

# Draft Federal XML Developer's Guide

December 2001

U.S. Federal CIO Council  
Enterprise Interoperability and Emerging Information  
Technology Committee  
XML Working Group

# Status Of This Document

---

This document is under review by the WG, and will represent the consensus of the WG as an initial set of guidance for the development of XML components within the government when finalized. It is intended to be a living document that will be updated frequently.

## Executive Summary

The global economy is increasingly dependent upon creative solutions to escalating information technology demands. The potential advantages of Internet transmission of payloads of information have highlighted the need to integrate data across applications, systems, and organizations. With the Internet—and particularly web-enabled exchange of data—still in its relative infancy, we have a unique opportunity for governments, business, and industry to foster the cooperative international development of a standardized platform-independent syntax designed to facilitate identification, exchange, and display of data using web transfer protocols. Extensible Markup Language (XML) is emerging as the preeminent tool for developers interested in maximizing system interoperability.

In recognition of XML's significance as a web-based end-to-end solution for information interchange, the Enterprise Interoperability and Emerging Information Technology (EIEITC) committee of the U.S. Federal CIO Council created the Federal XML Working Group in June 2000. The Group's primary responsibilities are to partner with national and international standards organizations in the development of XML and to guide the U.S. Government's transition to XML for electronic data interchange (EDI).

Integral to the Working Group's leadership is the promulgation of written guidelines promoting best practices and recommending federal standards for XML. The CIO Council submits this draft Federal XML Developer's Guide, version .1 for review by all federal agencies pending consideration by the Office of Management and Budget as future Federal Agency policy.

# Contents

---

<b>Chapter 1 Background</b>	<b>1-1</b>
1.1 INTRODUCTION	1-1
1.2 TERMINOLOGY	1-2
1.3 IMPLEMENTATION REQUIREMENTS	1-2
1.4 APPENDICES	1-3
<b>Chapter 2 Software Application Specifications</b>	<b>2-1</b>
2.1 RECOMMENDED XML SPECIFICATIONS	2-1
<b>Chapter 3 XML Component Conventions</b>	<b>3-1</b>
3.1 STANDARDIZED CASE CONVENTION	3-1
3.2 USAGE OF ACRONYMS AND ABBREVIATIONS	3-2
3.3 XML COMPONENT SELECTION AND CREATION	3-3
3.3.2 Creating XML Component Names from ISO 11179 Data Elements	3-8
3.3.3 Choosing XML Component Names	3-9
<b>Chapter 4 Schema Design Conventions</b>	<b>4-1</b>
4.1 SCHEMA LANGUAGES	4-1
4.2 RECOMMENDED SCHEMA DEVELOPMENT METHODOLOGY	4-2
4.3 CAPTURING METADATA	4-5
4.3.1 Application Specific Metadata	4-7
4.3.2 Capturing XML Component Definitions	4-7
4.3.3 Enumerations and Capturing Code Lists	4-8
<b>Chapter 5 Document Annotation Conventions</b>	<b>5-1</b>
5.1 DOCUMENT VERSIONING	5-1
5.1.1 Versioning DTDs	5-2
5.1.2 Versioning XML Schemas	5-2
5.1.3 Versioning Stylesheets	5-3
5.2 HEADERS	5-3

5.2.1 Schemas .....	5-4
5.2.2 Stylesheets.....	5-4
5.2.3 Instances.....	5-5
<b>Chapter 6 Attribute Versus Element Conventions.....</b>	<b>6-1</b>
<b>Chapter 7 Federal XML Registry.....</b>	<b>7-1</b>
<b>Appendix A ebXML and UN/CEFACT.....</b>	<b>1</b>
REPRESENTATION TERMS .....	6
<b>Appendix B Schema Development.....</b>	<b>1</b>
POSSIBLE SCHEMA DEVELOPMENT PROCEDURE SUMMARY .....	1
<b>Appendix C Tools and References .....</b>	<b>1</b>
TOOLS 1	
<b>Appendix D Combined XML Schema Example .....</b>	<b>1</b>
<b>Appendix E Sample XML Document Headers .....</b>	<b>1</b>
Sample Schema Header .....	1
Sample Stylesheet Header.....	5
Sample Instance header.....	8
<b>Appendix F Points of Contact.....</b>	<b>1</b>
<b>Appendix G Glossary and Acronyms .....</b>	<b>1</b>

## FIGURES

Error! No table of figures entries found.

## TABLES

Table A-1. Representation Terms .....	6
Table A-2. Other Representation Terms.....	7



# Chapter 1 Background

---

## 1.1 INTRODUCTION

Federal initiatives are implementing XML-enabled applications very quickly. This document will assist government activities in developing XML implementations in the short term, while lessons learned are collected. It is an adaptation of the updated consensus draft of the Department of the Navy (DON) XML Developer's Guide (version 1.1) of November 7, 2001. The Navy's latest status of this document series is maintained at the NavyXML Quickplace.

This document is an early deliverable of the overall federal strategy for employing XML. It provides general development guidance for the many XML initiatives currently taking place within Departments and Agencies, while the WG is in the process of developing a long-term strategy for aligning XML implementations with government business needs.

This version of the guidance is primarily written to assist developers in creating standardized schemas that describe XML payloads of information. It should be noted that payloads represent only one component required for secure, reliable information exchange. Other components include a specification for reliable messaging (including authentication, encryption, queuing, and error handling), business service registry and repository functions, and transport protocols. Emerging technologies and specifications are, or will shortly, provide XML-based solutions to many of these needs. Various committee's within the CIO Council as well as the architecture team within the Quicksilver Initiative are addressing these areas as part of the overall federal architecture. The XML Working Group will work with the Quicksilver Initiative to develop an XML Primer that will describe each of these components and bring together the overall strategy for capitalizing on XML as a tool for enterprise interoperability.

Paragraphs of this document are broken into three parts:

- ◆ "Guidance" provides a concise summary of requirements and recommendations.
- ◆ "Explanation" provides a brief explanation of the reasoning behind the guidance provided.
- ◆ "Example" provides one or more non-normative examples pertaining to the guidance.

---

The bulk of this document is contained in appendices that are provided as non-normative supplementary information. The appendices should be considered to have a “draft” status, and do not represent the consensus of the WG.

The document is primarily intended for developers already familiar with XML; however, it has a comprehensive glossary, to assist XML beginners. Some of this document focuses on XML Schemas as a tool for interoperability. To realize the maximum benefit, we suggest that you take the time to become familiar with the XML Schema language. An excellent tutorial with labs is available at <http://www.xfront.com/>.

We encourage developers to try the techniques recommended here and provide feedback to the WG. We will collect lessons learned and best practices, updating and expanding this document periodically.

## 1.2 TERMINOLOGY

The terms MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL are used throughout this document, and should be interpreted in accordance with the Harvard University Network Working Group “Request for Comments” # 2119 of “Best Current Practice” # 14 ( RFC 2119i).

The term XML is used to describe a large range of specifications and technologies associated with XML markup. It is critical that activities developing XML-enabled applications have a firm understanding of basic XML terminology. Appendix G provides a list of applicable acronyms and terms.

Many schema languages have been created for expressing validation rules; however, throughout this document the term ‘schema’ with a small ‘s’ is used generically (to include DTDs), while the term XML Schema or just Schema (capital ‘S’) refers specifically to schemas authored in accordance with the World Wide Web Consortium (W3C) XML Schema recommendation.

## 1.3 IMPLEMENTATION REQUIREMENTS

The WG understands that short timeframe XML implementations or pre-existing schema that do not follow this guidance cannot be changed immediately. Activities should read this document and develop a migration plan to evolve their current XML implementations; additionally, the WG encourages submission of feedback as lessons learned are collected.

Most items in this document should be considered as guidance rather than requirements. Specifically, items using the terms MUST, MUST NOT, REQUIRED, SHALL, and SHALL NOT should be considered as the only requirements.



## 1.4 APPENDICES

The appendices are presented in draft form. They do not represent a consensus of the WG. They are provided, as-is, and are to be considered non-normative. The only exceptions are the portions of the ebXML Specifications and Technical Reports quoted in Appendix A.



---

# Chapter 2 Software Application Specifications

---

## 2.1 RECOMMENDED XML SPECIFICATIONS

### GUIDANCE

In general, production applications SHOULD only use software that implements W3C Final Recommendations. Applications using software that implement W3C technical reports at other stages of the development process, or non-W3C specifications, MUST do so with the following restrictions:

- ◆ Production Applications:
  - Prior to creating, incorporating or using software that implements non-W3C specifications activities MUST:
    - Ensure that no competing W3C endorsed recommendation exists or is being developed.
    - Ensure that the specification is a product of a credible, recognized consortium or organization such as the Organization for the Advancement of Structured Information Standards (OASIS), the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT), the OMG, OAG, UDDI, RossettaNet, or BizTalk.
  - Activities MAY choose to implement W3C technical reports with a Proposed Recommendation status provided they are committed to immediately update software should any changes be made when the report reaches final status
- ◆ Pilot Applications: Activities developing pilot applications (as a precursor to production) MAY also implement software that conforms to W3C technical reports with a Candidate Recommendation status.
- ◆ (Advanced Concept) Demonstrations: Activities developing demonstration applications (as a proof of concept) MAY also implement software that conforms to W3C technical reports with a Working Draft status.
  - Exception: Activities MAY implement software that conforms to the SOAP 1.1 W3C Note, but MUST then be ready and committed to update software to the SOAP 1.2 specification when it reaches Final Recommendation status.
- ◆ Regardless of the stage of application productions, activities MUST NOT implement specifications or standards that compete with W3C technical work.

---

## OTHER GUIDANCE

- ◆ All component software (XML parsers, generators, validators, enabled applications, servers, databases, operating systems), and other software acquired or used by federal activities SHALL be fully compliant with all W3C XML technical reports holding recommended status when such reports exist.
- ◆ Proprietary extensions to W3C Technical Reports:
  - MUST NOT be employed in any software or XML document (instance, schema, stylesheet) that will be shared publicly with activities outside a particular development environment.
  - SHOULD only be employed locally (within a homogeneous development environment) after careful evaluation of possible impacts on cross-platform interoperability, and dependency on software from a single vendor. This decision MUST only be made by government program managers.

## EXPLANATION

W3C Technical Reports has a complete list of W3C reports in all stages of development. The following table provides a list of XML specifications or standards that are not W3C recommendations (yet). Two categories are provided. The “Recommended” column represents widely adopted standards that are believed to be mature and uniformly supported by software implementations. The “Maturing” column represents other standards that the WG believes to be sufficiently mature, however they may not be uniformly supported in existing software implementations, so caution is advised. Future versions of this document will add additional specifications from other standards bodies and efforts such as ebXML, OASIS, UN/CEFACT, etc.

Recommended	Maturing
SAX1 1.0 and 2.0	SOAP 1.1 (W3C Note)

SOAP 1.1 has been adopted by various commercial and federal activities such as ebXML. Members of the WG have evaluated the specification and believe that it is sufficiently stable and mature to support production implementation. SOAP 1.1 exists as a W3C Note, however SOAP 1.2 is being pursued by the W3C XML Protocol Working Group. When it becomes a Final Recommendation, activities

---

<sup>1</sup> SAX is not a specification developed by a standards body or the W3C. It is an open source project maintained by a community of developers. SAX parsers have been written for several languages, but the only platform independent version is the Java API. A parser that is SAX compliant must implement an equivalent to the Java API, which is provided at the SAX homepage.

with SOAP 1.1 implementation **MUST** have planned for and be ready to migrate to SOAP 1.2.

Application vendors often provide proprietary extensions to adopted standards. These extensions may simplify the job of software developers, but they also make developed systems dependent on software from a single vendor, and often they also restrict the software to being run on a single vendor's operating system or hardware. Government program managers must have the final say in the decision to employ such extensions, even when doing so inside a single system's boundaries or within a homogeneous development environment. When making such a decision, government program managers will be guided by potential impacts on interoperability, not only by current information exchange instances, but by potential information exchange instances as well.

#### EXAMPLE

To further illustrate the guidance regarding use of proprietary extensions to W3C Technical Reports, we provide two examples:

- ◆ Example 1: An activity developing an XSL stylesheet is using the XALAN XSL processor. Developers discover that the XALAN software has implemented an extension to XSLT that allows generation of multiple output HTML documents from a single stylesheet. This is convenient since the project requires multiple outputs. The lead project manager consults with the government program manager; the program manager agrees to allow the use of this proprietary extension, provided a stylesheet without the extension is also delivered.
- ◆ Example 2: An activity is developing a Visual Basic application for deployment in a Windows 2000 environment. In that application, the MSXML DOM API is used to manipulate XML. Microsoft has added many convenient extensions to the W3C DOM recommendation that the developers want to use. Since the programming environment is restricted to the Microsoft environment (Windows and Visual Basic), the government program manager agrees to allow the use of the MSXML DOM.

The key difference between these examples is software code portability. In the first example, the stylesheet should be able to run in any environment (operating system, language, and XSL processor), therefore a strictly XSLT conformant deliverable was required. In the second example, code portability was not an issue since the project was restricted to the Microsoft environment already due to the choice of programming language and operating system.



## Chapter 3 XML Component Conventions

---

### 3.1 STANDARDIZED CASE CONVENTION

#### GUIDANCE

Federal developers SHALL adopt the camel case convention, as defined by the ebXML Technical Architecture, when creating XML component names. Excerpts are provided in Appendix A.

- ◆ XML Elements and XML Schema Types use upper camel case: The first letter in the name is upper case, as is the letter beginning each subsequent word.
- ◆ XML Attributes use lower camel case: Like upper camel case, except the first letter of the first word is lower case.

#### EXPLANATION

Major XML consortia such as OASIS, Universal Business Language, UN/CEFACT, RosettaNet, Biztalk and ebXML (see Internet references in Appendix C) have all adopted the camel case convention for XML component naming, with ebXML differentiating between upper and lower camel case.

#### EXAMPLE

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
Example of an upper camel case element and lower camel case
attribute
-->
<UpperCamelCaseElement
  lowerCamelCaseAttribute="foo"/>
```

---

## 3.2 USAGE OF ACRONYMS AND ABBREVIATIONS

### GUIDANCE

Federal developers SHOULD follow the ebXML guidance for usage of acronyms or abbreviations in XML component names<sup>2</sup> with the following caveats:

- ◆ Acronyms SHOULD NOT be used in XML element and attribute names. When acronyms are used they MUST be in upper case.
- ◆ Abbreviations MUST NOT be used in XML element and attribute names.
- ◆ For XML Schema types, abbreviations MUST be avoided while acronyms SHOULD NOT be used consistent with the rest of this guidance.
- ◆ While commonly used acronyms SHOULD NOT be used in element and attribute names, the decision to use an acronym SHALL be made by program and/or functional managers rather than by application developers. The decision to use an acronym MUST be based the belief that its use will promote common understanding of the information both inside a community of interest as well as across multiple communities of interest. When an acronym does not come from a credible, identifiable source or when it introduces a margin for interpretation error, it MUST NOT be used.
- ◆ Acronyms used in component names MUST be spelled out in the component definition that is required to be included via schema annotations (as XML comments or inside XML Schema annotation `<xsd:documentation>` elements) (see Section 4.3.2). References to authoritative sources from which the acronyms are taken MUST also be included in schema documentation.
- ◆ Underscores ( \_ ), periods ( . ) and dashes ( - ) MUST NOT be used.
- ◆ Verbosity in tag size should be limited to what is required to conform to the Tag Name Content recommendations. When tags will be used in data-base structures, a limit of 30 characters is recommended.

### EXPLANATION

XML documents that rely heavily on terse abbreviated component names are difficult to understand, and subject to misinterpretation. The general consensus among the major XML standards development consortia is that abbreviations should be avoided and acronyms used sparingly. Government business language is heavily laden with both acronyms and abbreviations, and it is often difficult to

---

<sup>2</sup> This guidance is restricted to component names only; it does not apply to attribute or element values. For instance the attribute `measureUnitCode="HZ"` where HZ represents Hertz is acceptable as long as the code list from which HZ was taken is published a readily available.



distinguish between an acronym and an abbreviation (e.g., CONOPS). After significant deliberation, the WG has adopted the position that abbreviations result in ambiguous markup and should never be used, and that acronyms for use in element and attribute names are acceptable where they make sense, but should in general be avoided. The working group strongly recommends that the decision for exception acronym usage be based on a management decision that such usage will actually promote understanding.

#### EXAMPLE:

This is an example of providing an element definition in a DTD. Note that the acronym “EPA” is spelled out in the definition.

