United States Legislative Markup

User Guide for the USLM Schema

Prepared by: Office of the Law Revision Counsel

(with assistance from the Legislative Branch XML Working Group)

U.S. House of Representatives

Revised August 2016 (0.2.0)

CONTENTS

1 USLM SCHEMA

1.1 OVERVIEW

United States Legislative Markup (USLM) is an XML information model designed to represent the legislation of United States Congress. Initially, USLM is being used to produce titles of the United States Code in XML, but it is designed to be adaptable for appendices to titles of the United States Code as well as bills, resolutions, statutes, and certain other legislative materials. USLM is intended to meet the following needs:

1. Allow existing titles of the United States Code to be converted into XML.
2. Support ongoing maintenance of the United States Code.
3. Support the drafting of new positive law codification bills and related materials.
4. Provide a flexible foundation to meet future needs of the United States Congress.
5. Be compatible with other legislative documents that already exist in other XML formats.

This User Guide describes, at a high level, how USML is designed and how it can be used. USLM is designed using the XML Schema Definition language (XSD). This User Guide is intended for individuals familiar with XSDs and with document information modeling. For more information on XSDs, see the W3C website at http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/ and other reference materials on these subjects.

Note: *Version 1.0 of XML Schema is used for USLM. A more recent version, V1.1, is available as a recommendation, but currently there is very limited tool support for the more recent version.*

1.2 PRINCIPLES

Following a 1999 feasibility study on XML/SGML,[1] the Committee on House Administration adopted XML as a data standard for the exchange of legislative documents. As a result, the U.S. House of Representatives first publicly released Congressional bill data in XML in 2004. Since that time, a number of best practices and consistent approaches have evolved around the use of XML across various industries. To the greatest extent possible, the design of USLM leverages current best practices in legislative information modeling.

1.2.1 Model the Data as It Appears

To the greatest extent possible, text that is published is maintained in the main body of the document in the order it appears when presented in publication.

[1] The feasibility study is rooted in a 1996 directive from the Committee on House Oversight (now known as the Committee on House Administration) and the Senate Committee on Rules and Administration to the Clerk of the House and Secretary of Senate, respectively, to work together toward establishing common data standards for the exchange of legislative information. See also 2 U.S.C. 181.

1.2.2 Leverage Existing Standards

To the greatest extent possible, established XML standards are incorporated into USLM. For example, XHTML[2] is used for tables and the Dublin Core[3] is used for metadata.

1.2.3 Element Text vs. Attributes

XML attributes are reserved for metadata and/or normalized representations of the element text. No attribute text should ever appear, as is, in any manifestation of a USLM document.

1.2.4 Generated Content

In general, generated text is avoided in USLM. Over the past decade, as Congress has gained experience with bill drafting in XML, drafters have found that the use of generated text can be problematic, particularly when working with existing law. This general rule does not prevent specific renderings that use the USLM data to generate supplemental display information, such as the header or footer of the printed versions.

1.3 SCOPE

USLM is designed to support a specific set of documents that are legislative in nature. While it is not a general model for documents outside this specific set, USLM may be applicable for Federal regulatory publications

1.3.1 Types of Documents Covered

USLM is designed to represent the legislation of United States Congress. Initially, USLM is being used to produce titles of the United States Code in XML, but it is designed to be adaptable for appendices to titles of the United States Code as well as bills, resolutions, statutes, and certain other legislative materials.

1.4 GOALS

The USLM schema is defined with the following goals in mind:

1. **Ease of Learning** – XML schemas can be difficult to learn and master. It may be quite difficult to envision how all the pieces fit together and how various cases are to be accommodated. In addition, the heavy use of jargon may complicate the picture. To counter this, USLM is designed to be learned incrementally and to favor legislative and end-user terminology over computer jargon. USLM is laid out to allow someone learning it to learn each stage separately and only progress to the next stage when the prior stage has been mastered.

2. **Extensibility** – It is not possible to anticipate all the document structures that may arise in legislation. The sheer volume of data makes an exhaustive analysis of documents difficult, and it is impossible to predict all future needs for alternative legislative structures or drafting styles. A legislative schema must be able to evolve to allow changes to be incorporated, with a minimum of impact, in either a temporary or permanent way. USLM is designed, using XML Schema's inheritance mechanisms, to allow for easy extensibility.

[2] For more information, see: http://www.w3.org/TR/xhtml11/Overview.html#toc
[3] For more information, see: http://dublincore.org/

3. **Editability** – One of the greatest challenges when designing an XML-based information model is to define a model which can be used for editing. USLM is designed to support both completely-formed documents and the editing of documents under construction.

1.5 CONVENTIONS

Conventions are important for establishing consistency.

1.5.1 Conventions Used in the User Guide

The following conventions are used in the User Guide:

1. XML element names are denoted with angled brackets and in courier. For example, `<title>` is an XML element.
2. XML attribute names are denoted with an "`@`" prefix and in courier. For example, `@href` is an XML attribute.
3. Enumerated values are denoted in courier. For example, `landscape` is an enumeration.
4. String values are denoted with double quotes and in courier. For example, "`title1-s1`" is a string value.
5. A new ***term*** being defined is shown in bold italic.

1.5.2 Conventions Used in the XML

The conventions below are used in the XML schema and XML document instances. It is important that these conventions be adhered to when the information model is initially defined, when it is modified, and when instances of documents that conform to it are created.

1.5.2.1 Namespaces

USLM supports the use of namespaces. XML provides a namespace mechanism to allow XML element names to be defined without naming collisions.

1.5.2.2 Namespace URI

XML namespaces are associated with a URI (Uniform Resource Identifier) which is known to be unique. This URI acts as an identifier defining, unambiguously, to which model the element belongs. It is possible to define a namespace URI as either a URN URI (a naming convention) or a URL URI (a locating convention). In USLM, URL URIs are used as a convention, which is the most common current practice.

For USLM, the namespace URI is defined as the following URL:

```
http://xml.house.gov/schemas/uslm/1.0
```

1.5.2.3 Namespace Prefix

Ordinarily, a namespace prefix is not necessary and should not be used. However, in cases where a namespace prefix is deemed necessary, the preferred prefix is "`USLM`".

1.5.2.4 Schema Versioning

A schema versioning model is defined for use in USLM documents. If USLM is modified, a new version number in the format “n.n.n” is assigned to the schema. The lower order digit is used to represent evolutionary additions to the schema, the next digit will change only if there is a an incompatible modification to the schema, often called a “breaking change” and the highest order digit will change if there is a major rewrite of the standard. Breaking changes will only be implemented after all other options have been exhausted. The schema version for a USLM document is identified using the version attribute on the root element.

Note: *Schema changes will not be undertaken lightly. Changes can be costly and can severely impact systems that rely on the information model.*

1.5.3 Naming

In USLM, naming conventions are used to promote consistency, cleaner design, and ease of use. In general, names are chosen to be short and to the point, while still conveying the primary purpose. Abbreviations are used only when the abbreviation is commonly accepted and not likely to cause confusion.

In general, one-word names are preferred. If necessary, two- or three-word names are used. Two-word names are in the form adjective + noun. For three-word names, the last word defines a general class or type.

1.5.3.1 Type & Groups

Type and group names are defined in UpperCamelCase. For example, `IdentificationGroup` is an attribute group and is expressed in a form where all the words start with an upper case character.

1.5.3.2 Elements

Element names are defined in lowerCamelCase. For example, `<longTitle>` is an element name and is expressed in a form where all the words, except the first one, start with an upper case character.

1.5.3.3 Attributes

Attribute names are defined in lowerCamelCase. For example, `@alignTo` is an attribute name and is expressed in a form where all the words, except the first one, start with an upper case character.

1.5.3.4 Names and Identifiers

All `@temporalId` names are written in lower case, with underscore (“_“) separators between significant portions of the name. All `@id`s are defined to be globally unique as GUIDS and, for compatibility reasons, start with the "`id`" prefix.

1.6 RELATIONSHIP TO XHTML

USLM leverages the XML version of HTML 4.0, commonly called XHTML. Many USLM attribute and element names purposely coincide with their XHTML equivalents.

1.6.1 Identity and locating attributes: id, idref, href, and src.

The `@id`, `@idref`, `@href`, and `@src` reference are identical to their XHTML equivalents.

