-OSTA.org-
Optical Storage
Technology Association

MultiPhoto/Video

Manifest, Metadata and Practices for Digital Photo-Video Collections

MultiPhoto Video
DRAFT
LOGO

Normalized Metadata Format Specification

Revision 0.40
Working Draft

June 7, 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/NMF-Spec-0.40WD.PDF

POINTS OF CONTACT

OSTA 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/	*Technical Content* Gabe Beged-Dov Editor, Normalized Metadata Format Specification Tel: +1 541-715-7347 Email: Gabe_Beged-Dov@hp.com 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

Normalized Metadata Format (NMF) is an open specification which provides an approach to describing metadata in a manner that enables interoperability with several existing metadata representations and processing using mainstream XML tools and technologies.
.

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

Normalized Metadata Format(NMF) is an open specification that describes an XML Schema based representation for metadata. This representation has several primary goals:

- To provide a simple, flexible way to define and interchange metadata using mainstream XML tools and technologies.
- To provide a mechanical schema-less mapping between a subset of Resource Description Format (RDF) based metadata and NMF.
- To provide a straight-forward mechanical mapping to relational databases.

It is **NOT** a goal for NMF to be able to represent all content that can currently be represented in RDF/XML. It is a goal for all NMF content to be representable in RDF/XML.

The specification defines a base layer of functionality, which provides the core mechanisms necessary for definition, and use of metadata that conforms to the NMF model and syntax. A set of profiles are also being defined that make use of the NMF model and syntax. These profiles will be separate specifications that will rely on the NMF specification.

These profiles are composed of both schemas and best practices for use of those schemas. Some of the NMF schemas in these profiles represent a mechanical mapping of existing metadata formats that are specified using RDF and other metadata representations. Others have been specified directly in NMF and do not have explicit specifications outside of NMF although they can be interchanged and processed using RDF toolsets once they have been mechanically converted to RDF.

NMF based modules may be specified via a stand-alone specification or they may be specified as part of a larger specification such as a MultiPhoto/Video Profile. The following NMF based specifications are being developed:
:

- **Dublin Core Profile**: Dublin Core profile. Mirrors the Dublin Core metadata set. The Dublin Core profile is specified in the [NMF-DC].

- **MultiPhoto/Video Core Specification**: A module that specifies NMF equivalents to the identifiers that are defined by the MultiPhoto/Video core specification. This module is specified as part of the MultiPhoto/Video core specification.

- **OSTA Manifest Profile**: A specification that enables an OSTA Manifest to declare profile information.
.
- **MultiPhoto/Video Presentation Profile**: A profile that provides metadata that is useful for describing aspects of multi-media presentations like slideshows.

1.2 Overview

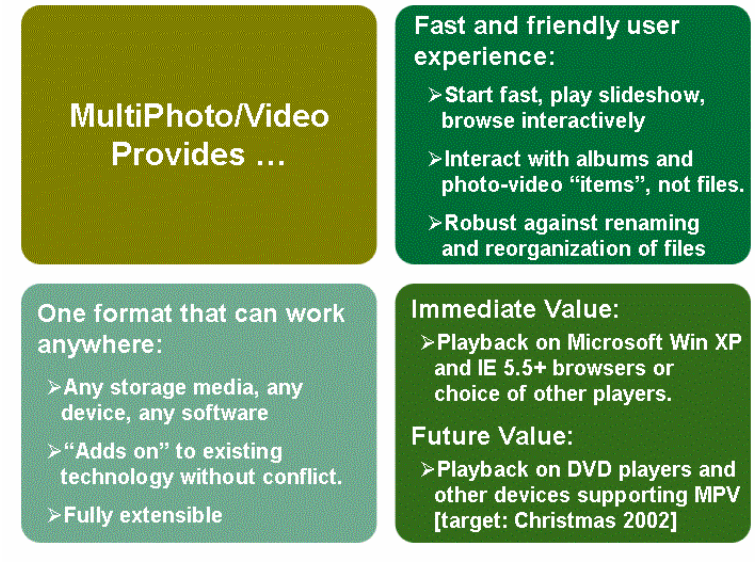
Normalized Metadata Format (NMF) is an open specification which provides an approach to describing metadata in a manner that enables:

- interoperability with several existing metadata representations such as RDF.
- processing using mainstream XML tools and technologies
- interoperation with relational storage systems

The primary driver for the development of NMF was to provide a metadata representation for information about the assets that are described in the MultiPhoto/Video initiative. A short overview of the MultiPhoto/Video initiative is provided below. A complete description is available in the various specifications and white papers that are accessable from the OSTA web site [OSTA-WEB]

1.2.1 What is the MultiPhoto/Video Initiative?

The MultiPhoto/Video initiative has the goal 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.
This is done by defining a basic mechanism for specifying metadata about assets and collections of assets and also defining a rich set of metadata that makes use of this mechanism. The collections and assets are defined by the MultiPhoto/Video (MPV) specification which provides a context for interpreting the metadata that is specified using the NMF encoding.
MPV is made available at low cost and without royalty from the Optical Storage Technology Association (OSTA) and the International Imaging Industry Association (I3A). OSTA is an industry association promoting the use and interoperability of recordable CD and DVD discs in computer and consumer electronics devices. I3A is an industry association promoting digital and film imaging technologies.



MPV enables PC software and consumer electronics devices like DVD players to playback and manipulate collections of digital photo/video content including still images, still with audio, still sequences, video clips, audio-only clips, and related files. The emphasis is on personal content originating from many sources including digital cameras, film, scanners and video digitizer and stored on a range of media including memory cards, recordable or stamped CDs and DVDs, and even computer hard disks or internet services. MPV provides specific manifest and metadata formats and implementation practices that support existing industry specifications such as the World Wide Web Consortium's SMIL, I3A's DIG35, and Adobe's Extensible Metadata Platform XMP. MPV is compatible with

and supports the DCF and Exif specifications from the JEITA and JCIA that are widely used in digital cameras. New metadata elements will be developed as necessary. The work is oriented to deliver tangible and useful results in the near-term.

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 specification document for a precise and legal description.

This 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.

Normalized Metadata Format 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: Concepts

NMF metadata is primarily designed to be used to associate metadata with MultiPhoto/Video (MPV) elements including the MPV Manifest as a whole, Collections, Assets and Marklists. In addition, NMF can be used as a stand-alone representation that can encode a large subset of metadata content that is encoded in RDF/XML.

The metadata that NMF can encode is designed to support a lossless mapping to and from RDF. This allows several significant RDF based applications to be translated to NMF and then processed in the normalized XML Schema environment that NMF enables.

2.1 Round-tripping Metadata

There are two primary representations for Metadata in the NMF approach. One is the NMF representation, which is based on XML Schema, and the other is the RDF representation, which may or may not be based on a schema representation.

NMF is designed to allow both a natural description of metadata in NMF while also allowing as large a subset of RDF metadata to be translated into NMF. This translation between the NMF representation and the RDF representation must be able to occur in both a type aware and type unaware environment. In practice, even the type-aware (schemas available) environment may not have access to the type information for all of the metadata that the mapping is being applied to.

The NMF model and syntax is an attempt to balance the requirement for a natural approach to specifying metadata using an XML schema based definition while also supporting a natural mechanical mapping of RDF metadata to and from NMF.

Many of the constraints and stylized patterns of definition that NMF employs derive from this need to support these requirements in type aware and type unaware processing environments.

2.2 Resource Description Format (RDF)

The Resource Description Format (RDF) is a W3C recommendation that provides a XML syntax and model for metadata interchange. The specification [RDF] describes the RDF model in section 2.1 as follows:

> The foundation of RDF is a model for representing named properties and property values. The RDF model draws on well-established principles from various data representation communities. RDF properties may be thought of as attributes of resources and in this sense correspond to traditional attribute-value pairs. RDF properties also represent relationships between resources and an RDF model

can therefore resemble an entity-relationship diagram. (More precisely, RDF Schemas — which are themselves instances of RDF data models — are ER diagrams.) In object-oriented design terminology, resources correspond to objects and properties correspond to instance variables.

The RDF data model is a syntax-neutral way of representing RDF expressions. The data model representation is used to evaluate equivalence in meaning. Two RDF expressions are equivalent if and only if their data model representations are the same. This definition of equivalence permits some syntactic variation in expression without altering the meaning.

The basic data model consists of three object types:

Resources All things being described by RDF expressions are called resources. A resource may be an entire Web page; such as the HTML document "http://www.w3.org/Overview.html" for example. A resource may be a part of a Web page; e.g. a specific HTML or XML element within the document source. A resource may also be a whole collection of pages; e.g. an entire Web site. A resource may also be an object that is not directly accessible via the Web; e.g. a printed book. Resources are always named by URIs plus optional anchor ids (see [URI]). Anything can have a URI; the extensibility of URIs allows the introduction of identifiers for any entity imaginable.

