Companion Document for SPL Implementation[1]

Version 1

January, 2006

Contributors:
Lori Baranoski
Sandy Boyer
Pamela Budny
Glenda Casper
Michelle McGuinness
Yoshi Murata
Toni Stifano
Gunther Schadow
Keith Thomas
Craig Trautman
Robert Wallace

[1] This document is not an HL7 informative document.

Note: Questions or comments regarding this document should be directed to Binh Ta at Binh.Ta@fda.gov

Table of Contents

Record of Changes

History of changes:		
Version / Revision #	**Date of Change**	**Description of Changes**
1 / 0	December, 2005	First standalone release.
1 / 0	January, 2006	Incorporate comments from SPL WG.

1. Benefits of an XML approach to SPL

The benefits that accrue to all stakeholders in the drug regulatory process by adoption of Structured Product Labeling (SPL) make the business case for implementation of SPL overwhelming. The benefits of SPL derive from use of standard, universally adopted information standards such as XML, from the specific aspects of the SPL model for describing prescription drug content, and from adoption of an open standard for SPL

Specific benefits of SPL include:

- Stakeholders can develop automated, customized presentations of SPL content to reduce medical errors and improve patient care.
- SPL can be used by decision-support systems to improve patient care and reduce medical errors.
- Communication within industry (e.g., business-business communication) is facilitated by the ability to use SPL with XML-compliant tools or services.
- The use of well-defined vocabularies and coding systems within SPL enables uniform and unambiguous description of prescription drug products for data information systems
- XML-based transformations of SPL content by third-parties can increase the information and medical value of SPL documents while ensuring the integrity of FDA-approved labeling content.
- Internal business processes can repurpose the data contained within SPL through the use of XML-compliant databases.
- SPL insures data integrity of labeling content (and other SPL data elements) between industry and FDA databases.
- XML compliant consumer and health practitioner tools can use SPL in multiple settings.
- SPL can be readily integrated with HL7-based hospital information systems due to compliance with CDA architecture.
- Tools or objects that implement the standard can be utilized across all instances of SPL.
- Labeling content in SPL is not tied to proprietary tooling, allowing the development of SPL documents by different tools while retaining compatibility.
- SPL-associated XML stylesheets allow consistent presentation (rendering) of label content across different package inserts.
- Labeling changes and updates can be transmitted and immediately integrated into information systems.
- Non-rendered data elements (e.g., metadata about the SPL document or tagged content abstracted from the content of labeling) can be encompassed by the SPL standard to allow SPL data to integrate with other FDA and stakeholder systems.

These benefits represent only a fraction of the extensive benefits that accrue from adoption of SPL. The principles underlying these benefits are discussed below.

1.1 SPL as an Open Standard

SPL has been adopted as an open standard by Health Level Seven (HL7), an ANSI-accredited Standards Developing Organization (SDO). SPL development within HL7 has followed a well-defined, rigorous, ANSI-specified set of standards development procedures that has ensured consensus, openness and balance of interest among all participants in the development process. Participants in this process have

included representatives of the pharmaceutical industry, regulatory participants from the United States, and observers from other international agencies, domain and technical experts from HL7, health care providers, and software vendors.

The SPL document standard is defined by an XML schema; XML schemas are also an open standard supported by the World Wide Web consortium (W3C).[2] The vast penetration of XML technology in business and consumer products allows SPL to leverage the large number of tools available that support XML and other W3C standards[3]. The use of the HL7 information model and HL7 modeling conventions for the development of SPL ensures that SPL documents (also referred to as instances) are universally readable and unambiguously understood to all users of SPL documents and the developers of SPL tooling.[4]

As an open standard, SPL facilitates interoperability between systems. This provides substantial benefits to users through the reduced cost of software applications due to marketplace competition and the decreased need to customize applications around a proprietary standard. In addition, developers benefit substantially by full, immediate access to the information necessary to implement SPL systems. As noted earlier, adherence to a standard dramatically improves integration of different software applications and permits increased sophistication of software systems based on SPL.

The HL7 ANSI-mandated open development process ensures that all stakeholder concerns are addressed during the process of future enhancements to the standard. The publicly-open HL7 consensus process allows the contribution from a far greater array of participants than would be accessible to FDA in a more restricted standards-development approach. This permits a substantial number of individuals to contribute to the standard who otherwise would be excluded.

Overall, the use of an open standard such as SPL enables direct and predictable industry-to-FDA communication of package insert information, with subsequent direct communication of content to health care professionals and other third parties in a manner previously not possible with earlier FDA standards, e.g., PDF.

1.2 Machine Processability & Data Integration

As an XML standard SPL contains machine-readable context (i.e., tagged) information. Using XML enables the textual content to also be represented as data, i.e., "entries" in a data layer not visible to the human reader in the document but machine readable.

By structurally and semantically identifying content in this fashion, standardized data exchange between systems can occur. It permits business-to-business, industry-to-regulatory agency, and regulatory agency-to-public data transfer in a seamless manner. Knowledge of the sending or receiving system is not needed as long as there is conformance to the standard; in addition, since the definition of SPL (i.e., the XML schema for SPL) is an open standard, others systems can readily integrate structured

[2] See http://www.w3c.org/XML/.
[3] For example, XML is supported by all major Internet browsers.
[4] SPL embodies syntactic interoperability through the use of XML schemas and semantic interoperability (i.e., unambiguous interpretation of the XML tags in SPL) through derivation from the HL7 RIM model. See http://www.hl7.org for more information or http://www.healthcare-informatics.com/webinars/05_20_04.htm for a web-based discussion of HL7 and semantic interoperability.

information from SPL documents. During the exchange across systems, the need for additional transformation steps is eliminated.

Through use of XML, SPL can contain machine processable information separate from labeling content, i.e., 'data elements' related to or derived from label content but not strictly present in an identical format in current label content. For example, a tagged list of inactive ingredients may not appear as such in the content of a package insert but could be included as machine-processable data elements included by the SPL author. Data elements (or any tagged information) can serve as data for other systems separate from SPL, e.g., drug registration systems.[5]

The structure and machine readability of SPL also enables certain previously manual review functions within FDA and industry to become machine processable activities, e.g., identifying changes to the content of individual package inserts or comparing information across a broad range of SPL instances.

1.3 Machine Processability & Data Integration

SPL expresses the content of drug labeling separately from its presentation. A uniform presentation (or rendering) of content across all package inserts can be supported by FDA; this is an important benefit to potential end-users of SPL content.

Equally valuable is the ability to have customized presentations (renderings) for end users and within regulatory agencies and industry. Such renderings may exist for a single SPL instance or across multiple SPL documents. Customized rendering should permit use of the content of pharmaceutical labeling by both the health care industry and regulatory agencies in medically and regulatory important ways that were previously inaccessible. The way an SPL instance is rendered is determined by a stylesheet.

As an XML-based document, an SPL document retains human-readability in its native document form; sections of SPL can be easily comprehended, displayed, and edited without transformation or special rendering.

1.4 Streamlined Processing

SPL has significant advantages for internal processing of documents. XML-based documents permit separation of content for different purposes, e.g., for generating different regulatory submissions by pharmaceutical companies or for entry into different database systems by FDA, e.g., drug listing. SPL itself can be directly constructed via database systems. Modular use of document sections promotes consistency of content in company-generated documents and the reduction or elimination of redundancies; this similarly aids FDA in maintaining consistency across documents.

SPL facilitates the comparisons of document content (both narrative and data elements) across documents or between current and earlier versions of the same document. Document revision histories can be automated and viewed.

[5] This converse may also be supported, i.e., SPL could be populated with information from other information systems (rather than supplying information), ensuring data integrity across data systems. However, in either role (i.e., as receiver or supplier of 'data'), SPL serves as an envelope for this information.

The use of a structured standard would permit processes developed by one organization to be used seamlessly by others; for examples, objects to 'test' whether an SPL meets certain regulatory criteria (e.g., CBE) developed by a regulatory agency could be distributed for use by pharmaceutical companies. (Similar functionality could also be provided as web services.)

