that will be used in an MPV document. This attribute group is included in the various base types defined by MPV so it will be part of most type's content model without having to be explicitly included in the type definition.

attributeGroup **AnyAttrGroup**

namespace http://ns.osta.org/mpv/1.0/

used by
- elements: **LastURL Metadata**
- complexTypes: **AssetRefBaseType AssetRefListBaseType CompositeAssetBaseType ListRefBaseType RelatedType RenditionType**
- attributeGroups: **ResourceFileAttrGroup ResourceIdAttrGroup**

attributes

Name	Type	Use	Default	Fixed

source

```
<xs:attributeGroup name="AnyAttrGroup">
  <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:attributeGroup>
```

5.4 Resource Identification

Separate media asset data resources, such as image files, video files, audio files, text files, etc, are organized into collections using MPV. Identifiers are the means by which references are made between a collection and the media asset data it references and between elements of the collection itself.

Because the MPV collection is separate from the actual media asset data, the robustness of the references in the MPV collection is of critical importance; these references should be able to withstand renaming, reorganization , and even the minor editing of the media asset files themselves. Without this ability, MPV collections would be too fragile to be useful in many settings.

Consequently, MPV makes a substantial effort to enable robust identification of referenced media asset data. All media asset resources in MPV can be identified robustly uing a variety of identification techniques, whose values are stored as attributes and elements of MPV media assets in the collection.

All basic and composite media assets in a collection are identified by two or more identifiers. There are five kinds of identifiers overall:

- id – an XML-style identifier for reference to elements in an XML document. This identifier is unique within its document but not globally unique
- instanceID – globally unique identifier for every asset
- documentID – a globally unique identifier that is the same for all renditions
- contentID – a globally unique identifier for every asset based on the asset's content
- 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 assets 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. They are vulnerable to being lost or damaged and 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. However, some algorithms are vulnerable to changes in the content by even one bit.

Arbitrary identifiers are typically computed using an algorithm 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 preferentially should be supported first because of its simplicity and universal applicability. MD5 computes a 128-bit hash of the byte values in an arbitrary set of content.

This identification schema may be expressed in native MPV syntax or alternately in NMF-encoded syntax. Both schema are defined below.

5.5 Unique Identifers – Attributes mpv:id, mpv:instanceID, mpv:documentID, mpv:contentID; Elements <mpv:DocumentID>, <mpv:ContentID>

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

Example:

```
...
  <mpv:Still mpv:id="ID000100" mpv:instanceID="AC937BCFA3B340da971BAF09B17DBC324"
             mpv:lastURL="The name of the image.jpg" />

  <mpv:Still mpv:id="ID000200" mpv:instanceID="AC937BCFA3B340da971BAF09B17DBC324">
    <mpv:ContentID>urn:osta-
 org:mpv:dsig:md5:all:EF886AEFA3B340da971BAF09B17DBC122</mpv:ContentID>
    <mpv:LastURL mpv:filesystem="NTFS">The name of the image.jpg</mpv:LastURL>
    <mpv:LastURL mpv:filesystem="ISO9660-1">THE_NAME_.JPG </mpv:LastURL>
  </mpv:Still>
```

. . .

ID: XML ELEMENT IDENTIFIER

MPV uses this value to identify an XML element in an MPV document. "mpv:id" is locally unique within the MPV document. Implementations may also make mpv:id globally unique, such as through the use of a UUID value. The "mpv:id" attribute is widely used because all references to assets make use of the asset id.

INSTANCEID: INSTANCE IDENTIFIER

MPV uses this value to identify any referenced asset, such as an image file. When practical and possible, the instanceID value used in an MPV document should be extracted from the referenced asset according to the practices of metadata formats used by that type of asset, such as Exif 2.2 for images [Exif2002]. If not already present, the instanceID should be embedded in the referenced asset in accordance with industry practice and other specifications.

The typical instanceID value is a UUID, a 128-bit identifier readily generated by most modern operating systems. In MPV, by convention, UUID string values do not use dash separators and are represented as 32-character strings.

DOCUMENTID: 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.

CONTENTID: CONTENT IDENTIFIERS

Content identifiers are computed from the actual content of the asset; MPV considers these to be "digital signatures". These type of identifiers are attractive because some kinds can be computed without any modification to the asset, making them attractive in situations where the asset may not be modified.

Many content identifier computation methods exist. MPV distinguishes the MD5 algorithm as the basic algorithm that preferentially should be supported first by a processing application. 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.

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:md5:all

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

In particular, the body MD5 signature is recommended for JPEG and Exif images. The signature is computed only on the image pixel data, allowing for the file's metadata blocks to be edited without damaging the digital signature.

Also note that composite assets may also have an contentID. For the MD5 signature, the algorithm to generate the composite signature is to concatentate the ordered list of sub-assets. For a StillWithAudio asset, this would be done by streaming first the still image and then the audio through the algorithm. This makes it possible to have a contentID for both leaf and composite assets with almost no additional computational overhead.

IDENTIFIER ATTRIBUTES SCHEMA

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

attributes **id, instanceID, documentID, contentID**

namespace	http://ns.osta.org/mpv/1.0/
source	<xs:attribute name="id" type="xs:ID"/> <xs:attribute name="instanceID" type="xs:anyURI"/> <xs:attribute name="documentID" type="xs:anyURI"/> <xs:attribute name="contentID" type="xs:anyURI"/>

id

MPV uses this value to identify any referenced XML element in an MPV collection. It must be locally unique within the XML document that contains it.

instanceID

An identifier that uniquely identifies the asset. Typically retrieved from the asset.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

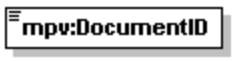
An identifier that is different for each rendition, allowing the rendition to be uniquely identified, using the value syntax defined below.

None of the attributes is required.

IDENTIFIER ELEMENTS SCHEMA

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

element **mpv:DocumentID**

diagram 

namespace http://ns.osta.org/mpv/1.0/

type **xs:anyURI**

used by groups **mpv:ResourceFileElemGroup mpv:ResourceIdElemGroup**

source <xs:element name="DocumentID" type="xs:anyURI"/>

element **mpv:ContentID**

diagram 

namespace http://ns.osta.org/mpv/1.0/

type **xs:anyURI**

used by groups **mpv:ResourceFileElemGroup mpv:ResourceIdElemGroup**

source <xs:element name="ContentID" type="xs:anyURI"/>

<mpv:DocumentID>

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

For broadest compatibility, all MPV instanceID values SHOULD be UUID-style unique ids encoded as 128-bit UUIDs in 32-hexcharacter string format, without hyphens ("-"). Only one instanceID value is permitted.

DOCUMENT ID AND CONTENT ID VALUE SYNTAX SPECIFICATION

These MPV identifiers combine the type of identifier and its value in the value string. This allows a variety of identification algorithms to be applied. A processing application must be able to interpret the algorithm string in order to accurately regenerate or extract the identifier from a candidate asset.

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:<algorithm>:<params>:<value>

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

In MPV, the MD5 algorithm is defined to use the string “md5” in the identifier value string. It has several parameter values:

urn:osta-org:mpv:dsig:md5:all:<value>

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

urn:osta-org:mpv:dsig:md5:body:<value>

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:md5:body:EF886AEFA3B340da971BAF09B17DBC122"`

urn:osta-org:mpv:dsig:md5:head:<byte count>:<value>

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:md5:head:30000:EF886AEFA3B340da971BAF09B17DBC122"`

urn:osta-org:mpv:dsig:md5:tail:<byte count>:<value>

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:md5:tail:30000:EF886AEFA3B340da971BAF09B17DBC122"`

5.6 Location Identifiers – Attributes mpv:lastURL, mpv:byteOffset, mpv:leaseID, mpv:leaseDur, mpv:leaseExpiresDate; Element <mpv:LastURL>

Assets can be qualified according to a path to last known location. 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. A fragment identifier in the path denoted by a "#" and an identifier string can refer to embedded content in an asset. Alternately or additionally, hints can be given as to the byte offset within a file to find the specific data.

However, lastURL and byteOffset 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 and byteOffset 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.

To be robust against byteOffset, MPV provides the fragment identifier. The fragment identifier syntax is defined on a per-file-type basis and can allow access to tagged content within a file without specifying its exact location.

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 9660-1, Joliet, HFS, and UDF.

Multiple lastURL values can also be used to express multiple locations and pathnames where an asset can be found. For example, both relative and absolute pathnames to an asset may be provided, allowing the manifest file either to move with the assets or separate from the assets and yet always be able to find the assets.

SCHEMA

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

attributes **lastURL, byteOffset, xmlPacket, leaseExpiresDate, leaseDur, leaseID**

namespace	http://ns.osta.org/mpv/1.0/

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

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 of longest filename to shortest. 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.

Note: The use of the fragment identifier, as discussed in section 5.7 Identification of embedded assets, is recommended over the use of byteOffset to refer to embedded content in an asset, but byteOffset is provided for its expressive power. ByteOffset does not have to be honored by general purpose MPV processors.

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.

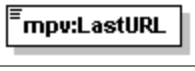
Multiple lastURLs may be provided to allow for different filenames in different file systems, such as on a CD, or both relative and absolute pathnames.

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, except that they use the UpperCamelCase naming convention of elements.

<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 of longest filename string to shortest. This minimizes the risk of using a LastURL that resolves to the wrong file due to OS-supplied aliasing. This problem has been encountered with the hidden 8.3 filenames on Microsoft Windows operating systems.

element **mpv:LastURL**

diagram	
namespace	http://ns.osta.org/mpv/1.0/

type	extension of **xs:anyURI**
used by	group **mpv:ResourceFileElemGroup**
attributes	Name / Type / Use / Default / Fixed mpv:hint / xs:anyURI mpv:filesystem / mpv:FilesystemType mpv:byteOffset / xs:integer mpv:leaseExpiresDate / xs:date mpv:leaseDur / xs:float mpv:leaseID / xs:string
source	<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:leaseExpiresDate"/> <xs:attribute ref="mpv:leaseDur"/> <xs:attribute ref="mpv:leaseID"/> <xs:attributeGroup ref="mpv:AnyAttrGroup"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element>

simpleType **FilesystemType**

namespace	http://ns.osta.org/mpv/1.0/
type	union of (**mpv:FilesystemBaseType** , **xs:anyURI**)
used by	attribute **LastURL/@filesystem**
source	<xs:simpleType name="FilesystemType"> <xs:union memberTypes="mpv:FilesystemBaseType xs:anyURI"/> </xs:simpleType>

simpleType **FilesystemBaseType**

namespace	http://ns.osta.org/mpv/1.0/
type	restriction of **xs:string**
used by	simpleType **FilesystemType**
facets	enumeration URI enumeration ISO9660-1 enumeration ISO9660-2 enumeration ISO9660-3 enumeration HFS enumeration Joliet enumeration UDF enumeration RockRidge enumeration FAT16 enumeration FAT32 enumeration NTFS enumeration Windows enumeration Unix
source	<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="UDF"/>

```
    <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>
```

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
- "UDF" – file and directory names compliant with UDF 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:myfirm-com:myproject: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 asset. The order of the arguments is not relevant and argument names are case-insensitive. All MPV arguments carry the "mpv" prefix in the argument name. The "<value>" string uses the syntax appropriate to the argument. Any URN-illegal characters are translated in the usual way.

As discussed in section 5.7 Identification of embedded assets, the LastURL may also contain a fragment identifier to refer to embedded assets.

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=<mpv_instanceID value>
&mpv_documentID=<value>&mpv_contentID=<value>
&mpv_filesystem=<value>&mpv_hint=<value>
&mpv_byteOffset=<value>
&mpv_leaseID=<value>&mpv_leaseDur=<value>
&mpv_leaseExpiresDate=<value>#<mpv_ID value>

The following arguments are defined by MPV for lastURL:

<mpv_ID value>

Indicates the fragment id in the referenced document. This is interpreted based on the media type of the asset.

mpv_instanceID, 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:myfirm-com:myproject:original".

mpv_byteOffset

Indicates a byte offset into the referenced file, such as "`lastURL?mpv_byteOffset=3342`". This usage is available but use of the fragment identifier for referencing embedded assets is the preferred approach.

mpv_leaseID

The leaseID of the URL

mpv_leaseDur

The lease duration of the URL

mpv_leaseExpiresDate

The lease expiration date of the URL

5.7 Identification of embedded assets

There are many cases where the asset that is being described is not itself a top-level file. Instead, the asset is embedded in the byte-stream of a file which serves as a container for the asset.

MPV does not attempt to provide a complete hierarchical location approach for the various container file types such as Quicktime and AVI. Instead, it utilizes an open scheme for referencing embedded content of media assets that is

already widely used for HTML and XML content. MPV specifies a limited set of fragment identifier definitions for the most common cases, such as Exif thumbnails and audio annotations.

MPV assets are located using URI which are the value of the mpv:lastURL attribute or the mpv:LastURL element. The MPV approach to identifying embedded assets is based on Section 4.1 of the URI specification[URI] which notes that the semantics of a fragment identifier (part of a URI after a "#") is a property of the data resulting from a retrieval action, and that the format and interpretation of fragment identifiers is dependent on the media type of the retrieval result. This means that if the media type of an asset indicates that it is embedded, then the media type can also specify how to interpret the fragment identifier portion of the URI in order to locate the embedded asset within the containing asset.

The media type of an asset is specified using MIME-types which are described in Appendix I:. If the scheme of the URL is "http:", then this value can be obtained by performing an HTTP GET request on the URI. The mime-type will be returned as the value of the "ContentType" header of the http reply. If the scheme is "file", then the mime-type of the asset must be obtained through some other means.

MPV defines several MIME-types that allow the application to recognize that the mpv:LastURL value identifies an embedded asset. These are also described in Appendix I: In addition, MPV defines an approach to specifying the fragment identifier so that it can be interpreted even when the media-type specifier is not available.

The following example shows two still assets. The second still is a thumbnail rendition of the first. Note that both assets have the same URI which is the relative URL, "IMG001.JPG" in the example. The thumbnail also has a fragment identifier part in its LastURL value which indicates that asset is actually embedded. The fragment identifier in this case is identical to the MIME-type specified as the value of the format property.

There may be more sophisticated embedded media types that require additional location information to distinguish between multiple embedded assets of the same media type. In general, the fragment identifier usage defined by MPV allows an application to process the fragment identifier portion of the URL without needing additional context on the format of the asset.

In this example, the fragment identifier syntax for the Exif thumbnail is compliant with the MPV usage described in Appendix I:

```
..
  <mpv:Still mpv:id="ID000100">
     <mpv:LastURL>IMG001.JPG</mpv:LastURL>
     <mpv:Rendition mpv:renditionUsage="thumbnail">
         <mpv:StillRef mpv:idRef="ID000200" />
     </mpv:Rendition>
  </mpv:Still>
...

  <mpv:Still mpv:id="ID000200" mpv:instanceID="AC937BCFA3B340da971BAF09B17DBC324">
    <mpv:LastURL>IMG001.JPG#vnd.osta-org.exif-thumb</mpv:LastURL>
    <nmf:Metadata>
       <dc:Properties>
           <dc:format>image/vnd.osta-org.exif-thumb</dc:format>
       </dc:Properties>
    </nmf:Metadata>
  </mpv:Still>
...
```


5.8 MPV Schema Identity Groups

MPV defines a small set of XML schema groups which are used widely throughout the specification's grammarto convey consistent attributes and sub-elements to user-visible elements. 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 assets in the collection that do not have equivalents in discrete files.
- **ResourceFileAttrGroup, ResourceFileElemGroup**: Provides for the basis for associating robust identification and metadata with assets in the collection that operate as proxies for discrete files with associated data.

None of these schema groups are user-visible, but they are important conceptual groups and are applied to the user-visible MPVelements.

5.9 Groups: ElemIdAttrGroup, ElemIdElemGroup

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.

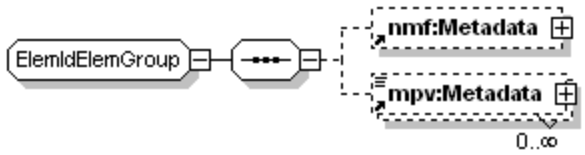
attributeGroup **mpv:ElemIdAttrGroup**

namespace	http://ns.osta.org/mpv/1.0/				
used by	complexType attributeGroup	**mpv:RefBaseType** **mpv:ResourceIdAttrGroup**			
attributes	Name mpv:id	Type xs:id	Use	Default	Fixed
source	<xs:attributeGroup name="ElemIdAttrGroup"> <xs:attribute ref="mpv:id"/> </xs:attributeGroup>				

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

group **mpv:ElemIdElemGroup**

diagram



namespace http://ns.osta.org/mpv/1.0/

children **nmf:Metadata mpv:Metadata**

used by complexTypes **mpv:ListRefBaseType mpv:RelatedType**
groups **mpv:ResourceFileElemGroup mpv:ResourceIdElemGroup**

source
```
<xs:group name="ElemIdElemGroup">
  <xs:sequence>
    <xs:element ref="nmf:Metadata" minOccurs="0"/>
    <xs:element ref="mpv:Metadata" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
```

```
</xs:group>
```

5.10 Groups: ResourceIdAttrGroup, ResourceIdElemGroup

An element that has its own identity in a 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.

attributeGroup **mpv:ResourceIdAttrGroup**

namespace http://ns.osta.org/mpv/1.0/

used by
- complexTypes **mpv:CompositeAssetBaseType mpv:ManifestChildType**
- attributeGroup **mpv:ResourceFileAttrGroup**

attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			
mpv:instanceID	xs:anyURI			
mpv:documented	xs:anyURI			
mpv:contentID	xs:anyURI			

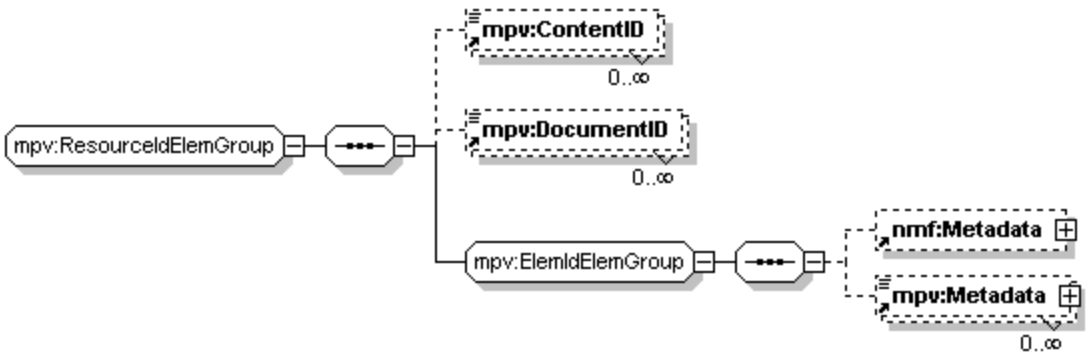
source

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

group **mpv:ResourceIdElemGroup**

diagram



namespace http://ns.osta.org/mpv/1.0/

children **mpv:DocumentID mpv:ContentID nmf:Metadata mpv:Metadata**

used by
- complexTypes **mpv:CompositeAssetBaseType mpv:ManifestChildType mpv:ManifestType**

source

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

```
</xs:group>
```

5.11 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.

attributeGroup **mpv:ResourceFileAttrGroup**

namespace http://ns.osta.org/mpv/1.0/

used by complexType **mpv:SimpleAssetBaseType**

attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			
mpv:instanceID	xs:anyURI			
mpv:documentID	xs:anyURI			
mpv:contentID	xs:anyURI			
mpv:lastURL	xs:anyURI			
mpv:byteOffset	xs:integer			
mpv:leaseExpiresDate	xs:date			
mpv:leaseDur	xs:float			
mpv:leaseID	xs:string			

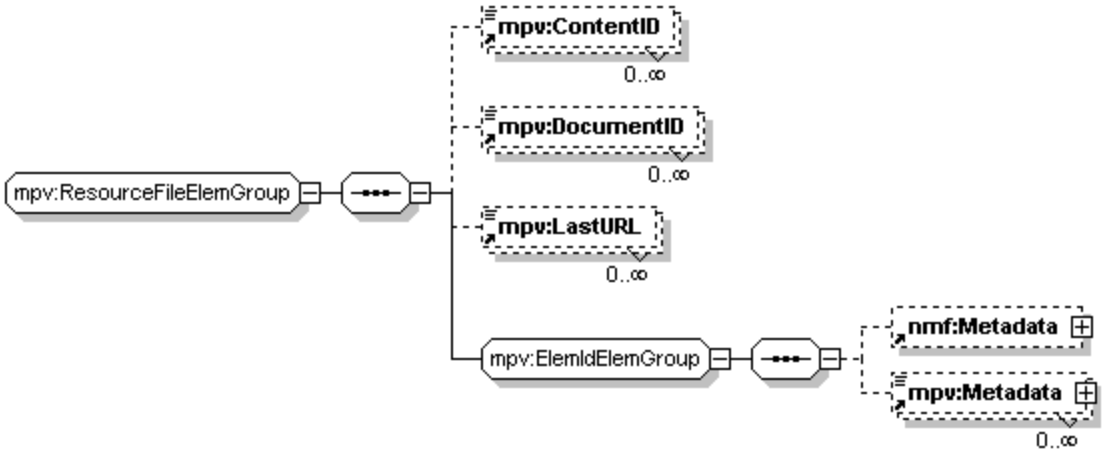
source

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

group **mpv:ResourceFileElemGroup**

diagram



namespace http://ns.osta.org/mpv/1.0/

children **mpv:DocumentID mpv:ContentID mpv:LastURL nmf:Metadata mpv:Metadata**

used by complexType **mpv:SimpleAssetBaseType**

source

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

5.12 <IdentProperties> -- Resource Identification in NMF Metadata

MPV provides an identification model. This model may be specified using native MPV syntax, as described earlier. It may also be necessary to specify identity information as part of NMF-structured metadata. The following schema implements the same semantics for identification discussed previously, but using NMF-structured metadata elements. It can be used within any <nmf:Metadata> element.

In an MPV-based schema definition, the introductory schema information is expressed as follows. As seen in the example, to avoid creation of excessive numbers of prefixed namespaces, NMF convention is to set the default namespace at every level as needed.

Example:

```
...
  <nmf:Metadata>
    <IdentProperties xmlns="http://ns.osta.org/mpv/1.0/ident/">
```

```
      <ContentID>urn:osta-org:mpv:dsig:md5:all:EF886AEFA3B340da971BAF09B17DBC122</ContentID>
      <InstanceID>AC937BCFA3B340da971BAF09B17DBC324</InstanceID>
      <LastURLBag>
        <LastURL>
           <LastURLProperties xmlns="http://ns.osta.org/mpv/1.0/ident/lasturl/">
               <Filesystem>NTFS</Filesystem>
               <URL>The name of the image.JPG</URL>
           </LastURLProperties>
        </LastURL>
        <LastURL>
           <LastURLProperties xmlns="http://ns.osta.org/mpv/1.0/ident/lasturl/">
               <Filesystem>ISO9660-1</Filesystem>
               <URL>THE_NAME_.JPG</URL>
           </LastURLProperties>
        </LastURL>
      </LastURLBag>
    </IdentProperties>
  </nmf:Metadata>
...
```

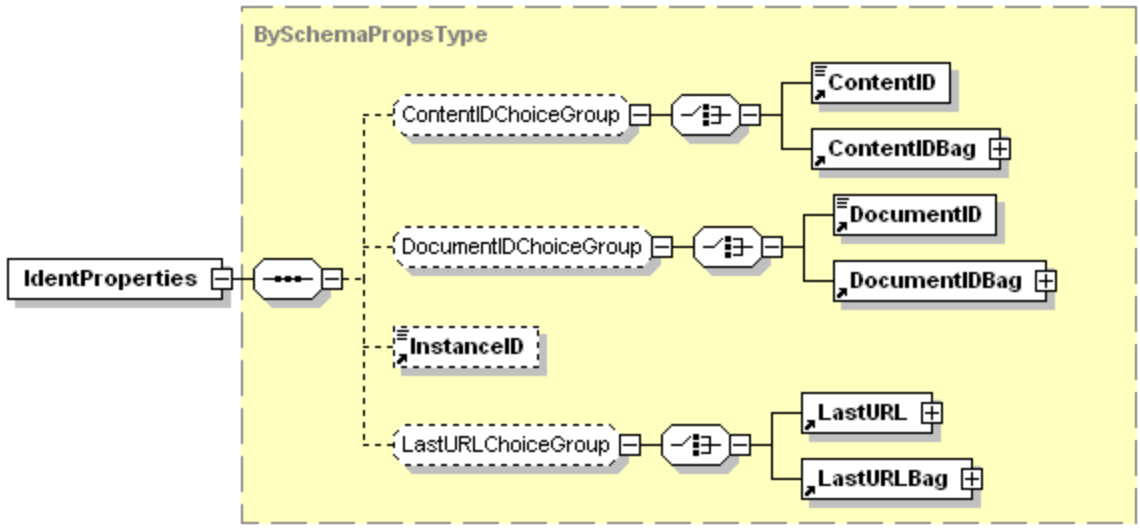
IDENT PROPERTIES

IdentProperties is the outer element of the MPV identity schema encoded as NMF Metadata. It can contain the few elements defined below. Note that ContentID, DocumentID, and LastURL may also be specified as Bags. A Bag contains one or more unordered children elements. Because NMF metadata do not use attributes, the LastURL attributes are encoded as property elements of the NMF LastURL element.

