Subject: Re: Small bug in DeXml
Posted by rylek on Sat, 07 Mar 2009 11:30:13 GMT
View Forum Message <> Reply to Message

Hello there!

Perhaps the main problem is that our U++ XML parser goes a few steps beyond the XML parser logic as defined by the standard, implementing a few tasks which the standard delegates to the end application using the XML interface. For instance, honoring or discarding whitespace characters (spaces, tabs, end-of-line characters) is, according to the standard, a task for the processing application. Under normal circumstances, it seemed to us much more useful if the XML parser did a little normalization on its own, and that includes disposing of leading and trailing whitespace when xml:space is not set to "preserve". For instance, it would be often burdensome if the parser had to return every standalone linebreak between tags as a text entity and if the application had to skip all leading and trailing whitespace when parsing structured data.

I admit this can be undesirable in certain situations and it might be useful to extend the parser to more strictly conform to the specification by not performing these additional tasks, albeit at a greater cost to the host application which would have to take care of all this by itself. Discarding or honoring superfluous whitespace, however, shouldn't depend (as I read the XML standard) on whether the characters were originally represented in the XML file directly or by escape sequences. The section you mention in fact just guarantees that UNIX and Windows files parse to the same thing no matter which CR-LF combination they use for actual line breaks; application-dependent handling of these whitespaces (currently built into our XML parser) constitutes an independent next step.

As concerns different types of data potentially stored into XML documents, I believe there are really three such types: 1) semantic encoded data like numbers or dates, in which leading and trailing whitespaces usually don't matter at all; 2) chunks of text consisting of printable characters plus spaces, tabs, and end-of-line characters which are normally supposed to be preserved, and 3) arbitrary binary data sequences which must be preserved on a byte-per-byte basis and which cannot be directly stored into XML at all (among others exactly because of the CR-LF normalization which might distort the data).

As I understand from your original mail, the data which is currently your concern belongs to the 2nd category. What I said was just that, in my opinion, the XML standard envisions such types of data, and if you are getting them from our XML parser distorted, it is not because they are not sufficiently escaped, but because of the reasons described above; if you wish, because our XML parser is not sufficiently conformant to the standard. That's why I believe that fixing the problem by escaping linefeeds on output to the XML file would not make a better XML (in fact it would make a much worse XML consisting of potentially infinitely long lines and severely complicating direct readability and editability using plaintext viewers and editors, which has always been a conceptual advantage, if not an inherent property, of XML compared to other, especially binary formats), rather it would patch a design flaw thereby inventing a non-standard XML construct which our parser would see differently from other parsers.

To make the long story short: as the current U++ XML parser stands, I believe you could solve your problem by adding the xml:space attribute to a tag surrounding your text; the "application part" of the XML parser logic understands it and it will return your text undistorted whether LF's are escaped or not. That means, if you don't need to store here other strange non-printable characters, in which case, as we both seem to agree, it would be best to avoid direct plaintext storage at all and convert the binary data prior to storing in XML using an ASCII binary data expansion algorithm like BinHex, Base64 or ASCII85. I'm not sure whether I'd call this the "customization of XML", from my point of view it's just a matter of selection of an appropriate serialization tool to represent a certain datatype, just one which should be used sparingly as it breaks the direct readability and introduces artificial entropy in the data.

I admit that, to make hardcore XML purists happier, it might be wise to introduce a "better compatibility" mode of the XML parser (perhaps using a constructor parameter or a member attribute) which would, among others, preserve all whitespaces when passing character data to the application which would then treat them accordingly. Please also note that the U++ XML parser is currently not designed as a "complete" parser implementing all features defined in the standard, but rather as a "tiny" tool enabling simple use of XML-conformant files for easy storage and retrieval of application configuration and other structured data. If it showed up that a truly "official" XML parser was necessary, I believe that it would be in fact much easier to implement an existing parser like Expat using the plugin technology rather than trying to code all this in the existing parser.

Regards

Tomas