SPL also permits automated insertion and deletion of sections while confirming and maintaining the data integrity of non-altered content.

1.5 Scalability & Flexibility

The SPL format permits scalable implementation of document markups that may be of varying complexities. Such markups are flexible, i.e., naming or nesting of sections are not imposed upon the document, and permits integration of XML and non-XML sources with common metadata. Lastly, multiple data types (e.g. images, tables, and/or text) may be embedded into SPL.

As a schema-based standard, evolution of SPL can occur in a backward-compatible fashion that maintains compatibility with earlier versions of the standard. Similarly, transformations of XML documents can permit third parties to easily convert non-SPL defined information into clinical products and data sources while insuring integrity of the SPL content.

1.6 Tool independence

SPL as a non-proprietary open standard is compatible with a wide range of XML document creation applications while remaining independent of underlying storage mechanisms and/or document management systems. It is available to vendors who can apply the standard to any XML-based product they currently market. Vendors can build new tools or modify any current tools they may have to work with the standard.

Each business may define their own processes to incorporate SPL and choose the tools which will best complement the way they do their work, recognizing that SPL documents will be compatible with any other system.

The development of SPL as an open standard within the HL7 standards development process ensures that future development of the standard reflects a consensus process among industry, regulatory authorities, and the general health care community.

Use of SPL as an unambiguous, standardized markup (including use of standard controlled vocabularies and coding systems) should ultimately facilitate use of the content of a text document as data, permitting accurate database-like searches across SPL documents to improve patient care and safety. With data in SPL format, there may be opportunities to eliminate redundant data collection and manual processing that are used in other submissions, such as drug listing. Additionally, process efficiencies may surface with other organizations that use package insert information.

The potential benefits of SPL apply to every stakeholder who implements SPL. Even without the obvious benefit for use within FDA, benefit to other organizations that will serve as information-providers (such as the National Library of Medicine [NLM]), and information-consumers such as health care professionals and the general public make the transition to SPL by FDA necessary.

2. SPL Standard Stylesheet and FDA Implementation of Stylesheets[6]

2.1 Introduction

SPL is a form of document markup that is based on the HL7 clinical document architecture that will enable the standardization of the structure and semantics of the required content of drug product labeling as described in FDA regulations and guidance. While the SPL document is human readable, there are no formatting structures defined under the SPL-schema definition per se: formatting requirements specific for individual SPL instances are communicated via the *styleCode* attribute. The process of generating a formatted human readable display (or presentation of an SPL document [instance]) is referred to as rendering[7] ; for SPL, this means having default display characteristics modified when needed by the *styleCode* attribute value.

This SPL standard stylesheet is used for the display of an SPL document. It consists collectively of two files, an SPL extensible stylesheet language (XSL) file and an SPL cascading stylesheet (CSS) file. These files are W3C standard-compliant file types that define the baseline set of style conventions for viewing an SPL XML file in a browser, such as Microsoft Internet Explorer 6.0, Mozilla 1.7, Netscape Navigator 7.1, or Opera 7.54 or greater.[8] The XSL transforms an SPL document to a W3C standard-compliant XHTML document. The CSS defines cascading stylesheet classes for rendering the XHTML-transformed SPL document. Both files were created to be "browser-independent," meaning software products that support the W3C standards should render a valid SPL document similarly.[9]

As of this writing, the current version of the XSL is named 'spl-2a.1.xsl' and the CSS is 'spl-2a.1.css'. To minimize changes to various computer system, the name will be reverted back to its original name 'spl.xsl' in the future.

If additional cascading stylesheet are needed for the proper display of the FDA content of labeling that have not been adopted in the standard stylesheet, these will be included in a FDA-specific stylesheet that will add, or cascade, with the standard stylesheet. Currently there is not an FDA specific stylesheet; if so, this would be named as spl-fda.css.

Spl-2a.1.xsl and spl-2a.1.css are made available at http://www.fda.gov/oc/datacouncil/spl.html. This site should be checked regularly as the stylesheet may be updated more frequently than the implementation guide to address fixes, suggestions, etc.

---

[6] As noted earlier, the current 'SPL 2a' version that this implementation guide addresses uses an FDA-specific schema that has not been approved by HL7. SPL Release 1 contained an 'HL7 standard stylesheet (spl-1.0.xsl)'; the stylesheet to support this release, spl-2a.1.xsl , located at http:// www.fda.gov/oc/datacouncil/spl.html, is not equivalent to the HL7 stylesheet. Any reference to an 'HL7 standard stylesheet' in the discussion below should refer to the FDA-supplied stylesheet. It is the intention that if SPL Release 2 passes membership ballot, an HL7 standard stylesheet will be made available.

[7] In this discussion, 'presentation,' 'display,' and 'rendition' are used synonymously.

[8] Support for a W3C-complaint XML stylesheet transformation is not necessarily for properly rendering SPL. Server-based implementations of SPL may transform SPL via the standard XSL file and transmit the transformed XHTML document for rendering on a local device. In this model, only software support for rendering an XML/XHTML document with stylesheet classes is necessary for display.

[9] The end-user display/presentation is the XHTML transformed SPL document. Direct viewing of the SPL document, although possible via a specific CSS file, is not supported or recommended.

It should be noted that the css files are coded in the xsl file, i.e., in the actual coding of the SPL file the names of the active css files are unnecessary. Specifying the current xsl file in the SPL document will automatically select the css files current at the time the SPL is created.

2.2 SPL Stylesheet Components

The files collectively referred to as the SPL Standard Stylesheet (i.e., the separate xsl and css files) define the baseline set of style conventions for viewing an SPL-compliant XML file in a browser.

To illustrate how they are used, the following code represents a snippet from the gemcitabine demonstration SPL document cited earlier:

```
<component>
   <section>
      <id  root="A856E13A-8AA7-4C45-B378-97957CDFFC81"/>
        <code code="34090-1" codeSystem="2.16.840.1.113883.6.1" displayName="CLINICAL
PHARMACOLOGY SECTION"/>
      <title>CLINICAL PHARMACOLOGY</title>
      <component>
         <section>
            <id root="DD761815-06CE-47CA-A2D2-EFB2F24EFA44"/>
            <text>
                  <paragraph>MyDrugX demonstrated dose-dependent synergistic activity with penicllin
<content styleCode="italics">in vitro</content>. No effect of penicllin on MyDrugX accumulation was
observed. <content styleCode="italics">In vivo</content>, MyDrugX ABC<sub>197</sub> is a
nontoxic variant of diphtheria toxin isolated from cultures of <content
styleCode="italics">Corynebacterium diphtheriae</content> . In combination with macrolides against
the MX-1 human lung xenograft, minimal activity was seen with the NCI-H460 or NCI-H520
xenografts. MyDrugX was synergistic with fluconazole in the Lewis lung murine xenograft. Sequential
exposure to MyDrugX 18 hours before MyDrugY produced the greatest interaction.
                  </paragraph>
            </text>
         </section>
```

These excerpts of SPL-compliant XML are transformed by spl-2a.1.xsl into this HTML-compliant markup:

```
<a name="section-3"></a><p /><h1>CLINICAL PHARMACOLOGY</h1><a name="section-
3.1"></a><p /><p>MyDrugX demonstrated dose-dependent synergistic activity with penicllin <span>in
vitro</span>. No effect of penicllin on MyDrugX accumulation was observed. <span>In vivo</span>,
MyDrugX ABC<span class="Sub">197</span> is a nontoxic variant of diphtheria toxin isolated from
cultures of <span>Corynebacterium diphtheriae</span> . In combination with macrolides against the
MX-1 human lung xenograft, minimal activity was seen with the NCI-H460 or NCI-H520 xenografts.
MyDrugX was synergistic with fluconazole in the Lewis lung murine xenograft. Sequential exposure to
MyDrugX 18 hours before MyDrugY produced the greatest interaction. </p>
```

The HTML <p> element is rendered according to the CSS classes defined in the css file. For example, the <p> element is displayed by this class definition:[10]

```
p {
        text-indent: 0em;
        margin-top: 1.2ex;
        margin-bottom: 0ex;
        line-height: 2.2ex;
}
```

with default styles inherited from the body element:

```
body {
        background-color: white;
        color: black;
        font-family: Arial Unicode MS
        margin-left: 4em;
        margin-right: 4em;
        }
```

Therefore, in a compliant browser supporting CSS formatting, the snippet above would be presented as black text on a white background. There would be 4 cm left and right margins, and a line-height of 2.2 ex.

