



Date : 21/07/2006

How localization challenges international portals: character sets and international access

Pierre Clavel
Swiss National Library
Bern, Switzerland

Meeting:	77 UNIMARC
Simultaneous Interpretation:	Yes

1. *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>

Abstract:

Access to electronic resources available in another country may be technically difficult. This paper will show which obstacles hinder access on the level of character sets. Most software and parts of hardware such as keyboards are localized, i.e. distributed in versions adapted to a language and its usage in a defined area. Users are now accustomed to this and consider it standard. Despite the fact that Unicode greatly facilitates interoperability, international portals face challenges in this respect because of significant differences between languages and/or countries in handling data, which can still hamper search, display and collation of data.

This study has been carried out within the project TEL-ME-MOR with the support of the European Commission.

1. Introduction

Several factors have helped the increase in the number of portals, computerized platforms accessing several databases simultaneously. In addition, the geographical and linguistic spread of partners has grown. Beside better performance of networks and computers, one factor in this growth is certainly the decrease in the number of character sets used, requiring fewer conversions to enable interoperability. It remains a minor, and sometimes unrecognized, factor however, the most important one being the greater availability of online documents, facilitating access across the world. One could wonder then, whether language differences would not be the barrier cancelling the opportunities brought by the progress mentioned above. However, although the language barrier remains, its impact is reduced by the fact that most libraries have some material in other languages than the local one and that many online documents, such as images and sounds, are not related to a language. Nevertheless, these need to be described and indexed, so the same issues of character sets arise. Although this article focuses on library portals, in particular The European Library (www.theeuropeanlibrary.org), the issues discussed here also concern other kinds of portals and web search engines.

The aim of The European Library is to provide access to several resources through a combination of a centralized index and distributed searches. These resources are digital documents of various formats and catalogue records of currently 16 European national libraries. The ultimate goal is to incorporate

all national libraries of the 46 member countries of the Council of Europe (English home page under www.coe.int/DefaultEN.asp, available in 32 other languages). This will involve no less than five different scripts: Armenian, Cyrillic, Georgian, Greek and Latin. This paper however discusses the character set issues within the Latin script only.

The aim of the project TEL-ME-MOR (The European Library: Modular Extensions for Mediating Online Resources, www.telmemor.net) is to help the national libraries of the New Member States of the European Union to become full members of The European Library. One was already a member (Slovenia), three have already joined at the time of writing this paper (Estonia, Latvia, Slovakia) and the six remaining should join by the beginning of 2007. Six other partners, including the Swiss National Library, contribute by bringing their technical or administrative expertise on specific tasks.

2. Basics about character sets and their issues

Computers store and process information in a succession of *bits*, memory elements able to have only two possible values, 0 and 1. To extend the number of possible values, bits are combined in groups where the number of possible values is equal to 2 to the power of the number of bits, since each added bit doubles the number of possible values. Examples:

2 bits: 00, 01, 10, 11 = 4 values = 2^2

3 bits: 000, 001, 010, 011, 100, 101, 110, 111 = 8 values = 2^3

etc.

One such group of bits can then represent a character. The succession of possible values paired to the characters assigned to them makes a character set.

Each constructor of early computer models defined their own character set, but the need for standardisation was felt in the late 1950s to allow data to be exchanged.

The first standard character set, ASCII (American Standard Code for Information Interchange), had 7 bits, hence 128 available characters and allowed texts in English to be correctly stored. At this stage, languages using another script or the Latin script with diacritics had simply no way yet to be processed correctly. ASCII was internationalized in 1967 as ISO 646, with the definition of 10 character positions reserved for 'national variants'.

Data transmission progressively became reliable enough to use the 8th bit for characters instead of a 'parity check', which opened the road to the use of computers in any language with an alphabetic script.

A large number of new 8-bit standards were developed to enrich the available character palette and meet the needs of various languages and user groups. About 180 of them incorporate ASCII and provide compatibility for the first half of the character set but not for the second, as the example below will show.

A person working on a PC configured for Western Europe writes in a simple text file the sentence
'Le théâtre sera réparé après le mois d'août, là où même le plâtre paraît neuf.'