Properties A property is a specific aspect, characteristic, attribute, or relation used to describe a resource. Each property has a specific meaning, defines its permitted values, the types of resources it can describe, and its relationship with other properties. This document does not address how the characteristics of properties are expressed; for such information, refer to the RDF Schema specification).

Statements A specific resource together with a named property plus the value of that property for that resource is an RDF statement. These three individual parts of a statement are called, respectively, the subject, the predicate, and the object. The object of a statement (i.e., the property value) can be another resource or it can be a literal; i.e., a resource (specified by a URI) or a simple string or other primitive datatype defined by XML. In RDF terms, a literal may have content that is XML markup but is not further evaluated by the RDF processor. There are some syntactic restrictions on how markup in literals may be expressed; see Section 2.2.1.

NMF provides an XML Schema based alternative encoding of a large subset of RDF/XML content and leverages the RDF data model which allows mechanical schemaless interchange of metadata between the NMF based representation and the RDF/XML based representation.

2.3 Terminology

Resource

A Resource is anything that can be identified by a URI

Schema

A Schema is a set of property definitions that is identified by a URI.

Property

A Property is a named entry in a schema. In NMF, this property must have the same namespace as the schema and also have a well-defined value type that is expressible in an XML Schema [XSCHEMA].

Statement

A statement is the binding of a property instance to a particular resource.

Literal Property Value

A literal property value is either textual content or well-formed XML that is not a composite property value.

Composite Property Value

A composite property value is a set of properties from one or more schemas.

Top-level Composite Property Value

A top-level composite property value is a composite property value that is not contained in a property element.

Nested Composite Property Value

A nested composite property value is a composite property value that is contained in a property element.

Array Property Value

An Array property value is sequence of properties that are either ordered, unordered or alternatives.

Qualifed Property Value

A qualified property value is one where the base property value (either literal or composite) has zero or more additional properties assocated with it. These additional properties are called qualifiers of the base property and provide additional information about how to interpret the base property.

2.3.1 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].

2.4 Host Context

NMF metadata is describing a particular resource which is can be identified via a range of mechanisms. These mechanisms include:

1. Explicit identification using the equivalent of the RDF about URI.
2. Implicit identification provided by the host context in the form of a base uri.
3. Explicit identification using the MPVI identification properties.
4. Implicit identification using MPVI identification properties specified by the host context.
5. Implicit identification using some other host context mechanism.

NMF processors are only required to support the first two mechanisms. MPVI processors will support 3 and 4. Finally, specialized processors that are aware of the host context mechanism will support 5.

2.5 Design Approach

NMF employs a specific approach to schema design and extension. This approach tries to maximize the utility of strong typing while allowing runtime interoperability with metadata for which type information is not available. In addition, NMF attempts to provide as large an amount of interoperability as possible with RDF metadata.

The approach NMF employs provides a simple flat mix-in style of object-oriented design. This approach can be found in use in some Object to Relational mapping layers, which have several alternative ways to address the mapping of class hierarchies to relational tables [AMBLER1].

NMF makes use of the single table per schema in its mapping approach. This means that if you have a type that is conceptually derived from multiple base types, each base type's contribution to the data the derived type encapsulates is maintained in a separate table in the relational model and in a separate BySchema container element in the NMF model.

NMF allows the mix-in of metadata for which type information is not available in specific locations in the NMF data model. The locations are:

- Manifest (specified using the OSTA Manifest file:Manifest element)
- Top-level Composite Property (nmf:Metadata)
- Nested Composite Property (instances of nmf:CompositePropType)

The Manifest and the top-level Composite property are weakly typed in the NMF model in order to allow applications to mix in both metadata and (in the case of the Manifest) non-metadata content for which type information may or may not be available.

The Nested Composite Properties can be typed either strongly or weakly based on the processing assumptions of the schema that is defining the composite property. See best practices for more information.

2.5.1 Naming Patterns and Reserved Names

There are several reserved local-name suffixes that are used by the NMF encoding. The reserved local-name suffixes are any of these character sequences:

- Bag
- Seq
- Alt
- QVal
- Ref
- AnyXML

If these suffixes are to be used in a property name, they need a trailing underscore. The RDF to NMF mapping defines the algorithm for handling these reserved values when they are encountered in the RDF encoding and how to maintain them in a lossless manner. The algorithm uses a simple escaping mechanism where an underscore character is appended to the suffix string when mapping from RDF to NMF and removed when mapping from NMF to RDF.

Chapter 3: NMF Encoding

3.1 Overview

A particular NMF property has a base name and one or more types. The most likely usage of NMF properties will be where a property has a single type. In addition, there will be some scenarios that involve interoperation with existing systems where a property might take on different variant forms.

NMF Properties can be of the following base types:

1. Simple Properties whose values are textual.
2. Composite Properties whose value is a set of properties.
3. Ref Properties whose values are a URI visible to the RDF representation (not textual content).
4. AnyXML Properties whose value is well formed XML.

These base types can be encapsulated in one of the following higher-order types:

1. Qualified Properties whose values are qualified by additional contextual metadata.
2. Array Properties whose values is an array of the base property type (Seq, Bag, Alt).

Each property has an initial base local name that is independent of the type of the property. In fact, some properties can be specified using several alternate value types. As an example, in Dublin Core, some of the properties can take on either a single value or an array of values. if an array property has a single entry, then an alternate form of specifying the property can be employed. Depending on the type of the underlying array element, the property might then be of any of the non-array property types.

Another common variation in property type is encountered in general purpose RDF where any composite property can be specified either inline as a nested resource or out of line as an Ref.

NMF uses a system of naming patterns to explicitly indicate the property type in the name of the property. This is used to allow mapping algorithms to operate even when there isn't access to schema information that indicates the property types.

The naming patterns append a suffix to the base localname of the property that indicates the property type. Here are the naming patterns with a hypothetical property whose localname is "Destination":

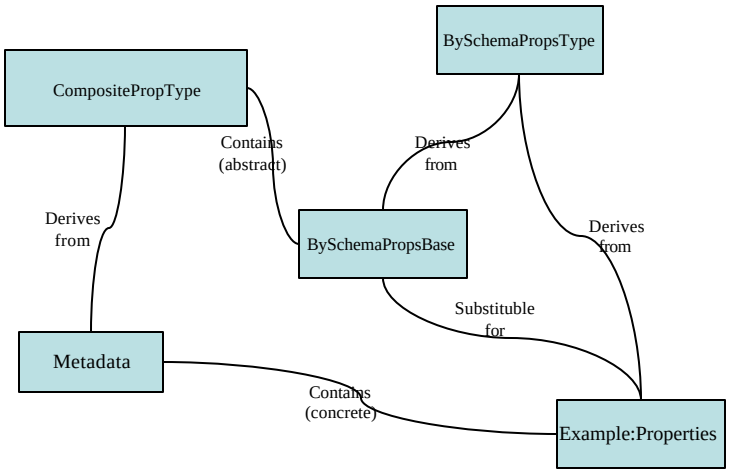
Property Type	Suffix	Example
Simple	none	Destination
Qualified	QVal	DestinationQVal
Composite	none	Destination

Ref	Ref	DestinationRef
Array(Bag)	Bag	DestinationBag
Array(Seq)	Seq	DestinationSeq
Array(Alt)	Alt	DestinationAlt
XML Literal	AnyXML	DestinationAnyXML

Properties are packaged together based on the schema that they are defined in. One or more BySchema Properties elements are then contained in a Properties container. These containers can either occur at the top-level or at a nested level.

The diagram below shows the relationships between the main types, abstract elements and concrete elements. All Composite properties derive from CompositePropType. The two types of composite properties are the top-level Metadata container and nested composite properties.

CompositePropType derived elements contain one or more elements that are substitutable for BySchemaPropsBase. In order to be substitutable for BySchemaPropsBase, they need to derive from the BySchemaPropsType. In the diagram, a hypothetical Properties element defined in a namespace whose prefix is Example is shown as being substitutable for BySchemaPropsBase and derived from BySchemaPropsType. This allows it to be used as a child of any CompositePropType derived element such as Metadata or any composite property that allows BySchemaPropsBase substitutable elements[1].



3.2 Root Metadata Container (file:Manifest)

NMF metadata is usually employed in a scenario where only a single resource is being described. In this context, the top-most element for NMF is the nmf:Metadata (see below).

[1] This is the scenario if we only supported full validation. In practice, we don't make use of this actual content model since it wouldn't allow us to provide partial validation, i.e. validation where we ignore child elements for which schema information is not available.