When displayed in a W3C complaint browser, the text would appear similar to the following[11]:

> **CLINICAL PHARMACOLOGY**
> MyDrugX demonstrated dose-dependent synergistic activity with penicllin *in vitro*. No effect of penicllin on MyDrugX accumulation was observed. *In vivo*, MyDrugX ABC_{197} is a nontoxic variant of diphtheria toxin isolated from cultures of *Corynebacterium diphtheriae*. In combination with macrolides against the MX-1 human lung xenograft, minimal activity was seen with the NCI-H460 or NCI-H520 xenografts. MyDrugX was synergistic with fluconazole in the Lewis lung murine xenograft. Sequential exposure to MyDrugX 18 hours before MyDrugY produced the greatest interaction.

The cascading stylesheet defines a default CSS class for each XHTML element resulting from the transformation of SPL to XHTML. Additional formatting choices are available to render SPL content in specific ways via the *styleCode* attribute. If necessary, these additional formatting choices (classes) would be available in the FDA SPL stylesheet for the display of the FDA content of labeling (see the next section for more details), For example, the following SPL snippet:

……

[10] Note: the unit 'em' is a measure of horizontal space (width) that depends on the currently effective font (approximately the width of the lower case m. The unit 'ex' is a measure of vertical space (height) that depends on the currently effective font (approximately the height of a lower case x). For example, the 4 em value indicates that for the margin-left and margin-right properties, the left and right margins will be four times the height of the default font rendering the document. The sans-serif value for the font-family property specifies the default non-serif font for the browser, e.g., Arial or Helvetica fonts.

[11] The margins are inaccurate in this example, having been reformatted for this specific document.

```
<tr styleCode="Botrule">
<td>&amp;#160;&amp;#160;Range</td>
<td>36 to 88</td>
<td>35 to 79</td>
<td/>
        <td>33 to 76</td>
        <td>35 to 75</td>
        <td/>
                        </tr>
                        <tr>
<td>Stage IIIA</td>
<td>7%</td>
<td>7%</td>
<td/>
<td>N/A</td>
<td>N/A</td>
<td/>
                        </tr>
                ….
```

will render as two rows in the following table:[12]

Table 2: Randomized Trials of Combination Therapy with Gemzar plus Cisplatin in NSCLC

Trial	28-day Schedule[a]			21-day Schedule[b]		
Treatment Arm	Gemzar/ Cisplatin	Cisplatin		Gemzar/ Cisplatin	Cisplatin/ Etoposide	
Number of patients	260	262		69	66	
Male	182	186		64	61	
Female	78	76		5	5	
Median age, years	62	63		58	60	
Range	36 to 88	35 to 79		33 to 76	35 to 75	
Stage IIIA	7%	7%		N/A	N/A	
Stage IIIB	26%	23%		48%	52%	
Stage IV	67%	70%		52%	49%	
Baseline KPS[c] 70 to 80	41%	44%		45%	52%	
Baseline KPS[c] 90 to 100	57%	55%		55%	49%	
Survival			p=0.008			p=0.18
Median, months	9.0	7.6		8.7	7.0	
(95%,C.I.) months	8.2, 11.0	6.6, 8.8		7.8, 10.1	6.0, 9.7	
Time to Disease Progression			p=0.009			p=0.015
Median, months	5.2	3.7		5.0	4.1	
(95%,C.I.) months	4.2, 5.7	3.0, 4.3		4.2, 6.4	2.4, 4.5	
Tumor Response	26%	10%	p<0.0001[d]	33%	14%	p<0.01[d]

[a] 28-day schedule – Gemzar plus cisplatin: Gemzar 1000 mg/m 2 on Days 1, 8, and 15 and cisplatin 100 mg/m 2 on Day 1 every 28 days; Single-agent cisplatin: cisplatin 100 mg/m 2 on Day 1 every 28 days.

[b] 21-day schedule – Gemzar plus cisplatin: Gemzar 1250 mg/m 2 on Days 1 and 8 and cisplatin 100 mg/m 2 on Day 1 every 21 days; Etoposide plus Cisplatin: cisplatin 100 mg/m 2 on Day 1 and I.V. etoposide 100 mg/m 2 on Days 1, 2, and 3 every 21 days.

[c] Karnofsky Performance Status.

[d] p-value for tumor response was calculated using the two-sided Fisher's exact test for difference in binomial proportions. All other p-values were calculated using the Logrank test for difference in overall time to an event.

In this example, specifying "Botrule" for the styleCode attribute to the first <tr> element creates a horizontal rule below the first row in the table (bottom rule)

2.3 Creation and Use of SPL Stylesheets

[12] In a browser supporting CSS1 and certain class properties available in CSS2.

To the extent possible, users should adopt the CSS classes available in the standard CSS for creating SPL documents. If specific requirements necessitate additional CSS classes to accommodate the proper display of a valid SPL document, HL7 permits authorized agencies to define a local cascading stylesheet that extends the existing standard stylesheet.

FDA is responsible for defining the local SPL practices for regulatory submissions in the United States. At the present time, no additional spl classes have been defined. When (and if) these become necessary, the additional cascading stylesheet available for SPL submissions in the U.S. (spl-fda-x.css) will be located at http://www.fda.gov/oc/datacouncil/spl.html. By cascading the FDA CSS with the SPL CSS, users would have access to a superset of CSS formatting classes beyond that available in spl-2a.1.css alone. The addition of local classes to the standard SPL CSS provides a means for supplementing the standard CSS in those circumstances for which this is necessary. Local classes are not intended to fundamentally alter the appearance of a rendered SPL document as defined by the standard CSS through, for example, extensively modifying common elements, such as the <paragraph> or <list> elements.

FDA will maintain its own CSS, if applicable. Use of local classes permits FDA to act independently from HL7 when the rapid addition of CSS classes is necessary to meet immediate needs. FDA will forward any added classes to the HL7 for consideration in adding to the SPL standard stylesheet with the goal of minimizing locally (i.e., FDA) defined classes. All added classes adopted in the SPL standard stylesheet will be removed from the locally defined stylesheets.

Formatting classes not present in the SPL standard stylesheet (e.g., spl-2a.css) or in FDA-specific additional classes defined in spl-fda-x.css will not be supported by FDA and should not be used in SPL documents submitted to FDA.

The SPL transformation stylesheet associated with HL7-balloted releases is maintained by HL7 and is designed to render an XHTML-compliant document formatted for human readability according to standard document constructs. It is anticipated that virtually all formatting accommodations necessary for rendering SPL will be accommodated though HL7 or realm-specific CSS classes as opposed to alterations in the transformation (xsl) stylesheet.

3. XML Primer

3.1 Introduction

XML (Extensible Markup Language) is a method for structuring electronically transmitted information; however, it does not specify the structure of the document itself. Instead, XML standards provide a mechanism that permits individual rules to be written for specific document types; one such document type is SPL or Structured Product Labeling. SPL is a specific type of XML document that is defined by a schema, or rules, to create this type of document. The schema for a specific type of document (or the document itself) is not part of the XML standard but must be written following rules of the XML standard. This concept is analogous to FDA regulations or a CTD, in that the regulations and CTD define the content and format a New Drug Application or a Biologics License Application, but do not specify the actual content of any specific application.

The schema or 'rules' list and define the components that make the document XML-compliant. The rules for a specific type of XML document define the structure for the information in that particular document type, e.g., an ‘address' for one specific document type may be specified to consist of a 'street name', a 'city', and a 'zip code'. The zip code may be further specified as only a 5 digit number.[13] In more technical terms, the rules of an XML document type, or the 'schema' for that particular type, unambiguously specify the 'syntax' that a document can take.

