-OSTA.org-
Optical Storage
Technology Association

Digital Music, Photo, and Video Collections

XML Manifest Specification

Revision 2.00

14 June 2005

This document is available at http://www.osta.org/mpv/public/specs/XML-Manifest-Spec-2.00.pdf

POINTS OF CONTACT

OSTA	*Technical Content*
David Bunzel OSTA President Tel: +1 (408) 253-3695 Email: dbunzel@osta.org http://www.osta.org *MPV Website* http://www.osta.org/mpv/	Pieter van Zee Editor, MPV Specification MPV Initiative Lead Tel: +1 541-715-8658 Email: Pieter.van.Zee@hp.com Felix Nemirovsky Chairman, MPV Subcommittee Tel: +1 415 643 0944 Email: felixn@pacbell.net

ABSTRACT

The XML Manifest specification defines a manifest along with minimal profile management that is used to contain the XML content.

COPYRIGHT NOTICE

RELEASE HISTORY

Version	*Date*	*Comments*
1.00	23 October 2002	First public release.
1.01	11 March 2003	Schema and document format is unchanged. Changed name from MultiPhoto/Video to MPV and MusicPhotoVideo; updated logo, contact info, copyright. Added Music Profile mention. Updated graphics to use MPV and Music Profile. No other changes.
2.00	14 June 2005	Added support for MPV manifest identification and manifest writer identification, and handling manifest updates by different writers. Added support for product category specifications that refine profiles.

IMPORTANT NOTICES
(Revised 01-DEC-2004)

(a) THIS DOCUMENT IS AN AUTHORIZED AND APPROVED PUBLICATION OF THE OPTICAL STORAGE TECHNOLOGY ASSOCIATION (OSTA). THE SPECIFICATIONS CONTAINED HEREIN 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.

(b) COMPLIANCE WITH THIS DOCUMENT MAY REQUIRE USE OF ONE OR MORE FEATURES COVERED BY THE PATENT RIGHTS OF AN OSTA MEMBER, ASSOCIATE OR THIRD PARTY. NO POSITION IS TAKEN BY OSTA WITH RESPECT TO THE VALIDITY OR INFRINGEMENT OF ANY PATENT, COPYRIGHT 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 AND SPECIFICATIONS 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) OSTA MAKES NO REPRESENTATION OR WARRANTY REGARDING ANY SPECIFICATION, AND ANY COMPANY USING A SPECIFICATION SHALL DO SO AT ITS SOLE RISK, INCLUDING SPECIFICALLY THE RISKS THAT A PRODUCT DEVELOPED WILL NOT BE COMPATIBLE WITH ANY OTHER PRODUCT OR THAT ANY PARTICULAR PERFORMANCE WILL NOT BE ACHIEVED. OSTA SHALL NOT BE LIABLE FOR ANY EXEMPLARY, INCIDENTAL, PROXIMATE OR CONSEQUENTIAL DAMAGES OR EXPENSES ARISING FROM THE USE OR IMPLEMENTATION OF THIS DOCUMENT. THIS DOCUMENT DEFINES ONLY ONE APPROACH TO COMPATIBILITY, AND OTHER APPROACHES MAY BE AVAILABLE IN THE INDUSTRY.

(d) THIS DOCUMENT IS A SPECIFICATION ADOPTED BY OSTA. THIS DOCUMENT MAY BE REVISED BY OSTA AT ANY TIME AND WITHOUT NOTICE AND USERS ARE ADVISED TO OBTAIN THE LATEST VERSION. IT IS INTENDED SOLELY AS A GUIDE FOR ORGANIZATIONS INTERESTED IN DEVELOPING PRODUCTS WHICH CAN BE COMPATIBLE WITH OTHER PRODUCTS DEVELOPED USING THIS DOCUMENT. THIS DOCUMENT AND THE SPECIFICATIONS ARE PROVIDED “AS IS”.

(e) MPV, MusicPhotoVideo AND THE MPV LOGO ARE TRADEMARKS OF OSTA. ALL OTHER TRADEMARKS ARE THE PROPERTY OF THEIR RESPECTIVE OWNERS. THE TRADEMARKS MPV, Music/Photo/Video AND THE MPV LOGO MAY NOT BE USED EXCEPT FOR JOURNALISTIC PURPOSES WITHOUT AN EXPLICIT LICENSE FROM OSTA. INFORMATION REGARDING THE MPV TRADEMARK LICENSE IS AVAILABLE AT www.osta.org/mpv/licensing. OBTAINING A LICENSE TO THE OSTA TRADEMARKS MPV, MusicPhotoVideo, AND/OR THE MPV LOGO FROM OSTA DOES NOT PROVIDE A LICENSE TO ANY PATENT REFERENCED HEREIN.

(f) USE OF OSTA’S MPV TRADEMARKS OR ANY STATEMENT TO THE EFFECT THAT A PRODUCT IS “COMPATIBLE” WITH THE MPV SPECIFICATIONS REQUIRES THE LICENSE DESCRIBED ABOVE. USE OF THE MPV TRADEMARKS WITHOUT THE LICENSE IS PROHIBITED, EXCEPT THAT ATTRIBUTION FOR USE OF THE MPV SPECIFICATIONS MAY BE STATED AS FOLLOWS:

> “BASED ON THE MPV™ (MusicPhotoVideo™) SPECIFICATIONS DEVELOPED BY THE OPTICAL STORAGE TECHNOLOGY ASSOCIATION (OSTA). MPV IS A TRADEMARK OF OSTA.”

(g) THE MPV SPECIFICATIONS MAY BE USED ONLY FOR PRODUCTS USED LAWFULLY. OSTA DOES NOT PROMOTE OR CONDONE THE USE OF THE SPECIFICATIONS IN ANY PRODUCT WHICH IS USED FOR AN UNLAWFUL PURPOSE.

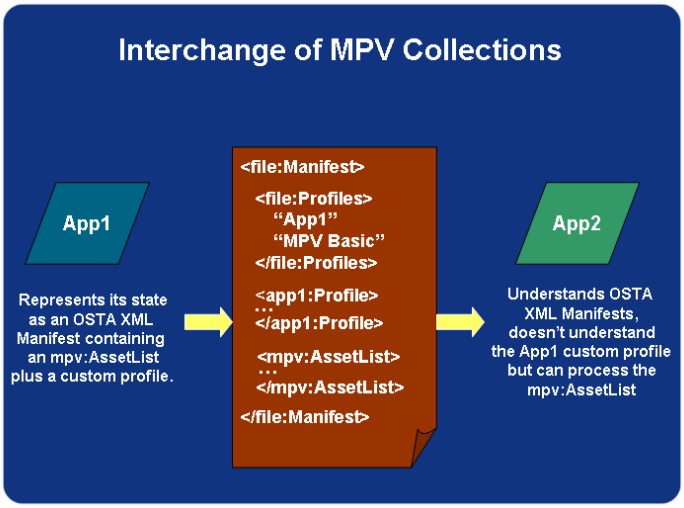
Contents

Chapter 1: Introduction

1.1 Executive Summary

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.