There are other usage scenarios where metadata about several different resources is contained in a single document portion without employing a host context that provides the separation between each resource description. In these applications, The OSTA Manifest specification [OSTA-MAN] provides a top-level container element with

namespace http://ns.osta.org/manifest/1.0/
localname "Manifest

This element has an open content model so an NMF metadata processor would scan for top-level nmf:Metadata elements and ignore other top-level elements.

3.3 Composite Properties (Top-level and Nested)

NMF metadata about a particular resource (top-level or nested) is specified using one or more properties each of which may be defined in a different schema. Rather than having a single child container element as the value of the composite property, NMF groups the properties from each schema that is being used to describe the resource.

These schema grouped properties are contained in an element that is an instance of nmf:BySchemaPropsType (or a type derived from nmf:BySchemaPropsType). These schema container elements are then contained in the composite property element itself which MUST be an instance of nmf:CompositePropType (or a type derived from nmf:CompositePropType).

The composite property may choose a range of constraints about what properties can be contained in it. These can range from allowing any possible property to be contained and all the way to tightly constraining the contained properties to a specific set as defined in a particular schema.

NMF has as one of its goals to interoperate with representations that are relatively open about what properties can be associated with a resource (composite property). In addition, NMF content may often be processed in environments for which incomplete type information is available.

The nmf:Metadata top-level container supports this goal by using an open content model that allows any child elements to occur in it.

1. the composite property MAY allow content for which type information is not available as part of the content model.

   NOTE: This requires the property to use an open content model due to restrictions of XML schema and current prevalent XML processors.

   a. The children of the composite element MUST still meet all the requirements defined in section 3.4 although the schema will not be able to enforce these requirements.

2. The composite property MAY require the presence of a specific element that is an instance of a specific BySchemaPropsType.

NOTE: In order to extend this schema with additional properties, a new schema must be used that contains the extension properties. This schema appears, as a peer of the base schema as far as the composite property is concerned.

It may be desirable to indicate to a type aware processor that this additional schema is an extension of the base schema. It can't actually be derived from the base schema because it would then violate the

requirement that each schema only contain elements from its namespace. In addition, this would break the ability to have a schema-less mapping algorithm between RDF and NMF. [2]

3. In addition, the composite property MAY require that additional elements (other than the ones allowed by bullet 1) conform to the same base type as the specific element that it specified[3].
4. The composite property MUST be an instance of nmf:CompositePropType (or derived from nmf:CompositePropType).
5. If the element is a top-level composite property it MUST have the name nmf:Metadata.
6. If it is a nested composite property, it can have whatever local name makes sense in the schema[4].
7. The composite property MUST contain only elements that are instances of nmf:BySchemaPropsType (or a type derived from nmf:BySchemaPropsType)[5].
8. Each of these child elements SHOULD be from a different namespace.
    a. If it is not, then the processor MAY choose to only process the first child element from a particular namespace.
9. The child elements MAY be ordered in lexicographical sort order based on the namespace-uri.

3.3.1.1 element CompositePropBase

diagram	
namespace	http://ns.osta.org/nmf/1.0/
type	**CompositePropType**
source	<xs:element name="CompositePropBase" type="nmf:CompositePropType" abstract="true"/>

3.3.1.2 complexType CompositePropType

diagram	
namespace	http://ns.osta.org/nmf/1.0/
used by	element **CompositePropBase** complexType **MetadataType**
source	<xs:complexType name="CompositePropType"/>

3.3.2 Top-level Composite Properties (Metadata and MetadataType)

2 the primary technique for en-masse substituion of a source schema is to define a new target schema that has aliases for all the elements in the source schema. The target schema should also define all the same constraints as the source schema.
VXMP encourages an explicit mix-in style of extension where the schema where a particular property is defined is maintained in the instance document. In the en-masse substitution scenario, the processor is really required to always use the alias type to dereference the property since the property will have a different namespace and may have a different local-name.

[3] The method for specifying this conformance is not described in this specification. It may be described in a future revision.

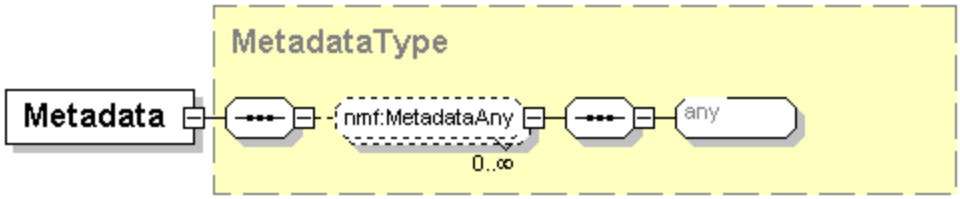
[4] The rules for localname selection are described in the mapping section.

[5] Note again that this cannot be enforced by schema that is supporting partial validation.

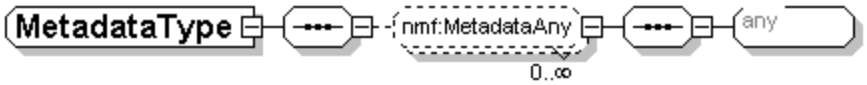
In a top-level context (when not wrapped with a file:Manifest or compatible element), the nmf:Metadata element. provides the binding point between the metadata and the host context. The Metadata element contains one or more elements that are substitutable for the nmf:BySchemaPropsBase abstract container elements . These elements must be instances of a type that derives from nmf:BySchemaPropsBaseType.

The top-level Properties container element can either be the nmf:Metadata element or it can be another element that is derived from nmf:MetadataType. The nmf:MetadataType is derived from nmf:CompositePropType. It adds a single optional attribute called nmf:about which mirrors the functionality of the rdf:about attribute while restricting it to use on top-level resources.

3.3.2.1 **element** Metadata

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"/>`

3.3.2.2 **complexType** MetadataType

diagram	
namespace	http://ns.osta.org/nmf/1.0/
type	extension of **CompositePropType**
used by	element **Metadata**
attributes	Name: about; Type: xs:anyURI; Use: optional; Default: ; Fixed:

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>`

3.3.3 Nested Composite Properties

In a nested context, the composite property will have a name that is specific to the purpose of the property in the schema that is defining it. Property sub-containers encapsulate the properties from a particular schema.

3.4 BySchema Properties

The contents of a composite property are grouped by schema. Each schema's properties are contains in an element that is an instance of a type that derives from nmf:BySchemaPropsType. As described in the previous section, these elements that wrap each schema's properties are themselves wrapped in an element that is an instance of nmf:CompositePropType.

There are two styles of naming the BySchema container element that depend on whether the schema is being defined for:

- compatibility with an existing RDF/XML schema that doesn't employ typednode syntax or make use of rdf:type definitions
- A new NMF schema or an RDF/XML schema that does use typednode syntax and the rdf:type definitions.

- In the first case, the schema MUST use the special localname of 'Properties" for the container element.

NOTE: This provides a minimal level of processing compatability for the mapping from RDF where the composite property may not include the rdf:type information to allow the custom localname to be used.

- In the second case, the schema MAY use a custom local name which will be translated into an rdf:type by the RDF/NMF mapping.

- The container element **MUST** be an instance of a type that derives from the nmf:BySchemaPropsType type either directly or via an intermediate type.

NOTE: The NMF Schema set makes use of a schema definition style that defines all the information for a Properties schema in a ComplexType that extends the nmf:BySchemaPropsBaseType

The properties in the schema **MUST** be specified in alphabetical order sorted on the local-name of the element.

NOTE: This is required in order to have an unambiguous algorithm for mapping external unordered representations such as RDF and relational tables to NMF even when schema information isn't available to the mapping tool.

XML Schema only supports a constrained ability to specify unordered properties, which is not sufficient for non-trivial usage. This requires NMF to mandate an ordering of the children of the Properties element.

- This element **MAY** be substitutable for the nmf:BySchemaPropsBase element.
- Reserved localname suffixes **MUST NOT** be used (other than for their intended use in naming patterns) unless followed by one or more underscore characters. The NMF to RDF mapping algorithm will remove the underscore character that immediately follow the reserved localname suffix.
- If the schema is being mapped from an existing RDF schema, then the RDF/NMF mapping (see 4.4) **MUST** be used to avoid the occurrence of the reserved name suffix values.

3.4.1 Custom Naming (TypedNode)

RDF provides a simple typing system based on a special property called rdf:type that can be associated with any resource. RDF also provides a special syntactic form called typednode syntax where the qname representation of the rdf:type value is used as the name of the resource element (instead of rdf:Description).

NMF directly supports typednode syntax and rdf:type by allowing a schema to specify an additional wrapper element for the schema (in addition to the required element with the "Properties" localname). This allows a very natural and compatible syntax to be mirrored between NMF and RDF (see example below).