```
<!--EPAFacilityIdentificationCode
  Definition: A 6-digit code used to uniquely identify
  Facilities with statutory environmental data reporting
  responsibilities to the Environmental Protection Agency (EPA)
-->
<!ELEMENT EPAFacilityIdentificationCode (#PCData)>
```

## 3.3 XML COMPONENT SELECTION AND CREATION

#### GUIDANCE

Draft Federal XML Tag Standards Policy requires that existing federal XML components be used if suitable. Therefore, the Federal XML Registry (FXR) MUST be searched for existing suitable components prior to creation of new components. There are three possible results for this search. Components may be fully suitable, partially suitable, or undiscovered.

- ◆ A component is suitable if:
  - it satisfies the element domain requirements,
  - it is in upper/lower camel case depending upon whether it is an element, attribute or type,
  - is either named after a “business term“, or conforms to ISO 11179 conventions, and
  - abbreviations and acronyms are spelled out in the component definition.

If the component is suitable, it MUST be used, and use of that component MUST be registered within the FXR.

- 
- ◆ A discovered component is considered not suitable if any of the above conditions are not met.

If the component is not suitable, you **SHOULD** create a suitable component, selecting XML component names as follows:

- ◆ Create a “dictionary entry” using the ISO 11179 rules as modified by ebXML and UN/CEFACT (see Appendix A).
- ◆ Create an XML Schema Type derived from ISO 11179-compliant name converted to upper camel case. You **SHOULD** document the type with metadata from the FXR, such as the definition, URL to the item, and registry identifier. Additionally you **SHOULD** apply any domain restrictions to the type rather than the element. Additionally you **MAY** document any mappings to authoritative federal data models or data element definitions in the element’s definition (see Section 4.3.2).
- ◆ Create an XML Element that is named according to the ISO 11179-derived type name. Additionally, you **MAY** document any mappings to existing standard business terms (Section 3.3.1).
- ◆ For XML Attributes, use an ISO 11179-compliant name in lower camel case.
- ◆ For an XML DTD, create elements that are named after ISO 11179-compliant names in upper camel case, and document the ISO 11179 name in the DTD as an XML comment.
- ◆ Register the new element and its relationship to the existing FXR element in the appropriate namespace of the Federal XML Registry.
- ◆ If no component is found, you **SHOULD** create XML component names following the rules defined above for unsuitable components, except that there will be no reference to an existing FXR element.

When used as XML component names, the ISO 11179 element names **SHALL** be converted to camel case by removing the periods and spaces, and adjusting the capitalization.

#### EXPLANATION

The Draft Federal XML policy requires the reuse of XML elements registered in the FXR if those tags are found suitable. The intent of this guidance is to provide clarification as to what suitability means, and reinforce the mandate that the registry be searched as a starting point for suitability determination.

In the case where an element has been identified as a candidate for reuse but it fails suitability criteria, the above guidance provides a solution for creation of a

suitable element while maintaining a semantic relationship to the initially discovered candidate.

For creation of XML elements when no suitable element exists in the FXR, the WG recommends the ebXML modified ISO 11179 data element naming convention as solid basis for XML component creation. In summary, an ISO 11179-compliant data element name consists of three parts:

- ◆ An “Object Class” term, which describes the kind of thing to which you refer. This Object Class may consist of one or more words, some of which may be context terms.

For example, the ISO 11179 name ‘Acoustic Signal. Frequency. Measure’ has the Object Class ‘Acoustic Signal’.

- ◆ A “Property Term” which is the property of the thing to which you refer, which may consist of one or more words. For example, the ISO 11179 name ‘Acoustic Signal. Frequency. Measure’ has the Property Term ‘Frequency’.
- ◆ A “Representation Term” which identifies allowable values for an element. This list is taken from an enumerated list of allowable representation types (see Appendix A). For example, the ISO 11179 name ‘Acoustic Signal. Frequency. Measure’ has the Representation Term ‘Measure’.

The ebXML Technical Report, Naming Convention for Core Components provides 14 “rules” for constructing proper data element names. Some considerations are:

- ◆ When the Representation Type and the Property Term are redundant, the Property Term is dropped, so ‘Item. Identification. Identifier’ becomes ‘Item. Identifier’
- ◆ When an element describes an entire class of things (e.g., not a specific property of it), the Property Term may again be dropped, for instance ‘Documentation. Identifier’
- ◆ An aggregate component shall have a representation type of ‘details’.

Note that ISO 11179 names **SHOULD** be made directly into XML component names:

- ◆ For XML Schema types and XML attribute names
- ◆ For XML element names.

---

The excerpts provided in Appendix A were taken from draft documents that are evolving rapidly. This information SHOULD be used as guidance only, but may prove helpful.

#### EXAMPLE

The following example is an excerpt from that provided in Appendix D.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDe-
  fault="qualified" attributeFormDefault="unqualified">
  + <xs:complexType name="MeasureType">
  - <!--
    Full content of MeasureType not provided here. See Appendix E.
  -->
  </xs:complexType>
  - <xs:annotation>
    - <xs:documentation source="http://www.ebxml.org/specs/ccDICT.pdf">
      - <ebXML>
        <CoreComponent UID="core000152">Text. Type</CoreComponent>
      </ebXML>
    </xs:documentation>
  </xs:annotation>
  - <xs:simpleContent>
    - <xs:extension base="xs:decimal">
      <xs:attribute name="measureUnitCode" type="xs:string" use="optional" de-
        fault="HZ"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
- <!--
  ISO 11179-derived type name
-->
- <xs:complexType name="AcousticSignalFrequencyMeasure">
  - <xs:simpleContent>
```

```

- <!--
Domain restriction placed in type
-->
- <xs:restriction base="MeasureType">
    <xs:totalDigits value="10"/>
    <xs:fractionDigits value="3"/>
    <xs:pattern value="\d*.\d{3}"/>
    <xs:attribute name="measureUnitCode" fixed="HZ"/>
</xs:restriction>
</xs:simpleContent>
</xs:complexType>
- <!--
Element named after business term, "Acoustic Frequency"
-->
- <xs:element name="AcousticFrequency" type="AcousticSignalFrequencyMeasure">
    - <xs:annotation>
        - <!--
        Annotation maps element to DII COE registered element
        -->
        - <xs:documentation
            source="http://diides.ncr.disa.mil/xmlreg/user/detail.cfm?ir_id=8358">
            - <COEXMLRegistry>
                <Namespace prefix="TAR">Tracks and Reports</Namespace>
                <TagName>ACOUST_SIGNA_FREQ</TagName>
                <Definition>ACOUSTIC SIGNATURE FREQUENCY. THE FREQUENCY
                OF AN EMITTED ACOUSTIC SIGNAL TO THE NEAREST ONE
                THOUSANDTH HERTZ.</Definition>
                <RegistryID>8358</RegistryID>
            </COEXMLRegistry>
        </xs:documentation>
    </xs:annotation>
</xs:element>
- <!--

```

---

COE element name made synonymous with camel case business term through use of substitution group

—>

```
- <xs:element name="ACOUST_SIGNA_FREQ"
  type="AcousticSignalFrequencyMeasure" substitution-
  Group="AcousticFrequency">
  - <xs:annotation>
    <xs:documentation>Business Term</xs:documentation>
  </xs:annotation>
</xs:element>
</xs:schema>
```

### 3.3.2 Creating XML Component Names from ISO 11179 Data Elements

#### GUIDANCE

XML components SHOULD be named after ISO 11179 data element names:

- ◆ XML Elements SHOULD be named after ISO 11179 data element definitions.
- ◆ XML Attributes SHOULD be named after ISO 11179 data elements.
- ◆ XML Schema types MUST be named after ISO 11179 data elements

ISO 11179 names MUST be converted to suitable XML component names by converting to camel case by removing white space and periods:

- ◆ Types and elements are converted to upper camel case.
- ◆ Attributes are converted to lower camel case.

#### EXPLANATION

As discussed in the explanation portion of Section 3.3, ISO 11179 part 5 provides a standard for creating data elements. This standard employs a dot notation and white space to separate the various parts of the element and multiple words in a part respectively. In order to meet XML requirements for component naming, the ISO 11179 name must be converted to a Name Token.

The ISO 11179 part 5 standard provides a way to precisely create a data element definition and name. Using or referencing this name in a schema provides analysts with a better understanding of XML component semantics, while using business terms improves readability, it results in ambiguous element names that fail to convey sufficient semantic values to support a comprehensive understanding of the metadata to be conveyed in the markup.

Requiring type to conform to ISO 11179 conventions will facilitate automated analysis of schema components during any harmonization efforts.

The upper and lower camel case conventions are adopted from ebXML.

#### EXAMPLE

In the example of Section 3.3, the type ‘AcousticSignalFrequencyMeasure’ was created from the ISO 11179 standard data element ‘Acoustic Signal. Frequency. Measure’.

### 3.3.3 Choosing XML Component Names

#### GUIDANCE

The selection of XML component names **MUST** be a thoughtful process involving business, functional, database, and system subject matter experts. In the schema design process, federal XML developers **MAY** use temporary or dummy XML component names while consensus is sought on more carefully designed and defined names.

The creation and/or selection of XML component names and business terms

- ◆ **MUST** involve domain subject matter experts (operational personnel, program managers, etc), functional data experts (database administrators, functional data manager, data modelers, etc.) and software developers—application developers **MUST NOT** be left on their own to perform this function,
- ◆ uses existing definitions from standard data element definitions, and
- ◆ **SHOULD NOT** occur just for the sake of having one—the existence and use of business terms **SHOULD** be determined by consensus of a community of users. When a business term is not apparent or does not exist, the ISO 11179-compliant name **MAY** be used as the XML component name instead.

#### EXPLANATION

At a business level, the primary function of XML is to provide a metalanguage for rigorously specifying the syntax of information exchange. Since information ex-

---

change involves multiple parties (at a minimum one sender and one receiver), XML specifies agreements between parties within a community of interest for a particular domain of information. XML itself however does not require or provide a mechanism for defining semantics (precisely what is meant by a particular term), however to achieve interoperability, both the syntax and semantics must be explicitly defined. The process of selecting proper component names and reaching agreements on the definitions is primarily a business function of XML and MUST involve all stakeholders. Frequently, application developers who are on the leading edge of technology will understand the benefits of and implement XML in IT systems before business personnel become involved. As a result, XML component names often are not useable by an entire community, seriously impeding interoperability.



## Chapter 4 Schema Design Conventions

---

### 4.1 SCHEMA LANGUAGES

#### GUIDANCE

Only W3C-recommended languages SHALL be used within the government for describing documents—specifically, the DTD and the W3C-recommended XML Schema language.

All activities developing data-oriented schema in DTD syntax SHOULD plan on migrating to XML Schemas in their next software release.

Federal XML developers MAY elect to use DTDs for markup of data that is strictly document-oriented (paragraph, chapter, appendix...), however the XML Schema language is the preferred method.

#### EXPLANATION

The WG recommends the XML Schema language method for creating schemas. XML Schemas provide a rich syntax for expressing metadata. Some of its features include:

- ◆ Structures are defined, in Part 1 of the XML Schema recommendation, that allow the expression of relational (keyed) data, and object-oriented (type inheritance) relationships. While the Schema allows both relational and object relationships, XML is a hierarchical language that lends itself more to the object-like behaviors.
- ◆ Several flexible options for defining element content models are specified.
- ◆ The language also provides for better modularity by allowing two different ways—”include” or “import”—to reuse external Schemas.
- ◆ Part 2 of the XML Schema recommendation deals with data types. The DTD syntax allows for expression of only a few data types, and only one data type (a string) may be assigned as contents of an XML element. The others are for different types of XML attributes. XML Schemas have dozens of built-in data types, and allow creating custom data types from combinations of the built-in set.
- ◆ The concept of a “type” is extended beyond simple data types (string, Boolean, integer, etc.). Complex types may be declared and named, creating stereotype content models of other XML elements; these can be ex-

---

tended, restricted, and assigned to XML elements in an “object-like” fashion.

- ◆ Several means of further-constraining element and attribute values are provided, including use of Regular Expressions and predefined enumerated value lists to constrain element and attribute values. An entire range of facets (e.g., minimum/maximum values and lengths) are also available for further constraining an element’s range of possible values. DTDs do not allow use of regular expressions and only allow enumerated value lists for attributes.

For activities that intend to migrate towards XML Schemas, an excellent free XML schema tutorial<sup>ii</sup> is available. It provides both detailed presentations and hands-on labs. Additionally, a series of XML Schema best practice papers<sup>iii</sup> is available. These papers provide more XML Schema development technical detail than is provided here.

## 4.2 RECOMMENDED SCHEMA DEVELOPMENT METHODOLOGY

### GUIDANCE

Federal XML developers **SHOULD** adopt the practice of developing schemas based on information exchange requirements identified via business process modeling. The information modeling process and the XML schema creation process **SHOULD** be separate and distinct steps.

Business process models and corresponding document models describing information exchanged in the processes **MAY** use the Unified Modeling Language (UML) if appropriate. For example, the UN/CEFACT-adopted Unified Modeling Methodology (UMM), based on UML, **MAY** be used for the process modeling. The WG expects to evaluate the UMM for applicability to government data domains for possible official adoption at a later date.

Database modeling languages that are oriented primarily toward describing information via relational (keyed) structures **SHOULD NOT** be used for modeling of systems and information that will primarily use XML as the data exchange format.

Schema development **SHOULD** take place as a team effort, involving functional data experts, business experts, program managers, and IT specialists. The WG also highly encourages collaboration between activities developing schemas within related information domains.

Conversely, schema development **SHOULD NOT** be solely the function of IT specialists. XML component names in general **SHOULD NOT** be taken directly from underlying relational database table and column names, unless the elements

within that database have been named and created in accordance with a federal standard that represents concurrence by an entire Community of Interest (COI).

#### EXPLANATION

The single most critical factor in creating logical, reusable schemas for information exchange in XML is the separation of the information modeling process from the schema creation process. Information should be modeled independently of creating a schema. This allows stakeholders to focus on creating logical, consistent representations of information without getting distracted by the myriad of schema-design options that have nothing to do with the information. Once an acceptable information model has been created, mapping rules from the model to a schema can be used or developed, which makes schema creation straightforward. This is the most important step, as well as the most often neglected.

Newly-trained or inexperienced developers typically begin creating schemas on an ad hoc basis, without the involvement of business functional experts or a carefully crafted information model that lends itself to expressing hierarchical, object like relationships. Application developers working without management and functional involvement and without an appropriate model are often tempted to create XML quickly and easily from relational database table and column names. XML components produced in this fashion have very terse, abbreviated, and generally unreadable names, which are often not reusable by other systems or lack concurrence within the community of users.

The result of the actions in the above paragraph is inevitably a poorly-designed set of schemas with little reusability, extensibility, or readability. This translates into rework later at additional expense.

Most XML uses can be conceptualized as business processes in which communities of users share information. Successful schema development should be based on analyzing, documenting, and reaching consensus on the business processes, the parcels of information (documents) exchanged in those processes, and the structure of a commonly understood vocabulary/grammar for creating the documents.

The focus of XML schema and component development should be on creating XML languages that are understood by a community of stakeholders who engage in business processes together. In this context, the term business process has a greater scope than just business-to-business transactions (B2B) where products are bought and sold for money. Some examples:

- ◆ An EPA activity wishes to make reference tables of XML-format code lists available to its community. Here the process is consumer-to-application (C2A)/application-to-consumer (A2C) and application-to-application (A2A). A user (consumer) may request the table data via a web-browser (C2A); the activity receives the request and returns XML

---

that is transformed to HTML (A2C). An application may also request and receive the same information in XML format via SOAP (A2A).

- ◆ A Treasury application wishes to make fiscal data—from messages available on a publish-subscribe or broadcast basis—to Treasury analysts and other Treasury applications.
- ◆ DoD logistics activity wishes to store product data from an acquisition in a neutral format so that at some future point it can be parsed and read into any database for future processing by other activities needing it. In this case, the process can be thought of as consumer-to-consumer (C2C), because the product data that is received by the acquiring consumer should be represented in an XML language that is understood by other consumers within the community.

Relational modeling languages like IDEF1x are appropriate for logical and physical enterprise data modeling of complex systems or data warehouses that will be implemented primarily by relational data bases. However, it is more difficult to model hierarchical, object-like relationships expressed by XML in this language. Relational modeling focuses the efforts of the modeling exercise on the efficient representation of data as a set of normalized entities. This simplifies the process of creating relational databases, but complicates the process of understanding the hierarchical nature of information, and it often hides or neglects critical object like aspects of the domain.

