Item 3 Stay with XML 1.0

Everything you need to know about XML 1.1 can be summed up in two rules.

1. Don't use it.
2. (For experts only) If you speak Mongolian, Yi, Cambodian, Amharic, Dhivehi, Burmese, or a very few other languages and you want to write your markup (not your text but your markup) in these languages, you can set the `version` attribute of the XML declaration to 1.1. Otherwise, refer to rule 1.

XML 1.1 does several things, one of them marginally useful to a few developers, the rest actively harmful.

- It expands the set of characters allowed as name characters.
- The C0 control characters (except for NUL) such as form feed, vertical tab, BEL, and DC1 through DC4 are now allowed in XML text provided they are escaped as character references.
- The C1 control characters (except for NEL) must now be escaped as character references.
- NEL can be used in XML documents but is resolved to a line feed on parsing.
- Parsers may (but do not have to) tell client applications that Unicode data was not normalized.
- Namespace prefixes can be undeclared.

Let's look at these changes in more detail.

New Characters in XML Names

XML 1.1 expands the set of characters allowed in XML names (that is, element names, attribute names, entity names, ID-type attribute values, and so forth) to allow characters that were not defined in Unicode 2.0, the version that was extant when XML 1.0 was first defined. Unicode 2.0 is fully adequate to cover the needs of markup in English, French, German, Russian, Chinese, Japanese, Spanish, Danish, Dutch, Arabic, Turkish, Hebrew, Farsi, Thai, Hindi, and most other languages you're likely to be familiar with as well as several thousand you aren't. However, Unicode 2.0 did miss a few important living languages including Mongolian, Yi, Cambodian, Amharic, Dhivehi, and Burmese, so if you want to write your markup in these languages, XML 1.1 is worthwhile.

However, note that this is relevant only if we're talking about *markup*, particularly element and attribute names. It is not necessary to use XML 1.1 to write XML data, particularly element content and attribute values, in these languages. For example, here's the beginning of an Amharic translation of the Book of Matthew written in XML 1.0.

```
<?xml version="1.0" encoding="UTF-8">
<book>
<title>የማቴዎስ  ወንጌል</title>
<chapter number="፩">
<title>የኢየሱስ የትው ልድ ሐረግ</title>
<verse number="፩">
 የዳዊት  ልጅ፣ የአከርሃም ልጅ የሁ ነው  የኢየሱስ ከርስቶስ የትው  ልድ ሐረግ የሚከተለው  ነው፤
```

```
</verse>
<verse number="፪">
  ኢከርሃም ይሰሐቅን ወለደ፤
  ይሰሐቅ ያዕቆብን  ወለደ፤
  ያዕቆብ ይሁዳንና ወንድሞቹን ወለደ፤
</verse>
</chapter>
</book>
```

Here the element and attribute names are in English although the content and attribute values are in Amharic. On the other hand, if we were to write the element and attribute names in Amharic, we would need to use XML 1.1.

```
<?xml version="1.1" encoding="UTF-8">
<መጽሐፋ>
<ኢርእስት>የማቴዎስ  ወንጌል</ኢርእስት>
<ምዕራፋ  ዌጥር="፩">
<ኢርእስት>የኢየሱሰ የትው ልድ ሐረግ</ኢርእስት>
<ቤት  ዌጥር="፩">
የዳዊት  ልጅ፣ የኢከርሃም ልጅ የሁ ነው የኢየሱሰ ከርሰቶሰ የትው ልድ ሐረግ የሚከተለው  ነው፤
</ቤት>
<ቤት  ዌጥር="፪">
 ኢከርሃም ይሰሐቅን ወለደ፤
  ይሰሐቅ ያዕቆብን  ወለደ፤
  ያዕቆብ ይሁዳንና ወንድሞቹን ወለደ፤
</ቤት>
</ምዕራፋ>
</መጽሐፋ>
```

This is plausible. A native Amharic speaker might well want to write markup like this. However, the loosening of XML's name character rules have effects far beyond the few extra languages they're intended to enable. Whereas XML 1.0 is conservative (everything not permitted is forbidden), 29837 01 pp001-288 r9ah.ps 8/18/03 4:40 PM Page 13

XML 1.1 is liberal (everything not forbidden is permitted). XML 1.0 lists the characters you can use in names. XML 1.1 lists the characters you can't use in names. Characters XML 1.1 allows in names include:

- Symbols like the copyright sign (©)
- Mathematical operators such as ±
- Superscript 7 (^{7})
- The musical symbol for a six-string fretboard
- The zero-width space
- Private-use characters
- Several hundred thousand characters that aren't even defined in Unicode and probably never will be

XML 1.1's lax name character rules have the potential to make documents much more opaque and obfuscated.

