Dealing with data in XML

Use the CDATA section effectively

Chris Herborth

January 12, 2010

Normally, when you store data in an XML file, you need to be careful about encoding it in a way that's safe and won't confuse the XML parser. The special XML markup characters need to be translated into entities, which can be cumbersome if you're writing the XML yourself in a text editor. To avoid this, you can use the CDATA section to store the data directly without having to worry about encoding. This article will tell you about XML CDATA sections and show you how to use them when you need to ship marked-up data along with your XML file.

Introduction

Frequently used acronyms

- Ajax: Asynchronous JavaScript + XML
- API: Application programming interface
- CSS: Cascading stylesheets
- DOM: Document Object Model
- DTD: Document Type Definition
- HTML: Hypertext Markup Language
- HTTP: Hypertext Transfer Protocol
- IIS: Internet Information Services
- LAN: Local area network
- MIME: Multipurpose Internet Mail Extensions
- UTF: Unicode Transformation Format
- VPN: Virtual Private Network
- XHTML: Extensible Hypertext Markup Language
- XML: Extensible Markup Language
- XSD: XML Schema Definition

XML is a well-supported Internet standard for encoding structured data in a way that can be easily decoded by practically any programming language and even read or written by humans using standard text editors. Many applications, especially modern standards-compliant Web browsers, can deal directly with XML data.

As a text-based standard, XML is well-suited for exchanging data between client and server systems. Much data is already text-based (file paths, descriptions, addresses, names, and so on), and things like integers, floating-point numbers, and dates can be easily converted to and from string representations.
Unfortunately, some data, such as XHTML or XML markup, is troublesome or cumbersome to include in an XML document. One method of putting markup into an XML element is to replace the markup characters [less than (&lt;), greater than (&gt;), and ampersand (&amp;)] with their equivalent entities (&lt;, &gt;, and &amp; respectively). This expands the data and makes it extremely hard for humans to read, not to mention the annoyance of translating markup if you write the XML manually in a text editor.

A better solution might be to put the data directly into your XML document. That's where XML's CDATA section comes into play.

### What is CDATA?

Text in an XML document is generally parsed character data, or (in Document Type Definition terms) PCDATA. XML's special characters (&, <, and >) are recognized in PCDATA and used to parse element names and entities. CDATA (character data) sections are treated as a block of data by the parser, allowing you to include any character in the data stream.

If you've ever tried to put some HTML or XML into an XML document, maybe as documentation, you've run into this problem as soon as it comes time to include an example. Listing 1 shows a simple paragraph sample with some emphasized text.

**Listing 1. Some sample XHTML in a sample element**

```xml
<?xml version='1.0' encoding='UTF-8'?>
<sample>
  <description>
    Paragraphs can include emphasized text.
  </description>

  <example>
    <p>The pug snoring on the couch next to me is <em>extremely</em> cute.</p>
  </example>
</sample>
```

It becomes a bit of a nightmare when you want to show the markup (see Listing 2).

**Listing 2. The sample XHTML with markup showing**

```xml
<?xml version='1.0' encoding='UTF-8'?>
<sample>
  <description>
    Paragraphs can include emphasized text.
  </description>

  <example>
    &lt;p&gt;The pug snoring on the couch next to me is &lt;em&gt;extremely&lt;/em&gt; cute.&lt;/p&gt;
  </example>
</sample>
```

Wrapping the sample markup in a CDATA section lets you write it as-is, without having the XML parser attempt to interpret it as a <p> element containing an <em> element. If your XML is being validated against a DTD or XML Schema, this is required (unless the elements actually exist in the DTD or XSD and can be included at that point in the document). See Listing 3.
### Listing 3. Using CDATA to protect the sample

```xml
<?xml version="1.0" encoding="UTF-8"?>
<sample>
  <description>
    Paragraphs can include emphasized text.
  </description>

  <example>
    <![CDATA[<p>The pug snoring on the couch next to me is <em>extremely</em> cute</p>]]>
  </example>
</sample>
```

### Using CDATA

As you can tell from the short example in Listing 3, a CDATA section starts with the special sequence `<![CDATA[` and ends with the `]]>` sequence. Anything between those bits of markup will pass through the XML parser untouched. Some development platforms have a special CDATA object (such as the CDATASection found in the XML DOM) to represent the contents of the CDATA section, but others will provide it as something more generic, generally an XML text node. In either case, the contents of the CDATA section will be available without modification.

