COM3110/4115/6115: Text Processing

Unicode – An Introduction

Rob Gaizauskas

Department of Computer Science
University of Sheffield

Overview

"The Unicode Standard is the universal character encoding standard for written characters and text. It defines a consistent way of encoding multilingual text that enables the exchange of text data internationally and creates the foundation for global software." (version 11.0.0, p. 1)

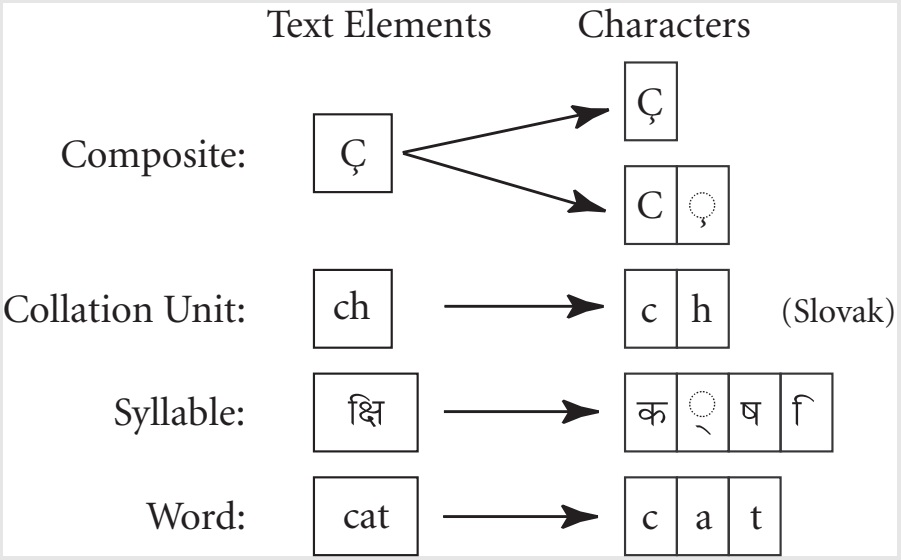
- Architectural Context
- Design Principles
- Encoding Model
- Allocation Areas
- Surrogate Pairs
- Character Encoding Forms and Character Encoding Schemes
- Worked Examples of Encoding Forms and Byte Serialization

Unicode: Architectural Context – Text Processes

- No character encoding scheme is an end in itself: it is a means to enable useful text processing on computers.
- Basic low-level text processes computers expected to support include:
  - rendering characters visible (including ligatures, contextual forms, and so on)
  - breaking lines while rendering (including hyphenation)
  - modifying appearance, such point size, kerning, underlining, slant and weight (light, demi, bold, etc.)
  - determing units such as "word" and "sentence"
  - interacting with users in processes such as selecting and highlighting text
  - accepting keyboard input and editing stored text through insertion and deletion
  - comparing text in operations such as in sorting or or determining the sort order of two strings
  - analysing text content for, e.g. spell checking, hyphenation, parsing morphology (finding word roots/stems/affixes)
  - treating text as bulk data for, e.g., compression, transmission

Unicode: Architectural Context – Text Elements, Code Values and Text Processes

- Unfortunately, for each text process, languages differ in what constitutes a text element.
  - ◇ e.g. In Spanish "ll" sorts between "l" and "m" (i.e. should be treated as a text element), but in rendering it is best treated as two "l" elements.
- Thus, the first challenge in designing an encoding scheme for a language is to agree on the set of **abstract characters** to be encoded – those characters that will be assigned a **code value**.
- For English this seems straightforward (but, e.g., should "A" and "a" get 2 code values or 1?); other languages are not so simple.



Unicode: Design Principles

- Underpinning Unicode are 10 design principles (version 10.0.0):

Principle	**Statement**
Universality	The Unicode Standard provides a single, universal repertoire
Efficiency	Unicode text is simple to parse and process
Characters, not glyphs	Unicode encodes Characters, not glyphs
Semantics	Characters have well-defined semantics
Plain text	Unicode characters represent plain text
Logical order	Default for memory representation is logical order
Unification	Unicode unifies duplicate characters within scripts across languages
Dynamic composition	Accented forms can be dynamically composed
Stability	Characters, once assigned, cannot be reassigned and key properties are immutable
Convertibility	Accurate convertibility is guaranteed between Unicode and other widely accepted standards

Unicode: Design Principles

- **Universality**
  - Unicode encodes a single, very large set of characters, encompassing all the characters needed for worldwide use.
  - intended to be universal in coverage, containing all characters for textual representation in all modern writing systems, in most historic writing systems, and for symbols used in plain text.
  - designed to meet the diverse needs of business, educational, liturgical and scientific users
  - out of scope:
    - writing systems for which insufficient information is available to enable reliable encoding of characters
    - writing systems that have not become standardized through use
    - writing systems that are nontextual in nature.

