ISO/IEC JTC 1/SC 2/WG 3 N 472

Date : 1998-09-28

ISO/IEC JTC 1/SC 2/WG 3
7-bit and 8-bit codes and their extension
SECRETARIAT : ELOT

DOC TYPE :	Officer's Contribution
TITLE :	Towards a Model of Character Encoding
SOURCE :	Ken Whistler
PROJECT:	----
STATUS :	Expert Contribution
ACTION ID :	For the consideration of UTC and L2
DUE DATE :	---
DISTRIBUTION :	P, O and L Members of ISO/IEC JTC 1/SC 2 WG Conveners, Secretariats WG 3 Members ISO/IEC JTC 1 Secretariat ISO/IEC ITTF UTC and L2 Members
MEDIUM :	P, Def
NO OF PAGES :	9

Contact 1: Secretariat ISO/IEC JTC 1/SC 2/WG 3 ELOT Mrs K.Velli (acting)
Acharnon 313, 111 45 Kato Patissia, ATHENS – GREECE
Tel: +30 1 22 80 001 Fax : +30 1 22 86 219 E-mail : kkb@elot.gr

Contact 2 : Convenor ISO/IEC JTC 1/SC 2/WG 3 Mr E.Melagrakis
Acharnon 313, 111 45 Kato Patissia, ATHENS – GREECE
Tel: +30 1 22 80 001 Fax : +30 1 22 86 219 E-mail: eem@elot.gr

Towards a Model of Character Encoding

Introduction

The recent discussions about the attempt to register "UTF-16" as an IANA charset for the Internet, as well as editorial problems resulting from the attempt to treat UFT-16 and UTD-8 with equal status in the revision of the text for the Unicode Standard, Version 3.0, have highlighted a number of inconsistencies and misunderstandings about just what Unicode *is* in the context of character encodings of all types.

This contribution continues the rectification of names regarding various concepts, which apply to Unicode as a character encoding. In this I draw upon carious other formulations, which have been coming out of the editorial committee in the last, couple weeks - particularly those by Joe Becker. What I write here is a slight formalization of the email note that I sent around to the *unicore* list on July 27. 1998. It should not be taken as a final statement. I am only hoping that this will help frame and enlighten the debate about the immediate problem of what to do about the "UTF-16" charset registration.

[In time, this should turn into a Unicode Technical Report on Character Encoding.]

Definitions and Acronyms

The main body of this contribution consists of an attempt at detailed definition of several terms related to character encoding. This section merely clarifies acronyms and a few other subsidiary terms used in various contexts.

CCS	Coded Character Set
CEN	European Committee for Standardization
CES	Character Encoding Scheme
CDRA	Character Data Representation Architecture from IBM
IAB	Internet Architecture Board
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Taskforce
RFC	Request For Comments (term used for an Internet standard)
RCSU	Reuters Compression Scheme for Unicode
SCSU	Standard Compression Scheme for Unicode
TES	Transfer Encoding Syntax
UCS	Universal Character Set; Universal Multiple-Octet Coded Character Set - the repertoire and encoding represented by ISO/IEC 10646-1:1993 and its amendments.
UDC	User-defined Character

References

RFC 2144 (etc.-- need to be filled out).

The Character Encoding Model

The character encoding model proposed here draws on the character architecture promoted by the IAB for use on the Internet. It also draws in part on the CRDA used by IBM for organizing and cataloguing its own vendor-specific array of character encodings. The focus here is on clarifying how these models should be extended and clarified to over the needs of the Unicode Standard, and by extension, the UCS.

The IAB model makes three distinctions with respect to level: Coded Character Set (CCS), Character Encoding Scheme (CES), and Transfer Encoding Syntax (TES). However, to adequately cover the distractions required for the character encoding model, I claim that *five* levels need to be defined. One of these, the *repertoire*, is implicit in the IAB model. However, I distinguish an additional level between the CCS and CES.

The five levels can be summarized as:

- repertoire (the set of abstract characters to encode)
- coded character set (mapped to integers)
- character encoding form (specified to particular datatype widths)
- character encoding scheme (serialized to byte sequences)
- transfer encoding syntax (hacked or compressed for data transmission)

1. Repertoire

A *repertoire* is defined as the set of abstract characters to be encoded. A repertoire is an unordered set.

Repertoires come in two types: fixed and open.

For most character encodings, the repertoire is fixed (and often small). Once the repertoire is decided upon, it is never changed. Addition of a new abstract character to a given repertoire is conceived of as creating a new repertoire, which then will be given its own catalogue number, constituting a new object.

