XML and TPF - the new world meets the old

by Morry Veer

This article is a reflection of a process I started around one year ago - the process of learning XML. While evaluating it for use both on my website and on my résumé, I came to realize that this was where the internet was headed, and that it would be invaluable on TPF. I offered to summarize my knowledge to several people, and when I offered it to the good people (person?) at *TPF Today*, I was supported whole-heartedly. Thus begins what may turn out to be a very open ended project.

XML is, quite simply, a structured data format using plain text - no binary data. Very similar to EDIFACT, but without the committees, the recommended data elements, and so on. XML is actually an outgrowth of the same source as HTML: SGML, which has been an ISO standard since 1986 and has been linked with IBM. XML is more of a Meta-data format, limiting and describing how you structure all data. Within that framework you create your own structures, also known as Document type definitions.

So why XML? Well, for one, it is rapidly becoming the standard for cross platform data communications over the internet. With the ever increasing interconnectivity of TPF systems and internet applications (Orbitz, Travelocity, Expedia) the pressure will only increase for TPF to connect to the world wide web. And it needs to be done quickly – the term „internet time“ was not coined without reason. It seems that, despite its virtues, we have failed to establish EDIFACT as the „standard for everyone“, so XML has come along and filled that gap. With XML being designated as a core technology in Microsoft's .NET strategy, I think we can all be assured that XML will be around for quite some time.

My objectives with this project are twofold. The first is to describe enough about XML so that the general TPF community understands its capabilities. The second is to pursue the structure of developing an XML engine within a TPF system, and analyze the potential pitfalls, dilemmas, and optimizations.

Before I continue, I should point out that IBM has already ported an XML engine to the TPF platform, and is making it available with the PUT 14 TPF release. I encourage everyone to go read about it online (see below). It covers many of the points I will raise here, but there is also a large point of division. Because the majority of the TPF community is not familiar with C++, I plan on discussing XML from the standard TPF platform point of view – databases and assembler. I myself know C++ quite well, but I presume less than expert knowledge in my target audience.

XML - Lots of Character(s)

XML is character based. That means that you should be able to open up an XML document with a text editor and be able to read and edit it. XML is based upon the Unicode standard, which means it supports more than the 80 or so characters with which most English speakers are familiar. Just to make things interesting, there are "… a grand total of 94,140 encoded characters in Unicode 3.1."[1] This creates a few problems. The first is that 94k characters cannot be encoded in 8 bits, or even 16. Unicode for all intensive purposes is a 32 bit character encoding scheme. The lucky part is that most business related applications and communications, especially the kind that would be useful in TPF, would be using the standard Latin characters 99.999% of the time, if not always. Given that, you can simplify processing considerably by either rejecting, ignoring or creating special case processing for characters outside the standard set.

The other difficulty comes with the character encoding. The first 127 character values x'00' – x'7f' in Unicode are ASCII characters. There needs to be a translation into EBCDIC for processing on the TPF box. There already exists a UTF (Unicode translation format) that does what we need: UTF-EBCDIC. It preserves all the information in the original document, by creating multi-byte sequences for the Unicode characters higher than x'7f', and translates everything into EBCDIC (see link below).

TPF is above all a system designed for performance, and some flexibility has and will be sacrificed. It would be nice if we could use Chinese characters (Hanzi) for passengers from Beijing, but – TPF isn't there yet. How do you calculate the length of a name using only 2 Hanzi, but maybe 8 bytes in UTF-EBCDIC? To this end I propose the following solution: reject XML documents back to the sender (presuming that's possible) if they use Unicode characters beyond x'7f'. Translate the ASCII into EBCDIC, and treat it as if the translation will eventually become UTF-EBCDIC.[2]

Note that all this presumes we need to work in EBCDIC. There is no reason why the XML processor cannot work in Unicode and translate individual pieces of data as requested by the application. I leave this as an exercise for the reader (I have *always* wanted to say that).

Footnotes

[1] From unicode.org, last updated 10.08.2001

[2] anyone familiar with Unicode will undoubtedly be aware of big-endian and little-endian issues. I am essentially ignoring these issues, however they would be addressed in the same way as the character encoding. It is most likely that no matter how much simplification we try to achieve, some sort of whole-document transformation will be necessary.

Links and Resources

UTF-EBCDIC	http://www.unicode.org/unicode/reports/tr16/
XML.org	http://www.xml.org
Worldwide web consortium	http://www.w3.org
Unicode	http://www.unicode.org
TPF TODAY	http://www.tpftoday.com/
This article online	http://members.bigfoot.com/~morryveer/index.html/tpfxml
IBM TPF	http://www-4.ibm.com/software/ts/tpf/
XML for TPF online users guide	http://www-4.ibm.com/software/ts/tpf/pubs/xml/xhome.htm
Apache project	http://xml.apache.org/