Schema group	Namespace Identifier	Schema Location	Conventional Namespace Prefix
Core	http://ns.osta.org/mpv/1.0/ ident/	core/imp/nmf/ident.xsd	none

Elements	Groups	Complex types
ContentID	**ContentIDChoiceGroup**	**BySchemaPropsType**
ContentIDBag	**DocumentIDChoiceGroup**	**ContentIDBagType**
DocumentID	**LastURLChoiceGroup**	**ContentIDType**
DocumentIDBag		**DocumentIDBagType**
IdentProperties		**DocumentIDType**
InstanceID		**InstanceIDType**
LastURL		**LastURLBagType**
LastURLBag		**LastURLType**

element **IdentProperties**

diagram	
namespace	http://ns.osta.org/mpv/1.0/ident/
type	**BySchemaPropsType**
children	**ContentID ContentIDBag DocumentID DocumentIDBag InstanceID LastURL LastURLBag**
source	`<xs:element name="IdentProperties" type="BySchemaPropsType" substitutionGroup="nmf:BySchemaPropsBase"/>`

source	`<xs:complexType name="BySchemaPropsType">`
	`<xs:complexContent>`
	`<xs:extension base="nmf:BySchemaPropsType">`
	`<xs:sequence>`
	`<xs:group ref="ContentIDChoiceGroup" minOccurs="0"/>`
	`<xs:group ref="DocumentIDChoiceGroup" minOccurs="0"/>`
	`<xs:element ref="InstanceID" minOccurs="0"/>`
	`<xs:group ref="LastURLChoiceGroup" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:extension>`
	`</xs:complexContent>`
	`</xs:complexType>`

source	`<xs:group name="ContentIDChoiceGroup">`
	`<xs:choice>`
	`<xs:element ref="ContentID"/>`
	`<xs:element ref="ContentIDBag"/>`
	`</xs:choice>`
	`</xs:group>`

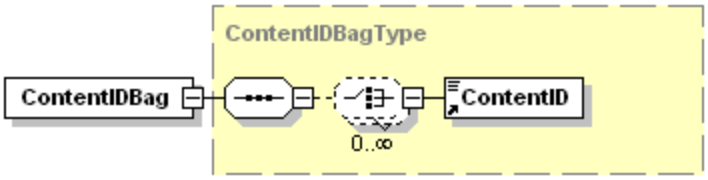
source	`<xs:group name="DocumentIDChoiceGroup">`
	`<xs:choice>`
	`<xs:element ref="DocumentID"/>`
	`<xs:element ref="DocumentIDBag"/>`
	`</xs:choice>`
	`</xs:group>`

source	`<xs:group name="LastURLChoiceGroup">`
	`<xs:choice>`
	`<xs:element ref="LastURL"/>`
	`<xs:element ref="LastURLBag"/>`
	`</xs:choice>`
	`</xs:group>`

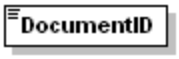
element **ContentID**

diagram	
namespace	http://ns.osta.org/mpv/1.0/ident/
type	**ContentIDType**
used by	complexType **ContentIDBagType** group **ContentIDChoiceGroup**
source	<xs:element name="ContentID" type="ContentIDType"/>
source	<xs:complexType name="ContentIDType"> <xs:simpleContent> <xs:extension base="xs:anyURI"/> </xs:simpleContent> </xs:complexType>

element **ContentIDBag**

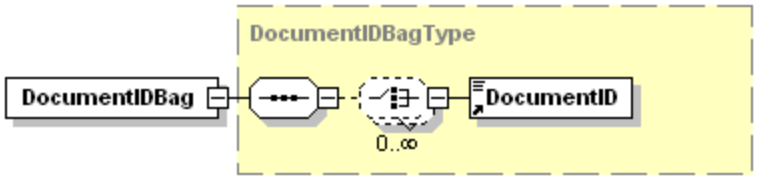
diagram	
namespace	http://ns.osta.org/mpv/1.0/ident/
type	**ContentIDBagType**
children	**ContentID**
used by	group **ContentIDChoiceGroup**
source	<xs:element name="ContentIDBag" type="ContentIDBagType"/>
source	<xs:complexType name="ContentIDBagType"> <xs:complexContent> <xs:extension base="nmf:BagPropType"> <xs:sequence> <xs:choice minOccurs="0" maxOccurs="unbounded"> <xs:element ref="ContentID"/> </xs:choice> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

element **DocumentID**

diagram	
namespace	http://ns.osta.org/mpv/1.0/ident/
type	**DocumentIDType**
used by	complexType **DocumentIDBagType** group **DocumentIDChoiceGroup**
source	<xs:element name="DocumentID" type="DocumentIDType"/>
source	<xs:complexType name="DocumentIDType">

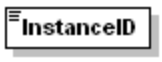
```
  <xs:simpleContent>
    <xs:extension base="xs:anyURI"/>
  </xs:simpleContent>
</xs:complexType>
```

element **DocumentIDBag**

diagram	
namespace	http://ns.osta.org/mpv/1.0/ident/
type	**DocumentIDBagType**
children	**DocumentID**
used by	group **DocumentIDChoiceGroup**
source	`<xs:element name="DocumentIDBag" type="DocumentIDBagType"/>`

source	`<xs:complexType name="DocumentIDBagType">`
	`  <xs:complexContent>`
	`  <xs:extension base="nmf:BagPropType">`
	`  <xs:sequence>`
	`  <xs:choice minOccurs="0" maxOccurs="unbounded">`
	`  <xs:element ref="DocumentID"/>`
	`  </xs:choice>`
	`  </xs:sequence>`
	`  </xs:extension>`
	`  </xs:complexContent>`
	`</xs:complexType>`

element **InstanceID**

diagram	
namespace	http://ns.osta.org/mpv/1.0/ident/
type	**InstanceIDType**
used by	complexType **BySchemaPropsType**
source	`<xs:element name="InstanceID" type="InstanceIDType"/>`

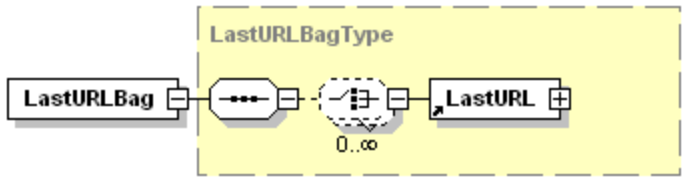
source	`<xs:complexType name="InstanceIDType">`
	`  <xs:simpleContent>`
	`  <xs:extension base="xs:anyURI"/>`
	`  </xs:simpleContent>`
	`</xs:complexType>`

element **LastURL**

diagram	
namespace	http://ns.osta.org/mpv/1.0/ident/

type	**LastURLType**
children	**LastURLProperties**
used by	complexType **LastURLBagType** group **LastURLChoiceGroup**
source	<xs:element name="LastURL" type="LastURLType"/>
source	<xs:complexType name="LastURLType"> <xs:complexContent> <xs:extension base="nmf:CompositePropType"> <xs:sequence> <xs:element ref="lasturl:LastURLProperties"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

element **LastURLBag**

diagram	
namespace	http://ns.osta.org/mpv/1.0/ident/
type	**LastURLBagType**
children	**LastURL**
used by	group **LastURLChoiceGroup**
source	<xs:element name="LastURLBag" type="LastURLBagType"/>
source	<xs:complexType name="LastURLBagType"> <xs:complexContent> <xs:extension base="nmf:BagPropType"> <xs:sequence> <xs:choice minOccurs="0" maxOccurs="unbounded"> <xs:element ref="LastURL"/> </xs:choice> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

5.12.1 LastURLProperties

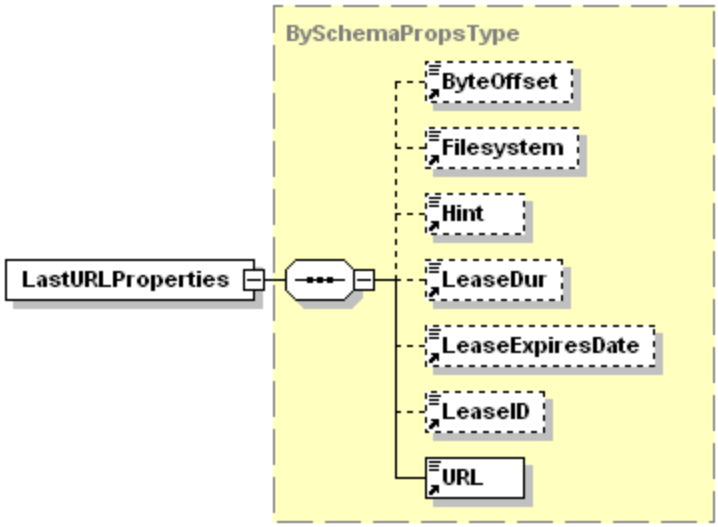
LastURLProperties is the outer element of the MPV LastURL concept schema encoded as NMF Metadata. It can contain the few elements defined below.

Schema group	Namespace Identifier	Schema Location	Conventional Namespace Prefix
Core	http://ns.osta.org/mpv/1.0/ ident/lasturl/	core/imp/nmf/structs/lasturl.xsd	none

Elements	Complex types
ByteOffset	**BySchemaPropsType**
Filesystem	**ByteOffsetType**

Hint	**FilesystemType**
LastURLProperties	**HintType**
LeaseDur	**LeaseDurType**
LeaseExpiresDate	**LeaseExpiresDateType**
LeaseID	**LeaseIDType**
URL	**URLType**

element **LastURLProperties**

diagram	
namespace	http://ns.os ta.org/mpv/1.0/ident/lasturl/
type	**BySchemaPropsType**
children	**ByteOffset Filesystem Hint LeaseDur LeaseExpiresDate LeaseID URL**
used by	complexType **LastURLType**
source	`<xs:element name="LastURLProperties" type="BySchemaPropsType" substitutionGroup="nmf:BySchemaPropsBase"/>`

source	`<xs:complexType name="BySchemaPropsType">`
	`<xs:complexContent>`
	`<xs:extension base="nmf:BySchemaPropsType">`
	`<xs:sequence>`
	`<xs:element ref="ByteOffset" minOccurs="0"/>`
	`<xs:element ref="Filesystem" minOccurs="0"/>`
	`<xs:element ref="Hint" minOccurs="0"/>`
	`<xs:element ref="LeaseDur" minOccurs="0"/>`
	`<xs:element ref="LeaseExpiresDate" minOccurs="0"/>`
	`<xs:element ref="LeaseID" minOccurs="0"/>`
	`<xs:element ref="URL"/>`
	`</xs:sequence>`
	`</xs:extension>`
	`</xs:complexContent>`
	`</xs:complexType>`

source	`<xs:element name="ByteOffset" type="ByteOffsetType"/>`

source	`<xs:complexType name="ByteOffsetType">`
	`<xs:simpleContent>`
	`<xs:extension base="xs:integer"/>`
	`</xs:simpleContent>`
	`</xs:complexType>`

source	`<xs:element name="Filesystem" type="FilesystemType"/>`

source	<xs:complexType name="FilesystemType"> <xs:simpleContent> <xs:extension base="mpv:FilesystemType"/> </xs:simpleContent> </xs:complexType>
source	<xs:element name="Hint" type="HintType"/>
source	<xs:complexType name="HintType"> <xs:simpleContent> <xs:extension base="xs:string"/> </xs:simpleContent> </xs:complexType>
source	<xs:element name="LeaseDur" type="LeaseDurType"/>
source	<xs:complexType name="LeaseDurType"> <xs:simpleContent> <xs:extension base="xs:float"/> </xs:simpleContent> </xs:complexType>
source	<xs:element name="LeaseExpiresDate" type="LeaseExpiresDateType"/>
source	<xs:complexType name="LeaseExpiresDateType"> <xs:simpleContent> <xs:extension base="xs:date"/> </xs:simpleContent> </xs:complexType>
source	<xs:element name="LeaseID" type="LeaseIDType"/>
source	<xs:complexType name="LeaseIDType"> <xs:simpleContent> <xs:extension base="xs:string"/> </xs:simpleContent> </xs:complexType>
source	<xs:element name="URL" type="URLType"/>
source	<xs:complexType name="URLType"> <xs:simpleContent> <xs:extension base="xs:anyURI"/> </xs:simpleContent> </xs:complexType>

Chapter 6: MPV Core Schema, Part 2: Collection

6.1 <mpv:AssetList>

An asset list is an unordered set of assets that each have a unique local identifier in the MPV collection. It is the only place photo-video assets may be defined as part of the collection – everything else in MPV is metadata and references to assets. A MPV collection contains at least one asset list or link to an asset list in another file. By analogy, an asset list may be considered a table of assets in a database and the id is the foreign key. Another analogy would be to the entries in a Unix file system inode.

When used by a Profile, the AssetList is a top-level child of the OSTA XML Manifest. It is the one well-defined top-level child that every MPV-aware processor can count on being present within an OSTA XML Manifest, and it is the element which achieves interchange of collections across applications, regardless of which additional profiles are implemented by those applications.

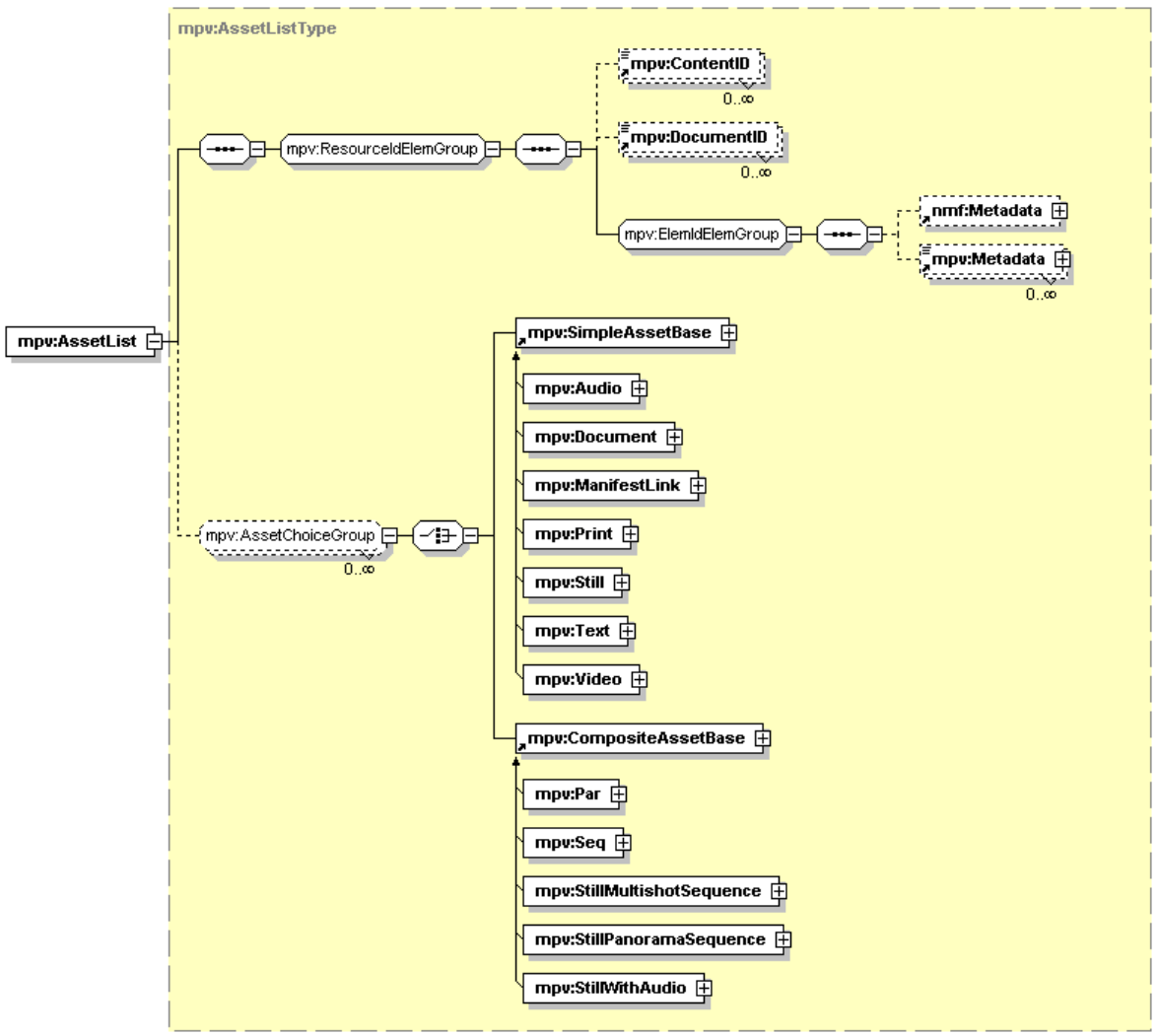
Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<file:Manifest
        xmlns:file="http://ns.osta.org/manifest/1.0/"
        xmlns:mpv="http://ns.osta.org/mpv/1.0/"
        xmlns:nmf="http://ns.osta.org/nmf/1.0/" >
  <nmf:Metadata>
    <ManifestProperties xmlns="http://ns.osta.org/manifest/1.0/">
       <Profile>http://ns.osta.org/mpv/basic/1.0/</Profile>
    </ManifestProperties>
  </nmf:Metadata>

  <mpv:AssetList>
    <mpv:Still mpv:id="ID000100">
      <mpv:LastURL>DSC09075.JPG</mpv:LastURL>
      <mpv:ContentID>urn:osta-
 org:mpv:dsig:md5:all:EF886AEFA3B340da971BAF09B17DBC122</mpv:ContentID>
      <nmf:Metadata>
        <Properties xmlns="http://purl.org/dc/elements/1.1/">
          <title>Riding the roller coaster at the fair.</title>
        </Properties>
```

```
      </nmf:Metadata>
    </mpv:Still>
  </mpv:AssetList>
</file:Manifest>
```

The AssetList may contain identifiers and metadata. These can provide context around the list of assets. Primarily, the AssetList contains elements defining MPV assets. MPV assets are considered proxies for actual media assets that can be located using the identifiers supplied with each MPV asset.

element **mpv:AssetList,** complexType **mpv:AssetListType**

diagram	
namespace	http://ns.osta.org/mpv/1.0/
type	**mpv:AssetListType**
children	**mpv:DocumentID mpv:ContentID nmf:Metadata mpv:Metadata mpv:SimpleAssetBase mpv:CompositeAssetBase**
used by	complexType **mpv:ManifestType**

attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			
mpv:instanceID	xs:anyURI			
mpv:documentID	xs:anyURI			
mpv:contentID	xs:anyURI			

source	<xs:element name="AssetList" type="mpv:AssetListType"/>

```
source   <xs:complexType name="AssetListType">
           <xs:complexContent>
             <xs:extension base="mpv:ManifestChildType">
               <xs:group ref="mpv:AssetChoiceGroup" minOccurs="0" maxOccurs="unbounded"/>
             </xs:extension>
           </xs:complexContent>
         </xs:complexType>
```

6.2 <mpv:MarkList>

There are many situations where a subset of the AssetList items needs to be identified. Examples include a subset that is marked temporarily for handoff to downstream processing like editing, printing or e-mail. In addition, interactive profiles need to provide the user with the ability to add and remove AssetList items from a selected set.

MPV provides the MarkList as a general facility for dealing with these types of requirements and to enahble interchange of selected assets across applications. MarkLists can be employed by any schema and are typically used within Profile schema that exist side-by-side with the AssetList. The MarkList enables the Profile-specific data to be held external to the AssetList. Alternately, Profile-specific metadata may be included with the asset in the AssetList.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<file:Manifest
        xmlns:file="http://ns.osta.org/manifest/1.0/"
        xmlns:mpv="http://ns.osta.org/mpv/1.0/"
        xmlns:mpvb="http://ns.osta.org/mpv/basic/1.0/"
        xmlns:nmf="http://ns.osta.org/nmf/1.0/" >
  <nmf:Metadata>
    <ManifestProperties xmlns="http://ns.osta.org/manifest/1.0/">
       <Profile>http://ns.osta.org/mpv/basic/1.0/</Profile>
    </ManifestProperties>
  </nmf:Metadata>

  <mpvb:MarkedAssets>
    <mpv:MarkList mpv:markType="selected">
      <nmf:Metadata>
        <Properties xmlns="http://purl.org/dc/elements/1.1/">
          <title>Assets selected by the user</title>
        </Properties>
      </nmf:Metadata>
      <mpv:StillRef mpv:idRef="ID000200"/>
    </mpv:MarkList>
  </mpvb:MarkedAssets>

  <mpv:AssetList>
    <mpv:Still mpv:id="ID000100" mpv:lastURL="DSC09342.JPG"/>
    <mpv:Still mpv:id="ID000200" mpv:lastURL="DSC09343.JPG"/>
    <mpv:Still mpv:id="ID000300" mpv:lastURL="DSC09344.JPG"/>
  </mpv:AssetList>
</file:Manifest>
```

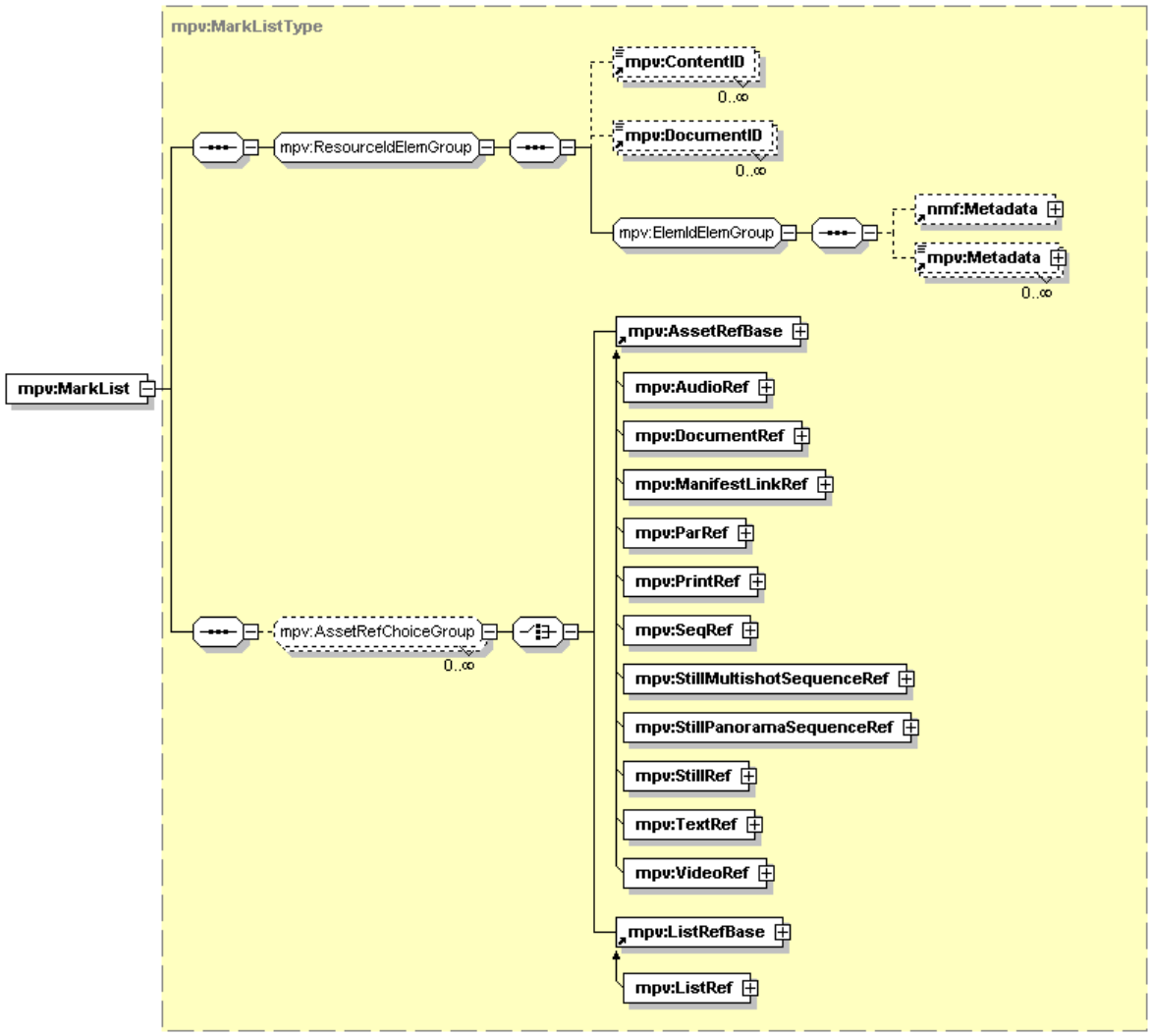
MarkLists may also make reference to assets in another Manifest. This can be done on a per asset reference basis or by providing a default list id for the whole MarkList.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<file:Manifest ... >
...
  <mpvb:MarkedAssets>
    <mpv:MarkList mpv:defaultManifestLinkIDRef="ID000800" mpv:markType="selected">
      <mpv:StillRef mpv:id="AAAA9430"/>  <!-- default manifest used -->
      <mpv:StillRef mpv:id="AAAA9433" mpv:manifestLinkIDRef="ID000800"/>  <!-- explicit -->
    </mpv:MarkList>
  </mpvb:MarkedAssets>

