

XML - Extensible Markup Language

Jens-Fabian Goetzmann

June 13, 2006

Contents

Table of Contents	ii
References	iii
1 Introduction	1
1.1 XML's History so Far	1
1.1.1 SGML: The Roots of XML	1
1.1.2 HTML: The most Successful SGML Application	1
1.1.3 Enter XML	1
1.2 Why XML?	2
1.2.1 Ease of Use	2
1.2.2 Separation of Content and Presentation	2
1.2.3 Persistence	2
1.2.4 Further Standards	2
2 XML Fundamentals	3
2.1 XML Data Modeling	3
2.2 Entities: The Physical Structure	3
2.3 XML Documents	3
2.4 A First XML Document	3
2.4.1 XML Declaration	4
2.4.2 Document Type Declaration	4
2.4.3 Processing Instructions	4
2.4.4 Comments	4
2.4.5 CDATA Sections	5
2.4.6 Elements	5
2.5 Well-formedness	5
3 Document Type Definitions (DTDs)	6
3.1 Validity	6
3.2 Including a Document Type Declaration in an XML Document	6
3.2.1 The Internal DTD subset	6
3.3 The Contents of a DTD	7
3.3.1 Element Definitions	7
3.3.2 Attribute Definitions	8
3.3.3 Definition of General Parsed Entities	8
3.3.4 Definition of Parameter Entities	9
3.3.5 Notations and Unparsed Entities	9
4 Namespaces	9
4.1 Binding Namespaces To Prefixes	10
4.2 Setting the Default Namespace	10
4.3 Namespaces and Attributes	10
5 Example Applications	10
5.1 Narrative Applications	10
5.2 Data Set Applications	11
6 Summary	11

References

- [GOL2000] Charles F. Goldfarb, Paul Prescod: The XML Handbook, Second Edition. Prentice Hall PTR, Upper Saddle River, U.S.A. 2000
- [MOR2000] Michael Morrision: XML Unleashed. Sams Publishing, Indianapolis, U.S.A. 2000
- [HAR2005] Elliotte Rusty Harold, W. Scott Means: XML in a Nutshell, Deutsche Ausgabe der 3. Auflage. O'Reilly Verlag, Köln, Deutschland 2005
- [LOB2000] Henning Lobin: Informationsmodellierung in XML und SGML. Springer, Berlin, Deutschland 2000.
- [W3C2004] Tim Bray et al.: Extensible Markup Language (XML) 1.0, Third Edition. W3C Recommendation, 04 February 2004. <http://www.w3.org/TR/2004/REC-xml-20040204>
- [W3C1999] Tim Bray et al.: Namespaces in XML. W3C Recommendation, 14 January 1999. <http://www.w3.org/TR/1999/REC-xml-names-19990114>

1 Introduction

1.1 XML's History so Far

Since XML was conceived in the late nineties of the 20th century, it has been getting ever more important to developers, enterprises, data base specialists, web designers and last but not least the users. But why is it that a document format that did not even bring any "new" ideas to the world when compared to its predecessor SGML happens to be so successful while SGML is not even known to most of those now using XML?

1.1.1 SGML: The Roots of XML

In the late sixties, IBM needed a system for storing and managing legal documents. They needed it to be so general that it would allow people using different systems and programs exchanging their data. The team asked to develop this system was Charles Goldfarb, Ed Mosher and Ray Lorie. They quickly recognized three requirements that such a system would have to meet:

1. Common document representation: The system should have a *document representation* or *file format* that should be common to all programs. It should be easy to understand by human readers and machines and just as easy to write.
2. Customized document types: Whereas the document representation itself should be fixed, the type of the content should be able to be *customized* so that for different areas such as medical or legal documents specific content and even specific structure could be applied to the documents.
3. Rule-based markup: As the system should allow different customized document types, there should also be a specific way of defining the *rules for the structure* of a document of a certain document type and enforcing the rules so a program working with a document of the customized type could reliably interpret it.

In 1969 the IBM team developed the *Generalized Markup Language (GML)*¹. Five years later, Goldfarb proved the concept of a *validating parser* that could verify the structure of a document with respect to its document type. In the years between 1978 and 1986, Goldfarb was the technical leader of a team of users, programmers and academics who gradually developed GML into the ISO standard 8879, called the *Standard Generalized Markup Language (SGML)*.