Information in an XML document may be intended for a display to be read by humans through web browsers, while other XML documents contain primarily 'data' that is meant for receipt and processing by another computer. In both cases XML provides the structure so both the 'sender' and the 'receiver' can understand the information, regardless of whether the 'receiver' is a human or another computer. XML documents, such as SPL contain both types of information in one document, i.e., the 'content of labeling' is meant for display and reading by humans, and 'data elements' which are meant to be 'interpreted' by computers and used in database systems where they may be subsequently converted for presentation to a human reader.

The following is an abbreviated introduction to basic XML terminology used in this implementation guide. Should you wish to learn more about XML, Internet or printed resources are readily available.

3.2 Elements and Tags

An element is the basic building block of an XML document. Element names are defined by the author of the schema for the document; in some cases they may be standardized (e.g., elements used for web pages, SPL elements based on the HL7 information model). The basic format of an XML element is: <tag>information</tag> (i.e., a set of tags wraps a particular piece of the information being structured), as shown in the following example:

```
<model>Ferrari 360 Modena</model>
```

Every XML element has a start tag (<model> in this example), an end tag </model>, and the element content ("Ferrari 360 Modena "). A start tag always has a left angle bracket (<), the element name (model), and a right angle bracket (>). The end tag is identical to the start tag with the addition of a slash (/) after the left angle bracket to indicate “close tag”.[14] Start tags and end tags are considered *markup*, i.e., the information in an XML document that is not meant to be displayed but identifies the structure of the document. Processing instructions, as well as comments (both described later in this section) are other examples of markup.

A schema consists of defined elements usually specified to be in a particular order, possibly with other elements nested inside them. (This is not always true: a schema can define a collection of elements below the root element that can exist in potentially different, or even random, orders.) The root element is the first element in an XML document. All the other elements in an XML document are nested inside the root element, also referred to as being children of the root element. Each child element may consist

[13] The rules could have just as easily specified a 9 digit zip code -- the form an XML document takes is dependent on the rules that were written for that document type by the author of the document type.

[14] Because of this, a left angle bracket (<) cannot be used in the content of an XML element since this will be misinterpreted as the start of a new element. To include a left angle bracket in SPL content an entity reference must be used, e.g., <model>Ford &#lt;car &#gt;</model> which would be displayed as "Ford <car>" in a browser.

of other elements, content information (i.e., text), or a combination of text and other elements. A special type of element is the empty element which may contain attributes (see below) but neither text nor other elements. An empty tag has a special form where there is no close tag (essentially a start and end tag combined). <emptyTag/> is an example of the form an empty tag would take.

Since the first element in an XML document is always the root element, e.g., <document>, the last tag is always the end tag for the root element, e.g., </document>. Element names (and therefore tags) are case sensitive, i.e., <document> is a different tag from <Document>.[15]

For example, if *car* is the "parent" or root element of an XML document, sub-elements or children of *car* might include *make, model* and *color* so, in XML markup, the car: a red 360 Modena Spider would appear as:

```
<car>
   <make>Ferrari</make>
   <model>360 Modena Spider</model>
   <color>RED</color>
   <vin vinNumber="12335566699"/>
</car>
```

car is an element which contains only other elements; these elements are the make, *model, color,* and *vin* elements. Another way of expressing this is that make, model, color and vin are children of the car element, or 'nested' within the car element. 'vin' is an empty element with the attribute vinNumber (see below).[16]

3.3 Attributes

Attributes are name/value pairs that are associated with a particular element; they may further define or modify the element. Attributes appear inside start tags or empty tags. Attributes must have a "value" (including a possible default value) assigned; this value will always be enclosed by quotes.[17] The name/value pair is always expressed as:

<element attributeName="value">text content</element>

For example, one method to identify a person's age via an attribute in an XML document could be:

[15] The distinction between elements and tags is frequently confusing and some people use them interchangeably. In the example just presented, *car* is an element, with child elements *make*, *model*, etc. Another element in the example below is *make*, where the value is Ford and is identified by <make>Ford</make>. A tag is specific markup in an XML document, e.g., </make> is an end tag for the element make. Elements may be referred to simply by their name whereas a tag always has brackets, e.g., <car> is the start tag for the car element. However, a common convention in narrative about XML is to surround element names with angle brackets (this convention was used in the SPL specification). If the example of *car* above were a complete XML document (rather than being part of another document), then *car* would be the root element of the document.

[16] An important point is illustrated by comparison of the element <model> in this and the earlier example. Although the element names are identical, they may have different meanings. In the earlier example, model referred to the manufacturer and model together. In the latter example, model was used only for the specific model, e.g., mustang. Although the rules, or schema, for a specific XML document specify the syntax of the document, they can't ensure the 'semantic' interpretation of a document, i.e., exactly what the elements mean. Elements should be unambiguously defined in a data dictionary or similar document; alternatively, an information model such as the HL7 information model will convey explicit semantics to element names when XML documents are created based on schemas created by HL7 methodology.

[17] A default value would be assigned in the schema, or rules, for that particular type of XML document.

<person age=”18”>Toni</person>

In this example, the element *person* has an attribute *age* with a value of 18.[18] In SPL, an example of an element containing an attribute could be:

<paragraph>This drug is <content styleCode="bold">contraindicated</content> with food.</paragraph>

The element *content* has the attribute *stylecode*, with a value of "bold". When displayed, this code would likely be displayed as follows:

This drug is **contraindicated** with food[19]

Empty elements may be used to hold attributes and values, e.g.,

<renderMultiMedia referencedObject="MM1"/>

In this example there is no content for the *renderMultiMedia* element since it is an empty element.

3.4 The Structure of an XML Document

XML documents generally consist of two parts, the 'processing instructions' which are at the start of an XML document, and the content or body of the document. This is not to be confused with the header and body of an SPL document which *together* form the body of an XML document.

Within the body of an XML document lays the root or parent element and all its “children” or sub-elements. This document hierarchy is often depicted using the metaphor of a tree. As described previously, elements can contain content, other elements (sub-elements or children) or both (mixed content). In XML, it is imperative that the children or sub-elements be nested properly.[20] Using the previous example:

<paragraph>this drug is <content styleCode="bold">contraindicated </content> with food.</paragraph>

The *paragraph* element is mixed, because it contains text content as well as other elements. The element *content* is nested within the paragraph.

The element *text* in the SPL example below is an illustration of an element that contains only other elements[21]:

[18] The reader may ask why this isn't an element, e.g., <person><name>Toni</name> <age>18</age></person>. How to make the decision whether a certain value should be specified as an attribute or element in the structure of an XML document is beyond the scope of this primer.

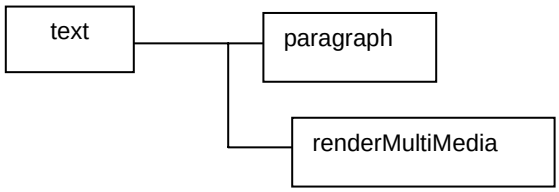
[19] The process whereby the word contraindicated is displayed as bolded text is a property of the stylesheet and rendering method that interprets 'bold' to mean 'bold the content of this element when it is displayed.' It is not a property of XML or the SPL document per se. The stylesheet could just as easily interpret 'bold' as 'important' and display it as italicized text; however, within SPL the semantic meaning of bold is to bold the element when displayed so that is what the stylesheet is designed to do.

[20] Nesting in this context refers to the order of the children of an element, i.e., that an actual XML document follows the structure permitted in the rules (or schema) for that document.

[21] Text is being used indifferent contexts in this discussion. Text refers to the content, or value, of an element (e.g., the text, or value, or content of the *paragraph* element in <paragraph>green olive</paragraph> is 'green olive'. *text* is also an element name in SPL as in <text><paragraph>green olive</paragraph></text>.