This specification defines the minimal mechanisms and policies that are necessary in order to allow a wide range of metadata to be encapsulated in a single document. Its initial purpose is to support the interchange of MPV (MusicPhotoVideo) metadata but it can also be used to transport metadata that does not directly conform to the mechanisms and policies defined by the various specifications that make up the MPV initiative.



The MPV (MusicPhotoVideo) Initiative is defining a family of open specifications 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.

The development and promotion of MusicPhotoVideo 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.

MPV is not only a family of specifications. 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 Manifest content may be contributed by interested parties.

1.2 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 MusicPhotoVideo specification document for a precise and legal description.

The MusicPhotoVideo Initiative, which includes this specification, is developed using an open process. The resulting specifications are available from OSTA. No royalty is charged by OSTA for use of the specifications. 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.

MusicPhotoVideo is not only a family of specifications. 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 Manifest content may be contributed by interested parties.

1.3 Requirements

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, if and where they appear in this document, are to be interpreted as described in [RFC2119].

Chapter 2: Key Concepts of the XML Manifest and Related Specifications

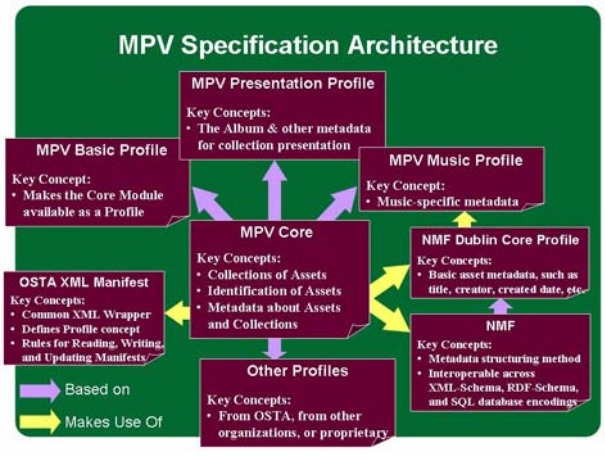
The XML Manifest is a stand-alone specification that is being developed as part of the MusicPhotoVideo (MPV) initiative. It makes use of the Normalized Metadata Format (NMF) which is another stand-alone specification which is being developed as part of the MPV initiative. It is possible to use the XML Manifest without using any other parts of the MPV initiative (other than NMF). While this is possible, it is important to note that the initial focus of the definition process for the XML Manifest is to provide a carrier for MPV based content.

Understanding the mechanisms and policies of the XML Manifest is done in the context of the MPV initiatives various components which are briefly discussed in the following sections. Complete coverage of these components can be found in the MPV specifications which are accessable from the OSTA MPV web site [OSTA-WEB]

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.
- The **MPV Music Profile** defines music-specific metadata, making use of the MPV Presentation Profile and Dublin Core.
- MPV makes use of the **OSTA XML Manifest**, which defines the Profile concept and defines rules for reading, writing, and updating Manifests.
- 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, Product Categories, Schema and Practices

The MPV specifications utilize the following core concepts and content:

Schema define the structure of the content, providing 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 MusicPhotoVideo defines only those schema and practices that are necessary for the key tasks targeted by the profile.

Categories are a set of Practices to be used in implementation of a specific product category, such as DVD players, digital cameras, home printers, etc. Category practices are defined in a specification and refine the use of the profile according to the specific needs of the product category. The usage defined by a category specification all falls within that required by all Profiles which are specified.

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

2.3 NMF-structured Metadata

MPV makes use of a format called Normalized Metadata Format. 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], 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.

2.4 More about Profiles

Profiles are the most important unit of modular extension in the interchange standards that make use of the XML Manifest. Any number of profiles can co-exist within an XML Manifest.

The minimum requirement that the XML Manifest makes of profiles is that they provide a single namespace identifier for the profile which corresponds to the namespace of the top-level schema of the profile.This namespace must then be declared via the mechanism described in the next chapter.

Profiles can make use of other profiles in which case, these profiles are considered to be embedded in the using profile. Embedded profiles do not need to be declared, except if interoperability is desired with consumers of embedded profiles and can be meaningfully processed independent of the top-level profile. Top level profiles MUST be declared.

The XML manifest metadata schema provides a property for specifying the profiles that are being transported in the Manifest. This list of profiles MUST be produced and SHOULD be processed by every Manifest-aware application. Applications processing a manifest that encounter valid content not expected by the Profiles they are processing should leave it untouched and carry it forward.

There are no rules in the XML Manifest regarding the design of Profile schema, but consistency with existing design practices is recommended. It is important to recognize the purpose of the XML Manifest wrapper element. A wrapper element is required of all XML documents. The XML Manifest defines a wrapper element that can be conveniently recognized by Manifest-aware applications.

2.5 XML Usage

XML LEVERAGE

Manifest content is well-formed XML. This allows the XML Manifest 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 XML Manifest 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 XML Manifest 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: Overall Required and Best Practices

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

3.1 Reading an XML Manifest

An XML Manifest may be read (processed) in any manner that complies with XML processing conventions and is consistent with the XML specification and the XML Schema specifications. XML processing instructions shall be permitted; if the processor cannot honor the processing instructions, they may be ignored.

Significantly, processors MUST support the DOCTYPE and external parameter entity 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.

3.2 Character Set

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

3.3 Allowable Characters

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

3.4 Writing and Updating an XML Manifest

We must assume an environment in which there are many XML Manifest Readers and Writers, implemented by different companies and organizations and containing a variety of standard and proprietary content.

In this environment, what are the specifications and practices that allow proper reading, writing, and update of XML Manifests from different Writers? This question is addressed by the XML Manifest 2.0 specficiation, which introduces new schema and practices for Manifest Readers and Writers.

