Appendix: XML Theory

Latest update: 24 July 2002.

History of released versions: 24 July 2002 (Initial released version)

For any remarks regarding this document, please send an e-mail to us: frank@vandervelpen.com

Author: Frank vandervelpen
I want to thank Wil van Antwerpen [Antwise Solutions] for the critical comments on our work.

Goal

This document is written to give a brief introduction on XML and related topics. This document does not have the intention to be complete but to give an overall definition on some terminology. This document can be used as an appendix to documents describing some XML based projects.

Line separated, EDI, XML…

Over the time, we had several methods to exchange information between several computer systems. However, since the introduction of the internet and the business to business applications, the need for a special designed open source format was advised (http://www.w3c.org/xml). We can support the project with several definitions on data exchange, but we plan on describing the XML format. Once it is properly described, other formats can use it as a referential analysis too.

XML Facts

What is XML (The Extensible Markup Language)?

XML, XLink, Namespace, DTD, Schema, CSS, XHTML,... If you are new to XML, it may be hard to know where to begin. This summary in 10 points attempts to capture enough of the basic concepts to enable a beginner to see the forest through the trees. And if you are giving a presentation on XML, why not start with these 10 points? They are hereby offered for your use.

1. XML is a method for putting structured data in a text file.

For "structured data" think of such things as spreadsheets, address books, configuration parameters, financial transactions, technical drawings, etc. Programs that produce such data often also store it on disk, for which they can use either a binary format or a text format. The latter allows you, if necessary, to look at the data without the program that produced it. XML is a set of rules, guidelines, conventions, whatever you want to call them, for designing text formats for such data, in a way that produces files that are easy to generate and read (by a computer), that are unambiguous, and that avoid common pitfalls, such as lack of extensibility, lack of support for internationalization/localization, and platform-dependency.

2. XML looks a bit like HTML but isn't HTML

Like HTML, XML makes use of tags (words bracketed by '<' and '>') and attributes (of the form name="value"), but while HTML specifies what each tag & attribute means (and often how the text between them will look in a browser), XML uses the tags only to delimit pieces of data, and leaves the interpretation of the data completely to the application that reads it. In other words, if you see "<p>" in an XML file, don't assume it is a paragraph. Depending on the context, it may be a price, a parameter, a person, a p... (b.t.w., who says it has to be a word with a "p"?)

3. XML is text, but isn't meant to be read

XML files are text files, as I said above, but even less than HTML are they meant to be read by humans. They are text files, because that allows experts (such as programmers) to more easily debug applications, and in emergencies, they can use a simple text editor to fix a broken XML file. But the rules for XML files are much stricter than for HTML. A forgotten tag, or a an attribute without quotes makes the file unusable, while in HTML such practice is often explicitly allowed, or at least tolerated. It is written in the official XML specification: applications are not allowed to try to second-guess the creator of a broken XML file; if the file is broken, an application has to stop right there and issue an error.

4. XML is a family of technologies

There is XML 1.0, the specification that defines what "tags" and "attributes" are, but around XML 1.0, there is a growing set of optional modules that provide sets of tags & attributes, or guidelines for specific tasks. There is, e.g., Xlink (still in development as of November 1999) which describes a standard way to add hyperlinks to an XML file. XPointer & XFragments (also still being developed) are syntaxes for pointing to parts of an XML document. (An Xpointer is a bit like a URL, but instead of pointing to documents on the Web, it points to pieces of data inside an XML file.) CSS, the style sheet language, is applicable to XML as it is to HTML. XSL (autumn 1999) is the advanced language for expressing style sheets. It is based on XSLT, a transformation language that is often useful outside XSL as well, for rearranging, adding or deleting tags & attributes. The DOM is a standard set of function calls for manipulating XML (and HTML) files from a programming language. XML Namespaces is a specification that describes how you can associate a URL with every single tag and attribute in an XML document. What that URL is used for is up to the application that reads the URL, though. (RDF, W3C's standard for metadata, uses it to link every piece of metadata to a file defining the type of that data.) XML Schemas 1 and 2 help developers to precisely define their own XML-based formats. There are several more modules and tools available or under development. Keep an eye on W3C's technical reports page.