- The schema **MAY** define an element. This element will be mapped to the RDF representation using typednode syntax.
  - In this case, the schema namespace **MUST** end in either the "/" or "#" characters in order to allow namespace-uri to qname mapping to be performed.

NOTE: This is a best practise in any case, and SHOULD be used for any new schema that is specified directly in NMF.

In the examples below, the RDF representation uses properties from two schemas to describe the anonymous resource that is the value of the ns3:factsAndFictions property which is itself describing the resource with rdf:about URI of "http://www.foo.com/cool.html". Both schemas have an rdf:type associated with them but only one of them can be used for the typednode syntax form.

The NMF representation generates a separate BySchemaProps element for the properties from each schema and maps the rdf:type of each schema to the name for the wrapper element.

In the first example, the RDF representation chooses one of the two rdf:types to be used as the typednode identifier. In this case, the typednode for the schema identified by the ns1 namespace prefix. The other rdf:type is specified as a reference property.

RDF/XML Representation	NMF Representation
<?xml version="1.0"?> <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:ns1="http://ns1#" xmlns:ns2=http://ns2# xmlns:ns3="http://ns3#">	<?xml version="1.0"?> <nmf:Manifest xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:nmf="http://ns.osta.org/nmf/1.0/" xmlns:ns1="http://ns1#" xmlns:ns2=http://ns2# xmlns:ns3="http://ns3#">
<rdf:Description about=" http://www.foo.com/cool.html">	<nmf:Metadata nmf:about="http://www.foo.com/cool.html">
<ns3:factsAndFictions>	<ns3:Properties> <ns3:factsAndFictions>

<ns1:Facts> <ns1:Fact1>a fact</ns1:Fact1> <rdf:type rdf:resource=”http://ns2#Fictions” /> <ns2:Fiction1>a fiction</ns2:Fiction1> </ns1:Facts>	<ns1:Facts> <ns1:Fact1>a fact</ns1:Fact1> </ns:Facts> <ns2:Fictions> <ns2:Fiction1>a fiction</ns2:Fiction1> </ns2:Fictions>
</ns3:factsAndFictions>	</ns3:factsAndFictions> </ns3:Properties>
</rdf:Description>	</nmf:Metadata>
</rdf:RDF	</nmf:Manifest>

This next example shows the exact same NMF representation resulting from both the schema types being specified using the property form rather than one being specified using the typednode syntax and the other using the property form.

RDF/XML Representation	NMF Representation
<?xml version="1.0"?> <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:ns1="http://ns1#" xmlns:ns2=http://ns2# xmlns:ns3=”http://ns3#”>	<?xml version="1.0"?> <nmf:Manifest xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:nmf="http://ns.osta.org/nmf/1.0/" xmlns:ns1="http://ns1#" xmlns:ns2=http://ns2# xmlns:ns3=”http://ns3#”>
<rdf:Description about=” http://www.foo.com/cool.html”>	<nmf:Metadata nmf:about=”http://www.foo.com/cool.html”>
<ns3:factsAndFictions>	<ns3:Properties> <ns3:factsAndFictions>
<rdf:Description> <rdf:type rdf:resource=”http://ns2#Fictions” /> <ns1:Fact1>a fact</ns1:Fact1> <rdf:type rdf:resource=”http://ns2#Fictions” /> <ns2:Fiction1>a fiction</ns2:Fiction1> </ns1:Facts>	<ns1:Facts> <ns1:Fact1>a fact</ns1:Fact1> </ns:Facts> <ns2:Fictions> <ns2:Fiction1>a fiction</ns2:Fiction1> </ns2:Fictions>
</ns3:factsAndFictions>	</ns3:factsAndFictions> </ns3:Properties>
</rdf:Description>	</nmf:Metadata>
</rdf:RDF	</nmf:Manifest>

3.4.2 **complexType** BySchemaPropsType

diagram	
namespace	http://ns.osta.org/nmf/1.0/
used by	element **BySchemaPropsBase** complexTypes **BySchemaPropsType** **BySchemaPropsType**
source	<xs:complexType name="BySchemaPropsType"/>

3.5 Simple Properties

Simple properties have textual content as their value. The complete set of XML Schema data types are available to specify the constraints on the allowable content. In addition, NMF provides a mapping of the set of property value types specified in RDF [XMP-FW].

3.6 Ref Properties

Ref properties directly model the reference property form of RDF, i.e. the property whose value is a URI. This is explicitly indicated in the RDF/XML syntax using the rdf:resource attribute. NMF distinguishes between the reference and non-reference variations of the property using naming patterns. The reference variation of the property is indicated by appending “Ref” to the localname of the property.

NMF Schema that support both an inline and out-of-line variant of their value need to be able to support the schemaless mapping between NMF and RDF.

They **SHOULD** distinguish between the inline and reference variants by appending “Ref” to the local name of the inline property.

3.6.1 complexType RefPropType

diagram	
namespace	http://ns.osta.org/nmf/1.0/
type	extension of **xs:anyURI**
used by	complexTypes
source	<xs:complexType name="RefPropType"> <xs:simpleContent> <xs:extension base="xs:anyURI"/> </xs:simpleContent> </xs:complexType>

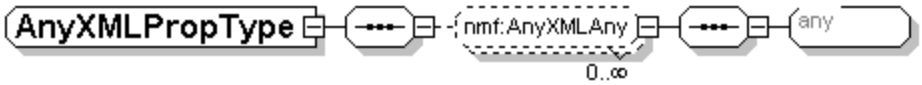
3.7 XML Literal Properties

The NMF syntax and data model is based on a structured definition of hierarchical data structures that are encoded in the XML syntax. This corresponds to a variation of the basic syntax of RDF. There are scenarios where the value of a property is well formed XML that doesn’t correspond to a nested property definition. In this case, the NMF processor requires this to be indicated explicitly since the content model of the property should be treated as opaque XML.

RDF has an analogous issue with differentiating between a property value which is a nested resource and a property value which is well-formed XML. RDF employs a special attribute called rdf:parseType which if given a value of “Literal” indicates that the content of the property is an XML Literal.

In the NMF representation, the fact that the property contains an XML Literal rather than a nested resource is indicated via a naming pattern that appends the string, “AnyXML”, to the base property name.

3.7.1 **complexType** AnyXMLPropType

diagram	
namespace	http://ns.osta.org/nmf/1.0/
type	extension of **PropType**
used by	complexTypes
source	<xs:complexType name="AnyXMLPropType" mixed="true"> <xs:complexContent mixed="true"> <xs:extension base="PropType"> <xs:sequence> <xs:group ref="nmf:AnyXMLAny" minOccurs="0" maxOccurs="unbounded"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

3.8 Array Properties

Array properties can be either ordered, unordered or specify a one of N semantics. This corresponds to the RDF concepts of Bag, Seq and Alt container types.

One big difference between the RDF encoding of Array valued properties and the NMF encoding is that NMF doesn't make use of the explicit array element between the property element and the property values. This is necessary since XML Schema only supports the specification of the immediate content of an element whereas the RDF style of encoding requires the ability to specify the content across three levels of element hierarchy.

NMF supports an explicit typing of the array property via the schema and also uses a naming convention in order to support mappings to/from RDF when schema based translation isn't available. The naming convention is to append the localname of the Array type to the property name that is used in the RDF encoding.

1. A Property MAY be specified to support both a special case syntax for a single entry array and the general purpose syntax for one or more entries.
2. If this special-case syntax if supported, the name of the single entry form **MUST** be the name of the array element (see below).
3. An Array valued property **MUST** have a localname that contains one of the the following character sequences as the final characters in the string:
    a. "Bag" or "Seq" or "Alt"
4. The Array valued property element **MUST** contain one or more child elements whose namespace-uri is the that of the parent and whose local name is that of the parent without the array type character sequence (i.e. with the "Bag" or "Seq" or "Alt" stripped off).
5. If the special case syntax for a single entry is supported then, the parent property will be identical to the child elements used in the array valued syntax.

3.8.1 nmf:Bag

nmf:Bag is the base type for unordered arrays of propertys.

3.8.1.1 complexType BagPropType

diagram	
namespace	http://ns.osta.org/nmf/1.0/
type	extension of **ArrayPropType**
used by	complexTypes
source	<xs:complexType name="BagPropType" abstract="true"> <xs:complexContent> <xs:extension base="ArrayPropType"/> </xs:complexContent> </xs:complexType>

3.8.2 nmf:Seq

nmf:Seq is the base type for both simple and complex ordered arrays.

3.8.2.1 complexType SeqPropType

diagram	
namespace	http://ns.osta.org/nmf/1.0/
type	extension of **ArrayPropType**
used by	complexTypes
source	<xs:complexType name="SeqPropType" abstract="true"> <xs:complexContent> <xs:extension base="ArrayPropType"/> </xs:complexContent> </xs:complexType>

3.8.3 nmf:Alt

nmf:Alt is the base type for arrays of alternative properties.