Chapter 4: XML Manifest Structure

This chapter defines the mechanisms and policies that encompass the XML Manifest. The XML Manifest makes use of a set of XML constructs that are defined in several schemas. The top-level schema for the XML manifest (which in turn includes the other schemas) is identified by the following information:

Schema group	Namespace Identifier	Schema Location	Conventional Namespace Prefix
XML Manifest	http://ns.osta.org/manifest/1.0/	manifest2/manifest.xsd	file:
XML Manifest2	http://ns.osta.org/manifest/2.0/	manifest2/manifest.xsd (which will load the 2.0 schema too)	file2:

Note that the preferred namespace prefix for the overall manifest schema is "file". This prefix is not required but is used throughout the documentation and in many examples. The preferred namespace prefix for the XML Manifest 2.0 schema is file2. Note that the schema file location has been updated to manifest2/manifest.xsd for both the 1.0 and 2.0 namespaces. This is an important update to apply to schema-based validation processes.

The XML manifest groups all the MPV and other top-level components into a single XML document. It is defined to allow any well-formed XML content inside of it. In practice, specific profiles will define implied content models that describe what elements should occur as top-level elements in the manifest.

In typical usage, a XML manifest is stored in a stand-alone file. Any application 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 implication of terminology, an XML manifest contains reference to all the content that is relevant – it makes manifest the content.

4.1 XML Manifest 1.0 and 2.0 Compatibility and Coexistence

All schema elements defined by XML Manifest 1.0 continue to utilize that namespace! Only new elements defined by XML Manifest 2.0 utilize the 2.0 namespace.

This is possible because the XML Manifest 2.0 schema and specification is additive relative to the XML Manifest 1.0 schema – it does not replace or obsolete the XML Manifest 1.0 schema and specification. By clever definition of the XML Manifest 2.0 schema, both the Manifest 1.0 and Manifest 2.0 schema may in effect be utilized and co-

exist in the same manifest. This approach is used to avoid incompatibility with XML Manifest 1.0 readers, thus giving enhanced functionality to XML Manifest 2.0 readers and writers while preserving backwards compatibility.

In order to achieve this behaviour, two assumptions are made:

1. The XML Manifest 1.0 Reader will gracefully ignore unknown elements that it encounters. This is consistent with the best practices of the [MPV-CORE] specification.
2. When schema-based validation is performed, the validator will only access the XML Manifest 2.0 schema definition file(s), ie. manifest2/manifest.xsd.

These two assumptions are reasonable, but non-conforming processors will not exhibit graceful behaviour. If assumption 1 is not met by the XML Manifest 1.0 Reader, use of XML Manifest 2.0 elements will cause the 1.0 Reader to fail. If assumption 2 is not met, the validator will report an error for all XML Manifest 2.0 elements.

4.2 Root Element: <file:Manifest>

The top-level element of an XML manifest MUST have a namespace of http://ns.osta.org/manifest/1.0/ and a localname of "Manifest". This element SHOULD be the root element of the XML document.

This element is the outer element of a Manifest document. It wraps any top-elements that are defined by profiles that are being transported in the manifest.

The file:Manifest element uses an open content model which means that it can contain any element irrespective of the namespace or localname of the element. This is done in order to allow a wide range of content to be transported as top-level child elements of the Manifest and also in order to support partial validation.

Partial validation refers to the ability of an XML Manifest processor to apply either strict or lax validation to elements that occur as children of the Manifest element. These two levels of validation are specified using the processContents attribute of the xs:any element. They are:

strict

the XML processor must obtain the schema for the required namespaces and validate any element from those namespaces.

lax

The XML processor attempts to obtain the schema for the required namespaces and validate any element from those namespaces; however, if the schema cannot be obtained, no errors will occur.

NMF provides two versions of a utility schema that defines group elements for all the open content models that are used in NMF schema. This includes the open content model used by the Manifest element.

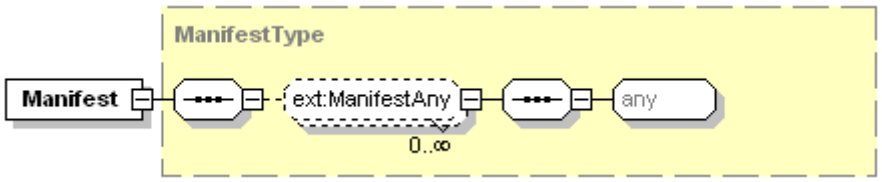
One version uses lax validation and is intended for runtime and production environments. The other uses strict validation and is intended for development environments.

4.2.1 element Manifest, type ManifestType, group ManifestAny

Manifest is the top-level element of an XML Manifest document. The contents of the manifest are defined by the profiles used by the manifest.

Note: The ManifestAny helper group element is from the production environment version of the utility schema and therefore have the value of lax for their processContents attributes.