5. XML is verbose, but that is not a problem

Since XML is a text format, and it uses tags to delimit the data, XML files are nearly always larger than comparable binary formats. That was a conscious decision by the XML developers. The advantages of a text format are evident (see 3 above), and the disadvantages can usually be compensated at a different level. Disk space isn't as expensive anymore as it used to be, and programs like zip and gzip can compress files very well and very fast. Those programs are available for nearly all platforms (and are usually free). In addition, communication protocols such as modem protocols and Hypertext Transfer Protocol 1.1 (HTTP/1.1 the core protocol of the

Web) can compress data on the fly, thus saving bandwith as effectively as a binary format.

6. XML is new, but not that new

Development of XML started in 1996 and it is a W3C standard since February 1998, which may make you suspect that this is rather immature technology. But in fact the technology isn't very new. Before XML there was SGML, developed in the early '80s, an ISO standard since 1986, and widely used for large documentation projects. And of course HTML, whose development started in 1990. The designers of XML simply took the best parts of SGML, guided by the experience with HTML, and produced something that is no less powerful than SGML, but vastly more regular and simpler to use. Some evolutions, however, are hard to distinguish from revolutions... And it must be said that while SGML is mostly used for technical documentation and much less for other kinds of data, with XML it is exactly the opposite.

7. XML is license-free, platform-independent and well-supported

By choosing XML as the basis for some project, you buy into a large and growing community of tools (one of which may already do what you need!) and engineers experienced in the technology. Opting for XML is a bit like choosing SQL for databases: you still have to build your own database and your own programs/procedures that manipulate it, but there are many tools available and many people that can help you. And since XML, as a W3C technology, is license-free, you can build your own software around it without paying anybody anything. The large and growing support means that you are also not tied to a single vendor. XML isn't always the best solution, but it is always worth considering

Some XML Samples.

Accounting Plan.

```
<vaccount>
        <row>
                <code><![CDATA[
70
]]></code>
                <code_group><![CDATA[
9
]]></code_group>
                <analytic_yn><![CDATA[

]]></analytic_yn>
                <code_cancel><![CDATA[

]]></code_cancel>
                <keyname><![CDATA[
#
]]></keyname>
                <descr1><![CDATA[
omzet
]]></descr1>
                <descr2><![CDATA[
chiffresd'affaires
]]></descr2>
                <descr3><![CDATA[
chiffresd'affaires
]]></descr3>
                <descr4><![CDATA[
omzet
]]></descr4>
                <sign_debet_cred><![CDATA[

]]></sign_debet_cred>
                <detail_global><![CDATA[

]]></detail_global>
```