1.1.2 HTML: The most Successful SGML Application

In 1989 Tim Berners-Lee and Anders Berglund, two researchers at the European Nuclear Research Facility CERN, developed a simple SGML application that utilized hyperlinks² to share information over the Internet. The newly created markup language was dubbed *Hypertext Markup Language (HTML)* and the hypertext system got the promising name *World Wide Web*. The WWW was such a success for the Internet that in the early nineties an ever growing number of people got "on line" to enjoy the benefits of this rapidly growing source of information. In fact, today, there are lots of people who even think that the World Wide Web *is* the Internet.

The simplicity of HTML allowed everyone able to use a text editor to create his or her own webpage easily. But as personal computers evolved from simple working machines into multimedia centers, web designers wanted more control about the looks of their websites. Browser vendors reacted by introducing more and more elements into the language damaging HTML's separation of content and presentation. Developers also tried to use HTML for purposes it just did not fit - after all, it was just a language for marking up web pages.

1.1.3 Enter XML

In 1996, members of the *World Wide Web Consortium (W3C)* realized that there was a common need for a new kind of general markup language - a language that should be as extensible and flexible as SGML, but easy to learn, use and process just like HTML. After all, the SGML specification was so complex that most of the SGML software did not implement it fully. SGML had not been designed with a global computer network as a target and while it was fit to deal with huge technical documents, it could not be used for real time operation on small files sent over the network and subject to instant processing.

In 1998, the W3C released the first edition of the specification for the *Extensible Markup Language (XML)*. The newly developed language had a bunch of key advantages:

- XML is a *real subset* of SGML, which means that existing programs such as SGML parsers can also easily process XML documents. Some major SGML applications like TEI (Text Encoding Initiative) or DocBook have already been ported from SGML to XML.

¹Which - not coincidentally - has the same initials as the team members.

²A technology that had also been around for a while at least in research projects but never really gotten off.

- XML meets the three key requirements mentioned above - making it flexible enough to fit almost every possible purpose yet allowing validation of certain strict rules defining the structure of a document type.
- The XML specification is relatively small - it has about a tenth of the size of the SGML standard. This makes it easier for users to learn and for developers to implement the full extent of the language.

1.2 Why XML?

Now that the road that led to the development of XML was discussed in detail, the major question that arises is: Why should anybody want to use XML? There are several advantages of XML - besides the already mentioned extensibility and possibility of specialization - that make it the document format of choice for many companies, developers and researchers.

1.2.1 Ease of Use

XML is (like all languages of the SGML family) a text format, that means that on every computer system that has a text editor one can easily create, view or modify XML documents. There is no need for special software for writing or maintaining XML documents.

XML parsers and other software are also widely available and ready to use for all kinds of platforms. If somebody wanted to write a program using XML as a document format for saving data, he or she could use one of the many libraries already available - or write one on his or her own, which is not difficult either because of the text-only syntax of XML.

1.2.2 Separation of Content and Presentation

Most well-designed XML documents contain only the real *content* of the document and not information on how to present it on a screen or a printout. This makes it easier to write and maintain these documents. The customization of XML makes it also possible that a person new to the particular XML application will quickly understand what the elements in an XML document mean.

There are several possibilities of specifying presentation information for an XML document, such as *Cascading Stylesheets* (CSS), or - getting ever more important - the *Extensible Stylesheet Language* (XSL) with its enormously powerful *XSL Transformations* (XSLT). The way of specifying in a document which kind of stylesheet should be used is also standardized.

1.2.3 Persistence

XML documents are more likely to be readable for a long time than any generic document format before.³ First, they do not rely on a specific platform or software to be available. Second, the information and meta-information contained in an XML document⁴ is encoded in a manner allowing a person reading the file itself and not utilizing any special software but a text editor to reconstruct the contents of the document.

1.2.4 Further Standards

Since the advent of XML in 1998, the W3C and other organizations have not been idle. Although there has only been a minor upgrade to the XML specification itself, several other XML technologies have emerged, such as

- *Namespaces in XML*, a way of generating unique identifiers and merging together or embedding document parts using different document types, discussed in depth in section 4.
- *XSL*, the powerful stylesheet language already mentioned above.
- *XML Schema*, which was developed by the W3C and is an even more versatile way of defining what elements and what data are allowed in certain contexts. There are also various other Schema languages from third party organizations.
- *XPath* and *XPointer* that allow referring to a exactly defined portion of an XML document instead of the whole document, e.g. in hyperlinks.
- *XLink*, which is a standard for hyperlinking that goes way beyond the possibilities of HTML. With XLink it is possible to create links between more than two documents or even between documents that the author of the linking XML document does not have writing access to.
- The *Document Object Model* (DOM) and the *Simple API for XML* (SAX) that define standard ways of accessing XML documents and have been implemented in various programming languages.