<text>
    <paragraph>This is just the paragraph content</paragraph>
    <renderMultiMedia referencedObject="MM5"/>
</text>>

In this example, the element *text* contains the elements *paragraph* and *renderMultimedia*. p*aragraph* and *renderMultimedia* are children or sub-elements of *text*. Using the tree metaphor to represent this relationship, we have:



3.5 XML Instructions and the Root Element

XML documents contain instructions that do not contain either content or data elements, but are used by the program processing the XML document for different purposes (e.g., a web browser opening an XML document). The XML declaration at the start of an XML document identifies the document as an XML document and specifies the version of the XML standard used when creating the document. Other processing instructions may be included that, for example, specify a transformation file that will aid in displaying the XML content. The following is an example of a declaration plus processing instructions in an XML document that specifies a transformation file:

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="spl-1.0.xsl "?>

The ? after the left angle bracket identifies each line of this markup as an XML instruction.

The root element of an XML document is a required and critical aspect of an XML document as it contains all the other elements that constitute the document (i.e., it defines the starting point and ending point for the document). The root element uses a special syntax and identifies the schema(s) that will be used to validate an XML document.

The root element has many optional attributes, the description of which is beyond the scope of this primer, and may often appear quite complex. The specific form of the root element to be used with SPL is specified in the implementation guide above. However, the general order of XML document is:

<?xml version="1.0" encoding="UTF-8"?>
<!-- ? Other Processing instructions… "? -->
<rootElement> <!-- start tag for root element of document -->
….
….
</rootElement> <!-- end tag for root element -->

3.6 XML Comments

Comments may exist almost anywhere in an XML document except within tags, i.e., inside markup. Comments are ignored by the XML processor. Comments are identified by the special character sequence <!-- (start comment) and --> (end comment). For example:

<example><!-- this is a comment --></example>

A more detailed example of valid XML comments is:

<text>
<!-- this was written to address Matt's concerns -->
<paragraph>drug x may cause…. </paragraph>
<paragraph><! -- another comment, but this time mixed in the content of an element -->drug x may also cause…. </paragraph>
</text>

The following would not be permissible because the comment is inside the tag:

<text>
<paragraph <!-- this is a comment --> styleCode="bold">this is invalid since the comment is in markup (in this case the start tag for paragraph)……</paragraph>
</text>

3.7 XML Schemas

A schema is a separate XML document that describes the rules, or syntax, of an XML document type (e.g., SPL). A schema defines the permissible elements and attributes in a document, the order in which they occur in an XML document, and possible values for these elements[22]. For example, a schema might permit:

<text><paragraph>this is OK</paragraph></text>

but not permit:

<paragraph><text>this is not OK</text></paragraph>

or just as easily, the schema could permit the reverse.

The schema also specifies the values that can be used for attributes, where the value is restricted, and can even specify the type of content that is acceptable for the value (the data type), e.g., specifying certain attributes to be dates and others only as numbers.

[22] Another means of specifying the rules, or the syntax of XML document is a DTD, or document type definition. Both are similar in that they define the rules of an XML document, (a DTD is actually a type of schema) but the details of each differ substantially. Discussion of the details of both schemas and DTDs is beyond the scope of this primer, although it is important to note that schemas rather that DTDs are used to define the structure of an SPL document.

XML schemas can be straightforward or extremely complex depending on the document type the schema is intended to address. Creating schemas is beyond the scope of this primer. However, it should be noted that authors of XML documents (e.g., SPL authors) would not alter a schema in creating a document; changes to a schema can only be made by the creator of the schema. Authors, however, may frequently refer to the schema to understand the structure of a document and determine what elements and attributes are permissible for that document type.

3.8 Well formed and Valid XML Documents

An XML document must be well-formed. A well-formed XML document follows the basic rules of every XML document; the rules are the XML standard and are independent of the schema used for a particular document type. These rules apply to all XML documents regardless of the schema, i.e., that all elements have a start tag and end tag (except 'empty tags'), that nesting is correct, that attribute values are in quotes, etc. For example:

<test><paragraph>paragraph content…</paragraph></test>

is well formed.

while <test><paragraph>paragraph content…</test>

is not well-formed since there is a missing end tag for the paragraph element.

Similarly,

<test><paragraph>paragraph content….</test></paragraph>

is not well-formed since the tags are improperly nested.

Only if a document is well-formed it can be tested for validity. Validity tests whether an XML document follows the rules of the specific schema for that document type; if it does, then, it is considered a valid document. For example, in SPL:

<text><paragraph>this is a paragraph</paragraph></text>

is valid but:

<paragraph><text>this is an invalid paragraph</text></paragraph>

is invalid since the order of elements in the latter example is not permitted by the SPL schema. However, both examples are well-formed XML.[23]

3.9 Tables

[23] There is no requirement that XML documents be created according to a schema, i.e., a well-formed XML document is an XML document even if not created according to a schema. However, without a schema, it would be difficult to insure that any two people creating similar documents did so according to the same rules (i.e., it would be difficult, if not impossible, to insure the documents created without using the same schema would be 'compatible').

Tables are not part of the XML standard; they are defined by the schema of a document type that includes tables. Tables can take different forms in different document types, i.e., the permitted children of the *table* element can de different in different schema. Because tables are so important in SPL, the structure of the *table* element in SPL is discussed briefly below.[24]

The table element in SPL is identified by the <table> tag. A table may contain rows and columns. The number of columns in a table is defined by the number of cells in a row. '*Tr'* identifies row elements and the cells in a row are *td* elements. An empty cell would be <td/>.

A simple table (including comments) could be:

```
<table>
        <tbody><!-- this defines the section as the body of a table rather than the header or footer
sections  -->
        <tr>  <!-- row 1  -->
                <td>col1 value for this row</td><td>col2 value for this row</td>
        </tr> <!-- end tag for row 1 -->
        <tr>  <!-- row 2  -->
                <td>col1 value for this row</td><td>col2 value for this row</td>
        </tr> <!-- end tag for row 2 -->
</tbody>
</table>
```

A heading row in a table is the *th* element. Use is identical to the *td cell* element, e.g.,

```
 <table>
        <thead>
<tr>  <!-- a heading row (there can be more than 1  -->
                        <th>column 1 heading</th><th>column 2 heading</th>
                </tr> <!-- end tag for heading row 1 -->
        </thead>
        <tbody>
<tr>  <!-- row 1  -->
                        <td>col1 value for row</td><td>col2 value for this row</td>
                </tr> <!-- end tag for row 1 -->
                <tr>  <!-- row 2  -->
                        <td>col1 value for row</td><td>col2 value for this row</td>
                </tr> <!-- end tag for row 2 -->
        </tbody>
</table>
```

When headings are used, it is regarded as good practice to divide the table into heading and body sections by use of the '*thead'* and '*tbody'* elements, as shown in the example above. (In SPL, the <thead> element is optional but <tbody> is required.)

[24] Although the *table* element in SPL is very similar to the *table* element in HTML, it is not identical. The SPL schema must be used for constructing tables in SPL.

If footnotes are to be included, they must be addressed as a separate section at the start of the table, even though they are displayed at the end of the table. For example:

```
<thead>
…..
</thead>
<tfoot>
        <tr>
                <td align="left" colspan="5">
<paragraph>   <caption>a</caption> Half-life for patients receiving a short infusion (&lt;70 min).
                        </paragraph>
                </td>
        </tr>
</tfoot>
<tbody>
…..
</tbody>
```

Presented below is an example that illustrates several aspects of an SPL table model that includes the use of table footnotes (*tfoot* element): (1) Cells in a row, header, or footnote can contain an align attribute, (2) cells can contain the colspan attribute, and (3) cells can contain *content, linkHtml*, or other constructs (see Section 6, Technical Note: CDA (SPL) Narrative Block DTD). Colspan allows a cell to span multiple columns. In a five column table, 5 'footnote' columns would make little sense; colspan allows the footnote to appear as anticipated, i.e., across all columns of a table in a separate row. Colspan is also frequently used in the body of a table where a particular row may need to combine columns for presentation. For example:

Column header for 1 and 2		Column head 3	Column head 4	Column head 5
Male[a]	**Female**			
4	5	100	300	500
6	7	388	887	543
P = .007		P=.34		
[a] This is an example of a footnote				

```
<table>
        <thead>
                <tr>
                        <th colspan=”2”>Column header for 1 and 2</th>
                        <th>Column head 3</th>
                        <th>Column head 4</th>
                        <th>Column head 5</th>
                </tr>
                <tr>
                        <th align="center" styleCode=”italics”>Male<sup>a</sup></th>
                        <th align="center"><content styleCode=”bold”>female</content></th>
                        <th colspan="3"></th>
```

```
            </tr>
        </thead>
        <tfoot>
            <tr>
                <td colspan=”5”><caption>a</caption>This is an example of a footnote</td>
            </tr>
        </tfoot>
        <tbody>
            <tr>
                <td>4</td>
                <td>5</td>
                <td>100</td>
                <td>300</td>
                <td>500</td>
            </tr>
            <tr>
                <td>6</td>
                <td>7</td>
                <td>388</td>
                <td>887</td>
                <td>543</td>
            </tr>
            <tr>
                <td align="center" colpsan="2">P=.007</td>
                <td colspan="3">P=.34</td>
            </tr>
        </tbody>
</table>
```

Note that the illustration of footnote construction in the preceding examples is legal and will work according to both the schema and the standard stylesheet; however, it is unnecessary and complicated. The usual practice would be to omit the <tfoot> element and create the footnote by using a <footnote> element in line in place of the ^a construct in the header row. The standard stylesheet would generate the <tfoot> element, including the footnote text, in the XHTML and insert the reference symbols from its repertoire of table footnote symbols.

The table has two header rows; in the second header row, the code that yields merging, or spanning the last 3 columns is: <th align="center">Male^a</th><th align="center">female</th><th colspan="3"></th>. For the fifth row, similar code is: <tr><td align="center" colpsan="2">P=.007</td><td colspan="3">P=.34</td></tr>.

The *rowspan* attribute exists for spanning rows of a table: use is similar to colpan, e.g., <td rowspan="2'>this spans 2 rows<td>.

A table may have a caption by use of the *caption* element, e.g., <table><caption>. This is the caption for this table</caption><thead>…… </table>.

Two additional elements are available for organizing the structure of SPL tables: *colgroup* , with the associated *col* element. Colgroup is used organize the columns of a table so they can be presented as a group, e.g., with a heavier vertical rule between groups. Colgroup can be used as follows:

```
<table>
        <colgroup>
                <col align="left"/>
                <col align="center"/>
        </colgroup>
        <colgroup>
                <col align="right"/>
                <col align=""left"/>
        </colgroup>
<thead>
                …….
                …….
        </thead>

        <tbody>
                …….
                …….
        </tbody>
</table>
```

This defines a table with 4 columns, 2 column groups with 2 columns each. The default alignment for each column is specified (optionally) by the align attribute. Note that the col element can appear without a colgroup element, e.g., <table><col/><col/>.

4. Technical Note: The Nature and Use of Identifiers in SPL

The term “identifier”, or “id” for short, has several definitions in the context of SPL. It may refer to:

(a) the <id> element defined in the schema;
(b) the <setId> element defined in the schema;
(c) the XML *ID* attribute defined for certain elements, e.g., <section> (written as “<… *ID*>”); or,
(d) an object identifier (OID) or universally unique identifier (UUID) , also called a globally unique identifier (GUID) used as a value of specified attributes in certain elements.

Throughout this section the generic term “ID” will not be used; the specific element or attribute names, or the terms OID or UUID will be used to make the meaning clear.

4.1 The <id> element

Purpose: The <id> element is primarily used to uniquely identify a single logical instance of an element in (or of) an SPL instance, by means of the value of the <id *root*> attribute.

The value of <id *root*> is defined to be a Unique Identifier (i.e., of data type *uid*; see Section 4.6 below) and must be *globally unique*.

4.1.1 <id *extension*> attribute not used

The schema defines an additional, optional attribute, <id *extension*>, but that attribute is *not* to be used in SPL Instances.

4.2 Declarative usage of the <id> element:

The primary usage of the <id> element is *declarative*; that is, where it carries a value uniquely identifying the instance of the element that encloses it. This usage is found in the following constructs where an <id> element is enclosed within:

(a) the <document> element [required],
(b) a <section> element [required],

4.2.1 <id root> attribute required

While the attribute <id *root*> is optional under the schema, the rule for the use of <id> elements in SPL is that wherever an <id> element is used it *must* carry a globally unique identifier in the <id *root*> value.

4.2.2 bit image identification

The globally unique identifier identifies a single logical instance of the element in question, i.e., a specific *bit image* of the element, not a specific copy. Any change, however trivial, to the content of that element creates a new instance of the element which must carry a different globally unique identifier when released by the originator of the instance.

4.2.3 identification only

The information carried by an <id *root*> value serves on only to distinguish that instance of the SPL element carrying that <id> from every other instance of an SPL element carrying an <id>; no other meaning can be inferred from the value, and no other use can be made of it.

4.3 Referential usage of the <id> element:

A secondary usage of the <id> element is referential where it carries a value identifying an element in another (i.e. external) SPL instance. This usage is found in the following constructs where the <id> element is enclosed within:

(a) a <relatedDocument> element.
(b) a <sectionReplaced> element.

The <id *root*> value is that of a <document><id> or <section><id> in another (i.e. external) instance.

4.4 The <setId> element

The <setId> element is used to uniquely identify a set of SPL document instances that form a sequence of versions of the same content of labeling. It carries an attribute, <setId *root*>, which functions here as in an <id> element, and carries the value of the <document><id> of the first document in the set.

4.4.1 <setId> value

While the attribute <setId *root*> is optional under the schema, it is required for SPL submitted to FDA. <setId root> and carries the value of the <document><id> of the first document in the set.

4.4.2 <setId root extension> attribute not used

The schema defines an additional, optional attribute <setId *extension*>, but that attribute is ***not*** to be used in SPL Instances.

4.4.3 Referential Usage

The <setId> element is always used referentially. The value in the <setId *root* > attribute must always contain the value of the root attribute of the <document><id> element of the first SPL instance of the sequence; thus the first SPL instance in a set carries a self-referencing <setId>).

4.5 The XML <… ID> attribute

A number of SPL elements may have an <… *ID*> attribute (or an attribute with another name of type *xs:ID*) that may be given a value to uniquely identify the particular element instance within the host SPL instance. The element bearing the <… *ID*> may be referenced by means of the same value in an <… *idref*> attribute (or an attribute with another name of type *xs:IDREFS*) in another element within the same instance.

One important use of *id/idref* style relationships in an SPL instance is to relate references to the rendering of graphics in the narrative text to the definition of the source file for the corresponding graphic held in a separate element. The <observationMedia *ID*> attribute is of type *xs:ID* and its value identifies the element, which carries the definition of a graphic file; the<renderMultiMedia *referencedObject*> attribute is of type xs:IDREFS and its value identifies the value of the <observationMedia *ID*> attribute in the appropriate <observationMedia> element.

Another use of an <… ID> attribute is as a target of internal document links expressed by the use of the <linkHTML> element. The <linkHTML> attribute carries the value of an <… ID> within the SPL instance, to which a browser will switch the user if they click on the content of a <linkHTML> element. The value of the <linkHTML> attribute is of type string, not type IDREFS, so the relationship is not an id/idref style and is not checked for validity by standard parsers.

4.5.1 no cross reference to <content> elements

The SPL standard also discusses a possible use for <content *ID*> and <reference *value*> relationships to "import" the content of a given <content> element appearing in the narrative text into an element in the structured data. This discussion should be regarded only as an illustration; the use of <content *ID*> and <reference *value*> relationships to represent narrative text inside structured data is **not** permitted in SPL instances at the present time.

4.6 Unique Identifiers

Unique Identifiers are of HL7 data type *uid*, and can be instantiated by values derived from one of two different schemes for creating unique identifiers, either:

(a) Universally Unique Identifiers (UUID's) also known as Globally Unique Identifiers (GUID's), created according to an IETF scheme; or,
(b) Object Identifiers (OID's), created according to the ISO/ITU Object Identification scheme.

