Date : 20/07/2006

Unicode implementation in UNIMARC: expectations and reality

Olga Zhlobinskaya
National Library of Russia

Meeting:	**77 UNIMARC**
Simultaneous Interpretation:	**Yes**

WORLD LIBRARY AND INFORMATION CONGRESS: 72ND IFLA GENERAL CONFERENCE AND COUNCIL
20-24 August 2006, Seoul, Korea
http://www.ifla.org/IV/ifla72/index.htm

Unicode providing for encoding characters of practically all known written languages seems to be a brilliant instrument for library systems since now we have (at least we like to think so) the ability of multiscript support for MARC-records. In fact Unicode provides merely a *potential* for multiscript support rather than a ready solution, and to implement this potential we need to solve a number of new problems imposed with the arrival of Unicode.

Talking on Unicode, we must remember, that Unicode is first of all code table assigning numbers to characters. For computers to be able to understand sequences of these numbers, every character should be represented as a unique sequence of bytes. Unicode defined a number of such character encoding forms, the most widely used are UTF-8, UTF-16 and UTF-32.
UTF-8 is the UNICODE transformation format that allows UNICODE values to be represented as variable-length sequences of 8-bit bytes (octets). UTF-8 transforms every Unicode character into a sequence of one to four octets. ASCII characters are coded with single octet. All other characters are encoded with two or more bytes. The most significant bits of the first byte of a multi-byte sequence determine how many bytes follow for this character. Theoretically, 2^{31} characters can be encoded, but in practice ca. 96 000 code positions are used. So, UTF-8 is compatible with ASCII – any files that contain only ASCII characters will have the same encoding both under ASCII and UTF-8, thus preserving the basic structural elements of the MARC record. On the other hand, UTF-8 is able to represent any Unicode character.

Besides backwards compatibility with ASCII, there is another feature of UTF-8, which is worth of mentioning. For any text containing mostly ASCII characters, UTF-8 helps to save storage space, which may be of some importance in ISO 2709.

In case of UTF-16 and UTF-32 each character is encoded with two- or four-bytes code values respectively. Any multi-byte string can be stored with the most significant byte first or last. The former is called big-endian, the latter little-endian. In order to allow the receiving system to detect the byte order in the string being received, every Unicode entity should begin with the character, known as the Byte-Order Mark (BOM), which serves to indicate both that it is a Unicode file, and which of the formats it is in. With BOM the system can always distinguish if it is UTF-16BE, UTF-16LE, UTF-32BE or UTF-32LE. As for UTF-8, it always has the same byte order, and if initial BOM is used, it is *only* an indication that the text file is in UTF-8.

Multiscript MARC-records for a long time have been a sot of dream for library community, and the desire to use entirely all the opportunities provided with the appearance of Unicode is quite understandable. Nowadays more and more vendors declare that their systems provide full Unicode support. But what are perspectives of such full-scale intervention of Unicode?

As we already told, UTF-8 is some kind of bridge from ASCII to Unicode. It allows us to indicate in standard record, which character set is used, and the system can recognize this indication correctly and treat data in the record correspondingly. We can give such an instruction with usual UNIMARC mechanism – indicating code 50 in the field 100$a, pos. 26-27, or, as it is done in MARC21, with a code in the leader (pos.9).

The problem begins with transition to UTF-16 and UTF-32. But before dashing off to search ways of solving the problem, it is of some use to think – what is the point?

In case of UTF-16 and UTF-32 indicating characters set in the 100 field would not have the effect, since to read the field and the record as a whole the system must be aware which environment it works in, is it UTF-8, UTF-16 or UTF-32. Then, if the system knows it beforehand, what is the point of indicating character set anywhere in the record?

Let us look at some possible solutions.

(a) the solution used in MARC21, i.e. indication of using Unicode in the leader can not be considered as an absolute one – yes, information on character set used is now contained in the leader rather than in the body of the record, but to read and interpret the leader, the system still needs to know the character set – well, we come back to where we have been.

(b) making changes to ISO 2709, which nowadays is still the most heavily used container for MARC-records.

ISO 2709 was developed when any character could be represented with one, and only one, byte. Now it was recognized necessary to revise the standard to incorporate specifications for use it for records with Unicode. According to the draft of ISO 2709 revision, the term “octet” is defined; record label is fixed in length to 24 octets, each representing one character, irrespective how many bytes are used for a character in the record; finally, it is determined, that basic character encoding is ISO 646 or ISO 10646 with UTF-8 encoding, i.e. the record label and directory, indicators, subfield identifiers, field terminators, and record terminators use one octet per character encoding. Thus, new ISO is supposed to clarify the use of Unicode with UTF-8 encoding. It does not say anything about UTF-16 or UTF-32, and it seems absolutely correct, since ISO 2709 describes structure of the record, and encoding is not the question of syntax. But now we have the question: can we use two different UTF-s in the same record, i.e. UTF-8 for main structural elements, as prescribed by ISO 2709, and UTF-16 (or UTF-32) for

data fields? It sounds a bit strange for the author; anyway, this point should be stated explicitly in the format in order to avoid misunderstanding.

But, ISO 2709 is not the only container for library data transportation, as it used to be earlier. In particular, XML can be used as such container, and number of schemas are already developed (MARCXML, UNIMARC XML Slim). It is evident, that the above solution would not work for XML, and in this case we need another solution.

Although the default XML Character Set Encodings are UTF-8 and UTF-16, specific encodings for XML documents can be defined in the initial XML declaration for the whole document or entity (which can be regarded as a separately stored part of the whole document). In addition to BOM, indicating exactly which encoding is used in the record, this declaration seems to be sufficient for the receiving system to interpret the information.

One could say – OK, if there is no problem with XML, let's forget ISO 2709 and use XML. But we should remember, that most users of MARC-records still use ISO 2709 rather than XML, and we may not leave them overboard.

Now it' time to stop for a moment to decide – should we do with a set of solutions for any particular type of container, or search for a solution which would be some general both for ISO 2709, XML, and, possibly, any other syntax.

We are convinced that the solution should be general, though at the moment we can not propose any ready non-container-specific instrument, and we are not aware if anybody can do it right now. But is there really such urgency in any concrete solution immediately? In such situation we can not guarantee that the best decision is made. In practice so far we have not come across the situation when it is necessary to use character codes, which could not be represented with UTF-8, or the encoding form does not work for any other reason. We believe that this reality gives us some time not to make decision in a too hasty manner.