Subject: Small bug in DeXml
Posted by phirox on Fri, 06 Mar 2009 19:00:45 GMT
View Forum Message <> Reply to Message

I use XML for storing settings, and even communication through sockets. But I found a weird thing, you can't put a \n in a string and then parse it back. Eventually I discovered this is because of the function DeXml() in the file Core/XML.cpp, in this part:

```
else if((byte)*s < ' ' && *s != '\n') result.Cat(NFormat("&#x%02x;", (byte)*s));
```

This should be changed to the code below.

```
else if((byte)*s < ' ') result.Cat(NFormat("&#x%02x;", (byte)*s));
```

I don't see any reason why it should not be coded.

Subject: Re: Small bug in DeXml
Posted by rylek on Fri, 06 Mar 2009 19:56:11 GMT
View Forum Message <> Reply to Message

Hi Phirox!

The trouble is, that even if we recode LF as &#x0A;, you still aren't guaranteed you'll parse back exactly what you wrote into the XML previously, unless you use the xml:space="preserve" attribute. I believe that under normal circumstances you should use this attribute to ensure that the delimiting whitespaces are honored when parsing XML text elements. It might be useful to extend the DeXml function to allow escaping LF (at least based on an additional flag parameter or by making a separate version of the function for this purpose), but then it's questionable whether it shouldn't also escape leading and trailing blanks. Although the XML standard specifically doesn't forbid it, I personally find the concept of storing arbitrary binary data sequences as &#-escaped plain text into XML at the very least questionable; after all, we're trying not to invent our own XML but to use the one that's supposed to be a widespread standard, i.e. readable by non-U++ software without difficulties and possibly without semantic differences as well.

Regards

Tomas

Subject: Re: Small bug in DeXml
Posted by phirox on Sat, 07 Mar 2009 00:07:23 GMT
View Forum Message <> Reply to Message

Quote:The trouble is, that even if we recode LF as &#x0A;, you still aren't guaranteed you'll parse back exactly what you wrote into the XML previously, unless you use the xml:space="preserve" attribute. I believe that under normal circumstances you should use this attribute to ensure that the delimiting whitespaces are honored when parsing XML text elements.

In this case you are actually guaranteed that the LF character gets through since it is escaped. Also the xml:space="preserve" attribute only works as the name implies; for whitespace. All end of line characters should even be normalized to #xA according to the specifications, so there is no way to ever preserve any combination. See here.

I can't think of any reason you wouldn't want to escape it. Not doing so actually breaks the current usability and is outside of the specifications.

Quote:Although the XML standard specifically doesn't forbid it, I personally find the concept of storing arbitrary binary data sequences as &#-escaped plain text into XML at the very least questionable; after all, we're trying not to invent our own XML but to use the one that's supposed to be a widespread standard, i.e. readable by non-U++ software without difficulties and possibly without semantic differences as well.

Even though this is quite unrelated to the bug; there is no other way to officially do binary encoding in XML then the &#-escapes. Doing something like a BASE64 function is actually "inventing your own XML" which btw is what XML is all about; customization.

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

Subject: Re: Small bug in DeXml
Posted by mirek on Sat, 07 Mar 2009 12:24:07 GMT
View Forum Message <> Reply to Message

> phirox wrote on Fri, 06 March 2009 19:07
> In this case you are actually guaranteed that the LF character gets through since it is escaped. Also the xml:space="preserve" attribute only works as the name implies; for whitespace. All end of line characters should even be normalized to #xA according to the specifications, so there is no way to ever preserve any combination. See here.

Well, I have checked it and after careful reading I cannot say I agree with what you see.

Indeed, xml:space="preserve" has probably slightly different meaning than Tom expects - if I understand the document well, it would only affect elimination of *empty* lines. However, it looks like for lines that are not empty, CR, CR-LF and LF should always be granted.

Maybe the real problem is not in DeXml, but in XmlParser. It is also quite hard to define what is "empty line", e.g. in case:

```
<tag>
Something
</tag>
```

I would expect to get "Something" without any line endings, unless "preserve" is active.

Also

```
<tag>
Something

</tag>
```

I still want to get just "Something"...

I am looking forward to further discussion of this topic, personally I see no big problem with fixing DeXml, but I think Tom is right too.

Mirek

Subject: Re: Small bug in DeXml
Posted by mirek on Sat, 07 Mar 2009 12:28:08 GMT

View Forum Message <> Reply to Message

One more issue, perhaps we should use your DeXml variant for attributes....