Even though XML is generally very forgiving about white space, the `]]>` section ending cannot contain spaces or line breaks.

### CDATA in XHTML

You've probably seen CDATA in action if you've looked at many Web pages that have embedded JavaScript. You'll often see something like Listing 4.

### Listing 4. CDATA in XHTML's `<script>` element

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="application/xhtml+xml; charset=utf-8"/>
<title>CDATA Section in Action</title>
<script type="text/javascript">
  // <![CDATA[
  function nowWeAreSafe( x, y, z ) {
    // Without the CDATA section, these would cause
    // parsing errors:
    if( x < y && y > z ) {
      return y--;
    }
    return 0;
  }
  // ]]>
</script>
</head>
<body>...
</body>
</html>
```
The JavaScript in the `<script>` element starts with a comment containing the beginning of a CDATA section, and ends with a comment that closes off the CDATA section. This seems like a pointless way to make your XHTML and JavaScript noisier until you realize that without the CDATA section, your script is going to run through the Web browser's XHTML parser!

This isn't generally going to cause trouble unless you're very, very unlucky, but it can certainly cause parser errors that lead to confusing and hard-to-debug rendering errors. Why?

As you might have guessed, the `<`, `>`, and `&` characters could be flagged as elements or entities (or as stray markup characters). Also, the dash dash (`--`) sequence can be seen as the unexpected start (or end) of an XHTML comment block. In fact, that's the reason why you should wrap an embedded script in a CDATA section instead of an XML comment—comments are too fragile.

CDATA sometimes shows up in inline `<style>` elements as well, although this isn't nearly as common (see Listing 5).

Listing 5. CDATA prevents parsing errors in `<style>` elements

```html
<style type="text/css">
/* <![CDATA[ */
body { background-image: url("marble.png?width=300&height=300")
/* ]]> */
</style>
```

Note again how the CDATA markers are hidden inside of language-specific comments so they don't confuse the CSS parser in the client Web browser.

Limitations of CDATA

Clearly the CDATA section is useful, but like all good things, it has a couple of limitations for you to keep in mind.

Browsers aren't usually XML parsers

Browsers don't do CDATA in HTML or XHTML reliably, if at all. CDATA sections are allowed anywhere in XHTML (as they would be in any XML application) but in practice they're completely ignored. You'll either lose their contents (the CDATA section has vanished from the normal DOM) or have the contents rendered as text with some stray markup characters showing up.

To see this effect, look at a page that shows the sample paragraph, the sample paragraph with the markup visible (using entities), and an attempt to show the sample paragraph with the markup visible using CDATA. The XHTML page source is in Listing 6.

Listing 6. Trying to use CDATA in XHTML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

Dealing with data in XML
Firefox 3 eats the CDATA section's contents, as in Figure 1. (View a text-only version of Figure 1.)
Figure 1. Firefox ignores CDATA sections

CDATA Section in Action

A sample paragraph:
The pug snoring on the couch next to me is extremely cute.

Markup version:
<p>The pug snoring on the couch next to me is <em>extremely</em> cute.</p>

CDATA version:
Uh, where?
Wait, what?

WebKit-based browsers such as Safari and Chrome render it with spurious markup characters (see Figure 2). (View a text-only version of Figure 2.)

Figure 2. Safari and Chrome render CDATA sections

CDATA Section in Action

A sample paragraph:
The pug snoring on the couch next to me is extremely cute.

Markup version:
<p>The pug snoring on the couch next to me is <em>extremely</em> cute.</p>

CDATA version:
Uh,
The pug snoring on the couch next to me is extremely cute.

II> where?
Wait, what?
Internet Explorer® also renders it with similarly spurious markup characters (see Figure 3). (View a text-only version of Figure 3.)

**Figure 3. Internet Explorer 8 also renders CDATA sections**

![CDATA Section in Action](image)

Although browsers don't behave properly when CDATA sections are included in XHTML documents, they do have to handle them properly in XML documents loaded through Ajax. If they didn't, the browser's XML parser would be considered "non-conforming" and people would mock it mercilessly before marking it as horrifically broken for Ajax.

**Section end is still special**

Even though you can put anything into a CDATA section, the sequence for the section end marker, ```]]>``` is considered special. You absolutely cannot nest CDATA sections. If the XML parser reads this sequence, it's the end of your CDATA section and you might end up getting a parser error when it hits the real section end.

To put it another way, the XML (or XHTML) parser can't see your use of ```<![CDATA[``` inside a CDATA section because the parser ignores markup characters except for the section end marker ```]>``` (see Listing 7).
Listing 7. This is invalid XML. You can't nest CDATA sections

