XML Character Encoding and Decoding

January 2013

Table of Contents

1. Excellent quotes

> ... around 1972 ... "Why is character coding so complicated?" ... it hasn't become any simpler in the intervening 40 years.
>
> To be able to track end-to-end the path of conversions and validate that your application from authoring through to storage through to search and retrieval is completely correct is amazingly difficult ... it's a skill far too few programmers have, or even recognize that they do not have.
>
> We can't really tell what's going on without access to your entire tool chain.
>
> It's possible that your editor changed the character encoding of your text when you changed the XML declaration (emacs does this)!
>
> It's unlikely that the encoding of the characters in this email is byte-identical with the files you created.
>
> ... my preferred solution is to stick to a single encoding everywhere

... I vote for UTF-8 ...

... make sure *every single link in the chain* uses that encoding.

2. Lots of character conversions taking place inside our computers and on the Web

All of the characters in the following XML are encoded in iso-8859-1:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Name>López</Name>
```

Now consider this problem:

> Suppose we search the XML for the string López,
> where the characters in López are encoded in UTF-8.
> Will the search application find a match?

I did the search and it found a match.

How did it find a match?

The underlying byte sequence for the iso-8859-1 López is: 4C F3 70 65 7A (one byte -- F3 -- is used to encode ó).

The underlying byte sequence for the UTF-8 López is: 4C C3 B3 70 65 7A (two bytes -- C3 B3 -- are used to encode ó).

The search application cannot be doing a byte-for-byte match, else it would find no match.

The answer is that the search application either converted the XML to UTF-8 or converted the search string (López) to iso-8859-1.

Inside our computers, inside our software applications, and on the Web there is a whole lot of character encoding conversions occurring ... transparently ... without our knowing.

3. Well-formedness error when encoding="..." does not match actual character encoding

Create an XML document and encode all the characters in it using UTF-8:

```
<Name>López</Name>
```

Add an XML declaration and specify that the encoding is iso-8859-1:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Name>López</Name>
```

There is a mismatch between what encoding="..." says and the actual encoding of the characters in the document.

Check the XML document for well-formedness and you will NOT get an error.

Next, create the same XML document but encode all the characters using iso-8859-1:

```
<Name>López</Name>
```

Add an XML declaration and specify the encoding is UTF-8:

```
<?xml version="1.0" encoding="utf-8"?>
<Name>López</Name>
```

Again there is a mismatch between what encoding="..." says and the actual encoding of the characters in the document.

But this time when you check the XML document for well-formedness you WILL get an error.

Here's why.

In UTF-8 the ó symbol is encoded using these two bytes: C3 B3

In iso-8859-1, C3 and B3 represent two perfectly fine characters, so the UTF-8 encoded XML is a fine encoding="iso-8859-1" document.

In iso-8859-1 the ó symbol is encoded using one byte: F3

F3 is not a legal UTF-8 byte, so the iso-8859-1 encoded XML fails as an encoding="UTF-8" document.

4. How to generate an encoding well-formedness error

George Cristian Bina from oXygen XML gave the scoop on how things work inside oXygen.

a. Create an XML document and encode all the characters in it using iso-8859-1:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Name>López</Name>
```

b. Using a hex editor, change encoding="iso-8859-1" to encoding="utf-8":

```
<?xml version="1.0" encoding="utf-8"?>
<Name>López</Name>
```

c. Drag and drop the file into oXygen.

d. oXygen will generate an encoding exception:

```
Cannot open the specified file. Got a character
encoding exception [snip]
```

George described the encoding conversions that occur inside oXygen behind-the-scenes:

> If you have an iso-8859-1 encoded XML file loaded into oXygen
> and change encoding="iso-8859-1" to encoding="utf-8" then
> oXygen will automatically change the encoding of every character
> in the document to UTF-8.

George also made this important comment:

> Please note that the encoding is important only when the file is loaded
> and saved. When the file is loaded the bytes are converted to characters
> and then the application works only with characters. When the file is
> saved then those characters need to be converted to bytes and the
> encoding used will be determined from the XML header with a default to
> UTF-8 if no encoding can be detected.

5. If everyone used UTF-8 would that be best for everyone?

There are a multiplicity of character encodings and a huge number of character encoding conversions taking place behind-the-scene. If everyone used UTF-8 then there would be no need to convert encodings.

Suppose every application, every IDE, every text editor, and every system worldwide used one character encoding, UTF-8.

Would that be a good thing?

The trouble with that is that UTF-8 makes larger files than UTF-16 for great numbers of people who use ideographic scripts such as Chinese. The real choice for them is between 16 and 32.

UTF-16 is also somewhat harder to process in some older programming languages, most notably C and C++, where a zero-valued byte (NUL, as opposed to the zero-valued machine address, NULL) is used as a string terminator.

So UTF-8 isn't a universal solution.

There isn't a single solution today that's best for everyone.

See this excellent discussion on StackOverflow titled, *Should UTF-16 be Considered Harmful?*

http://programmers.stackexchange.com/questions/102205/should-utf-16-be-considered-harmful

In that discussion one person wrote:

> After long research and discussions, the development conventions at my company ban using UTF-16 anywhere except OS API calls ...

6. Interoperability of XML (i.e., Character Encoding Interoperability)

Remember not long ago you would visit a web page and see strange characters like this:

> â€œGood morning, Daveâ€

You don't see that much anymore.

Why?

The answer is this:

> Interoperability is getting better.

In the context of character encoding and decoding, what does that mean?

Interoperability means that you and I interpret (decode) the bytes in the same way.

Example: I create an XML file, encode all the characters in it using UTF-8, and send the XML file to you.

Here is a graphical depiction (i.e., glyphs) of the bytes that I send to you:

<Name>López</Name>

You receive my XML document and interpret the bytes as iso-8859-1.

In UTF-8 the ó symbol is a graphical depiction of the "LATIN SMALL LETTER O WITH ACUTE" character and it is encoded using these two bytes: C3 B3

But in iso-8859-1, the two bytes C3 B3 is the encoding of two characters:

C3 is the encoding of the Ã character
B3 is the encoding of the ³ character

Thus you interpret my XML as:

<Name>LÃ³pez</Name>

We are interpreting the same XML (i.e., the same set of bytes) differently.

Interoperability has failed.

So when we say:

Interoperability is getting better.

we mean that the number of incidences of senders and receivers interpreting the same bytes differently is decreasing.

Let's revisit our first example, you open a document and see this:

â€œGood morning, Daveâ€

Here's how that happened:

I created a document, using a UTF-8 editor, containing this text:

“Good morning, Dave”

The left quote character is U+201C and the right quote character is U+201D.

You open the document using Microsoft Word (or any Windows-1252 editor) and see:

> â€œGood morning, Daveâ€□

Here's why:

In UTF-8 the left smart quote is codepoint 201C, which is encoded inside the computer as these hex values: E2 80 9C

In Windows-1252 the byte E2 is displayed as â and the byte 80 is displayed as € and the byte 9C is displayed as œ

In UTF-8 the right smart quote is codepoint 201D, which is encoded inside the computer as these hex values: E2 80 9D

In Windows-1252 the byte E2 is displayed as â and the byte 80 is displayed as € and the byte 9D has no representation

7. Acknowledgements

Thanks to the following people for their contributions to this paper:

David Allen
Roger Costello
John Cowan
Jim DeLaHunt
Ken Holman
Michael Kay
David Lee
Chris Maloney
Liam Quin
Michael Sokolov
Andrew Welch
Hermann Stam-Wilbrandt