XML Basics (part 2)

By icarus

Table of Contents

Tell Me More

In the first part of this article, I examined the need and rationale for XML, together with a brief look at the rapidly−increasing number of XML−related technologies. I discussed the basic structure and components of an XML document, played with the document prolog, and spent some time explaining how elements and attributes work. I also explained the difference between well−formed and valid XML, and demonstrated how the document prolog can be used to link an XML document to a DTD.

In this concluding article, I'll be examining some of the other things that go into an XML document, including CDATA, processing instructions, namespaces and entity references. Don't even think about going anywhere!

Splitting Up

First up, CDATA. As explained in the previous article, the XML specification considers all text enclosed within tags to be character data. There is one important exception to this − CDATA blocks.

CDATA blocks are document sections explicitly marked as not containing markup, and are hence treated as character data by the parser. These blocks can contain pretty much anything − strings, numbers, symbols, ancient Egyptian hieroglyphics − and will be ignored by the parser.

A CDATA block typically begins with

```
<![CDATA[
```

and ends with

```
]]>
```

with the data enclosed within the two. Here's an example:

```
<?xml version="1.0"?>

<manual>
<function>split(str, pattern)</function>

<description>Split a string <param>str</param> into component
parts on the
basis of <param>pattern</param></description>

<example>
<![CDATA[
<?

split("apple, vanilla, orange", ",");

?>
]]>
</example>
```

```
</manual>
```

CDATA blocks make it easy to add large blocks of text (including text containing special characters, symbols or program code) to an XML document, yet have the parser treat it as regular character data. And so, while a parser might choke on this,

```
<?xml version="1.0"?>

<secret_message>
<from>Our man in Paris</from>
<to>Director, Special Operations</to>

<coded_body_text>
12637 0%%348 83483 89238 82383 10341 0*049 27216 02039 84585
18127 45759
3@492 83%84 22829 238#3 92345 72310 53467 12941 92461 40149
7^21271
46101 42356 74(@1 4!128 47353 #511~ 473~7 12942 38#53 45628
</coded_body_text>

</secret_message>
```

it will be absolutely fine with this.

```
<?xml version="1.0"?>

<secret_message>
<from>Our man in Paris</from>
<to>Director, Special Operations</to>

<coded_body_text>
<![CDATA[
12637 0%%348 83483 89238 82383 10341 0*049 27216 02039 84585
18127 45759
3@492 83%84 22829 238#3 92345 72310 53467 12941 92461 40149
7^21271
46101 42356 74(@1 4!128 47353 #511~ 473~7 12942 38#53 45628
]]>
</coded_body_text>
```

```
</secret_message>
```

Obviously, you cannot include the ending sequence

```
]]>
```

within a CDATA block, as this would merely serve to confuse the parser. If you need to include this sequence within a CDATA block, it needs to be written as

```
]]&gt;
```

Eating Humble PI

In addition to character data, XML also allows document authors to include specific instructions or commands to the processing application within the document. These instructions are referred to as "processing instructions", or PIs. PIs are not part of character data; instead, when an XML parser encounters a PI, it simply hands it over to the calling application, which has the option of using it (if it recognizes it) or ignoring it (if it doesn't.)

Every PI includes a target − this is the string used to identify the application to which the instruction is directed − followed by some data. This target−and−data combination is enclosed with <?...?> tags, as demonstrated by the following example:

```
<?xml version="1.0"?>

<directory>

<category>Online Shopping<?rating popular?></category>

<url>http://www.amazon.com</url>
<desc>Amazon.com, the planet's foremost e−tailer<?link_with_ad
?></desc>

<url>http://www.cdnow.com</url>
<desc>CDNow.com, for all your music</desc>

<url>http://www.bn.com</url>
<desc>Barnes & Noble's online bookstore</desc>

</directory>
```

This data will be used by the XML application − for example, the first PI could indicate that the category be marked as "popular", while the second could link the item description with an advertisement.

If you take a look at the document prolog (discussed in the last article), you'll see that the first line in any XML document,

```
<?xml version="1.0"?>
```

is actually a processing instruction.

XML And Alcohol

You've already seen how XML allows you to use descriptive tags to mark up text in a document. These tags are usually free−form; a document author has complete freedom to name these tags anything he or she desires. And while this flexibility is one of the reasons for XML's popularity, it's a double−edged sword, because it begs the question: what happens if tag names in different documents clash with each other?