C0 Control Characters

The first 32 Unicode characters with code points from 0 to 31 are known as the C0 controls. They were originally defined in ASCII to control teletypes and other monospace dumb terminals. Aside from the tab, carriage return, and line feed they have no obvious meaning in text. Since XML is text, it does not include binary characters such as NULL (#x00), BEL (#x07), DC1 (#x11) through DC4 (#x14), and so forth. These noncharacters are historical relics. XML 1.0 does not allow them. This is a good thing. Although dumb terminals and binary-hostile gateways are far less common today than they were twenty years ago, they are still used, and passing these characters through equipment that expects to see plain text can have nasty consequences, including disabling the screen. (One common problem that still occurs is accidentally paging a binary file on a console. This is generally quite ugly and often disables the console.)

A few of these characters occasionally do appear in non-XML text data. For example, the form feed (#x0C) is sometimes used to indicate a page break. Thus moving data from a non-XML system such as a BLOB or CLOB field in a database into an XML document can unexpectedly cause malformedness errors. Text may need to be cleaned before it can be added to an XML document. However, the far more common problem is that a document's encoding is misidentified, for example, defaulted as UTF-8 when it's really UTF-16 or ISO-8859-1. In this case, the parser will notice unexpected nulls and throw a well-formedness error.

XML 1.1 fortunately still does not allow raw binary data in an XML document. However, it does allow you to use character references to escape the C0 controls such as form feed and BEL. The parser will resolve them into the actual characters before reporting the data to the client application. You simply can't include them directly. For example, the following document uses form feeds to separate pages.

```
<?xml version="1.1">
<book>
  <title>Nursery Rhymes</title>
  <rhyme>
    <verse>Mary, Mary quite contrary</verse>
    <verse>How does your garden grow?</verse>
  </rhyme>
  &#x0C;
  <rhyme>
    <verse>Little Miss Muffet sat on a tuffet</verse>
    <verse>Eating her curds and whey</verse>
  </rhyme>
  &#x0C;
  <rhyme>
    <verse>Old King Cole was a merry old soul</verse>
    <verse>And a merry old soul was he</verse>
  </rhyme>
</book>
```

However, this style of page break died out with the line printer. Modern systems use stylesheets or explicit markup to indicate page boundaries. For example, you might place each separate page inside a `page` element or add a `pagebreak` element where you wanted the break to occur, as shown below.

```
<?xml version="1.1">
<book>
  <title>Nursery Rhymes</title>
  <rhyme>
    <verse>Mary, Mary quite contrary</verse>
    <verse>How does your garden grow?</verse>
  </rhyme>
  <pagebreak/>
  <rhyme>
```

```
    <verse>Little Miss Muffet sat on a tuffet</verse>
    <verse>Eating her curds and whey</verse>
  </rhyme>
  <pagebreak/>
  <rhyme>
    <verse>Old King Cole was a merry old soul</verse>
    <verse>And a merry old soul was he</verse>
  </rhyme>
</book>
```

Better yet, you might not change the markup at all, just write a stylesheet that assigns each rhyme to a separate page. Any of these options would be superior to using form feeds. Most uses of the other C0 controls are equally obsolete.

There is one exception. You still cannot embed a null in an XML document, not even with a character reference. Allowing this would have caused massive problems for C, C++, and other languages that use null-terminated strings. The null is still forbidden, even with character escaping, which means it's still not possible to directly embed binary data in XML. You have to encode it using Base64 or some similar format first. (See Item 19.)

C1 Control Characters

There is a less common block of C1 control characters between 128 (#x80) and 159 (#x9F). These include start of string, end of string, cancel character, privacy message, and a few other equally obscure characters. For the most part these are even less useful and less appropriate for XML documents than the C0 control characters. However, they were allowed in XML 1.0 mostly by mistake. XML 1.1 rectifies this error (with one notable exception, which I'll address shortly) by requiring that these control characters be escaped with character references as well. For example, you can no longer include a "break permitted here" character in element content or attribute values. You have to write it as `&#x82;` instead.

This actually does have one salutary effect. There are a lot of documents in the world that are labeled as ISO-8859-1 but actually use the nonstandard Microsoft Cp1252 character set instead. Cp1252 does not include the C1 controls. Instead it uses this space for extra graphic characters such as €, Œ, and ™. This causes significant interoperability problems when moving documents between Windows and non-Windows systems, and these problems are not always easy to detect.