³Besides ASCII plain text, maybe.

⁴If the XML application is well designed.

2 XML Fundamentals

2.1 XML Data Modeling

The first and most important thing to notice in order to understand the structure of an XML document or even design a new XML application is that all data modeled in XML will have a *hierarchical tree structure*. A book, for example, can be broken down into chapters, the chapters into sections and the sections further down into paragraphs and so on. Most (natural) documents can be viewed this way, but there are also some to which such a form can not be easily applied.

The conceptual tree of an XML document consists mainly of element nodes, text nodes and attribute nodes.⁵ Elements have got a name and can have other elements or text as their children. They also can have named attributes attached to them.

2.2 Entities: The Physical Structure

While the logical structure of an XML document is always that of a tree, its physical structure may vary. XML uses the concept of *entities* when it comes to the physical structure of XML documents. The most common form of an entity is a file, but there are also other kind of entities: An XML document created on the fly by a web server from some data in a data base and sent to a web browser on a client machine for output, for example, is not a file, as it will probably never be physically stored with a file name etc.⁶ - but it is still an entity. Entities may even be portions of files, XML also allows defining various kinds of entities inside an XML document.

2.3 XML Documents

The term *XML document*, that appeared in this paper quite a few times up to now, refers to the totality of text⁷ that has to be read to "understand" the contents and structure of - well, the XML document. An XML document can be spread over several entities, but every document has at least one entity, in which it is started. This entity is called the *document entity*. XML utilizes several methods of including other entities from the document entity, such as the *Document Type Definition* (DTD), or external parsed or unparsed entities. These are discussed in Chapter 3.

A document consists of the actual data of the document, such as the text of a book, for example, and metadata that is included in the markup.⁸

2.4 A First XML Document

In listing 1 an example of an XML document that illustrates the basic concepts of XML is shown.

Listing 1: A First XML Document

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE person SYSTEM "person.dtd">
<?xml-stylesheet type="text/xsl" href="stylesheet.xsl" ?>
<!-- This is an example file -->
<person>
  <name>
    <firstName>Kurt</firstName>
    <lastName>Gödel</lastName>
  </name>
  <profession>Mathematician</profession>
  <literature>
    <book type="work">
      Über formal unentscheidbare Sätze der
      Principia mathematica und verwandter Systeme
    </book>
    <book type="biography">
      Kurt Gödel. Leben und Werk.
    </book>
  </literature>
</person>
```

As pointed out earlier, this document may reside in a file but also in another kind of entity, such as an answer to a HTTP request sent to a web server, but the physical location is irrelevant in this context.⁹

⁵The terminology that is used here is close to the DOM terminology.

⁶Besides in the web browser's cache, maybe.

⁷The term "text" is used in this context to denote the bits and bytes - depending on the character encoding that is used - that the processing program maps to characters. There is an important distinction between this text and the text that is contained in the XML document itself, as this "physical" text also includes the markup.

⁸Goldfarb and Prescott [GOL2000] make up the catchy equation "Data + Markup = DocuMent".

⁹In fact, it is possible to work with XML very well even if one has never heard of the concept of entities as a superset of files - some XML books do not even mention it. But as this is a central concept of SGML, it is included in this paper.

In the following sections, the different parts of an XML document are discussed in detail.

2.4.1 XML Declaration

An XML declaration is shown in listing 2. It does not have to be included in an XML document,¹⁰ but if it is, it must be the very first line. It has three so-called pseudo-attributes.¹¹

Listing 2: The XML Declaration

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
```

XML Version The first pseudo-attribute declares the XML version. There are two versions of the XML recommendation right now: 1.0 and 1.1. The only difference between those versions that really matters is that XML 1.1 always refers to the latest version of the Unicode standard and allows the use of some control characters in XML documents that were forbidden in XML 1.0. However, for documents that do not use characters from a language such as Cambodian, Burmese or Mongolian, the new standard is not needed and the old standard should still be used - because most parsers do not work with the new standard yet.