This document replaces all prior versions

```
		<statistics_yn><![CDATA[

]]></statistics_yn>
		<quantity_yn><![CDATA[

]]></quantity_yn>
		<code_vat><![CDATA[
0
]]></code_vat>
		<account_no_ded><![CDATA[

]]></account_no_ded>
		<pc_deductable><![CDATA[
0
]]></pc_deductable>
		<currency_short><![CDATA[

]]></currency_short>
		<createdate><![CDATA[
]]></createdate>
		<createtime><![CDATA[

]]></createtime>
		<createuser><![CDATA[

]]></createuser>
		<modifydate><![CDATA[
]]></modifydate>
		<modifytime><![CDATA[

]]></modifytime>
		<modifyuser><![CDATA[

]]></modifyuser>
		<delete_flag><![CDATA[

]]></delete_flag>
		<action_flag><![CDATA[

]]></action_flag>
		<reserved1><![CDATA[

]]></reserved1>
		<reserved2><![CDATA[

]]></reserved2>
	</row>
	<row>
		<code><![CDATA[
700000
]]></code>
		<code_group><![CDATA[
2
]]></code_group>
		<analytic_yn><![CDATA[
x
]]></analytic_yn>
		<code_cancel><![CDATA[

]]></code_cancel>
		<keyname><![CDATA[
verkopen
]]></keyname>
		<descr1><![CDATA[
verkopenvanhandelsgoederen
]]></descr1>
		<descr2><![CDATA[
ventesdemarchadises.
]]></descr2>
		<descr3><![CDATA[
ventesdemarchadises
]]></descr3>
		<descr4><![CDATA[
verkopenvanhandelsgoederen
]]></descr4>
		<sign_debet_cred><![CDATA[

]]></sign_debet_cred>
		<detail_global><![CDATA[

]]></detail_global>
		<statistics_yn><![CDATA[

]]></statistics_yn>
		<quantity_yn><![CDATA[
]]></quantity_yn>
          <code_vat><![CDATA[
4
]]></code_vat>
          <account_no_ded><![CDATA[
700000
]]></account_no_ded>
          <pc_deductable><![CDATA[
100
]]></pc_deductable>
          <currency_short><![CDATA[

]]></currency_short>
          <createdate><![CDATA[
]]></createdate>
          <createtime><![CDATA[

]]></createtime>
          <createuser><![CDATA[

]]></createuser>
          <modifydate><![CDATA[
31/10/2000
]]></modifydate>
          <modifytime><![CDATA[
103415
]]></modifytime>
          <modifyuser><![CDATA[
administrator
]]></modifyuser>
          <delete_flag><![CDATA[

]]></delete_flag>
          <action_flag><![CDATA[
x
]]></action_flag>
          <reserved1><![CDATA[

]]></reserved1>
          <reserved2><![CDATA[

]]></reserved2>
     </row>
</vaccount>
```

In the above sample, we have generated a XML document for 3 records in the database of the accounting plan. With this sample, we can define some extra basics in the XML topology.

Some XML related theory.

Because in 2001, the awareness of XML related solutions is increased and its implementations are few, we think it is best to give some overview of the newest elements on this brand new methodology. XML and its related issues are supposed to be vendor independent and are regulated under the wings of the world wide web consortium. (http://www.w3c.org)

SGML.

1. SGML uses a document type definition (DTD) to define the structure of the document. The DTD specifies the elements and attributes that can be used within the document and specifies what characters will be used to mark the text. In SGML, you can use brackets (<>), dashes (-), or any other character to mark up your document as long as the special character is properly defined in the DTD.
2. Disadvantage of SGML is the difficulty to learn and adapt people to it. It has a specification manual of 500 pages and a set of 100 pages in annex. It is also

to much overkill for the three reasons why we want to use XML. We will propose a combination of XML and DTDs.

XML XSL DNS.

1. XML is data aware, XSL (Extensible Stylesheet Language) is presentation aware. XSL describes how the XML data should be presented.
2. DNS: Digital Nervous System.

How XML and DNS can help your company.

DNS is a Digital nervous system.

1. Provide scalability.
2. Facilitate Internet Use.
3. Customer feedback.
4. Business partner communication.

XML Basics

1. In the XML sample, we see that information is seeded over several lines. However, this was generated by the parser. (Microsoft Internet Explorer) In general, an XML document is a continuous document without carriage return or line feed characters in it.
2. Any XML document that meets the basic rules as defined by the XML specification is called a *well-formed XML document*. An XML document can be checked to determine whether it is *well formed*—that is, whether the document has the correct structure (syntax).
3. When an XML document meets the rules defined in the DTD, it is called a *valid XML document.* (DTD: document type definiton)
4. *Schemas* are similar to DTDs, but they use a different format. DTDs and schemas are useful when the content of a group of documents shares a common set of rules.
5. XML gives you the opportunity to create messages in standard forms.
6. XML separates data from presentation.
7. XML gives you the possibility to call methods behind firewalls and between different platforms.
8. XML describes the contents but not the layout.
9. XML is tagged based: every tag begins with <descr> and an end tag </descr> , where descry is the name of the tag
10. You can define the character encoding used by encoding = “UTF-8”.
11. Tags should only be written in lowercase to be compatible with the XHTML definitions (upcoming format)
12. You have some predefined tokens that can’t be used unless they actually are part of a CDATA section. These tokens are also referred to as *Predefined*

entieis (see later in this manual)