If this file is then transmitted and opened on a Macintosh or a very old MS-DOS PC also configured for Western Europe, or on a PC configured for Eastern Europe, the sentence will read, respectively:

'Le thÈ,tre sera rÈparÈ aprÈs le mois d'ao't, l¸ o~ m¸me le pl,tre paraÓt neuf.'

'Le thŒtre sera rŒparŒ aprŒs le mois d'ao'vt, l¸ o· mŒme le pl'tre para¸t neuf.'

'Le théâtre sera réparé apr¸s le mois d'ao¸t, l¸ o¸ m¸me le pl¸tre para¸t neuf.'

Someone knowing the language may have a good chance to guess the right letters. However, not only can the display be considered unsatisfactory, but searching in a catalogue using the wrong character set will not return accurate results.

3. First step towards a solution

Although software now allows loading and using several 8-bit character sets at the same time, it requires tagging any bit of text with the character set being used, unless it is the default one. But the default character set may be not the same between two communicating computers and this harms interoperability since announcing its default character set is not part of all communication protocols. This is why industry and standardization bodies looked for a solution leading to a single comprehensive character set. After separate (and even conflictual) early work, Unicode (www.unicode.org, the industry initiative) and ISO 10646 (the standard bodies' initiative) agreed to standards where at least the code point for any character is common between them. They only differ in additional information related to a character, Unicode prescribing more properties than ISO.

Unicode is a 16 to 32-bit character set aiming to cover all needed characters in a single character set. 16 bits allow for 65,536 characters, quite enough to handle all alphabets and syllabaries. This number has been extended to 1,114,112 by defining a virtual 32-bit section for ideographs. Like 8-bit character sets incorporating ASCII, it incorporates Latin-1 (ISO 8859-1), providing compatibility for Western Europe without other conversion than inserting empty bytes.

Unicode and ISO 10646 describe characters, not glyphs. The following example will show this better than an explanation:

Character	Code	Name
В	U+0412	Cyrillic capital letter ve
Β	U+0392	Greek capital letter beta
B	U+0042	Latin capital letter b

In Cyrillic, Greek and Latin, the capital letters ve, beta and respectively b have exactly the same appearance. They use the same *glyph*. They remain however different *characters*, with different codes, since they are used within different scripts, which shows when one sees their lower case counterparts: в, β and b respectively.

In 8-bit character sets, the link between a character's code and the numerical value actually processed by, and stored in, computers is straightforward: it is always the same in (almost) all cases and applications because 8-bit is still a basic unit recognized by microprocessors. Unicode on the other hand makes a distinction between a character's code as an abstraction and how it is translated into bits by an *encoding scheme*. The choice of an encoding scheme depends on what is to be done with the characters, e.g. storage or processing, and the constraints specific to these actions. There are 3 main schemes, called UTF-32, UTF-16 and UTF-8.

In UTF-32, each character is encoded on 32 bits, or 4 bytes. It is really relevant only where mostly ideographs are concerned. In UTF-16, each character is encoded on 16 bits, or 2 bytes, except characters whose code is higher than 65,535 (hexadecimal FFFF), where 4 bytes are needed. This encoding scheme is preferred in applications where the predictability of the position of a particular character within a string should not depend on what precedes it. In UTF-8, each character is encoded on 1 to 4 bytes according to a specific algorithm. ASCII characters need 1 byte, all other European characters (including non-Latin scripts) and Middle-Eastern scripts need 2 bytes, and 3-4 bytes are needed for Asian scripts and some special characters such as mathematical symbols. This encoding scheme is preferred where space or transmission speed is an issue and is therefore widespread on the web.

The Unicode standard also defines some technical properties of characters. Each character has:

- A standardized *name*.
- A *general category*, such as letter, number, punctuation etc.
- A *canonical combining class value*: in many languages, diacritics or similar signs may be added to letters to mark intonation or change its sound. Unicode contains diacritics individually and this parameter defines where they are placed around the letters.
- A *bidirectional class value*, defining the directional behaviour of the character towards its neighbours in a text, left-to-right, right-to-left as well as options when the two directions are mixed.
- References to code points of other characters, such as constituents of a letter with diacritic, lowercase of an uppercase and vice-versa, as well as decimal value of numerical characters.