diagram



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

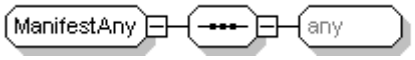
type **ManifestType**

source
```
<xs:element name="Manifest" type="ManifestType"/>
```

source
```
<xs:complexType name="ManifestType">
  <xs:sequence>
    <xs:group ref="ext:ManifestAny" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

group **ManifestAny**

diagram



namespace http://ns.osta.org/manifest/1.0/ext/

used by complexType **ManifestType**

source
```
<xs:group name="ManifestAny">
  <xs:sequence>
    <xs:any processContents="lax"/>
  </xs:sequence>
</xs:group>
```

4.3 XML Manifest Metadata

The XML Manifest defines a single NMF based properties schema. An XML Manifest MUST contain a single instance of this properties schema. This properties schema MUST be contained in an nmf:Metadata element which in turn MUST occur as the first child of the file:Manifest element[1].

4.3.1 Properties: ManifestProperties

The ManifestProperties element can contain several properties. The required properties are the Profile property and WrittenBy property. Optional properties are the AboutManifestMPVDocumentID, Category, and Redirect properties.

[1] This is not enforced directly by the XML schema of the Manifest but is still the required behavior. The compliance test suite for the XML Manifest will contain software that will check for compliance with this requirement.

ELEMENT MANIFESTPROPERTIES, COMPLEXTYPE BYSCHEMAPROPSTYPE

diagram	
namespace	http://ns.osta.org/manifest/1.0/
type	**BySchemaPropsType**
children	**Profile ProfileBag Redirect**
source	<xs:element name="ManifestProperties" type="BySchemaPropsType" substitutionGroup="nmf:BySchemaPropsBase"/>
source	<xs:complexType name="BySchemaPropsType"> <xs:complexContent> <xs:extension base="nmf:BySchemaPropsType"> <xs:sequence> <xs:element ref="file2:AboutManifestMPVDocumentID" minOccurs="0"/> <xs:group ref="file2:CategoryChoiceGroup" minOccurs="0"/> <xs:group ref="ProfileChoiceGroup" minOccurs="0"/> <xs:element ref="Redirect" minOccurs="0"/> <xs:element ref="file2:WrittenBy" minOccurs="0"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

4.3.2 Property: AboutManifestMPVDocumentID

The AboutManifestMPVDocumentID property is an optional property that is used to specify an ID of a mpv:Document asset in the mpv:AssetList in which information about the current manifest can be found. The AboutManifestMPVDocumentID property SHOULD be honored by an Manifest-aware processor. It instructs the processor to consider the metadata specified by the mpv:Document asset (such as manifest locale) when processing the manifest.

4.3.2.1 element AboutManifestMPVDocumentID, complexType AboutManifestMPVDocumentIDType

diagram	
namespace	http://ns.osta.org/manifest/2.0/
type	**AboutManifestMPVDocumentIDType**
used by	complexType **BySchemaPropsType**
source	<xs:element name="AboutManifestMPVDocumentID" type="AboutManifestMPVDocumentIDType"/>
source	<xs:complexType name="AboutManifestMPVDocumentIDType"> <xs:simpleContent> <xs:extension base="xs:ID"/> </xs:simpleContent> </xs:complexType>

4.3.3 Property: Category

The Category property is used to indicate that there is content in the Manifest that conforms to a category specification that is identified by the URI which is the value of the property.

There are two variations of the Category property. The base property, with a localname of “Category”, is used to indicate the value of a single category. If more than one category is present in the manifest, the unordered array wrapper version of the base property should be used which has a localname of “CategoryBag”.

practice

an application that adds content to an XML Manifest that conforms to a category specification SHOULD make sure that there is a category property whose value is equal to the category namespace.

4.3.3.1 group CategoryChoiceGroup

The CategoryChoiceGroup provides the alternate versions of the Category property, Category for the single profile case, and CategoryBag for the multiple profile case.

diagram	
namespace	http://ns.osta.org/manifest/2.0/
children	**Category** **CategoryBag**
used by	complexType **BySchemaPropsType**
source	<xs:group name="CategoryChoiceGroup"> <xs:choice> <xs:element ref="Category"/> <xs:element ref="CategoryBag"/> </xs:choice> </xs:group>

4.3.3.2 element Category, complexType CategoryType

The Category element is used to specify the value of a single profile. It is an instance of CategoryType which is derived from xs:anyURI. The category value must be a valid URI.

diagram	
namespace	http://ns.osta.org/manifest/2.0/
type	**CategoryType**
used by	complexType **CategoryBagType** group **CategoryChoiceGroup**
source	<xs:element name="Category" type="CategoryType"/>
source	<xs:complexType name="CategoryType"> <xs:simpleContent> <xs:extension base="xs:anyURI"/> </xs:simpleContent> </xs:complexType>

4.3.3.3 element CategoryBag, complexType CategoryBagType

diagram	
namespace	http://ns.osta.org/manifest/2.0/
type	**CategoryBagType**
children	**Category**
used by	group **CategoryChoiceGroup**
source	<xs:element name="CategoryBag" type="CategoryBagType"/>
source	<xs:complexType name="CategoryBagType"> <xs:complexContent> <xs:extension base="nmf:BagPropType"> <xs:sequence> <xs:choice minOccurs="0" maxOccurs="unbounded"> <xs:element ref="Category"/> </xs:choice> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

4.3.4 Property: Profile

The Profile property is used to indicate that there is content in the Manifest that conforms to a profile that is identified by the URI which is the value of the property.

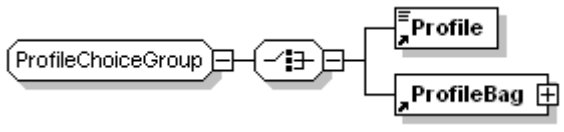
There are two variations of the Profile property. The base property, with a localname of “Profile”, is used to indicate the value of a single profile. If more than one profile is present in the manifest, the unordered array wrapper version of the base property should be used which has a localname of “PropertyBag”.

practice

an application that adds content to an XML Manifest that conforms to a profile SHOULD make sure that there is a profile property whose value is equal to the profile namespace.

4.3.4.1 group ProfileChoiceGroup

The ProfileChoiceGroup provides the alternate versions of the Profile property, Profile for the single profile case, and ProfileBag for the multiple profile case.

diagram	
namespace	http://ns.osta.org/manifest/1.0/
children	**Profile** **ProfileBag**
used by	complexType **BySchemaPropsType**
source	<xs:group name="ProfileChoiceGroup"> <xs:choice> <xs:element ref="Profile"/>

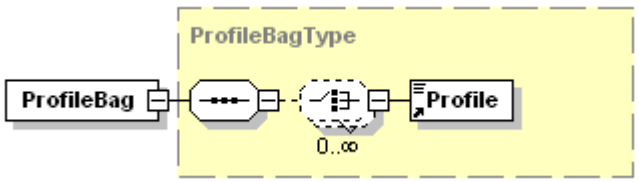
	<xs:element ref="ProfileBag"/> </xs:choice> </xs:group>

4.3.4.2 element Profile, complexType ProfileType

The Profile element is used to specify the value of a single profile. It is an instance of ProfileType which is derived from xs:anyURI. The profile value must be a valid URI.

diagram	
namespace	http://ns.osta.org/manifest/1.0/
type	**ProfileType**
used by	complexType **ProfileBagType** group **ProfileChoiceGroup**
source	<xs:element name="Profile" type="ProfileType"/>
source	<xs:complexType name="ProfileType"> <xs:simpleContent> <xs:extension base="xs:anyURI"/> </xs:simpleContent> </xs:complexType>

4.3.4.3 element ProfileBag, complexType ProfileBagType

diagram	
namespace	http://ns.osta.org/manifest/1.0/
type	**ProfileBagType**
children	**Profile**
used by	group **ProfileChoiceGroup**
source	<xs:element name="ProfileBag" type="ProfileBagType"/>
source	<xs:complexType name="ProfileBagType"> <xs:complexContent> <xs:extension base="nmf:BagPropType"> <xs:sequence> <xs:choice minOccurs="0" maxOccurs="unbounded"> <xs:element ref="Profile"/> </xs:choice> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

4.3.5 Property: Redirect

The Redirect property is an optional property that is used to specify a URL at which the actual manifest can be found. The Redirect property SHOULD be honored by an Manifest-aware processor. It instructs the processor to redirect its processing operations to the referenced document

4.3.5.1 element Redirect, complexType RedirectType

diagram	
namespace	http://ns.osta.org/manifest/1.0/
type	**RedirectType**
used by	complexType **BySchemaPropsType**
source	<xs:element name="Redirect" type="RedirectType"/>
source	<xs:complexType name="RedirectType"> <xs:simpleContent> <xs:extension base="xs:anyURI"/> </xs:simpleContent> </xs:complexType>

4.3.6 Property: WrittenBy

The WrittenBy property is an optional property that is used to specify a URI that uniquely identifies the device or software application that write this Manifest. The WrittenBy property can be used in two ways: by Manifest Readers and Manifest Writers.

A Manifest Reader MAY test the WrittenBy element to determine easily whether the Manifest was written by a known Writer or an unknown Writer. The Reader may choose to process Manifests differently depending on this result. For example, this entry could be used to select one of many available Readers to do further processing. This would allow that extensions implemented by a given application or device could be honored to be best degree possible.

A XML Manifest 2.0 Writer MUST set the WrittenBy element value to a URI that uniquely identifies the device or software application that is writing the Manifest.

A Manifest Writer SHOULD honor the WrittenBy element when an existing Manifest is going to be updated. When the WrittenBy element value is not the same as the identifier of the current Manifest Writer, the assumption is that the Manifest Writer will NOT be able to represent with full fidelity the Manifest's current contents. In other words, whatever is written to the Updated Manifest will by lossy with respect to the existing Manifest. This is due to the assumption that a given Manifest Reader will imperfectly capture the contents of any Manifest in the general case, as it is presumed that all extensions and elements not relevant to the current Reader will be discarded.

When an existing Manifest is updated and the WrittenBy element value is not the same as the identifier of the current Manifest Writer, then steps SHOULD be taken to preserve the existing Manifest and link to it from the updated Manifest. The details of this process are described in the Practices section 5.5 Updating an Existing XML Manifest.

4.3.6.1 element WrittenBy, complexType WrittenByType

diagram	
namespace	http://ns.osta.org/manifest/2.0/

type	**WrittenBy**
used by	complexType **BySchemaPropsType**
source	<xs:element name="WrittenBy" type="WrittenByType"/>
source	<xs:complexType name="WrittenByType"> <xs:simpleContent> <xs:extension base="xs:anyURI"/> </xs:simpleContent> </xs:complexType>

4.4 Example

The following example demonstrates use of each of the XML Manifest 2.0 schema elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<file:Manifest
        xmlns:file="http://ns.osta.org/manifest/1.0/"
        xmlns:file2="http://ns.osta.org/manifest/2.0/"
        xmlns:mpv="http://ns.osta.org/mpv/1.0/"
        xmlns:nmf="http://ns.osta.org/nmf/1.0/"
        xmlns:Profile1_1="http://www.companyA.com/Profile1/1.0/"
  <nmf:Metadata>
    <file:ManifestProperties>
      <file2:AboutManifestMPVDocumentID>IDDoc01/</file2:AboutManifestMPVDocumentID>
      <file:ProfileBag>
          <file:Profile>http://ns.osta.org/mpv/basic/1.0/</file:Profile>
          <file:Profile>http://www.companyA.com/Profile1/1.0/</file:Profile>
      </file:ProfileBag >
      <file2:WrittenBy>http://www.companyA.com/Device/Model934/</file2:WrittenBy>
    </file:ManifestProperties>
  </nmf:Metadata>

  …

  <mpv:AssetList>
     …
     <mpv:Document mpv:id="IDDoc01">
      …
     </mpv:Document>
  </mpv:AssetList>

</file:Manifest>
```