Character Encoding The second pseudo-attribute declares the encoding used in the file. It is optional. All XML parsers have to support the Unicode standards UTF-8 and UTF-16, but one might also use other encodings such as ASCII, ISO-8859-1 (a.k.a. Latin-1), or Cp1252, which is Microsoft Windows' standard character encoding. The encoding declaration might lead to a hen and egg problem: If a processing program does not know the character encoding, how can it read the declaration? This is why processing programs are allowed to get the character encoding from other sources, such as HTTP headers, or even guess the encoding. However, the declaration still makes sense to distinguish between the various ASCII supersets Latin-*, UTF-8 etc.

Standalone Declaration The last pseudo-attribute is the so-called standalone document declaration. It is optional as well. It may have the value "yes" or "no". A value of "yes" means that the document can be correctly parsed without reading other entities beside the document entity. Most of the time, if a document refers to a DTD, the value should be set to "no". A value of "no" is also the default value.

2.4.2 Document Type Declaration

The second line of listing 1 contains a document type declaration. Document type declarations are discussed in detail in section 3.

2.4.3 Processing Instructions

An XML document may contain any number of processing instructions at various positions of the document. Every processing instruction starts with the characters <? and ends with the characters ?>. The characters from the beginning of the processing instruction up to the first white space inside the processing instruction are called the *target* of the processing instruction. The rest of the processing instructions may have an arbitrary form, but most processing instructions use a pseudo-attribute form like the XML declaration. Processing instructions are meant to pass information to processing programs. Target names starting with the characters "XML" in any combination of upper and lower case are reserved for future standardization. In fact, there is already a standard processing instruction for linking stylesheets for presentation as shown in listing 1.

2.4.4 Comments

While processing instructions may be included in XML documents to give information to processing programs, comments may be included to give information to a user reading the document. However, they should not be used to include information that is *necessary* to understand the document, as they might be stripped out or silently ignored by parsers.

¹⁰Nevertheless, it is recommended to include it in documents, the possibility to omit it is mainly intended to maintain compatibility with existing SGML parsers.

¹¹Although they look like ordinary attributes, they are none. The only significant difference is that pseudo-attributes must be in the correct order, while normal attributes may be placed in any order.

2.4.5 CDATA Sections

CDATA Sections are not shown in listing 1. CDATA is short for "character data". These sections can occur in the content of elements, where normal text would be allowed. They are a way of including text in an XML document that should not be parsed for markup. CDATA sections are typically used for including XML, SGML or HTML examples in XML documents. They start with the text `<![CDATA[` and end with `]]>`. An example section is shown in listing 3.

Listing 3: CDATA Sections

```
<example>
  Here is an XML example: <![CDATA[
    <ex1>
      This is not the content of an ex1 element, because
      the start tag is not recognized as markup.
    </ex1>
    <!-- This is not a comment either -->
  ]]>
</example>
```

2.4.6 Elements

The last (and most important) thing an XML document has to contain is the *root element*. It is the root of the tree discussed in section 2.1.

Every element has a start tag and an end tag, or a single empty-element tag. Tags are always delimited by angled brackets. Both types of elements are shown in listing 4.

Listing 4: Tags and Elements

```
<element attribute1="value" attribute2="value">
  <!-- Element Content -->
  <subelement>
    <!-- Subelement Content -->
  </subelement>
</element>

<emptyElement anotherattribute="value" />
```

The characters in the start tag up to the first white space are the *element name*. It is important to distinguish between tags and elements: Although a tag in an XML document denotes an element, the element itself is the node in the conceptual structural tree of the XML document.

The start tag or the empty element tag might contain one or more attributes, as already shown in listing 4. The characters on the left side of the equals sign is the *attribute name*. The values of the attribute on the right side of the equals sign are always delimited by single or double quotes.

The content of an element is everything between the start tag and the end tag. Elements that are defined here are *children* of the element in the structural tree. Text that is included here is a text child node of the element.

2.5 Well-formedness

A central concept of XML is the well-formedness of XML documents. An XML document is well-formed, if it fulfills all rules of the XML recommendation.¹² The most important rules are the following:

- All elements must be properly nested, i.e. `<a>...` instead of `<a>...`.
- All attribute values must be delimited with single or double quotes.
- The characters `<`, `>`, `'`, `"` and `&` may have to be replaced with character references or named entities - more on that in section 3.3.3.
- All elements must have a start tag and an end tag or an empty element tag - just a start tag, like in HTML (``) is not allowed.
- The document must include one root element.