Unique identifiers are to be used in the following constructs in SPL:

(c) as values of <… *root*> attributes, and
(d) as values of <… *codeSystem*> attributes.

In declarative <id> elements, a UUID/GUID is to be used as the value of the <id root> attribute.

In all other cases the unique identifiers are used referentially, and always take the value used externally to declare the unique identity of the thing referenced, whether that value is an OID or a UUID/GUID.

4.6.1 UUID/GUID's

Universally /Globally Unique Identifiers are generated algorithmically on a computer according to the specification set out in the Internet Engineering Task Force (IETF) Working Draft, "UUIDs and GUIDs", which can be found at http://www.ics.uci.edu/~ejw/authoring/uuid-guid/draft-leach-uuids-guids-01.txt

All computer operating environments have available utilities that implement the algorithm to generate UUID/GUID's with the property that each generated instance is reliably unique among all other UUID/GUID's generated by the same algorithm, regardless of where and when it is created.

The UUID/GUID is a 128-bit binary value, represented in an attribute value as a hexadecimal string (i.e. UTF-8 encodings of the digits 0-9, and letters A-F, each representing 4-bit segments of the value).

A GUID is usually formatted as a 36 character string, in quotes, with dashes between groups of hex digits as 8hex-4hex-4hex-4hex-12hex, e.g. "1E41AC26-B25D-492E-9699-AD9DD856F73A". This formatting is required where GUID's are used as values of the root attribute of an <id> or <setID> element in SPL. The hexadecimal values a to f inclusive are output as lower case characters, and are case insensitive on input.

In SPL, GUID's must be used as the root attribute value of declarative <id> elements.

4.6.2 OID's

The ISO/ITU-T Object Identifier (OID) scheme is a dotted notation (d.d.d …), where each "d" is a positive integer of one or more digits, representing a tree structure; that is, an OID represents a node in a tree in terms of numbered branches starting from the root of that tree.

By convention, *nationally registered* organization ID's will be used in SPL as the initial (left most) portion of the root attribute value to distinguish the numbering domains of each organization

For corporations registered in the US the left most portion of the root attribute will be "2.16.840.1.nnnnnn" where

2 = joint ISO/ITU-T assigned codes
16 = country assignments

840 = United States
1 = United States Companies/Organizations
Nnnnnn = Company ID

(the examples shown in the SPL specification use "2.16.1.840.1.113883", where 113883 is HL7)

Organizations may apply for a Company ID number from the *Organizational Name Registry* operated by the American National Standards Institute (ANSI). There is a fee for this service. Details can be found at
http://www.ansi.org/other_services/registration_programs/reg_org.aspx?menuid=10

Organizations may also apply for a Company ID number from HL7, which has authority from ANSI to distribute such ID's within its Company root. These are of the form
"2.16.840.1. 113883.3.nnnn" where

2 = joint ISO/ITU-T assigned codes
16 = country assignments
840 = United Stated
1 = US Companies
113883 = HL7
3 = HL7 Company/Organizational Registry
nnnn = Company ID

Details can be found at http://www.hl7.org/. OID's are used in SPL to identify code systems.

4.6.3 Use of Unique Identifiers in <… *codeSystem*> attributes

In an SPL instance external, predefined code systems must be used in association with the following elements:

(a) <code>
(b) <formCode>
(c) <routeCode>
(d) <languageCode>
(e) <confidentialityCode>
(f) <signatureCode>
(g) <translation>

Each of these elements has a <… *codeSystem*> attribute, which by HL7 definition takes an OID as a value. The particular code systems to be used in SPL are specified by the regulatory agency. The OID's that identify these are externally predefined, like the code system they reference; an alternative way to describe this is that each code system is identified by an externally registered OID. See section 5, Technical Note: The Nature and Use of Code Systems in SPL for more details on codes and code systems.

Since code systems are always externally defined, the use of <… *codeSystem*> attribute values is always referential.

By convention, code systems used under HL7 standards use an OID as the registered identifier of a code system.

4.7 Document and Section Identification

Both the <document> element and the <section> element each require a globally-unique instance identifier, where "instance" means that the entire digital content (as a bit string) of the enclosing element from the opening "<" of the start tag to the closing ">" of the end tag is unique; i.e. it constitutes a single, specific version of the content. This globally unique identification is expressed as the value of an <id *root*> attribute is to be a UUID/GUID as described in section 4.6.1.

Good practice suggests that instance identifiers be managed and assigned by automated processes integrated into organization's document management and/or document authoring software.

4.7.1 Identification Within Structured Data

Within structured data each <section> element must have an <id> element. These <id> elements have the same purpose and are subject to the same rules as noted above for <section> elements in narrative text. This applies to the <id> elements in the SPL Structured Data Section, each SPL Drug Product Description Subsection, and each Drug Product Component Description Subsection.

The schema also provides for an optional <manufacturedProduct><id> element, but it is *not* to be used in SPL instances.

4.8 Document Versioning

The SPL standard defines the <versionNumber> element as an optional part of the document header. Version numbering is a well-established and well-understood practice. <versionNumber> is assigned by FDA at the time the document is exported from the ELIPS system.

4.9 Summary of Identification Markup for Updates of Whole SPL Instances

FDA may publish at a later date a separate reference document addressing details of versioning and life cycle requirements for SPL. For the present, each time a new version of an SPL instance is sent to NLM (DailyMed), it will carry:

(a) a new <section><id *root*> attribute value for each changed section.
(b) a new <section><id *root*> attribute value for each new (inserted) section.
(c) a new <section><id *root*> attribute value for each section, if any, enclosing a changed or inserted section..
(d) a new <document><id *root*> attribute value for the entire SPL instance..
(e) a <setId> element with the value being the <document> <id> of the first released instance.
(f) If a section is unchanged from a previous submission in a new submission then the identifier/GUID should be unchanged as well (unless, as stated above, it encloses an altered subsection).

5. Technical Note: The Nature and Use of Code Systems in SPL

Code systems are used as attribute values in the following elements:

(a) <code>
(b) <formCode>
(c) <routeCode>
(d) <languageCode>
(e) <confidentialityCode>
(f) <signatureCode>
(g) <translation>

(Note that code systems used in the definition of abstract data types used in the schema, notably class codes and mood codes defined for XML attribute values, are not used explicitly in an SPL instance and are therefore not discussed in the Implementation Guide.)

In addition, the <code> element itself is conditioned by its enclosing element, and is defined in SPL in the following contexts:

(a) <document><code>
(b) <section><code>
(c) <manufacturedMedicine><code>
(d) <monitoringProgramEvent><code>
(e) <policy><code>
(f) <activeMoiety><code>
(g) <activeIngredientSubstance><code>
(h) <containerPackagedMedicine><code>

5.1 Required Attributes

All elements that use code systems are defined as carrying the following attributes:

(a) code (code value from the code system applicable to the specific occurrence)
(b) codeSystem (OID, see section 4.6.2)
(c) codeSystemName (string, formal name of system)
(d) codeSystemVersion (string, version used)
(e) displayName (string, text associated with code value, from code system)

While all of these attributes are optional under the schema, the rule for the use of the <code> element in SPL is that wherever it is used the following attributes *must* be populated correctly:

(a) code (code value from the code system applicable to the specific occurrence)
(b) codeSystem (OID, see section **Error! Reference source not found.**)
(c) displayName (string, text associated with code value, from code system)

The codeSystemName may be omitted as this is determined by (and therefore redundant) to the code and codeSystem attribute. codeSystemVersion attributes should be omitted if the code system used has no defined version number. (At present FDA is not requiring any codeSystemVersion attributes to be included, although this may change in the future.)

Although displayName may be specified, FDA will check and add/correct for accurate FDA Preferred Term from NCI Thesaurus before the SPL is sent to NLM.

5.2 Source of Code Systems

The code systems to be used are determined by the regulatory realm. Examples of code system used in SPL are given in the following subsection.