Chapter 5: XML Manifest Practices

An XML Manifest can contain top-level elements from zero or more profiles. A profile is identified by a namespace URI. The top-level elements used by the profile do not need to use the same namespace as the one used to identify the profile.

5.1 Finding an XML Manifest File

Specifications that make use of the XML Manifest as their interchange container SHOULD define specific practices for the placement and location of the XML Manifest. An example of the definition of such practices is the provided in the MPV Core Specification.

5.2 Top-level Elements

A profile MAY define its top-level elements (elements that are children of the Manifest root element) using the same namespace as the profile. This allows processors to reliably ignore top-level children from namespaces that aren't recognized based on namespace matching.

5.3 Extending Profiles

There are two mechanisms that are available to allow processors to recognize whether they can accept a particular XML Manifest document. One mechanism is based on the namespaces that are used by the various elements and attributes that are contained in the document. The other is based on the Profile URI that are specified in the Profile properties contained in the nmf:Metadata (see section 4.3.3).

When a profile is extended, use of namespaces as a versioning mechanism SHOULD only be used for incompatible changes to the syntax or semantics of the profile. If all of the extensions are backwards compatible with the existing profile, then the namespaces used for the elements and attributes SHOULD be left alone in order to not break any existing processors.

This then leaves the question of how backwards compatible changes should be communicated to the consumers of a profile contained in an XML Manifest. The alternatives are either to replace the existing profile announcer or add an additional profile announcer. Changing the existing profile would break any processors that key off the profile

announcer in order to determine if they can process the document. This is why backwards compatible extensions to profiles SHOULD be announced via one or more additional Profile properties. These profile announcers MAY use a common syntactic convention in their profile URI (the value of the Profile property).

In the example below, a Profile identified by the URI value of http://www.companyA.com/Profile1/1.0/ has now been extended by a 2.0 version that is backwards compatible. This new version is identified by the URI value of http://www.companyA.com/Profile1/2.0/ . This extension defines some new elements that are in a new namespace with the URI value of http://www.companyA/Profile1/2.0/newElems/. This shows that the profile URI don't have to be used for any element or attribute naming but simply as an announcer for the profile.

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_1="http://www.companyA.com/Profile1/1.0/"
        xmlns:newElems="http://www.companyA.com/Profile1/2.0/newElems/">
  <nmf:Metadata>
    <ManifestProperties xmlns="http://ns.osta.org/manifest/1.0/">
      <ProfileBag>
          <Profile>http://ns.osta.org/mpv/basic/1.0/</Profile>
          <Profile>http://www.companyA.com/Profile1/1.0/</Profile>
          <Profile>http://www.companyA.com/Profile1/2.0/</Profile>
      </ProfileBag >
    </ManifestProperties>
  </nmf:Metadata>

  <Profile1_1:Outer1>
     …
     <newElems:FooRef />
  </Profile1_1:Outer1>

  …

  <mpv:AssetList>
     …
     <newElems:Foo>
      …
     </newElems:Foo>
  </mpv:AssetList>