  <mpv:AssetList>
    <mpv:ManifestLink mpv:id="ID000800" mpv:instanceID="EF886AEFA3B340da971BAF09B17DBC122">
      <mpv:LastURL>2002-06-23/album.pvm</mpv:LastURL>
      <mpv:LastURL>C:/My Documents/My Pictures/2002-06-23/album.pvm</mpv:LastURL>
...
    </mpv:ManifestLink>
  </mpv:AssetList>
```

element **mpv:MarkList,** complexType **mpv:MarkListType**

diagram



namespace http://ns.osta.org/mpv/1.0/

type	**mpv:MarkListType**
children	**mpv:DocumentID mpv:ContentID nmf:Metadata mpv:Metadata mpv:AssetRefBase mpv:ListRefBase**
used by	complexType **mpv:ManifestChildType**

attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			
mpv:instanceID	xs:anyURI			
mpv:documentID	xs:anyURI			
mpv:contentID	xs:anyURI			
mpv:defaultListIDRef	xs:IDREF	optional		
mpv:defaultManifestIDRef	xs:IDREF	optional		
markType	mpv:MarkType			

source

```
<xs:element name="MarkList" type="mpv:MarkListType"/>
```

source

```
<xs:complexType name="MarkListType">
  <xs:complexContent>
    <xs:extension base="mpv:AssetRefListBaseType">
      <xs:attribute name="markType" type="mpv:MarkType"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

simpleType **MarkType**

namespace	http://ns.osta.org/mpv/1.0/
type	union of (**mpv:MarkTypeBaseType** , **xs:anyURI**)
used by	attribute **MarkListType/@markType**
facets	enumeration primary enumeration selected enumeration hidden

source	`<xs:simpleType name="MarkType">`
	`  <xs:union memberTypes="mpv:MarkTypeBaseType xs:anyURI"/>`
	`</xs:simpleType>`

source	`<xs:simpleType name="MarkTypeBaseType">`
	`  <xs:restriction base="xs:string">`
	`  <xs:enumeration value="primary"/>`
	`  <xs:enumeration value="selected"/>`
	`  <xs:enumeration value="hidden"/>`
	`  </xs:restriction>`
	`</xs:simpleType>`

defaultListIDRef

Provides the “mpv:id” value of the AssetList or MarkList that contains the referenced assets in the mark list. When no defaultListIDRef is present, the AssetList in the same manifest is used.

defaultManifestLinkIDRef

Provides the “mpv:id” value of the ManifestLink asset that contains the referenced assets in the mark list. When no defaultManifestLinkIDRef is present, the current manifest is used.

markType

The type of mark to apply to all the referenced items. The markType has an open vocabulary with the following initial values that are reserved by MPV. Only MPV should define additional marktypes without URN-qualified names. Applications defining new mark types should use URN-qualified names such as “urn:myfirm-com:mpv:someMarkType” to avoid the possibility of name collisions.

"primary"

The primary list of items in the asset list from a user's perspective. The concept of primary is that an asset list may contain many unordered assets at various levels of hierarchy, such as many screen and thumbnail resolution images of master images. The primary asset marklist defines a sequence of primary assets.

"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.

"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.

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.

6.3 <mpv:Related>

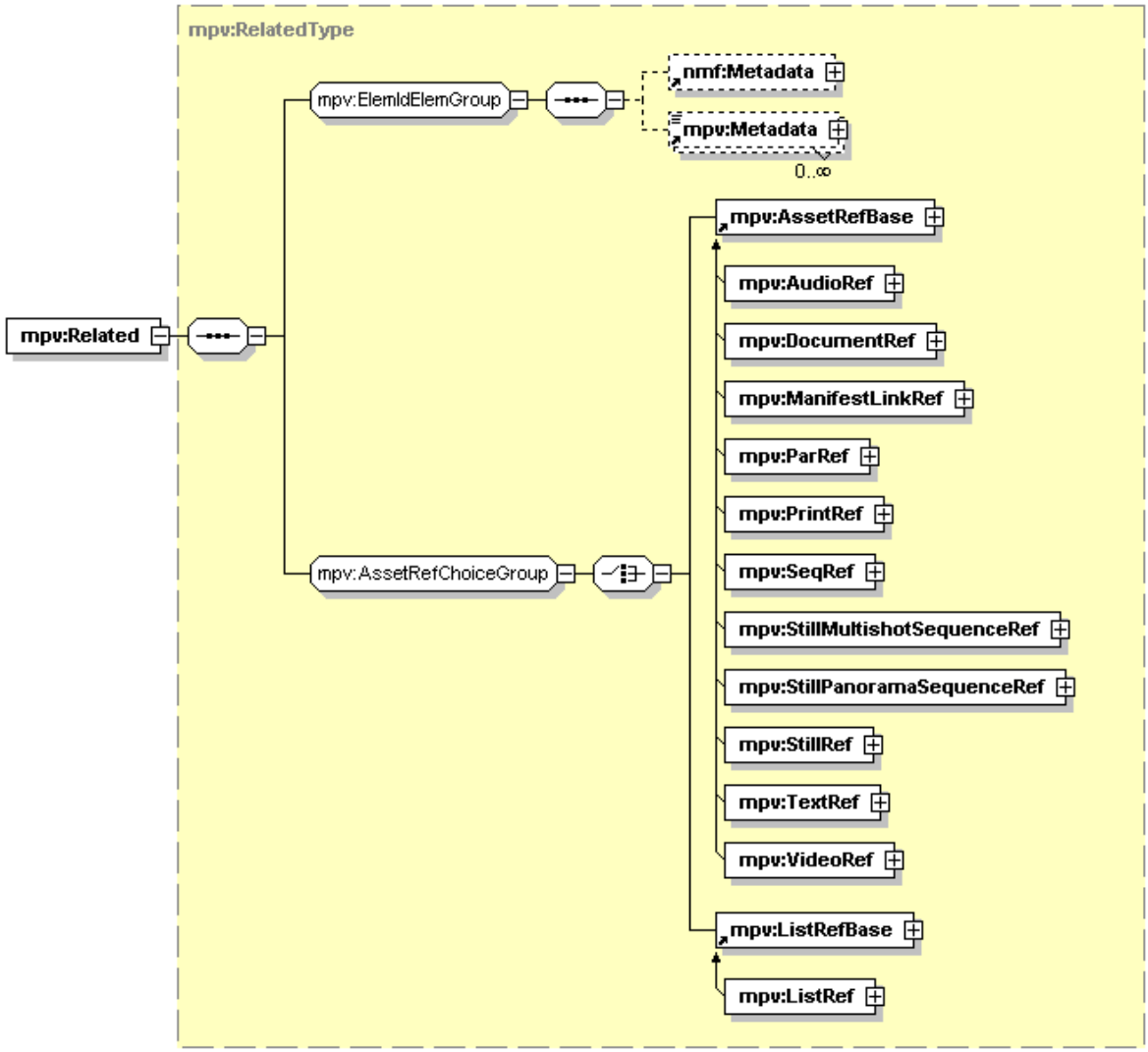
Related is a generic container which carries no specific semantics other than that the contents are related to the asset that contains them. It may contain any number of media assets.

Example:

```
...
  <mpv:AssetList>
...
    <mpv:Still mpv:id="ID000400" mpv:lastURL="DSC09344-cropped.JPG">
      <mpv:Related mpv:relationship="derivedFrom">
        <mpv:StillRef mpv:idRef="ID000300"/>
      <mpv:Related>
    </mpv:Still>
...
  </mpv:AssetList>
...
```

element **mpv:Related,** complexType **mpv:RelatedType**

diagram



namespace http://ns.osta.org/mpv/1.0/

type **mpv:RelatedType**

children **nmf:Metadata mpv:Metadata mpv:AssetRefBase mpv:ListRefBase**

used by complexTypes **mpv:ParType mpv:SeqType mpv:SimpleAssetBaseType mpv:StillMultishotSequenceType mpv:StillPanoramaSequenceType mpv:StillWithAudioType**

attributes

Name	Type	Use	Default	Fixed
relationship	xs:anyURI			

source
```
<xs:element name="Related" type="mpv:RelatedType"/>
```

source
```
<xs:element name="Related" type="mpv:RelatedType"/>
<xs:complexType name="RelatedType">
    <xs:sequence>
        <xs:group ref="mpv:ElemIdElemGroup"/>
        <xs:group ref="mpv:AssetRefChoiceGroup"/>
    </xs:sequence>
    <xs:attribute name="relationship" type="mpv:RelationshipType"/>
    <xs:attributeGroup ref="mpv:AnyAttrGroup"/>
</xs:complexType>
```

source
```
<xs:simpleType name="RelationshipType">
    <xs:union memberTypes="mpv:RelationshipBaseType xs:anyURI"/>
</xs:simpleType>
```

```
<xs:simpleType name="RelationshipBaseType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="derivedFrom"/>
    </xs:restriction>
</xs:simpleType>
```

relationship

The relationship hint applies to the related item and has an open vocabulary. The MPV Core reserves the set of relationship strings that are not URN-qualified. Relationship values provided by new Profiles must use URN-qualified names to avoid the possibility of name collisions, such as "urn:my firm-com:mpv:somerelation".

"derivedFrom"

The derivedFrom relationship indicates that the primary asset was derived in some fashion from the asset(s) identified as Related. Typically, additional metadata can be specified to articulate more about the derivation, such as series of edit operations applied to the related asset(s) to result in the primary asset. This metadata is not defined by MPV Core 1.0.

6.4 <mpv:Rendition>

A rendition is a derivative of a media asset. Renditions should have the same "documentID" as the parent media asset but different instanceIDs and contentIDs. A rendition can contain any number of media assets.

Take note that there is no "default", "master" or "original" renditionUsage. The proper practice is that the LastURL and other identifiers of the asset containing renditions are the default identifiers for the asset and by convention are considered the "master" or "original" rendition of that asset.

Renditions are a very powerful concept. What is interesting is how they can evolve as the context in which they are used shifts. Consider the case in which a collection of assets on a computer harddrive represented by a Photo/Video Manifest is moved to a recordable optical disc for distribution. In the simple case, all assets are transferred; the manifest remains unchanged except perhaps for LastURL fixup. In the case where some assets are not transferred, such as the master versions of the assets, then the Photo/Video Manifest on the optical disc should be fixed up so that the reference to the master assets are moved to a "derivedFrom" Related asset and one of the renditions is promoted to be the "default" asset, becoming the new master asset of the item in the new, on-disc, collection. Of course, if the manifest is not fixed up and not all the assets are present, then a robust processing application will look for the best rendition that is available.

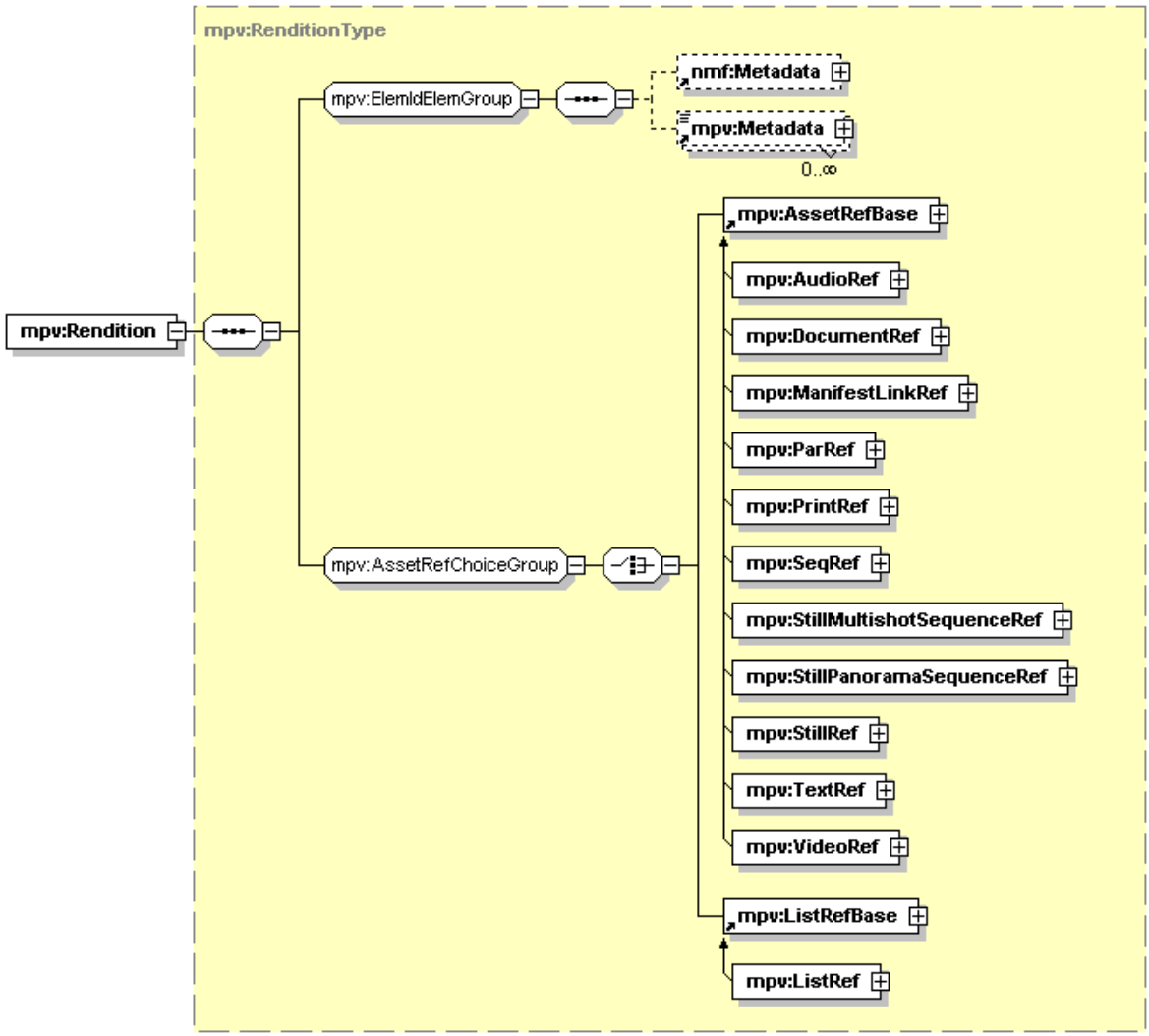
Example:

```
...
  <mpv:AssetList>
...
    <mpv:Still mpv:id="ID000500" mpv:lastURL="DSC09345.JPG">
      <mpv:Rendition mpv:renditionType="thumbnail">
        <mpv:StillRef mpv:idRef="ID000600"/>
      <mpv:Rendition>
      <mpv:Rendition mpv:renditionType="screen">
        <mpv:StillRef mpv:idRef="ID000700"/>
      <mpv:Rendition>
    </mpv:Still>
    <mpv:Still mpv:id="ID000600" mpv:lastURL="thumbs/DSC09345.JPG"/>
    <mpv:Still mpv:id="ID000700" mpv:lastURL="screen/DSC09345.JPG"/>
...
  </mpv:AssetList>
```

. . .

element **mpv:Rendition,** complexType **mpv:RenditionType**

diagram



namespace	http://ns.osta.org/mpv/1.0/
type	**mpv:RenditionType**
children	**nmf:Metadata mpv:Metadata mpv:AssetRefBase mpv:ListRefBase**
used by	complexTypes **mpv:ParType mpv:SeqType mpv:SimpleAssetBaseType mpv:StillMultishotSequenceType mpv:StillPanoramaSequenceType mpv:StillWithAudioType**

attributes

Name	Type	Use	Default	Fixed
renditionUsage	mpv:RenditionUsageType			

source

```
<xs:element name="Rendition" type="mpv:RenditionType"/>
```

source

```
<xs:complexType name="RenditionType">
  <xs:sequence>
    <xs:group ref="mpv:ElemIdElemGroup"/>
    <xs:group ref="mpv:AssetRefChoiceGroup"/>
  </xs:sequence>
  <xs:attribute name="renditionUsage" type="mpv:RenditionUsageType"/>
```

	</xs:complexType>

source
```
<xs:complexType name="RenditionType">
  <xs:group ref="mpv:AssetRefChoiceGroup"/>
  <xs:attribute name="renditionUsage" type="mpv:RenditionUsageType"/>
  <xs:attributeGroup ref="mpv:AnyAttrGroup"/>
</xs:complexType>
```

simpleType **mpv:RenditionUsageType**

namespace	http://ns.osta.org/mpv/1.0/
type	union of (**mpv:RenditionUsageBaseType** , **xs:anyURI**)
used by	attribute **mpv:RenditionType/@renditionUsage**
facets	enumeration thumbnail enumeration screen enumeration subsampled enumeration lowRes enumeration highRes enumeration print enumeration show enumeration proof enumeration draft enumeration targetSystem enumeration alt
source	<xs:simpleType name="RenditionUsageType"> <xs:union memberTypes="mpv:RenditionUsageBaseType xs:anyURI"/> </xs:simpleType>
source	<xs:simpleType name="RenditionUsageBaseType"> <xs:restriction base="xs:string"> <xs:enumeration value="thumbnail"/> <xs:enumeration value="screen"/> <xs:enumeration value="highRes"/> <xs:enumeration value="lowRes"/> <xs:enumeration value="print"/> <xs:enumeration value="show "/> <xs:enumeration value="subsampled"/> <xs:enumeration value="proof "/> <xs:enumeration value="draft"/> <xs:enumeration value="targetSystem"/> <xs:enumeration value="alt"/> </xs:restriction> </xs:simpleType>

renditionUsage

The vocabulary is an open vocabulary with the following initial values. Additional vocabulary values must use URN-qualified names to avoid the possibility of name collisions, such as "urn:my firm-com:mpv:somerendition". Take note that there is no “master”, “default”, or “original” renditionUsage. The specified practice in MPV is that the LastURL and other identifiers of the asset containing renditions are the default identifiers for the asset and by convention are considered the “master” or “original” rendition of that asset.

“thumbnail”

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

“screen”

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

“highRes”

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

“lowRes”

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

“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 assets (such as Album, StillWithAudio, StillMultishotSequence, StillPanoramaSequence, Par, Seq) that may offer a presentation-oriened rendition.

“subsampled”

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

“proof”

For a review proof

“draft”

For a review 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 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.

6.5 <mpv:ManifestLink>

The mpv:ManifestLink provides for linking to another Photo/Video Manifest. The rich resource referencing abilities of MPV may be applied to the link. The mpv:ManifestLink is a first-class asset and is used in the mpv:AssetList.

The semantic of a ManifestLink is that of a resource link, not an “include”. For example, a presentation of an AssetList that encounters a ManifestLink should not include the assets in the referenced Manifest. Instead, it should offer the user the ability to follow the link and open the new Manifest.

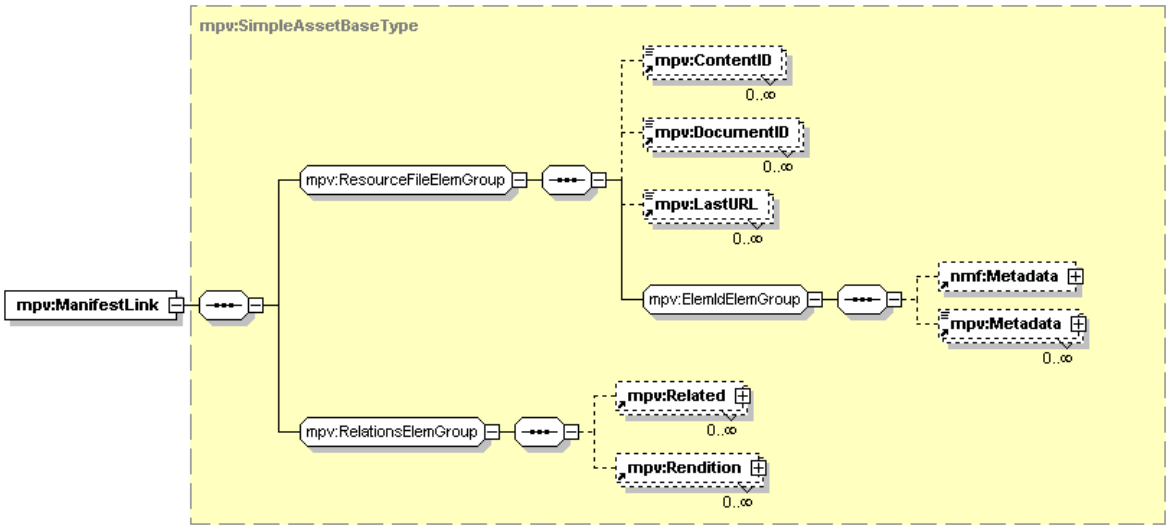
When the desired behaviour is to make reference to assets in another Manifest, then the ManifestLink’s id is used as part of an asset reference to identify the AssetList to use. This usage is encountered in MarkLists.