In the context of the Unicode standard, on the other hand, the repertoire is inherently open. Because Unicode is intended to be the universal encoding, any abstract character that ever could be encoded is potentially a member of the actual set to be encoded, whether we currently know of that character or not.

Microsoft, for its Windows character sets, also makes use of a limited notion of open repertoires. The repertoires for particular character sets are periodically extended by adding a handful of characters to an existing repertoire. The recently occurred when the EURO SIGN was added to the repertoire for a number of Windows character sets, for example.

The Unicode standard versions its repertoire by publication of major and minor editions of the standard: 1,0. 1.1. 2.0, 2.1, 3,0, ... The repertoire for each version is defined by the enumeration of abstract characters included in that version. There was a major glitch between versions 1.0 and 1.1, occasioned by the merger with ISO/ IEC 10646, but starting with version 1.1 and continuing forward indefinitely into future versions, no character once included is ever removed from the repertoire.

ISO/IEC 10646 has a different mechanism of extending its repertoire. The 10646 repertoire is extended by a formal amendment process. Ad each individual amendment is balloted, approved, and published, that may constitute an extension to the UCS repertoire, depending on the content of the amendment. The tricky part about keeping the repertoires of the Unicode Standard and of ISO/IEC 10646 in alignment is coordinating the publication of major version of the Unicode Standard with publication of a well-defined list of amendments for 10646 (or a major revision and republication of 10646).

Repertoires are the things that in the IBM CDRA architecture get CS ("character set") values.

Examples:

- the repertoire of JIS X 0208 (fixed)
- the repertoire of Latin-1 (fixed)
- the POSIX portable character repertoire (fixed)
- the IBM host Japanese repertoire (CS 01001) (fixed)
- the Windows Western European repertoire (open)
- the UCS repertoire (open)

Subsets

Unlike most character repertoires, the UCS is deliberately intended to be *universal* in coverage. What this implies in practice, given the complexity of many writing systems, is that nearly all implementations will implement some subset of the total repertoire, rather than all the characters.

Formal subset mechanisms are occasionally seen in implementations of some Asian character sets, where for example, the distinction between "Level 1 JIS" and "Level 2 JIS" support refers to particular parts of the repertoire of the JIS X 0208 kanji characters to be included in the implementation.

However, subsetting is a major formal aspect of ISO/IEC 10646-1. The standard includes a set of internal catalogue numbers for named subsets, and further makes a distinction between subsets that are "fixed collections" and open collections that are defined by a range of code positions. (see Technical Corrigendum No. 2 to ISO/IEC 10646-1:1993(E) for details). The collections that are defined by a range of code positions are themselves open subsets of the repertoire, since they could be extended at any time by an addition to the repertoire which happens to get encoded in a code position between the range limits which define such a collection.

The current TC304 effort to define multilingual European subsets (MES-1. MES-2, and MES-3) of ISO/IEC 10646-1 is a CEN effort to define three more subsets (each a fixed collection) that will, no doubt, at some point be added as named subsets in 10646.

For the Unicode Standard, subsets are nowhere formally defined. It is considered up to the implementation to define and support the subset of the universal that it wishes to interpret.

2. Coded Character Set (CCS)

A *coded character set* is defined to be a mapping from a set abstract characters to the set nonnegative integers.

Note: Mathematically, this mapping may not be 1:1. For example, katakana ka is a single abstract character, but it has two representations in both Unicode and in SJIS. Also, the range of integers used for the mapping need not be contiguous.

Definition: An abstract character is said to be *in a coded character set* if the coded character set maps from it to an integer. The integer is said to be the *value* (or *coded value*) of the abstract character.

Effectively, coded character sets are the basic object that both ISO and vendor character encoding committees produce. They relate a defined repertoire to nonnegative integers which then can be used unambiguously to refer to particular abstract characters from the repertoire.

The Unicode 2.0 concept of the Unicode scalar value (cf. D28, page 3-7 of the Unicode Standard, Version 2.0) is explicitly this nonnegative integer used for mapping of the UCS.

AKA: Character Encoding, Coded Character Repertoire, Character Set Definition, Code Page.

Coded character sets are the things that in the IBM CDRA architecture get CP ("code page") values. (Note that this use of the term *code page* is quite precise and limited, and should not be, but generally is, confused with the generic use of *code page* to refer to character encoding schemes. See below.)