1.6.2 Role attribute

The `@role` attribute is similar to the proposed `@role` attribute in XHTML and HTML5. It is used to provide additional semantic information about the purpose of an element. It is particularly useful with abstract elements that lack clear semantics.

1.6.3 Style attributes: class and style

The `@class` attribute is similar to its XHTML equivalent. It should be used to associate presentation classes, defined in a CSS file, with an element.

The `@style` attribute is used to specify instance-specific styling, as an override to the class specific styling. Its use in USLM is identical to that in XHTML.

The `@class` and `@style` attributes are explained in greater detail in section 9.2.

1.6.4 Generic elements

A number of elements in USLM are defined to be identical in name and function to elements in XHTML. This set is limited to elements that are likely to be needed within the main legislative language in situations where utilizing the XHTML namespace would be cumbersome or would prevent further use of the legislative structures available in USLM. In other cases where a USLM equivalent element is not defined, it is acceptable and recommended to use the XHTML element with the appropriate namespace information.

The elements borrowed from XHTML are: `<p>`, `<br>`, `<img>`, `<center>`, `<b>`, `<i>`, `<sub>`, `<sup>`, `<del>`, and `<ins>`. In addition to this set, there are other elements which share the same name as XHTML elements, but in those cases, the semantics behind the elements are either not completely similar or are totally different. Do not assume XHTML semantics merely because the element name coincides with an XHTML element name.

1.7 RELATIONSHIP TO AKOMA NTOSO

USLM is not defined to be either a derivative or subset of Akoma Ntoso (http://www.akomantoso.org/). It would be premature to define USLM in that way while the effort is still underway in the OASIS (http://www.oasis-open.org/committees/legaldocml) standards group to establish Akoma Ntoso as the XML standard for legislative documents. However, USLM is designed to be consistent with Akoma Ntoso to the extent practicable. Many of the element and attribute names in USLM match the Akoma Ntoso equivalents. As Akoma Ntoso becomes a standard, and as demand for it emerges, it should be possible to produce an Akoma Ntoso XML rendition of a USLM document through a simple transformation.

1.8 External Dependencies

USLM has no dependencies on any other information model aside from the core XML model. However, the United States Code titles in USLM currently depend on the Dublin Core and XHTML namespaces. USLM supports the optional use of other information models to create composite documents made up of multiple namespaces. In particular, the use of the following information models is encouraged:

1) Dublin Core for metadata.
2) XHTML for tables and other loosely structure content.
3) MathML for equations.
4) SVG for vector graphics.

2 ABSTRACT MODEL VS. CONCRETE MODEL

There are two basic document models in USLM - the abstract model and the concrete model:

1. The abstract model is a general, highly flexible model defined using a minimum set of tags.

2. The concrete model is derived from the abstract model, but it is narrower. The concrete model is defined to precisely model the United States Code. It exists as a simple derivation from the abstract model. While the abstract model uses general terminology, the concrete model uses specific terminology. The specific terminology used in the concrete model is based on well-established terminology used in the Office of the Law Revision Counsel, which produces the United States Code.

If a tag is defined in the abstract model, and it is sufficient for direct use, the tag is not redefined in the concrete model. The abstract tag is used in documents without change. For example, the `<num>` tag is defined in the abstract model and used without change in all documents. The inheritance technique discussed in section 4 is used to establish these two models.

3 INHERITANCE

Inheritance is a technique, available in many computer languages, to define a basic object with basic behavior and then produce specializations by adding or modifying the behaviors. Inheritance is also available in XML. There are two ways to implement inheritance in USLM:

1. In most situations, inheritance can be implemented in USLM using an approach inspired by the XML Schema approach. A base class is defined, and variants are derived from the base class to create new elements and modify the attribute and content models. The formality of this approach is an advantage. Adding classes requires a schema change, which constrains the creation of new classes without due consideration.

2. For anomalous situations occurring so infrequently that modification of the schema is unwarranted, inheritance can be implemented in USLM with an approach inspired by XHTML, using the `@role` attribute to create subclasses. The element name represents the base class, and the `@role` attribute value represents the subclass. This approach makes it easy to create subclasses without changing the schema. Although this method provides welcome flexibility for special cases, it should be used sparingly to avoid the creation of many poorly defined and poorly supported subclasses.

Note: *There is a rough equivalence between the two approaches. For instance,* `<level role="chapter">` *is roughly analogous to* `<chapter>`*. However, there is no formal mechanism within XML to establish this equivalence.*

3.1 POLYMORPHISM

In a programming language, ***polymorphism*** is the ability to use an instance of a subclass wherever an instance of the base class is expected. XML schemas support polymorphism. However, in XML schemas, unlike programming languages, polymorphism is not an implicit capability that comes along with inheritance. In XML schemas, polymorphism is achieved by defining a base level element and then defining subclassed elements to be part of the base elements **substitution group**.

Note: *In XML Schema V1.0, substitution groups have limitations that were put in place to prevent ambiguous situations from arising which might be difficult for a tool or program to understand. These limitations have been addressed in XML Schema V1.1. Unfortunately, XML Schema V1.1 has not yet been adopted widely enough to serve as the basis for USLM.*

3.2 ANOMALOUS STRUCTURES

Two issues drive the wide variety of legislative structures found in the United States Code.

First, the United States Code contains Federal law enacted over a period of centuries, and legislative drafting styles evolve over time. To handle this issue, the data is converted using the XML schema subclassing mechanism.

Second, some Federal statutes contain features that do not conform to any commonly accepted drafting style, past or present. To handle this issue, the data remains in the anomalous form, subclassed when possible using the `@role` attribute.

4 ABSTRACT MODEL

4.1 CONCEPT

The foundation of USLM is an abstract model that is designed to be a complete, but generic, legislative model using a minimum number of XML elements. It is possible to markup any U.S. bill or resolution or any title of the United States Code using the abstract model alone, albeit in a very general way.

The abstract model contains three basic sets of elements: the primitive set, the core set, and the generic set.

- The **primitive set** defines the fundamental building blocks used to construct everything else.
- The **core set** defines the basic document model.
- The **generic set** defines a basic set of general-purpose tags used to markup general structures.

4.2 PRIMITIVE SET

The primitive set is a set of four primitive elements that are the fundamental building blocks of the abstract model. All USLM elements can be traced back through the derivation hierarchy to one of the four primitive elements. The four primitive elements are the following:

1	`<marker>`	A marker is an empty XML element. It is used to denote a location or position within an XML document.
2	`<inline>`	An inline is an XML element that can be placed within text content – similar to an XHTML `<span>` element. An inline element can contain other inline elements, markers, or text.
3	`<block>`	A block is an XML element that is presented as a block-like structure and does not contain direct child text content.
4	`<content>`	A content is an XML element that is presented as a block-like structure and that can contain a mixture of text and XML elements.

4.3 CORE SET

The core set is a set of twenty-nine elements. Taken together, these twenty-nine core elements define the basic document model, which consists of six parts:

1. The root level document.
2. The metadata block.
3. The main document body (including a table of contents, statements, a preamble or enacting clause, and hierarchical levels).
4. A structure for references.
5. A structure for amendments.
6. A structure for any appendices.

The twenty-nine core elements that define the six-part basic document model are the following:

1	<lawDoc>	The document root for a legislative document.
2	<document>	The document root for a loosely-structured non-legislative document.
3	<meta>	An optional container at the start of the document for metadata.
4	<property>	A piece of metadata, usually in the <meta> block.
5	<set>	A set of metadata, usually containing properties.
6	<toc>	A table of contents.
7	<tocItem>	An item within a table of contents.
8	<main>	The primary container for the body of the document.
9	<statement>	Any statement at the start of the document.
10	<preamble>	A collection of recitals, ending with an enacting formula at the start of the document.
11	<recital>	A clause within the preamble.
12	<enactingFormula>	The enactment words at the end of the preamble or found in place of a preamble if the preamble is omitted.
13	<level>	A hierarchical item within the document.
14	<num>	The number, letter, or alphanumeric combination assigned to a hierarchical level.
15	<text>	A block of text, to be used whenever text is required in a block level structure but a parent element is required to conform to the schema. This is a base class for the <chapeau> and the <continuation> elements.
16	<heading>	An optional name or designation of a level element.
17	<subheading>	An optional name or designation, to be used below a <heading>.
18	<crossheading>	Placed within and amongst heading levels to separate items within a level.
19	<instruction>	A container used to describe an amendment to legislation. Contains <action> elements, as well as the relevant <quotedText> or <quotedContent>.
20	<action>	Describes the change to be made in an <instruction> element.