Example:

```
<file:Manifest ... >
...
  <mpvb:MarkedAssets>
    <mpv:MarkList mpv:defaultManifestLinkIDRef="ID000800" mpv:markType="selected">
      <mpv:StillRef mpv:id="AAAA9430"/>  <!-- A Still in another manifest -->
      <mpv:StillRef mpv:id="AAAA9433" mpv:manifestLinkIDRef="ID000800"/>  <!-- alt -->
    </mpv:MarkList>
  </mpvb:MarkedAssets>

  <mpv:AssetList>
...
    <mpv:ManifestLink mpv:id="ID000800" mpv:instanceID="EF886AEFA3B340da971BAF09B17DBC122">
      <mpv:LastURL>2002-06-23/album.pvm</mpv:LastURL>
      <mpv:LastURL>C:/My Documents/My Pictures/2002-06-23/album.pvm</mpv:LastURL>
      <nmf:Metadata>
        <Properties xmlns="http://purl.org/dc/elements/1.1/">
          <title>June 23, 2002 – visit to Benton County Fair</title>
        </Properties>
      </nmf:Metadata>
      <mpv:Rendition mpv:renditionType="thumbnail">
        <mpv:StillRef mpv:idRef="ID000900"/>
      <mpv:Rendition>
    </mpv:ManifestLink>
    <mpv:Still mpv:id="ID000900" mpv:lastURL="thumbs/album-349sdf293.JPG"/>
...
  </mpv:AssetList>
</file:Manifest>
```

element **mpv:ManifestLink**

diagram



namespace http://ns.osta.org/mpv/1.0/

type **mpv:SimpleAssetBaseType**

children **mpv:DocumentID mpv:ContentID mpv:LastURL nmf:Metadata mpv:Metadata mpv:Related mpv:Rendition**

attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			

mpv:instanceID	xs:anyURI
mpv:documentID	xs:anyURI
mpv:contentID	xs:anyURI
mpv:lastURL	xs:anyURI
mpv:byteOffset	xs:integer
mpv:leaseExpiresDate	xs:date
mpv:leaseDur	xs:float
mpv:leaseID	xs:string

source `<xs:element name="ManifestLink" type="mpv:SimpleAssetBaseType" substitutionGroup="mpv:SimpleAssetBase"/>`

6.6 <mpv:Audio>

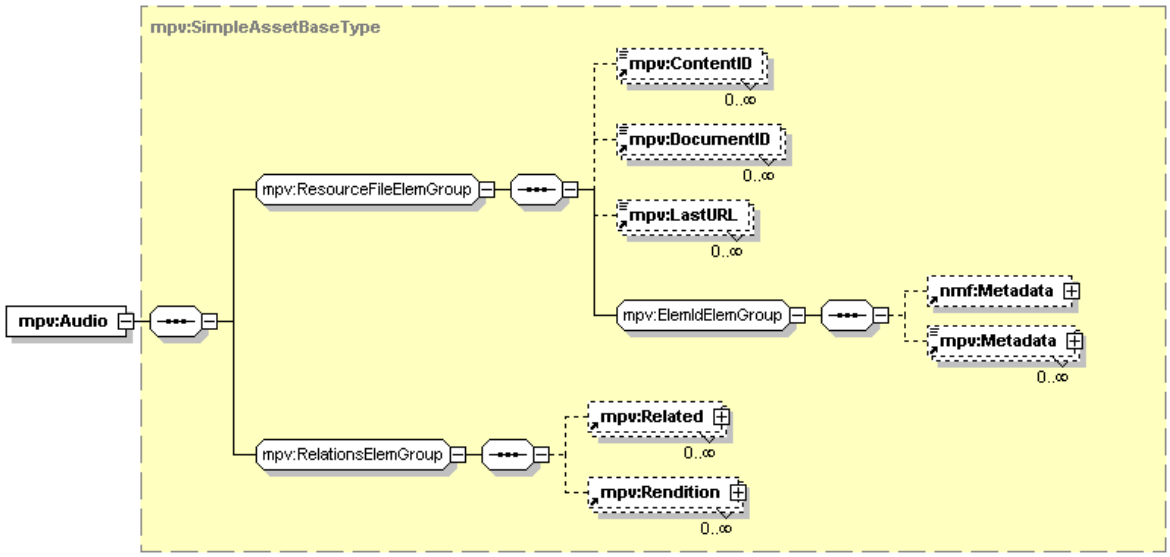
The audio element specifies an audio asset. A typical rendition would be a thumbnail trailer representing the audio track or a "subsampled" resolution representing a lower sampling rate rendition.

Example:

```
...
  <mpv:AssetList>
    <mpv:Audio mpv:id="ID000900">
       <mpv:ContentID>urn:osta-org:mpv:dsig:md5:head:10000:
  EF886AEFA3B340da971BAF09B17DBC122</mpv:ContentID>
       <mpv:LastURL>Waves washing on the shore.wav</mpv:LastURL>
       <mpv:Rendition mpv:renditionUsage="subsampled">
          <mpv:AudioRef mpv:idRef="ID001000"/>
       </mpv:Rendition>
    </mpv:Audio>
    <mpv:Audio mpv:id="ID001000">
       <mpv:ContentID>urn:osta-org:mpv:dsig:md5:head:10000:
  AB893AF0A33B40AD971BFA09B17DBC193</mpv:ContentID>
       <mpv:LastURL>Waves washing on the shore.mp3</mpv:LastURL>
    </mpv:Audio>
  </mpv:AssetList>
...
```

element **mpv:Audio**

diagram



namespace http://ns.osta.org/mpv/1.0/

type **mpv:SimpleAssetBaseType**

children **mpv:DocumentID mpv:ContentID mpv:LastURL nmf:Metadata mpv:Metadata mpv:Related mpv:Rendition**

attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			
mpv:instanceID	xs:anyURI			
mpv:documentID	xs:anyURI			
mpv:contentID	xs:anyURI			
mpv:lastURL	xs:anyURI			
mpv:byteOffset	xs:integer			
mpv:leaseExpiresDate	xs:date			
mpv:leaseDur	xs:float			
mpv:leaseID	xs:string			

source `<xs:element name="Audio" type="mpv:SimpleAssetBaseType" substitutionGroup="mpv:SimpleAssetBase"/>`

6.7 <mpv:Still>

The still element specifies an image asset. Typical renditions would be thumbnail and screen resolution images. When the master is of a media type that is not widely supported, alternate highRes renditions may be provided in alternate formats. The Dublin Core [DC] element "format" may be used to indicate the media type; other elements provide other common properties.

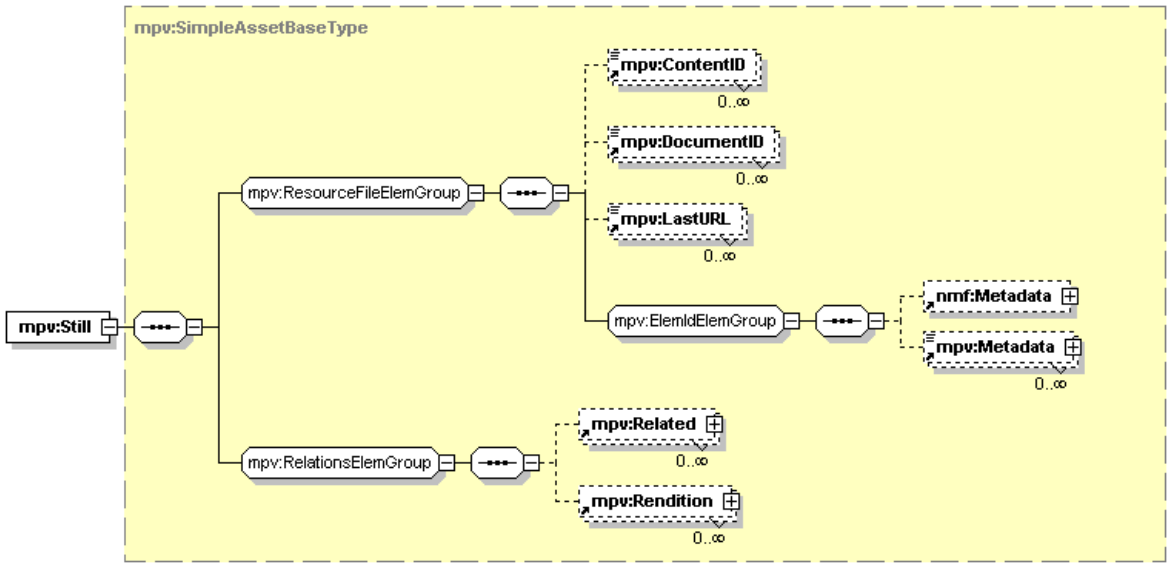
Example:

```
...
  <mpv:AssetList>
    <mpv:Still mpv:id="ID001100"
          mpv:contentID="urn:osta-org:mpv:dsig:md5:all:AB893AF0A33B40AD971BFA09B17DBC193">
       <mpv:LastURL>DSC09346.TIF</mpv:LastURL>
       <nmf:Metadata>
         <Properties xmlns="http://purl.org/dc/elements/1.1/">
            <format>image/tiff</format>
            <title>June 9, 2002, 14:34</title>
         </Properties>
       </nmf:Metadata>
       <mpv:Rendition mpv:renditionUsage="highRes">
          <mpv:StillRef mpv:idRef="ID001200"/>
       </mpv:Rendition>
       <mpv:Rendition mpv:renditionUsage="thumbnail">
          <mpv:StillRef mpv:idRef="ID001300"/>
       </mpv:Rendition>
       <mpv:Rendition mpv:renditionUsage="screen">
          <mpv:StillRef mpv:idRef="ID001400"/>
       </mpv:Rendition>
    </mpv:Still>
    <mpv:Still mpv:id="ID001200"
          mpv:contentID="urn:osta-org:mpv:dsig:md5:all:AB893AF0A33B40AD971BFA09B17DBC145">
       <mpv:LastURL>alt/DSC09346.JPG</mpv:LastURL>
    </mpv:Still>
    <mpv:Still mpv:id="ID001300"
          mpv:contentID="urn:osta-org:mpv:dsig:md5:all:AB893AF0A33B40AD971BFA09B17DBC136">
       <mpv:LastURL>thumbs/DSC09346.JPG</mpv:LastURL>
    </mpv:Still>
    <mpv:Still mpv:id="ID001400"
          mpv:contentID="urn:osta-org:mpv:dsig:md5:all:AB893AF0A33B40AD971BFA09B17DBC192">
```

```
      <mpv:LastURL>screen/DSC09346.JPG</mpv:LastURL>
    </mpv:Still>
  </mpv:AssetList>
...
```

element **mpv:Still**

diagram



namespace http://ns.osta.org/mpv/1.0/

type **mpv:SimpleAssetBaseType**

children **mpv:DocumentID mpv:ContentID mpv:LastURL nmf:Metadata mpv:Metadata mpv:Related mpv:Rendition**

attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			
mpv:instanceID	xs:anyURI			
mpv:documentID	xs:anyURI			
mpv:contentID	xs:anyURI			
mpv:lastURL	xs:anyURI			
mpv:byteOffset	xs:integer			
mpv:leaseExpiresDate	xs:date			
mpv:leaseDur	xs:float			
mpv:leaseID	xs:string			

source

```
<xs:element name="Still" type="mpv:SimpleAssetBaseType" substitutionGroup="mpv:SimpleAssetBase"/>
```

6.8 <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. Each of the component images may also have renditions.

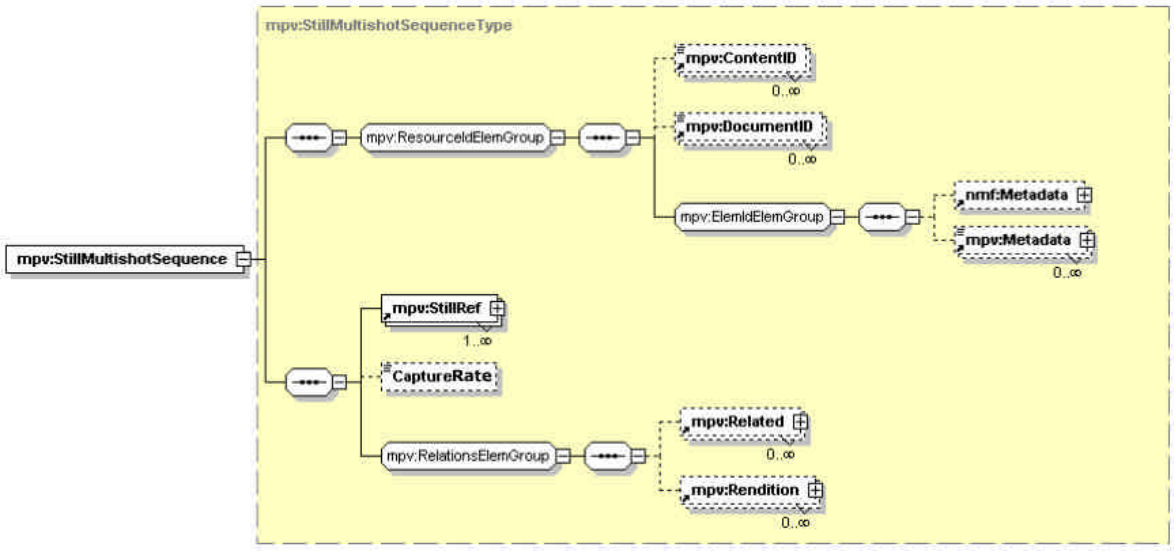
Example:

```
...
 <mpv:AssetList>

  <mpv:StillMultishotSequence mpv:id="ID001400">
```

```
      <mpv:StillRef mpv:idRef="ID001401"/>
      <mpv:StillRef mpv:idRef="ID001404"/>
      <mpv:StillRef mpv:idRef="ID001407"/>
      <mpv:CaptureRate>FrameToFrame:0.3</mpv:CaptureRate>
    </mpv:StillMultishotSequence>

   <mpv:Still mpv:id="ID001401"
      mpv:contentID="urn:osta-org:mpv:dsig:md5:all:EF886AEFA3B340da971BAF09B17DBC122">
      <mpv:LastURL mpv:filesystem="ISO9660-1">ZEBRABUT.JPG</mpv:LastURL>
      <mpv:LastURL mpv:filesystem="Joliet">Zebra butterfly 1.JPG</mpv:LastURL>
      <mpv:LastURL mpv:filesystem="NTFS">Zebra butterfly 1.JPG</mpv:LastURL>
    </mpv:Still>

    <mpv:Still mpv:id="ID001404"
      mpv:contentID="urn:osta-org:mpv:dsig:md5:all:EF886AEFA3B340da971BAF09B17DBC198">
      <mpv:LastURL mpv:filesystem="ISO9660-1">ZEBRAB~1.JPG</mpv:LastURL>
      <mpv:LastURL mpv:filesystem="Joliet">Zebra butterfly 2.JPG</mpv:LastURL>
      <mpv:LastURL mpv:filesystem="NTFS">Zebra butterfly 2.JPG</mpv:LastURL>
    </mpv:Still>

    <mpv:Still mpv:id="ID001407"
      mpv:contentID="urn:osta-org:mpv:dsig:md5:all:EF886AEFA3B340da971BAF09B17DBC166">
      <mpv:LastURL mpv:filesystem="ISO9660-1">ZEBRAB~2.JPG</mpv:LastURL>
      <mpv:LastURL mpv:filesystem="Joliet">Zebra butterfly 3.JPG</mpv:LastURL>
      <mpv:LastURL mpv:filesystem="NTFS">Zebra butterfly 3.JPG</mpv:LastURL>
    </mpv:Still>
  </mpv:AssetList>
...
```

element **mpv:StillMultishotSequence,** complexType **mpv:StillMultishotSequenceType**

diagram



namespace http://ns.osta.org/mpv/1.0/

type **mpv:StillMultishotSequenceType**

children **mpv:DocumentID mpv:ContentID nmf:Metadata mpv:Metadata mpv:StillRef CaptureDur mpv:Related mpv:Rendition**

attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			
mpv:instanceID	xs:anyURI			
mpv:documentID	xs:anyURI			

	mpv:contentID xs:anyURI
source	<xs:element name="StillMultishotSequence" type="mpv:StillMultishotSequenceType" substitutionGroup="mpv:CompositeAssetBase"/>

```
source
<xs:complexType name="StillMultishotSequenceType">
  <xs:complexContent>
    <xs:extension base="mpv:CompositeAssetBaseType">
      <xs:sequence>
        <xs:element ref="mpv:StillRef" maxOccurs="unbounded"/>
        <xs:element name="CaptureRate" type="xs:string" minOccurs="0"/>
        <xs:group ref="mpv:RelationsElemGroup"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

CaptureRate

The value of CaptureRate 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 described in BNF. This BNF conforms to the BNF usage from IETF specifications such as in [URI]. Clock value is always in relative time in seconds to the previous frame.

CaptureRate	=	"FrameToFrame:" (<offset-clock-value> ";")? <frame-clock-value>
offset-clock-value	=	"o" <clock-value>
frame-clock-value	=	<clock-value> (";" <clock-value>)*
clock-value	=	<decimal number> ("." <decimal number>)?
decimal-number	=	[0-9] [0-9]*

There are as many as N clock values for N images. N clock values are possible and not N-1 because the first value optionally can be used to indicate the offset time between an arbitrary timepoint and when the first frame is captured. 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:o3.1;0.4;0.4;0.4": 4 images, the first 3.1 seconds from when timing began, each frame 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.

6.9 <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.

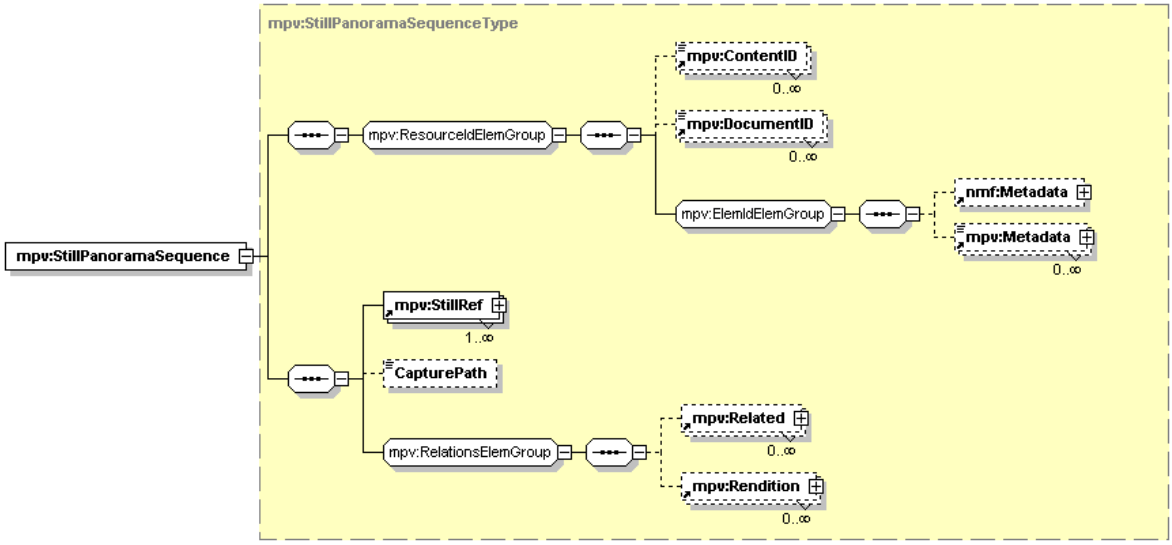
The StillMultishotSequence element groups a sequence of still images and specifies the capture rate. A typical rendition would be a thumbnail representing the still sequence. Each of the component images may also have renditions.

Example:

```
...
 <mpv:AssetList>

  <mpv:StillPanoramaSequence mpv:id="ID000300">
     <mpv:StillRef mpv:idRef="ID001501"/>
     <mpv:StillRef mpv:idRef="ID001504"/>
     <mpv:StillRef mpv:idRef="ID001507"/>
     <mpv:CapturePath>FixedPt:270Y0P0R</mpv:CapturePath>
   </mpv:StillPanoramaSequence>

  <mpv:Still mpv:id="ID001501"
     mpv:contentID="urn:osta-org:mpv:dsig:md5:all:EF886AEFA3B340da971BAF09B17DBC122">
     mpv:lastURL="STA_1039.JPG" />

  <mpv:Still mpv:id="ID001505"
     mpv:contentID="urn:osta-org:mpv:dsig:md5:all:EF886AEFA3B340da971BAF09B17DBC153">
     mpv:lastURL="STB_1040.JPG" />

  <mpv:Still mpv:id="ID001507"
     mpv:contentID="urn:osta-org:mpv:dsig:md5:all:EF886AEFA3B340da971BAF09B17DBC185">
     mpv:lastURL="STC_1041.JPG" />

 </mpv:AssetList>
...
```

element **mpv:StillPanoramaSequence,** complexType **mpv:StillPanoramaSequenceType**

diagram



namespace http://ns.osta.org/mpv/1.0/

type **mpv:StillPanoramaSequenceType**

children **mpv:DocumentID mpv:ContentID nmf:Metadata mpv:Metadata mpv:StillRef CapturePath mpv:Related mpv:Rendition**

attributes	Name	Type	Use	Default	Fixed
	mpv:id	xs:ID			
	mpv:instanceID	xs:anyURI			
	mpv:documentID	xs:anyURI			
	mpv:contentID	xs:anyURI			

source

```
<xs:element name="StillPanoramaSequence" type="mpv:StillPanoramaSequenceType"
substitutionGroup="mpv:CompositeAssetBase"/>
```

source

```
<xs:complexType name="StillPanoramaSequenceType">
  <xs:complexContent>
    <xs:extension base="mpv:CompositeAssetBaseType">
      <xs:sequence>
        <xs:element ref="mpv:StillRef" maxOccurs="unbounded"/>
        <xs:element name="CapturePath" type="xs:string" minOccurs="0"/>
        <xs:group ref="mpv:RelationsElemGroup"/>
      </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 CapturePath syntax described in BNF. This BNF conforms to the BNF usage from IETF specifications such as in [URI]. Motion-vector is in degrees, not radians.

CapturePath	=	"FixedPt:" <motion-vector> (";" <motion-vector>)*
motion-vector	=	`<degrees>Y<degreees>P<degrees>R`
degrees	=	<decimal number> ("." <decimal number>)?
decimal-number	=	[0-9] [0-9]* in the range 0 to 359.9999999

Yaw-Pitch-Roll motions are in positive decimal degrees in 3D space assuming a fixed reference point. There are as many as N-1 motions for N images. The last value provided is reused for all subsequent durations.