XML is an information sharing metalanguage that is inherently hierarchical. It is better represented via graphical modeling languages, which allow capture of object relationships versus key/key-reference relationships of normalized entities. The WG recommends that activities interested in capitalizing on XML as an information exchange medium take the time to learn the UML. UML is rapidly becoming the de facto industry standard for system requirements analysis and business-process and information modeling, as well as software design. It provides a common language that business experts, managers and IT specialist can use throughout all phases of a system's implementation (requirements discovery, analysis, business rules and workflow documentation, software design, and deployment).

Many data-modeling languages have an object orientation. However, products supporting the direct creation of XML DTDs and/or Schema from UML are becoming available. A number of different, non-standards based approaches are incorporated in these tools. However, the UN/CEFACT EDIFACTiv (EDI for Administration, Commerce and Transport) Working Group (EWG) is undertaking to develop an international standard for UML to XMLv mapping that will even further improve future tool support. By taking the time to create UML static structure models of information exchange requirements, schemas can be automatically generated and updated as standards and models evolve. This will ultimately drive down the cost of implementing XML based systems.

UML to XML tools are in their infancy. Due to lack of a standard, each tool works differently at present. However, by taking the time to learn UML now, and beginning the process of creating information models in UML, government activities will position themselves to capitalize on future advancements.

**Regardless of the modeling language chosen, it is useful to construct and use information and data models that are independent of XML specific syntax. This will allow stakeholders involved in schema design to separate information modeling decisions from XML design decisions.**

#### EXAMPLE

A proposed procedure for schema development is presented in Appendix D. It is non-normative, and provided as an example only.

### 4.3. CAPTURING METADATA

#### GUIDANCE

Federal XML developers **SHOULD**, within reason, capture as much metadata as possible in a schema.

The schema language chosen (DTDs or XML Schema) will impact the amount of metadata that can be expressed and the ability of applications to access the metadata for processing.

- ◆ For DTDs, XML comments **MAY** be used to annotate the DTD with definitions and constraints, which the DTD syntax does not allow.
- ◆ Alternatively, for DTDs, fixed attributes **MAY** be used to capture the metadata.
- ◆ For XML Schema, metadata may be captured in a number of ways, as is discussed in the following sections. These are the four primary ways of capturing metadata:
  - Domain value restrictions **SHOULD** be captured by the use of built-in Schema data types, the construction of custom data types, the assignment of enumerations to XML component values, the use of regular expressions, and minimum/maximum value constraints.
  - Metadata regarding the structure and cardinality of components **SHOULD** be captured by expressing element order as a (set of) choice(s) or an ordered or unordered sequence. Additionally, the exact number of times an element can, (or must) be repeated **MAY** be specified.

- 
- Logical relationships or relationships to existing data dictionaries and models (such as the Department of Defense Data Dictionary System (DDDS), ebXML core components, or Federal Reference Data Sets) may be expressed by the use of types or Schema annotations.
  - An element's definition, sources of definitions or code lists, version information, and other metadata MAY be captured by the use of Schema annotations.
  - ◆ Developers MAY consider the creation of a verbose semantic schema and a compact schema strictly for document validation purposes.
  - ◆ Alternatively, schema documentation and annotations MAY be provided by creating a schema guide that is URL accessible and referenced in the header of the schema. Tools such as XML Authority and XML Spy 4.x provide excellent documentation generation capabilities that can partially automate this process.

## EXPLANATION

The schema is more than just a document structure validation tool. The XML Schema language, in particular, has a rich feature set for capturing extra metadata that can provide:

- ◆ Data element definitions through the use of annotations
- ◆ Detailed domain value constraints
- ◆ Logical data element pedigree through the use of annotations and types.

By capturing this metadata, the schema becomes an interoperability tool, because analysts can read it and understand the meaning and derivation of various XML components. Several sources of metadata exist that can be used to derive XML components. These include:

- ◆ The Federal XML Registryvi.
- ◆ The initial set of ebXML core components (see the ebXML Technical Reportsvii on Core Components)
- ◆ Agency Specific Data Dictionaries (such as the DDDS)
- ◆ Various commercial standards (ISO, UN/CEFACT, ANSI ASC X12, etc.)

With the exception of the FXR, the sources named do not provide readily reusable XML component names, however they do provide accepted, reusable data element definitions.

A fully documented XML Schema may be quite verbose. Such “semantic” Schemas can provide critical insight to analysts who desire to understand and interoperate by making use of the information in the Schema. However, they contain much more information than is really necessary for document structure validation. A “compact” Schema that is equivalent to the “semantic” Schema may be quickly built for validation purposes. Having both a full “semantic” Schema and a “compact” schema may be appropriate for activities wishing to provide extensive Schema annotations, or underlying type relationships while having a smaller schema used strictly for validation.

A schema guide document that fully defines and explains each component in the schema and the schema’s logical structure is an alternative to creating a fully documented semantic schema.

#### EXAMPLE

Appendix D provides an example that combines several of the concepts discussed so far, including capturing definitions and relationships.

### 4.3.1. Application Specific Metadata

#### GUIDANCE

Application specific metadata (such as SQL statements or API calls) that are of interest only to a single application SHALL NOT be included in instances or schemas.

#### EXPLANATION

Including application specific metadata in an instance unnecessarily clutters the document, increases bandwidth requirements, and is only useful to one application.

### 4.3.2. Capturing XML Component Definitions

#### GUIDANCE

Federal XML developers MUST—through XML comments, XML Schema annotations, a schema guide, or data dictionaries—document XML element and XML Schema type definitions. These definitions SHOULD be related to underlying ISO 11179 data element definitions.

Definitions SHOULD be brief and when possible be taken from existing standard data element definitions, such as those provided by the DDDS, ebXML Core Components, Federal Reference Data Sets, or other agency Standards.

---

Definitions **SHOULD** contain URL or other pointers to the definition's source, so that analysts can look up additional information.

Developers **MAY** extend the XML Schema annotation `<xsd:documentation>` tag by further marking up information provided with custom tags. No standards for this yet exist; however, the general guidelines of this document should be followed, and custom metadata tag names should follow the naming convention of the source data dictionary.

Developers **MAY** elect to publish schema documentation in a separate schema guide, however if this option is chosen, the schema must be URL accessible and referenced in the schema header.

#### EXPLANATION

Many activities in the government are rapidly developing schemas. Mandating that schema developers take the time to provide element and Schema type definitions will facilitate identifying commonalities and reusable components. Furthermore, it will start to enforce some rigor and thought in the creation of XML components as business and technical experts come together to create definitions for components and map their context-specific elements back to applicable government enterprise data standards.

Section 6 provides guidance on use of XML elements versus attributes. The WG recommends that attributes be minimized, and only used to provide supplementary metadata necessary to understand the business value of an XML element. By adopting this convention, and that of naming attributes in camel case according to ISO 11179 conventions, attributes will be reasonably self-explanatory and therefore not require a definition in most cases.

#### EXAMPLE

Appendix D provides a consolidated example of capturing definitions in XML Schema.

The example in Section 3.2 also illustrates these concepts.

### 4.3.3. Enumerations and Capturing Code Lists

#### GUIDANCE

Federal XML schema developers **SHOULD** use XML Schemas to express enumeration constraints on XML element and attribute values, when enumeration is considered essential to XML based data validation, such enumerated lists are of reasonable length, and the enumerations are considered stable (not likely to change frequently).



The decision to explicitly enumerate in a schema **SHOULD** be made by program managers based on the resulting size of the schema, bandwidth availability, and validation requirements.

Code lists from which enumerations are taken **SHOULD** be referenced by URI or other pointers so that analysts can lookup code values.

#### EXPLANATION

The government frequently represents data element values as codes rather than as free text. Codes are much easier for an application to understand and process because they are taken from a finite list of possible values, each with accepted semantics. Application developers create software to execute actions based on those code definitions and a specified set of business rules. XML can be used to exchange data that uses codes to abbreviate information, and the schema can be used to provide metadata about codes and their associated definitions (reference tables). Again, the way this is accomplished depends on the schema language chosen, with XML Schemas offering the most functionality. Capturing a reference to a list of valid codes and code values will greatly enhance implementations and allow future analysis to identify standard code reference tables. However, for code lists that historically change frequently, a URI pointer to the authoritative code list source is preferable.

#### EXAMPLE

A DTD example of an element taken from the MIL-STD-6040 (USMTF) with an enumerated set of possible values and an XML comment referencing the source of the code definitions. Note, the only way to express an enumeration in a DTD is via an attribute. In this example, the 'casualtyCategoryCode' attribute is better made an XML element (see Section 6). Use of the XML Schema language would have allowed expressing this enumeration as an element.

```
<!ELEMENT Casualty EMPTY>
<!ATTLIST Casualty casualtyCategoryCode (1 | 2 | 3 | 4 )
#REQUIRED>
<!--casualtyCategoryCode
Definition: A CATEGORY DENOTING THE EFFECT OF A CASUALTY ON A
UNIT'S PRIMARY AND/OR SECONDARY MISSION AREAS.
Source: MIL-STD-6040 Baseline 2001 FFIRN 1207 FUDN 0001-->
```



## Chapter 5 Document Annotation Conventions

---

### GUIDANCE

Federal XML schema developers **MUST** provide carefully thought out comments within schema and stylesheets, which provide basic information necessary to use and understand the document.

In general, Instances **SHOULD NOT** be documented, however, there may be situations in which it is appropriate.

### EXPLANATION

Just as it is good programming practice to document application code using a coding standard, it is important that XML schemas and stylesheets be well-documented in a standard fashion. The following paragraphs provide some recommended guidance.

The simplest way to express annotations is through the use of XML comments. Comments can be inserted anywhere in an XML document after the XML declaration.

XML Schema annotations provide a more flexible, extensible way to document Schemas as illustrated by many examples in this document.

## 5.1. DOCUMENT VERSIONING

### GUIDANCE

Version information for instances, schemas, and stylesheets **MUST** be available via document annotations (XML comments or Schema annotations).

### EXPLANATION

Having a schema's version number available to developers will assist in creating implementation that will maintain backward compatibility. Version information is also necessary for stylesheets in order to determine which version of a stylesheet correctly transforms an instance that conforms to a version of a schema.

---

## 5.1.1 Versioning DTDs

### GUIDANCE

DTD version information **SHOULD** be captured as an XML comment in the header of the DTD, and **MAY** be captured as a fixed attribute of the root element.

### EXPLANATION

DTDs offer two methods of documenting version numbers. The most straightforward is to put the DTD version number in the header XML comment. A second method is to declare a fixed schema version attribute to the XML Root Element. This will make the version generally available to applications via an API call.

### EXAMPLE

```
<?xml version='1.0' encoding='UTF-8' ?>
<!ELEMENT root EMPTY>
<!ATTLIST root schemaVersion CDATA #FIXED '1.0' >
```

Section 5.2 discusses providing version information in an XML comment in the header of a schema

## 5.1.2. Versioning XML Schemas

### GUIDANCE

XML Schemas **MUST** include a version using the 'version' attribute of the XML Schema specification.

### EXPLANATION

The schema header as discussed in Section 5.2 provides a uniform method to capture a consistent body of information required for a schema. However, developers can make version information more easily available to applications through the use of the version attribute as shown in the example.

**EXAMPLE**

Example of using Schema annotations to capture schema version information in an `<xsd:appInfo>` tag:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" ele-
  mentFormDefault="unqualified" version="1.0" >
  ...
</xsd:schema>
```

### 5.1.3. Versioning Stylesheets

**GUIDANCE**

A stylesheet **MUST** contain both its own version number (by using the built in version attribute of the XSLT language) and references to the name and versions of the schema that describe instances upon which the stylesheet performs correctly.

**EXPLANATION**

Tracking stylesheet versions is very important because a new version of a stylesheet may or may not correctly transform an instance conforming to an old version of a schema. Explicitly asserting which versions of a schema are supported in a stylesheet will alleviate potential interoperability issues as implementations evolve.

**EXAMPLE**

See example provided in Appendix E.

## 5.2 HEADERS

**GUIDANCE**

To promote interoperability, every schema, stylesheet, or instance **MUST** contain some basic metadata.

---

The metadata identified in Sections 5.2.1 and 5.2.2 SHOULD be provided.

[Ed. Note—There already exists a Federal electronic information exchange header standard. the metadata in 5.2.1 and 5.2.2 should be normalized with that existing standard.]

## 5.2.1 Schemas

- ◆ Schema Name
- ◆ Federal XML Registry Namespace(s)
- ◆ Functional Data Area
- ◆ URL to most current version
- ◆ For XML Schema, other Schemas imported or included to include FXR Namespace and version Schema file name, and URL.
- ◆ For DTD, external entities referenced to include FXR Namespace and version (in the case of parameter entities that are modular DTDs)
- ◆ A description of the purpose of the schema
- ◆ The name of the application or program of record that created and and/or manages the schema
- ◆ The version of the application or program of record
- ◆ A short description of the application interface that uses the description. A URL reference to a more detailed interface description may be provided
- ◆ Developer point of contact information to include activity, name and email
- ◆ A change history log that includes change number, version, date and change description.

## 5.2.2 Stylesheets

- ◆ Stylesheet Name
  - ◆ A list of schemas<sup>3</sup> and XSL processors against which the stylesheet has been tested.
  - ◆ The FXR Namespace where the stylesheet is registered
  - ◆ Functional Data Area of the application that makes use of the stylesheet
-

- ◆ URL to most current version
- ◆ Other stylesheets imported to include name and URL
- ◆ A description of the purpose and function of the stylesheet
- ◆ Application or program of record (with version) responsible for developing and maintaining the stylesheet
- ◆ Developer point of contact information to include activity, name and email
- ◆ A change history log that includes change number, version, date and change description.

### 5.2.3 Instances

- ◆ The name and URL of the schema that validates, and the stylesheet (if any) that correctly transforms it, if these are not specified already as part of the instance.

#### EXPLANATION

Other interested parties must be able to read a document and understand how to implement it or use information from it. Much of the information captured in a header XML comment can be better made available to applications through the use of fixed attributes or XML Schema annotations. However, having a consistent set of header information in a consistent location in an XML document will promote better configuration management and interoperability as methods for making this information available to applications are standardized. While examples are provided that show the above information captured in a single comment after the XML declaration, this should not discourage innovative developers from providing the same information as Schema annotations (possible with custom markup inside a `<xsd:documentation>` tag). Some information may also be captured as fixed attributes if developing in DTDs, as illustrated by previous examples.

#### EXAMPLE

Appendix E provides non-normative examples of document headers.





## Chapter 6 Attribute Versus Element Conventions

---

### GUIDANCE

The use of attributes **SHOULD** be carefully considered. Attributes **SHOULD** only be used to convey metadata that will not be parsed. Attributes, if used, **SHOULD** provide extra metadata required to better understand the business value of an element.

Some additional guidelines are:

- ◆ Attribute values **SHOULD** be short, preferably numbers or conforming to the XML Name Token convention. Attributes with long string values **SHOULD NOT** be created.
- ◆ Attributes **SHOULD** only be used to describe information units that cannot or will not be further extended or subdivided.
- ◆ Information specific to an application or database **MUST NOT** be expressed as values of attributes (see Section 4.3.1).
- ◆ Use attributes to provide metadata that describes the entire contents of an element. If the element has children, any attributes should be generally applicable to all the children.

### EXPLANATION

One of the key schema design decisions is whether to represent an information element as an XML element or attribute. Once an information element has been made an attribute, it cannot be extended further; for this reason and to promote better uniformity within the federal sector, we discourage the use of attributes.

### EXAMPLE

In Example 1, the code KTS (for knots) provides extra metadata required to understand the ‘business value’ of the element—600. It answers the question, “600 what?”

Example 1:

```
<TargetVelocityMeasure measureUnit-  
  Code="KTS">600</TargetVelocityMeasure>
```

---

In the other examples, we illustrate several appropriate ways of expressing coded values.

Examples of inappropriate attribute usage:

Example 2:

```
<TargetVelocity measure="600" measureUnitCode="KTS"/>
```

Example 3:

```
<CasualtyCategoryCode definition="[TRAINING ACTIVITY ONLY]  
EQUIPMENT CASUALTY EXISTS BUT WILL NOT IMPACT  
TRAINING WITHIN 30 DAYS."> 1</CasualtyCategoryCode>
```

In Example 3, both the business value and descriptive metadata are attribute values. This provides no mechanism for applications to determine which piece of information describes the other. In Example 4, the attribute is used to provide a verbose definition while the code value comprises the element contents; because XML parsers normalize white space in attribute values, attributes are inappropriate for use in this manner.

## Chapter 7 Federal XML Registry

---

### GUIDANCE

Draft Federal XML Tag Standards Policy **REQUIRES** all federal developers to reuse existing tags in the FXR if sufficient, or re-use commercial industry standard vocabularies if applicable, before developing their own.

It furthermore **REQUIRES** activities to register developed XML components with the FXR. When XML components from a commercial XML consortium are used, they also **MUST** be registered.

Developers **MUST** familiarize themselves with FXR site and the FXR order scheme. Each activity submitting a registration package to the registry is **REQUIRED** to do so via the registry manager.

### EXPLANATION

While this guidance provides many recommendations and examples of how to create more interoperable XML, the single biggest factors affecting interoperability are visibility and reuse. The intent of the FXR is to provide visibility into XML components that are being used throughout the government.

The WG is working with government representatives to develop specific guidance for developers as to the appropriate registration scheme with which they should register. Until this is promulgated, activities should study the registry site, and contact the FXR Information Manager for what appears to be the most appropriate place for registration. If unable to locate an appropriate registration scheme location, register with the 'To Be Determined' (TBD) registration scheme URN.

Pending resolution, a single application should submit its registration package to a single Federal Registry. In the case where an application's data crosses boundaries, request the FXR Information Manager to provide guidance.

### EXAMPLE

An example of a Federal Registration package was obtained from the FXR and is available for download the FXR information library.



# Appendix A

## ebXML and UN/CEFACT

---

### DESCRIPTION

ebXML was an 18-month international project sponsored jointly by OASIS<sup>viii</sup> and UN/CEFACT<sup>ix</sup> that ended in May, 2001 with the delivery of several specifications, technical reports and white papers available at [www.ebxml.org/specs](http://www.ebxml.org/specs). The ebXML deliverables define an architecture with two distinct views. The Functional Service View (FSV) defines

- ◆ functional capabilities,
- ◆ Business Service Interfaces, and
- ◆ protocols and Messaging Services.

In other words, the FSV consists of specifications and standards that describe how an ebXML-compliant system will physically operate to include interfaces, protocols, and registry/repository operations.

The Business Operational View (BOV) addresses:

- ◆ the semantics of business data in transactions and associated data interchanges
- ◆ the architecture for business transactions, including:
  - operational conventions,
  - agreements and arrangements, and
  - mutual obligations and requirements.

The BOV work focused on two areas. The first focus was on creating a methodology by which business processes can be modeled as orchestrated collaborations between business partners who exchange payloads of information (which may be XML documents). The UMM was chosen as the modeling methodology and a BPSS was created. Second, the BOV work focused on creating a methodology for creating two types of reusable components:

- ◆ process components which can be used to build complex business process models

- 
- ◆ information components which can be used to construct business documents as payloads of ebXML messages.

Some of the ebXML technical reports discuss the concept of core components as universal, domain independent information entities defined in an XML-neutral syntax. This is significant because the ebXML authors intentionally did not address how components (core and domain specific) should be used to produce business documents (in XML). According to the ebXML architecture, ebXML components exist as registered objects within an ebXML registry/repository system; the work of defining production rules for creating XML payloads from registry entries was deferred. This decision has drawn sharp criticism from some, however it makes sense. The ebXML strategy was to first address how to represent information (semantics and context) independently of how it is syntactically expressed as an XML document; consequently the ebXML technical reports on core components adopt the ISO 11179 naming convention for creation of dictionary entries for information entities. They do not specify how to create XML component names for schemas describing business documents containing payloads of information.

The ebXML deliverables provide a basis for future work required to make the vision of global interoperability a reality. OASIS and UN/CEFACT agreed to divide that work between them with OASIS assuming responsibility for the FSV aspects while UN/CEFACT took on the BOV portion. Since that time, UN/CEFACT has established the EWGx,

“...for the purpose of continuing the UN/CEFACT’s role in pioneering the development of XML standards for electronic business. The group was formed to build on the success of the earlier ebXML Joint Initiative between UN/CEFACT and OASIS, which delivered its first set of specifications in May 2001.”

One of the key deliverables of this group will be a final Core Component Specification that will combine and further refine the ebXML Core Component Technical Reportsxi.

The rest of the information presented in this appendix is taken from the deliverables of the ebXML project. These documents are works in progress. They may be useful in selecting data element and XML component names, however developers must and should expect the rules and specifications presented here to evolve rapidly.

## EBXML NAMING RULES

Quoted<sup>4</sup> from the ebXML Technical Architecture<sup>xii</sup>, Section 4.3 Design Conventions for ebXML Specifications:

“In order to enforce a consistent capitalization and naming convention across all ebXML specifications “Upper Camel Case” (UCC) and “Lower Camel Case” (LCC) Capitalization styles SHALL be used. UCC style capitalizes the first character of each word and compounds the name. LCC style capitalizes the first character of each word except the first word.

1. ebXML DTD, XML Schema and XML instance documents SHALL have the effect of producing ebXML XML instance documents such that:
  - Element names SHALL be in UCC convention (example: <UpperCamelCaseElement/>).
  - Attribute names SHALL be in LCC convention (example: <UpperCamelCaseElement lowerCamelCaseAttribute=“Whatever”/>)...
2. General rules for all names are:
  - Acronyms SHOULD be avoided, but in cases where they are used, the capitalization SHALL remain (example: XMLSignature).
  - Underscore ( \_ ), periods ( . ) and dashes ( - ) MUST NOT be used (don’t use: header.manifest, stock\_quote\_5, commercial-transaction, use HeaderManifest, stockQuote5, CommercialTransaction instead).”

The following are component-naming rules as quoted from the technical report, Naming Convention for Core Components<sup>xiii</sup> Section 5.2. They are based on the ISO 11179 Part 5 draft specification. In reading these understand that:

- ◆ Since the publication of this report, the UN/CEFACT has changed “representation type” to “representation term”:

---

<sup>4</sup> Copyright © ebXML 2001. All Rights Reserved.

“This document and translations of it MAY be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation MAY be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself MAY not be modified in any way, such as by removing the copyright notice or references to ebXML, UN/CEFACT, or OASIS, except as required to translate it into languages other than English.”

- 
- ◆ These rules apply to creation of ebXML “core components” but may be used in the creation of DON specific elements as well.
  - ◆ These initial rules are being incorporated into the UN/CEFACT’s Core Components Specification, which is being developed by the Core Component project team. Developers may choose to use the rules specified in the draft Core Components Specification rather than these. When that document reaches final status, this appendix will be updated accordingly. For now the May 2001 Core Component Naming Convention rules as specified by the initial ebXML project are provided for reference.
    - Rule 1: The Dictionary Entry Name shall be unique and shall consist of Object Class, a Property Term, and Representation Type.
    - Rule 2: The Object Class represents the logical data grouping (in a logical data model) to which a data element belongs” (ISO 11179). The Object Class is the part of a core component’s Dictionary Entry Name that represents an activity or object in a context.

An Object Class may be individual or aggregated from core components. It may be named by using more than one word.

- Rule 3: The Property Term shall represent the distinguishing characteristic of the business entity. The Property Term shall occur naturally in the definition.
- Rule 4: The Representation Type shall describe the form of the set of valid values for an information element<sup>5</sup>. It shall be one of the terms specified in the “list of Representation Types” as included in this document.

Note: If the Representation Type of an entry is “code” there is often a need for an additional entry for its textual representation. The Object Class and Property Term of such entries shall be the same.

(Example: “Car. Colour. Code” and “Car. Colour. Text”).

- Rule 5: A Dictionary Entry Name shall not contain consecutive redundant words. If the Property Term uses the same word as the Representation Type, this word shall be removed from the Property Term part of the Dictionary Entry Name.

---

<sup>5</sup> The term ‘information element’ is used generically in the same context as the term data element, and should not be confused with [XML Elements](#). An information element (or entity as ebXML refers to them) can be expressed as any of several XML components (XML Elements, attributes, or XML Schema types).



For example: If the Object Class is “goods”, the Property Term is “delivery date”, and Representation Type is “date”, the Dictionary Entry Name is ‘Goods. Delivery. Date’.

In adoption of this rule the Property Term “Identification” could be omitted if the Representation Type is “Identifier”.

For example: The identifier of a party (“Party. Identification. Identifier”) will be truncated to “Party. Identifier”.

- Rule 6: One and only one Property Term is normally present in a Dictionary Entry Name although there may be circumstances where no property term is included (e.g., Currency. Code).
- Rule 7: The Representation Type shall be present in a Dictionary Entry Name. It must not be truncated.
- Rule 8: To identify an object or a person by its name the Representation Type “name” shall be used.
- Rule 9: A Dictionary Entry Name and all its components shall be in singular form unless the concept itself is plural (e.g., goods).
- Rule 10: An Object Class as well as a Property Term may be composed of one or more words.
- Rule 11: The components of a Dictionary Entry Name shall be separated by dots followed by a space character. The words in multi-word Object Classes and multi-word Property Terms shall be separated by the space character. Every word shall start with a capital letter.
- Rule 12: Non-letter characters may only be used if required by language rules.
- Rule 13: Abbreviations, acronyms and initials shall not be used as part of a Dictionary Entry Name, except where they are used within business terms like real words (e.g., EAN.UCC global location number, DUNS number (see Section 3.2)).
- Rule 14: All accepted acronyms and abbreviations shall be included in an ebXML glossary [read, “...included in the element definition in the schema annotation, see Section 3.2“].

---

## REPRESENTATION TERMS

The following extract is provided from a 12 October 2001 draft of the UN/CEFACT core component specification. It is provided for information only: Here Representation Term is used versus the earlier Representation Type initially used in the ebXML technical reports.

*Table A-1. Representation Terms*

Representation Term	Definition	Links to Core Component Type
Amount	A number of monetary units specified in a currency where the unit of currency is explicit or implied.	Amount. Type
Code	A character string (letters, figures or symbols) that for brevity and/or language independence may be used to represent or replace a definitive value or text of an attribute. Codes usually are maintained in code lists per attribute type (e.g., colour).	Code. Type
Date	A day within a particular calendar year (ISO 8601).	Date Time. Type
Date Time	A particular point in the progression of time (ISO 8601).	Date Time. Type
Graphic	A diagram, graph, mathematical curve, or similar representation	Graphic. Type
Identifier	A character string used to identify and uniquely distinguish one instance of an object within an identification scheme from all other objects within the same scheme.  [Note: Type shall not be used when a person or an object is identified by its name. In this case the Representation Term "Name" shall be used.]	Identifier. Type
Indicator	A list of two, and only two, values that indicate a condition such as on/off; true/false etc. (synonym: "Boolean").	Indicator. Type
Measure	A numeric value determined by measuring an object. Measures are specified with a unit of measure. The applicable unit of measure is taken from UN/ECE Rec. 20.	Measure. Type
Name	A word or phrase that constitutes the distinctive designation of a person, place, thing, or concept.	Text. Type
Percent	A rate expressed in hundredths between two values that have the same unit of measure.	Numeric. Type

*Table A-1. Representation Terms*

Representation Term	Definition	Links to Core Component Type
Picture	A visual representation of a person, object, or scene	Picture. Type
Quantity	A number of non-monetary units. It is associated with the indication of objects. Quantities need to be specified with a unit of quantity.	Quantity. Type
Rate	A quantity or amount measured with respect to another measured quantity or amount, or a fixed or appropriate charge, cost or value (e.g., US Dollars per hour, US Dollars per EURO, kilometre per litre, etc).	Numeric. Type
Text	A character string generally in the form of words of a language.	Text. Type
Time	The time within a (not specified) day (ISO 8601).	Date Time. Type
Value	Numeric information that is assigned or is determined by calculation, counting or sequencing. It does not require a unit of quantity or a unit of measure	Numeric. Type

The following representation terms apply to aggregate Core Components or Core Component types.

*Table A-2. Other Representation Terms*

Representation Term	Definition	Links to Core Component Type
Details	The expression of the aggregation of Core Components to indicate higher levelled information entities	Not Applicable
Type	The expression of the aggregation of Core Components to indicate the aggregation of lower levelled information entities to become Core Component Types. All Core Component Types shall use this Representation Term	Not Applicable
Content	The actual content of an information entity. Content is the first information entity in a Core Component Type	Used with the content components of Core Component Types

The ebXML core components technical reports require that name of “aggregate information entities” use the special representation type, ‘details’. Federal XML developers may omit the term ‘details’ from the end of tag names when XML

---

element names are generated from the ISO 11179 name. For example, the ISO 11179 data element name 'Address. Details' would be represented in the XML instance as <Address>; in the XML Schema that describes the instance, the element Address would be created from the ISO 1179 derived Schema type AddressDetails.

The Representation Terms provided by ISO 11179 may not be adequate for a number of engineering, scientific and operational concepts. In these cases, temporary use of other term names, such as until the list of types is expanded, MAY be considered; however do this with caution.





# Appendix B

## Schema Development

---

### POSSIBLE SCHEMA DEVELOPMENT PROCEDURE SUMMARY

The following is presented as a possible procedure for developing schema. It does not represent the consensus of the WG; rather it is presented for your consideration and feedback. It is purely developmental; all or none of it may be useful.

#### STEPS

In creating XML components according to these conventions, try the following:

- Step 1. Analyze the business processes in which your application will exchange, use or store information. Understand who the consumers (both human and machine) of the information your application provides are. The WG recommends the use of the UMM and UML for this process, however any model that provides a basic understanding of how information will be exchanged across system boundaries (application to application, application to human, or human to application) can provide a basis for development as more rigorous modeling techniques, such as the UMM, are learned. The business process modeling should identify and name actors (persons, organizations, or systems) that participate in the process. The roles that each actor plays should also be identified and named. It is important to separate the name of the actor from the name of the role because often the same actor will participate in multiple roles within a process.
- Step 2. Based on the information exchange requirements identified in step 1, spend the time to model the data in each document that will be exchanged within the processes defined in step 1. The WG strongly recommends using the UML to conduct the modeling. Several efforts are underway to create production rules by which UML models can be directly used to generate XML documents. An excellent online resource is [xmlmodeling.com](http://xmlmodeling.com).
- Step 3. Look for previously developed XML components that can be reused, either in the FXR or schema developed by commercial consortia. (W3C Technical Reports provides references).
- Step 4. Create the ebXML/ISO 11179 compliant name and definition for each element identified in step 2 that will be used in an information exchange scenario.

---

Step 5. Identify extra metadata required to understand the business value of each element. This extra metadata may be expressed in either the schema or the instance as attributes (Section 6 provides detailed guidance).

Step 6. Analyze the information element. Ensure you have identified specific physical elements for each data item that will appear in the XML instance. This process will help the team identify underlying logical elements or generic physical elements that can be reused by declaring them as XML Schema Types or as abstract elements. This analysis should supplement the model you defined in step 2, and may require that you iterate through step 2 again. The UML static structure artifact is extremely useful here. Last, determine relationships between elements defined here and existing data models and definitions (such as the ebXML core components, the DDDS, and the Federal XML Registry).

Step 7. Identify any common business terms that are associated with the information elements defined in step 2.

Step 8. Create the schema 6.

- a. If creating the schema as a DTD, your choices are to make the model elements defined as an XML element or an attribute.
- b. If employing the XML Schema language, you have some extra choices in deciding how to express a model element. Model elements can be expressed
  - as types, which may be declared abstract,
  - as abstract XML elements, or
  - as (non-abstract) XML elements or attributes.
- c. One strategy for creating XML Schemas is as follows:
  - Create an underlying set of simple and complex XML Schema types describing base data types, reusable logical and generic physical elements.
  - Declare every model element that will appear in the XML instance as type that derives from the types declared previously.

---

<sup>6</sup> Up until now, we have not considered how we will express the information in XML. It is a good XML engineering practice to go through the process of defining and modeling information before the additional complications and design alternatives of XML are addressed. Trying to do both information modeling and XML design at the same time is confusing, and often, critical aspects of one or the other are missed.



- Create XML Schema types and attributes using the same name as the ISO 11179 named model elements.
  - Create XML element names that consist of an optional context term plus the ISO 11179 Object Class (plus property term if appropriate) plus representation term. For example <DoDMaterialItemIdentifier>, where the context term is “DoD” indicating that the element is specific to the Department of Defense.
  - For element names that also have common business terms with commonly used synonyms, such as NSN for National Stock Number, create a substitution group for the additional business terms and synonyms.
- d. Build the schema from the bottom-up and top-down.
- e. Register any newly created XML elements with the FXR.



# Appendix C

## Tools and References

---

### TOOLS

Tools for developing and employing XML in applications are flooding the market. However, most if not all of these tools are in early stages of development. In future revisions to this publication, recommendations will be provided as to tools that have either been used, evaluated or are known by reputation. Pros and cons of each will be presented in the case where they are known. Application developers who have used a particular tool may request that it be included in this list, provided it meets at least two of the following criteria:

- ◆ It is relatively mature or produced by an established vendor (such as IBM or Microsoft). A beta tool from Microsoft, or from IBM Alphaworks may be included, however a beta tool from CrazyXMLTools.com should not.
- ◆ It is a leader in a developing area, such as X2X's XLink processor. While still immature, it is currently one of the leaders in XLink processing software.
- ◆ It has been used by a federal activity and found to be useful and relatively free of bugs, or the bugs are well documented.
- ◆ It has been evaluated by a neutral third party (such as Forrester or the Gartner Group, or an established periodical) with favorable results.

Submit proposed tools to the editor using the format in Table C-1:

*C-1. Proposed Tools*

<i>Name and Link</i>	<i>Description</i>	<i>Pros</i>	<i>Cons</i>
XML, XSL and Schema Development			
XML Parsers and XSL Processors			
Databases			
“Servers”			

---

Miscellaneous			
---------------	--	--	--

A more complete list of available XML software is maintained at [www.xmlsoftware.com](http://www.xmlsoftware.com).

## PUBLICATIONS

Table C-2 lists publications that have been reviewed and found to be good reference material. The table presents several levels of reader and recommends appropriate reading for each.

## C-2. Reference Publications

Audience	Title	ISBN	Author(s)	Date
<b>Management/Business</b>	XML: A Manager's Guide	0-201-43335-4	Dick	2000
	ebXML: The New Global Standard for Doing Business on the Internet	0-735-71117-8	Kotok & Weber	2001
<b>Business/Technical</b>	XML in a Nutshell: A Desktop Quick Reference (Nutshell Handbook)	0-596-00058-8	Harold & Means	2001
	Metadata Solutions: Using Metamodels, Repositories, XML, and Enterprise Portals to Generate Information on Demand	0-201-71976-2	Tannenbaum	2001
	Modeling XML Applications with UML: Practical e-Business Applications	0-201-70915-5	Carlson	2001
<b>Technical</b>	The Wrox Professional XML Series		Wrox	
	Building B2B Applications with XML: A Resource Guide	0-471-40401-2	Fitzgerald	2001
	Java & XML, 2nd Edition: Solutions to Real-World Problems	0-596-00197-5	McLaughlin	2001
	SOAP: Cross Platform Internet Development Using XML	0-130-90763-4	Seely & Sharkey	2001
	Inside XSLT	0-735-71136-4	Holzner	2001
	XML Schema Development: An Object-Oriented Approach	0-672-32059-2	Brauer	2001

## INTERNET

Listed below are related Internet resources:

- 
- ◆ BizTalk <http://www.biztalk.org/home/default.asp>
  - ◆ ebXML <http://www.ebxml.org>
  - ◆ UN/CEFACT <http://www.UN/CEFACT.org/>
  - ◆ OASIS <http://www.oasis-open.org/>
  - ◆ Open Applications Group <http://www.openapplications.org/>
  - ◆ The Object Management Group [www.omg.org](http://www.omg.org)
  - ◆ RosettaNet  
<http://www.rosettanet.org/rosettanet/Rooms/DisplayPages/LayoutInitial>
  - ◆ Schema.net <http://www.schema.net>
  - ◆ W3C <http://www.w3.org>
  - ◆ XML.com <http://www.xml.com/>
  - ◆ The XML Cover Pages <http://www.oasis-open.org/cover/sgml-xml.html>
  - ◆ XML Software.com <http://www.xmlsoftware.com/>

# Appendix D

## Combined XML Schema Example

---

The following XML Schema is a combined example illustrating some of the guidance and concepts discussed in this document. The example is non-normative, and does not represent a consensus. It is provided for information only.

In this example, a tag from the FXR, <ACOUST\_SIGNA\_FREQ> is reused, but the principles of ISO 11179 and camel case are applied using the functionality of the XML Schema language to maintain interoperability.

[Ed. Note—This example currently uses business terms as element names, in lieu of the required ISO 11179 type names. This example will need to be updated prior to formal release of the document. This example is also DoD centric, and may not be easily understandable by other agencies. The example should be changed to a common business process such as purchase order that is non agency specific.]

The FXR defines a tag <ACOUST\_SIGNA\_FREQ> in the Tracks & Reports Namespace. An instance might look like this:

```
<ACOUST_SIGNA_FREQ>12.100</ACOUST_SIGNA_FREQ>  
  
Definition: ACOUSTIC SIGNATURE FREQ. THE FREQUENCY OF AN  
EMITTED ACOUSTIC SIGNAL TO THE NEAREST ONE THOUSANDTH HERTZ.  
  
Maximum Length: 10
```

You can view this tag definition at  
[http://diides.ncr.disa.mil/xmlreg/user/detail.cfm?ir\\_id=8358](http://diides.ncr.disa.mil/xmlreg/user/detail.cfm?ir_id=8358).

A possible XML Schema for this element:

```
<?xml version="1.0" encoding="UTF-8" ?>  
- <!--  
  edited with XML Spy v4.1 U (http://www.xmlspy.com) by Brian  
  Hopkins (Logicon/CISD)  
-->  
· <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" element-  
  FormDefault="qualified" attributeFormDefault="unqualified">  
· <xs:complexType name="MeasureType">
```

---

```

· <xs:annotation>
· <xs:documentation source="http://www.ebxml.org/specs/ccDICT.pdf">
· <ebXML>
<CoreComponent UID="core000152">Text. Type</CoreComponent>
  </ebXML>
  </xs:documentation>
  </xs:annotation>
· <xs:simpleContent>
· <xs:extension base="xs:decimal">
<xs:attribute name="measureUnitCode" type="xs:string" use="optional" de-
  fault="HZ"/>
  </xs:extension>
  </xs:simpleContent>
  </xs:complexType>
- <!--
  ISO 11179-derived type name
  -->
· <xs:complexType name="AcousticSignalFrequencyMeasure">
· <xs:annotation>
· <xs:documentation
  source="http://www.spawar.navy.mil/VPO/dataDictionary.doc#ID1234">
- <!--
  example source attribute points to notional data dictionary
  where the ISO name is defined. If the dictionary is readily
  URL accessible, then the <ISO11179Name> element below is
  redundant and may be omitted. Shown here for example.
  -->
· <ISO11179Name>
<ObjectClass>Acoustic Signal</ObjectClass>
<PropertyTerm>Frequency</PropertyTerm>
<RepresentationTerm>Measure</RepresentationTerm>
  </ISO11179Name>
  </xs:documentation>
· <xs:documentation
  source="http://diides.ncr.disa.mil/xmlreg/user/detail.cfm?ir_id=8358">
- <!--
  example source attribute points to Federal XML Registry
  Namespace from which element is derived

```



```

—>
· <COEXMLRegistry>
<Namespace prefix="TAR">Tracks and Reports</Namespace>
<TagName>ACOUST_SIGNA_FREQ</TagName>
<Definition>acoustic SIGNATURE FREQ. THE FREQUENCY OF AN
    EMITTED ACOUSTIC SIGNAL TO THE NEAREST ONE
    THOUSANDTH HERTZ.</Definition>
<RegistryID>8358</RegistryID>
</COEXMLRegistry>
</xs:documentation>
</xs:annotation>
· <xs:simpleContent>
· <xs:restriction base="MeasureType">
<xs:totalDigits value="10"/>
<xs:fractionDigits value="3"/>
<xs:pattern value="\d*.\d{3}"/>
<xs:attribute name="measureUnitCode" fixed="HZ"/>
</xs:restriction>
</xs:simpleContent>
- <!--
Annotations provide logical pedigree of element: Its ISO 11179
name and it mapping to an existing component already
registered with Federal XML Registry
—>
</xs:complexType>
- <!--
Element named after business term, "Acoustic Frequency"
—>
· <xs:element name="AcousticFrequency"
    type="AcousticSignalFrequencyMeasure">
· <xs:annotation>
<xs:documentation>Business Term</xs:documentation>
</xs:annotation>
</xs:element>
- <!--
COE element name made synonymous with camel case business term
through use of substitution group

```

```
—>
· <xs:element name="ACOUST_SIGNA_FREQ"
  type="AcousticSignalFrequencyMeasure" substitution-
  Group="AcousticFrequency">
· <xs:annotation>
<xs:documentation>COE Registered name</xs:documentation>
  </xs:annotation>
  </xs:element>
</xs:schema>
```

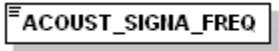
SCHEMA GUIDE FOR ACOUSTICSIGNALFREQUENCYMEASURE SCHEMA TYPE AND ASSOCIATED ELEMENTS

The Schema defines 5 XML Components: 2 types, 2 elements and 1 attribute.

Elements	Complex types
ACOUST_SIGNA_FREQ	AcousticSignalFrequencyMeasure
AcousticFrequency	MeasureType

The Federal XML Registry element name is defined as:

‘element ACOUST\_SIGNA\_FREQ’

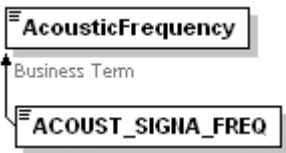
diagram	 Federal XML Registry name				
type	AcousticSignalFrequencyMeasure				
facets	totalDigits 10 fractionDigits 3 pattern \d*\d{3}				
attributes	Name	Type	Use	Default	Fixed
	measureUnit-Code				HZ

	measureUnit- Code	HZ
annotation	documentation	Federal XML Registry name
source	<pre> &lt;xs:element name="ACOUST_SIGNA_FREQ" type="AcousticSignalFrequencyMeasure" substitution- Group="AcousticFrequency"&gt;   &lt;xs:annotation&gt;     &lt;xs:documentation&gt;Federal XML Registry name&lt;/xs:documentation&gt;   &lt;/xs:annotation&gt; &lt;/xs:element&gt; </pre>	

Points to note:

- ◆ It is derived from a type ‘AcousticsSignalFrequencyMeasure’.
- ◆ It has several facets that restrict its domain.
- ◆ It has one attribute, ‘measureUnitCode’ that is fixed with a value of HZ.
- ◆ It is declared to be in the substitution group of the element ‘AcousticFrequency’.

‘element AcousticFrequency is a business term’ (notionally agreed to by all stakeholders within a COI).

diagram	 <p>Federal XML Registry name</p>				
type	AcousticSignalFrequencyMeasure				
facets	<p>totalDigits 10</p> <p>fraction-Digits 3</p> <p>pattern \d*.\d{3}</p>				
attributes	Name	Type	Use	Default	Fixed
	measureUnit-Code				HZ
annotation	documenta- tion	Business Term			
source	<pre> &lt;xs:element name="AcousticFrequency" type="AcousticSignalFrequencyMeasure"&gt;    &lt;xs:annotation&gt;      &lt;xs:documentation&gt;Business Term&lt;/xs:documentation&gt;    &lt;/xs:annotation&gt;  &lt;/xs:element&gt; </pre>				

Points to note:

- ◆ The business term has a synonym, ‘ACOUST\_SIGNA\_FREQ’, defined above and declared to be in the substitution group.
- ◆ It has the same attributes and facets as ‘ACOUST\_SIGNA\_FREQ’ because it derives from the same type.

‘complexType AcousticSignalFrequencyMeasure’ is the common Schema type from which both elements are derived.

diagram	<div> <div>AcousticSignalFrequencyMeasu...</div> <p>           &lt;!-- example source attribute points to notional data dictionary where the ISO name is defined. If the dictionary is readily URL accessible, then the &lt;ISO11179Name&gt; element below is redundant and may be omitted. Shown here for example.--&gt;         </p> <pre> &lt;ISO11179Name&gt;   □&lt;ObjectClass&gt;Acoustic     Signal&lt;/ObjectClass&gt;   □     &lt;PropertyTerm&gt;Frequency&lt;/PropertyTerm&gt;   □     &lt;RepresentationTerm&gt;Measure&lt;/Representati       onTerm&gt;   &lt;/ISO11179Name&gt; </pre> </div>				
type	restriction of MeasureType				
used by	<div> <div>elements</div> <div>ACOUST_SIGNA_FREQ</div> <div>AcousticFrequency</div> </div>				
facets	<div> <div>totalDigits</div> <div>10</div> </div> <div> <div>fraction-Digits</div> <div>3</div> </div> <div> <div>pattern</div> <div>\d*.\d{3}</div> </div>				
attrib-utes	Name	Type	Use	Default	Fixed
	measureUnit-Code	xs:string	optional		HZ
	<!--example source attribute points to notional data dictionary where the ISO name is defined. If the dictionary is readily URL accessible, then the <ISO11179Name> element below is redundant and may be omitted. Shown here for example.—>				

---

		<p>&lt;ISO11179Name&gt;</p> <p>    &lt;ObjectClass&gt;Acoustic Signal&lt;/ObjectClass&gt;</p> <p>    &lt;PropertyTerm&gt;Frequency&lt;/PropertyTerm&gt;</p> <p>    &lt;Representation- Term&gt;Measure&lt;/RepresentationTerm&gt;</p> <p>&lt;/ISO11179Name&gt;&lt;!--example source attribute points to Federal XML Registry Namespace from which element is derived--&gt;</p> <p>&lt;FederalXMLRegistry&gt;</p> <p>    &lt;Namespace prefix="TAR"&gt;Tracks and Re- ports&lt;/Namespace&gt;</p> <p>    &lt;Tag- Name&gt;ACOUST_SIGNAL_FREQ&lt;/TagName&gt;</p> <p>    &lt;Definition&gt;acoustic SIGNATURE FREQ. THE FREQUENCY OF AN EMITTED ACOUSTIC SIGNAL TO THE NEAREST ONE THOUSANDTH HERTZ.&lt;/Definition&gt;</p> <p>    &lt;RegistryID&gt;8358&lt;/RegistryID&gt;</p> <p>&lt;/FederalXMLRegistry&gt;</p>
--	--	---

source	<pre> &lt;xs:complexType name="AcousticSignalFrequencyMeasure"&gt;   &lt;xs:annotation&gt;     &lt;xs:documentation       source="http://www.spawar.navy.mil/VPO/dataDictionary.doc#ID1234 "&gt;       &lt;!--example source attribute points to notional data dictionary where the ISO       name is defined. If the dictionary is readily URL accessible, then the       &lt;ISO11179Name&gt; element below is redundant and may be omitted. Shown       here for example.--&gt;       &lt;ISO11179Name&gt;         &lt;ObjectClass&gt;Acoustic Signal&lt;/ObjectClass&gt;         &lt;PropertyTerm&gt;Frequency&lt;/PropertyTerm&gt;         &lt;RepresentationTerm&gt;Measure&lt;/RepresentationTerm&gt;       &lt;/ISO11179Name&gt;     &lt;/xs:documentation&gt;     &lt;xs:documentation       source="http://diides.ncr.disa.mil/xmlreg/user/detail.cfm?ir_id=8358"&gt;       &lt;!--example source attribute points to Federal XML Registry Namespace from       which element is derived--&gt;       &lt;FederalXMLRegistry&gt;         &lt;Namespace prefix="TAR"&gt;Tracks and Reports&lt;/Namespace&gt;         &lt;TagName&gt;ACOUST_SIGNA_FREQ&lt;/TagName&gt;         &lt;Definition&gt;acoustic SIGNATURE FREQ. THE FREQUENCY OF AN         EMITTED ACOUSTIC SIGNAL TO THE NEAREST ONE THOUSANDTH         HERTZ.&lt;/Definition&gt;         &lt;RegistryID&gt;8358&lt;/RegistryID&gt;       &lt;/FederalXMLRegistry&gt;     &lt;/xs:documentation&gt;   &lt;/xs:annotation&gt;   &lt;xs:simpleContent&gt;     &lt;xs:restriction base="MeasureType"&gt;       &lt;xs:totalDigits value="10"/&gt;       &lt;xs:fractionDigits value="3"/&gt;       &lt;xs:pattern value="d*.d{3}" /&gt;       &lt;xs:attribute name="measureUnitCode" fixed="HZ"/&gt;     &lt;/xs:restriction&gt;   &lt;/xs:simpleContent&gt; &lt;/complexType&gt; </pre>
--------	---

---



Points to note:

- ◆ The Type annotation provides the following:
  - ISO 11179 name parts. The source of this documentation is provided as a notional data dictionary referenced by URL and ID.
  - FXR Metadata including the definition.
- ◆ The domain restrictions are placed in the type versus at the element level.
- ◆ The attribute, 'measureUnitCode' has an optional value of HZ. It is set to fixed in the element declaration.
- ◆ The type is derived from an ebXML "core component"

'complexType MeasureType' is a complex type derived from an ebXML core component.

diagram	<div><div>MeasureType</div><div>&lt;ebXML&gt; □ &lt;CoreComponent   UID="core000152"&gt;Text,   Type&lt;/CoreComponent&gt; &lt;/ebXML&gt;</div></div>										
type	extension of xs:decimal										
used by	complex- AcousticSignalFre- Type quencyMeasure										
attributes	<table><tr><td>Name</td><td>Type</td><td>Use</td><td>Default</td><td>Fixed</td></tr><tr><td>measureUnit- Code</td><td>xs:string</td><td>optional</td><td>HZ</td><td></td></tr></table>	Name	Type	Use	Default	Fixed	measureUnit- Code	xs:string	optional	HZ	
Name	Type	Use	Default	Fixed							
measureUnit- Code	xs:string	optional	HZ								
annota- tion	<div>documenta- tion</div> <div>&lt;ebXML&gt;   &lt;CoreComponent UID="core000152"&gt;Text.   Type&lt;/CoreComponent&gt; &lt;/ebXML&gt;</div>										
source	<xs:complexType name="MeasureType"> <xs:annotation> <xs:documentation source="http://www.ebxml.org/specs/ccDICT.pdf"> <ebXML>										

	<pre> &lt;CoreComponent UID="core000152"&gt;Text. Type&lt;/CoreComponent&gt; &lt;/ebXML&gt;  &lt;/xs:documentation&gt;  &lt;/xs:annotation&gt;  &lt;xs:simpleContent&gt;   &lt;xs:extension base="xs:decimal"&gt;     &lt;xs:attribute name="measureUnitCode" type="xs:string" use="optional" default="HZ"/&gt;   &lt;/xs:extension&gt; &lt;/xs:simpleContent&gt; &lt;/xs:complexType&gt; </pre>
--	--

Points to note:

- ◆ The measureUnitCode attribute common to all other types and elements is defined only once—here.
- ◆ The type extends from the simpleType of decimal, again defined only once—here.
- ◆ The annotations provide mapping to the initial ebXML core component UID.

XML Schema documentation generated with XML Spy Schema Editor  
[www.xmlspy.com](http://www.xmlspy.com)

Some examples of XML instance fragments this document will validate:

```

<ACOUST_SIGNA_FREQ>100.000</ACOUST_SIGNA_FREQ>

    or

<ACOUST_SIGNA_FREQ measureUnit-
  Code="HZ">100.000</ACOUST_SIGNA_FREQ>

    or

<AcousticFrequency measureUnitCode="HZ">100.000</AcousticFrequency
>

```

# Appendix E

## Sample XML Document Headers

---

### Sample Schema Header

```
<?xml version="1.0" encoding="UTF-8">
```

<!--Schema/DTD Header \*\*\*\*\*-->

Schema Name: SPAWARVPO\$2-1\_FolderData\$1-1.xsd

*Federal XML Registry Information: TBD*

Functional Data Area: Administration

Current version available at (URL):  
[https://www.spawar.navy.mil/vpo/schemas/SPAWARVPO\\$2-1\\_FolderData\\$1-1.xsd](https://www.spawar.navy.mil/vpo/schemas/SPAWARVPO$2-1_FolderData$1-1.xsd)

Other Schemas Imported (XML Schema only):

\*\*\*\* Namespace Prefix: PER  
"http://diides.ncr.disa.mil/xmlreg/user/namespace\_list.cfm"

\*\*\*\* Schema File Name: BUPERSBUPERSOnLine\$3-0\_Document\$2-2.xsd

\*\*\*\* Available at URL: [www.bupers.navy.mil/bupersOnLine/schemas/](http://www.bupers.navy.mil/bupersOnLine/schemas/)

Other Schemas Included (XML Schema only): None

External DTDs Referenced (DTD only): n/a

\*\*\*\* Name: n/a

\*\*\*\* Available at (URL): n/a

Description: Provides information regarding the content of VPO folders such as content file names, file sizes, file owner, file status, and file access information.

Application: Virtual Program Office

Application Version: 2.1

Application Interface:

---

XML data is available from the VPO application via HTTP at <https://www.spawar.navy.mil/vpo/GetFolderInfo.asp>. Input queries via HTTP GET with query string format, "...?dir=directoryName". A complete interface description document is available at <https://www.spawar.navy.mil/vpo/interfaces/GetFolderInfo.txt>

Associated Stylesheet:

\*\*\*\* Name: SPAWARVPO\$2-1\_ViewFolderContents\$1-0.xsl

\*\*\*\* Available at (URL): <https://www.spawar.navy.mil/vpo/stylesheets/>

Developed by (Gov't Activity): SPAWAR 08

Point of Contact Name: Joe Smith

Point of Contact Email: [jsmith@spawar.navy.mil](mailto:jsmith@spawar.navy.mil)

Change History:

CHANGE # Version DATE DESCRIPTION OF CHANGE

0	1.0	15 Sep 2001	Initial release
1	1.1	30 Sep 2001	Updated to include file size information

\*\*\*\*\*

—>

This is a generic header that is provided in text-only, non-XML format. It can be used for either a DTD or XML Schema. A possibly more useful approach would be to markup header information using XML. The tags could be encapsulated by XML comment markup (<!--...--> or in the case of XML Schemas, included as an annotation following the XML Schema declaration.

Marking up header information could be very useful; for instance a large number of schemas could be automatically analyzed to determine which Federal XML Namespaces and Functional Data Areas they fell into. This would be a time consuming manual process otherwise.

The WG may work to standardize the tags and procedures for providing header information in XML markup. Until then, it is important to get the information somewhere in the document. Activities wishing to experiment with different strategies and techniques for providing header data are encouraged to do so and

report their findings to the WG. Consider the above example the minimum information we think will be required. Your input is encouraged.

Notes on header fields:

Header Item	Description
Schema Name:	The standard name of the schema file. See Document Naming Convention
Tested With:	List the name and version number of the XML processor(s) that have been tested and are known to correctly validate this schema.
Federal XML Registry Information:	Identify the Federal XML Registry Information related to elements from this schema by specifying the Federal XML Registry Information Prefix data. You can specify multiple Namespaces for XML Schemas that use tags from multiple namespaces. This is only possible through the use of XML Schemas because DTDs do not support XML Namespace prefixing.
Functional Data Area:	Indicate the Functional Data Area to which the application that uses this schema belongs.
Current version available at (URL):	If this schema is URL accessible, put the address here. We highly recommend that all schemas be available on-line to assist other activities desiring to interoperate.
Other Schemas Imported (XML Schema only):	The XML Schema language allows the reuse of existing XML Schema so that schemas can be modularized. The first way of doing this is via the XML Schema Import syntax.
The next three fields are repeatable	
**** Namespace Prefix and URL:	The XML Schema Import syntax is used when desiring to reuse a schema whose elements belong to a different XML Namespace from the elements into which the import is being conducted.
**** Schema File Name:	The standard name of the imported schema file. See Document Naming Convention
*** Available at (URL):	If this schema is URL accessible, put the address here. We highly recommend that all schemas be available on-line to assist other activities desiring to

Header Item	Description
	interoperate.
Other Schemas Included (XML Schema only):	The second way XML Schemas allow reuse of other schemas is through the XML Schema Include syntax. Includes can be used when the elements in the included schema belong to the same XML Name-space as the schema into which the include is occurring. A schema may both include and import.
The next two fields are repeatable	
**** Schema File Name:	The standard name of the imported schema file, see Document Naming Convention
*** Available at (URL):	If the schema file to be imported is URL accessible, put its address here. We highly recommend that all schemas be available on-line to assist other activities desiring to interoperate.
External DTDs Referenced (DTD only):	Information regarding any External Parameter Entity references are made to an external DTD. This approximates the modular design capability available in XML Schema.
The next two fields are repeatable	
**** Name:	The standard name of the DTD file, see Document Naming Convention
**** Available at(URL):	If this schema DTD is URL accessible, put its address here. We highly recommend that all schema DTDs be available on-line to assist other activities desiring to interoperate.
Description:	Plain text description of the type of information described by the schema.
Application:	The name of the application which produces XML documents that validate to this schema.
Application Version:	The version (major.minor) of the application that produces this schema.
Application Interface:	A plain text descriptive summary of how other applications interface with this application (e.g., via HTTP, using query parameters passed via HTTP POST or GET). Examples of query name/value pairs may be provided. If SOAP is used, you should provide a brief description of the method calls and

Header Item	Description
	parameters. A good XML engineering practice is to completely document your application interface; if you have done so, reference that documentation here. Making the interface specification available via a (secure) URL will assist other developers in interoperating.
Associated Stylesheet:	If a stylesheet is available to render instances that validate to this schema, provide information here.
**** Name:	The standard name of the stylesheet file, see Document Naming Convention
**** Available at (URL)	If the stylesheet is URL accessible, put the its address here. We highly recommend that all stylesheets be available on-line to assist other activities desiring to interoperate.
Developed by (Gov't Activity):	Government Activity and Office code.
Point of Contact Name: Joe Smith	Name of person to contact with questionions regarding the schema.
Change History:	The following fields provide an audit trail of changes.
CHANGE #	Keep a sequentially numbered list of changes.
Version	You should also assign Major and minor version numbers.
DATE	Date implemented
DESCRIPTION OF CHANGE	Plain text description.

## Sample Stylesheet Header

This sample stylesheet header is the similar to the schema header with the addition of information regarding the version of a schema from which the stylesheet is written, and the removal of non-applicable items.

---

<?xml version="1.0">

<!--Stylesheet Header \*\*\*\*\*

Stylesheet Name: SPAWARVPO\$2-1\_ViewFolderData\$1-1.xsl

Tested to:

\*\*\*\* Schema Name: SPAWARVPO\$2-1\_FolderData\$1-1.xsd

\*\*\*\* Schema Version: 1.1

\*\*\*\* *XSL Processors: MSXML 3.0, XALAN 1.2.2*

Federal XML Namespace: TBD

Functional Data Area: Administration

Current version available at (URL): <https://www.spawar.navy.mil/vpo/stylsheets/>

Other Stylsheets Imported:

\*\*\*\* File Name: BUPERSBUPERSONLine\$3\_Document\$2-2.xsl

\*\*\*\* Available at URL: [www.bupers.navy.mil/bupersOnLine/stylsheets/](http://www.bupers.navy.mil/bupersOnLine/stylsheets/)

Description: XSLT compliant stylesheet renders folder contents as an HTML table

Application: Virtual Program Office

Application Version: 2.1

Developed by (Gov't Activity): SPAWAR 08

Point of Contact Name: Joe Smith

Point of Contact Email: [jsmith@spawar.navy.mil](mailto:jsmith@spawar.navy.mil)

Change History:

CHANGE # Version DATE DESCRIPTION OF CHANGE

|   |     |             |  |
|---|-----|-------------|--|
| 0 | 1.0 | 15 Sep 2001 | Initial release                          |
| 1 | 1.1 | 30 Sep 2001 | Updated to include file size information |



\*\*\*\*\*

→

---

The following notes indicate differences between the stylesheet and schema header only.

| Header Item                        | Description   |
|------------------------------------|---|
| Stylesheet Name:                   | The standard name of the schemastylesheet file. See Document Naming Convention                                      |
| Tested to:                         | Information regarding the specific schema and software with which this stylesheet has been tested.                  |
| **** Schema Name:                  | Name(s) of the schemas with which this stylesheet has been tested.  |
| **** Schema Version:               | Version(s) of the schemas with which this stylesheet has been tested.   |
| **** XSL Processors:               | Name(s) of the XSL processors with which this stylesheet has been tested.   |
| Other Stylesheets Imported         | Stylesheets—like schemas—can be constructed modularly. Provide information here regarding other reused stylesheets. |
| The next two fields are repeatable |   |
| **** File Name:                    | The standard name of the file. See Document Naming Convention   |
| *** Available at (URL):            | If this Stylesheet is URL accessible, put its address here.   |

## Sample Instance header

It is important that XML documents include some basic information. Most of the needed information can be gleaned from the header data provided by the schema that describes the document and the stylesheet(s) that transform or render it. The XML specifications provide syntax for pointing to schemas and stylesheets at the beginning of an XML document. In cases where validation against a schema and/or transformation with a stylesheet is not required, it is still desirable to provide references to schemas and stylesheets if available. Consider this example:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<!--
```

Schema and Stylesheet Reference Data:

```
stylesheet type = xslt
```

```
url = http://spawar.navy.mil/stylesheets/SPAWARVPO$2-  
1_ViewFolderData$1-1.xsl
```

```
version = 1.1
```

```
schema type = XML Schema (W3C)
```

```
url = http://spawar.navy.mil/schemas/SPAWARVPOV2-  
1FolderDataV1-1.xsd
```

```
version = 1.1
```

```
-->
```

```
</root/>
```



# Appendix F

## Points of Contact

---

Federal XML WG Government Lead:

(To Be Determined) Joint Federal CIO Council XML Working Group Lead and Editor:



# Appendix G

## Glossary and Acronyms

---

The following draft glossary is provided in advance of the WG's future XML Glossary deliverable. It does not reflect the consensus of the WG. These items are provided for information only.

### TERMS

**Abstract**—In the context of an XML Schema, an XML element or Schema type may be declared abstract, meaning that it may not be used directly. An abstract element may not be directly used in an instance, but must have a non-abstract element in its substitution group (e.g., an abstract element 'Address', which defines the contents of an address). A non-abstract 'HomeAddress' element that is substitutable for 'Address' can be used as an XML element. The 'HomeAddress' structure reuses the previously defined 'Address' contents, but the tag provides a specific context. Schema types may also be declared abstract. Similar to abstract elements, abstract types may not be directly used to reference elements, but must have a non-abstract type that extends/restricts from it. The non-abstract type can then be used to reference XML elements. The concept of abstractness is taken from object-oriented programming, where an abstract class may be defined; requiring sub-typing prior to instantiation.

**Binding** - A term frequently used in reference to XML applications taken from the field of computer science. In the context of applications that have a public interface that communicates in XML (such as the case with a web service), binding refers to the information required and the process by which an external source connects to, and interacts with it to get data in XML. Binding can also refer to the process and application required to connect a software module (e.g. a Java class, or COM object) to a public XML interface, or the way in which the public XML is related to an underlying data source (such as a relational database).

**BPSS** - The Business Process Specification Schema was developed as part of the ebXML project as a schema for describing a business process in an XML instance. It may be created from UML models of business processes developed according to the UMM as described in the technical report, Business Process and Business Information Analysis Overview v1.0xiv. The BPSS is available in either DTD format xv or XML Schema (Candidate Recommendation) formatxvi.

**Business Term** - The ebXML specifications refers to a business term as a commonly used term referencing a commonly understood concept within a specific domain. To enhance understandability, it is appropriate to use business terms as

---

XML Element names (when they exist), rather than the often esoteric ISO 11179 syntax.

**Camel Case**—A convention in which names of elements and attributes are all lower case with the exception of the beginning of a new word, which is in upper-case. ebXML differentiates between upper camel case—where the first letter of the name is also capitalized—and lower camel case, where it is not. Example of an upper camel case name: UpperCamelCase. A lower or just camel case name: lowerCamelCase. Camel case is emerging as the industry norm for XML element naming. ebXML specifies elements to be in upper and attributes to be in lower camel case, while BizTalk, RosettaNet, and Oasis use straight camel case for both elements and attributes.

**CSS - Cascading Style Sheets.** A set of W3C recommendations for styling HTML and XML documents based on the application of formatting instructions in a linear, cascading fashion. CSS is an alternative to styling XML with XSL, but CSS does not have the transformational component of XSLT.

**Class**—A software component that provides instructions for the creation of an object. Applications are said to create instances of a class (“objects”) through a process referred to as instantiation. In the context of XML, a schema is a “class” that describes XML instances (data “objects”).

**Federal XML Registry**—The Federal XML Registry provides a baseline set of XML components developed through coordination and approval among the federal XML community. The Registry allows you to browse, search, and retrieve data that satisfy your requirements.” Draft Federal XML policy requires that all activities developing XML register components be developed with the appropriate Federal XML Namespace.

**Federal XML Registry Information**—The Federal XML Registry is divided into “Namespaces”. A Namespace is a collection of people, agencies, activities, and system builders who share an interest in a particular problem domain or practical application. This implies a common worldview as well as common abstractions, common data representations, and common metadata. The Federal XML Registry allows Namespaces to publish their existence and their available information resources so that outsiders may discover them and assess whether they want to share.” Federal XML Registry Information is an extension of the Federal XML Registry concept.

**Federal XML Registry Information Manager**—Each Federal XML Namespace has a central activity responsible for it. The individual responsible for coordinating and administering the Namespace is the Registry Information Manager. Point of contact information for the Registry Information Managers is available by clicking on the Namespace hyperlinksxvii on the registry’s web site.