21	<notes>	A container for <note>s.
22	<note>	An individual note.
23	<appendix>	A stand-alone appendix document, such as a United States Code title appendix, or an appendix at the end of the document. This can be either an embedded document or a referenced document.
24	<signatures>	A container for <signature>s.
25	<signature>	An individual signature, containing a name and, optionally, a role, affiliation, and/or date.
26	<ref>	A reference or link to another whole document, a specific location in another document, or another location within the same document.
27	<date>	A date.
28	<quotedText>	Plain text quoted from another document or intended to be placed, as stated, in another document.
29	<quotedContent>	Quoted content that may be plain text, XML elements, or text plus elements. The content may be quoted from another document or may be intended to be placed, as stated, in another document.

4.4 GENERIC SET

The generic set defines a set of general-purpose tags used to markup basic structures. Many of the USLM generic elements are borrowed from XHTML, but exist within the USLM namespace because it is often impractical or impossible to use XHTML structures directly, such as when special legislative structures are embedded within general structures (or general structures are embedded within special legislative structures). In such instances, the USLM generic set is used. The USLM generic set includes the following:

1	<layout>	A region to be presented in a column-oriented layout – similar to a table.
2	<header>	A heading row in a <layout> structure.
3	<row>	A normal row in a <layout> structure. In general, this level can be omitted.
4	<column>	A column cell in a <layout> structure.
5	<p>	A normal (unnumbered) paragraph. The semantics for a paragraph should be preserved. Do not use the <p> element as a general block like element.
6	 	A line break. This element should only be used to force a line break when other more semantic elements are not sufficient to achieve the desired

		formatting.
7	`<img>`	An embedded image. This is a marker element which points, via a URL, to the image to be embedded.
8	`<center>`	Centered text. While this tag is deprecated in HTML 4.01, it is provided here for convenience as centering text is common.
9	`<fillIn>`	A region of text intended to be filled in on a form.
10	`<checkBox>`	A check box intended to be checked on a form.
11	`<b>`	Bold text.
12	`<i>`	Italic text.
13	`<sub>`	Subscripted text.
14	`<sup>`	Superscripted text.
15	`<del>`	Deleted text within a modification.
16	`<ins>`	Inserted text within a modification.

4.5 ATTRIBUTE GROUPS

The general-purpose attributes that are used in the abstract model (or in the concrete model derived from the abstract model) can be grouped into several categories.

4.5.1 Identification

The following attributes are used to identify elements in various ways. The identification group is a universal group and can be used on all elements.

1	`@id`	An immutable (unchangeable) id assigned to an element upon creation. It should be preserved as is when an element is moved. However, when an element is copied, new values for all `@id` attributes in the copied fragment should be generated.
2	`@name`	A name assigned to an element that can be parameterized to support computation of a name or id a point in time. For example: "`s{num}`".
3	`@temporalId`	An evolving name assigned to an element that reflects it current type and any numbering.

4	`@identifier`	Used on the root elements to specify the URL-based path reference to that element. The `@identifier` is specified as an absolute path (i.e. a path beginning with a single slash) in accordance to the rules of the Reference Model described in section 13. `@identifier` is an evolving identity and may only be valid for a specific time.

4.5.2 Classification

The following attributes are used to classify elements. These are primarily used for informal subclassing and styling purposes.

1	`@role`	Assigns a single semantic subclass to an element. This attribute is primarily for use with abstract elements, but may be used across all elements to provide a richer semantic association.
2	`@class`	Assigns one or more presentation classes to an element. These classes relate to CSS classes.
3	`@style`	Embeds CSS styles, particular to a specific instance, with the element.

4.5.3 Annotation

The following attributes are used to annotate or mark elements, usually for editorial reasons. Attribute values are not shown in a published form. The annotation group is universal and can be used on all elements.

1	`@note`	A simple text note not to be published.
2	`@alt`	An alternate description of an element. The `@alt` attribute is intended to map to the HTML `@alt` attribute for use with WCAG 2.0 and other accessibility initiatives.
3	`@meta`	An association with an element. How this attribute is used is not prescribed.
4	`@misc`	[Reserved for future use.]
5	`@draftingTip`	For internal use by the Offices of the Legislative Counsel of the U.S. House of Representatives or the U.S. Senate (or other entities in the legislative branch).
6	`@codificationTip`	For internal use by the Office of the Law Revision Counsel of the U.S. House of Representatives (or other entities in the legislative branch).

4.5.4 Description

The following attributes are used to describe or categorize elements.

1	`@title`	A short textual description of an item.
2	`@brief`	A longer textual description of an item.

3	`@sortOrder`	A numeric integer used to signify the order in which an item should appear with its siblings.

4.5.5 Reference

The following attributes are used to establish pointers or references to other documents or locations within the same document.

1	`@href`	A URL reference to another document or part of another document. The `@href` URL in USLM is generally a path beginning with a single slash. A "resolver", described in section 13, maps between the path found in the URL and the full URL required to retrieve the target item.
2	`@idref`	A reference to an item within the same document, identified by specifying the value of the `@id` attribute for the target element. If the `@idref` points to a `<ref>` element, then the referencing attributes of the target element are inherited, in a recursive fashion. This concept is described in more detail later in section 13.
3	`@portion`	Specifies an additional part of a reference to be tacked onto the existing context of a reference (usually established through the `@idref`). If the `@portion` does not begin with a separator character ("/", "!", "@"), then a "/" is assumed.

4.5.6 Action

The following attributes are used in amendments to declare how amendments are to be made.

1	`@type`	Describes, through an enumerated value, the action being taken.
2	`@occurrence`	Describes which occurrence of an item on which the action is to be taken.
3	`@commencementDate`	Specifies the date on which the action is to be taken.

4.5.7 Amending

Attributes used to point to items being amended. The amending group is used with the Reference Group and should be used only within amending instructions.

1	`@pos`	Specifies a relative position in which an action is to occur, such as `before` or `after` the item being referenced.
2	`@posText`	Used in conjunction with the `@pos` attribute to specify a location. It specifies text within the referenced item that the `@pos` is relative to.

3	`@posCount`	Used in conjunction with the `@posText` attribute when an occurrence other than the first occurrence of a string of text is to be acted upon. In addition to specifying a particular instance, enumerated values are available to specify other sets, such as `all`.

4.5.8 Link

The following attribute is used to link other documents or images intended to be embedded within the primary document.

1	`@src`	A URL reference to a document, image, or other item to be embedded in a document. If `@src` contains an absolute path, it should be handled by a resolver, similar to an `@href` attribute.

4.5.9 Value

The following attributes are used to hold normalized values computed from the text content.

1	`@value`	A normalized value representing the content of the element.
2	`@startValue`	A normalized value representing the start of a range expressed in the content of the element.
3	`@endValue`	A normalized value representing the end of a range expressed in the content of the element.

4.5.10 Date

The following attributes are used to hold normalized forms of date and time values computed from the text content. All date and date times are expressed as YYYY-MM-DD[Thh:mm:ss[Z|(+|-)hh:mm]].

1	`@date`	A normalized date (or date and time) representing the content of the element.
2	`@beginDate`	A normalized date (or date and time) representing the start of a time/date range expressed as content of the element
3	`@endDate`	A normalized date (or date and time) representing the end of a time/date range expressed as content of the element

4.5.11 Versioning

The following attributes are used to manage the temporal, or time-based, aspects of legislation and the law. All time/dates used in the versioning group use the same time/date format as the Date group.

1	`@startPeriod`	The earliest date (or date and time) a version applies to.
2	`@endPeriod`	The latest date (or date and time) a version applies to.
3	`@status`	The state of a provision during the period.

4	`@partial`	A Boolean specifying that the status is only partially in effect.

4.5.12 Cell

The following attributes are used for managing column structures. The `@colspan` and `@rowspan` attributes are borrowed from HTML and follow HTML's all lowercase convention.

1	`@colspan`	Specifies how many columns the current column cell should span. By default, a column cell only spans a single column.
2	`@rowspan`	Specifies how many rows the current column cell should span. By default, a column cell only spans a single row.
3	`@leaders`	Specifies a character to be used as a leader. Typically a dot leader (“.”) is used to create a series of dots.