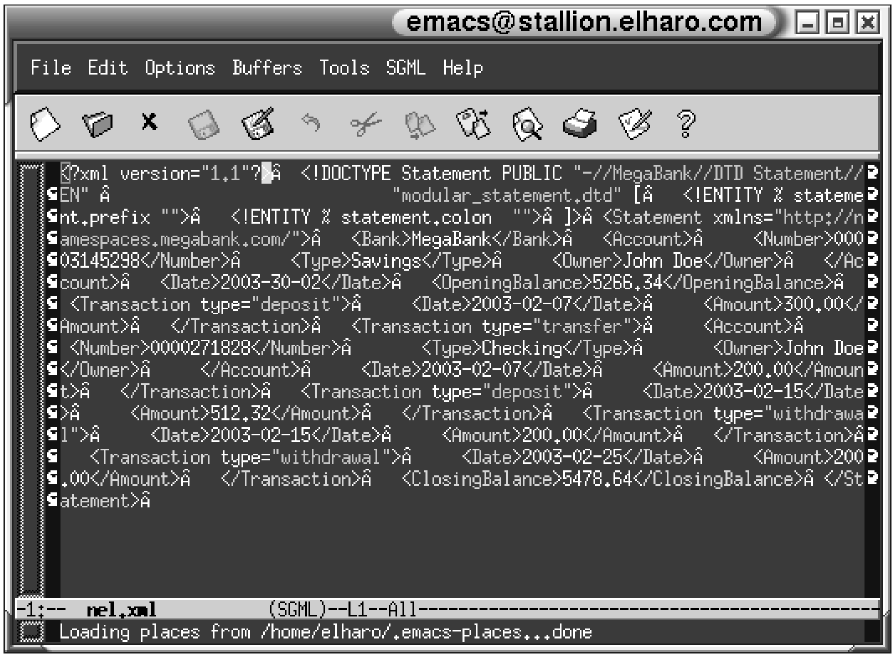
By making escaping of the C1 controls mandatory, such mislabeled documents will now be obvious to parsers. Any document that contains an unescaped C1 character labeled as ISO-8859-1 is malformed. Documents that correctly identify themselves as Cp1252 are still allowed.

The downside to this improvement is that there is now a class of XML documents that is well-formed XML 1.0 but not well-formed XML 1.1. XML 1.1 is not a superset of XML 1.0. It is neither forward nor backward compatible.

NEL Used as a Line Break

The fourth change XML 1.1 makes is of no use to anyone and should never have been adopted. XML 1.1 allows the Unicode next line character (#x85, NEL) to be used anywhere a carriage return, line feed, or carriage return–line feed pair is used in XML 1.0 documents. Note that a NEL doesn't mean anything different than a carriage return or line feed. It's just one more way of adding extra white space. However, it is incompatible not only with the installed base of XML software but also with all the various text editors on UNIX, Windows, Mac, OS/2, and almost every other non-IBM platform on Earth. For instance, you can't open an XML 1.1 document that uses NELs in emacs, vi, BBEdit, UltraEdit, jEdit, or most other text editors and expect it to put the line breaks in the right places. Figure 3–1 shows what happens when you load a NEL-delimited file into emacs. Most other editors have equal or bigger problems, especially on large documents.

If so many people and platforms have such problems with NEL, why has it been added to XML 1.1? The problem is that there's a certain huge monopolist of a computer company that doesn't want to use the same standard everyone else in the industry uses. And—surprise, surprise—its name isn't Microsoft. No, this time the villain is IBM. Certain IBM mainframe software, particularly console-based text editors like XEdit and OS/390 C compilers, do not use the same two line-ending characters (carriage return and line feed) that everybody else on the planet has been using for at least the last twenty years. Instead those text editors use character #x85, NEL.