Unicode today is supported on all platforms and in many applications, including database systems and printers. Since Unicode incorporated characters from all the other existing character sets (bar some East Asian ones) a lossless conversion is always possible from these character sets to Unicode, the reverse being of course not true.

4. Are character sets still an issue?

Yes, even in an all-Unicode world, there are still issues to consider at the three main steps of user interaction with a portal or search engine:

1. Dealing with the different ways special characters are indexed from database to database.
2. Inputting search terms and search resources with characters that are not part of the default character subset installed on the user's computer.

not. Systems are indeed tolerant to 'excessive' detail and accept requests with diacritics even when they are ignored in indexing.

2. If some characters of the search terms encoded in Unicode have no equivalent in some of the non-Unicode character sets of the targets, a lossless conversion is impossible. The loss is however likely to be unimportant: the missing character will be a letter not taken into account in the target's library systems anyway, which would remove its diacritic or convert it into its base letter for searching. In this case, the portal could do it upstream, before sending requests in character sets where the original character is absent.
3. A loss induced by the conversion may actually help by extending the search. For instance, if a user searches 'Łodz', this term will be sent as is in requests to targets having Unicode or the East European character set. 'Ł' being absent from the other 8-bit character sets, libraries using one of those have been forced to reduce this character to 'L' and encode 'Lodz'. A search term reduced that way will still be successful in accessing the relevant records.

Second, users are rarely conscious of locales variations outside the languages they speak. To take the previous example again, if a user not speaking Polish but looking for an image of Łodz enters 'Lodz' as search term, Polish databases will only return records where this city was written like that in the first place (e.g. because they describe documents of non-Polish origin) or linked to an authority record where this form is added as a see-reference. The user will miss many records, since these databases are very likely to contain more material, at least in the case of The European Library. The opposite problem may indeed occur in community databases such as flickr, www.flickr.com, where 'Łodz' gives 50 times fewer hits than 'Lodz'. It is a database of digital photographs where non-Polish tourists obviously contributed more than nationals.

Third, even if the user knows that it would be better to type 'Łodz' instead of 'Lodz', there is an issue on input: keyboards are nationally, regionally and/or linguistically localized to allow input of special characters of the national language(s), plus a few others almost arbitrarily chosen. Despite the fact that Unicode is now the character set lying at the heart of today's operating systems, keyboard drivers are still restricted to keys and key combinations needed for the national language(s) for which their keyboards are designed. Input by other means, when possible, is clumsy.

Result display

As seen before, target servers may return records in different character sets. Mixing character sets in a single web page is no easy task, since the character set is declared in the <head> statement and is not supposed to change along the page's body. Although there are various ways to nevertheless do it, there is actually no absolute need to mix character sets on the result screen: everything not returned in Unicode should be converted to it.

Returned records may be displayed as they came or reordered according to various criteria. As soon as alphabetical order is involved somewhere, as it is in the result sets of some systems, the diversity of locales brings a risk that users will miss resources because they expect them elsewhere within the set. This is especially the case because locales in alphabetical order do not differ so widely that the application of an unknown locale would make the set seem unsorted.

One can compare for instance the alphabetical orders of English, Estonian and Slovak locales:

English: a b c d e f g h i j k l m n o p q r s t u v w x y z
Estonian: a b c d e f g h i j k l m n o p q r s **š ž ž t u v w õ ä ö ü x y**
Slovak: a b c **č** d e f g h **ch** i j k l m n o p q r **ř s š ť** t u v w x y z **ž**

The sequences for Estonian and Slovak show four cases of local adaptations to the standard alphabetical order:

- Shifting some letters: t, u, v, w, x and y come after z in Estonian.
- Putting a letter with diacritic after its base letter: č and ř in Slovak, š and ž in both locales.
- Putting a letter with diacritic at the end of the alphabet: õ, ä, ö and ü in Estonian.
- Treating a pair of letters as if they were a single separate character: ch in Slovak.

Depending on the user-server locales combination, up to 15% of the records would look misplaced to the user, not necessarily enough to allow her to immediately spot this fact but quite enough to bring difficulties in finding a resource just because its name or title contains a special character. The effects of these differences should not be underestimated: alphabetical order is something one learns very early, together with the letters themselves.