Examples:

- JIS X 0208 (assigns pairs of integers, know as kuten points)
- ISO/IEC 8859-1
- ISO/IEC 8859-2 (different repertoire than 8859-1)
- Code Page 037 (same repertoire as 8859-1; different integers)
- Code Page 500 (same repertoire as 8859-1 and Code Page 037; different integers)
- The Unicode Standard, Version 2.0
- ISO/IEC 10646:1993 + amendments 1-7 (exactly the same repertoire and mapping as Unicode 2.0)

Character Naming

In the JTC1/SC2 context, coded character sets also require the assignment of unique names to each abstract character in the repertoire to be encoded. This practice is not generally followed in vendor coded character sets or the encodings produced by standards committees outside SC2, where the names provided for the characters, if any, are often variable and annotative, rather than normative parts of the character encoding.

The main rationale for the SC2 practice of character naming was to provide a mechanism to unambiguously identify abstract characters across different repertoires given different mapping to integers in different coded character sets. Thus LATIN SMALL LETTER A WITH GRAVE would be seen as the *same* abstract character, even when it occurred in different repertoires and assigned different integers, depending on the particular coded character set.

This functionality of ensuring character identity across different coded character sets (or "code pages") is handled in the IBM CDRA model instead by assigning a catalogue number, known as a GCGID (graphic character glyphic identifier), to every abstract character used in any of the repertoires accounted for by the CDRA. Abstract characters that have the same GCGID in two different coded character sets are by definition the same characters. Other vendors have made use of similar internal identifier systems for abstract characters.

The advent of the UCS has largely rendered such schemes obsolete. The identity of abstract characters in all other coded character sets is increasingly being defined by reference to the UCS itself. Part of the pressure to include every "character" from every existing coded character set into Unicode results from the desire by many to get rid of subsidiary mechanisms for tracking bits and pieces, odds and ends that aren’t part of Unicode, and instead just make use of Unicode as the universal catalogue of characters.

Code Spaces

The range of nonnegative integers used for the mapping of abstract characters defined a related concept of *code space.* Traditional boundaries for types of code spaces are closely tied to the encoding forms (see below), since the mappings of abstract characters to nonnegative integers are not done arbitrarily, but with particular encoding forms in mind. Example of significant code spaces are 0../F, 0…FF, 0…FFFF, 0…10FFFF, 0…7FFFFFFF, 0…FFFFFFFF.

Code spaces can also have fairly elaborated structures, depending on whether the range of integers is conceived of as continuous, or whether particular range of values are disallowed. Most complications again result from considerations of encoding form; when an encoding form specifies that the integers used in encoding are to be realized as sequences of octets, there are often constraints placed on the particular values that those octets may have - mostly to avoid control code values. Expressed back in terms of code space, this results in multiple ranges of integers that are disallowed for mapping a character repertoire. (see Ken Lunde´s publications on Asian information processing to see two-dimensional diagrams of typical code space for Asian coded character sets.)

3. Character Encoding Form

A *character encoding form* is a datatype -specific width specification of each of the integers used in a CCS.

Another way of putting this is that encoding form enables a character representation as actual data in a computer.

Definitions (tentative, added by Mark Davis):

An *integral datatype* is an integer occupying a certain binary width in a computer architecture, such as a byte.

A *character encoding form* is defined to be a mapping from abstract character to sequences of the same integral datatype. The sequences do not necessarily have the same length.

A character encoding form whose sequences are all of the same length is known as *fixed width.*

A character encoding form whose sequences are not all of the same length is known as *variable width.*

An abstract character is said to be in *a character encoding form* if the character encoding form maps it to a datatype sequence. That sequence is said to be the *datatype-specified* value of the abstract character, and also is known as an *encoded character.*

A character encoding form *for a coded character set* is defined to be a character encoding form for all of the abstract characters in the coded character set, and whose datatype-specified values can be algorithmically generated from the values of the coded character set.

Note: In many cases, there is only one character encoding form for a given coded character set. In some such cases only the character encoding form has been specified. This leaves the coded character set implicitly defined, based on an implicit relation between the datatype sequence and integers.

[It is currently unclear to me whether the SC2 concept of CC-data-element fits in at level. The CC-data-element is defined as: "(Coded-Character-Data-Element): An element of interchanged information that is specified to consist of a sequence of coded representations of characters, in accordance with one more identified standards for coded character sets."]

The encoding form may result in either fixed-width or variable-width collections of numbers associated with abstract characters. The encoding form may involve an arbitrary functional mapping (reversible and algorithmic) of the integers of the CCS to a new set of integers. There is, in general, no constraint that the resulting set of integers in the encoding form maintain a one-to-one mapping between abstract character and integer; an abstract character may be mapped to a sequence of integers of defined data width.