</file:Manifest>
```

5.4 Writing an XML Manifest

The element

file:Manifest/nmf:Metadata/file:ManifestProperties/file2:WrittenBy

contains a URI that uniquely identifies the device or application that wrote the existing Manifest. Whenever a Manifest is written, the XML Manifest 2.0-compliant Writer MUST specify the WrittenBy element.

If the new Manifest is an Update of an Existing Manifest, then the new Manifest SHOULD follow the practices in the next section, 5.5 Updating an Existing XML Manifest.

5.5 Updating an Existing XML Manifest

5.5.1 Case: Writer Updating a Manifest It Wrote

When an existing Manifest is updated and the WrittenBy element value is exactly the same as the identifier of the current Manifest Writer, then it is presumed that all the data in the existing Manifest has been preserved by the Reader and will be fully retained where appropriate in the updated Manifest. If this is case, the existing Manifest may be rewritten without further steps or precautions.

5.5.2 Case: Writer Updating a Manifest It Did NOT Write

If this is not true, or when the WrittenBy element of the existing Manifest is not the same as the identifier of the current Manifest Writer, then four steps SHOULD be taken:

1. the existing manifest in preserved in its pristine original state
2. the existing manifest is renamed
3. an updated Manifest is written to take the place of the existing Manifest
4. the updated Manifest SHOULD contain a reference back to the existing manifest, with its new name.

STEP 1: PRESERVE THE EXISTING MANIFEST IN PRISTINE STATE

The existing manifest is presumed to have metadata that cannot be fully rewritten by the current Writer. At the same time, the exact filename of the existing Manifest must be used for the Updated Manifest to preserve compatibility with any links to that manifest from other manifests and the existing behaviour of the Manifest search algorithm employed.

A Manifest Writer SHOULD NOT attempt to preserve content it does not understand in an existing Manifest when it writes an updated Manifest, unless the Writer is confident the data can be faithfully preserved. In most cases, however, this is not possible.

To avoid the unnecesary loss of metadata, the existing manifest SHOULD be preserved in its Pristine state. This is achieved by renaming the file, including optionally moving the file to another location.

STEP 2: BY RENAMING THE EXISTING MANIFEST

Many manifest renaming practices are possible, and right one to use should be considered in a broader context of the usage of the XML Manifest and its contents.

When a specific manifest renaming practice is not otherwise specified, the following Existing Manifest Renaming Practice SHOULD be utilized.

1. Create a subdirectory named METAHIST in the existing manifest location. METAHIST = Metadata History.
2. Within the METAHIST directory, preserved manifests are renamed in sequential order using the naming pattern HIST<dddd>.<ext>, where dddd are digits starting at 0000 and proceeding to 9999 by decimal counting.
3. A new number should always be previously unused. Where gaps in numbering exist because a manifest has been deleted, the gap number SHOULD NOT be reused.

4. In the rare case that 9999 rewrites have occured, the pattern may be adapted to be <ddddddddd>.<ext>.

STEP 3: WRITE THE UPDATED MANIFEST

The updated Manifest is written to the filename of the existing Manifest. The Updated Manifest must have the element

file:Manifest/nmf:Metadata/file:ManifestProperties/file2:WrittenBy

containing a URI that uniquely identifies the device or application that wrote the existing Manifest.

STEP 4: LINK BACK TO EXISTING MANIFEST IN UPDATED MANIFEST

The Updated Manifest SHOULD also contain a link back to the existing Manifest. This is achieved by following the specification in section 6.2.5 Chaining to a "DerivedFrom" Manifest.

5.5.3 Example

The following example demonstrates the state before and after an update.

EXISTING MANIFEST BEFORE UPDATE

Before the update, the manifest exists by itself in the file system.

Location: /MISC/ALBUM.PVM
Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<file:Manifest
        xmlns:file="http://ns.osta.org/manifest/1.0/"
        xmlns:file2="http://ns.osta.org/manifest/2.0/"
        xmlns:mpv="http://ns.osta.org/mpv/1.0/"
        xmlns:nmf="http://ns.osta.org/nmf/1.0/"
  <nmf:Metadata>
    <file:ManifestProperties>
      <file2:AboutManifestMPVDocumentID>IDDoc01/</file2:AboutManifestMPVDocumentID>
      <file:ProfileBag>
          <file:Profile>http://ns.osta.org/mpv/basic/1.0/</file:Profile>
      </file:ProfileBag >
      <file2:WrittenBy>http://www.companyA.com/Device/Model934/</file2:WrittenBy>
    </file:ManifestProperties>
  </nmf:Metadata>

  <mpv:AssetList>
     …
     <mpv:Document mpv:id="IDDoc01">
        <mpv:InstanceID>urn:osta-
 org:mpv:uuid:AC937BCFA3B340da971BAF09B17DBC324</mpv:InstanceID>
        <mpv:LastURL>ALBUM.PVM</mpv:LastURL>
        <mpv:LastURL>/MISC/ALBUM.PVM</mpv:LastURL>
     …
     </mpv:Document>
```



```
  </mpv:AssetList>

</file:Manifest>
```

UPDATED MANIFEST AFTER UPDATE

After the update, the existing manifest has been renamed but left pristine while the updated manifest chains back to it.

Location: /MISC/METAHIST/HIST0001.PVM
Content: same as above

Location: /MISC/ALBUM.PVM
Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<file:Manifest
        xmlns:file="http://ns.osta.org/manifest/1.0/"
        xmlns:file2="http://ns.osta.org/manifest/2.0/"
        xmlns:mpv="http://ns.osta.org/mpv/1.0/"
        xmlns:nmf="http://ns.osta.org/nmf/1.0/"
  <nmf:Metadata>
    <file:ManifestProperties>
      <file2:AboutManifestMPVDocumentID>IDDoc02/</file2:AboutManifestMPVDocumentID>
      <file:ProfileBag>
          <file:Profile>http://ns.osta.org/mpv/basic/1.0/</file:Profile>
      </file:ProfileBag >
      <file2:WrittenBy>http://www.companyB.com/App/NameV2.1.3/</file2:WrittenBy>
    </file:ManifestProperties>
  </nmf:Metadata>

  <mpv:AssetList>
     …
     <!-- This asset describes this one from which the current Manifest was derived -->
     <mpv:Document mpv:id="IDDoc01">
        <mpv:InstanceID>urn:osta-
  org:mpv:uuid:AC937BCFA3B340da971BAF09B17DBC324</mpv:InstanceID>
        <mpv:LastURL>METAHIST/HIST0001.PVM</mpv:LastURL>
        <mpv:LastURL>/MISC/METAHIST/HIST0001.PVM</mpv:LastURL>
     …
     </mpv:Document>
     <!-- This asset describes this very MPV Manifest -->
     <mpv:Document mpv:id="IDDoc02">
        <mpv:InstanceID>urn:osta-
  org:mpv:uuid:AC937BCFA3B340da971BAF09B17DBC324</mpv:InstanceID>
        <mpv:LastURL>ALBUM.PVM</mpv:LastURL>
        <mpv:LastURL>/MISC/ALBUM.PVM</mpv:LastURL>
        <mpv:Related mpv:relationship="derivedFrom">
            <mpv:DocumentRef mpv:idRef="IDDoc01"/>
        </mpv:Related>
     …
     </mpv:Document>
  </mpv:AssetList>

