UsingXML <blog>Sam Page</blog>

White Space in XML Documents

Understanding how white space works in XML documents can help keep you out of trouble when you're working with a variety of XML technologies. In this essay we'll learn how XML parsers treat white space and the fundamental mechanisms for controlling white space in XML documents. We'll also look at some white space handling behavior particular to Microsoft's XML services.

Consider the white space in the following XML document:

```
<?xml version="1.0" ?>
<List name="Fruit List">
   <Item>Apple</Item>
   <Item>Banana</Item>
   <Item>Pear</Item>
</List>
```

The document contains some white space that delimits various aspects of the XML syntax. When the white space is part of the XML syntax, it is discarded by XML parsers and not passed on to processing applications. XML allows for unbounded white space wherever white space is permitted in the XML syntax. This is useful for pretty printing an XML document.

In the figure below, the locations where white space that's part of the XML syntax may appear are marked with a (·) dot:

```
<?xml·version·=·"1.0"·?>
<List·name·=·"Fruit List"·>
   <Item·>Apple</Item·>
   <Item·>Banana</Item·>
   <Item·>Pear</Item·>
</List·>·
```

White space in *any* other location *must* be passed on to the processing application, according to the XML specification. In the figure below, the locations for this *significant* white space are marked with a (·) dot:

```
<?xml version="1.0" ?>
<List name="·Fruit·List·">·
   ·<Item>·Apple·</Item>·
   ·<Item>·Banana·</Item>·
   ·<Item>·Pear·</Item>·
</List>
```

Now when the XML specification says *any* white space, they don't really mean it. HA! The standards leave some aspects of white space handling up to the implementers, or at least that's what the implementers would have us believe. I suspect some implementers choose to ignore parts of the standards they don't like or can't accommodate easily in their toolsets. It's inevitable that different XML parsers make different interpretations of the standards. This leads to some fuzzy behavior where white space is concerned.

Attribute White Space Handling

The first exception to the *significant* white space rule deals with attribute values. The XML parser uses a set of rules to *normalize* attribute values. The rules are specified in the XML specification in section 3.3.3 Attribute-Value Normalization. Here is the gist of the rules:

- Replace all white space characters with space characters.
- Expand character references to characters.
- Recursively expand entity references to characters.
- If a Document Type Definition (DTD) is present and the attribute is declared as a non-CDATA type, trim leading and trailing space and collapse consecutive white space to a single space.

Without getting into what the CDATA type is, by default, with no DTD, attribute values are treated as CDATA, so the last white space collapsing rule above doesn't apply.

CDATA normalization seems half-baked compared to non-CDATA normalization. It neither wholly preserves nor wholly cleans up white space. The reason for this must lie in some SGML legacy or perhaps out of consideration for a use case where really long attribute values are split with new line characters over multiple lines and need to be considered one continuous string joined by spaces. Either way, just having extra white space left unmolested in attributes is not an option.

The non-CDATA normalization is useful when dealing with numeric types or date formats that may need to be parsed and validated further by your application. The XPath `normalize-space` function performs the same trimming and space collapsing as the non-CDATA normalization.

Element White Space Handling

Validating parsers using Document Type Definitions (DTD) and XML Schemas give you a little more control over how white space is treated in your XML documents.

If a DTD or Schema declares an element to contain only child element nodes and not text nodes, then a validating XML parser can safely throw away the white space between elements. In the figure below, assume the `List` element was declared as containing only `Item` elements in a DTD. The additional white space that could now be *discarded as insignificant* is marked with a (·) dot:

```
<?xml version="1.0" ?>
<List name="Fruit List">·
   ·<Item>Apple</Item>·
   ·<Item>Banana</Item>·
   ·<Item>Pear</Item>·
</List>
```

In other words, the *insignificant* space is part of the pretty printing of the document and not part of the content of the document. By contrast, if the element is declared as having *mixed* content, both text and element child nodes, then the XML parser must pass on all the white space found within the element.

Validating parser white space behavior is a decidedly fuzzy area. The behavior described above is consistent with the Microsoft XML parser.

XML Schema White Space Control

XML Schema gives more flexibility than DTDs for controlling how a validating XML parser deals with white space. Using the `whiteSpace` facet of a data type allows you to specify `preserve`, `replace`, or `collapse` white space handling for element and attribute content.

The `whiteSpace` facet is described in XML Schema Part 2: Datatypes in section 4.3.6 whiteSpace. Here is a quick description of the `whiteSpace` facet values:

- `preserve` is the default and keeps all white space.
- `replace` replaces all white space characters with spaces (like CDATA).
- `collapse` trims and collapses white space (like non-CDATA).

Microsoft XML Parser Behavior

If you've been working with Microsoft's XML parser and DOM implementation, especially if you've been using the Microsoft XSLT processor, you might be scratching your head and saying, “Hey! This isn't the way white space handling works in Microsoft's XML tools!” And you would be partially correct. The Microsoft XML parser's white space handling is true to the specification, but by default, their DOM builder aggressively throws out white space.

When Microsoft's DOM builder receives a text node from the parser that contains only white space, it is thrown away. From a practical perspective, this is a pretty savvy approach on Microsoft's part. Most of the time the extra white space is *insignificant*, and likely a result of pretty printing the XML. Throwing away the extra white space can save a lot of memory in the DOM and make DOM performance faster with fewer text nodes. It's easy to pretty print the XML and restore that kind of white space when saving, especially with the built-in formatting features of the Microsoft `XmlTextWriter` class.