Encoding forms come in various types. Some of them are exclusive t the UCS, whereas other represent general patterns that are repeated over and over for hundreds of coded character sets. What follows here is a typology of some of the more important encoding forms.

Fixed width:

- 7-bit (each encoded character is represented in a 7-bit quantity) For example, as in ISO 646.
- 8-bit G0/G1 (each encoded character is represented in a 8-bit quantity, with constraints on use of CD and C1 spaces)
- 8-bit (each encoded character is represented in an 8-bit quantity, with no constraints on use C1 space)

- 8-bit EBCDIC (each encoded character is represented in an 8 bit quantity, with the EBCDIC conventions rather than ASCII conventions)
- 16-bit (UCS-2) (each encoded character is represented in a 16-bit quantity)
- 32-bit (UCS-4) (each encoded character is represented in a 32-bit quantity)
- 16-bit DBCS process code (as for UNIX widechar implementations of Asian CCS´s)
- 32-bit DBCS process code (as for UNIX widechar implementations of Asian CCS´s)
- DBCS Host (two 8 bit quantities, following IBM host conventions)

Variable width:

- UTF-8 8used only with Unicode/10646: mix of one to six 8 bit quantities; in practice only one to four, because of the actual range of integers used for encoding the UCS.)
- UTF-16 (used only with Unicode/10646: mix of one to two 16 bit quantities)

Note that it is at the level of an encoding form that most API's must be specified, since it is here that characters are actually bound to datatypes. This is the fundamental difference between UTF-8 and UTF-16, which cannot coexist amicably for the same textual API (at least without playing type-switching tricks in the API); otherwise they represent exactly the same coded character set. However, the byte order of the platform is generally not relevant at the API level; the same API can be compiled on platforms with any byte polarity, and will simply expect character data (as for any integral-based data) to be passed to the API in the byte polarity for that platform.

The encoding form also defines one of the fundamental relations that internationalized software cares about: how many data elements are there for each character. This used to be expressed in terms of how many bytes each character was represented in, but the introduction of UCS-2, UCS-4, and UFT-16, with wider datatypes for Unicode and 10646 means we must now generalize this to both a specification of the width of the fundamental datatype used for representing character data and the width map which specifies for each character in the coded character set how many of those data elements are used to represent that character.
UTF-8 provides a good example:

> The fundamental datatype used for representing character data is 8 bits wide (a byte).
>
> The width map for UTF-8 is:

0x00…0x7F	→ 1 byte
0x80…0x3FF	→ 2 bytes
0x400…0xD7FF,0xE000…0xFFFF	→ 3 bytes
0X10000…0x10FFFF	→ 4 bytes

Example of encoding schemes as applied to particular coded character sets:

- JIS X 0208 is generally transformed from the kuten notation to a 16-bit "JIS code" encoding form, e.g. nichi, 38 92 (kuten) → 0x467C JIS code.
- ISO 8859-1 has the 8-bit G0/G1 encoding form
- CP 037 and CP 500 both have the 8-bit EBCDIC encoding form
- US ASCII and ISO 646 have the 7-bit encoding form
- Windows CP 1252 has the 8-bit encoding form
- Unicode 2.0 has either the UTF-16 (default) or UTF-8 encoding form
- Unicode 1.1 has either the UCS-2 (default) or UTF-8 encoding form

- ISO/IEC 10646, depending on the declared implementation levels, may have either UCS-2, UCS-4, UTF-16, or UTF-8.

4. Character Encoding Scheme (CES)

A character encoding scheme is a mapping of a sequence of abstract characters form one or more CCS´s, each using a defined encoding form, into serialized byte sequences.

[Note to Mark: I want to keep the more baroque formulation, rather than collapse this into "mapping of a sequence of abstract characters form one or more character encoding forms", because the CES typically has to be specified in terms of identifiers which are associated with the CCS´s and not with the encoding forms. The encoding forms are generic transforms on CCS´s, rather than being unique to each character set. Also, while it would be possible to define CES´s for **any** integral width, in practice the charset definitions really depend on serialization into 8-bit byte sequences.]

Character encoding schemes are the things that in the IAB architecture get IANA charset identifiers. The important thing, from the IANA charset point of view, is that a sequence of encoded characters must be unambiguously mapped onto a sequence of bytes by the charset. The charset (=CES) must be specified in all instances, as in Internet protocols, where textual content is treated as a ordered sequence of bytes, and where the textual content must be reconstructible from that sequence of bytes.

Character encoding schemes are the things that in the IBM CDRA architecture get CCSID (coded character set identifier) values.