➔ &lt	<
➔ &gt	>
➔ &amp	&
➔ &apos	’
➔ &quot	”

13. The most basic components of an XML document are *elements, attributes,* and *comments*.

14. Every tag should have an end-tag.
    Correct is: <tag> <element> … </element></tag>
    Incorrect is: <tag> <element> … </tag> </element>

15. All attribute values should be written between single or double quotes.
    i.e. <element id=”myvalue”>

16. The designer of the XML document defines the structure of the document and the mark-up elements.

17. XML must be properly written to get interpreted (not with HTML) This means that every tag must have an end tag.
    Elements can be nested. I.e.

```
<Patient>
   <PatientName>John Smith</PatientName>
   <PatientAge>108</PatientAge>
   <PatientWeight>155</PatientWeight>
</Patient>
```

18. -Names consist of one or more “no space” characters. If a name has only one character, that character must be a letter, either uppercase (A-Z) or lowercase (a-z).
    -A name can only begin with a letter or an underscore.
    -Beyond the first character, any character can be used, including those defined in the Unicode standard.
    -Element names are case sensitive.

19. An attribute is a mechanism for adding descriptive information to an element.
    <PatientWeight unit="KG">81</PatientWeight>
    or
    <PatientWeight unit="KG"/> (=empty element)

20. Comments are descriptions embedded in an XML document to provide additional information about the document. They are placed between the tags <!-- and -->
    <!-- Comment text -->

21. An element without a value can be written as *<sample/>*, a.k.a. singleton

22. Some HTML tools can’t interpret the singletons. Therefore, you might want to change the code towards <sample> … </sample>. Another solution is to add a comment field.

Well Formed.

Well-formed XML documents can be created by using elements, attributes, and comments. These components define content within the document. Using these definitions, applications can be created that will manipulate the content

To be well formed, your XML document must meet the following requirements:

1. The document must contain a single root element.
2. Every element must be correctly nested.
3. Each attribute can have only one value.
4. All attribute values must be enclosed in double quotation marks or single quotation marks.
5. Elements must have begin and end tags, unless they are empty elements.
6. Empty elements are denoted by a single tag ending with a slash (/).
7. Isolated markup characters are not allowed in content. The special characters <, &, and > are represented as &gt, &amp, &lt in content sections.
8. A double quotation mark is represented as &quot, and a single quotation mark is represented as &apos in content sections.
9. The sequence <[[and]]> cannot be used.
10. If a document does not have a DTD, the values for all attributes must be of type CDATA by default.

XML Declaration

```
<?xml version="version_number" encoding="encoding_declaration"
    standalone="standalone status"?>
```

11. The version attribute is the version of the XML standard that this document complies with. The encoding attribute is the Unicode character set that this document complies with. Using this encoding, you can create documents in any language or character set. The standalone attribute specifies whether the document is dependent on other files (standalone = "no") or complete by itself (standalone = "yes").

SOAP

1. SOAP solves the problems concerning addressing calling methods behind a firewall. They do this by using XML and HTML to pass messages to methods behind the firewall. It is embedded in a Windows DNA system.
2. It can connect to other components on other systems.
3. This will be a component that we might introduce at a later time.

Our implementation

Our rules.

1. All used XML documents should be *Well Formed*.

XML tools

This is a non exclusive list of tools to use when working with XML files.