An example might help to make this clearer. Let's suppose that I decided to encode my stock portfolio as an XML document. Here's what it might look like:

```
<?xml version="1.0"?>

<portfolio>

<stock>Cisco Systems</stock>
<stock>Nortel Networks</stock>
<stock>eToys</stock>
<stock>IBM</stock>

</portfolio>
```

And now let's suppose that Tom, my next−door neighbour and the proud owner of his own computer store, hears about XML, gets really excited, and assigns some of his employees to the task of encoding his store's inventory into XML. Here's what his XML document might look like:

```
<?xml version="1.0"?>

<inventory>

<category>Mice</category>
<item>Mouse C106</item>
<vendor>Logitech</vendor>
<stock>100</stock>

<category>Handhelds</category>
<item>Visor Deluxe</item>
<vendor>HandSpring</vendor>
<stock>23</stock>

<category>MP3 players</category>
<item>Nomad</item>
```

```
<vendor>Creative</vendor>
<stock>2</stock>

</inventory>
```

Finally, let's suppose that Tom and I get together for a drink, tell each other about our XML experiments and (in a moment of tequila−induced clarity) decide to put XML's capabilities to the test by combining our two documents into one. However, since both documents include a tag named

```
<stock>
```

whose meaning is entirely dependent on its context, it's pretty obvious that our attempt at integration will fail, since an XML application would have no way of telling whether the data enclosed between <stock>...</stock> tags belonged to my portfolio or Tom's inventory.

It's precisely to avoid this kind of ambiguity that the XML specification now provides for namespaces. Namespaces are a way to uniquely identify specific elements within an XML document. This is accomplished by assigning a unique prefix to an element, thereby immediately associating it with a particular data universe and eliminating ambiguity.

The Name Game

Setting up a namespace is simple − here's the syntax:

```
<elementName xmlns: prefix="namespaceURL">
```

In this case, the prefix is the unique string used to identify the namespace; this is linked to a specific namespace URL.

A namsepace is usually declared at the root element level, although authors are free to declare it at a lower level of the tree structure too.

Once the namespace has been declared within the document, it can be used by prefixing each element within that namespace with the unique namespace identifier. Take a look at my revised stock portfolio, which now uses a "mytrades" namespace to avoid name clashes.

```
<mytrades:portfolio
xmlns:mytrades="http://www.somedomain.com/namespaces/mytrades/">

<mytrades:stock>Cisco Systems</mytrades:stock>
<mytrades:stock>Nortel Networks</mytrades:stock>
<mytrades:stock>eToys</mytrades:stock>
<mytrades:stock>IBM</mytrades:stock>

</mytrades:portfolio>
```

And once Tom gets over his hangover, I guess he'd find it pretty easy to fix his document too.

```
<?xml version="1.0"?>

<toms_store:inventory
xmlns:toms_store="http://www.toms_store.com/">

<toms_store:category>Mice</toms_store:category>
<toms_store:item>Mouse C106</toms_store:item>
<toms_store:vendor>Logitech</toms_store:vendor>
<toms_store:stock>100</toms_store:stock>
```

```
<toms_store:category>Handhelds</toms_store:category>
<toms_store:item>Visor Deluxe</toms_store:item>
<toms_store:vendor>HandSpring</toms_store:vendor>
<toms_store:stock>23</toms_store:stock>

<toms_store:category>MP3 players</toms_store:category>
<toms_store:item>Nomad</toms_store:item>
<toms_store:vendor>Creative</toms_store:vendor>
<toms_store:stock>2</toms_store:stock>

</toms_store:inventory>
```

In case you're wondering, the namespace URL is simply a pointer to a Web address, and is meaningless in practical terms; the XML specification doesn't really care where the URL points, or even if it's a valid link.

In case a single document contains two or more namespaces, adding namespace declarations and prefixes to every element can get kind of messy − as the following example demonstrates.

```
<?xml version="1.0"?>

<me:person xmlns:me="http://www.mywebsite.com/">
My name is <me:name>Huey</me:name>. I'm <me:age>seven</me:age>
years old,
and I live in <me:address>Ducktown</me:address> with
<rel:relationships
xmlns:rel="http://www.mywebsite.com/relationships/">my
brothers
<rel:name>Dewey</rel:name> and
<rel:name>Louie</rel:name></rel:relationships>.
</me:person>
```

In such a situation, XML allows you to specify any one namespace as the default namespace, by omitting the prefix from the namespace declaration. Modifying the document above to make "me" the default namespace, we have

```
<?xml version="1.0"?>

<person xmlns="http://www.mywebsite.com/">
My name is <name>Huey</name>. I'm <age>seven</age> years old,
and I live in
```

```
<address>Ducktown</address> with <rel:relationships
xmlns:rel="http://www.mywebsite.com/relationships/">my
brothers
<rel:name>Dewey</rel:name> and
<rel:name>Louie</rel:name></rel:relationships>.
</person>
```