**Federal XML Registry Information Prefix**—Each Federal XML Namespace has been assigned a three-letter prefix that may be used as XML Namespace qualifiers in XML instances and Schemas.

**Federal XML Registration Package**—Activities developing XML within the government are required to submit a specially formatted package of information to the FXR containing metadata about the components registered. Information about how and what to register can be found here<sup>xviii</sup>.

**COM Object**—The Common Object Model is a Microsoft sponsored interface specification for creating interoperable software components. Distributed COM or DCOM is Microsoft's COM interface standard for distributed computing, i.e., where an "application" consists of software "objects" distributed across nodes of a network. DCOM is similar to the Java based EJB specification, but works only for Microsoft operating systems. DCOM objects can communicate via TCP/IP and their own proprietary messaging framework (Windows Distributed iNternet Architecture or DNA). Alternatively, COM objects can communicate with other non-COM/non-Windows objects such as Java Classes or EJBs via XML and SOAP.

**CORBA**—Common Object Request Broker Architecture. CORBA is a framework created by the Object Management Group<sup>xix</sup> (OMG) to facilitate platform/operating system/programming language-neutral distributed computing. Software components or "objects" interact in client-server relationships, with an Object Request Broker (ORB) software component acting as intermediary. Via the IIOP, CORBA-based distributed applications can operate across the Internet. Most commonly used with the Java language, though CORBA is language independent.

**Core Components**—One goal of the ebXML effort is to define a set of universal core components that are contextually neutral and can be used across all domains to express semantics of common business concepts. Core components may be information entities, defined in the ebXML Core Component Dictionary technical reports, or process components discussed in the ebXML Business Process technical reports. Note that the core component technical reports do not address how an information component will be expressed in XML. This was an intentional omission on the part of ebXML. It was felt that prior to defining rules for creation of XML, a necessary first step was to create a schema neutral standard for defining components in business terminology. The work of defining how core components map to XML will be undertaken by the Core Component Project Team<sup>xx</sup> of the UN/CEFACT sponsored **EWG**.

**DDDS**—The Defense Data Dictionary System<sup>xxi</sup> defines standard data elements per the DoD 8320 series of documents<sup>xxii</sup>. The DDDS provides definitions of Standard Data Elements (SDEs) from core data models across all DoD data domains. The DDDS elements are mainly logical in nature, and may be used to ex-

---

press logical, semantic relationships between XML elements. XML Schema types may be used to express relationships to DDS standard data elements.

**Document Type Declaration**—A declaration at the beginning of an XML document indicating a DTD to which the instance must conform.

**DOM** - The Document Object Model. The set of W3C DOM recommendations form application interface descriptions (APIs) for expressing the contents of XML or HTML “documents” as hierarchical tree-like models of information with data forming the “leaves” of the tree. XML Processors that implement the DOM interface parse an entire XML document, creating a data tree in memory. Applications that call a DOM parser access data from the XML object tree through a set of programmatic instructions defined by the specifications. The instructions allow applications to “walk the document tree”, searching for elements and attributes that meet query criteria (XPath expressions). Results are returned to the calling application and assigned to application variables for further processing.

**DTD** - Document Type Definition. A schema syntax that is part of the XML 1.0 specification and derived from SGML.

**EJB**—Enterprise Java Beans. EJB is an interface specification which a Java class may implement. Software objects that implement the EJB interface may interoperate in an enterprise (distributed) environment—even across the Internet via TCP/IP and the CORBA IIOP. In this fashion, an “application” may consist of a number of independent software components (“objects”) that are physically separated at different nodes of a network, but functioning together as a single application similar to the Microsoft (D)COM specification.

**Entity**—In the context of a DTD, an entity is a declarative construct defining or referencing text, or a binary file. Entities are defined in the DTD, and referenced elsewhere in the DTD (parameter entity) or in the body of the XML (general entity). A validating parser encountering a reference to a previously defined entity during the validation process will insert the entity’s value in place of the entity reference. Internal entities are declared in the DTD and may be general or parameter. External entities point to an external file containing the entity declaration via URI reference; they also may be internal or external. A parsed entity is some form of encoded text and is therefore processed by a parser. An unparsed entity is a reference to a binary file that will not be parsed. Unparsed entities are always external. Through entities, DTDs may declare a common construct once, and reuse it many times throughout the DTD or in the instance. A common use for parameter entities is to declare a common set of attributes in the DTD. Assigning the attributes to an element only requires a reference to the parameter entity, versus retyping the entire attribute list many times. A second use of external unparsed general entities is to make reference to a binary file (such as an image or sound file) within an XML instance.

EDI—Electronic Data Interchange. A term referring to the conduct of eBusiness through the exchange of electronic messages. Two message standards exist as rigorously defined sets and segments, one maintained by the U.S. led ANSI X12 body, and the second led by UN/EDIFACT.

Fatal Error - [From the XML 1.0 specification] “An error which a conforming XML parser must detect and report to the application. After encountering a fatal error, the parser may continue processing the data to search for further errors and may report such errors to the application. In order to support correction of errors, the processor may make unprocessed data from the document (with intermingled character data and markup) available to the application. Once a fatal error is detected, however, the processor must not continue normal processing (i.e., it must not continue to pass character data and information about the document’s logical structure to the application in the normal way).” In other words, upon detecting a fatal error (such as a well-formedness violation), the parser is unable to provide information from the XML document to the calling application such that the application may continue functioning normally.

HTML - Hypertext Markup Language<sup>xiv</sup>

Interface—The process by which a software application interacts with other software or users. In object-oriented programming an (software) “object’s” interface is often described separately from the internal logic in a process known as “encapsulation”. Essentially the interface encapsulates and hides the internal logic. This allows flexibility to change and improve object code without affecting other objects. An interface description is made public so other objects/applications know how to interact. Software is said to “implement” an interface if it conforms to the behavior as defined in an interface description. The Object Management Group (OMG) has defined a formal syntax (language) for defining interfaces in a programming language neutral fashion. This is called the OMG Interface Description Language<sup>xv</sup> (OMG IDL). This IDL is used to define interface specifications such as the DOM API and CORBA. For developers implementing public XML interfaces, it is a good idea to document exactly how other applications connect, query, and receive (i.e. bind to) your application; while it is not necessary to go to the trouble of writing a formal IDL interface description, some kind of formal document will greatly aid other applications desiring to share data.

IIOP—Internet Inter-Orb Protocol. A TCP/IP based protocol that facilitates communication between CORBA ORBs. Via IIOP, CORBA client objects at one location on the Internet can communicate with CORBA server objects at another node and vice versa.

ISO 11179 - Information Technology - Specification and Standardization of Data Elements is a 6-part ISO standard providing a framework and methodologies for developing, documenting, and registering standard data elements. Of interest to XML developers is Part 5: Naming And Identification Principles For Data Elements upon which the ebXML naming convention is based. The specifications are

---

available from the ISO Store<sup>xxvi</sup> under section 35.040 - Character Sets And Information Coding for a small fee.

Markup - Special characters used by Markup Languages (SGML, XML, HTML) to differentiate data from metadata. SGML allows document authors the flexibility of specifying which characters are used for markup, where as in XML the markup characters are fixed. Markup characters may not be used in data text (unless special precautions are taken). In the tags definition example, the markup characters are '<' (greater than), '>' (less than), and '/' (forward slash). The XML specification<sup>xxvii</sup> defines start tag markup as opening with a '<' and ending with a '>'. It specifies that end tag markup opens with '</' and ends with '>'.

Metadata - Data about data. For example, for the data '3000N', the metadata might be 'latitude'. Markup languages such as SGML and XML encapsulate data with tags that contain text describing the metadata. See the example provided in the tags definition.

Normative—A term frequently used by software specifications to mean required, mandatory, or representing the only way to accomplish something. Often references are cited as normative, meaning that the requirements of these references apply to the document being read, or as non-normative, meaning they are provided as information only.

Object—A term used frequently in relation to XML and computer science. Strictly speaking, an object is a run-time software construct that resides in the Random Access Memory (RAM) of the host computer. Objects are created by applications from code that defines the object's behavior; this code is called a class. In object-oriented programs, objects interact with other objects to create the behavior of the application. An object's behavior is described by an Interface consisting of methods and properties. A method can be thought of as a behavior of the object that can be triggered by calling it and optionally passing parameters. For instance, the object 'myAccount' might have the method 'getBalance(accountNumber)'. Object oriented languages use the 'dot' notation to refer to objects and methods. From the previous example, 'currentBalance == myAccount.getBalance(accountNumber)' is a code snippet that assigns to the 'currentBalance' variable the balance returned from the 'myAccount' object when the 'getBalance()' method is called by passing in the 'accountNumber' variable. Object properties are similar to methods, but instead of calling a behavior, a property call to an object returns a previously set value of the property. Returning to the example, 'myName == myAccount.accountOwner' sets the 'myName' variable equal to the 'accountOwner' property of the 'myAccount' object, conversely 'myAccount.accountOwner == myName' sets the 'accountOwner' property of the 'myAccount' object to the value of the 'myName' variable. XML that has been parsed by an XML processor implementing the DOM API is transformed into a set of objects that may be used by the calling application to extract data from the XML. Also, an application may construct a DOM tree of objects in memory then transmit the data to another application or object as a textually encoded string of

XML. The receiving object then accesses the data via the DOM or SAX APIs. Since the XML format is neutral, a COM object created by a Windows application may interact with an EJB object running on a Unix platform for true cross-platform, language-independent distributed computing.

Payload (XML)—Protocols and frameworks such as SOAP, BizTalk, and ebXML use XML to markup message header information necessary for binding, reliable messaging, and security. The term ‘payload’ refers to the XML being transmitted that contains the actual business information being communicated.

Public (XML) Interface—XML may be employed internal to an application or it may be used to communicate information to other systems outside the originating applications environment. The term ‘Public Interface’ refers to XML used by an application or set of homogeneous applications to communicate with other applications across system boundaries. Federal policy for registration of XML components applies to public interfaces; these policies are not intended to restrict the use of XML internal to systems; in fact, it is recommended that applications separate internal XML grammars processed by application code from that used for external communications.

Qualified (elements and attributes)—The practice of prefixing an element or an attribute with an XML Namespace qualifier in accordance with the Namespaces in XML<sup>xxviii</sup> W3C Recommendation. This allows two elements with the same name to be disambiguated by an XML processor.

Regular Expression—A language for defining patterns in strings and numbers. The XML Schema language allows elements and attributes to be constrained by regular expressions to provide a precise description of the range of possible values. For instance, an element of type=‘integer’ could be further constrained to be only a 3-digit integer by the regular expression ‘/d{3}’.

Rendering (XML) - XML is not easily useable to readers in its native format and should be transformed for presentation (rendered), rendered for presentation, either by a CSS, XSLT (to well-formed HTML) for browser viewing, or by XSL-FO into a format for viewing by other presentation applications (e.g. into Adobe Acrobat.pdf, or MS Word.doc files.) Note: It is a common assumption that all XML must be rendered (by a stylesheet) to be useful, and that therefore all XML must have a stylesheet. This is a mistake; XML data can be used by an application via an API and never get rendered at all.

SAX - Simple API for XML. SAX<sup>xxix</sup> is an open-source interface for accessing information from XML documents. SAX parsers process a document, triggering events in the calling application corresponding to the parser encountering opening tags, closing tags and character data. Accessing XML data via SAX is very quick and places fewer demands on system resources than DOM, however once processed, a document must be re-parsed if the required information was not retained initially. This can be conceptualized as “serial” access to the information.

---

Schema - Within the context of XML, a document describing a set of XML Instances. Schemas may be expressed in a number of different languages. Most familiar is the Document Type Definition (DTD) syntax described in the XML 1.0 specification. Schemas provide the rules against which a validating parser validates an instance of XML.

SGML - The Standard Generalized Markup Language [ISO 8879xxx]. SGML is the parent of both HTML and XML.

SOAP - “SOAP is the Simple Object Access Protocol—a way to create widely distributed, complex computing environments that run over the Internet using existing Internet infrastructure. SOAP is about applications communicating directly with each other over the Internet in a very rich way.” [MS] “SOAP is a protocol specification for invoking methods on servers, services, components, and objects. SOAP codifies the existing practice of using XML and HTTP as a method invocation mechanism. The SOAP specification mandates a small number of HTTP headers that facilitate firewall/proxy filtering. The SOAP specification also mandates an XML vocabulary that is used for representing method parameters, return values, and exceptions.” [DevelopMentor]. Taken from the XML Cover Pagesxxxi. The current SOAP 1.1 specificationxxxii is a W3C Note; SOAP 1.2xxxiii is going through the W3C consensus processxxxiv and was published as a first working draft in July 2001.

SQL - Structured Query Language - A language for querying, writing to, and constructing relational databases. Many versions of SQL exist; meaning that an SQL query that works for one database will not necessarily work against another.

SDE—Standard Data Element as defined by the DoD 8320 series and used in the DDDS.

Stylesheet - A generic term that may refer to an XSL Stylesheet or a CSS. Often the term is used to reference XSL Stylesheets implicitly, however this is not technically correct, as a stylesheet may be CSS conformant, and have nothing to do with XML whatsoever. The primary function of a stylesheet is to render XML to a presentation format. However, XSLT can transform one XML instance into another different instance. Application of a stylesheet by an XSL processor to an XML document for the purpose of creating another XML document (i.e. an XML to XML transformation) does not render a presentation format at all. More simply, applying a stylesheet to XML doesn’t imply that the output is ready for viewing; you have to understand what the stylesheet is doing.

Substitution Group—In the context of XML Schemas, a substitution group may be declared for an element to define a synonymous group of tag names. A top-level element is declared, then other elements are declared with an attribute indicating they belong in the substitution group of the top element. Different elements do not necessarily have to have the same structures—used in this fashion they are functionally similar to a group of optional elements where only one may be chosen.

The top-level element may be declared abstract. In this case the top level element may not be used but can serve as a generic model for non-abstract elements in the substitution group. This is similar and somewhat redundant of the functionality provided by XML Schema types.

Throw (an error)—A term adopted from the Java language to indicate that a processing error has occurred. Conceptually, Java “throws” the error to an error-handling object, which “catches” it, or may “throw” it to another object, and so on.

UID—Unique Identifier. A generic term used to indicate that an object or item has a string or number that identifies it uniquely within a specific context or environment. Universally Unique Identifiers (UUIDs) and Globally Unique Identifiers (GUIDs) are special identifiers that are guaranteed universal uniqueness via an identifier assignment algorithm.

UML - The Unified Modeling Language<sup>xxxv</sup> defines a standard language and graphical notation for creating models of business and technical systems. UML is not only for programmers. It defines several model types that span a range from functional requirements definition and activity work-flow (business process) models to logical and physical software design and deployment. The UML has over the last few years become the lingua franca for business and technical stakeholders to communicate and develop IT systems. Through the UMM, UML has been adopted by UN/CEFACT and ebXML as the modeling language of choice.

UMM - The Unified Modeling Methodology<sup>xxxvi</sup> is a product of UN/CEFACT, and describes the UN/CEFACT-recommended methodology for modeling business processes to support the development of the next generation EDI. It is based upon the Rational Unified Process<sup>xxxvii</sup>, and uses the UML as its modeling language. In the UMM, business processes are modeled by deconstructing them into a series of document exchanges which are orchestrated to form a complex process. The ebXML Technical Report Business Process and Business Information Analysis Overview v1.0 further develops the UMM. The ebXML Business Process Specification Schema v1.01 (BPSS) provides a schema in the form of a DTD for specifying business processes as an XML instance. It may be developed as part of a UMM modeling process.

URL/URI/URN—Uniform Resource Locators, Uniform Resource Indicators, and Uniform Resource Names are different, related methods of uniformly referencing resources across networked environments. A W3C Note explains the difference<sup>xxxviii</sup>.

Valid (XML) - An XML instance (document) whose structure has been verified in conformance to a schema by a validating parser. Note that an XML instance must be well-formed to be valid, but it does not need to be valid to be well-formed. This is because a parser will always check well-formedness constraints but will only check validation constraints if it is a validating parser.

---

**Validating Parser** - An XML parser that enforces validity constraints by comparing the structure and syntax of an XML instance to the rules specified in a schema. Not all parsers are validating parsers, and validating parsers enforce validation according to specific schema languages. Most validating parsers are capable of enforcing validity against a DTD, while some can enforce validation rules described in other schema languages.

**W3C** - The World Wide Web Consortium<sup>xxxix</sup> was created in October 1994 to lead the World Wide Web to its full potential by developing common protocols that promote its evolution and ensure its interoperability. W3C has more than 500 Member organizations<sup>xl</sup> from around the world and has earned international recognition for its contributions to the growth of the Web.