AKA: charset, Character Set, Code Page (broadly construed).

Character encoding schemes are also relevant to the issue of cross-platform persistant data involving datatypes wider than a byte, where byte swapping may be required to put data into the byte polarity canonical for a particular platform.

Most fixed-width byte-oriented encoding forms have a trivial mapping into a CES: each 7-bit or 8-bit quantity maps to a byte of the same value.

Most mixed-width byte-oriented encoding forms also simply serialize the sequence of CC-data-elements to bytes, UTF-8, since it is already a byte-oriented encoding form, follows this pattern. UTF-16, on the other hand, which involves 16-bit quantities must specify byte-order for the byte serialization. Thus the difference between UTF-16BE, where the two bytes of the 16-bit quantity are serialized in big-endian order and UTF-16LE, where they are serialized in little-endian order.

Character encoding schemes may also partake of some of the features of transfer encoding syntaxes proper (see below). Thus both UTF-8 and UTF-7 are designed to be byte-oriented in their datatype and to avoid control code values for transmission and other protocols. UTF-7 goes further in incorporating some of the features of Base64 to avoid a number of byte values in the ASCII range. On the other hand, the Unicode-specific compression schemes that convert directly from Unicode data in a specified encoding form to a sequence of bytes that compresses the textual data, can also be conceived of as a character encoding scheme.

The important differences between a CES and an Encoding Form are:

a. The CES must take into account the byte-order serialization of all datatypes wider than a byte that are used in the Encoding Form.

b. The CES may consist of two or more CCS´s, and may include byte values (e.g single shifts SI/SO, or escape sequences) that are not part of the CCS per se, but which are defined by the character encoding architecture and which may require an external registry of particular values (as for the 2022 escape sequence).

c. The CES may also make distinctions such as the number of UDC´s that are allowable. (This applies in particular to the IBM CDRA architecture, which may distinguish host CCSID´s based on whether the set of UDC´s is conformably convertible to the corresponding PC code page or not).

Examples:

- Unicode 2.0 has four character encoding schemes: UTF-8, UTF-16BE, UTF-16LE, x-UNICODE-2-0-UTF-7 (Note that UTF-7 is now generally denigrated, and "x" marks a private MIME charset identifier, not an IANA registered identifier).
- Unicode 1.1 had four character encoding schemes: UTF-8, UCS-2BE. UCS-2LE, UNICODE-1-1-UTF-7.
- ISO 2022-based charsets (ISO-2022-JP, ISO-2022-KR, etc.), which use embedded escape sequences
- DBCS Shift (mix of one single-byte CCS, e.g. JIS X 0201 and a DBCS CCS, e.g. based in JIS X0208, with a numeric shift of the integer values) For example, Code Page 932 on Windows.
- EUC (similar to the DBCS Shift encodings, with the application of different numeric shift rules, and the introduction of single-shift bytes: 0x8E and 0x8F, that may introduce 3-byte and 4-byte sequences) For example, EUC-JJP or EUC-CNS on UNIX.
- IBM host mixed code page Asian character sets, which formally mix two district CCS´s with the SI/SO switching conventions. For example, CCSID 5035 on IBM Japanese host machines.
- SCSU and RCSU should also be conceived of as character encoding schemes. They are appropriate for registration to get formal charset identifiers.

5. Transfer Encoding Syntax (TES)

A *transfer encoding syntax* is a reversible transform of coded **data** which may (or may not) include textual data represented in one or more CES´s.

Note: A more appropriate term for this might be *Transfer Encoding Form*, but *Transfer Encoding Syntax* already has widespread usage in the Internet community.

Typically TES´s are engineered either to:

a. Avoid particular byte values that would confuse one or more Internet or other transmission/storage protocols: base64, uuencode, BinHex, quoted-printable, etc.

or to:

b. Formally apply various compressions, deflations, squeezings, packings, and general degasifying to data to minimize the number of bits to be passed down a communication channel: pkzip, gzip, winzip, etc.

The Internet Content-Transfer-Encoding tags "7bit" and "8bit" are special cases. These are data width specifications relevant basically to mail protocols and which, I believe, predate true TES's like quoted-printable. Encountering a "7bit" tag doesn't imply any actual transform of data; it merely is an indication that the charset of the data can be represented in 7bits and will pass 7-bit channels - it is really an indication of the encoding form. In contrast, quoted-printable actually does a conversion of various characters (including some ASCII) to forms like "=2D", "=20", etc., and should be reversed on receipt to regenerate legible text in the designated character encoding scheme.