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

6.10 <mpv:StillWithAudio>

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

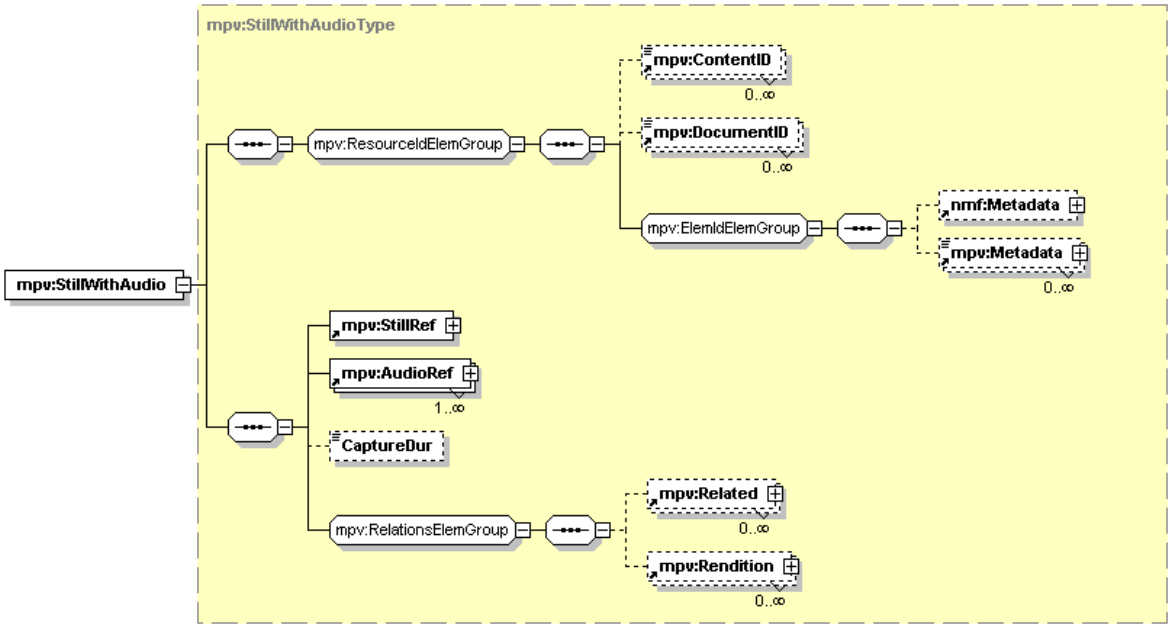
Example:

```
...
  <mpv:AssetList>
```

```
    <mpv:Still mpv:id="ID001100"
          mpv:contentID="urn:osta-org:mpv:dsig:md5:all:AB893AF0A33B40AD971BFA09B17DBC193">
       <mpv:LastURL>DSC09346.TIF</mpv:LastURL>
       <nmf:Metadata>
         <Properties xmlns="http://purl.org/dc/elements/1.1/">
            <format>image/tiff</format>
            <title>June 9, 2002, 14:34</title>
         </Properties>
       </nmf:Metadata>
       <mpv:Rendition mpv:renditionUsage="highRes">
          <mpv:StillRef mpv:idRef="ID001200"/>
       </mpv:Rendition>
       <mpv:Rendition mpv:renditionUsage="thumbnail">
          <mpv:StillRef mpv:idRef="ID001300"/>
       </mpv:Rendition>
       <mpv:Rendition mpv:renditionUsage="screen">
          <mpv:StillRef mpv:idRef="ID001400"/>
       </mpv:Rendition>
    </mpv:Still>
    <mpv:Still mpv:id="ID001200"
          mpv:contentID="urn:osta-org:mpv:dsig:md5:all:AB893AF0A33B40AD971BFA09B17DBC145">
       <mpv:LastURL>alt/DSC09346.JPG</mpv:LastURL>
    </mpv:Still>
    <mpv:Still mpv:id="ID001300"
          mpv:contentID="urn:osta-org:mpv:dsig:md5:all:AB893AF0A33B40AD971BFA09B17DBC136">
       <mpv:LastURL>thumbs/DSC09346.JPG</mpv:LastURL>
    </mpv:Still>
    <mpv:Still mpv:id="ID001400"
          mpv:contentID="urn:osta-org:mpv:dsig:md5:all:AB893AF0A33B40AD971BFA09B17DBC192">
       <mpv:LastURL>screen/DSC09346.JPG</mpv:LastURL>
    </mpv:Still>
  </mpv:AssetList>
...
```

element **mpv:StillWithAudio,** complexType **mpv:StillWithAudioType**

diagram



namespace http://ns.osta.org/mpv/1.0/

type **mpv:StillWithAudioType**

children **mpv:DocumentID mpv:ContentID nmf:Metadata mpv:Metadata mpv:StillRef mpv:AudioRef CaptureDur mpv:Related mpv:Rendition**

attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			
mpv:instanceID	xs:anyURI			
mpv:documentID	xs:anyURI			
mpv:contented	xs:anyURI			

source
```
<xs:element name="StillWithAudio" type="mpv:StillWithAudioType" substitutionGroup="mpv:CompositeAssetBase"/>
```

source
```
<xs:complexType name="StillWithAudioType">
 <xs:complexContent>
  <xs:extension base="mpv:CompositeAssetBaseType">
   <xs:sequence>
    <xs:element ref="mpv:StillRef"/>
    <xs:element ref="mpv:AudioRef" maxOccurs="unbounded"/>
    <xs:element name="CaptureDur" type="xs:string" minOccurs="0"/>
    <xs:group ref="mpv:RelationsElemGroup"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

CaptureDur

The value of CaptureDur indicates the duration of the each audio, in seconds. Clock value is always in relative time, in seconds and decimal portions of a second. A missing clock-value means the duration is unknown.

CaptureDur	=	<clock-value> (“;” <clock-value>)*
clock-value	=	(<seconds> | <unknown-dur>)
unknown-dur	=	the empty string
seconds	=	<decimal number> (“.” <decimal number>)?
decimal-number	=	[0-9] [0-9]*

Example:
"4.3": 4.3 seconds audio duration

"4.3;6.9": two audios, the first 4.3 seconds duration and the second 6.9 seconds

";6.9": two audios, the first of unknown duration, the second 6.9 seconds

6.11 <mpv:Video>

The video element references a video stream of some kind. A typical rendition would be a thumbnail image representing the video. To represent frames extracted from the video, use an “alt” rendition of a StillMultishotSequence of the stills.

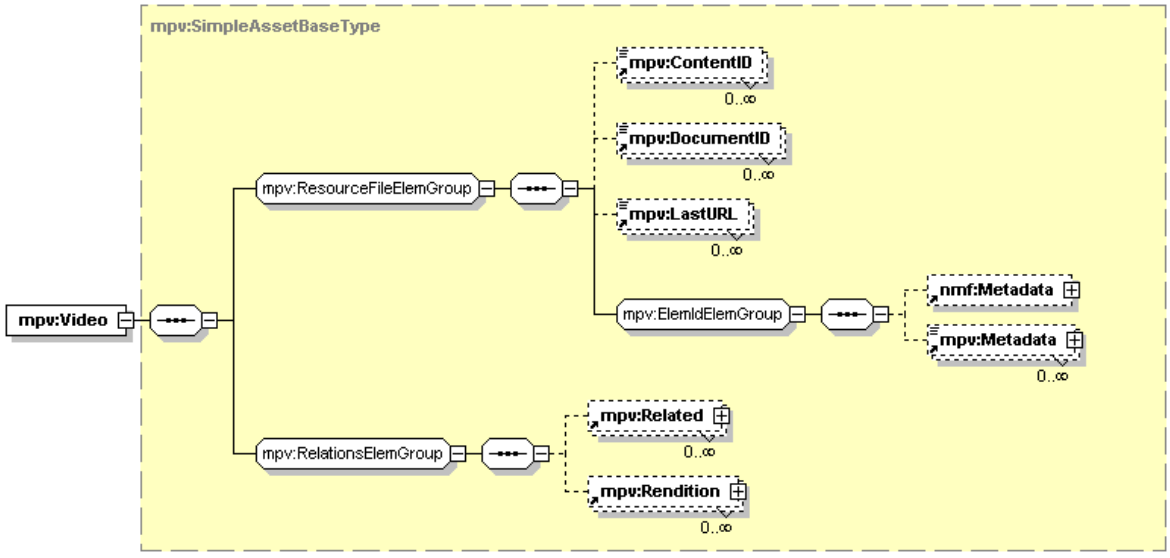
Example:

```
...
  <mpv:AssetList>
    <mpv:Video mpv:id=”ID001700”
          mpv:contentID=”urn:osta-org:mpv:dsig:md5:all:AB893AF0A33B40AD971BFA09B17DBC193”>
       <mpv:LastURL>MOV09324.AVI</mpv:LastURL>
       <mpv:Rendition mpv:renditionUsage=”thumbnail”>
          <mpv:StillRef mpv:idRef=”ID001701”/>
```

```
        </mpv:Rendition>
        <mpv:Rendition mpv:renditionUsage="alt">
           <mpv:StillMultishotSequenceRef mpv:idRef="ID001702"/>
        </mpv:Rendition>
     </mpv:Video>

     <mpv:Still mpv:id="ID001701"
           mpv:contentID="urn:osta-org:mpv:dsig:md5:all:AB893AF0A33B40AD971BFA09B17DBC145">
        <mpv:LastURL>thumbs/MOV09324.JPG</mpv:LastURL>
     </mpv:Still>

     <mpv:StillMultishotSequence mpv:id="ID001702">
       <mpv:StillRef mpv:idRef="ID001703"/>
       <mpv:StillRef mpv:idRef="ID001704"/>
       <mpv:StillRef mpv:idRef="ID001705"/>
       <mpv:CaptureRate>FrameToFrame:o0.3;5.3;2.7</mpv:CaptureRate>
     </mpv:StillMultishotSequence>

     <mpv:Still mpv:id="ID001703"
           mpv:contentID="urn:osta-org:mpv:dsig:md5:all:AB893AF0A33B40AD971BFA09B17DBC136">
        <mpv:LastURL>alt/MOV09324-01.JPG</mpv:LastURL>
     </mpv:Still>
     <mpv:Still mpv:id="ID001704"
           mpv:contentID="urn:osta-org:mpv:dsig:md5:all:AB893AF0A33B40AD971BFA09B17DBC133">
        <mpv:LastURL> alt/MOV09324-02.JPG</mpv:LastURL>
     </mpv:Still>
     <mpv:Still mpv:id="ID001705"
           mpv:contentID="urn:osta-org:mpv:dsig:md5:all:AB893AF0A33B40AD971BFA09B17DBC127">
        <mpv:LastURL> alt/MOV09324-03.JPG</mpv:LastURL>
     </mpv:Still>
   </mpv:AssetList>
...
```

element **mpv:Video**

diagram



namespace http://ns.osta.org/mpv/1.0/

type **mpv:SimpleAssetBaseType**

children **mpv:DocumentID mpv:ContentID mpv:LastURL nmf:Metadata mpv:Metadata mpv:Related mpv:Rendition**

attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			
mpv:instanceID	xs:anyURI			
mpv:documentID	xs:anyURI			
mpv:contentID	xs:anyURI			
mpv:lastURL	xs:anyURI			
mpv:byteOffset	xs:integer			
mpv:leaseExpiresDate	xs:date			
mpv:leaseDur	xs:float			
mpv:leaseID	xs:string			

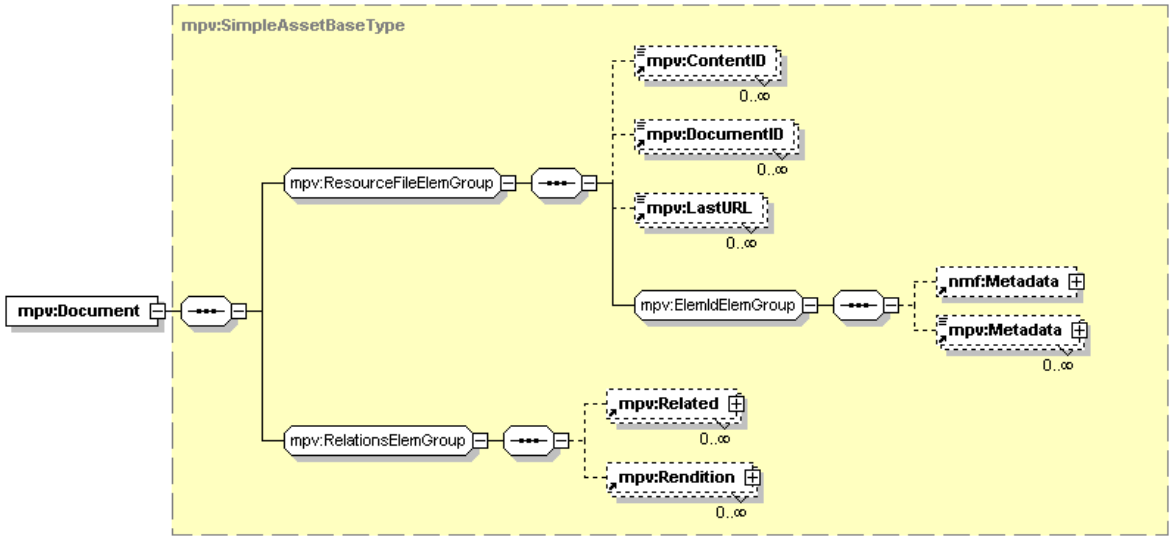
source
```
<xs:element name="Video" type="mpv:SimpleAssetBaseType" substitutionGroup="mpv:SimpleAssetBase"/>
```

6.12 <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..

element **mpv:Document**

diagram



namespace http://ns.osta.org/mpv/1.0/

type **mpv:SimpleAssetBaseType**

children **mpv:DocumentID mpv:ContentID mpv:LastURL nmf:Metadata mpv:Metadata mpv:Related mpv:Rendition**

attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			
mpv:instanceID	xs:anyURI			
mpv:documentID	xs:anyURI			
mpv:contentID	xs:anyURI			
mpv:lastURL	xs:anyURI			
mpv:byteOffset	xs:integer			
mpv:leaseExpiresDate	xs:date			
mpv:leaseDur	xs:float			
mpv:leaseID	xs:string			

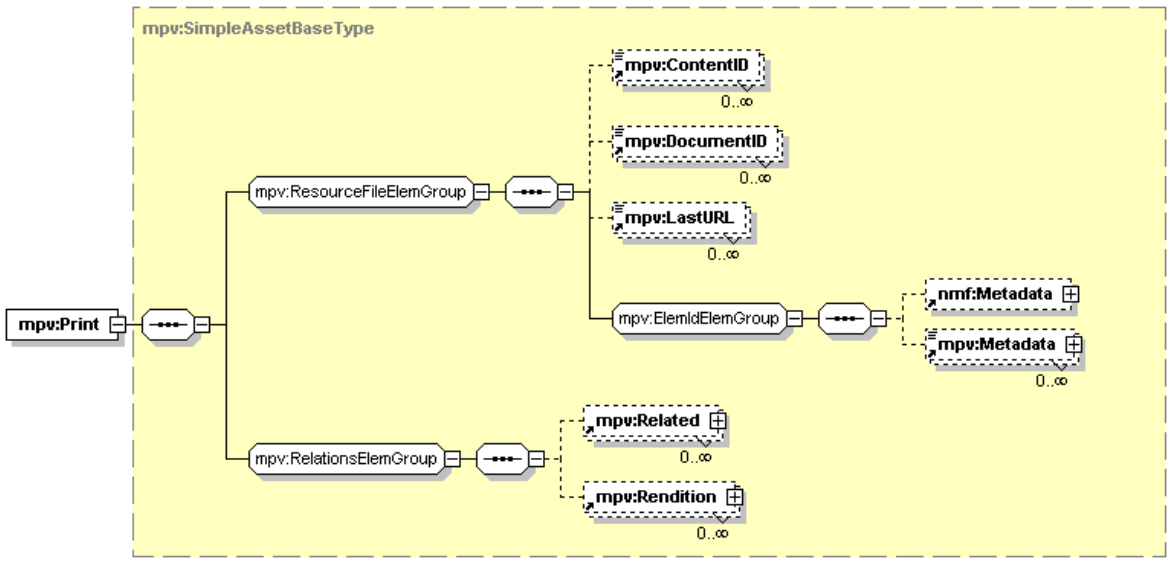
source
```
<xs:element name="Document" type="mpv:SimpleAssetBaseType" substitutionGroup="mpv:SimpleAssetBase"/>
```

6.13 <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.

element **mpv:Print**

diagram



namespace http://ns.osta.org/mpv/1.0/

type **mpv:SimpleAssetBaseType**

children **mpv:DocumentID mpv:ContentID mpv:LastURL nmf:Metadata mpv:Metadata mpv:Related mpv:Rendition**

attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			
mpv:instanceID	xs:anyURI			
mpv:documentID	xs:anyURI			
mpv:contentID	xs:anyURI			
mpv:lastURL	xs:anyURI			
mpv:byteOffset	xs:integer			
mpv:leaseExpiresDate	xs:date			
mpv:leaseDur	xs:float			
mpv:leaseID	xs:string			

source

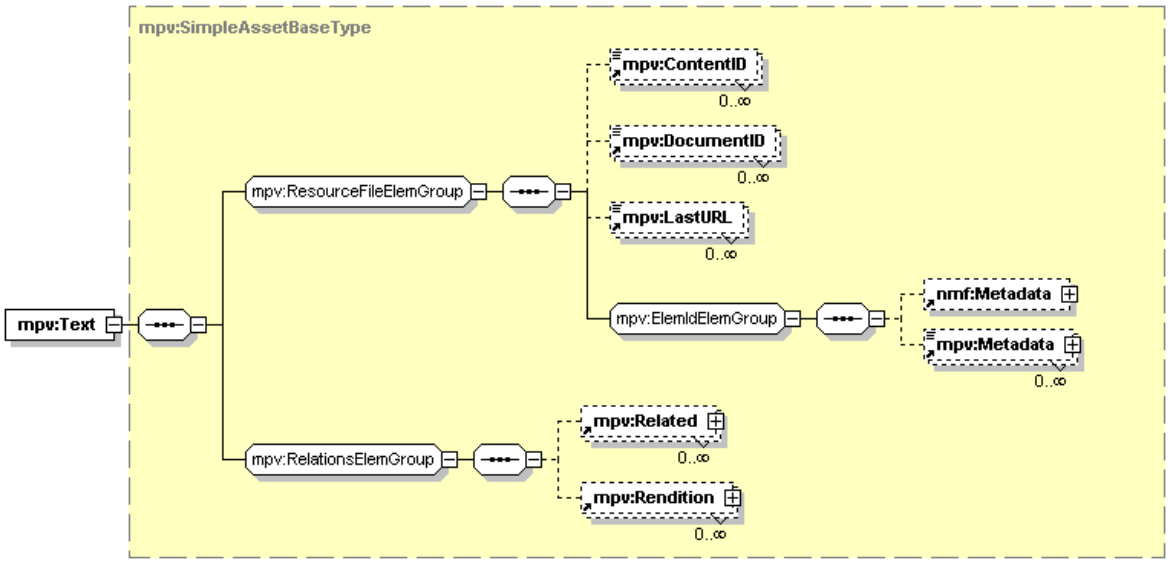
```
<xs:element name="Print" type="mpv:SimpleAssetBaseType" substitutionGroup="mpv:SimpleAssetBase"/>
```

6.14 <mpv:Text>

The text element specifies a document containing text content. The formatting language may be specified by the Dublin core “format” property whose value is a mimetype (see DC-NMF). A typical rendition would be a thumbnail representing the file.

element **mpv:Text**

diagram



namespace http://ns.osta.org/mpv/1.0/

type **mpv:SimpleAssetBaseType**

children **mpv:DocumentID mpv:ContentID mpv:LastURL nmf:Metadata mpv:Metadata mpv:Related mpv:Rendition**

attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			
mpv:instanceID	xs:anyURI			
mpv:documentID	xs:anyURI			
mpv:contentID	xs:anyURI			
mpv:lastURL	xs:anyURI			
mpv:byteOffset	xs:integer			
mpv:leaseExpiresDate	xs:date			
mpv:leaseDur	xs:float			
mpv:leaseID	xs:string			

source

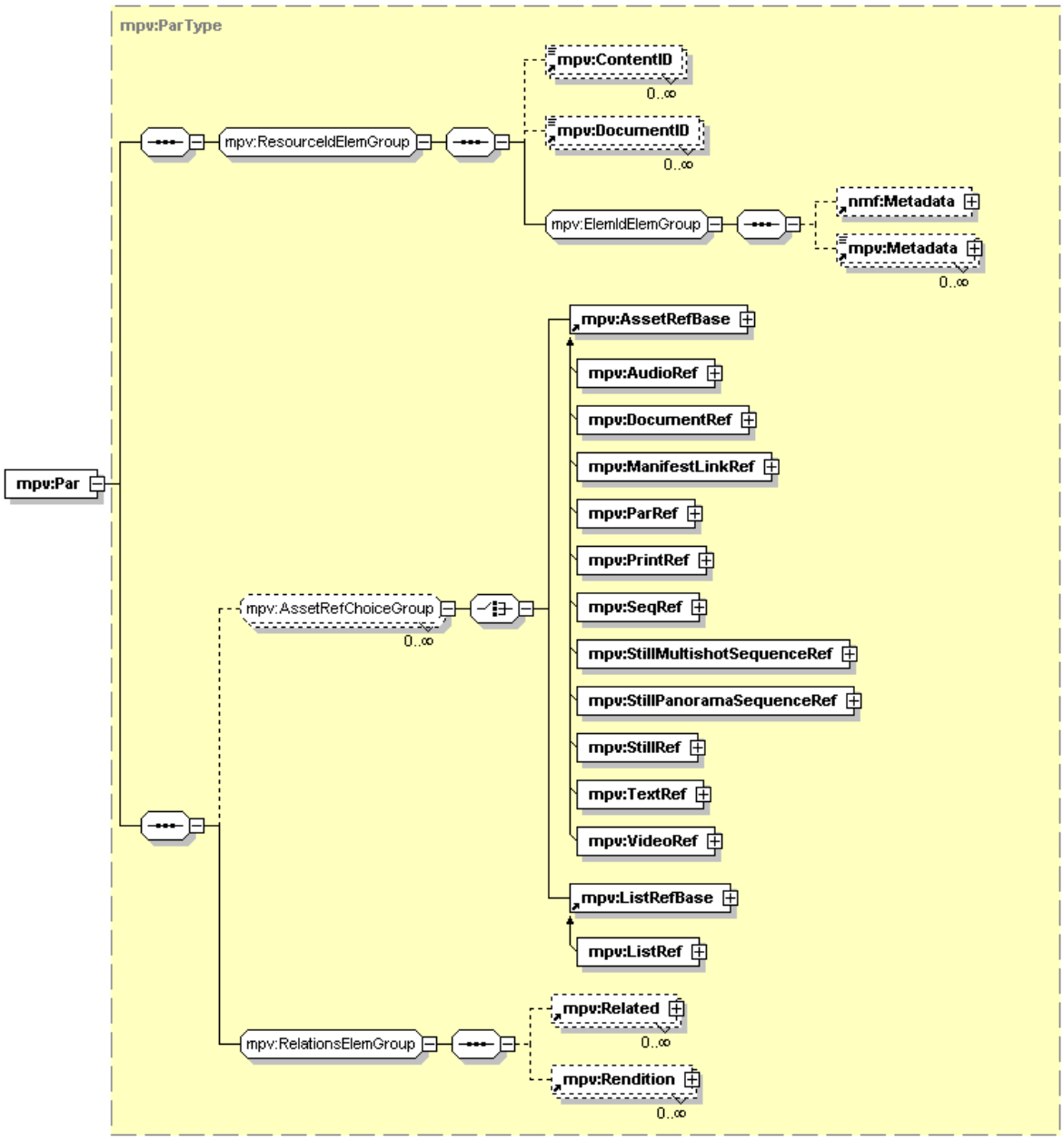
```
<xs:element name="Text" type="mpv:SimpleAssetBaseType" substitutionGroup="mpv:SimpleAssetBase"/>
```

6.15 <mpv:Par>

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

element **mpv:Par,** complexType **mpv:ParType**

diagram



namespace http://ns.osta.org/mpv/1.0/

type **mpv:ParType**

children **mpv:DocumentID mpv:ContentID nmf:Metadata mpv:Metadata mpv:AssetRefBase mpv:ListRefBase mpv:Related mpv:Rendition**

attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			
mpv:instanceID	xs:anyURI			
mpv:documentID	xs:anyURI			
mpv:contentID	xs:anyURI			
hint	xs:anyURI			

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

source
```
<xs:complexType name="ParType">
  <xs:complexContent>
```

```
    <xs:extension base="mpv:CompositeAssetBaseType">
      <xs:sequence>
        <xs:group ref="mpv:AssetRefChoiceGroup" minOccurs="0" maxOccurs="unbounded"/>
        <xs:group ref="mpv:RelationsElemGroup"/>
      </xs:sequence>
      <xs:attribute name="hint" type="xs:anyURI"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

hint

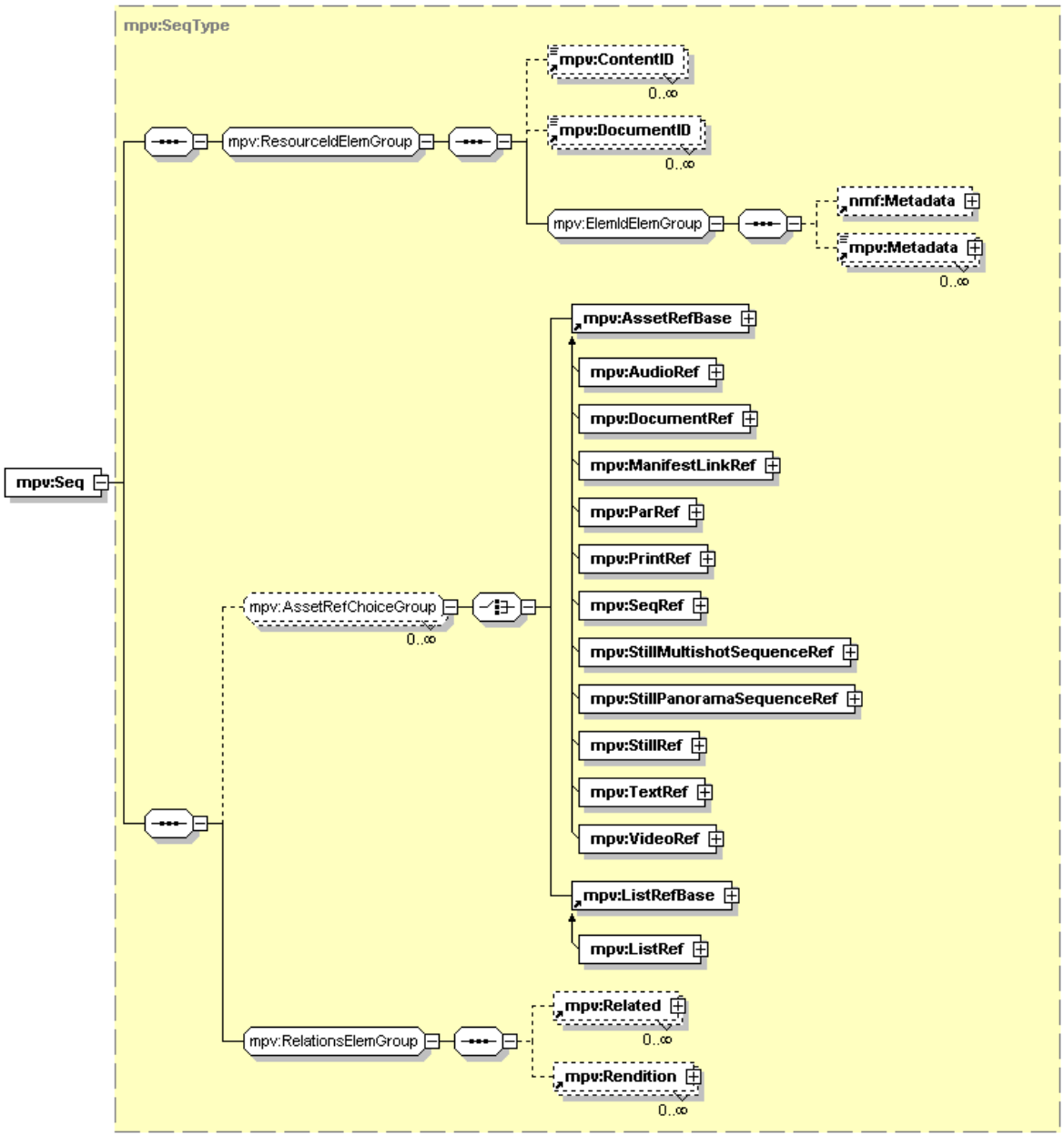
The hint applies to the Par asset and has an open vocabulary. Hints values must use URN-qualified names to avoid the possibility of name collisions, such as "urn:mycompany-com:mpv:someasset".