If you need the white space preserved more consistently with other tool sets, then simply set the `preserveWhiteSpace` property of the `XmlDocument` object to `true` *before* loading the XML document. Bear in mind that this causes the DOM to consume more memory. You'll seldom find it necessary in practice to override Microsoft's DOM loading default behavior.

The xml:space Attribute

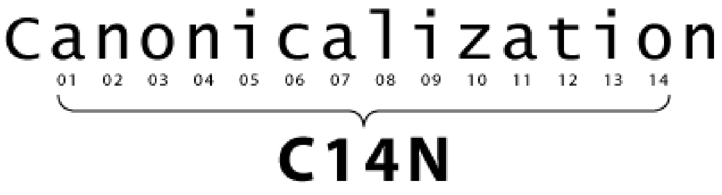
The `xml:space` attribute is another standard mechanism that exists for preserving white space in XML applications. It's described in the XML specification in section 2.10 White Space Handling. The `xml:space` attribute can be placed on any elements in the XML document and given a value of `preserve` to signal that the white space is *significant*. The `xml:space` behavior cascades to all descendant elements but can be turned off locally by setting the `xml:space` attribute to `default`. In order to use `xml:space` in a validating context, the attribute must be declared in a DTD or Schema attribute list for the elements in which it is used.

The `xml:space` attribute is one of the standards-based mechanisms that you can use in all the XML applications you create. We'll discuss some of the other `xml:*` attributes (`xml:base`, `xml:lang` and `xml:id`) later in this series.

Treating White Space in a Uniform Way

XML documents can vary widely by *insignificant* white space but produce identical results from an XML parser. If you need to compare two XML documents it would be nice if you could write the XML and its white space in a uniform way so that comparisons can be made more easily with traditional diff tools. The Canonical XML 1.0 specification provides a set of rules for writing XML documents in a uniform way.

XML canonicalization is commonly referred to as C14N, which is a bit of an inside joke in XML standards circles.



Considering that my spell check keeps suggesting cannibalization as a substitute for canonicalization, C14N is a pretty handy abbreviation. You may also run into I18N as shorthand for *internationalization* among other W3C standards. Lazy bastards!

The rules for C14N are numerous but not difficult to understand. In addition to white space rules, attribute ordering, namespace declaration ordering, and character entity reference formats are specified by C14N. Here is an *incomplete* list of C14N rules:

- Remove any XML declaration and document type declarations
- Encode document in UTF-8
- Expand entities to their character equivalent
- Replace CDATA sections with their character equivalent
- Encode the special XML entities `&lt; &gt; &quot;`
- Normalize attribute values, as if by a validating parser
- Open empty elements with start and end tags
- Sort namespace declarations and attributes

Consider the following C14N sample:

Before C14N

```
1 |<?xml version="1.0" ?>
2 |<List verified="true" name="Fruit List" count="3">
3 |   <Item>Apple</Item>
4 |   <Item>Banana</Item>
5 |   <Item />
6 |</List>
```

After C14N

```
1 |<List count="3" name="Fruit List" verified="true">
2 |   <Item>Apple</Item>
3 |   <Item>Banana</Item>
4 |   <Item></Item>
5 |</List>
```

Note that the attributes of the `List` element are sorted, the XML declaration is removed, and the empty `Item` element on line four has been opened.

Microsoft provides a C14N implementation in the `XmlDsigC14NTransform` class of the System.Security.Cryptography.Xml namespace in the .NET framework. You must add a reference to the `System.Security.dll` to your project before using it.

Exclusive Canonicalization
C14N plays an important role in developing web services. Digital signatures and other hash functions rely on a precisely consistent representation of the XML.
The Exclusive XML Canonicalization specification handles namespace issues surrounding the canonicalization of subsets of XML within other XML documents. In many messaging scenarios, it's desirable to have your message payload independent of the rest of the layers of the SOAP stack.

Be mindful of Microsoft's default white space handling if you're going to interoperate with non-Microsoft C14N implementations. If you don't act to preserve white space or if you don't use a validating parser, then the C14N process would result in the following single line result:

```
1 |<List count="3" name="Fruit List" verified="true">
- |   <Item>Orange</Item><Item>Grape</Item><Item></Item></List>
```

The specifications discussed in this article all impact white space handling and you should be comfortable with their use when designing XML applications. Vendor implementation differences will often be an area for fuzziness with regards to the specifications because both the implementations and specifications change over time. Familiarity with your XML toolset's white space handling is helpful. Use the Microsoft-specific white space handling samples above as a basis for exploring other vendor implementations.

I chose to cover white space first in this series because handling white space is especially important in code generation with XML and XSL transforms—a central topic of future essays. Code generation using XML technologies is a valuable programming technique that often requires precise control over white space.

References

XML 1.1 Specification, 2.10 White Space Handling
http://w3c.org/TR/2004/REC-xml11-20040204/#sec-white-space

XML 1.1 Specification, 3.3.3 Attribute-Value Normalization
http://w3c.org/TR/2004/REC-xml11-20040204/#AVNormalize

XML Schema Part 2: Datatypes, 4.3.6 whiteSpace
http://www.w3.org/TR/xmlschema-2/#dt-whiteSpace

Canonical XML Version 1.0
http://www.w3.org/TR/xml-c14n

Exclusive XML Canonicalization Version 1.0

http://www.w3.org/TR/xml-exc-c14n