emacs@stallion.elharo.com
File Edit Options Buffers Tools SGML Help

```
<?xml version="1.1"?Â  <!DOCTYPE Statement PUBLIC "-//MegaBank//DTD Statement//
EN" Â                             "modular_statement.dtd" [Â   <!ENTITY % stateme
nt.prefix "">Â   <!ENTITY % statement.colon  "">Â ]>Â <Statement xmlns="http://n
amespaces.megabank.com/">Â   <Bank>MegaBank</Bank>Â   <Account>Â     <Number>000
03145298</Number>Â     <Type>Savings</Type>Â     <Owner>John Doe</Owner>Â   </Ac
count>Â   <Date>2003-30-02</Date>Â   <OpeningBalance>5266.34</OpeningBalance>Â
 <Transaction type="deposit">Â     <Date>2003-02-07</Date>Â     <Amount>300.00</
Amount>Â   </Transaction>Â   <Transaction type="transfer">Â     <Account>Â
 <Number>0000271828</Number>Â       <Type>Checking</Type>Â       <Owner>John Doe
</Owner>Â     </Account>Â     <Date>2003-02-07</Date>Â     <Amount>200.00</Amoun
t>Â   </Transaction>Â   <Transaction type="deposit">Â     <Date>2003-02-15</Date
>Â     <Amount>512.32</Amount>Â   </Transaction>Â   <Transaction type="withdrawa
l">Â     <Date>2003-02-15</Date>Â     <Amount>200.00</Amount>Â   </Transaction>Â
   <Transaction type="withdrawal">Â     <Date>2003-02-25</Date>Â     <Amount>200
.00</Amount>Â   </Transaction>Â   <ClosingBalance>5478.64</ClosingBalance>Â </St
atement>Â

-1:--  nel.xml          (SGML)--L1--All-------------------------------------------
Loading places from /home/elharo/.emacs-places...done
```

Figure 3–1 | Loading a NEL-Delimited File into a Non-IBM Text Editor

If you're one of those few developers writing XML by hand with a plain console editor on an IBM mainframe, you should upgrade your editor to support the line-ending conventions the rest of the world has standardized on. If you're writing C code to generate XML documents on a mainframe, you just need to use \x0A instead of \n to represent the line end. (Java does not have this problem.) If you're reading XML documents, the parser should convert the line endings for you. There's no need to use XML 1.1.

Unicode Normalization

For reasons of compatibility with legacy character sets such as ISO-8859-1 (as well as occasional mistakes) Unicode sometimes provides multiple representations of the same character. For example, the e with accent acute (é) can be represented as either the single character #xE9 or with the two characters #x65 (e) followed by #x301 (combining accent acute). XML 1.1 suggests that all generators of XML text should normalize such alternatives into a canonical form. In this case, you should use the single character rather than the double character.

However, both forms are still accepted. Neither is malformed. Furthermore, parsers are explicitly prohibited from doing the normalization for the client program. They may merely report a nonfatal error if the XML is found to be unnormalized. In fact, this is nothing that parsers couldn't have done with XML 1.0, except that it didn't occur to anyone to do it. Normalization is more of a strongly recommended best practice than an actual change in the language.

Undeclaring Namespace Prefixes

There's one other new feature that's effectively part of XML 1.1: namespaces 1.1, which adds the ability to undeclare namespace prefix mappings. For example, consider the following API element.

```
<?xml version="1.0" encoding="UTF-8">
<API xmlns:public="http://www.example.com"
      xmlns:private="http://www.example.org" >
  <title>Geometry</title>
  <cpp xmlns:public="" xmlns:private="">
    class CRectangle {
      int x, y;
      public:void set_values (int,int);
      private:int area (void); }
  </cpp>
</API>
```

A system that was looking for qualified names in element content might accidentally confuse the `public:void` and `private:int` in the `cpp` element with qualified names instead of just part of C++ syntax (albeit ugly C++ syntax that no good programmer would write). Undeclaring the public and private prefixes allows them to stand out for what they actually are, just plain unadorned text.

In practice, however, very little code looks for qualified names in element content. Some code does look for these things in attribute values, but in those cases it's normally clear whether or not a given attribute can contain qualified names. Indeed this example is so forced precisely because prefix undeclaration is very rarely needed in practice and never needed if you're only using prefixes on element and attribute names.

That's it. There is nothing else new in XML 1.1. It doesn't move namespaces or schemas into the core. It doesn't correct admitted mistakes in the design of XML such as attribute value normalization. It doesn't simplify XML by removing rarely used features like unparsed entities and notations. It doesn't even clear up the confusion about what parsers should and should not report. All it does is change the list of name and white space characters. This very limited benefit comes at an extremely high cost. There is a huge installed base of XML 1.0–aware parsers, browsers, databases, viewers, editors, and other tools that don't work with XML 1.1. They will report well-formedness errors when presented with an XML 1.1 document.

The disadvantages of XML 1.1 (including the cost in both time and money of upgrading all your software to support it) are just too great for the extremely limited benefits it provides most developers. If you're more comfortable working in Mongolian, Yi, Cambodian, Amharic, Dhivehi, or Burmese and you only need to exchange data with other speakers of one of these languages (for instance, you're developing a system exclusively for a local Amharic-language newspaper in Addis Ababa where everybody speaks Amharic), you can set the `version` attribute of the XML declaration to 1.1. Everyone else should stick to XML 1.0.