4.5.13 Note

The following attributes are used for positioning and categorizing individual notes and groups of notes.

1	`@type`	When used within a note or a notes container, `@type` specifies the position of the note. Setting the `@type` attribute to “footnote” indicates that the note or notes contained should be shown in the footnotes at the end of the page and setting to “endnote” indicates that the note or notes contained should be shown at the end of the document. If not specified, “footnote” is assumed. For users of @type other than within notes, see below.
2	`@topic`	Specifies the focus of the notes. The `@topic` attribute is set to a string value in order to categorize the note or group of notes. An open, but enumerated, list of string values should be used. Using a fixed list of values will better aid in categorization of notes later.

4.5.14 Other Attributes

The following are other miscellaneous attributes.

1	`@xml:lang`	The language of the text contained.
2	`@xml:base`	A URL that can be used to resolve all URLs found in the document. USLM URLs are generally specified as absolute paths. This is to make USLM documents portable, allowing them to be rehosted in a different location without modification. The `@xml:base` provides a preferred location of a resolver capable of resolving the contained references. However, this location is only an advisory. It is be possible for a local system to determine its own preferred resolver location.

3	`@orientation`	Used to specify when a landscape orientation is to be used when the item is published. If not set, the default orientation is portrait.
4	`@type`	Specifies an enumerated categorization beyond the classing and subclassing provided by USLM. Different elements that use the `@type` attribute provide different enumerated values to use. The `@type` attribute is generally used when the classification defines procedural or programmatic behaviors that must be implemented in the system.

5 CORE DOCUMENT MODEL

5.1 CONCEPT

The core document model uses the core elements of the abstract model discussed above to define a simple model for constructing legislation or law with abstract elements. This model is summarized below. A number of details are omitted for the sake of brevity.

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

<lawDoc
      xmlns=http://xml.house.gov/schemas/uslm/1.0
      xsi:schemaLocation=”http://xml.house.gov/schemas/uslm/1.0
        ./USLM-1.0.xsd”
      xml:base="http://resolver.mydomain.com"
      identifier=”/us/usc/t5”>
   <meta>
      <property name=”docTitle”>…</property>
      …
   </meta>

   <main>
      <layout>
         <header>Table of Contents</header>
         <toc>
            <tocItem title=”Chapter 1”>
               <column>1.</column>
               <column leaders=”.”>General Provisions</column>
               <column>101</column>
            </tocItem>
         </toc>
      </layout>