1. Plain text editor i.e. Multi-Edit, VIM or Notepad.
2. Internet Explorer 5 and newer.
3. XML Notepad (part of MS BizTalk Server)
   http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnxml/html/xmlpaddownload.asp
4. XMLSPY http://www.xmlspy.com
5. XML Pro http://www.vervet.com

DTD

Document Type Definition

The DTD can be used to verify that a set of XML documents is created according to the rules defined in the DTD by checking the validity of the documents.

- Every element used in your XML documents has to be declared by using the <!ELEMENT> tag in the DTD. The format for declaring an element in a DTD is shown here:
  <!ELEMENT ElementName Rule>
  The Rule component defines the rule for the content contained in the element. These rules define the logical structure of the XML document and can be used to check the document's validity. The rule can consist of a generic declaration and one or more elements, either grouped or unordered.
- The Predefined Content Declarations
  Three generic content declarations are predefined for XML DTDs: PCDATA, ANY, and EMPTY.
  - *PCDATA*
    The PCDATA declaration can be used when the content within an element is only text—that is, when the content contains no child elements. Our sample document-snippet contains several such elements, including *title*, *a*, *h1*, and *b*. These elements can be declared as follows. (The pound sign identifies a special predefined name.)
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT a (#PCDATA)>
    <!ELEMENT h1 (#PCDATA)>
    <!ELEMENT b (#PCDATA)>
    NOTE
    PCDATA is also valid with empty elements.
  - *ANY*
    The ANY declaration can include both text content and child elements. The *html* element, for example, could use the ANY declaration as follows:
    <!ELEMENT html ANY>
    The ANY declaration allows any content to be marked by the element tags, provided the content is well-formed XML. Although this flexibility might seem useful, it defeats the purpose of the DTD, which is to define the structure of the XML document so that the document can be validated. In brief, any element that uses ANY cannot be checked for validity, only for being well formed.

- ○ _EMPTY_
  It is possible for an element to have no content—that is, no child elements or text. The img element is an example of this scenario. The following is its definition:
  <!ELEMENT img EMPTY>

- _ONE or MORE elements_
  This (head, body) declaration signifies that the html element will have two child elements: head and body. You can list one child element within the parentheses or as many child elements as are required. You must separate each child element in your declaration with a comma.
  For the XML document to be valid, the order in which the child elements are declared must match the order of the elements in the XML document. The comma that separates each child element is interpreted as followed by; therefore, the preceding declaration tells us that the html element will have a head child element followed by a body child element.

- _REOCCURENCE_
  XML Element Markers

Marker	Meaning
?	The element either does not appear or can appear only once (0 or 1).
+	The element must appear at least once (1 or more).
*	The element can appear any number of times, or it might not appear at all (0 or more).

Putting no marker after the child element indicates that the element must be included and that it can appear only one time.
The head element contains an optional base child element.
Here will follow some sample markers.

To declare this element as optional, modify the preceding declaration as follows:
<!ELEMENT head (title, base?)>
The body element contains a basefont element and an a element that are also optional. In our example, the table element is a required element used to format the page, so you want to make table a required element that appears only once in the body element. You can now rewrite the Body element as follows:
<!ELEMENT body (basefont?, a?, table)>
The table element can have as many rows as are needed to format the page but must include at least one row. The table element should now be written as follows:
<!ELEMENT table (tr+)>
The same conditions hold true for the tr element: the row element must have at least one column, as shown here:
<!ELEMENT tr (td+)>
The a, ul, and ol elements might not be included in the p element, or they might be included many times, as shown here:
<!ELEMENT p (font+, img, br, a, ul*, ol*)>*
Because the br element formats text around an image, the img and br tags should always be used together.

- *Grouping child objects*
  - You can also group child objects.
    <!ELEMENT p (font, (img, br?)*, a*, ul*, ol*)>*
    or
    <!ELEMENT p (font, (img, br?)*, a*, ul*, ol*)+>*
    or
    <!ELEMENT p (font | (img, br?) | a | ul | ol)+>
  - You can separate the elements by a comma (,) or with a pipe (|). If you use the pipe, it indicates that one or the other child element will be included but not both. The latest sample defines an unsorted set of child elements.