3.8.4 complexType AltPropType

diagram	

namespace	http://ns.osta.org/nmf/1.0/
type	extension of **ArrayPropType**
used by	complexTypes
source	<xs:complexType name="AltPropType" abstract="true"> <xs:complexContent> <xs:extension base="ArrayPropType"/> </xs:complexContent> </xs:complexType>

3.9 Qualified Properties

Qualified Properties are described in section 2.3 of the RDF specification [RDF].. NMF provides minimal support for qualified properties that are one level deep.. NMF does not make any attempt to leverage qualified properties directly as part of its data model. The qualifications are treated as untyped information that is maintained in the syntax. If the qualifications conform to the NMF property model they will be validated

Qualified properties are properties which have both a primary value and an optional qualification that can be applied to the property value in order to provide additional context for interpretation of the value.

The Dublin Core Metadata Initiative makes extensive use of qualified properties and goes so far as to define two primary types of qualification:

- **Property Refinement.** These qualifiers make the meaning of an property narrower or more specific. A refined property shares the meaning of the unqualified property, but with a more restricted scope. A client that does not understand a specific property refinement term should be able to ignore the qualifier and treat the metadata value as if it were an unqualified (broader) element. The definitions of property refinement terms for qualifiers must be publicly available.

- **Encoding Scheme.** These qualifiers identify schemes that aid in the interpretation of an property value. These schemes include controlled vocabularies and formal notations or parsing rules. A value expressed using an encoding scheme will thus be a token selected from a controlled vocabulary (e.g., a term from a classification system or set of subject headings) or a string formatted in accordance with a formal notation (e.g., "2000-01-01" as the standard expression of a date). If an encoding scheme is not understood by a client or agent, the value may still be useful to a human reader. The definitive description of an encoding scheme for qualifiers must be clearly identified and available for public use.

RDF supports qualified properties via several mechanisms including the use of the **rdf:value** property. This property is used to indicate the primary value of a composite property value. All the other properties contained in the composite property value are then interpreted as qualifying the primary value.

The same problem of context dependant validation occurs with rdf:value as with the other elements from the rdf namespace that are used as wrappers for schema specific relationships. That is the fact that context free validation approachs like XML Schema can only validate immediate children types and the rdf:value cannot have a context dependant type based on the element that is parenting it.

NMF addresses this with the same technique that is used in the case of Arrays. That is to model the Qualified property as if it is a non-qualified property. The analog in the array case, is to model the array property the same as the singleton property.

NOTE: We may want to apply the same dual approach to the Qual/NonQual pairing as we do to the Array/Singleton pairing. I.e. first define the singleton case and then define the other case as an extension of the singleton that encapsulates it.

The Qualified property has the same name as the non-qualified property version but with "QVal" appended to it. It then contains an immediate child that is named the non-qualified version of the name. Below is an example using the Dublin Core Relation property with the RelationType qualifier which is a refinement qualifier.

RDF/XML	NMF
<dc:Relation rdf:parseType="Resource"> <rdf:value>Irene</rdf:value> <dcq:RelationType>cousin</dc:Relation>	<RelationQVal> <dcq:Properties> <dcq:RelationType>cousin</dcq:RelationType> </dcq:Properties> <Relation>Irene</Relation> </RelationQVal>

1. The Qualified form of a property MUST be derived from nmf:QValType.
2. The Qualified form of a property MUST have the same base local name as the unqualified property with a suffix of "QVal".
3. The Qualified form of a property MUST contain a single child element that is an instance of the unqualified form of the property
4. The Qualified form of a property MAY contain one or more child elements that are derived from nmf:BySchemaPropsType

3.9.1 **complexType** QValPropType

diagram	
namespace	http://ns.osta.org/nmf/1.0/
type	extension of **PropType**
used by	complexTypes
source	<xs:complexType name="QValPropType"> <xs:complexContent> <xs:extension base="PropType"/> </xs:complexContent> </xs:complexType>

3.10 Open Content Model Helpers

NMF makes use of weak typing in several key locations in its content model in order to allow partial validation of documents which contain elements for which schema information is not available. This is done using the XML Schema open content model facility using the xs:any construct.

The working assumption underlying this use of xs:any is that if the type information was available it would correctly validate against the schemas. NMF makes use of another feature of XML Schema open content model which is the ability to specify the level of validation that should be applied to the elements that are encountered in the loosely validated locations.

There are two levels of validation which 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. One version uses lax validation and is intended for runtime and production environments. The other uses strict validation and is intended for development environments.

This seems to be the best compromise at the moment for allowing some amount of validation at runtime while still allowing decentralized mix-in of additional property schema.

The helper group elements below are from the development environment version of the utility schema and therefore have the value of strict for their processContents attributes.

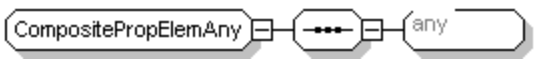
3.10.1 **group** AnyXMLAny

This group element is used by the XML Literal properties to control the level of validation.

diagram	
namespace	http://ns.osta.org/nmf/1.0/
used by	complexType **AnyXMLPropType**
source	<xs:group name="AnyXMLAny"> <xs:sequence> <xs:any processContents ="strict"/> </xs:sequence> </xs:group>

3.10.2 **group** CompositePropElemAny

This group element can be is used by nested Composite properties to control the level of validation. Nested composite properties can either use a tightly bound content model where the specific BySchemaPropsType instances that can be contained are specified or they can use the CompositePropElemAny

diagram	
namespace	http://ns.osta.org/nmf/1.0/
source	<xs:group name="CompositePropElemAny">

	<xs:sequence> <xs:any processContents="strict"/> </xs:sequence> </xs:group>

3.10.3 **group** ManifestAny

This group element is used by the file:Manifest element as a convenience to specify all the open content models in one utility schema.

diagram	
namespace	http://ns.osta.org/nmf/1.0/
source	<xs:group name="ManifestAny"> <xs:sequence> <xs:any processContents="strict"/> </xs:sequence> </xs:group>

3.10.4 **group** MetadataAny

This group element is used by the nmf:Metadata element.

diagram	
namespace	http://ns.osta.org/nmf/1.0/
used by	complexType **MetadataType**
source	<xs:group name="MetadataAny"> <xs:sequence> <xs:any processContents="strict"/> </xs:sequence> </xs:group>

3.10.5 **group** QValAny

This group element is used to control the level of validation for qualifier properties in the QVal properties.

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

used by	complexTypes **abstractQValType** **alternativeQValType** **conformsToQValType** **contributorQValType** **coverageQValType** **hasFormatQValType** **hasPartQValType** **hasVersionQValType** **isFormatOfQValType** **isPartOfQVal Type** **isReferencedByQValType** **isReplacedByQValType** **isRequiredByQValType** **isVersionOfQValType** **languageQValType** **publisherQValType** **referencesQValType** **relationQValType** **replacesQValType** **requiresQValType** **rightsQValType** **sourceQValType** **spatialQValType** **subjectQValType** **tableOfContentsQValType** **temporalQValType** **titleQValType** **typeQValType**
source	<xs:group name="QValAny"> <xs:sequence> <xs:any processContents ="strict"/> </xs:sequence> </xs:group>

Chapter 4: RDF/NMF Mapping

The XMP/NMF mapping is specified in such a way as to allow translators to exist that can perform the mapping without requiring access to schemas for the inputs. The mapping is defined to support bi-directional translation between the RDF/XML encoding and the NMF encoding.

The mapping algorithms make use of several representations including an intermediate representation that allows The XMP to NMF mapping to be described as being composed of several stages or passes. This is done for ease of specification. A translator may perform the mapping in any manner that achieves equivalent results.

The three representations are:

- RDF/XML
- RDF/NMF
- NMF

4.1 RDF/XML Encoding Constraints

The RDF/NMF mapping has a few mechanical constraints on allowable constructs that may be used in the RDF/XML documents that can be mapped to NMF.

NMF has the same constraints on features of RDF/XML as those defined in section 3.6 of the XMP Specification. They following features are not supported:

- The optional rdf:RDF element (rdf:RDF is required)
- Top-level containers (top-level must be rdf:Description or typedNode elements)
- The rdf:ID attribute (ignored on all rdf:Description or propertyElt elements)
- The rdf:bagID attribute (ignored)
- The rdf:aboutEach or rdf:aboutEachPrefix attributes (entire rdf:Description
- ignored)
- Reified statements (they are not generated in the model).
- all rdf:about attributes must appear on top-level rdf:Description or top level typed Node elements only.
- all rdf:about attributes must contain the same value.