      <level role=”Chapter”>
         <num value=”1”>CHAPTER 1.</num>
         <heading>General Provisions</heading>
         <content>
            ...
         </content>
      </level>
   </main>
</lawDoc>
```

5.2 METADATA

The metadata block, <meta>, is an optional block of named properties or sets of named properties at the start of the document. Properties are defined as simple strings. The metadata model is open and unconstrained to provide maximum flexibility.

Information found in the metadata block is generally not printed in the published form.

Some of the metadata may be generated through analysis of the main text of the document. Whenever the document is modified, this metadata is regenerated to accurately reflect the current state.

Metadata can be defined using (1) the abstract <property> and <set> elements, (2) the built-in <docNumber>, and other properties, or (3) elements defined in the Dublin Core.

5.3 MAIN

The main text of the document is contained in the <main> block. The main block will not have an @id or a @name attribute. References to items in the main body may skip or suppress the main level in the reference.

5.4 APPENDICES

An appendix can either follow the main part of a document or be a stand-alone document. To include an appendix as a stand-alone document, omit the <main> element and include the <appendix> element directly after the metadata. This technique is used for United States Code title appendices (e.g., Title 5 Appendix).

There can also be any number of appendices following the main part of the document. These may also be known as schedules, annexes, or explanatory notes/memoranda. An <appendix> element can either contain the content within the document or it can reference the content for inclusion using the @src attribute.

5.5 SIGNATURES

Some documents contain signatures of the people who introduce, sponsor, or approve the legislation. The signatures are held in a <signatures> block, either at the top of the main part of the document or in the appendices.

5.6 MULTIPLE MODELS

Models from multiple XML namespaces are used to construct a USLM document. The dcterms model is used for metadata. The XHTML model is used for tables. XHTML may also be used to mark the external document for inclusion within the larger legislative document. In the future, MathML may be used for equations and SVG may be used for vector graphics.

6 CONCRETE MODEL

6.1 CONCEPT

The concrete model builds on the abstract model discussed in section 5 of this document. The concrete model implements a broad set of tags to meet specific semantic needs now and in the future. These tags are generally implemented as simple synonyms of the tags in the abstract model. This approach preserves the simplicity and flexibility of the abstract model.

6.2 DOCUMENTS

	Element	Derived From	Contains
1	<bill>	<lawDoc>	A proposed bill
2	<statute>	<lawDoc>	An enacted bill
3	<resolution>	<lawDoc>	A proposed resolution
4	<amendment>	<lawDoc>	A document containing a pre-enactment stage amendment
5	<uscDoc>	<lawDoc>	A title or appendix of the United States Code

6.3 PROPERTIES

	Element	Derived From	Contains
1	<docNumber>	<property>	A numeric designation assigned to the document
2	<docPublicationName>	<property>	The name of the publication that the document is part of
3	<docReleasePoint>	<property>	The point (i.e. the Public Law number for a United States Code title) at which the document was released

6.4 TITLES

	Element	Derived From	Contains
1	<docTitle>	<statement>	A statement that precedes the long title in the bill
2	<longTitle>	<statement>	A statement that sets out the purposes of the bill
3	<shortTitle>	<inline>	The short title of a bill where it is first defined

6.5 LEVELS

	Element	Derived From	Contains
1	<preliminary>	<level>	A hierarchical region of the main document consisting of preliminary clauses that are outside of the main document hierarchy
2	<title>	<level>	A hierarchical level in a legislative document
3	<subtitle>	<level>	A level below <title>
4	<chapter>	<level>	A hierarchical level in a legislative document
5	<subchapter>	<level>	A level below <chapter>
6	<part>	<level>	A hierarchical level in a legislative document
7	<subpart>	<level>	A level below <part>
8	<division>	<level>	A hierarchical level in a legislative document
9	<subdivision>	<level>	A level below <division>
10	<article>	<level>	A hierarchical level in a legislative document
11	<subarticle>	<level>	A level below <article>
12	<section>	<level>	The primary hierarchical level in a <title> or <bill>
13	<subsection>	<level>	A level below <section>. Usually numbered with lower-case letters.
14	<paragraph>	<level>	A level below <section>,often below a <subsection>. Usually numbered with Arabic numbers.
15	<subparagraph>	<level>	A level below <paragraph>. Usually numbered with upper-case letters.

16	<clause>	<level>	A level below <subparagraph>. Usually numbered with lower-case Roman numerals.
17	<subclause>	<level>	A level below <clause>. Usually numbered with upper-case Roman numerals.
18	<item>	<level>	A level below <subclause>. Usually numbered with double lower-case letters.
19	<subitem>	<level>	A level below <item>. Usually numbered with double upper-case letters.
20	<subsubitem>	<level>	A level below <subitem>. Usually numbered with triple lower-case letters.
21	<compiledAct>	<level>	An act that is included in a title appendix. Amendments to these acts are typically compiled into the included act.
22	<courtRules>	<level>	A level container to hold a sequence of court rules. Found in title appendices.
23	<courtRule>	<level>	An individual court rule. Found in title appendices.
24	<reorganizationPlans>	<level>	A level container to hold a sequence of reorganization plans. Found in title appendices.
25	<reorganizationPlan>	<level>	An individual reorganization plan. Found in title appendices.

6.6 Other Structures

	Element	Derived From	Contains
1	<def>	<text>	One or more <term> elements, as well as their respective definitions
2	<term>	<inline>	A term in the document that is being defined
3	<chapeau>	<text>	Introductory text that comes before lower levels in a level hierarchy
4	<continuation>	<text>	Final or interstitial text that comes after or between lower levels in a level hierarchy

5	<proviso>	<text>	A paragraph of text, usually beginning with "Provided that" or "Provided", that states conditions on the law to which it is related

6.7 NOTES

	Element	Derived From	Contains
1	<sourceCredit>	<note>	Text containing the source of a provision, usually surrounded by parentheses
2	<statutoryNote>	<note>	A note that becomes part of the law
3	<editorialNote>	<note>	A note included for editorial purposes only
4	<changeNote>	<note>	A note that records a non-substantive change that has been made to the document, usually surrounded by square brackets

6.8 SIGNATURES

	Example	Derived From	
1	< made>	<signature>	The signatures of the people making the legislation
2	<approved>	<signature>	The signatures of the people approving the document

6.9 APPENDICES

	Example	Derived From	
1	<schedule>	<appendix>	An appendix to a document, often a list of numbered items, a table, or another document

7 VERSIONING MODEL

7.1 CONCEPT

Legislative documents frequently evolve as existing provisions are repeatedly amended. Managing the versioning of legislative documents is a complex problem.

Rather than managing versioning by creating a whole new document every time an amendment is made, the versioning is instead managed in a more granular way. USLM is designed to allow a large degree of flexibility in how the versioning takes place. This allows for limitations in the short term and for greater sophistication as needs evolve.

7.2 MANAGING VERSIONS

The USLM model for versioning allows versions to be handled in a hierarchical manner from the document root all the way down to individual provisions. In general, it is best to version at the lowest possible level of the hierarchy. This minimizes the likelihood of amendments overlapping in time, which would require the creation of different versions to handle the different states of the text through the overlapping time period.

In addition, if versioning is too high up in the hierarchy, then each new version will cause an entire copy of the lower levels to be generated. For this reason, it is preferable to push the versioning down to the lower parts of the level – such as the `<num>` or the `<heading>` elements when versioning upper (or big) levels in legislative text.

When choosing a level at which to apply versioning, it must be possible for multiple versions of an element to coexist alongside one another. For this reason, the structure defined in the schema allows multiple instances of an element (`maxOccurs="unbounded"`).

7.3 TEMPORAL PERIODS

Alternate versions of the same item exist alongside one another within the same document, with each version being associated with a specific temporal period defined by the `@startPeriod` and `@endPeriod` attributes. In cases where one of these attributes is missing, then the same attribute found at a higher level in the document hierarchy applies. If no such attribute is found, then the time period is open-ended in that temporal direction.

The `@startPeriod` or `@endPeriod` can be defined as either a simple date or as datetime. When defined as a simple date, then midnight at the beginning of the date is assumed when establishing the correct instant for point-in-time calculations.

7.4 STATUS

The `@status` attribute is used to declare the status of a version. It is important to realize that the `@startPeriod` and `@endPeriod` are defining a time period for a version of an item and may not correspond to an effective date or repeal date.

Statuses in USLM include `proposed`, `withdrawn`, `cancelled`, `pending`, `operational`, `suspended`, `renumbered`, `repealed`, `expired`, `terminated`, `hadItsEffect`, `omitted`, `notAdopted`, `transferred`, `redesignated`, `reserved`, `vacant`, `crossReference`, and `unknown`.

8 PRESENTATION MODEL

8.1 CONCEPT

Presentation is based on concepts similar to XHTML. An overriding rule is to separate, as much as possible, the formatting of the text for publication from the semantic model. To the greatest extent possible, the XML elements of USLM reflect the semantic model rather than the formatting model. Some formatting elements related to presentation are included in the XML, such as the `<layout>` structure and the `<b>` and `<i>` elements. These formatting elements are provided for the sake of practicality.

Most styling is handled using Cascading StyleSheets (CSS), as is the case with modern HTML.

8.2 CSS ATTRIBUTES

There are two primary attributes which are used to affect the presentation of the text:

1	`@class`	The `@class` attribute is used, as in HTML, to identify CSS classes.
2	`@style`	The `@style` attribute is used to specify CSS attributes. Ordinarily, all presentation attributes should be specified in a separate CSS file. However, there are many legacy cases where individual instances do not follow the standard form for presentation of that element. In these cases, the converter should leave the non-standard formatting with the `@style` attribute where it can override the CSS attributes defined in the external stylesheet.

8.3 HTML REPRESENTATION

There is a straightforward method to transform a USLM-based XML document to and from HTML5. This approach is designed to maintain the integrity of the information. Bidirectional transformations can occur in a web-based editing environment without losing or changing any data.

The method to transform from USLM-based XML to HTML5 is as follows:

1. All XML elements based on the marker, inline, or string types are transformed into HTML5 `<span>` elements.

2. All XML elements based on the block or content types are transformed into HTML5 `<div>` elements.

3. The `<layout>` XML elements are translated into HTML5 `<table>` elements. The `<layout>` child elements become `<tr>` elements unless they are the USLM `<column> elements.` In that case , a single `<tr>` row is defined to contain the columns, and the `<column>` elements become `<td>` HTML table cells.

4. Any `@role` attribute on the XML element becomes the `@role` attribute on the HTML5 element. If there is no `@role` attribute on the XML element, then the element name becomes the HTML5 `@role` attribute.

5. The `@class` attribute on the HTML5 element is composed of the local name of the XML element, followed by an underscore character and the XML @role attribute value if it exists, followed by the XML `@class` attribute values, space separated. For instance, the element `<level role="Chapter" class="indent1">` becomes the `@class="level_Chapter indent1"` attribute.

6. Any `@xml:lang` attribute on the XML element becomes the HTML5 `@lang` attribute.

7. The following attributes carry over from the XML element to an identically named HTML5 attribute: `@style`, `@id`, `@href`, `@idref`, `@src`, `@alt`, `@colspan`, and `@rowspan`.

8. All other attributes carry over to the HTML5 by inserting the prefix "`data-`" + namespacePrefix + "`-`" + XML attribute. For example, the `@name` attribute on the XML element becomes the `@data-uslm-name` attribute.

9. The entire resulting HTML5 document fragment is ordinarily placed within the HTML5 `<body>` element.

10. CSS classes are associated with the resulting HTML5 fragment using well-established HTML5 methods.

The transformation from HTML5 back to USLM-based XML is accomplished by reversing the process described above.

9 HIERARCHICAL MODEL

9.1 CONCEPT

Most legislative documents have a well-defined hierarchical structure of numbered levels. The abstract model provides a general-purpose hierarchy necessary for legislative documents. This hierarchy does not impose restrictions, but instead allows any `<level>` to be placed within any `<level>`. This flexibility allows varying structures, some of which are anomalous, to be supported without adding layers of complexity.

9.2 LEVELS

The fundamental unit of the hierarchy is the `<level>` element. A discussion below describes how the `@class` attribute and XML schema substitution groups can be used to subclass this fundamental unit to produce the various levels found in United States legislation.

A level is composed of (1) a `<num>` identification designation, (2) an optional `<heading>`, and (3) either primarily textual `<content>`, lower hierarchical `<level>` children, or a few other possible elements.

9.3 BIG LEVELS VS. SMALL LEVELS

The principal level is the `<section>` level. Levels above the `<section>` level are referred to as “big” levels and levels below the `<section>` level are referred to as “small” levels.

Big Levels	title, subtitle, chapter, subchapter, part, subpart, division, subdivision
Primary Level	section
Small Levels	subsection, paragraph, subparagraph, clause, subclause, item, subitem, subsubitem

The primary difference between big levels and small levels is in how they are referred to in references. Big levels and primary levels are referred to using a prefix to identify the level’s type. Small levels are referred to simply by using the number designation assigned to the level in a level hierarchy. Further details are discussed below under Referencing Model.

9.4 SANDWICH STRUCTURES

Sandwich structures are hierarchical levels that start and/or end with text rather than lower levels. This structure is quite common. Typically, some text will introduce the next lower level or will follow the last item of a lower level. The `<chapeau>` (French for "hat") and `<continuation>` elements are provided for this structure. The `<chapeau>` precedes the lower levels and the `<continuation>` follows the lower levels. The `<continuation>` element can also be used for cases where interstitial text is found between elements of the same level.

One specific type of continuation text is a paragraph-like structure beginning with “Provided that” or “Provided”. The <proviso> element is used in this case. Multiple provisos may exist.

9.5 TABLE OF CONTENTS

A table of contents (TOC) model is provided to model the level hierarchy. A TOC can appear either at the top of a hierarchy or at lower levels, and in the main part of the document or in an appendix. The root of the TOC structure is the <toc> element and the levels can be arranged into a hierarchy of <tocItem> elements. Attributes from the description group are used to define the information in the hierarchy.

The <toc> structure can be intermixed with the <layout> structure to define a tabular layout for a table of contents.

10 TABLE MODEL

Two table-like models can be used with USLM: (1) a column-oriented model and (2) the HTML table model.

10.1 COLUMN-ORIENTED

Use the column-oriented `<layout>` model when information related to the legislative structure is to be arranged in a column- or grid-oriented fashion, but not a true table. The advantage of the column-oriented `<layout>` model is that it is defined within USLM, so it can contain other USLM elements. The drawback is that it is a non-standard table model, so it is not inherently understood by tools that handle standard HTML tables.

The `<layout>` model is intended to be flexible. It provides a `<row>` element for defining individual rows. When there is a row structure, the parts of each row that belong within each column are identified using the `<column>` element. Any direct child of the `<layout>` element is considered to be a row unless it is a `<layout>` element.

However, when the `<column>` element is a direct child of the `<layout>` element, then there is no notion of rows, and the columns form a basic structure of the `<layout>`. If `<layout>` contains any `<column>` child, then it must only contain `<column>`s as children.

Like HTML tables, the `<layout>` model supports the `@colspan` and the `@rowspan` elements.

10.2 HTML TABLES

Use the HTML `<table>` model when (1) information is arranged in a tabular structure, (2) there is little information within the table that is part of the legislative structure, or (3) the structure is regarded as a normal table with gridlines and table cells.

An embedded HTML table will look something like this:

```
<schedule name=”sch{num}”>
   <num value="1">Schedule 1</num>
   <heading>…</heading>
   <table xmlns=http://www.w3.org/1999/xhtml”>
      <th>…</th>
      <tr>…</tr>
      …
   <table>
</schedule>
```

11 IDENTIFICATION MODEL

11.1 CONCEPT

Elements are assigned identifiers or names for two primary purposes. The first is to be able to reliably refer to the element throughout its lifetime, regardless of how it might be altered. The second is to be able to address the item based on its current state. To support both of these purposes, the available attributes are `@id`, `@temporalId`, `@name`, and `@identifier`.

11.2 IMMUTABLE IDENTIFIERS

Immutable identifiers are unchanging identifiers that are assigned when the element is created and do not change throughout the lifetime of the element. This makes them reliable handles with which to access the elements in a database or repository system. The `@id` attribute is used for this purpose. It is defined as an XML Schema ID, which requires that all `@id` attribute values be guaranteed to be unique within a document, with no exceptions.

For the purposes of document management in USLM, especially in the amending cycle, `@id` values should be computed as GUID (Globally Unique Identifiers) with an "id" prefix. This means that they should be computed using an algorithm that guarantees that no two identifiers, in any document, anywhere, will ever be the same. This is a broader definition of uniqueness than imposed by the XML Schema ID definition. There are many tools available to generate GUIDs.

Whenever an element is moved, its `@id` attribute value must be preserved. When an element is copied, a new value for the `@id` attribute value must be generated for the new element created. Special care must be taken to ensure that the `@id` value is managed correctly. Proper management of the `@id` attribute value will provide a reliable handle upon which to attach other metadata such as commentary.

It is important that the value of the `@id` attribute not reflect, in any way, some aspect of the element that might change over time. For instance, if there is a number associated with an element and that number is subject to renumbering, then the `@id` attribute value should have no relation to the number that is subject to renumbering.

11.3 TEMPORAL IDENTITY

A `@temporalId` is a human-readable identity, scoped at the document level. While the `@id` attribute is defined to be unique in the document and constant throughout the lifetime of the element, and the schema enforces this uniqueness, the `@temporalId` attribute is defined loosely and changes to reflect the current location and numbering of the element over time.

Because the `@temporalId` attribute is assigned a value that reflects the current state of the element, special care must be taken to ensure that the value of the `@temporalId` attribute is recomputed anytime the state of the element changes. It is usually a good practice to recompute the `@temporalId` values whenever the document is committed or saved.

Ideally, the `@temporalId` attribute should be unique in a document, but this is not always possible due to various anomalies. For this reason, the uniqueness of the `@temporalId` attribute is not enforced by the schema, and some ambiguity is possible. How the disambiguation of duplicate names is handled is a subject that must be dealt with in the design of the software systems which will encounter this situation.

A recommended approach for computing the `@temporalId` value is to base the name on the hierarchy to get to the element, almost in a path-like fashion. The `@temporalId` value can be constructed as follows:

```
[parentId + "_"] + [bigLevelPrefix] + num
```

Where:

`parentId` is the `@temporalId` value of the parent element. If the parent element does not have a `@temporalId` value or does not have a unique `@temporalId`, then the local XML name of the parent element is used, with special care being taken to ensure that all parent elements that are not unique have assigned `@name` values.

`bigLevelPrefix` = a prefix reflecting the level type, such as "`p`" for part, "`d`" for division, or "`sd`" for subdivision. For sections use "`s`". For small levels, no prefix is used.

`num` = the normalized value of the number or designation given to the level.

Exceptions:

The `<doc>` root level should be omitted from the computation.

The `<main>` level should be omitted.

Levels of the hierarchy should be omitted whenever the numbering of a level does not require references to the higher levels. For example, section numbers are usually unique throughout the document, so it is not necessary to use the higher big levels to compute a name. So a section can be identified as simply "`s1`" rather than "`p1_d1_s1`".

For example, part III of subchapter II of chapter 12 of Title 8 would have a `@temporalId` of "ch12_schII_ptIII", and subparagraph (A) of paragraph (1) of subsection (a) of section 1201 of Title 8 would have a `@temporalId` of "s1201_a_1_A".

11.4 Local Names

Local names are usually related to the parent element or container in which they are found. This is the purpose of the `@name` attribute. The most common use of the `@name` attribute is when naming a `<property>` element. The `@name` attribute is also used to name a level within the local context of its parent level.

There is a problem with naming a level: its name is subject to change through time. This is because levels are subject to renumbering. To support this, the `@name` can be defined in a parameterized way. The parameters will need to be evaluated whenever a document is requested for a specific point-in-time.

The parameters are specified within the `@name` attribute value using a curly braces notation. Two parameters can be specified:

1) Use the `{num}` parameter to include the current normalized value (i.e., the `@value` attribute of the `<num>` element) in the name of the level.
2) Use the `{index}` parameter to include the 1-based index position, calculated against other elements of the same type at the same level.

To better ensure the uniqueness of the `@name` attribute values generated in the future, a rational scheme must be designed. This is important because the `@name` attribute is also used in the mapping of links or references to elements. This process is accomplished by a web server add-in called a "resolver". Resolvers are described in the next chapter.

11.5 IDENTIFIERS

An `@identifier` is used on the root element to specify the URL reference to the document root. The `@identifier` is specified as an absolute path in accordance to the rules of the Reference Model described in this document.

An `@xml:base` attribute is also specified on the root element to specify the location of a preferred resolver capable of resolving a reference. The `@xml:base` concatenated with the `@identifier` forms a complete URL reference.

Typically, the `@identifier` will be established on the root element and all level elements.

12 REFERENCING MODEL

12.1 CONCEPT

References are a machine readable format for making very precise citations or establishing links between different things in a document. The prevailing method for establishing references is to use HTTP-based hyperlinks, using the familiar technology prevalent on websites.

These references are, like websites, modeled as Universal Resource Locators (URL). A URL is a string representing a hierarchical path down to the item being requested, using forward slashes "`/`" as hierarchical separators. In normal websites, each level in the URL represents a folder, terminating in a file that is being requested. URLs can be specified in one of three ways: (1) global references starting with "`http://{domain}`"; (2) absolute paths starting with "`/`"; or (3) relative references starting with "./". Absolute paths typically use the local domain as the context for the URL, while relative references use the current file as the context.

USLM references use a variation of the absolute path technique. All references thus start with a forward slash "/". However, rather than representing folder and files, the hierarchical path represents a conceptual hierarchy down to the item in question. This path is known as a logical path. The logical path does not represent the folder/file hierarchy as with a physical path. In fact, there may be no physical path for information stored in a database rather than in a file system.

Web servers usually handle the task of interpreting a URL and retrieving the requested file from the file system. With USLM references, however, the mapping is not so straightforward. A web server must interpret the logical path in the URL and retrieve the requested information from a database. This task is accomplished by a web server add-in called a "resolver".

How the resolver is constructed depends on the web server being used and the storage format for the documents. All modern web servers provide some form of facility to allow a resolver to be constructed. This issue is discussed in greater detail below under Reference Resolver.

12.2 URL REFERENCES

The International Federation of Library Associations and Institutions (IFLA) (http://www.ifla.org/) has developed a conceptual entity-relationship model for organizing bibliographic records (like index cards at a library). This model is called the Functional Requirements for Bibliographic Records (FRBR – pronounced "*Ferber*"). FRBR creates the conceptual framework for the USLM references.

References in USLM are composed using the following format:

```
[item][work][lang][portion][temporal][manifestation]
```

Where:

- `item` – identifies the location of an instance. For non-computer locations, this is expressed as an http domain. An example would be `http://uscode.house.gov`.

- `work` – identifies the logical hierarchy down to the document being referenced. This hierarchy starts by identifying the jurisdiction ("`/us`" for United States) and continues by identifying the document ("`/usc/t5`" for Title 5). The jurisdiction is included in order to distinguish between the library that serves the document and the jurisdiction where the document originated. With this approach, it is possible for a library to serve a document from a different jurisdiction.

- `lang expression` ("!" prefix) – identifies the language. If the lang is not specified, then the language is assumed to be the language of referencing document or referencing environment.

- `portion` ("/" prefix) – extends the work hierarchy to identify an item within the document. For example, "`/s1/a/2`" for paragraph (2) of subsection (a) of section 1 in the main body. Note that the portion is an easy mapping of the `@temporalId` for that element which is "s1_a_2". This gives a hint for how to resolve the `portion` part of a URL identifier.

- `temporal` expression ("@" prefix) - the date/time is expressed according to ISO 8601 ("`@2013-05-02`" for May 2, 2013). If the "@" is specified, but without a date/time, then the reference is to the current time. If no temporal expression is specified, the context may be used to identify the point-in-time, which is usually the date of the document making the reference.

- `manifestation` ("." prefix) – identifies the format as a simple file extension ("`.xml`" for the XML file, "`.htm`" for HTML, and "`.pdf`" for the PDF).

Examples:

- `/us/usc/t5/s1/a` – the current version of subsection (a) of section 1 of title 5.

- `/us/usc/t5/s1/a@2013-05-02` – the version of subsection (a) of section 1 of title 5 that was in effect on May 2, 2013.

- `/us/usc/t5/s1/a.htm` - the current version of subsection (a) of section 1 of title 5, rendered as HTML.

- `http://uscode.house.gov/download/us/usc/t5/main/s1/a.htm` the current version of subsection (a) of section 1 of title 5, rendered as HTML and delivered from http://uscode.house.gov/download.

Notes:

- References in documents (using the `@href` and `@src` attributes) should always be stored as absolute paths, which omit the item part. This allows the reference to be independent of the site hosting the document. The role of the resolver is to determine which location can best serve the desired item. This allows a document to be moved from one digital library to another without changing the references within the XML. The item location is implicit in the library containing the reference. An exception is when a specific item is desired, usually when referencing an item from a foreign jurisdiction.

- There are generally two common methods to identify a document's type. One method is by extension as described above. The other method is to use the MIME type. The MIME type is a more robust solution because it allows for a wide variety in file extensions for the same type (`e.g., ".htm" or ".html"` for HTML files). However, file extensions are simpler and less cumbersome. This means that an agreed upon registry of file extensions should be maintained by the system.

12.3 Reference Attributes

There are four attributes which contain references or portions of references:

1) `@href--` This is a pointer or link to another document. It is generally stored as an absolute path. Prepending the domain to identify a particular instance or library from which the information is to be sourced is left to the local resolver. This allows a document to be relocated in another digital library without changing all the references.

2) `@portion--` Often the textual representation of a reference is scattered in several places in a document. For instance, a set of amendments might be prefaced with an identification of the document affected, such as title 5 of the United States Code, while the individual amendments might specify only a portion of that document, such as subsection (a) of section 1. The `@portion` attribute allows a reference to be extended. This will generally be constructed as follows:

```
<ref id=”ref001” href=”/us/usc/t5”/>
…
<ref idref=”ref001” portion=”/s1/a”/>
```

The example above shows an initial reference to title 5, United States Code. The second reference refers to the first, acquiring the first reference's `@href` and then extending it with the `@portion` to produce the reference `/us/usc/t5/s1/a`. This approach can be recursive.

3) `@src` – in addition to pointing to other documents, it is often desirable to embed other documents within a primary document. The `@src` attribute is used in this case.

4) `@origin` – When a fragment of a document is copied into another document, and it is necessary to record the place from which the fragment was copied, the `@origin` attribute is used. This exists for the `<quotedText>` and `<quotedContent>` elements.

12.4 Referencing Nomenclature

The following case-insensitive referencing nomenclature is used;

Short Form	Long Form	Description
`pl[0-9]+`	`publicLaw[0-9]+`	Public Law + number – Statute
`t[0-9|a-z]+`	`title[0-9|a-z]+`	Title + number
`st[0-9|a-z]+`	`subtitle[0-9|a-z]+`	Subtitle + number
`ch[0-9|a-z]+`	`chapter[0-9|a-z]+`	Chapter + number
`sch[0-9|a-z]+`	`subchapter[0-9|a-z]+`	Subchapter + number
`p[0-9|a-z]+`	`part[0-9|a-z]+`	Part + number
`sp[0-9|a-z]+`	`subpart[0-9|a-z]+`	Subpart + number
`d[0-9|a-z]+`	`division[0-9|a-z]+`	Division + number
`sd[0-9|a-z]+`	`subdivision[0-9|a-z]+`	Subdivision + number
`s[0-9|a-z]+`	`section[0-9|a-z]+`	Section + number
`art[0-9|a-z]+`	`article[0-9|a-z]+`	Article + number
`r[0-9|a-z]+`	`rule[0-9|a-z]+`	Rule + number
`[a-z]+`	`[a-z]+`	Subsection letter
`[0-9]+`	`[0-9]+`	Paragraph number
`[A-Z]+`	`[A-Z]+`	Subparagraph Letter (capital letters)
`[i-x]+`	`[i-x]+`	Clause (lower case roman numeral)
`[I-X]+`	`[I-X]+`	Subclause (upper case roman numeral)
`[aa-zz]+`	`[aa-zz]+`	Item (double lower case letter)
`[AA-ZZ]+`	`[AA-ZZ]+`	Subitem (double upper case letter)
`[aaa-zzz]+`	`[aaz-zzz]+`	Subsubitem (triple lower case letter)
`(suppress)`	`main`	Main body
`shortTitle`	`shortTitle`	Short title
`longTitle`	`longTitle`	Long title
`preamble`	`preamble`	Preamble

`proviso`	`proviso`	Proviso
`app[0-9]*`	`appendix[0-9]*`	Numbered or unnumbered appendix

Note: *The prefixes are defined to be case-insensitive. This is done as case-sensitive URLs can be problematic in some environments.*

12.5 References within Amendment Instructions

Amendments refer to the item that they are amending. The reference may be complex, specifying not only the item affected, but a relative position either within, before, or after the item affected. Three additional attributes are provided with references to allow this sort of specification:

- `@pos` – Specifies a position that is either at the `start`, `before`, `inside`, `after`, or at the `end` of the context item.

- `@posText` – Establishes the context for the position relative to text contained within the referenced item.

- `@posCount` – Specifies which occurrence of the `@posText` within the referenced item is being acted upon. By default, the first occurrence is assumed. In addition to specifying which occurrence, the values `all`, `none`, `first`, and `last` may also be used.

12.6 Reference Resolver

The URL-based references that are established create the links between various documents within the system. A software component is added to the web server to interpret the references, find the relevant piece within the database repository, extract it, and perform any necessary assembly and transformation before returning the result to the requester. This web server add-in is called a resolver. How it is built is determined by the web servers being used. In general, the resolver will perform the following sequence of functions:

1) It will receive a reference from a requestor.

2) It will canonicalize the reference, normalizing the text to match one of the forms it understands.

3) If the reference is to the U.S. jurisdiction and the resolver understands the reference, then it will attempt to resolve it by retrieving the XML from the document. This might be either an entire document of a fraction thereof.

4) If the reference is to another jurisdiction, and the resolver is able to resolve the reference, either locally or by deferring to another web server, then the resolver will resolve the reference that way.

5) If the reference is not understood, then the resolve will return a *404 – file not found* error.

6) If the document is being resolved locally, and the XML has been extracted from the database, then it may need to be assembled or otherwise processed to ensure that the correct temporal state has been established. If no temporal information is contained in the URL, then the present state is assumed.

7) Once the correct XML has been created, if a format other than XML has been requested it will need to be transformed and/or converted into the correct format. This may involve transforming the XML into HTML or creating a PDF.

8) Some of the steps above may be circumvented in the interest of performance and efficiency with a good caching strategy.

9) Once the requested item has been retrieved, assembled, and transformed, it is returned to the requestor using HTTP.

There are several strategies that the resolver can use to find the item referenced by the `work` part of the reference URL:

1) The fastest method, if there is a reliable mapping between the `@name` value and the work part of the reference URL, is to map between the reference path and the `@name`. This approach is best when the XML documents are shredded into parts and stored as separate items, either in the file system or in a relational database.

2) Another strategy is to rewrite the reference URL hierarchy as an XPath query. This approach is best when there is a good mapping between the reference hierarchy and the document hierarchy, and the information is stored in an XML repository that supports XPath. Performance might be an issue for more complex XPath queries.

3) The third strategy is to create an indexing mechanism. This solution might rely on the inherent capabilities of the chosen database or repository, or it might be some sort of predefined mapping. How this strategy should ultimately be designed is beyond the scope of this User Guide.

For a specific document, the preferred resolver is identified using the `@xml:base` attribute on the root element. For instance:

```
xml:base="resolver.mydomain.com"
```

The `@xml:base` concatenated with the `@identifier` forms a complete URL reference.

A preferred resolver does not currently exist for USLM. Therefore, the `@xml:base` attribute is not provided in current USLM documents. The United States House intends to provide a resolver in the future. If and when that occurs, the `@xml:base` attribute will point to that resolver.

13 METADATA MODEL

13.1 CONCEPT

In addition to the text in an XML document, there is also a need to store a significant amount of metadata about a document. There are a few ways in which this metadata might be stored:

1) Within the document in a separate partition.
2) Scattered within the document.
3) In a separate file.
4) In a relational database.

All four of these approaches can be supported. First, there is an optional `<meta>` block defined at the start of the document. Within this block, properties and sets of properties can be stored. The model for this metadata is open and extensible to support a wide range of needs, while also keeping the core concepts very simple. The metadata stored here can either be generated in an ongoing fashion, or as the result of an analysis of the text after it has been committed, or as a combination of these.

In addition to the basic `<meta>` block, attributes are provided throughout the document for storing metadata about a particular element with that elements. Most of these attributes have prescribed usage, and the model is not as general and flexible as the `<meta>` block. However, there are a few attributes set aside for unprescribed uses. These include the `@misc`, `@draftingTip`, and `@codificationTip`.

It is possible to store metadata in a separate file or in a relational database. If a separate file is chosen, no format for this file is prescribed. It can be an XML file, some other text file, or even a binary file. One option for the format is to borrow the `<meta>` tag with its `<property>` and `<set>` children from USLM. This is merely an option; it is not prescribed.

If the information is stored in a separate file or is stored in a database, then it may be necessary to maintain a strict association between the XML elements and the records in the file. For this reason, element `@id` values are defined to be immutable, in order to provide a reliable handle for making associations. If the `@id` attribute cannot be managed reliably, then the separate file and database options should be avoided.

13.2 PROPERTIES

Properties are basic elements that may or may not have string content. The `@name` attribute acts as the primary identification for a `<property>`. The `@value` attribute (and its range siblings) or the `@date` attribute (and its range siblings) are used to place normalized values of dates. Sometimes, a value or date might exist as both text content in the element and, in a normalized form, as an attribute.

Properties are primarily intended for use within the `<meta>` block or within `<set>` groupings within the `<meta>` block. However, it is also possible for properties to be used as inline elements within the main part of the document.

13.3 SETS

Properties can be grouped into simple sets. A `<set>` is essentially a property folder. Like a `<property>`, the `@name` attribute acts as the primary identification for a property. Property sets can be nested.

14 NOTES MODEL

14.1 CONCEPT

Notes are found throughout the United States Code. USLM defines a very flexible model to support all the different types of notes that are found.

14.2 NOTE CLASSES

The abstract model provides two basic elements for implementing notes.

14.2.1 Individual notes

The basic `<note>` implement provides the fundamental model for a note. A note can be simple string of text or it can be a complex structure of text.

14.2.2 Notes collection

Notes can be grouped together into a collection using a `<notes>` container.

14.3 TYPE OF NOTES

There are four primary types of notes. Use the `@type` attribute to specify the note type:

1) `inline` – notes that are shown inline where they appear in the text.

2) `endnote` – notes that appear at the end of the level in which they appear. A `<ref>` pointer may be used to point to these notes

3) `footnote` - notes that appear at the bottom of the page in which a reference to that note appears. A `<ref>` pointer is used to point to these notes.

4) `uscNote` - notes that appear at the bottom of the section or heading, typically after the `sourceCredit`.

14.4 TOPIC OF NOTES

Notes in the United States Code often have a specific topic, such as "Amendments". Use the `@topic` attribute to specify the note's topic(s). More than one topic can be specified, separated by spaces, such as:

```
@topic="regulations construction"
```

15 FEEDBACK

The Office of the Law Revision Counsel of the U.S. House of Representatives welcomes any questions or comments about the United States Code in USLM at uscode@mail.house.gov. For questions, comments, or requests for proposed changes to the USLM schema or this User Guide, please visit GPO's GitHub repository and open an issue at https://github.com/usgpo/uslm/issues/new.

16 APPENDIX: UNITED STATES CODE

16.1 BRIEF INTRODUCTION TO THE UNITED STATES CODE

The schema described in this User Guide is used to produce the United States Code in XML. The United States Code contains the general and permanent laws of the United States, organized into titles based on subject matter.

The United States Code is prepared and published by the Office of the Law Revision Counsel of the U.S. House of Representatives pursuant to 2 U.S.C. 285b. For the printed version of the Code, a complete new edition is printed every six years, and five annual cumulative supplements are printed in the intervening years.

The Office of the Law Revision Counsel also produces an online HTML version of the United States Code for searching and browsing (http://uscode.house.gov/). The online HTML version of the United States Code is updated continuously as new laws are enacted.

The Office of the Law Revision Counsel also produces (beginning July 30, 2013) an XML version of the United States Code for download (http://uscode.house.gov/download/download.shtml). The XML version is updated continuously as new laws are enacted.

16.2 BULK DATA DOWNLOADS OF XML FILES

16.2.1 Directory Structure

The download directory (http://uscode.house.gov/download/download.shtml) contains one XML file for each title of the United States Code and a zip file for the entire Code. The directory also includes the XML schema files required for XML validation. A CSS stylesheet is provided for convenience. The CSS stylesheet is informational only and is not part of the United States Code.

16.2.2 Download Protocols

Bulk download is supported via HTTP protocols.

16.2.3 Versions

The most current version of the United States Code is available in XML, and prior versions in XML created on or after July 30, 2013, are also available. Eventually, the titles of some prior editions of the United States Code will also be available in XML. During the beta period, the XML format is subject to change. Some titles may be replaced with updated versions. The creation date and effective date of each title is provided on the website and in the metadata within the XML files.

16.3 Authenticity of Data

Section 204(a) of title 1, United States Code, which was enacted in 1947, relates to the printed version of the United States Code. It provides:

> In all courts, tribunals, and public offices of the United States, at home or abroad, of the District of Columbia, and of each State, Territory, or insular possession of the United States[,t]he matter set forth in the edition of the Code of Laws of the United States current at any time shall, together with the then current supplement, if any, establish prima facie the laws of the United States, general and permanent in their nature, in force on the day preceding the commencement of the session following the last session the legislation of which is included: *Provided, however*, That whenever titles of such Code shall have been enacted into positive law the text thereof shall be legal evidence of the laws therein contained, in all the courts of the United States, the several States, and the Territories and insular possessions of the United States.

In producing the United States Code, the Office of the Law Revision Counsel uses the same data to produce the printed version, the HTML version, and the XML version.

The HTML and XML files created by the Office of the Law Revision Counsel can be manipulated and enriched by others and offered to the public in new forms. Once data moves beyond the direct control of the Office of the Law Revision Counsel, the Office cannot vouch for its accuracy. Consumers should make their own determinations as to the reliability of data from other sources.