The !ATTLIST Statement

Every element can have a set of attributes associated with it. The attributes for an element are defined in an !ATTLIST statement. The format for the !ATTLIST statement is shown here:

<!ATTLIST ElementName AttributeDefinition>

ElementName is the name of the element to which these attributes belong.

AttributeDefinition consists of the following components:

AttributeName AttributeType DefaultDeclaration

AttributeName is the name of the attribute. *AttributeType* refers to the data type of the attribute. *DefaultDeclaration* contains the default declaration section of the attribute definition.

Attribute Data Types

XML DTD attributes can have the following data types: CDATA, enumerated, ENTITY, ENTITIES, ID, IDREF, IDREFS, NMTOKEN, and NMTOKENS.

CDATA

The CDATA data type indicates that the attribute can be set to any allowable character value.

Enumerated

The enumerated data type lists a set of values that are allowed for the attribute. The declaration is case sensitive.

ENTITY and ENTITIES

The ENTITY and ENTITIES data types are used to define reusable strings that are represented by a specific name.

ID, IDREF, and IDREFS

Within a document, you may want to be able to identify certain elements with an attribute that is of the ID data type. The name of the attribute with an ID data type must be unique for all of the elements in the document. Other elements can reference this ID by using the IDREF or IDREFS data types. IDREFS can be used to declare multiple attributes as IDREF.

NMTOKEN and NMTOKENS

The NMTOKEN and NMTOKENS data types are similar to the CDATA data type in that they represent character values. The *name tokens* are strings that consist of letters, digits, underscores, colons, hyphens, and periods. They cannot contain spaces.

The Default Declaration

The default declaration can consist of any valid value for your attributes, or it can consist of one of three predefined keywords: #REQUIRED, #IMPLIED, or #FIXED.

- The #REQUIRED keyword indicates that the attribute must be included with the element and that it must be assigned a value. There are no default values when #REQUIRED is used.
- The #IMPLIED keyword indicates that the attribute does not have to be included with the element and that there is no default value.
- The #FIXED keyword sets the attribute to one default value that cannot be changed. The default value is listed after the #FIXED keyword.
- If none of these three keywords are used, a default value can be assigned if an attribute is not set in the XML document.

Associating the DTD With an XML Document

There are two ways to associate a DTD with an XML document: the first is to place the DTD code within the XML document, and the second is to create a separate DTD document that is referenced by the XML document. Creating a separate DTD document allows multiple XML documents to reference the same DTD. We will use the external DTD in our project connecting to the accounting package. All defined DTD's will be stored in a *DTD Base Container*.

<!DOCTYPE DocName [DTD]>

or

<!DOCTYPE RootElementName SYSTEM|PUBLIC [Name]DTD-URI>

The SYSTEM keyword is needed when you are using an unpublished DTD.

We will use the second set of our own defined XML definitions. It will guarantee ourselves only to send well formed and valid messages. We will define it as PUBLIC with a reference to the DTD-container expressed as a UNC coded location.

Entities.

Entities are like macros in the C programming language in that they allow you to associate a string of characters with a name. This name can then be used in either the DTD or the XML document; the XML parser will replace the name with the string of characters. All entities consist of three parts: the word ENTITY, the name of the entity (called the literal entity value), and the replacement text—that is, the string of characters that the literal entity value will be replaced with. All entities are declared in either an internal or an external DTD.

> *When writing this manual, we do not have the intention to support this function for now. XML comes in several flavors and this is already an advanced feature. We don't know how other parsers can cope with this feature. However, we will support the default known entities.*

However, we will keep on describing these features so that we do have our own definitions for them.

Internal General Entities

Internal general entities are the simplest among the five types of entities. They are defined in the DTD section of the XML document.