6.16 <mpv:Seq>

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

element **mpv:Seq,** complexType **mpv:SeqType**

diagram



namespace http://ns.osta.org/mpv/1.0/

type **mpv:SeqType**

children **mpv:DocumentID mpv:ContentID nmf:Metadata mpv:Metadata mpv:AssetRefBase mpv:ListRefBase mpv:Related mpv:Rendition**

attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			
mpv:instanceID	xs:anyURI			
mpv:documentID	xs:anyURI			
mpv:contentID	xs:anyURI			
hint	xs:anyURI			

source
```
<xs:element name="Seq" type="mpv:SeqType" substitutionGroup="mpv:CompositeAssetBase"/>
```

source
```
<xs:complexType name="SeqType">
  <xs:complexContent>
```

```
    <xs:extension base="mpv:CompositeAssetBaseType">
      <xs:sequence>
        <xs:group ref="mpv:AssetRefChoiceGroup" minOccurs="0" maxOccurs="unbounded"/>
        <xs:group ref="mpv:RelationsElemGroup"/>
      </xs:sequence>
      <xs:attribute name="hint" type="xs:anyURI"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

hint

The hint applies to the Seq asset 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:someasset".

6.17 <mpv:AudioRef>, <mpv:StillRef>, <mpv:StillMultishotSequenceRef>, <mpv:StillPanoramaSequenceRef>, <mpv:StillWithAudioRef>, <mpv:ParRef>, <mpv:SeqRef>, <mpv:PrintRef>, <mpv:TextRef>, <mpv:VideoRef>, <mpv:DocumentRef>, <mpv:ManifestLinkRef>

Except in an AssetList, an asset may not be defined directly elsewhere in an Photo/Video Manifest. Instead, reference is made to an asset in the AssetList using one of the <Asset>Ref elements. All the asset ref elements have the same structure.

element **mpv:AudioRef, mpv:StillRef, mpv:StillMultishotSequenceRef, mpv:StillPanoramaSequenceRef, mpv:StillWithAudioRef, mpv:ParRef, mpv:SeqRef, mpv:PrintRef, mpv:TextRef, mpv:VideoRef, mpv:DocumentRef, mpv:ManifestLinkRef**

diagram

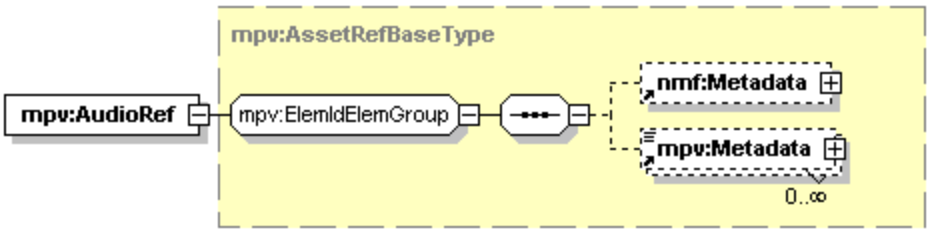


diagram The diagrams are similar for mpv:StillRef, mpv:StillMultishotSequenceRef, mpv:StillPanoramaSequenceRef, mpv:StillWithAudioRef, mpv:ParRef, mpv:SeqRef, mpv:PrintRef, mpv:TextRef, mpv:VideoRef, mpv:DocumentRef, mpv:ManifestLinkRef

namespace http://ns.osta.org/mpv/1.0/

children **nmf:Metadata mpv:Metadata**

attributes

Name	Type	Use	Default	Fixed
manifestLinkIDRef	xs:IDREF	optional		
listIDRef	xs:IDREF	optional		
mpv:id	xs:ID			

idRef	xs:Name	required

source

```
<xs:element name="AudioRef" type="mpv:AssetRefBaseType" substitutionGroup="mpv:AssetRefBase"/>
<xs:element name="DocumentRef" type="mpv:AssetRefBaseType" substitutionGroup="mpv:AssetRefBase"/>
<xs:element name="ManifestLinkRef" type="mpv:AssetRefBaseType" substitutionGroup="mpv:AssetRefBase"/>
<xs:element name="ParRef" type="mpv:AssetRefBaseType" substitutionGroup="mpv:AssetRefBase"/>
<xs:element name="PrintRef" type="mpv:AssetRefBaseType" substitutionGroup="mpv:AssetRefBase"/>
<xs:element name="SeqRef" type="mpv:AssetRefBaseType" substitutionGroup="mpv:AssetRefBase"/>
<xs:element name="StillRef" type="mpv:AssetRefBaseType" substitutionGroup="mpv:AssetRefBase"/>
<xs:element name="StillMultishotSequenceRef" type="mpv:AssetRefBaseType" substitutionGroup="mpv:AssetRefBase"/>
<xs:element name="StillPanoramaSequenceRef" type="mpv:AssetRefBaseType" substitutionGroup="mpv:AssetRefBase"/>
<xs:element name="StillWithAudioRef" type="mpv:AssetRefBaseType" substitutionGroup="mpv:AssetRefBase"/>
<xs:element name="TextRef" type="mpv:AssetRefBaseType" substitutionGroup="mpv:AssetRefBase"/>
<xs:element name="VideoRef" type="mpv:AssetRefBaseType" substitutionGroup="mpv:AssetRefBase"/>
```

idRef

Provides the “mpv:id” value of the referenced asset. When no listIDRef is present, the AssetList in the current manifest is used. When no manifestLinkIDRef is present, the current manifest is used.

listIDRef

Provides the “mpv:id” value of the AssetList or MarkList that contains the referenced asset. When no listIDRef is present, the AssetList in the same manifest is used.

manifestLinkIDRef

Provides the “mpv:id” value of the ManifestLink asset that contains the referenced asset. When no manifestLinkIDRef is present, the current manifest is used.

Chapter 7: MPV Core Schema, Part 3: Metadata

7.1 <nmf:Metadata>

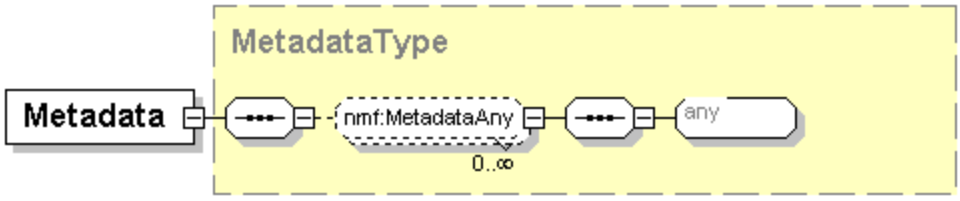
The <nmf:Metadata> element provides a means to utilize schema structured according to the NMF Specification [NMF]. NMF-structured metadata can be mechanically translated across several encodings, including XML Schema, RDF Schema, and SQL databases.

In addition to mechanical interchange of metadata across various encodings, NMF-structured metadata can be validated using standard XML Schema validation tools. This valuable characteristic can be applied to encodings for which validation is less accessible given available tools, such as for RDF-based schema. Metadata documents encoded in NMF can provide validation for many RDF-based content.

MPV makes use of NMF to structure its metadata schema beyond those defined within the MPV Core. This enhances the ability of MPV to take advantage of other widely adopted schema by encoding them in NMF, thus promoting consistency of metadata representation while providing interoperability and validation.

The following schema excerpt from the NMF specification is informative, not normative. It illustrates that the <nmf:Metadata> element can accept any well-structured XML content. However, MPV strictly requires all content of <nmf:Metadata> to be conformant to the NMF specification. The <mpv:Metadata> element is available for use by any well-structured XML content.

Further discussion of the usage of nmf:Metadata can be found in section 9.4.

diagram	
namespace	http://ns.osta.org/nmf/1.0/
type	**MetadataType**
attributes	Name: about; Type: xs:anyURI; Use: optional; Default: ; Fixed:
source	<xs:element name="Metadata" type="MetadataType"/>
source	<xs:complexType name="MetadataType">

	<xs:complexContent> <xs:extension base="CompositePropType"> <xs:sequence> <xs:group ref="nmf:MetadataAny" minOccurs="0" maxOccurs="unbounded"/> </xs:sequence> <xs:attribute name="about" type="xs:anyURI" use="optional"/> </xs:extension> </xs:complexContent> </xs:complexType>
source	<xs:group name="MetadataAny"> <xs:sequence> <xs:any processContents="strict"/> </xs:sequence> </xs:group >

This example uses the Dublin Core metadata specification in NMF format [DC-NMF]. More information about Dublin Core is found in NMF Metadata Practices, section 9.4.

```
<nmf:Metadata>
  <Properties xmlns="http://purl.org/dc/elements/1.1/">
    <creator>Pieter van Zee</creator>
    <description>The definitive specification introducing MultiPhoto/Video.</description>
    <format>application/pdf</format>
    <title>MPV Core Specification 1.0</title>
  </Properties>
  <Properties xmlns="http://purl.org/dc/terms/">
    <created>2002-03-25T21:07:00Z</created>
  </Properties>
</nmf:Metadata>
```

7.2 <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. A typical occurance of such data is to embed useful metadata, such as that defined by JPEG2000 Part 2 **[J2K-Part2]**, DIG35 **[DIG35]** or XMP metadata in native form **[XMP-FW]**, in the MPV document.

element **Metadata**

diagram

namespace http://ns.osta.org/mpv/1.0/

used by group **ElemIdElemGroup**

attributes

Name	Type	Use	Default	Fixed
schemaURI	xs:anyURI			

source

```
<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:attributeGroup ref="mpv:AnyAttrGroup"/>
  </xs:complexType>
</xs:element>
```

MPV does not recommend the use of any of the particular metadata schema mentioned or any others for use with the mpv:Metadata element. The following summaries and examples are strictly informative.

JPEG2000 Part 2 and DIG35 provide (almost) identical means for specifying five kinds of metadata:

- Basic image parameter metadata, such as the image size and number of components
- Image creation metadata, such as camera and lens information and capture condition
- Content description metadata, such as the "who", "what", "when" and "where" aspect of the image.
- History metadata, to provide *partial information* about *how* the image got to the present state
- Intellectual property rights metadata, to either protect the rights of the owner of the asset or provide further information to request permission to use it.

XMP provides a variety of schema, including content description metadata and intellectual property rights metadata.

Example of embedded JPEG2000 metadata:

```
<mpv:Metadata mpv:schemaURI="http://www.jpeg.org/jpx"
              xmlns="http://www.jpeg.org/jpx">
    <BASIC_IMAGE_PARAM>
       <BASIC_IMAGE_INFO>
         <IMAGE_SIZE>
           <WIDTH>1600</WIDTH>
           <HEIGHT>1200</HEIGHT>
         </IMAGE_SIZE>
      </BASIC_IMAGE_INFO>
    </BASIC_IMAGE_PARAM>
</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>
    <BASIC_IMAGE_PARAM>
       <BASIC_IMAGE_INFO>
         <IMAGE_SIZE>
           <WIDTH>1600</WIDTH>
           <HEIGHT>1200</HEIGHT>
         </IMAGE_SIZE>
      </BASIC_IMAGE_INFO>
    </BASIC_IMAGE_PARAM>
  </METADATA>
</mpv:Metadata>
```

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 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>
```


Chapter 8: MPV Core Schema, Part 4: Base Types

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

- **SimpleAssetBaseType, CompositeAssetBaseType**: Provides the base types for all MPV media assets to inherit from.
- **AssetChoiceGroupType**: Provides the set of known assets to choose among.
- **AssetRefChoiceGroupType**: Provides references to the set of known assets to choose among.
- **RelationsItemGroup**: Provides the Related and Rendition elements..
- **ListRef, ListRefBase, ListRefBaseType**: Provides references to a list of assets or assetRefs.
- **AssetRefBase, AssetRefBaseType**: Basis for each asset reference.

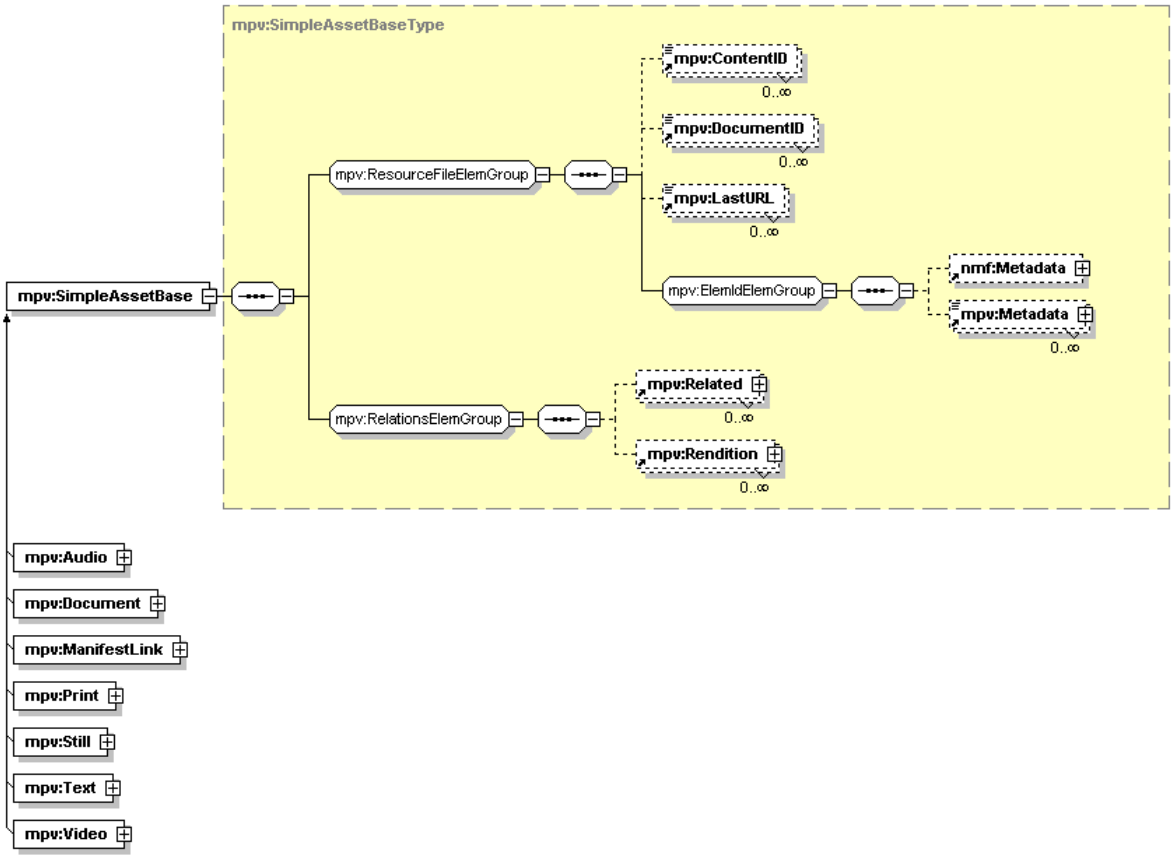
8.1 Types: SimpleAssetBase, SimpleAssetBaseType

mpv:SimpleAssetBase is an abstract type that is the base type for all simple media assets. New simple assets can be created by substituting for mpv:SimpleAssetBase. It is not used directly as an element.

Simple assets 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 asset are the same as identity values of the referenced file or resource.

element **mpv:SimpleAssetBase**

diagram



namespace http://ns.osta.org/mpv/1.0/

type **mpv:SimpleAssetBaseType**

children **mpv:DocumentID mpv:ContentID mpv:LastURL nmf:Metadata mpv:Metadata mpv:Related mpv:Rendition**

used by group **mpv:AssetChoiceGroup**

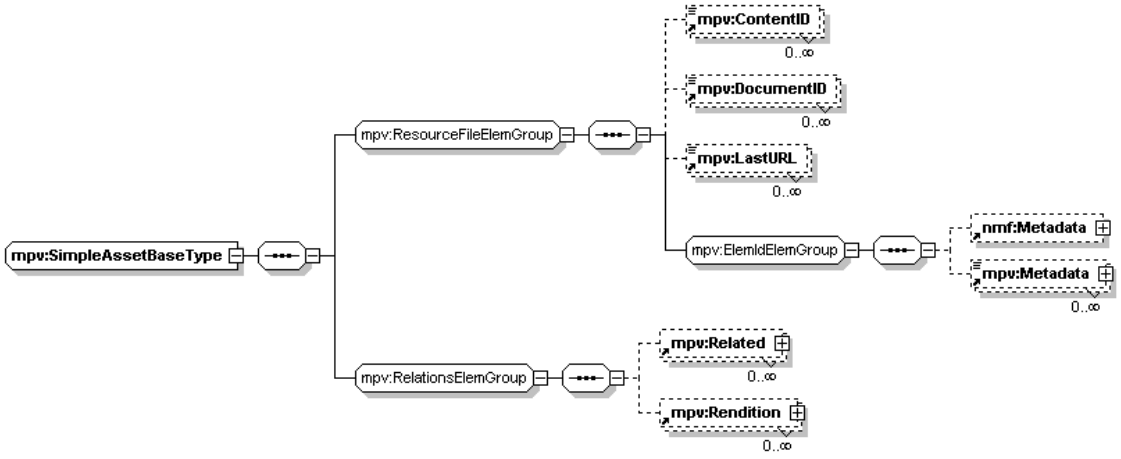
attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			
mpv:instanceID	xs:anyURI			
mpv:documentID	xs:anyURI			
mpv:contentID	xs:anyURI			
mpv:lastURL	xs:anyURI			
mpv:byteOffset	xs:integer			
mpv:leaseExpiresDate	xs:date			
mpv:leaseDur	xs:float			
mpv:leaseID	xs:string			

source
```
<xs:element name="SimpleAssetBase" type="mpv:SimpleAssetBaseType" abstract="true"/>
```

complexType **mpv:SimpleAssetBaseType**

diagram



namespace http://ns.osta.org/mpv/1.0/

children **mpv:DocumentID mpv:ContentID mpv:LastURL nmf:Metadata mpv:Metadata mpv:Related mpv:Rendition**

used by elements **mpv:Audio mpv:Document mpv:ManifestLink mpv:Print mpv:SimpleAssetBase mpv:Still mpv:Text mpv:Video**

attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			
mpv:instanceID	xs:anyURI			
mpv:documentID	xs:anyURI			
mpv:contentID	xs:anyURI			
mpv:lastURL	xs:anyURI			
mpv:byteOffset	xs:integer			
mpv:leaseExpiresDate	xs:date			
mpv:leaseDur	xs:float			
mpv:leaseID	xs:string			

source

```
<xs:complexType name="SimpleAssetBaseType">
  <xs:sequence>
    <xs:group ref="mpv:ResourceFileElemGroup"/>
    <xs:group ref="mpv:RelationsElemGroup"/>
  </xs:sequence>
  <xs:attributeGroup ref="mpv:ResourceFileAttrGroup"/>
</xs:complexType>
```

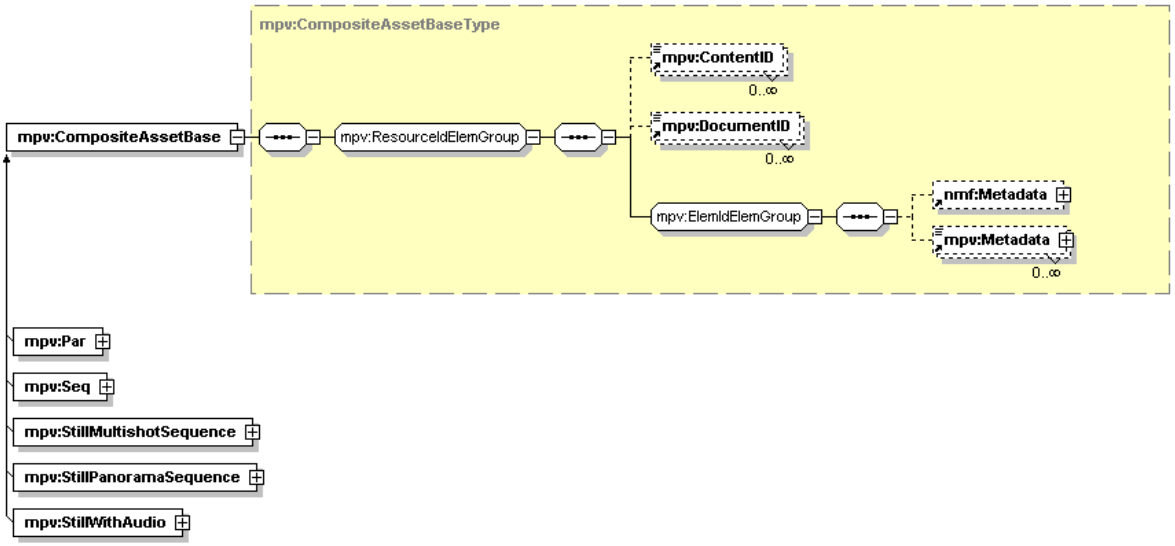
8.2 Types: CompositeAssetBase, CompositeAssetBaseType

mpv:CompositeAssetBase is an abstract type that is the base type for all composite media assets. New composite assets can be created by substituting for mpv:CompositeAssetBase. It is not used directly as an element.