4.2 RDF to NMF mapping

The XMP to NMF mapping is defined using a two pass algorithm.

1) The first pass converts the various RDF based syntaxes to a normalized canonical representation. It is described in 4.3.
2) The second pass converts from the normalized canonical RDF representation to NMF. It is described in

4.3 RDF to NMF/RDF (pass 1)

Generic RDF XML syntax is specified in the RDF syntax specification [RDF]. The XMP specification defines a subset of the RDF/XML syntax that it supports. This subset is referred to in this specification as XMP/RDF.

The first step in converting XMP/RDF to NMF/RDF is to convert the XMP/RDF into the basic syntax form of RDF. This forms uses no shorthand representations and is sometimes referred to as the striped representation of RDF. It is described in section 2.2.1 of the RDF specification [RDF].

An additional normalization step is performed to handle the container types and list elements. These must be encoded using the container syntax specified in rules 18-24 of the section 3.2 of the RDF specification [RDF].

Any nested rdf:Description elements with non-empty values for rdf:ID or rdf:about must be converted to independent top-level rdf:Descrption elements and a referenced via an rdf:resource at their previous location in the syntax.

Once these conversion steps have been performed, the next step is to sort all properties lexicographically based on the namespace-uri followed by the local-name in each of the rdf:Description elements at all levels of the hierarchy.

At this point, there will be one or more top-level rdf:Description (or typednode) elements each of which will have different values for their rdf:ID or rdf:about attributes. Nested rdf:Description (or typednode) elements will not have rdf identifiers.

4.4 NMF/RDF to NMF (pass 2)

The NMF/RDF representation is relatively equivalent to the NMF representation.

The following operations are performed on the NMF/RDF representation to obtain the NMF representation:

1. emit the nmf:Manifest element if there is more than one top-level element

   (NOTE: all top-level elements that describe the same resource will have been collapsed by pass 1)

2. for each top-level element element:
   - 2.1. emit the nmf:Metadata open tag
   - 2.2. if there is an rdf:ID or rdf:about attribute
     - 2.2.1. call HandleRDFIdentifier
   - 2.3. for each sequence of properties (from the same namespace) in the element:
     - 2.3.1. call MapProperties(namespace)
   - 2.4. if there are rdf:type properties whose namespace part doesn't match any properties

2.4.1. callMapProperties(rdf:type value)[6]

2.5. emit the nmf:Metadata close tag

4.4.1 HandleRDIdentifier

1. If there is an rdf:ID attribute
   1.1. nmf:about = value with a prepended "#" character.
2. else
   2.1. nmf:about = value of rdf:about
3. emit the nmf:about attribute

4.4.2 IdentifyPropType(PropElem)

This part of the algorithm is applied to a property element (and its contents) in order to determine what the property type is. Note that the input RDF/XML is assumed to be in canonical NMF/RDF representation.

1. if the element contains textual content
   1.1. return type="Simple"
2. else if the element has an rdf:resource attribute
   2.1. return type="Ref "
3. else If the child of the element is an RDF array element (rdf:Seq, rdf:Bag, rdf:Alt)
   3.1. return type="Array"
4. else if the element has an rdf:parseType attribute with a value of "Literal"
   4.1. return type="AnyXML"
5. else if the child is an rdf:Description element
   5.1. if there is a single rdf:value grandchild element
      5.1.1. return type="QVal"
   5.2. else
      5.2.1. return type="Composite"

4.4.3 MapProperties(namespace)

1. If there is an rdf:type whose qname representation has the the same namespace-uri as namespace
   1.1. emit open tag with xmlns=namespace and localname equal to the localname portion of the rdf:type qname.
2. else
   2.1. Emit open tag with xmlns=namespace and localpart="Properties"
3. For each element in the namespace (other than the rdf:type if present)
   3.1. call MapProp(localname of element, isFixed=false)
4. Emit close tag for element defined in 1.

4.4.4 MapProp(orig_localname, isFixed)

1) if the local-name ends with one of the reserved suffixes and isFixed=false:
   a) localname = localname that has a single underscore character appended to the localname.
2) else

[6] this allows rdf:type information to be maintained even for schemas that don't have properties on the resource but have been associated with the resource for other reasons.

a) localname = orig_localname

3) propType = IdentifyPropType()
4) if the proptype is simple
   a) propname = localname
5) else if the proptype is Array
   a) propname = localname + “Bag” or “Seq” or “Alt”
6) else if the proptype is QVal
   a) propname = localname + “QVal”
7) else if proptype is Composite
   a) propname = localname
8) else if proptype is AnyXML
   a) proptype = localname + “AnyXML”
9) else if proptype is Ref
   a) propname = localname + “Ref”
10) emit open tag with propname

11) if the proptype is simple
   a) emit the value
12) else if the proptype is array
   a) for each rdf:li child element
      i) call MapProp(localname, isFixed=true)
13) else if the proptype is QVal
   a) for each namespace that has properties
      i) call MapProperties(namespace).
   b) if the rdf:value child is a reference (rdf:resource attribute)
      i) emit open tag with localname
         (1) emit the value of the rdf:resource attribute
      ii) emit close tag
   c) else if the rdf:value is a simple value
      i) call MapProp(propname, isFixed=true)
14) else if the proptype is Composite
   a) for each namespace that has properties
      i) call MapProperties(namespace)
15) else if the proptype is AnyXML
   a) emit open tag with propname
      i) emit the contents of the element
16) else if the proptype is Ref
   a) emit open tag with propname
      i) emit the rdf:resource value as the text content of the element

17) emit close tag with propname

4.5 NMF to RDF mapping

The NMF to RDF mapping is defined using a two pass algorithm

1) The first pass converts from NMF to the normalized canonical RDF representation.
2) The second pass converts from the canonical RDF representation to the representation preferred by XMP.

4.6 NMF to NMF/RDF (pass 1)

The NMF/RDF representation is relatively equivalent to the NMF representation. The following operations are performed on the NMF representation to obtain the NMF/RDF representation:

1. emit the rdf:RDF open tag

2. for each nmf:Metadata element (in current context):
   2.1. for each top-level child of nmf:Metadata
      2.1.1. if the element doesn't have a localname = "Properties"[7]
         2.1.1.1. emit the element open tag
      2.1.2. else
         2.1.2.1. emit the rdf:Descripton open tag
      2.1.3. if the nmf:Metadata element has an nmf:about attribute
         2.1.3.1. emit it as an rdf:about attribute
      2.1.4. call MapProperties()
   2.2. emit the close tag (either nmf:Metadata or typednode)

3. emit the rdf:RDF close tag

4.6.1 IdentifyPropType(PropElem)

This part of the algorithm is applied to a property element (and its contents) in order to determine what the property type is. Note that the input RDF/XML is assumed to be in canonical NMF/RDF representation.

1. if the element ends with the string "AnyXML"
   1.1. return type="AnyXML"
2. else if the element ends with "Ref"
   2.1. return type = "Ref"
3. else If the element name ends with either "Seq", "Bag" or "Alt
   3.1. return type="Array"
4. else if the element name ends with QVal
   4.1. return type="QVal"
5. else if the element contains textual content
   5.1. return type="Simple"
6. else
   6.1. return type="Composite"

4.6.2 MapProperties()

1. If the localname of the BySchema wrapper element is not "Properties"
   1.1. emit an rdf:type element with an rdf:resource attribute whose value is the concatenation of the schema namespace and the localname.
2. For each element in the BySchemaPropsType wrapper
   2.1. call MapProp(localname of element, isFixed=false)

[7] If the typenode is using the localname of "Properties" this information will not be mapped. How would we be able to differentiate between this and non typednode usage?