Namespaces need not be restricted to elements alone − attributes can use namespaces too, as the following example demonstrates.

```
<?xml version="1.0"?>

<mystuff:collection
xmlns:mystuff="http://www.somedomain.com/mystuff"
xmlns:music="http://www.somedomain.com/music_genres">

<mystuff:artist>Spears, Britney</mystuff:artist>
<mystuff:title music:genre="Pop">Oops, I Did It
Again!</mystuff:title>

</mystuff:collection>
```

An Entity In The Attic

XML entities are a bit like variables in other programming languages − they're XML constructs which are referenced by a name and store text, images and file references. Once an entity has been defined, XML authors may call it by its name at different places within an XML document, and the XML parser will replace the entity name with its actual value.

XML entities come in particularly handy if you have a piece of text which recurs at different places within a document − examples would be a name, an email address or a standard header or footer. By defining an entity to hold this recurring data, XML allows document authors to make global alternations to a document by changing a single value.

Consider the following simple example:

```
<?xml version="1.0"?>

<!DOCTYPE article
[
<!ENTITY copyright "This material copyright Melonfire, 2001.
All rights
reserved.">
]>

<article>

<title>XML Basics (part 2)</title>

<abstract>A discussion of basic XML theory</abstract>

<body>
&copyright;
Article body goes here
&copyright;
</body>

</article>
```

Entities come in two parts. First comes the entity definition, which always appears within the document type declaration at the head of the document (after the prolog). In this case, the entity "copyright" has ben defined and mapped to the string "This material copyright Melonfire, 2001. All rights reserved."

```
<!ENTITY copyright "This material copyright Melonfire, 2001.
All rights
reserved.">
```

Once an entity has been defined, the next step is to use it. This is accomplished via entity references, placeholders for entity data within the document markup. Typically, an entity reference contains the entity name prefixed with either an ampersand ()or a percentage (%) symbol and suffixed with a semi−colon(;), as below:

```
<body>
&copyright;
Article body goes here
&copyright;
</body>
```

When a parser reads an XML document, it replaces the entity references with the actual values defined in the document type declaration. So this document

```
<?xml version="1.0"?>

<!DOCTYPE article
[
<!ENTITY copyright "This material copyright Melonfire, 2001.
All rights
reserved.">
]>

<article>
<title>XML Basics (part 2)</title>

<abstract>A discussion of basic XML theory</abstract>

<body>
&copyright;
Article body goes here
&copyright;
</body>

</article>
```

would look like this once a parser was through with it.

```
<?xml version="1.0" ?>

<title>XML Basics (part 2)</title>

<abstract>A discussion of basic XML theory</abstract>

<body>This material copyright Melonfire, 2001. All rights
reserved. Article
body goes here This material copyright Melonfire, 2001. All
rights
reserved.</body>

</article>
```

Note that entities must be declared before they are referenced, and must appear within the document type declaration. If a parser finds an entity reference without a corresponding entity declaration, it will barf and produce some nasty error messages.

XML comes with the following five pre−defined entities:

&lt; − represents the less−than (<) symbol.

&gt; represents the greater−than (>) symbol

&apos; represents the single−quote (') symbol

&quote; represents the double−quote(") symbol

&amp; represents the ampersand ()symbol

Entities can contain XML markup in addition to ordinary text − the following is a perfectly valid entity declaration:

```
<!ENTITY copyright "This material copyright
<link>Melonfire</link>,
<publication_year>2001</publication_year>. All rights
reserved.">
```

Entities may be "nested"; one entity can reference another. Consider the following entity declaration, which illustrates this rather novel concept.

```
<!ENTITY company "Melonfire">
<!ENTITY year "2001">
<!ENTITY copyright "This material copyright , . All rights
reserved.">
```

Note, however, that an entity cannot reference itself, either directly or indirectly, as this would result in an infinite loop (most parsers will warn you about this.) And now that I've said it, I just know that you're going to try it out to see how much damage it causes.

Digging For Treasure

Entities come in a variety of flavours. They can broadly be divided into general entities and parameter entities. The examples you've seen above are general entities; since parameter entities are used only with DTDs, you don't need to worry about them for the moment.

Entities may be further classified into internal entities (entities defined within the document), external entities (entities defined in a separate file), and unparsed entities (entities which are not processed by the parser).