¹² Formally [W3C2004]: A textual object is a well-formed XML document if:

1. Taken as a whole, it matches the production labeled "document" of the context free grammar included in the XML specification.
2. It meets all the well-formedness constraints given in the XML specification.
3. Each of the parsed entities which is referenced directly or indirectly within the document is well-formed.

A document that is not well-formed is - by definition - not an XML document. If a parser reads a document that violates these rules, it must report it to the program utilizing the parser. A parser is not allowed to try to "repair" the document. This might seem a bit weird, but this rule was included in the XML specification to ensure that XML would not be subject to "browser wars" as HTML was in its early days and still is up to today.

3 Document Type Definitions (DTDs)

As already mentioned, SGML included a possibility to set up rules that certain types of documents have to follow by writing a so called *Document Type Definition* (DTD). Today, there have been other approaches to defining these rules, such as XML Schema, but DTDs are still the only way of doing so that is defined in the XML standard itself.

A quick note on the terminology: The DTD itself is the definition of a document type. The document type *declaration*, sometimes abbreviated DOCTYPE, is the reference to a specific DTD from within an XML document.

3.1 Validity

As noted above, a DTD defines how a particular class of XML documents is structured. If an XML document includes a document type declaration and it conforms to the definitions of the DTD, the document is called *valid*. If an XML document includes a document type declaration and it *does not* conform to the definitions, it is called *invalid* (but it remains well-formed). Note that a well-formed XML document that includes no document type declaration is neither valid nor invalid, and a document that is not well-formed is not an XML document, so there is nothing to say about validity.

3.2 Including a Document Type Declaration in an XML Document

A document type declaration looks like the example in listing 5.

Listing 5: A Document Type Declaration

```
<!DOCTYPE rootElement SYSTEM "/path/to.dtd">
```

Instead of the text `rootElement`, a real document type declaration would include the name of the root element of the document. After the keyword `SYSTEM`, the URI of the entity that the DTD resides in has to be specified. It might be in the form of a full URI or a relative path as shown in listing 5.

Instead of using the keyword `SYSTEM` the keyword `PUBLIC` followed by a public identifier and an additional URI might be used. A public identifier is a unique string that refers to a standard DTD. Processing programs might incorporate some mechanism for caching these standard DTDs so they do not have to fetch it from the URI. A sample document type declaration using a public identifier (in this case, the public identifier for XHTML 1.0, which is a re-formulation of HTML 4.01 in XML), is included in listing 6.

Listing 6: Using Public Identifiers

```
<!DOCTYPE html PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

3.2.1 The Internal DTD subset

Before the actual contents of a DTD are discussed in detail, it is worth noting that an *internal subset* of the DTD may be included right in the document entity. This means that there is there is the possibility to define (as opposed to declare) the structure of a document right in the document itself. The internal subset is read and interpreted before the external subset, and definitions made in the internal subset cannot be overwritten by contrary definitions in the external subset. While this might sound strange at the moment, there is a possibility of creating "modular DTDs" that include portions that can be "switched" on and off, but this concept is too complex to explain it in detail in this paper. Listing 7 shows some possibilities of including an internal DTD subset.

Listing 7: Using an internal DTD subset

```
<!DOCTYPE [
  <!-- Internal DTD Subset -->
]>

<!DOCTYPE root SYSTEM "the.dtd" [
  <!-- Internal DTD Subset -->
]>
```

```
<!DOCTYPE root PUBLIC "-//PublicIDs//MyExampleDTD//EN" "another.dtd" [
  <!-- Internal DTD Subset -->
]>
```

3.3 The Contents of a DTD

3.3.1 Element Definitions

A DTD has to contain definitions of all the elements that will be used in XML documents conforming to the DTD. An element definition contains the *name* of the element followed by the *content model*.

The element name must be a valid XML name, that means it must be a character string that may include only alphanumeric and/or ideographic characters, underscores, colons, commas and dots and starts with a character, an ideographic symbol or the underscore. All names in XML are - in opposition to HTML - case sensitive, that means "element" and "eLeMeNt" are not the same. Element names starting with the letters "XML" in any combination of upper and lower case are reserved for future standardization.

The content model is a description of what other elements might occur as children of the element, and if and how text might be contained in the element. Some element definitions are shown in listing 8.

Listing 8: Element Definitions