Declaring an internal general entity.

The syntax for the declaration of an internal general entity is shown here:

```
<!ENTITY name "string_of_characters">
```

To reference a general entity in the XML document, you must precede the entity with an ampersand (&) and follow it with a semicolon (;).

Internal Parameter Entities

Internal parameter entities are interpreted and replaced within the DTD and can be used only within the DTD. While you need to use an ampersand (&) when referencing general entities, you need to use a percent sign (%) when referencing parameter entities.

Declaring an internal parameter entity

The syntax for declaring an internal parameter entity is shown here:

```
<!ENTITY % name "string_of_characters">
```

The XHTML Standard and Internal Parameter Entities

XHTML is currently out of the scope of this document. We just mention some key strokes. Whenever, applicable, consult a book concerning XHTML.

Inline entities and elements

The XHTML standard provides the following declarations for defining a series of internal parameter entities to be used to define the inline elements

```
<!ELEMENT p %Inline;>
```

Block entities and elements

The XHTML standard also declares a set of internal parameter entities that can be used in the declarations of the block elements

```
<!ELEMENT body %Block;>
```

XML Schemas

For the most part, XML documents fall into two categories: *document-oriented* and *data-oriented*. The document-oriented XML document contains text sections mixed with field data, whereas the data-oriented XML document contains only field data.

Up to this point, we've been looking only at document-oriented XML documents that contain only one data type (the *string* data type) because DTDs work best with document-oriented XML documents that contain only *string* data types.

Several built-in data types are known. For more information, please consult the next URL of the W3C http://www.w3.org/TR/xmlschema-2/ . Attached you can find a schematic overview of the most important data types.

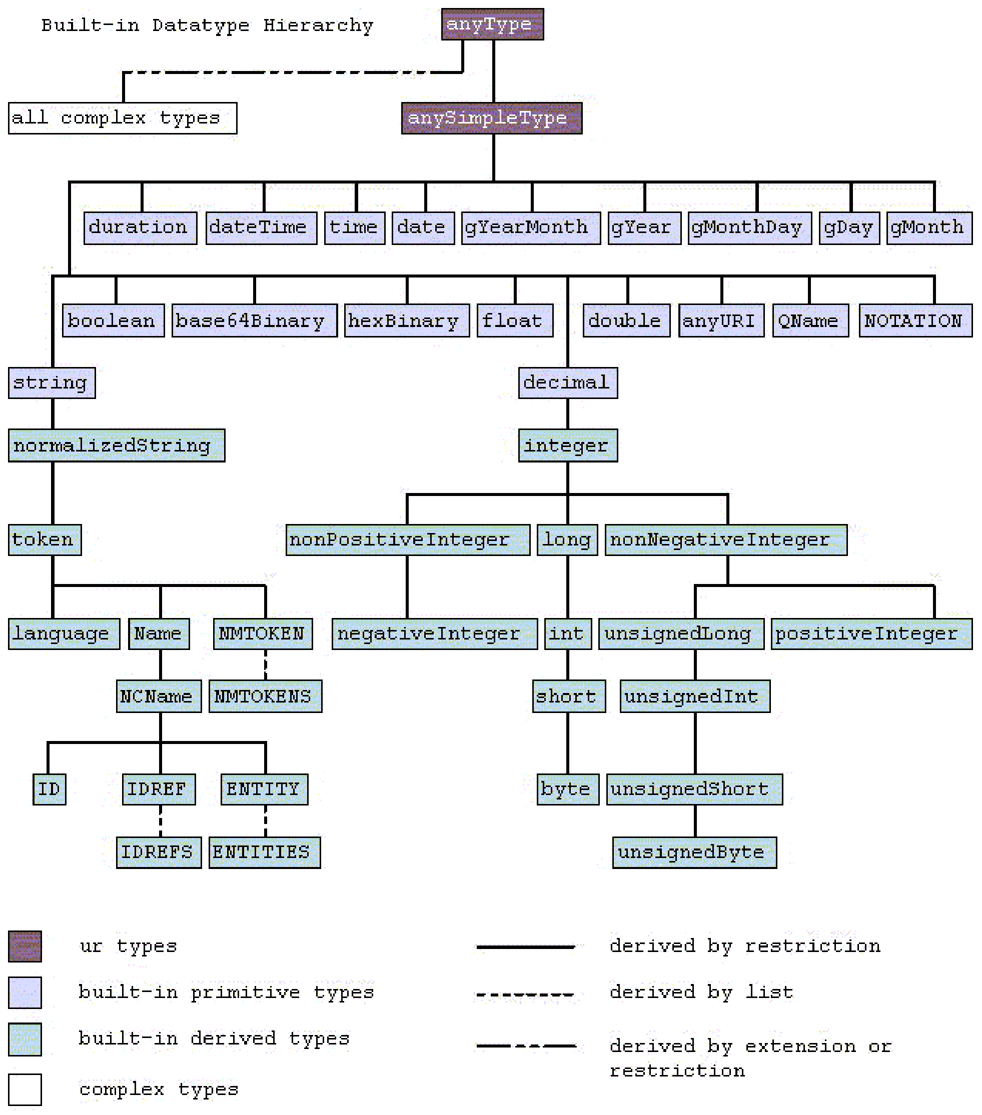
The figure that is attached can be found back on their web site.