</file:Manifest>
```


Chapter 6: mpv:Document Asset Usage

A primary use of the XML Manifest is to contain MPV metadata, as defined by other specifications such as [MPV-CORE]. In addition to the metadata about music, photo, and video assets, the mpv:AssetList may also contain one or morempv:Document assets.

6.1 Locating the AboutManifestMPVDocumentID mpv:Document

The element

file:Manifest/nmf:Metadata/file:ManifestProperties/file2:AboutManifestMPVDocumentID

contains an ID value will match a mpv :id of a mpv:Document in the mpv:AssetList. That mpv:Document asset contains useful information about the current Manifest. It may also reference other mpv:Document assets, which represent prior versions of the Manifest that have been updated.

6.2 mpv:Document Content

6.2.1 mpv:InstanceID of the Manifest

The mpv:Document asset representing the manifest itself SHOULD have the mpv:InstanceID element.

The mpv:InstanceID is a unique number for every manifest. It can be used to uniquely identify a manifest. In certain contexts, it may also be used to represent a tangible asset. For example, a MPV-aware application could use the mpv:InstanceID in an index.pvm file at the root of a storage file system to identify the storage product, such as a CD or DVD or HDD.

EXAMPLE:

```
<!-- This asset describes this very MPV Manifest -->
<mpv:Document mpv:id="ID000001">
   <mpv:InstanceID>urn.osta-org.mpv.uuid.23452234BDF9BBA934338DFBFFDE8342</mpv:InstanceID>
```



```
    <nmf:Metadata>
       <Properties xmlns="http://purl.org/dc/elements/1.1/"
...
       </Properties>
     </nmf:Metadata>
  </mpv:Document>
```

6.2.2 Dublin Core Properties of the Manifest

The mpv:Document asset representing the manifest itself SHOULD contain the dc properties to specify important information about the manifest's content, including: creator, description, format, identifier, publisher, rights, title, created, modified.

The mpv:Document representing the MPV manifest itself can have any kind of data associated with it. Of principle interest are the Dublin Core properties that are highly interchangeable.

The usage of the Dublin Core properties for the mpv:Document referring to a Manifest are:

DC Elements:
Creator – plain text name of creator application or device
Description – any kind of description, possibly from a user
Format – the MIME type string "application/vnd.osta-org.mpv+xml"
Identifier – plain text of any kind
Language – locale of the manifest
Publisher – plain text publisher info
Rights – plain text describing rights
Title – plain text for title

DC Terms:
Created
Mofied

6.2.3 Locale of the Manifest

The mpv:Document asset representing the manifest itself SHOULD have the dc:language element set to the default locale of the manifest's content. The dc:language value MUST conform to [DC-NMF].

The MPV manifest is authored for a single "locale", where a locale is a combination of language and territory, such as English-U.S. The locale of a manifest is recorded as a dc:language element of the manifest's own mpv:Document asset. The default locale of all text in the manifest is identified by the dc:language element within the mpv:Document element representing the current manifest. A Reader MAY honor the use of dc:language for purposes of sorting, line wraping, currency, and other values.

Example:

```
<file:Manifest
        xmlns:file="http://ns.osta.org/manifest/1.0/"
        xmlns:file2="http://ns.osta.org/manifest/2.0/"
        xmlns:mpv="http://ns.osta.org/mpv/1.0/"
        xmlns:nmf="http://ns.osta.org/nmf/1.0/" >
```



```
...
  <mpv:AssetList>
     <!-- This asset describes this very MPV Manifest -->
     <mpv:Document mpv:id="ID000001">
        <mpv:InstanceID>urn.osta-org.mpv.uuid.2234BDF9BBA934338DFBFFDE8342</mpv:InstanceID>
        <nmf:Metadata>
          <Properties xmlns="http://purl.org/dc/elements/1.1/"
...
             <language>en-US</language>
...
          </Properties>
        </nmf:Metadata>
     </mpv:Document>
  </mpv:AssetList>
</file:Manifest>
```

6.2.4 mpv:LastURL of the Manifest

An optional element that provides a self-reference to the expected name of the current manifest.

Note that when an Existing Manifest has be moved aside into the History area by renaming, the LastURL of the mpv:Document asset in the file will not have been changed. This LastURL value can be used by a sophisticated app to recover the name and location of the Manifest prior to it being Updated.

6.2.5 Chaining to a "DerivedFrom" Manifest

When an Existing Manifest is updated and is preserved by renaming, a link to the Existing Manifest SHOULD be created in the Updated Manifest by the current Writer. This creates a "derivedFrom" relationship between the updated and preserved manifest. This is achieved simply by the following steps:

1. The existing mpv:Document asset that refers to the Existing Manifest is retained
2. The mpv:LastURL element(s) of the existing mpv:Document are updated to the renamed filename.
3. A new mpv:Document asset that represents the Updated Manifest is created, with a new mpv:id
4. In the new mpv:Document asset, a new sub element <mpv:Related mpv:relationship="derivedFrom"> is added. The <mpv:DocumentRef mpv:idRef="xyz"> element gives the mpd:idRef of the mpv:Document representing the Existing Manifest.
5. The new mpv:id value is stored in the AboutManifestMPVDocumentID element.

6.3 Example

This is an example of the mpv:Document representing a well-formed MPV manifest.

