Creating Well-Formed XML Documents

By Andrew H. Watt

Mar 28, 2003

Other Characteristics of Well-Formedness

This section covers the well-formedness constraints as they apply to areas other than elements and attributes. Several of the well-formedness rules will make more sense after you have read Chapters 4 and 5.

Comments

To be well formed, XML comments must not end with the character sequence `--->`. A legal XML comment ends with `-->`— that is, two dashes and a greater than character.

Entities Must Be Declared

If a reference to an entity is present in an XML document and any of the following are true, any entity referenced must be declared:

- There is no Document Type Definition (DTD).
- A document has only an internal DTD subset with no parameter entity references.
- The value of the `standalone` attribute in the XML declaration is `yes`.

The exception to this rule is that the built-in entities `amp`, `apos`, `gt`, `lt`, and `quot` need not be declared.

External Parsed Entities

External parsed entities must be well-formed. They optionally begin with a *text declaration* (described more fully in Chapter 5). A text declaration is similar to an XML declaration but may have only `version` and `encoding` attributes. A text declaration does not have a `standalone` attribute. Following the text declaration, the structure need not all be contained in a single element—the document element is contained in the XML document entity that references the external parsed entity.

The allowed content is any combination of the following:

- Character data
- Elements
- Entity references
- Character references
- `CDATA` sections
- Processing instructions
- Comments

The content following the text declaration is essentially the same as the permitted content of an element anywhere in the document entity. That is not surprising because an external parsed entity can have replacement text that constitutes the content of an element in the doc ument entity.

An internal parsed entity is well-formed if the replacement text matches the content that was listed in the preceding list.

No element or other markup may begin in one external entity and end in another.

Parsed Entities: No Recursion

Entities are discussed in more detail in Chapter 5. A parsed entity is not permitted to directly or indirectly reference itself.

Parameter Entity References in the DTD

Parameter entity references may appear only in the DTD.

Parameter Entity References in Internal Subset

Parameter entity references in the internal subset of the DTD, which is described more fully in Chapter 4, can occur only where markup declarations can occur and not within other markup declarations.

External Subset of the DTD

The external subset of the DTD may have the following structure. It may optionally begin with a text declaration (described in Chapter 5) and may also contain markup declarations or con ditional sections (both described in Chapter 4).

ditional sections (both described in Chapter 4).

Two separators are allowed between markup declarations, parameter entities and whitespace.

Parameter Entities in Markup Declarations

A parameter entity has replacement text. The replacement text of a parameter entity must satisfy the constraints on declarations, as described in the preceding section, so that the re placement text nests properly.

Replacement Text

When an entity reference appears in an attribute value or a parameter entity appears in an entity declaration, the replacement text might contain either a double quotation mark or an apo strophe. In this situation, when the double quotation mark or the apostrophe is applied as re placement text, it is treated as a literal character of the relevant type, not the closing delimiter of the attribute value.

For example, with the entity declaration

```
<!ENTITY myEntity "something with an apostrophe'">
```

this entity reference is well-formed:

```
<someElement someAttribute='&myEntity' />
```

The same is true with a parameter entity such as this one:

```
<!ENTITY % hisStatement '"I agree."'>
<!ENTITY aSentence "He said, &hisStatement;">
```

The replacement text for the `hisStatement` parameter entity contains two double quotation marks, the first of which would normally be the closing delimiter of the replacement text of the `aSentence` entity. However, both double quotation marks contained in the parameter reference are treated literally, not as the closing delimiter.

Character References

In XML documents, a character may be referenced using a character reference. You might want to use a character reference when, for example, a character cannot be typed from the key board. A character reference beginning with `&x#` indicates a hexadecimal reference to a character's code point. For example, the uppercase `A` may be written as the character reference, `&x#0041;`.

```
<?xml version='1.0'?>
<capitalA>&#x0041;</capitalA>
```

Character references can also be expressed using the `&#` syntax, indicating a decimal reference to a character's code point. Using this syntax, you can represent the uppercase `A` as `&#0065;`, as shown here:

```
<capitalA>&#0065;</capitalA>
```

Declaring Predefined Entities

If compatibility with SGML is not an issue in your use of XML, this well-formedness constraint can perhaps be ignored. However, it might help you understand what at first sight could appear to be strange entity declarations in XML documents created by others.

All XML processors must recognize the entities `amp`, `apos`, `gt`, `lt`, and `quot`, whether they are explicitly declared or not. However, if compatibility with SGML is a relevant issue, these pre defined entities must be explicitly declared in the internal subset of the DTD.

You might recall that you cannot use the literal characters `&`, `'`, `>`, `<`, and `"` in well-formed character data without causing a well-formedness error. Therefore, you cannot use any of these characters literally as the replacement text of an entity declaration. The solution is to use character references for each of the characters.

Therefore, entity declarations used for compatibility with SGML can be written as follows:

```
<!ENTITY amp "&#x26; &#38">
<!ENTITY apos "&#x27; &#39">
<!ENTITY gt "&#x3E; &#62">
<!ENTITY lt "&#x3C; &#60">
<!ENTITY quot "&#x22; &#34">
```


< Back Page 5 of 7 Next >