5.2.1 LOINC Codes

The Logical Observation Identifier Names and Codes (LOINC) is a code system maintained by the Regenstrief Institute, and used by HL7. For more details see: http://www.loinc.org/

The OID for the LOINC code system is "2.16.840.1.113883.6.1".

LOINC will be used a source of some codes to be used for SPL instances bound to US (FDA) rules. At present there are codes defined in LOINC for FDA labeling documents and for sections of US prescription labeling. For example, the <code> element in the context <document><code> could be written like this:

```
<code value="34391-3 "
        codeSystem="2.16.840.1.113883.6.1"
                codeSystemName="LOINC"
                codeSystemVersion="2.1
                displayName="HUMAN PRESCRIPTION DRUG LABEL" />
```

At present the codeSystemName and codeSystemVersion should not be included. For example, the <code> element in the context <section><code> for the Clinical Pharmacology section would be written like this:

```
<code value="34090-1"
        codeSystem="2.16.840.1.113883.6.1"
                displayName="CLINICAL PHARMACOLOGY SECTION" />
```

The full sets of LOINC codes (as well as code systems) currently applicable to SPL are listed in the Implementation Guide of which this document is a companion of.

5.3 Registration of External Vocabulary Domains with HL7

Future external vocabulary domains needed that are not currently registered with HL7 will be registered by the FDA. Implementers will not need to deal with this issue.

5.4 The Role of Regulatory Rules & Guidance

Each regulatory realm sets rules for how the intellectual content of labeling is to be expressed and organized. These rules act on the content of the narrative text and are largely independent of the SPL Specification and Implementation Guide, except insofar as they the order and nesting of sections.

However, each regulatory agency also specifies rules for the code systems (and their associated vocabulary of code values) that are to be used in an SPL instance for use in that regulatory realm, as follows:

(a) the code system to identify an SPL instance as an SPL document; expressed by the <*code codeSystem*> attribute value in the <document.><code> construct.

(b) the code system to identify section topics; expressed by the <code *codeSystem*> attribute value in the <section><code> construct used in narrative text.

Note that regulatory realm-specific rules for the ordering and/or nesting of sections are not part of the SPL specification, so there is no provision in the SPL specification or in the SPL implementation guide to express or to enforce such rules. The standard rendering follows the order of sections as they appear in an instance and displays the contents of title elements within sections, if present, regardless of the codes that may or may not be assigned.

(c) the code systems to identify languages, confidentiality level and signature systems; expressed by the <code *codeSystem*> attribute value in the following elements:

<languageCode>
<confidentialityCode>
<signatureCode>

(d) the codes to be used in structured data; expressed by the <… *codeSystem*> attribute of the <code> element in the following contexts:

<manufacturedMedicine><formCode>
<policy><code>
<substance><code>
<activeMoiety><code>
<containerPackagedMedicine><formCode>
<containerPackagedMedicine><code>
<characteristic><code>

and by the <… *codeSystem*> attribute value in the following elements:

<formCode>
<routeCode>
<translation>

The full set of

6. Technical Note: CDA (SPL) Narrative Block DTD

The following is the DTD for the narrative block (text) sections of SPL. It is identical to the current Clinical Document Architecture model.

```
<!ENTITY % textAtts "
  ID ID #IMPLIED
  lang NMTOKEN #IMPLIED
 styleCode NMTOKENS #IMPLIED">

<!ELEMENT text (#PCDATA | content | linkHtml | sub | sup | br | footnote | footnoteRef |
renderMultiMedia | paragraph | list | table)*>

<!ELEMENT content (#PCDATA | content | linkHtml | sub | sup | br | footnote | footnoteRef |
renderMultiMedia)*>
<!ATTLIST content
        %textAtts;
        revised (insert | delete) #IMPLIED>

<!ELEMENT linkHtml (#PCDATA | footnote | footnoteRef)*>
<!ATTLIST linkHtml
        name CDATA #IMPLIED
        href CDATA #IMPLIED
        rel CDATA #IMPLIED
        rev CDATA #IMPLIED
        title CDATA #IMPLIED
        %textAtts; >

<!ELEMENT sub (#PCDATA)>

<!ELEMENT sup (#PCDATA)>

<!ELEMENT br EMPTY>

<!ELEMENT footnote (#PCDATA | content | linkHtml | sub | sup | br | renderMultiMedia | paragraph |
list | table)*>
<!ATTLIST footnote
        %textAtts; >

<!ELEMENT footnoteRef EMPTY>
<!ATTLIST footnoteRef
        %textAtts;
        IDREF IDREF #REQUIRED>

<!ELEMENT renderMultiMedia (caption?)>
<!ATTLIST renderMultiMedia
        referencedObject IDREFS #REQUIRED
        %textAtts; >

<!ELEMENT paragraph (#PCDATA | caption | content | linkHtml | sub | sup | br | footnote | footnoteRef
| renderMultiMedia)*>
```

```
<!ATTLIST paragraph
        %textAtts; >

<!ELEMENT list (caption?, item+)>
<!ATTLIST list
        %textAtts;
        listType (ordered | unordered) "unordered">

<!ELEMENT item (#PCDATA | caption | content | linkHtml | sub | sup | br | footnote | footnoteRef |
renderMultiMedia | paragraph | list | table)*>
<!ATTLIST item
        %textAtts; >

<!ENTITY % cellhalign "align      (left|center|right|justify|char) #IMPLIED
  char       CDATA          #IMPLIED
  charoff    CDATA          #IMPLIED">

<!ENTITY % cellvalign "valign     (top|middle|bottom|baseline) #IMPLIED">

<!ENTITY % Tframe "(void|above|below|hsides|lhs|rhs|vsides|box|border)">

<!ENTITY % Trules "(none | groups | rows | cols | all)">

<!ENTITY % Scope "(row|col|rowgroup|colgroup)">

<!ELEMENT table (caption?, (col* | colgroup*), thead?, tfoot?, tbody+)>

<!ELEMENT caption (#PCDATA | linkHtml | sub | sup | footnote | footnoteRef)*>

<!ELEMENT thead (tr)+>

<!ELEMENT tfoot (tr)+>

<!ELEMENT tbody (tr)+>

<!ELEMENT colgroup (col)*>

<!ELEMENT col EMPTY>

<!ELEMENT tr (th | td)+>

<!ELEMENT th (#PCDATA | content | linkHtml | sub | sup | br | footnote | footnoteRef |
renderMultiMedia)*>

<!ELEMENT td (#PCDATA | content | linkHtml | sub | sup | br | footnote | footnoteRef |
renderMultiMedia | paragraph | list)*>
```

```
<!ATTLIST table
        %textAtts;
        summary CDATA #IMPLIED
        width CDATA #IMPLIED
        frame %Tframe; #IMPLIED
        rules %Trules; #IMPLIED>

<!ATTLIST caption
        %textAtts; >

<!ATTLIST colgroup
        %textAtts;
        span CDATA "1"
        width CDATA #IMPLIED
        %cellhalign;
        %cellvalign; >

<!ATTLIST col
        %textAtts;
        span CDATA "1"
        width CDATA #IMPLIED
        %cellhalign;
        %cellvalign; >

<!ATTLIST thead
        %textAtts;
        %cellhalign;
        %cellvalign; >

<!ATTLIST tfoot
        %textAtts;
        %cellhalign;
        %cellvalign; >

<!ATTLIST tbody
        %textAtts;
        %cellhalign;
        %cellvalign; >

<!ATTLIST tr
        %textAtts;
        %cellhalign;
        %cellvalign; >

<!ATTLIST th
        %textAtts;
        abbr CDATA #IMPLIED
```

```
        axis CDATA #IMPLIED
        headers IDREFS #IMPLIED
        scope %Scope; #IMPLIED
        rowspan CDATA "1"
        colspan CDATA "1"
        %cellhalign;
        %cellvalign; >

<!ATTLIST td
        %textAtts;
        abbr CDATA #IMPLIED
        axis CDATA #IMPLIED
        headers IDREFS #IMPLIED
        scope %Scope; #IMPLIED
        rowspan CDATA "1"
        colspan CDATA "1"
        %cellhalign;
        %cellvalign; >
```