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