```xml
<?xml version="1.0" encoding="UTF-8"?>
<sample>
  <description>
    You can't nest CDATA sections.
  </description>

  <example>
    <![CDATA[You want a <![CDATA[ ]]> inside your example? No, this is wrong.]]>
  </example>
</sample>
```

What can you do if you need to put a section end marker into a CDATA section? You need to split it up into two CDATA sections (see Listing 8).

Listing 8. The right way to put a section end sequence inside a CDATA section

```xml
<?xml version="1.0" encoding="UTF-8"?>
<sample>
  <description>
    Split up the section end.
  </description>

  <example>
    <![CDATA[You want a ]]> <![CDATA[> inside your example? Do it this way.]]>
  </example>
</sample>
```

That is, replace any ]]> in your data with ]]> <![CDATA[> so the final > in the sequence is away from the brackets. The parser is looking for ]]> specifically as a three character sequence and by splitting it up, you broke the sequence.

And yes, ]]> <![CDATA[>] is a scary hunk of markup. Luckily, this situation doesn't come up very often.

**It's still text**

Even though the contents of the CDATA section pass through your parser untouched, they still need to be valid XML data characters, as specified by the document's character encoding. Using something like UTF-8 lets you use a huge range of characters for the data, but it's not 8-bit clean.

Any of the so-called control characters (those with a hex value below 0x20, the space character) can cause your parser to stop with an invalid token error. You can't take just any data and dump it into a CDATA section and still have a valid document.

**Size matters**

A final thing to keep in mind when adding chunks of data to your XML with CDATA sections is size. If you serve the XML files through a Web service, make sure that your client applications can deal with potentially large data transfers without timing out or blocking their user interface as the data trickles in over a 3G connection.
The reverse is also true; make sure your server can accept large up-stream transfers from clients sending XML data. Web servers (notably IIS on Windows® platforms) often have fairly small upload limits to help prevent denial of service attacks. Sending large blocks of data from the browser like this is error-prone (for example, what if the user cancels the transfer because they think it has crashed?) and it tends to lock up valuable resources on the server and the client.

And again, depending on what you're doing, you need to keep in mind that many people are using mobile platforms and others might also be stuck on dial-up connections (still!), assuming your application works outside of your LAN.

Even if you didn't design it that way, someone will try using it over a dial-up VPN connection on their iPhone, and they'll complain about your application's speed instead of their poor life choices!

**Storing binary data in XML**

When you do need to include some binary data in an XML document, you'll need to make sure it won't trip up the XML parser. If the data happens to be text, you can dump it into a CDATA section and be done with it, but true binary data needs to be encoded in a safe and recoverable manner.

Luckily the MIME standards define a safe encoding scheme that's well-supported, base64. The base64 encoding makes binary data approximately 137% its original size so you're trading off additional storage space (and a little processing throughput) for the ability to embed the binary data in your XML document.

Typically you'd want to indicate the encoding and original file name in your XML, as in Listing 9.

### Listing 9. One example of a base64-encoded file inside an XML document