Components of a schema data type.

In a schema, a data type has three parts: a *value space*, a *lexical space*, and a *facet*.

- The value space is the range of acceptable values for a data type.
- The lexical space is the set of valid *literals* that represent the ways in which a data type can be displayed.
- A data type can have many facets, each defining one or more characteristics. Facets specify how one data type is different from other data types. Facets define the value space for the data type.
  There are two kinds of facets: *fundamental* and *constraining*. Fundamental facets define the data type, and constraining facets place constraints on the data type.
  Examples of fundamental facets are rules specifying an order for the elements, a maximum or minimum allowable value, the finite or infinite nature of the data type, whether the instances of the data type are exact or approximate, and whether the data type is numeric. Constraining facets can include the limit on the length of a data type (number of characters for a string or number of bits for a binary data type), minimum and maximum lengths, enumerations, and patterns.
- We can categorize the data types along several dimensions. First, data types can be atomic or aggregate. An atomic data type cannot be divided. An integer value or a date that is represented as a single character string is an atomic data type. If a date is presented as day, month, and year values, the date is an aggregate data type.
- Data types can also be distinguished as primitive or generated. Primitive data types are not derived from any other data type; they are predefined. Generated data types are built from existing data types, called basetypes. Basetypes can be primitive or generated data types. Generated types, which will be discussed later in the chapter, can be either simple or complex data types.

Primitive data types include the following: string, Boolean, float, decimal, double, timeDuration, recurringDuration, binary, and uri. In addition, there is also the timeInstant data type that is derived from the recurringDuration data type. Among these primitive data types, two of them are specific to XML schemas: timeDuration, and recurringDuration. The timeInstant data type is also specific to XML.


Built-in Datatype Hierarchy
anyType
all complex types
anySimpleType
duration
dateTime
time
date
gYearMonth
gYear
gMonthDay
gDay
gMonth
boolean
base64Binary
hexBinary
float
double
anyURI
QName
NOTATION
string
decimal
normalizedString
integer
token
nonPositiveInteger
long
nonNegativeInteger
language
Name
NMTOKEN
negativeInteger
int
unsignedLong
positiveInteger
NCName
NMTOKENS
short
unsignedInt
ID
IDREF
ENTITY
byte
unsignedShort
IDREFS
ENTITIES
unsignedByte
ur types
built-in primitive types
built-in derived types
complex types
derived by restriction
derived by list
derived by extension or restriction

Index I