Most of the examples you've seen so far use internal entities − that is, the entity declaration and entity references are stored in the same physical document. XML also allows you to separate the entity declaration from the entity reference by storing it in a separate file, which comes in handy when the entity declaration contains a large block of text. Consider the following example:

```
<?xml version="1.0"?>

<!DOCTYPE article
[
<!ENTITY header "All source code copyright and proprietary
Melonfire, 2001.
All content, brand names and trademarks copyright and
proprietary
Melonfire, 2001. All rights reserved. Copyright infringement
is a violation
of law. This source code is provided with NO WARRANTY
WHATSOEVER. It is
meant for illustrative purposes only, and is NOT recommended
for use in
production environments. Read more articles like this one at
<url>http://www.melonfire.com/community/columns/trog/</url>
and
<url>http://www.melonfire.com/</url>">
]>

<article>
<title>XML Basics (part 2)</title>

<abstract>A discussion of basic XML theory</abstract>

<body>

Article body goes here
</body>

</article>
```

Since the entity contains a fairly large block of text, it may be more convenient to extract it and store it in a separate file, "header.xml". In that case, the example above would reduce to

```
<?xml version="1.0"?>

<!DOCTYPE article
[
<!ENTITY header SYSTEM "header.xml">
]>

<article>
<title>XML Basics (part 2)</title>

<abstract>A discussion of basic XML theory</abstract>

<body>

Article body goes here
</body>

</article>
```

In this case, the SYSTEM keyword is used to tell the parser the location of the file containing the replacement text for the entity.

Unparsed entities usually contain references to images, sound files or other binary data, and hence should not be processed by a parser (jeez, you think maybe that's why they're called "unparsed entities"?) Such entity declarations usually contain a link to the file (as with external entities) followed by an additional notation identifier which specifies the type of file.

In the following example, the NDATA keyword is used to tell the parser that the file being referenced is not to be processed in the usual manner; it is followed by a file type specification offering further information on the nature of the file.

```
<?xml version="1.0"?>

<!DOCTYPE article
[
<!ENTITY map SYSTEM "treasuremap.jpg" NDATA JPEG>
]>
```

```
<message>
To whoever finds this: here is a map showing the location of
pirate
treasure on an island in the South Pacific.



Remember, X marks the spot.
</message>
```

The Man From IDIOT

Like HTML and most programming languages, XML also allows you to place comments within an XML document. A comment is simply an explanatory statement in plain English, intended to help others to understand and read your document. Comments are ignored by the parser, and are meant only for readability purposes − it's good programming practice to use them in your code.

Comments may appear anywhere within an XML document, and are similar to those used in HTML − a text string enclosed between <!−− and −−> markers. Here's an example:

```
<?xml version="1.0"?>

<report>
<headline>Alien Life On Earth, Says IDIOT Official</headline>
<date> July 23, 2001</date>
<place>Alaska</place>
<reporter>Joe Cool</reporter>

<body>

<!−− who says you can't fool all of the people all of the time
−−>
In a not−unexpected turn of events, an IDIOT (I Doubt It's Out
There)
official today confirmed reports of alien sightings in Area
−10, the
coldest part of Northern Alaska, and again called on Pentagon
officials to
either confirm or deny that the sightings were part of a
decade−long
government project to breed alien lifeforms on Earth. IDIOT
also claims to
have a map displaying the exact location of the alien "farm",
and states
that it will be released to the press within the next
forty−eight hours.
However, posing as an IDIOT, this intrepid reporter has
successfully
obtained a copy of said map, reproduced below:

<!−− thanks, Mom −−>
<map> </map>

</body>
```

```
</report>
```

Endgame

And that's about it for this crash course in XML theory. You now know enough to begin encoding XML documents on your own, as well as begin reading some of the more advanced material available on the subject. Here are a few links to get you started:

The W3C's XML specification, at http://www.w3.org/TR/2000/REC−xml−20001006

The Annotated XML specification, at http://www.xml.com/pub/a/axml/axmlintro.html

The W3C's Namespaces in XML specification, at http://www.w3.org/TR/1999/REC−xml−names−19990114/

Microsoft's XML Web site, at http://msdn.microsoft.com/xml/default.asp

XML.com, one of the best Web sites for articles and columns on XML, at http://www.xml.com/

XML articles and papers at http://xml.coverpages.org and http://www.xml−zone.com/articles.asp

Don't stray too far, though − what you've just learned is merely the tip of a very large iceberg, and over the next few weeks, I'll be delving into the next level of detail, discussing things like DTD design, XSL transformations and XLink data linkages.

See you soon!