4.6.3 MapProp(orig_localname, isFixed)

1. propType = IdentifyPropType()
2. if the proptype is simple
   2.1. propname = localname
3. else if the proptype is Array
   3.1. propname = localname with suffix removed (“Bag” or “Seq” or “Alt”)
4. else if the proptype is QVal
   4.1. propname = localname with suffix removed (“QVal”)
5. else if proptype is AnyXML
   5.1. propname = propname – trailing “AnyXML”
6. else if proptype is Ref
   6.1. propname = propname – trailing “Ref”
7. else if proptype is Composite
   7.1. propname = localname
8. if propname ends with one of the reserved suffixes followed by a trailing “_”
   8.1. propname = propname minus trailing “_”
9. emit open tag with propname

10. if the proptype is simple
    10.1. emit the value
11. else if the proptype is array
    11.1. emit arraytype open tag (rdf:Seq, rdf:Alt, rdf:Bag)
    11.2. for each child element
        11.2.1. emit rdf:li open tag
            11.2.1.1. if the child’s property value is an Ref
                11.2.1.1.1. emit an rdf:resource attribute whose value is the content of child element.
            11.2.1.2. else if the child’s property value is text
                11.2.1.2.1. emit the text
            11.2.1.3. else
                11.2.1.3.1. emit rdf:Description open tag
                11.2.1.3.2. for each Properties child element
                    11.2.1.3.2.1. call MapProperties()
                11.2.1.3.3. emit rdf:Description close tag
        11.2.2. emit rdf:li close tag
12. else if the proptype is QVal
    12.1. if the rdf:value child is a reference (rdf:resource attribute)
        12.1.1. emit open tag with localname
            12.1.1.1. emit rdf:resource attribute with value of child
        12.1.2. emit close tag
    12.2. else if the rdf:value is a simple value
        12.2.1. call MapProp(localname, isFixed=true)
    12.3. for each Properties child element
        12.3.1. call MapProperties().
13. else if the proptype is Composite
    13.1. for each BySchemaProps child:
        13.1.1. call MapProperties()
14. else if the proptype is Ref
    14.1. emit rdf:resource attribute with value of URI.
15. emit close tag with propname

4.7 NMF/RDF to RDF

The NMF/RDF representation is already RDF and can be used as such. There is a transformation that can be performed on the NMF/RDF to make it less normalized.

The processing needs to collapse all top-level resource descriptions with the same rdf:about value into a single description element. In addition, if there is a single rdf:type value for the resource then that value should be used as the typednode representation. If there is more than one rdf:type then the regular resource syntax should be used and the rdf:type properties treated as regular properties as far as the syntax is concerned.

4.8 Metadata Examples

Below are complete representations of both the RDF/XML and NMF versions of Metadata.

4.8.1 Dublin Core

This example shows the use of stand-alone Dublin Core metadata.

RDF/XML Representation	NMF Representation
<?xml version="1.0"?> <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dc="http://purl.org/dc/elements/1.1/">	<?xml version="1.0"?> <nmf:Manifest xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:nmf="http://ns.adobe.com/vxmpns.osta.org/nmf/1.0/">
<rdf:Description about="http://www.foo.com/cool.html">	<nmf:Metadata nmf:about=”http://www.foo.com/cool.html”>
	<Properties xmlns="http://purl.org/dc/elements/1.1/">
<dc:creator> <rdf:Seq ID="CreatorsAlphabeticalBySurname"> <rdf:li>Mary Andrew</rdf:li> <rdf:li>Jacky Crystal</rdf:li> </rdf:Seq> </dc:creator>	<creatorSeq> <creator>Mary Andrew</creator> <creator>Jacky Crystal</creator> </creatorSeq>
<dc:identifier> <rdf:Bag ID="MirroredSites"> <rdf:li rdf:resource="http://www.foo.com.au/cool.html"/> <rdf:li rdf:resource="http://www.foo.com.it/cool.html"/> </rdf:Bag> </dc:identifier>	<identifierBag> <identifierRef>http://cool.html</identifierRef> <identifierRef>http://cool.html</identifierRef> </identifierBag>
<dc:title> <rdf:Alt> <rdf:li xml:lang="en">The Coolest Web Page</rdf:li> <rdf:li xml:lang="it">Il Pagio di Web Fuba</rdf:li> </rdf:Alt> </dc:title>	<titleAlt> <title xml:lang="en">The Coolest Web Page</title> <title xml:lang="it">Il Pagio di Web Fuba</title> </titleAlt>
	</Properties>
</rdf:Description>	</nmf:Metadata>
</rdf:RDF	<nmf:Manifest>

4.8.2 RSS

This example shows the use of RSS1. This format makes use of typenodes, rdf references and sequences. It also has properties from multiple namespaces describing the channel resource.

The properties associated with each resource are grouped by namespace and alphabetically reordered by the mapping between the RDF/XML and NMF representation.

RDF/XML Representation	NMF Representation
<?xml version="1.0"?> <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:sy="http://purl.org/rss/1.0/modules/syndication/" xmlns:co="http://purl.org/rss/1.0/modules/company/" xmlns:ti="http://purl.org/rss/1.0/modules/textinput/" xmlns="http://purl.org/rss/1.0/">	<?xml version="1.0"?> <nmf:Manifest xmlns="http://purl.org/rss/1.0/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:sy="http://purl.org/rss/1.0/modules/syndication/" xmlns:co="http://purl.org/rss/1.0/modules/company/" xmlns:ti="http://purl.org/rss/1.0/modules/textinput/" xmlns:nmf="http://ns.osta.org/nmf/1.0/">
<channel rdf:about="http://www.xml.com/xml/news.rss">	<nmf:Metadata nmf:about="http://www.xml.com/xml/news.rss">
	<channel xmlns="http://purl.org/rss/1.0/">
<title>XML.com</title> <link>http://xml.com/pub</link> <description> XML.com features a rich mix of information and services for the XML community. </description> <image rdf:resource="http://xml.com/universal/images/xml_tiny.gif"/>	<description> XML.com features a rich mix of information and services ... </description> <imageRef>http://xml.com/universal/images/xml_tiny.gif</imageRef> <link>http://xml.com/pub</link> <title>XML.com</title>
<items> <rdf:Seq> <rdf:li resource="http://2000/08/09/xslt/xslt.html"/> </rdf:Seq> </items>	<itemsSeq> <itemsRef>http://2000/08/09/xslt/xslt.html</itemsRef> </itemsSeq> </channel>
<dc:publisher>The O'Reilly Network</dc:publisher> <dc:creator>Rael Dornfest (mailto:rael@oreilly.com)</dc:creator> <dc:rights>Copyright © 2000 O'Reilly & Associates, Inc.</dc:rights> <dc:date>2000-01-01T12:00+00:00</dc:date>	<dc:Properties> <dc:creator>Rael Dornfest (mailto:rael@oreilly.com)</dc:creator> <dc:date>2000-01-01T12:00+00:00</dc:date> <dc:publisher>The O'Reilly Network</dc:publisher> <dc:rights>Copyright © 2000 O'Reilly & Associates, Inc.</dc:rights> </dc:Properties>
<sy:updatePeriod>hourly</sy:updatePeriod> <sy:updateFrequency>2</sy:updateFrequency> <sy:updateBase>2000-01-01T12:00+00:00</sy:updateBase	<sy:Properties> <sy:updateBase>2000-01-01T12:00+00:00</sy:updateBase> <sy:updateFrequency>2</sy:updateFrequency> <sy:updatePeriod>hourly</sy:updatePeriod> <sy:Properties>
</channel>	</nmf:Metadata>
<image rdf:about="http://xml.com/universal/images/xml_tiny.gif">	<nmf:Metadata nmf:about="http://xml.com/universal/images/xml_tiny.gif">
	<image xmlns="http://purl.org/rss/1.0/">
<title>XML.com</title> <link>http://www.xml.com</link> <url>http://xml.com/universal/images/xml_tiny.gif</url> </image>	<link>http://www.xml.com</link> <title>XML.com</title> <url>http://xml.com/universal/images/xml_tiny.gif</url> </image> </nmf:Metadata>
<item rdf:about="http://xml.com/pub/2000/08/09/xslt/xslt.html">	<nmf:Metadata nmf:about="http://xml.com/pub/2000/08/09/xslt/xslt.html">
	<item xmlns="http://purl.org/rss/1.0/">
<title>Processing Inclusions with XSLT</title> <link>http://xml.com/pub/2000/08/09/xslt/xslt.html</link> <description> Processing document inclusions with general XML tools ... </description>	<description> Processing document inclusions with general XML tools ... </description> <link>http://xml.com/pub/2000/08/09/xslt/xslt.html</link> <title>Processing Inclusions with XSLT</title>
</item>	</item>
	</nmf:Metadata>
</rdf:RDF>	</nmf:Manifest>