Mirek

Subject: Re: Small bug in DeXml
Posted by rylek on Sat, 07 Mar 2009 13:26:10 GMT
View Forum Message <> Reply to Message

I'm actually glad to have a discussion on this subject. When I originally wrote the XML parser and when Mirek rewrote it later on, we both had in mind (well, at least I had, I had better let Mirek explain his thoughts by himself) a rather specific tool trading absolute generality and 100% standard conformance for maximum ease of integration into our U++ applications. The current model works quite well when serializing ordinary structured hierarchical data like various configuration settings, but, as Mirek pointed out later on, it's not very well suited for more complicated scenarios like e.g. when XML serializing a formatted text document - imagine a XML-like RTF; doesn't OpenOffice use something like that?

When Mirek asks how exactly the "something" retrieved from his example tag should "finally" look, the standard in fact doesn't answer this. Please note the adjective "finally"; formally the standard says (as I understand it) that the "something" should be returned with all surrounding whitespace. This means, that the sequence

```
<tag>
something
</tag>
```

should be returned as (in C notation) "\nsomething\n". By the above "finally" I mean that the "\nsomething\n" string is then passed to the application, which, according to the standard, should process the whitespaces either in the "preserving" manner or in its "default" manner which is not specified in greater detail.

I believe that this notion of "default" vs. "preserving" manner of processing whitespaces is closely related to the conceptual kinds of data stored into XML I described above. Here the difference is that, in the 1st case, the serialization algorithm used to linearize and decode the data being stored is of such nature that it doesn't depend on the exact number of whitespaces anywhere; it may need blanks to separate entities (like in a number list) but one blank is usually as good as any number of successive blanks, LF's are freely interchangeable with spaces and leading and trailing blanks are generally ignored. Thus in the above example, if the <tag> expected an enumerated text constant as its value, the "\nsomething\n" would have to be stripped of its leading and trailing linefeed characters prior to being matched to an internal application table when being decoded.

On the other hand, the 2nd data type requires whitespace to be exactly preserved one-by-one (very much like the <pre>-formatted text in HTML). This is typically the case when you want to store an user-editable (potentially multiline) text in an XML tag. Not even empty lines (i.e. successive LF's) can be generally removed here as they might alter the semantics or even syntax of the data being serialized.

From this point of view I believe that it is impossible to give a general answer to Mirek's question, how should the parser process his example tag. Either the parser must always return the text element exactly as given together with all surrounding whitespace, intermediate sequences of empty lines etc., which we currently chose not to as it would make further XML data processing in applications unnecessarily complicated because it seems to us that in practice most data fall into the 1st category. If the parser is designed to help the host application a little, as it is now, it must somehow "decide" which records belong to the 1st and which to the 2nd category, there is no way to detect this automatically. When I came across this problem some time ago when using XML to store project definitions in one of my applications (Hydrocheck), it seemed natural to me to overcome it by implementing the xml:space attribute handler in the parser. Another possibility is either to make the XML parser revert to the "raw" processing as intended by the standard, or to split its public interface methods (for reading attributes and text elements) into pairs, one method for retrieving the "raw" data according to the standard, the other for retrieving the "normalized" data easier for further processing in applications.

As concerns tag attributes, I myself believe that in order to avoid running into compatibility problems, it is best to limit their content to the 1st data kind in the above sense. Mind you, this is not a quotation from the standard, just my personal opinion.

Regards

Tomas

Subject: Re: Small bug in DeXml
Posted by phirox on Sat, 07 Mar 2009 13:38:04 GMT

View Forum Message <> Reply to Message

Thank you both for your reactions.

Rereading this part, I have to agree that blank lines are preserved thus showing that a \n character should not be ignored by the parser and considered part of the data. As every end-of-line combination is normalized to LF for inter-compatibility; encoding CR, which is currently done, is correct. But as far as I can tell this only goes for tag elements.

Quote:One more issue, perhaps we should use your DeXml variant for attributes....

This is where I found the problem. In my application I use the Xmlize function to store a String object. At first I did this as a tag, but later I switched to arguments because of whitespace. But the parser for arguments does not pick up \n breaks. As I believe it shouldn't, because here breaks are ignored but whitespace is not.

edit: as reply to Tomas's last post, I agree that tags should be left alone for now. And the function should only adapt for arguments as I explained above.

Subject: Re: Small bug in DeXml
Posted by mirek on Sat, 07 Mar 2009 19:15:49 GMT
View Forum Message <> Reply to Message

OK, so the summary is to escape LF in attributes.

Fixed.

Mirek