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

element **mpv:CompositeAssetBase**

diagram



namespace http://ns.osta.org/mpv/1.0/

type **mpv:CompositeAssetBaseType**

children **mpv:DocumentID mpv:ContentID nmf:Metadata mpv:Metadata**

used by group **mpv:AssetChoiceGroup**

attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			
mpv:instanceID	xs:anyURI			
mpv:documentID	xs:anyURI			
mpv:contentID	xs:anyURI			

source
```
<xs:element name="CompositeAssetBase" type="mpv:CompositeAssetBaseType" abstract="true"/>
```

source
```
<xs:complexType name="CompositeAssetBaseType">
  <xs:sequence>
    <xs:group ref="mpv:ResourceIdElemGroup"/>
  </xs:sequence>
  <xs:attributeGroup ref="mpv:ResourceIdAttrGroup"/>
</xs:complexType>
```

8.3 Groups: AssetChoiceGroup, AssetRefChoiceGroup

The MPV specification defines a collection of media assets The AssetChoiceGroup defines the set of available media assets in MPV. This is not used directly as a top-level element.

group **AssetChoiceGroup**

diagram

namespace http://ns.osta.org/mpv/1.0/

children **mpv:SimpleAssetBase mpv:CompositeAssetBase**

used by complexType **mpv:AssetListType**

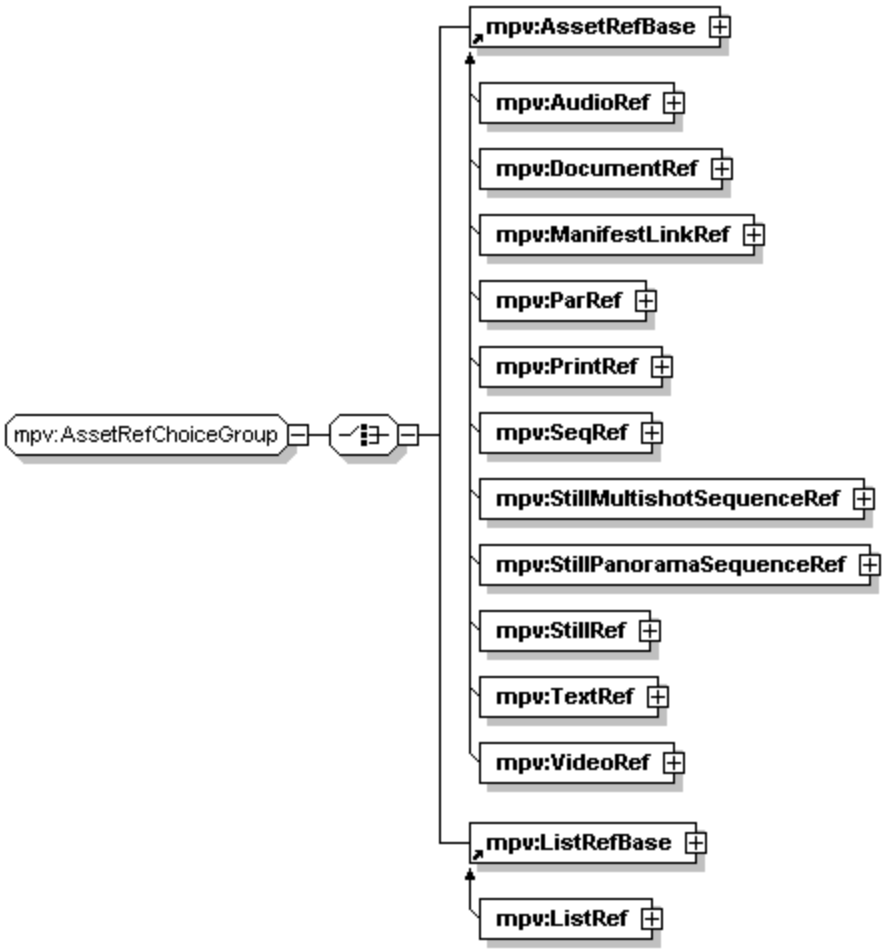
source

```
<xs:group name="AssetChoiceGroup">
  <xs:choice>
    <xs:element ref="mpv:SimpleAssetBase"/>
    <xs:element ref="mpv:CompositeAssetBase"/>
  </xs:choice>
</xs:group>
```

Similarly, the AssetRefChoiceGroup defines the set of available asset references in MPV. It is not used directly as a top-level element.

group **AssetRefChoiceGroup**

diagram



namespace http://ns.osta.org/mpv/1.0/

children **mpv:AssetRefBase mpv:ListRefBase**

used by complexTypes **mpv:AssetRefListBaseType mpv:ParType mpv:RelatedType mpv:RenditionType mpv:SeqType**

source

```
<xs:group name="AssetRefChoiceGroup">
  <xs:choice>
    <xs:element ref="mpv:AssetRefBase"/>
    <xs:element ref="mpv:ListRefBase"/>
  </xs:choice>
</xs:group>
```

8.4 Types: ListRef, ListRefBase, ListRefBaseType

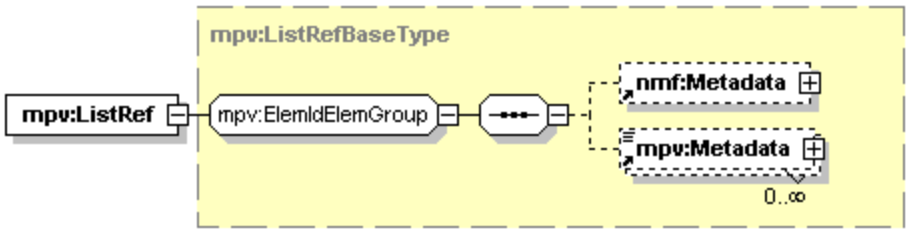
The ListRefBase is used as one of the kinds of references to an asset. It is constructed an an extension of ListRefBaseType and intended to be substituted by specializations. These types are not used directly as elements.

A reference to a list in an arbitrary file may be made using the mpv:ListRef. The ListRef may be used in the mpv:AssetRefChoiceGroup where the ListRefBase element is used.

The mpv:ListRef element is a substitution for ListRefBase and can be used wherever it is used, principally in the mpv:AssetRefChoiceGroup.

element **mpv:ListRef**

diagram



namespace http://ns.osta.org/mpv/1.0/

type **mpv:ListRefBaseType**

children **nmf:Metadata mpv:Metadata**

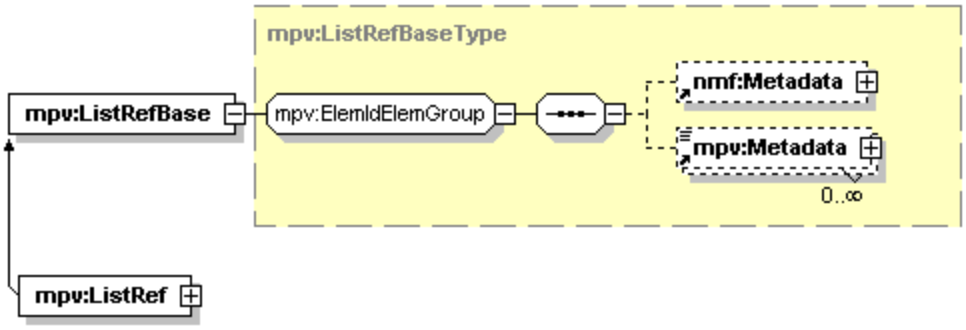
attributes

Name	Type	Use	Default	Fixed
manifestIDRef	xs:IDREF	optional		
listIDRef	xs:IDREF	optional		
mpv:id	xs:ID			

source
```
<xs:element name="ListRef" type="mpv:ListRefBaseType" substitutionGroup="mpv:ListRefBase"/>
```

element **mpv:ListRefBase ;** complexType **mpv:ListRefBaseType**

diagram



namespace http://ns.osta.org/mpv/1.0/

type **mpv:ListRefBaseType**

children **mpv:Metadata mpv:Metadata**

used by group **mpv:AssetRefChoiceGroup**

attributes

Name	Type	Use	Default	Fixed
manifestLinkIDRef	xs:IDREF	optional		
listIDRef	xs:IDREF	optional		
mpv:id	xs:ID			

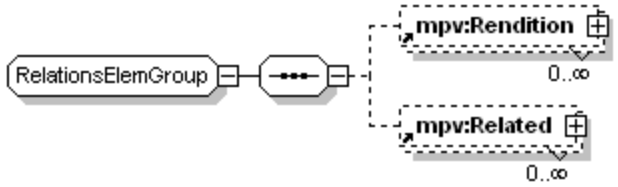
source
```
<xs:element name="ListRefBase" type="mpv:ListRefBaseType" abstract="true"/>
```

source
```
<xs:complexType name="ListRefBaseType">
  <xs:group ref="mpv:ElemIdElemGroup"/>
  <xs:attribute name="manifestLinkIDRef" type="xs:IDREF" use="optional"/>
  <xs:attribute name="listIDRef" type="xs:IDREF" use="optional"/>
  <xs:attributeGroup ref="mpv:ElemIdAttrGroup"/>
  <xs:attributeGroup ref="mpv:AnyAttrGroup"/>
</xs:complexType>
```


8.5 Groups: RelationsElemGroup

The MPV specification defines the ability to list renditions and related content for media assets. The RelationsElemGroup defines the set of related media assets in MPV. This is not used directly as a top-level element.

group **RelationsElemGroup**

diagram	
namespace	http://ns.osta.org/mpv/1.0/
children	**Rendition Related**
used by	complexTypes **AlbumType ParType SeqType SimpleAssetBaseType StillMultishotSequenceType StillPanoramaSequenceType StillWithAudioType**
source	<xs:group name="RelationsElemGroup"> <xs:sequence> <xs:element ref="mpv:Rendition" minOccurs="0" maxOccurs="unbounded"/> <xs:element ref="mpv:Related" minOccurs="0" maxOccurs="unbounded"/> </xs:sequence> </xs:group>

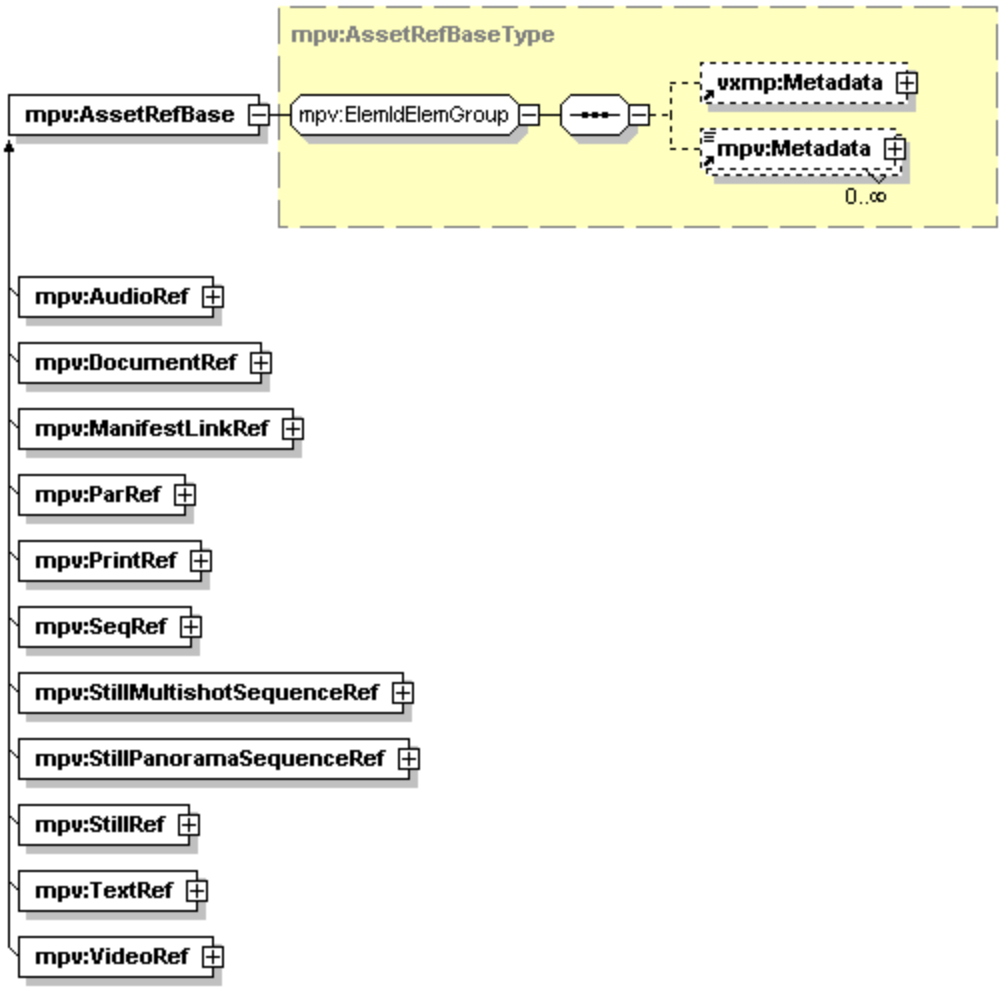
8.6 Types: AssetRefBase, AssetRefBaseType

The mpv:AssetRefBase is the mechanism for extending the set of known references types. It provides an abstract base type that can be substituted with concrete types. The set of concrete types defined by the Core specification are shown.

The AssetRefBaseType is an extension to ListRefBaseType that accommodates references to assets in separate files. The idRef attribute value must identify a unique element in the referenced list, which may be in any location as specified by the ResourceFileElemGroup and ResourceFileAttrGroup members.

element **mpv:AssetRefBase, mpv:AssetRefBaseType**

diagram



namespace http://ns.osta.org/mpv/1.0/

type **mpv:AssetRefBaseType**

type extension of **mpv:ListRefBaseType**

children **nmf:Metadata mpv:Metadata**

used by
elements **mpv:AssetRefBase mpv:AudioRef mpv:DocumentRef mpv:ManifestLinkRef mpv:ParRef mpv:PrintRef mpv:SeqRef mpv:StillMultishotSequenceRef mpv:StillPanoramaSequenceRef mpv:StillRef mpv:StillWithAudioRef mpv:TextRef mpv:VideoRef**

attributes

Name	Type	Use	Default	Fixed
manifestLinkIDRef	xs:IDREF	optional		
listIDRef	xs:IDREF	optional		
mpv:id	xs:ID			
idRef	xs:Name	required		

source
```
<xs:element name="AssetRefBase" type="mpv:AssetRefBaseType" abstract="true"/>
```

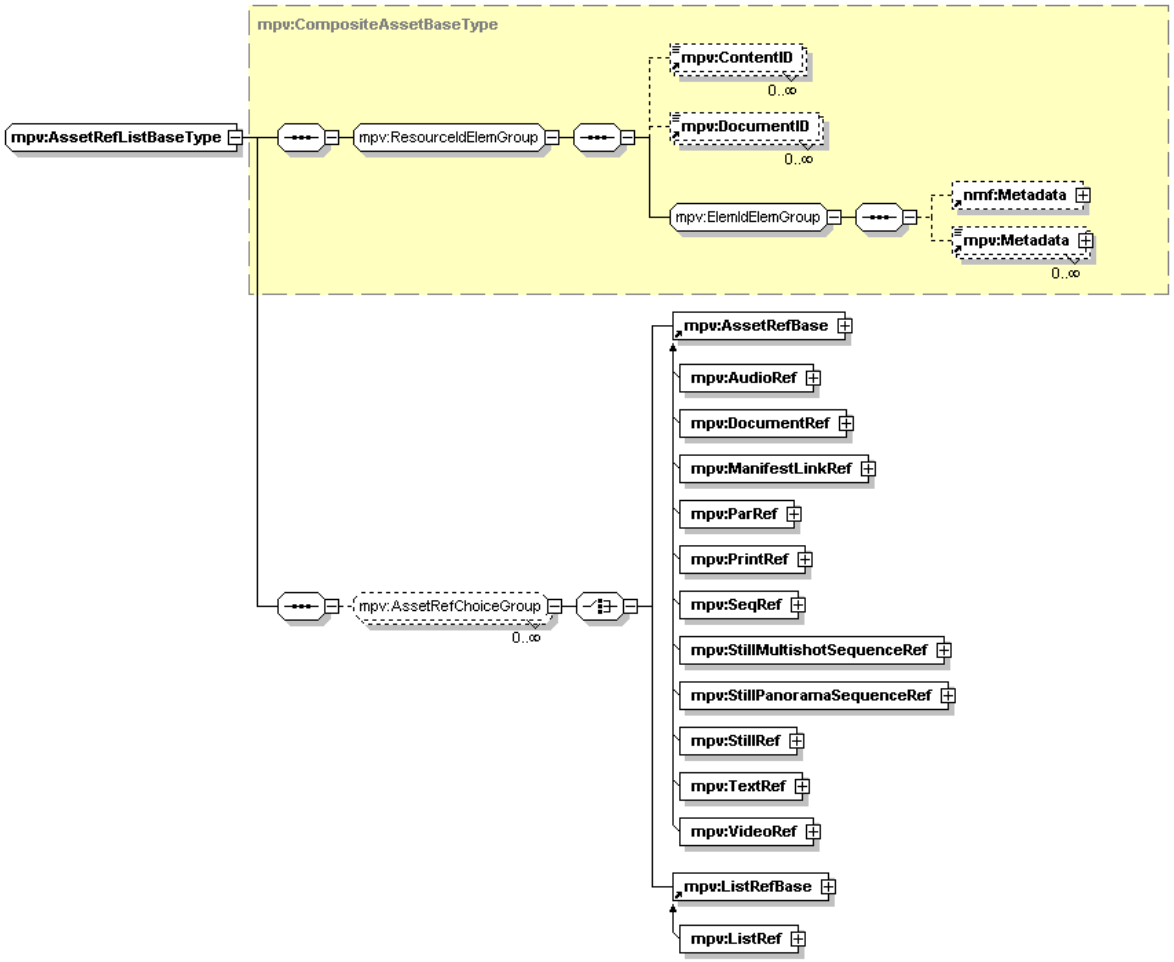
source
```
<xs:complexType name="AssetRefBaseType">
 <xs:complexContent>
  <xs:extension base="mpv:ListRefBaseType">
   <xs:attribute name="idRef" type="xs:Name" use="required" />
   <xs:attributeGroup ref="mpv:AnyAttrGroup"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

8.7 Type: AssetRefListBaseType

The AssetRefListBaseType is a composite asset that can be used to define a list of assets by reference. The defaultListIDRef attribute identifies the AssetList that is the default to be used for references to assets.

complexType **mpv:AssetRefListBaseType**

diagram



namespace http://ns.osta.org/mpv/1.0/

type extension of **mpv:CompositeAssetBaseType**

children **mpv:DocumentID mpv:ContentID nmf:Metadata mpv:Metadata mpv:AssetRefBase mpv:ListRefBase**

used by complexType **mpv:MarkListType**

attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			
mpv:instanceID	xs:anyURI			
mpv:documentID	xs:anyURI			
mpv:contentID	xs:anyURI			
defaultListIDRef	xs:IDREF	optional		

	defaultManifestLinkIDRef	xs:IDREF	optional

source
```
<xs:complexType name="AssetRefListBaseType">
 <xs:complexContent>
  <xs:extension base="mpv:CompositeAssetBaseType">
   <xs:sequence>
    <xs:group ref="mpv:AssetRefChoiceGroup" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="defaultListIDRef" type="xs:IDREF" use="optional"/>
   <xs:attribute name="defaultManifestLinkIDRef" type="xs:IDREF" use="optional"/>
   <xs:attributeGroup ref="mpv:AnyAttrGroup"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

8.8 Type: ManifestChildBaseType

All MPV elements that may occur at the top-level of an OSTA XML Manifest are extensions of mpv:ManifestChildType. This allows for improved validation of Manifest content.

element **mpv:ManifestChildBase**

diagram

namespace http://ns.osta.org/mpv/1.0/

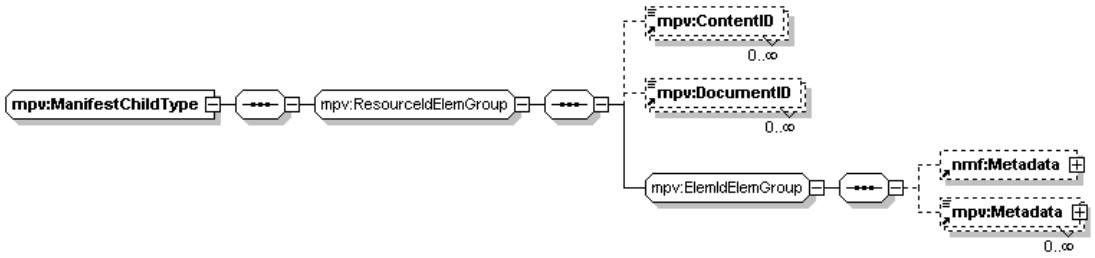
type **mpv:ManifestChildBaseType**

source
```
<xs:element name="ManifestChildBase" type="mpv:ManifestChildBaseType" abstract="true"/>
```

source
```
<xs:complexType name="ManifestChildBaseType"/>
```

complexType **mpv:ManifestChildType**

diagram



namespace http://ns.osta.org/mpv/1.0/

type extension of **mpv:ManifestChildBaseType**

children **mpv:ContentID mpv:DocumentID Metadata mpv:Metadata**

used by
complexType **mpv:AssetListType**

attributes

Name	Type	Use	Default	Fixed
mpv:id	xs:ID			
mpv:instanceID	xs:anyURI			
mpv:documentID	xs:anyURI			
mpv:contentID	xs:anyURI			

source
```
<xs:complexType name="ManifestChildType">
```

```
    <xs:complexContent>
      <xs:extension base="mpv:ManifestChildBaseType">
        <xs:sequence>
          <xs:group ref ="mpv:ResourceIdElemGroup"/>
        </xs:sequence>
        <xs:attributeGroup ref ="mpv:ResourceIdAttrGroup"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

Chapter 9: MPV Core Practices

The MPV specification defines use of 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 compliance test plans and logo certification requirements can guide or require implementation of many of these practices.

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 asset to a collection and when resolving an asset 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, without separating dashes.

MD5

MPV distinguishes the MD5 algorithm as the basic content ID algorithm that preferentially should be supported first by a processing application. 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, without separating dashes.

9.1.2 Identifier Insertion and Extraction

MPV does not define technology practices for the insertion and extraction identifiers in asset files.

9.1.3 Identifier Computation and Naming

MPV defines practices for computing and naming identifiers. MPV applications shall preferentially support these techniques first 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:md5:all
- urn:osta-org:mpv:dsig:md5:body
- urn:osta-org:mpv:dsig:md5:head:<byte count>
- urn:osta-org:mpv:dsig:md5:tail:<byte count>

Because MD5 operates blindly on bytestreams, all MD5 practices are well defined for all files except for the "body" algorithm. The "body" MD5 signature behaviour must be defined for each file type, as the meaning of "body" is file type-specific.

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 md5:all signature is recommended because it is simple and robust for read-only content. Also, this value can be used to determine if a file has been changed since it was referenced. The md5:body signature is also recommended because it is based on the media content of the file and is more robust than the "all" signature because it is less subject to damage by metadata editing.
2. When the referenced file already has an embedded instanceID that is a UUID, 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 embedded instanceID, one may be provided by computing a new UUID. If a new instanceID is generated, also may 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 an embedded documentID, 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, it may 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 producing or consuming MPV documents. The following best practices are recommended.

9.2.1 MPV Producers

Two situations need to be considered with respect to pathname references made in MPV documents.

(a) the MPV manifest moves along with the assets it refers to
(b) the MPV manifest moves independently of the assets it refers to

RELATIVE PATHNAMES

In case (a), the recommended practice is to use relative pathnames for LastURL values. Relative pathnames represent paths to local files relative to an implicit base location. This allows an MPV collection and its related files to be moved around within a filesystem without breaking the LastURL references.

BASE URI FOR RELATIVE PATHNAMES

The Base URI for relative pathnames is calculated using the resolution algorithm described in section 5.1 of [URI].

Within this context, the recommended mechanism for specifying a Base URI that is embedded in an MPV document is to use the LastURL property from the IdentProperties schema (see section 5.12), which describes the location of the manifest file inside the manifest file itself. This usage is further discussed in section 10.3.

XMLBASE USE DISCOURAGED

A more recent development for specifying the Base URI of relative pathnames is to use the xml:base attribute. [XMLBase] provides a mechanism for specifying a base URI for relative pathnames. While MPV applications MAY support the embedded URI resolution defined in [XMLBase] because MPV is a compliant XML specification, this usage is discouraged.

ABSOLUTE PATHNAMES

Absolute pathnames may also be provided as the primary or alternate LastURL values. This is one mechanism to accommodate case (b), in which the manifest moves independently of the assets.

MULTIPLE LASTURL ELEMENTS

Many MPV manifests will be used on storage media (such as CDs) with multiple filesystems overlaid on the same underlying content. To robustly reference a file via multiple filesystems may require multiple LastURL values, one for each filesystem. In addition, both relative and absolute pathnames may be provided.

PUT IDENTIFIERS ON THE PATH AS ARGUMENTS

In addition to specifying identifier values as attributes of an element, some applications may also put the identifiers as arguments on the lastURL value. This allows for MPV-aware file handling APIs 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 asset 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. However, the filesystem attribute is a hint and not grounds for ignoring other LastURL values.

The recommended use of all lastURL attribute and elements present is to try them in the order of longest filename string to shortest. This minimizes the risk of using a LastURL that resolves to the wrong file due to OS-supplied aliasing. This problem has been encountered with the hidden 8.3 filenames on Microsoft Windows operating systems that unexpectedly resolve to a valid file (but not the desired one) when testing the validity of a gropu of LastURLs.

STRIP OFF SUFFIX 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.

STRIP OFF PREFIX QUALIFIERS FOR LOCAL FILENAME REFERENCES

LastURL values for local filenames may include prefix qualifiers, such as "file://". Some file handling APIs may not support this syntax. Try stripping off the prefix 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 assets 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 in some contexts, lastURL references will break regularly and require frequent fixing. This can occur due 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 identifier values that are available in the MPV collection to re-establish connection to the referenced asset. 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 assets on demand.

A basic fixup algorithm will scan for candidate files in limited locations, such as the current working directory. An 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, leaving just a directory 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 InstanceID
     If the filetype is conducive to fast lookup of embedded InstanceIDs
          Do UUID-based identification test
If the MPV item needing fixup has a MD5-based ContentID
     Do MD5-based identification test
If the MPV item needing fixup has other id algorithms known to the processing app
     Do those id tests
For each candidate file
     If doing UUID-based id test
          look for UUID-based instanceID 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 instanceID and contentID values for target files be cached. This will allow subsequent fixup of other lastURL values to be very fast.

9.4 NMF Metadata Practices

The MPV core specification employs NMF based metadata for several different aspects of the content that it captures.

The areas where NMF based metadata is used include:

- **ManifestProperties** are defined in the OSTA XML Manifest specification [MANIFEST]. An example of their usage is shown in the section 10.3.
- **IdentProperties** are used to provide analogues to all the native MPV identifiers for use in metadata such as at the Manifest level where MPV native elements are not available. They are described in section 5.12.
- **Dublin Core properties** are used to provide the most common types of descriptive metadata. These can be applied usefully to almost all elements of an MPV document.

9.4.1 Dublin Core Metadata

MPV based profiles can make use of the NMF encoding of the properties defined by the Dublin Core metadata initiative [DCMI]. This encoding is fully described in the Dublin Core – NMF specification [DC-NMF]. The MPV Core specification provides additional constraints and guidelines on the use of the properties defined in the DC-NMF specification. These constraints and guidelines are described below.

The Dublin Core Element Set [DCES] defines fifteen properties that can be used to describe resources. Some of these properties may not be clearly enough defined to allow a high level of interoperability to occur between loosely coupled participants in a metadata interchange scenario.

DCMI is providing increased clarity on the the contents on encoding of the DCES via the definition of qualifiers that either refine the meaning of the core elements or nail down the details of the encoding of the core elements.

MPV recommends that specific properties be used in a specific manner in order to maximize interoperability between MPV applications. Applications may choose to employ any of the properties defined in the DC-NMF specification using any of the alternative typing mechanisms although this may decrease the level of interoperability between MPV applications.

The following properties are recommended for usage by MPV applications. They are described in more detail below along with examples. The property descriptions specify what additional constraints are placed on the property above and beyond those defined in the DC-NMF specification [DC-NMF].

The following namespace prefixes are employed in the following material:

Namespace Identifier	Namespace Prefix
http://purl.org/dc/elements/1.1/	dc:
http://purl.org/dc/terms/	dcterms:

RECOMMENDED PROPERTIES

- dc:creator
- dc:description
- dc:format
- dc:title
- dcterms:modified
- dcterms:created

Below is an example of Dublin Core NMF metadata. It contains properties from both the dc and dcterms schemas.