```
<!ELEMENT element1 (EMPTY) >
<!ELEMENT element2 (#PCDATA) >
<!ELEMENT element3 (element1,element2)* >
```

There are several different content models that are allowed in an element definition:

The EMPTY Content Model The content model `EMPTY` denotes that this element may contain neither child elements nor text. Most of the time, an empty element will have one or more attributes (Described in section 3.3.2).

The Content Model ANY The content model `ANY` denotes that this element may have any content, including all kinds of child elements, mixed content or character data. In final DTDs, it will most likely never be used, but it can be helpful when developing a new DTD.

The Content Model #PCDATA The content model `#PCDATA` (an abbreviation for "parsed character data") denotes that this element is not allowed to have any content but text.

Sequence The braces delimiting the content model may contain one or more names of other elements, separated by commas. In this case, the element has to have these elements as child elements, in the order in which they are specified in the element definition.

Alternative The braces delimiting the content model may contain one or more names of other elements, separated by pipes (`|`). In this case, the element has to have one (and only one) of these elements as a child element.

Repetition A content model may be followed by an asterisk (`*`) or a plus sign, denoting that this content model might be repeated zero or more times or one or more times, respectively. A content model followed by a question mark means that this content model might or might not be matched. Content models might be nested, so a detailed model like the one in listing 9 is possible.

Listing 9: Nested Content Models

```
<!ELEMENT name (academicTitle?, (lastName | (firstName+, middleInitial*, lastName))) >
```

Mixed Content There is another special content model, called "mixed content". Mixed content means that an element can contain text and child elements. The only way to define mixed content is with an alternative that can be repeated zero or more times and has the content model `#PCDATA` as its first option.¹³ A sample definition can be found in listing 10.

Listing 10: Definition of Mixed Content

```
<!ELEMENT paragraph (#PCDATA|quote|emphasize)* >
```

¹³SGML has more alternatives, but the XML inventors intentionally restricted this.

3.3.2 Attribute Definitions

For every element that was defined with an element definition it is also possible to specify the attributes it might get. An attribute definition looks like the example in listing 11.

Listing 11: Attribute Definitions

```
<!ATTLIST element attribute1 ID #REQUIRED
                attribute2 (value1|value2) #IMPLIED
                attribute3 CDATA #FIXED "fixed value"
                attribute4 CDATA "default value">
```

The attribute definition contains the element name, followed by a list of the attribute names, their *types* and their *default*.

Attribute names must be valid XML names. As with element names, all attribute names starting with "XML" are reserved.

The *default* occurring at the last position of the definition of each attribute can basically have four values: #IMPLIED means that there is no default value because there is an application specific way of implying the correct value. #REQUIRED means that the value of the attribute must always be specified in the element in the XML document. #FIXED followed by a value delimited by quotes means that the attribute always has the specified value, a specification of another value in the XML document is a fault. The last method is just specifying a default value delimited by quotes - this means that it can be overridden in the XML document itself, but if no value is specified in the XML document, the attribute gets the default value.

There are several *attribute types*:

Attribute Type CDATA: The attribute type CDATA (once again referring to "character data") is the most simple attribute type, the attribute value might be any string of characters.

Alternative: It is possible to give a list of possible attribute values delimited by brackets and separated by pipes. In this case, specifying a value not contained in the list in the XML document means that the document is invalid.

Attribute Type ID: The attribute type ID means that if the attribute is assigned a value, that value must be a valid XML name (only alphanumeric and ideographic characters, the underscore, the colon, the dot and the comma, starting with a character or the underscore) and it must be unique, that means no other attribute with the type ID may be assigned the same value.

Attribute Type IDREF: The attribute type IDREF means a reference to an ID, i.e. an attribute of this type may be assigned a value assigned to another attribute of the type ID anywhere in the document.

Attribute Type IDREFS: Attributes of this type might be assigned a sequence of IDREF values, separated by whitespace.

Attribute Type NMTOKEN: Attributes of the type NMTOKEN might be assigned a value containing of any combination of alphanumeric and ideographic characters, the underscore, the colon, the dot and the comma, starting with a character and the underscore.

Attribute Type NMTOKENS: As with IDREFS, attributes of this type may be assigned a sequence of NMTOKEN values, separated by whitespace.

Attribute Types NOTATION, ENTITY, ENTITIES These attribute types may be used to refer to extern unparsed entities, but as they are hardly ever used and a bit difficult, they are not explained in detail in this paper.

3.3.3 Definition of General Parsed Entities

Parsed entities are a way of defining "building blocks" that may be inserted everywhere in the text. Every entity has a *name* and a *replacement text*. Whenever a parser reads an entity reference, which is denoted by the ampersand (&) followed by the entity name and a semicolon (;), it replaces this reference with the replacement text of this entity.

There are five predefined general parsed entities representing single characters that normally denote markup, they are listed in table 1. Whenever a "<" symbol is used in text content, it has to be replaced by "<"; because otherwise a parser would interpret it as the beginning of markup. A similar rule applies to the character "&": If it occurs within a document's text, it has to be replaced by "&"; because otherwise it would be interpreted as the beginning of an entity reference. The other predefined general entities are for special cases such as an attribute value that contains both single and double

Table 1: Predefined General Entities

Entity	Replacement Text
<	<
>	>
&	&
"	"
'	'

quotes, where one type of them (the one that is also used for delimiting the attribute value itself) would have to be escaped with """ or "'".

As already mentioned, there is the possibility to define additional entities with complex replacement text (that may also include well-formed markup). It is worth noting however, that in order for a document to be valid, its DTD must define the five predefined entities as well.

Defining an internal general parsed entity works like demonstrated in listing 12.

Listing 12: Defining Internal Parsed Entities