4.8.3 Qualified Properties

RDF/XML Representation	NMF Representation
<?xml version="1.0"?> <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dc="http://purl.org/metadata/dublin_core#" xmlns:dcq="http://purl.org/metadata/dublin_core_qualifiers#">	<?xml version="1.0"?> <nmf:Manifest xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcq="http://purl.org/metadata/dublin_core_qualifiers#" xmlns:nmf="http://ns.adobe.com/vxmpns.osta.org/nmf/1.0/">
<rdf:Description about="http://www.dlib.org/dlib/may98/05contents.html">	<nmf:Metadata nmf:about="http://www.dlib.org/dlib/may98/05contents.html">
<dc:Title>DLIB Magazine - The Magazine for Digital Library Research - May 1998</dc:Title> <dc:Description>D-LIB magazine is a monthly compilation of contributed stories, commentary, and briefings.</dc:Description>	<Properties xmlns="http://purl.org/metadata/dublin_core#"> <Title>DLIB Magazine - The Magazine for Digital Library Research - May 1998</Title> <Description>D-LIB magazine is a monthly compilation of contributed stories, commentary, and briefings.</Description>
<dc:Contributor rdf:parseType="Resource"> <dcq:AgentType rdf:resource="http://purl.org/metadata/dublin_core_qualifiers#Editor"/> <rdf:value>Amy Friedlander</rdf:value> </dc:Contributor>	<ContributorQVal> <Contributor>Amy Friedlander</Contributor> <dcq:Properties> <dcq:AgenttypeRef> http://purl.org/metadata/dublin_core_qualifiers#Editor </dcq:AgenttypeRef> </dcq:Properties> </ContributorQVal>
<dc:Relation rdf:parseType="Resource"> <dcq:RelationType rdf:resource="http://purl.org/metadata/dublin_core_qualifiers#IsPartOf"/> <rdf:value resource="http://www.dlib.org"/> </dc:Relation>	<RelationQVal> <RelationRef>http://www.dlib.org</rdf:Properties> </Relation> <dcq:Properties> <dcq:RelationTypeRef>http://purl.org/metadata/dublin_core_qualifiers#IsPartOf</dcq:RelationTypeRef> </dcq:Properties> </RelationQVal>
</rdf:Description>	</Properties> </nmf:Metadata>
</rdf:RDF>	<nmf:Manifest>

Chapter 5: Best Practises

There are several areas of best practice in NMF. They are:

- Profile definition and usage.
- Schema definition and usage
- Instance document usage.
- Processing model.

These are described in the following sections.

5.1 Profile Usage

NMF schemas can either be grouped together into a stand-alone schema set or be grouped together as part of a larger schema set that also contains schemas that aren't NMF based. These schema groupings are referred to as profiles.

A profile can contain

- A top-level profile schema that includes various schemas defined in the profile.
- Zero or more non-NMF schemas.
- Zero or more Top-level Property schemas (see section 3.4).
- Zero or more Nested Property schemas (see section 3.4).
- Zero or more simple property types (see section 3.5) contained in one or more schemas.
- Documentation that describes the practices for usage of metadata that makes use of the schemas.

NMF is the preferred metadata interchange format for the MultiPhoto/Video initiative and there are several profiles that are defined as part of the initiative that incorporate a large number of NMF schemas.

Profiles provide a convenient unit of granularity for packaging up one or more schemas and associated documentation and practices.

- A profile **SHOULD** have a single schema that can be included by a user of the profile that in turn includes all schema that are part of the module.
- A profile **MAY** have a single namespace that is used as the target namespace for the module.

5.2 Schema Usage

5.2.1 Minimize Property Variations

NMF Properties can take on one or more of the property types that are supported.

- A property MUST only use at most one of the array forms.
- A property MUST support either the simple or composite form.

5.3 XML Instance Usage

5.3.1 Default Namespaces

NMF metadata is encoded with a potentially large number of schemas. This is due to the fact that NMF encourages a modular schema definition approach that allows a maximal amount of mixing and matching of Schemas. In addition, schemas for composite property values are distinct from those for top-level Property schemas.

All elements in NMF are namespace qualified which requires that either namespace prefixes are used or namespace defaulting is employed. NMF best practise is to use namespace defaulting at the least common ancestor element that is bound to a particular namespace.

Since NMF requires that top-level Property containers and Composite Property containers have the same namespace as all their children, this ends up being a very visually efficient way of encoding the instance data. Below is an example of a fully prefixed vs. a defaulted instance fragment.

Fully Prefixed instance	
<xapS:Properties xmlns:xapS="http://ns.adobe.com/xap/1.0/s/"> <xapS:EntityTag>W/"A weak ETag"</xapS:EntityTag> <xapS:FileDisposition> <fd:Properties xmlns:fdist="http://ns.adobe.com/xap/1.0/sType/FileDisposition#" > <fd:directoryPath>c:\mydir\path\</fd:directoryPath> <fd:filename>aFile</fd:filename> <fd:OS>Windows</fd:OS> </fd:Properties> </xapS:FileDisposition> <xapS:ResourceID>unique timestamp2001-08- 14T11:02:13Z</xapS:ResourceID> <xapS:Size>10000</xapS:Size> </xapS:Properties>	<Properties xmlns="http://ns.adobe.com/xap/1.0/s/"> <EntityTag>W/"A weak ETag"</EntityTag> <FileDisposition> <Properties xmlns="http://ns.adobe.com/xap/1.0/sType/FileDisposition#" > <directoryPath>c:\mydir\path\</directoryPath> <filename>aFile</filename> <OS>Windows</OS> </Properties> </FileDisposition> <ResourceID>unique timestamp2001-08-14T11:02:13Z</ResourceID> <Size>10000</Size> </Properties>

5.4 Processing Model

TBD

1.Change History

I.1 Version 0.40c

- filled out NMF->RDF mapping algorithm
- added description of aliasing support

I.2 Version 0.40b

- Changed namespace from ns.abobe.com/nmf/1.0/ to ns.osta.org/nmf/1.0/
- changed PropsPart to Properties for more readability
- added support for typednodes
    - typednodes can now be used as alternative names for BySchemaPropsType for increased compatability with some RDF representations.
- added support for Ref to deal with rdf:resource issues.
- updated the mapping algorithms.
    - the underscore is now trailing rather than preceding the suffix

I.3 Version 0.40

- removed logos and license verbiage.
- unified concept of Properties and Struct into single concept of PropsPart. There is a PropsContainerType that is derived by both Metadata (the top-level container) and the composite property types.
- added support for more RDF style mapping.
- explicitly support references using RefPart.
- explicitly support qualified values using QVal.
- moved module descriptions to appendices
    - the module descriptions are mostly going to be XML Spy generated documentation
- added initial aliasing support (see **Error! Reference source not found.**).
- added placeholders for the various modules including:
    - EXIF
    - Dublin Core

2.I Conventions

Examples of MPV metadata structures are in Courier font.

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


3.References

[AMBLER]

"Mapping objects to relational databases", Scott Ambler, July, 2000.

Available at: http://www-106.ibm.com/developerworks/library/mapping-to-rdb/#h9

[DC-NMF]

"Dublin Core Normalized Metadata Format Profile Specification 1.0"; OSTA, 19 June 2002,.
Available at http://www.osta.org/mpv/

[DC-XML]

"Guidelines for implementing Dublin Core in XML" Andy Powell,
http://www.ukoln.ac.uk/metadata/dcmi/dc-xml-guidelines/

[DCQ-RDF]

"Expressing Qualified Dublin Core in RDF / XML", Stefan Kokkelink and Roland Schwäenzl, Dublin Core Metadata Initiative Proposed Recommendation, August 29th, 2001.
Available at http://dublincore.org/documents/dcq-rdf-xml/

[DC-RDF]

"Guidance on expressing the Dublin Core within the Resource Description Framework (RDF)", Eric Miller, Paul Miller and Dan Brickley. Dublin Core Metadata Initiative Draft, July 1999.
Available at http://www.ukoln.ac.uk/metadata/resources/dc/datamodel/WD-dc-rdf/

[DCMI-NS]

"Namespace Policy for the Dublin Core Metadata Init iative", Andy Powell and Eric Wagner, Dublin Core Metadata Initiative Recommendation, October 26th, 2001.
Available at http://dublincore.org/documents/dcmi -namespace/

[DC-RDF-Simple]

"Expressing Simple Dublin Core in RDF/XML", Dave Beckett, Eric Miller and Dan Brickley, Dublin Core Metadata Initiative Proposed Recommendation, November 28th, 2001.
Available at http://dublincore.org/documents/dcmes-xml/

[MANIFEST]

"XML Manifest Specification 1.0"; OSTA, 19 June 2002,.
Available at http://www.osta.org/mpv/

[RDF]

"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/

[RSS1]

"Rich Site Summary 1.0", RSS Working Group, December 6th, 2000.
Available at http://purl.org/rss/1.0/spec.

[OSTA-WEB]

"OSTA MulitPhoto/Video Initiative", 2002,
Available at http://www.osta.org/mpv/

[RFC2119]

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

[PRISM]

"PRISM: Publishing Requirements for Industry Standard Metadata", Prism working group, April 9th, 2001.
Available at http://www.prismstandard.org/techdev/prismspec1.asp

[URI]

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

[UCS-2]

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

[UTF-8]

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

[W3C-NSURI]

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

[XML10]

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

[XML-NS]

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

[XMP-FW]

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

[XSCHEMA]

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

[XSL]

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

[XMLBASE]

"XML Base", Jonathan Marsh. W3C Recommendation, June 27th, 2001.
Available at http://www.w3.org/TR/xmlbase/