**W3C Recommendation** - A work that represents consensus<sup>xli</sup> within W3C and has the Director's stamp of approval. W3C considers that the ideas or technology specified by a Recommendation are appropriate for widespread deployment and promote W3C's mission.

**W3C Note**—A W3C Note is a publication of a member idea. Notes do not go through the consensus process. They represent the ideas of a single (group of) W3C member(s).

**(W3C) XML Schema** - A schema written in accordance with the W3C XML Schema language. [From the W3C Schema<sup>xlii</sup> page] “XML Schemas express shared vocabularies and allow machines to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents. The XML Activity Statement<sup>xliii</sup> explains the W3C's work on this topic in more detail.” The W3C XML Schema language is described in three recommendations: XML Schema Part 0: Primer<sup>xliv</sup>, XML Schema Part 1: Structures<sup>lv</sup>, and XML Schema Part 2: Datatypes<sup>lvi</sup>. In the Federal XML Developer's Guide (this document), the term XML Schema is used in reference to a W3C XML Schema language-compliant schema.

**Web-service**—A generic term used to refer to the use of Hypertext Transfer Protocol (HTTP) and XML to exchange information. Frequently the term implies the use of SOAP to exchange information between applications, versus application-to-human, which is done in HTML.

**Well-formed (XML)** - An XML instance that meets well-formedness constraints defined by the XML 1.0 specification. Well-formedness constraints are precise syntactic rules for markup of data. As an example, the XML specification stipulates that every open tag must have a corresponding and properly nested closing tag. A document must be well-formed in order to be considered XML. A parser processing a document will throw a fatal error if it detects a well-formedness violation.



Well-formed HTML - HTML that meets the well-formedness constraints of XML 1.0. Well-formed HTML is not the same as XHTML.

XHTML - Extensible HyperText Markup Language<sup>xlvii</sup>.

XML - [From the XML 1.0 specification] “Extensible Markup Language, abbreviated XML, describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML. By construction, XML documents are conforming SGML documents.” The XML 1.0 specification is a W3C Recommendation. In XML, metadata is described by an extensible set of tags; the tags are said to be extensible, because unlike HTML, where the markup tags are fixed, developers are given the flexibility to define their own tags or reuse tags defined by another party. This flexibility is both the key to XML’s power and the single biggest stumbling point to achieving interoperability when making use of XML.

(XML) API - Application Programming Interface. In the context of XML, parsers expose their data to a calling application via an interface. An interface is a specification (which the parser conforms to) that describes how the parser will pass data from an XML document to a calling application. The two accepted XML API’s are DOM and SAX.

(XML) Attributes—In the context of XML, attributes provide a mechanism for attaching additional metadata to an XML element. For example, <element attribute=“value”/>. An XML attribute is not equivalent to an object or relational model attribute. Data model entity attributes may be expressed as either XML attributes or elements. Frequently in discussions surrounding the application of XML to data models, one party will be referring to attributes in the context of XML and another to attributes in the context of data models, causing confusion.

XML Comments—The structure for inserting free text comments into XML. The same structure is used for SGML and HTML comments. <!--comment text here-->

XML Component—A generic term used to refer to XML elements, attributes, and XML Schema type definitions.

(XML) Document - - [Paraphrased from the XML 1.0 specification] “A data object is an XML document if it is well-formed, as defined in the XML 1.0, specification. A well-formed XML document may in addition be valid if it meets certain constraints” as described by a schema. Synonymous with XML instance.

(XML) Elements—The fundamental unit of information in XML. Elements are encapsulated by tags, and may contain (among other things) attributes (declared inside the opening tag), other elements, or data.

---

(XML) Child Element—The hierarchical nature of XML allows elements to contain or be nested inside other elements, forming a conceptual data tree (see DOM). Often XML elements are referenced in terms of parent-child relationships. A child element is an element contained between the tags of a parent element. Child elements are also referred to as descendants, while parent elements may be referred to as ancestors.

(XML) Grammar/Vocabulary—Related terms often used synonymously to indicate a set of element and attribute names and the structures described by a schema or set of related schemas that employ the elements and attributes. More precisely, the term vocabulary implies a commonly defined set of elements and attributes, while grammar refers to the composition of the vocabulary into meaningful business documents by one or more related schemas. An XML Namespace may be used to describe a vocabulary, while a schema may employ vocabulary from a single or multiple XML Namespaces.

(XML) Instance - Synonymous with XML Document. The term derives from object-oriented programming where objects are considered instances of classes. Programmers write code that defines application behavior in terms of classes of objects. In application execution, objects are instantiated (see object) from these class definitions. XML provides an object-like way to conceptualize textual data. Essentially, schemas are the equivalent of object classes, and XML documents are equivalent of object instances. Hence the term XML instance is widely used, however XML document is the official term used by the W3C.

XML Namespace—An XML Namespace is a conceptual “space” to which element and attribute names may be assigned. An XML Namespace is declared within an XML instance by assigning a URI reference and an optional qualification prefix to an element. The element and all its children are considered to be “in” the XML Namespace unless specifically qualified with another Namespace’s prefix. The URI reference does not have to an associated document physically at the URI. Within an XML Schema, the ‘targetNamespace’ attribute may be used to indicate that all elements declared within the schema are to be treated as “in” the target Namespace. The W3C Recommendation Namespaces in XML<sup>xlvi</sup> provides the full specification for XML Namespaces. Note: Federal XML Namespaces may use XML Namespaces, but the two terms are not synonymous.

(XML) Name Token—Per the XML 1.0 specification, a Name Token is “...any mixture of name characters...” where a “name” character obey the XML name convention. A [XML] Name “...is a token beginning with a letter or one of a few punctuation characters, and continuing with letters, digits, hyphens, underscores, colons, or full stops, together known as name characters. Names beginning with the string “xml”, or any string which would match ((‘X’|‘x’) (‘M’|‘m’) (‘L’|‘l’)), are reserved for standardization in this or future versions of this specification.” White space characters (hex #x20, #x9, #xD, #xA) are excluded from Name Tokens.

(XML) Parser - A software application (module) that either reads or receives a text encoded binary stream, decodes it, verifies the input conforms to “well-formedness” constraints of the XML 1.0 specification, (in the case of a Validating Parser) checks validity of the XML Instance against a schema if available, and exposes the content via an API to a calling application. A parser can be a stand-alone application, but it is most often a module called by a larger program (the calling application). A Parser may also be referred to as an XML Processor.

(XML) Processor - A synonym for an XML parser.

XML Declaration—Every well-formed XML document must begin with a statement that as a minimum declares the version of XML that the document conforms to. Example: `<?xml version=“1.0”>`,

XML Document Tree—Refers to the logical model of an XML document conceptualized as a data tree, with a Root Node and branch nodes ending at data that can be thought of as the leaves. See DOM.

(XML) Root Node—The first node originating the XML Document Tree. The Root Node is not the same as the root element.

(XML) Root Element—Refers to the XML element in which all other elements must be nested. The root element (a physical XML construct) is a child of the logical root node of the document tree.

(XML Schema) Type—An XML component defined by the XML Schema language. Types do not show up in XML instances; they are used within the Schema to express relationships, and through type inheritance, add an object-like capability to XML Schemas. Types may be simple, that is they allow definition of simple data-type constraints on element values; or they may be complex, that is they define structures consisting of other elements. For example a type could be defined `<xsd:complexType name=“AddressDetails”>...</xsd:complexType>`, then the definitions for XML elements, ‘ShippingAddress’ and ‘MailingAddress’ could reference the previously defined generic type.

(XML) Schema Annotation—The XML Schema language allows addition of annotations to schema components through an ‘annotation’ element (`<xsd:annotation>`) which must contain either a ‘documentation’ element (`<xsd:documentation>`) or ‘AppInfo’ element (`<xsd:appInfo>`). A ‘source’ attribute may be added to either element to provide a URL reference to the source of the annotation. Annotations provide a more sophisticated way to provide documentation and application information that may be parsed and accessed by applications via an API.

(XML) Tags - XML (and its parent SGML) annotate metadata through the use of tags that indicate which text in a document are considered metadata and which is to be considered data. Tags are surrounded by markup characters. As an example, the data ‘3000N’ can be marked up in XML, `<latitude>3000N</latitude>`. The

---

tags are <latitude> (start tag) and </latitude> (end tag). Note: As discussed in the XML definition presented here, developers are free to define tags. As an example, the data '3000N' could be alternatively marked up as, <lat>3000N</lat>, and still be well-formed. The document schema will specify which of all possible well-formed XML instances are valid for a particular application. An additional example is <Latitude hemisphere="N">3000</Latitude>; here the tag contains an XML attribute to specify the hemisphere. The choice as to the attribute name and possible values are also at the developer's discretion. Note that Parsers processing documents are sensitive to markup tag case, therefore in the first example the tag <latitude> is not equivalent to the later example tag, <Latitude>.

XPath—XPath is a W3C recommendation whose primary purpose is to provide a compact, non-XML notation for identifying parts of an XML document. It operates on the abstract, logical structure of an XML document, rather than its surface syntax by modeling an XML document as a tree of nodes. The document tree can be navigated by applications implementing XPath. XPath is the result of an effort to provide a common syntax and semantics for functionality shared between XSL Transformations [XSLT] and XPointer.

XSL - The Extensible Style Sheet Language. [From the W3C XSL pagexlix] "XSL is a language for expressing stylesheets. It consists of three parts: XSL Transformationsl (XSLT): a language for transforming XML documents, the XML Path Language (XPath), an expression language used by XSLT to access or refer to parts of an XML document (XPath is also used by the XML Linking specification). The third part is XSL Formatting Objects: an XML vocabulary for specifying formatting semantics. An XSL stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses the formatting vocabulary. For a more detailed explanation of how XSL works, see the What Is XSL page." As of 16 October 2001, XSL is a W3C final recommendation.

XSL Processor - The software (module) executing XSL transformation and formatting instructions. At a minimum, consists of an XSLT conformant transformation component, and an optional XSL-FO processing component. A word of caution: XSL processor vendors often add "extensions" to the XSLT specification. While often extremely useful, stylesheets written using these extensions will not perform correctly in another XSLT compliant processor, eliminating their cross-platform compatibility.

XSL-FO - XSL Formatting Objects: an XML vocabulary for specifying formatting semantics. XSL-FO works in conjunction with XSLT to markup transformed XML with formatting object tags. Applications capable of processing these tags render the XML to another application's presentation environment. For example, Apache's Formatting Object Processor (FOP) can transform XML to Adobe PDF format. Another example is jfor, an open-source formatting object processor for transforming XML to Rich Text Format (RTF).

XSLT - XSL Transformations, a W3C recommendation [from the XSLT recommendation] “...defines the syntax and semantics ... for transforming XML documents into other XML documents” [including well-formed HTML].” XSLT is the only W3C-recommended XML syntax for transforming XML documents. Developers writing stylesheets should ensure they are strictly conformant to this specification to ensure reusability. We recommend conformance testing using several XSLT-compliant XSL processors.

---

<sup>i</sup> RFC 2119, <http://www.ietf.org/rfc/rfc2119.txt>

<sup>ii</sup> XML Schema Tutorial, <http://www.xfront.com/xml-schema.html>

<sup>iii</sup> Schema Best Practices, <http://www.xfront.com/BestPracticesHomepage.html>

<sup>iv</sup> UN/CEFACT, <http://www.UN/CEFACT.org/>

<sup>v</sup> UN/CEFACT UML2XML, <http://www.UN/CEFACT.org/projects/u2xdr.html>

<sup>vi</sup> Federal XML Registry, <http://diides.ncr.disa.mil/xmlreg/user/index.cfm>

<sup>vii</sup> ebXML Specifications and Technical Reports, <http://www.ebxml.org/specs/>

<sup>viii</sup> OASIS, <http://www.oasis-open.org/>

<sup>ix</sup> UN/CEFACT, <http://www.unece.org/cefact>

<sup>x</sup> UN/CEFACT, <http://www.UN/CEFACT.org/>

<sup>xi</sup> ebXML Core Component Technical Reports,  
[http://www.ebxml.org/specs/#technical\\_reports](http://www.ebxml.org/specs/#technical_reports)

<sup>xii</sup> ebXML Technical Architecture, <http://www.ebxml.org/specs/ebTA.pdf>

<sup>xiii</sup> ebXML Technical Report, Naming Convention for Core Components  
<http://www.ebxml.org/specs/ebCCNAM.pdf>

<sup>xiv</sup> Business Process and Business Information Analysis Overview v1.0,  
<http://www.ebxml.org/specs/bpOVER.pdf>

<sup>xv</sup> ebXML Business Process Specification DTD, <http://www.ebxml.org/specs/ebBPSS.dtd>

<sup>xvi</sup> ebXML Business Process Specification XML Schema (CR),  
<http://www.ebxml.org/specs/ebBPSS.xsd>

- 
- xvii COE XML Namespace Managers, [http://diides.ncr.disa.mil/xmlreg/user/namespace\\_list.cfm](http://diides.ncr.disa.mil/xmlreg/user/namespace_list.cfm)
- xviii COE XML Registration Information, [http://diides.ncr.disa.mil/xmlreg/user/registry\\_info.cfm#submit](http://diides.ncr.disa.mil/xmlreg/user/registry_info.cfm#submit)
- xix The Object Management Group, <http://www.omg.org/>
- xx UN/CEFACT Core Component Project, <http://www.UN/CEFACT.org/projects/core.html>
- xxi DDDS, <http://www-datadmn.itsi.disa.mil/ddds/ddds40.html>
- xxii DOD 8320, <http://www-datadmn.itsi.disa.mil/guidance.html>
- xxiii W3C DOM, <http://www.w3.org/DOM/>
- xxiv HTML, <http://www.w3.org/MarkUp/>
- xxv OMG IDL, [http://www.omg.org/gettingstarted/omg\\_idl.htm](http://www.omg.org/gettingstarted/omg_idl.htm)
- xxvi ISO Store, <http://www.iso.ch/iso/en/prods-services/ISOstore/store.htm>
- xxvii XML 1.0, <http://www.w3.org/TR/2000/REC-xml-20001006>
- xxviii Namespaces in XML, <http://www.w3.org/TR/1999/REC-xml-names-19990114/>
- xxix SAX, <http://www.megginson.com/SAX/>
- xxx ISO 8879 (SGML), <http://www.w3.org/TR/2000/#ISO8879>
- xxxi XML Cover Pages - SOAP, <http://xml.coverpages.org/soap.html>
- xxxii SOAP 1.1, <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
- xxxiii SOAP 1.2, <http://www.w3.org/TR/2001/WD-soap12-20010709/>
- xxxiv W3C Process, <http://www.w3.org/Consortium/Process-20010719/submission>
- xxxv UML, <http://www.rational.com/uml/index.jsp>
- xxxvi Unified Modeling Methodology, <http://www.gefeg.com/tmwg/tm090.pdf>
- xxxvii Rational Unified Process, <http://www.rational.com/products/rup/index.jsp>
- xxxviii W3C Note, URI/URL/URN Clarification, <http://www.w3.org/TR/2001/NOTE-uri-clarification-20010921/>
- xxxix W3C, <http://www.w3.org/>
- xl W3C Members, <http://www.w3.org/Consortium/#membership>
- xli W3C Consensus Processes, <http://www.w3.org/Consortium/Process-20010719/submission>
- xlii W3C Schema page, <http://www.w3.org/XML/Schema>
- xliii W3C Activity Statement, <http://www.w3.org/XML/Activity.html>
- xliv XML Schemas: Part 0, <http://www.w3.org/TR/xmlschema-0/>
- xlv XML Schemas: Part 1, <http://www.w3.org/TR/xmlschema-1/>
- xlvi XML Schemas: Part 2, <http://www.w3.org/TR/xmlschema-2/>
- xlvii XHTML, <http://www.w3.org/MarkUp/#xhtml1>
- xlviii Namespaces in XML, <http://www.w3.org/TR/1999/REC-xml-names-19990114/>
- xlx W3C XSL Page, <http://www.w3.org/Style/XSL/>

- 
- <sup>i</sup> XSL Transformations, <http://www.w3.org/TR/xslt>
  - <sup>ii</sup> XPath, <http://www.w3.org/TR/xpath>
  - <sup>iii</sup> XLink, <http://www.w3.org/TR/xlink/>
  - <sup>iiii</sup> What is XSL, <http://www.w3.org/Style/XSL/WhatIsXSL.html>
  - <sup>iv</sup> XSL Final Recommendation, <http://www.w3.org/TR/2001/REC-xsl-20011015/>
  - <sup>lv</sup> XSLT, <http://www.w3.org/TR/xslt>