```xml
<?xml version="1.0" encoding="UTF-8"?>
<sample>
    <description>
    An embedded image file.
    </description>
    <image name="stop.png" encoding="base64"
        source="FamFamFam"
        href="http://www.famfamfam.com/lab/icons/silk/">
        iVBORw0KGgoAAAANSUhEUgAAABAAAAAQ
        CAYAAAAAf8/9hAAAABGdBTUEAAK/INwWK
        6QAABB10RhvhU29mdHdhcmUAQWRvYmUg
        SwihZ2VSZWlkeXHJZTwAAAJOSURBVdjLp
        ZI9T1RBFiaf3buAoBgJ8rl6QVBJVND
        ShMLohj66t+wNdudjY9WmpBi9yPaJlFA
        ShJ+9gYVYRPonyYI527QWQ77Qzc2fG,
        Ypf1y0y2Mjzk5J5M5z/vO5ESstfxPxAAe
        rlXuh4pLnoaiUZdq7XAKg0R4VbIBZ3J
        PLJa09c/eCj/CFgZfN5qK5q8EhTXdx
        LKgqJAFr0NK8ppOptn9n1D2gdz2rmsvOE
        lVcVooprKvPtriNzYS8rH+H8ECoEqEg
        Wkly1c2P8akZby51p6G3b6QAWB1431IS
        VThuFuZEM3YbH916HKhSv0q4ENFNFd
        C=ui5JI9I/V2Y///rkShA1H6faVXx0V
        07CcgKCB7+fSB8dzy7mpa19/kh1Uw
        caII9h7+wRrs0lSylcspHClil1RXIV8M
        YDZ1rKYRjHE1ACNeGc/KCQ0ofW85nuV
```
In a machine-generated XML document, you can leave out the white space, and run the entire base64-encoded file together without newline characters.

**Avoiding the issue**

The best way to deal with binary data in XML is to avoid it entirely. As you've seen in HTML, referring to an external file in a standardized way works well. This is a great option when you have some way for the client application to get at the external file. In the case of HTML, the browser just makes another HTTP request to get the data included through elements like `<img>`.

By not including the binary data directly in the XML, you avoid potentially wasteful text encodings and make it possible to implement other enhancements, such as the image caching most people love in their Web browsers.

**Summary**

You can use XML's CDATA section, which starts with `<![CDATA` and ends with `]]>`, to keep part of your document away from the parser. The data inside will come out of the parser with exactly the same text that went in, although you'll need to protect any `]]>` sequences by stopping and restarting the CDATA section.

Even though you can't take advantage of CDATA sections in XHTML documents, XML is well-supported in browsers and regular programming platforms. Using CDATA to embed marked-up data directly in your XML documents keeps you from having to encode the data, but you need to be careful and consider the effect of (potentially) large data transfers on your client and server applications.

When you need to store binary data in an XML document, you can use a text encoding such as the standard MIME base64 encoding, although it's probably a better idea to reference an external file.
Related topics

- **Process XML in the browser using jQuery** (Uche Ogbugi, developerWorks, December 2009): Find out how to process XML directly in the browser with jQuery.
- **Combine social media APIs and XML-based data formats** (J. Jeffrey Hanson, developerWorks, December 2009): Read more about encoding data in XML files.
- **Multipurpose Internet Mail Extensions specification**: Learn more about multimedia standards for e-mail and other text-based mediums in the MIME specification, including the standard for base64 encoding (RFC 2045).
- **XML 1.0 Specification** (W3C Recommendation, 26 November 2008): Check out this source for specific details about XML features such as the CDATA section.
- **The XML FAQ**: Explore another excellent source of XML information, the XML FAQ edited by Peter Flynn.
- **Using XMLHttpRequest**: Read an excellent article about dealing with the results of Ajax calls.
- **XML Entity Definitions for Characters** (W3C Working Draft, 17 November 2009): Learn about several sets of names which are assigned to Unicode characters.
- **XHTML™ 1.0: The Extensible HyperText Markup Language** (World Wide Web Consortium Recommendation, 26 January 2000): Read more about XHTML 1.0, a reformulation of HTML 4 as an XML 1.0 application and provides the foundation for future extensibility of XHTML.
- **IBM certification**: Find out how you can become an IBM-Certified Developer.
- **XML technical library**: See the developerWorks XML Zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.
- **IBM product evaluation versions**: Get your hands on application development tools and middleware products.

© Copyright IBM Corporation 2010

Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)