```
<nmf:Metadata>
  <Properties xmlns="http://purl.org/dc/elements/1.1/">
    <creator>Gabe Beged-Dov</creator>
    <description>The definitive specification of the NMF-structured encoding of Dublin Core
 metadata schema.
    </description>
    <format>application/pdf</format>
    <title>Dublin Core NMF Specification</title>
  </Properties>
  <Properties xmlns="http://purl.org/dc/terms/">
    <created>2002-03-25T21:07:00Z</created>
  </Properties>
</nmf:Metadata>
```

9.4.2 dc:creator

MPV applications SHOULD only make use of the simple property type variant of dc:creator. The contents of the simple property type SHOULD be an un-interpreted value of type xs:string.

9.4.3 dc:description

MPV applications SHOULD only make use of the simple property type variant of dc:description. The contents of the simple property type SHOULD be an un-interpreted value of type xs:string.

9.4.4 dc:format

MPV applications SHOULD only make use of the simple property type variant of dc:format. The contents of the simple property type SHOULD be interpreted as a MIME-type.

9.4.5 dc:title

MPV applications SHOULD only make use of the simple property type variant of dc:title. The contents of the simple property type SHOULD be an un-interpreted value of type xs:string.

9.4.6 dcterms:created

MPV applications SHOULD only make use of the simple property type variant of dcterms:created. The contents of the simple property type SHOULD be interpreted as conforming to the xs:dateTime datatype (see Appendix IV:)

9.4.7 dcterms:modified

MPV applications SHOULD only make use of the simple property type variant of dcterms:created. The contents of the simple property type SHOULD be interpreted as conforming to the xs:dateTime datatype(see Appendix IV:) .

9.5 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.

9.5.1 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:

DATACD (ORANGEBOOK) 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: provides 255 character Unicode filenames. Depending on the version of UDF and mode in which the media was written, the disc is accessible from a shifting variety of operating systems and hardware.

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.

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.

VIDEOCD (WHITE BOOK) WITH MPV CONTENT

It is possible to place a photo-video collection with Photo/Video 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.

9.5.2 DVD Best Practices

DVDs are fundamentally data discs that use the UDF 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.

DVD-VIDEO WITH MPV CONTENT

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

9.5.3 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.

9.5.4 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>.PVM`
- `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.

9.6 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 assets. 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.

MPV does not provide practices for accessing or storing metadata embedded in digital asset files beyond the ability provided to reference embedded content.

Chapter 10: Photo/Video Manifest (PVM) File Practices

10.1 The OSTA XML Manifest and Photo/Video Manifest

MPV utilizes the OSTA XML Manifest specification **[MANIFEST]** to provide a standard XML wrapper in which to place MPV content and to provide for extensibility while interoperability with a wide variety of applications.

WHAT IS THE MANIFEST?

In typical usage, a manifest is stored in a stand-alone file. OSTA defines a manifest that enables multiple applications to store and retrieve their own and other data in the same manifest. Any appliation that produces or consumes MPV content stored in stand-alone files in a storage filesystem shall be compliant with the Manifest schema and practices specification.

By convention, the top-level element of a manifest is called <file:Manifest>. This allows other profiles to place their content within the manifest without any implication that this is a MPV-specific manifest.

It is important to recognize the purpose of the <file:Manifest> wrapper element. A wrapper element is required of all XML documents. MPV utilizes the OSTA XML Manifest wrapper element because it can be conveniently recognized by MPV-aware applications. Because the MPV AssetList schema is well-defined and a core part of all MPV documents, it provides a useful point of interoperability across MPV-aware applications. While different MPV applications may not understand all the Profiles produced by each other, they can in all cases share the basic AssetList data. This provides basic interoperability of MPV collections across any application.

PHOTO/VIDEO MANIFEST

An OSTA XML Manifest is considered a Photo/Video Manifest (PVM) when it contains a <mpv:AssetList> first-level child element. A Photo/Video Manifest always contains an asset list and may contain zero or more additional peer elements which are defined by MPV or other profiles. By implication of terminology, a Photo/Video 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.

It is important to recognize that many manifests will contain other content in addition to MPV content. This is expected and appropriate. MPV provides the basis for represention and interchange of photo-video collections; it does not intent to be a comprehensive representation of all possible useful data.

In typical usage, a Photo/Video 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 schema and practices specification.

10.2 MPV Profiles To Use mpv:ManifestChildBaseType

All MPV profiles that define top-level elements to be placed in a manifest arrange for them to derive from mpv:ManifestChildBaseType. In MPV Core, this is only the <mpv:AssetList> element. There are no other rules in MPV regarding the design of Profile schema, but consistency with existing MPV design practices is recommended.

Profiles are one of the most important units of modular extension of MPV. Any number of profiles can co-exist within an MPV document. Profiles can consist of additional metadata attached to any MPV element, or they can add additional MPV elements at several levels of the MPV Core schema, including new asset types and new top-level elements that are children of the <file:Manifest> element

10.3 Specifying Manifest Profiles and LastURL

The <file:Manifest> element is the outer element of a OSTA XML manifest. It wraps the <mpv:AssetList> and also any number of additional elements defined by Profiles. The manifest's top-level metadata schema should be produced and processed by every MPV-aware application. It lists the profiles for which the document contains data. Applications that do not understand any given Profile schema at any level should leave it untouched and carry it forward.

NMF metadata associated with the <file:Manifest> element provides a means to identify easily all the Profiles implemented in the manifest. The manifest properties also provides a "Redirect" property that instructs a manifest processor to redirect its processing to a different file.

In addition, the NMF metadata can contain identification information for the manifest which is specified using the IdentProperties schema described in section 5.12. This usage is discussed in section 9.2.1. The LastURL property mirrors the mpv:LastURL element into the NMF encoding. The value of the URL sub-property of LastURL should be an absolute URL that can be used as the base URI value to absolutize any relative URL in the manifest.

This supports the following scenario. A directory hierarchy contains both the assets and a manifest that refers to those assets. The references to the assets all use relative pathnames. The Manifest includes an IdentProperties entry that provides its current location as an absolute URL. The manifest is then moved by the user to another location while the assets are left behind. When the manifest is processed, the relative URL values for the assets needs to be absolutized. The application can generate an absolute path to the assets by using the value of the LastURL in the Manifest's IdentProperties as the base URI.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<file:Manifest
        xmlns:file="http://ns.osta.org/manifest/1.0/"
        xmlns:mpv="http://ns.osta.org/mpv/1.0/"
        xmlns:nmf="http://ns.osta.org/nmf/1.0/"
        xmlns:Profile1="http://www.companyA.com/Profile1/1.0/"
        xmlns:Profile2="http://www.companyB.com/Profile2/3.5/" >
  <nmf:Metadata>
    <ManifestProperties xmlns="http://ns.osta.org/manifest/1.0/">
      <ProfileBag>
```



```
          <Profile>http://ns.osta.org/mpv/1.0/</Profile>
          <Profile>http://www.companyA.com/Profile1/1.0/</Profile>
          <Profile>http://www.companyB.com/Profile2/3.5/</Profile>
       <ProfileBag>
    </ManifestProperties>
    <IdentProperties xmlns="http://ns.osta.org/mpv/1.0/ident/">
       <LastURL>
          <LastURLProperties xmlns="http://ns.osta.org/mpv/1.0/ident/lasturl/">
              <Filesystem>JOLIET</Filesystem>
              <URL>file:///e:/album.pvm</URL>
          </LastURLProperties>
       </LastURL>
    </IdentProperties>
  </nmf:Metadata>

  </nmf:Metadata>

  <Profile1:Outer1>
     …
  </Profile1:Outer1>

  <Profile2:Outer2>
     …
  </Profile2:Outer2>

  <mpv:AssetList>
     …
  </mpv:AssetList>

</file:Manifest>
```

10.4 Creating Profiles Using <mpv:ManifestChildBase>, <mpv:ManifestChildBaseType>, <mpv:ManifestChildType>

The mechanism for defining MPV extensions of the manifest in a profile is the mpv:ManifestChildBase. A MPV profile commonly will substitute a new definition of mpv:ManifestChildBase. This creates a new child element of the manifest for use by that profile.

10.5 Finding an Photo/Video Manifest File

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

When searching a file system for a Photo/Video Manifest, they can be located by name or by extension. When requested by name, the manifest is either found or not found. If not found, the algorithm defined elsewhere for lastURL fixup should be applied.

The MPV Core defines the following algorithmthat describes how to locate a MPV manifest 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" Photo/Video 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 Photo/Video 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.
          INDEX.PVM
          INDEXMPV.XML
          ALBUM.PVM
          ALBUMMPV.XML
          <any name>.PVM, 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 Photo/Video
Manifest encountered, it is opened and scanned for an MPV Album (Presentation Profile) or
AssetList.  The first MPV Album encountered is used for presentation; if none is found, the
AssetList 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 Photo/Video 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 assets 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 Photo/Video 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 ALBUMMPV.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 Photo/Video Manifest file names.

10.6 Photo/Video Manifest File Types

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

.pvm

This extension identifies a file to be a **P**hoto/**V**ideo **M**anifest (PVM). Ideally, we would have used the ".mpv" extension but there are already existing uses of this extension on several platforms that would conflict with this choice.

Usage is case insensitive. This extension may be registered by an application to provide default and alternate processors of Photo/Video Manifests.

.xml

This extension identifies a file as containing XML content. Usage is case insensitive. A Photo/Video 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 Photo/Video Manifests on Macintosh systems. Apple no longer requires file type registration.

.pvm

This Apple Macintosh file type identifies the file to contain a Photo/Video Manifest. Usage is case sensitive. This extension may be registered by an application to provide a default processor of Photo/Video 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, but 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 Photo/Video Manifest. This requires parsing the document to locate the outer element defined by the manifest schema.

10.7 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 Photo/Video Manifest are:

application/vnd.osta-org.pvm+xml

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

application/xml

This MIME media type identifies the content as containing XML content. Usage is case sensitive. A Photo/Video 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.

10.8 Choosing Which File Type and MIME Media Type to Use

For products authoring Photo/Video 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 **.pvm** file extension and **application/vnd.osta-org.pvm+xml** media type registered to an application. A MPV-unaware environment will not.

Generally speaking, it is preferable to use a Photo/Video 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 asset to fixup the lastURL value.

Appendix I: Media Types Reference

This appendix is informative and provides a useful set of media types that can be employed to identify assets using the **[DC-NMF]** "format" property. The value of the "format" property is defined by the MIME Media Types **[MIME-2]** encoding for representing the media types of assets.

The selection of mimetypes contained in this appendix was chosen based on performing a filtering operation on the set of mimetypes that are specified in the IANA registry **[MIMETYPES-REG]** and those that are actively supported by various popular playback environments such as web browsers.

Note that while a particular profile may require that a specific set of media types be supported by compliant authoring and playback environments, the MPV Core specification does not mandate any particular media types.

The following is the set of media types distinguished by MPV for recognition and consistent use.

MEDIA TYPES FOR MPV:STILL

MIME Media Type	Mac File Type	PC File Suffixes	Description
image/bmp	BMPp	bmp	Microsoft Windows bitmap
image/gif	GIFf	gif	GIF 87 format
image/jpeg	JPEG	jpg, jpeg, jpe, jfif, pjpeg	Exif or Jfif encoded JPEG-compressed image
image/jp2	'jp2\040'	jp2	JPEG2000 Part 1 format
image/jpx	'jpx\040'	jpx	JPEG2000 Part 2 format. Note that if the JPX file is also JP2 compliant, it may be specified as a JP2 file.
image/png		png	Portable Network Graphics
image/tiff	TIFF	tif, tiff	TIFF, unknown version
image/x-pict	PICT	pic	Apple Macintosh PICT

MEDIA TYPES FOR MPV:VIDEO

MIME Media Type	Mac File Type	PC File Suffixes	Description
video/avi			Windows AudioVideoInterleave format
video/DV			IEC 61834 consumer DV and professional SMPTE 306M and 314M (DV-Based)

video/mpeg	MPEG	mpeg, mpg, mpe	MPEG1 and MPEG2 video
video/quicktime	MooV	qt, mov	Apple Quicktime video
video/x-msvideo		AVI	Windows AudioVideoInterleave format
video/x-ms-wmv		wmv	Windows Media Video

MEDIA TYPES FOR MPV:AUDIO

MIME Media Type	Mac File Type	PC File Suffixes	Description
audio/basic	ULAW	au, snd	8K, mono audio
audio/midi	MIDI	mid, midi	Musical Instrument Digital Interface sound file
audio/mpeg	MPEG	mp1, mp2, mp3	MPEG audio layers 1, 2, and 3
audio/wav	WAVE	wav	WAVE file
audio/x-aiff	AIFF	aif, aiff	Audio interchange file format
audio/x-ms-wma		wma	Windows Media Audio

MEDIA TYPES FOR MPV:TEXT

MIME Media Type	Mac File Type	PC File Suffixes	Description
text/html	HTML	htm, html	HTML content
text/plain	TEXT	txt	plain text

MEDIA TYPES FOR MPV:PRINT

MIME Media Type	Mac File Type	PC File Suffixes	Description / Application
application/pdf	PDF	pdf	Adobe Acrobat
application/postscript		ps, eps, ai	Adobe Postscript

I.1 Embedded Media Types

MPV supports the use of embedded media types using extensions to the MIME-types mechanism and URI fragment identifiers. The predefined embedded MIME-types are hosted in the OSTA namespace using its vendor root: "vnd.osta-org". New embedded MIME-types can be defined using compatible mechanisms in any namespace that complies with MIME naming conventions; "vnd.osta-org" MIME types MUST be reserved for use by OSTA.

The BNF below specifies the grammar for the fragment identifier used by MPV compliant embedded media types. This BNF conforms to the BNF usage from IETF specifications such as in [URI].

```
mpv-fragment        =   embedtype       [ locator ]
embedtype           =   (osta-root | other-root) "." embed-specific-part
embed-specific-part =   <the identifier for the embedded media type>
```


		This identifier must conform to the NCName production of XML
osta-root	=	"vnd.osta-org"
locator	=	"(" embed-specific-location-scheme ")"
embed-specific-location-scheme	=	<a well-formed locator >
		The locator must have balanced parentheses in order to allow the processor to recognize the end of the locator.

MPV defines two embedded media types that are shown below for Exif **[Exif2002]** embedded assets.

EMBEDDED MEDIA TYPES FOR MPV:STILL

MIME Media Type	Mac File Type	PC File Suffixes	Description
image/vnd.osta-org.exif-thumb	JPEG	jpg, jpeg, jpe, jfif, pjpeg	Exif embedded thumbnail

The fragment identifier for this media type uses the same syntax as the media type.

EMBEDDED MEDIA TYPES FOR MPV:AUDIO

MIME Media Type	Mac File Type	PC File Suffixes	Description
audio/vnd.osta-org.exif-audio	JPEG	jpg, jpeg, jpe, jfif, pjpeg	Exif embedded audio annotation

The fragment identifier for this media type uses the same syntax as the media type.

In this example, the MPV document describes a single Exif file containing the original image, an embedded thumbnail, and an embedded audio. The fragment identifier syntax for the Exif thumbnail and audio is compliant with the usage described above.

```
..
  <mpv:StillWithAudio mpv:id="ID000100">
     <mpv:Still mpv:idRef="ID000101"/>
     <mpv:Audio mpv:idRef="ID000102"/>
  </mpv:StillWithAudio>

  <mpv:Still mpv:id="ID000101">
     <mpv:LastURL>IMG001.JPG</mpv:LastURL>
     <mpv:Rendition mpv:renditionUsage="thumbnail">
         <mpv:StillRef mpv:idRef="ID000103" />
     </mpv:Rendition>
  </mpv:Still>

  <mpv:Audio mpv:id="ID000102">
    <mpv:LastURL>IMG001.JPG#vnd.osta-org.exif-audio</mpv:LastURL>
    <nmf:Metadata>
       <dc:Properties>
           <dc:format>image/vnd.osta-org.exif-audio</dc:format>
       </dc:Properties>
    </nmf:Metadata>
  </mpv:Audio>

  <mpv:Still mpv:id="ID000103" mpv:instanceID="AC937BCFA3B340da971BAF09B17DBC324">
    <mpv:LastURL>IMG001.JPG#vnd.osta-org.exif-thumb</mpv:LastURL>
    <nmf:Metadata>
       <dc:Properties>
```



```
            <dc:format>image/vnd.osta-org.exif-thumb</dc:format>
        </dc:Properties>
      </nmf:Metadata>
    </mpv:Still>
...
```

Appendix II: MPV Schema Source Files

A complete set of the source files for the MPV Core specification are available from OSTA via http://www.osta.org/mpv/

Appendix III: 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 .

III.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.

III.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]);
}
```

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

Definition of MD5 “body” identifiers is not available in this specification version.

Appendix IV: XML Schema Datatypes

XML Schema datatypes are used in several places by MPV. A key datatype that MPV employs is the dataTime datatype. For convenience, we have excerpted section 3.2.6 of the specification [XSchema2] below:

> [Definition:] **dateTime** represents a specific instant of time. The ·value space· of **dateTime** is the space of *Combinations of date and time of day* values as defined in § 5.4 of **[ISO 8601]**.

Lexical representation

A single lexical representation, which is a subset of the lexical representations allowed by **[ISO 8601**], is allowed for **dateTime**. This lexical representation is the **[ISO 8601]** extended format CCYY-MM-DDThh:mm:ss where "CC" represents the century, "YY" the year, "MM" the month and "DD" the day, preceded by an optional leading "-" sign to indicate a negative number. If the sign is omitted, "+" is assumed. The letter "T" is the date/time separator and "hh", "mm", "ss" represent hour, minute and second respectively. Additional digits can be used to increase the precision of fractional seconds if desired i.e the format ss.ss... with any number of digits after the decimal point is supported. The fractional seconds part is optional; other parts of the lexical form are not optional. To accommodate year values greater than 9999 additional digits can be added to the left of this representation. Leading zeros are required if the year value would otherwise have fewer than four digits; otherwise they are forbidden. The year 0000 is prohibited.

The CCYY field must have at least four digits, the MM, DD, SS, hh, mm and ss fields exactly two digits each (not counting fractional seconds); leading zeroes must be used if the field would otherwise have too few digits.

This representation may be immediately followed by a "Z" to indicate Coordinated Universal Time (UTC) or, to indicate the time zone, i.e. the difference between the local time and Coordinated Universal Time, immediately followed by a sign, + or -, followed by the difference from UTC represented as hh:mm (note: the minutes part is required). See appendix D of **[XSCHEMA2**] for details about legal values in the various fields. If the time zone is included, both hours and minutes must be present.

For example, to indicate 1:20 pm on May the 31st, 1999 for Eastern Standard Time which is 5 hours behind Coordinated Universal Time (UTC), one would write:

```
1999-05-31T13:20:00-05:00.
```


Appendix V: 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.

V.1 UUID Computation

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

V.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 VI: 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/

[DC-NMF]

"Dublin Core Normalized Metadata Format Profile Specification 1.0"; OSTA, 2002.
Available at http://www.osta.org/mpv/

[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

[DPOF]

"DPOF (Digital Print Order Format) Specification Version 1.1", July 17, 2000, Canon Inc, Eastman Kodak Company, Fuji Photo Film Co., Ltd., Matsushita Electric Industrial Co., Ltd.

[Exif2002]

"Exchangeable image file format for digital still cameras: Exif Version 2.2", JEITA CP-3451, Japan Electronics and Information Technology Industries Association (JEITA), February 19, 2002.

[EXPAT]

"expat, a C library for parsing XML", James Clark, multiple versions.
Available at http://expat.sourceforge.net/.

[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. "

[J2K-Part1]

"JPEG 2000 Part 1 FDIS (includes COR 1, COR 2, and DCOR3)", ISO/IEC JTC1/SC29 WG1, JPEG 2000 Editor Martin Boliek, Co-editors Charilaos Christopoulos and Eric Majani, December 4, 2001.

[J2K-Part2]

"JPEG 2000 Part 2 FDIS (includes DCOR 1)", ISO/IEC JTC1/SC29 WG1, JPEG 2000 Editor Martin Boliek, Co-editors Eric Majani, J. Scott Houchin, James Kasner, and Mathias Larsson Carlander, December 4, 2001.

[JFIF]

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

[MANIFEST]

"XML Manifest Specification 1.0"; OSTA, 2002.
Available at http://www.osta.org/mpv/

[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.iana.org/assignments/media-types/index.html

[MPV-Basic]

"MultiPhoto/Video Basic Profile Specification 1.0"; OSTA, 2002.
Available at http://www.osta.org/mpv/

[MPV-Core]

"MultiPhoto/Video Core Specification 1.0"; OSTA, 2002.
Available at http://www.osta.org/mpv/

[MPV-Pres]

"MultiPhoto/Video Presentation Profile Specification 1.0"; OSTA, 2002.
Available at http://www.osta.org/mpv/

[NMF]

"Normalized Metadata Format Specification 1.0"; OSTA, 2002.
Available at http://www.osta.org/mpv/

[OSTA-Web]

Optical Storage Technology Association website.
Available at http://www.osta.org/

[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].

[URI-FRAG]

"A generic fragment identifier syntax for URI references", J. Borden, S. St. Laurent, February 2002. Internet Draft
.Available at http://www.ietf.org/internet-drafts/draft-borden-frag-00.txt

[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

[XMLBase]

"XML Base", Tim Bray, Dave Hollander, Andrew Layman. W3C Recommendation 27 June 2001,
Available at http://www.w3.org/TR/xmlbase/

[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://partners.adobe.com/asn/developer/xmp/download/docs/MetadataFramework.pdf

[XSCHEMA]

"XML Schema, XML Schema Part 1: Structures". W3C Recommendation 2 May, 2001.
Available at http://www.w3.org/TR/xmlschema-1/

[XSCHEMA2]

"XML Schema, XML Schema Part 2: Datatypes". W3C Recommendation, 2 May, 2001.
Available at http://www.w3.org/TR/xmlschema-2/

[XSL]

"XSL Transformations (XSLT) Version 1.0", W3C Recommendation, 16 November, 1999.
Available at http://www.w3.org/TR/xsl/