```
<!ENTITY entity1 "This is the replacement text.">
<!ENTITY entity2 "<element>It may contain markup.</element>">
```

There is also the possibility to define external parsed entities that take their replacement text from another file¹⁴ etc. Defining them works as shown in listing 13.

Listing 13: Defining External Parsed Entities

```
<!ENTITY entity3 SYSTEM "path/to/entity">
```

3.3.4 Definition of Parameter Entities

Parameter entities work just like general entities, with one important difference: general entities may be used anywhere in the document except inside the DTD, parameter entities may be used inside the DTD only. They are used for modular or extensible DTDs or to abbreviate content models that appear more than once. In opposition to general entities, they are referenced with a percent sign (%) followed by the entity name and the semicolon. They are defined as shown in listing 14.

Listing 14: Defining Parameter Entities

```
<!ENTITY % parameterentity "replacement text">
```

3.3.5 Notations and Unparsed Entities

Notations and unparsed entities are XML's (and SGML's) native way of referring to non-XML, binary files outside the document. They are seldom used and so they are not discussed in detail in this paper.

4 Namespaces

Namespaces are a way of generating element or attribute names that are globally unique. This is important for embedding XML data from one XML application into a document of another application, e.g. MathML equations in an XHTML web page. A processing program reading such a document would need to figure out which element belongs to which XML application. Namespaces allow the creation of different "areas" or "collections" of element and attributes.¹⁵ To identify these namespaces, an URI reference is used. The URI itself does not have to point to anything special,¹⁶ it is just used as a source for globally unique names.

Unfortunately, namespaces and DTDs do not work together that well. When DTDs were invented,¹⁷ no one thought of namespaces, so creating a document that utilizes both of them takes some extra effort.

¹⁴More correctly, from another entity, but this would sound a bit irritating here.

¹⁵Definition [W3C1999]: An XML namespace is a collection of names, identified by a URI reference, which are used in XML documents as element types and attribute names.

¹⁶Although there are efforts to specify suggestions what should be located at a namespace URI, because people keep on entering namespace URIs in their browser.

¹⁷Back in the 1970s!

4.1 Binding Namespaces To Prefixes

As it would be rather annoying if one had to specify the namespace URI in every opening or closing tag, the W3C conceived the concept of *binding* an URI to a *prefix* and using the prefix followed by a colon and the element name instead of just the element name to denote that the element belongs in the namespace of the URI bound to the prefix. As shown in listing 15, binding a prefix works with a special kind of attribute: Its name starts with `xmlns:` followed by the prefix to be bound. The value assigned to the attribute should be the URI of the namespace.

The prefix is available in the element it was bound on and all its descendants.

Listing 15: Using Namespace Prefixes