```
<?xml version="1.0" encoding="UTF-8"?>
<file:Manifest
        xmlns:file="http://ns.osta.org/manifest/1.0/"
        xmlns:file2="http://ns.osta.org/manifest/2.0/"
        xmlns:mpv="http://ns.osta.org/mpv/1.0/"
        xmlns:nmf="http://ns.osta.org/nmf/1.0/"
  <nmf:Metadata>
    <file:ManifestProperties>
      <file2:AboutManifestMPVDocumentID>IDDoc02/</file2:AboutManifestMPVDocumentID>
```

```
      <file:ProfileBag>
          <file:Profile>http://ns.osta.org/mpv/basic/1.0/</file:Profile>
      </file:ProfileBag >
      <file2:WrittenBy>http://www.companyB.com/App/NameV2.1.3/</file2:WrittenBy>
    </file:ManifestProperties>
  </nmf:Metadata>

  <mpv:AssetList>
     …
     <!-- This asset describes the MPV Manifest from which the current one was derived -->
     <mpv:Document mpv:id="IDDoc01">
        <mpv:InstanceID>urn:osta-
 org:mpv:uuid:AC937BCFA3B340da971BAF09B17DBC324</mpv:InstanceID>
        <mpv:LastURL>METAHIST/HIST0001.PVM</mpv:LastURL>
        <mpv:LastURL>/MISC/METAHIST/HIST0001.PVM</mpv:LastURL>
     …
     </mpv:Document>
     <!-- This asset describes this very MPV Manifest -->
     <mpv:Document mpv:id="IDDoc02">
        <mpv:InstanceID>urn:osta-
 org:mpv:uuid:AC937BCFA3B340da971BAF09B17DBC324</mpv:InstanceID>
        <mpv:LastURL>ALBUM.PVM</mpv:LastURL>
        <mpv:LastURL>/MISC/ALBUM.PVM</mpv:LastURL>
        <nmf:Metadata>
          <Properties xmlns="http://purl.org/dc/elements/1.1/"
             <creator>CompanyB App V2.1.3</creator>
             <description>this is a description</description>
             <language>en-US</language>
             <title>some title</title>
          </Properties>
          <Properties xmlns="http://purl.org/dc/terms/1.0/"
             <createDate>2004-04-23T23:33:01</createDate>
             <modifiedDate>2004-05-01T12:44:10</modifiedDate>
          </Properties>
        </nmf:Metadata>
        <mpv:Related mpv:relationship="derivedFrom">
            <mpv:DocumentRef mpv:idRef="IDDoc01"/>
        </mpv:Related>
     …
     </mpv:Document>
  </mpv:AssetList>

</file:Manifest>
```


Chapter 7: Examples

7.1 Single Profile

This example shows a manifest that contains a contents from a single profile which is identified by the namespace URI of http://www.companyA.com/Profile1/1.0/.

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

    <ManifestProperties xmlns=”http://ns.osta.org/manifest/1.0/”>
      <Profile>http://ns.osta.org/mpv/basic/1.0/</Profile>
    </ManifestProperties>
  </nmf:Metadata>

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

</file:Manifest>
```

7.2 MPV with multiple profiles

This example shows the use of the MPV Basic profile along with several other profiles. Note that while the namespace used to identify the profile (and the top-level schema) is often used for the top-level elements from the profile, this is not necessary.

Top-level profiles can incorporate other profiles as embedded profiles. In these cases, the elements from the embedded profiles will use their own top-level namespace for elements that they may define as direct children of the Manifest.


```
<?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/basic/1.0/</Profile>
          <Profile>http://www.companyA.com/Profile1/1.0/</Profile>
          <Profile>http://www.companyB.com/Profile2/3.5/</Profile>
      <ProfileBag >
    </ManifestProperties>
  </nmf:Metadata>

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

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

  …

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

</file:Manifest>
```

7.3 Updated MPV Manifest With Document Metadata and Derivation Chain

This is an example of the mpv:Document representing a well-formed MPV manifest.

```
<?xml version="1.0" encoding="UTF-8"?>
<file:Manifest
        xmlns:file="http://ns.osta.org/manifest/1.0/"
        xmlns:file2="http://ns.osta.org/manifest/2.0/"
        xmlns:mpv="http://ns.osta.org/mpv/1.0/"
        xmlns:nmf="http://ns.osta.org/nmf/1.0/"
  <nmf:Metadata>
    <file:ManifestProperties>
      <file2:AboutManifestMPVDocumentID>IDDoc02/</file2:AboutManifestMPVDocumentID>
      <file:ProfileBag>
          <file:Profile>http://ns.osta.org/mpv/basic/1.0/</file:Profile>
```



```
      </file:ProfileBag >
      <file2:WrittenBy>http://www.companyB.com/App/NameV2.1.3/</file2:WrittenBy>
    </file:ManifestProperties>
  </nmf:Metadata>

  <mpv:AssetList>
     …
     <!-- This asset describes the MPV Manifest from which the current one was derived -->
     <mpv:Document mpv:id="IDDoc01">
        <mpv:InstanceID>urn:osta-
  org:mpv:uuid:AC937BCFA3B340da971BAF09B17DBC324</mpv:InstanceID>
        <mpv:LastURL>METAHIST/HIST0001.PVM</mpv:LastURL>
        <mpv:LastURL>/MISC/METAHIST/HIST0001.PVM</mpv:LastURL>
     …
     </mpv:Document>
     <!-- This asset describes this very MPV Manifest -->
     <mpv:Document mpv:id="IDDoc02">
        <mpv:InstanceID>urn:osta-
  org:mpv:uuid:AC937BCFA3B340da971BAF09B17DBC324</mpv:InstanceID>
        <mpv:LastURL>ALBUM.PVM</mpv:LastURL>
        <mpv:LastURL>/MISC/ALBUM.PVM</mpv:LastURL>
        <nmf:Metadata>
          <Properties xmlns="http://purl.org/dc/elements/1.1/"
             <creator>CompanyB App V2.1.3</creator>
             <description>this is a description</description>
             <language>en-US</language>
             <title>some title</title>
          </Properties>
          <Properties xmlns="http://purl.org/dc/terms/1.0/"
             <createDate>2004-04-23T23:33:01</createDate>
             <modifiedDate>2004-05-01T12:44:10</modifiedDate>
          </Properties>
        </nmf:Metadata>
        <mpv:Related mpv:relationship="derivedFrom">
            <mpv:DocumentRef mpv:idRef="IDDoc01"/>
        </mpv:Related>
     …
     </mpv:Document>
  </mpv:AssetList>

</file:Manifest>
```

Appendix I: 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 Specification 1.01"; OSTA, 2003.
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.

[RFC2119]

"Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, IETF RFC 2119
http://www.ietf.org/rfc/rfc2119.txt

[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. "

[JFIF]

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

[MD5]

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

[MIME-2]

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

[MIMETYPES-REG]

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

[MPV-CORE]

"MusicPhotoVideo Core 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/

[PNG-MIME]

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

[PNG-REC]

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

[QT]

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

[QT-MIME]

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

[RDFsyntax]

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

[RDFschema]

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

[RFC1766]

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

[SMIL10]

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

[SMIL20]

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

[SMIL-MOD]

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

[URI]

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

[UCS-2]

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

[UTF-8]

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

[W3C-NSURI]

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

[XML10]

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

[XML-NS]

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

[XMP-FW]

"*XMP – Extensible Metadata Platform 14 Sept 01"* , Copyright 2001 Adobe Inc,
Available at http://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/