```
<pre:element xmlns:pre="http://www.example.com/xmlns/">
  <pre:text>This element is in the example namespace</pre:text>
  <text>This element is not in the example namespace</text>
</pre>
```

4.2 Setting the Default Namespace

It is also possible to define a default namespace by assigning a namespace URI to the attribute `xmlns`. In this case, all elements that are descendants of the element on which the default namespaces was assigned and have no other namespace assigned are in this namespace. Listing 16 demonstrates.

Listing 16: The Default Namespace

```
<element xmlns="http://www.ex1.com/xmlns/" xmlns:ex2="http://www.ex2.com/xmlns/">
  <!-- The element "element" is in the ex1 namespace -->
  <ex2:text>This element is in the ex2 namespace.</ex2:text>
  <text>This element is in the ex1 namespace.</text>
  <text xmlns="http://www.ex3.com/xmlns/">This element is in the ex3 namespace.</text>
</element>
```

4.3 Namespaces and Attributes

Prefixes may also be used on attributes. It is worth noting that attributes that are not explicitly assigned a namespace via a prefix are not in the namespace of their element and not even in the default namespace. They are in a special attribute namespace. However, there are technologies like XLink that use attributes with a special namespace, e.g. to create hyperlinks.

5 Example Applications

Since the XML was officially presented eight years ago, there has been an ever growing movement to specify new XML applications. Companies of all sizes turned to XML as their new document format of choice. In this section a few examples of existing XML applications are given - although there is an uncountable number of others.

5.1 Narrative Applications

Narrative applications are all kinds of "textual" XML applications, i.e. everything that may be printed in a book-like form. This is what SGML (and XML) was intentionally designed for. There are several successful and widely used narrative XML applications:

Text Encoding Initiative (TEI): TEI was originally a SGML application and has been ported to XML. TEI is a document type intended for the scientific reading of ancient textual documents. It includes markup to denote unreadable or corrected parts or even spelling mistakes.

DocBook: Like TEI, DocBook was originally a SGML application now ported to XML. In contrast to TEI, DocBook is a format for writing new books. A lot of software documentation has been written using DocBook.

Open Document Format: The Open Document Format is a collection of office document formats which was developed from the OpenOffice.org format and has been standardized by the Organization for the Advancement of Structured Information Standards (OASIS) and only recently also by the ISO. A document in the Open Document Format is in fact a ZIP archive containing some XML files. The Open Document Format is not very abstract and rather presentation oriented.

Wordprocessing Markup Language: This XML application is used in Microsoft Word. It is more or less XML wrapped around a proprietary, binary format, so it does not contain any abstraction and is not very well readable.

5.2 Data Set Applications

Somewhat surprisingly, XML has been a success in areas SGML was never intended for. Data set documents are documents that are not meant to be rendered to the user in the form of a browser window or a printed paper, but they are generated by programs and intended for programs. There is an uncountable number of this kind of applications and most of them are not of any general interest. Still, there are a few that can be used for different purposes:

XML-RPC: RPC is short for Remote Procedure Call. As the name suggests, this is a concept used by client/server architectures: The client sends an XML document to the server requesting the result of a certain procedure, and the server sends back another XML document with the result. XML-RPC is a simple format which is not object-oriented and has a very flat hierarchy.

SOAP: SOAP (originally an acronym for "Simple Object Access Protocol") is an envelope format for messages. It allows richer structures than XML-RPC and can embed complex XML documents with a deep hierarchy.

Serialized Objects: Programming languages allow their objects to be serialized so they can be stored on a persistent memory or be sent over a network and later be instantiated again. Some use XML for this purpose, e.g. JavaBeans since version 1.4.

Scalable Vector Graphics (SVG): SVG is another W3C recommendation for a platform independent XML vector graphics format.

6 Summary

Although it is based on a few simple rules, XML is a document format as flexible as almost no other document format ever developed. There are only very few tasks for storing information that can not be solved with the help of XML. No week passes without new companies and users turning to XML for storing or processing their data.

XML technology keeps rapidly developing. Today's interactive web applications featuring AJAX techniques would not be possible without XML and the DOM. Standardized methods for exchanging information are being conceived, and new standards for document interchange are being based on the solid foundation of XML.

XML keeps what SGML has promised. Since its invention, it has been a single story of success for the W3C and those using XML for their documents as well.