Unicode: Design Principles

- **Efficiency** – designed to allow efficient implementations
  - no escape characters or shift states – each character has same status
  - all encoding forms self-synchronising: limited backup required to find character when randomly accessing a string
    - in UTF-16, at most one byte back-up when dealing with surrogate pairs
    - in UTF-8, at most three bytes backup
  - characters of a script grouped together as far as possible

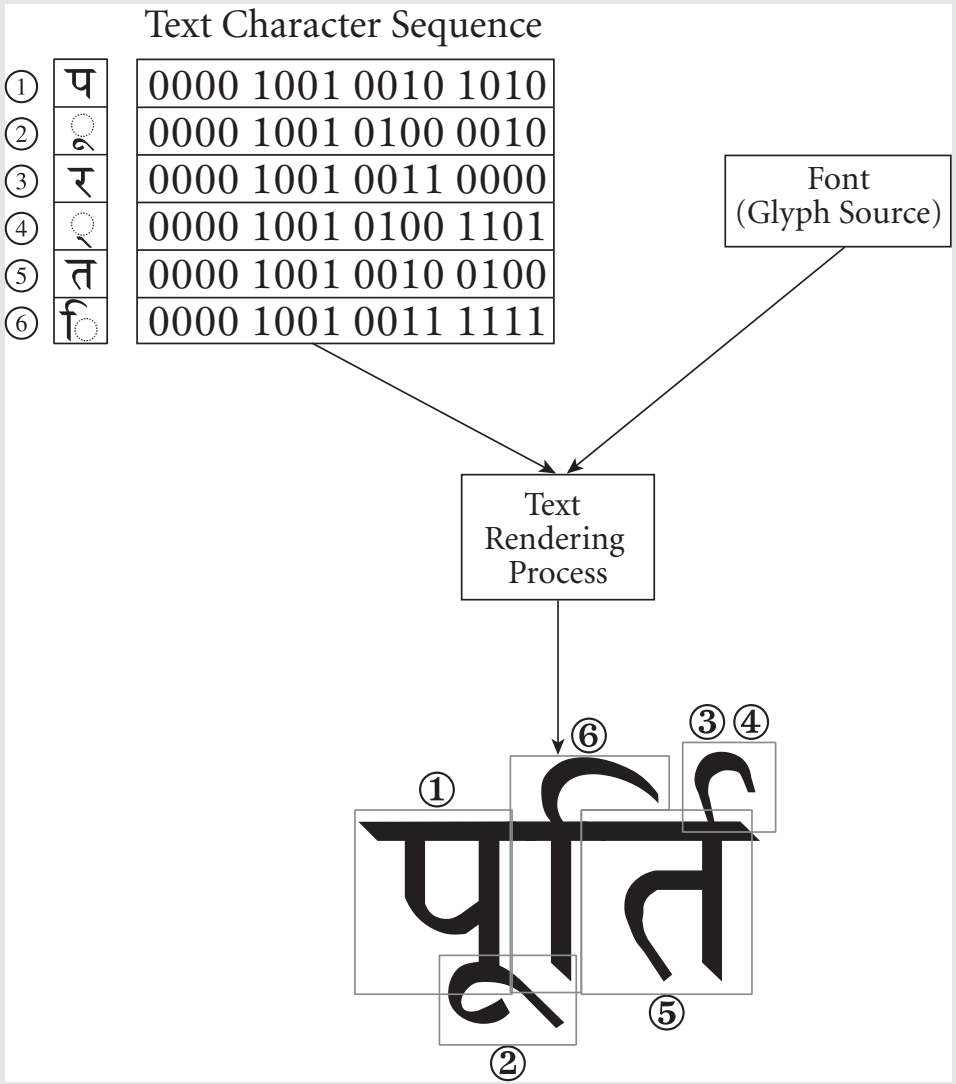
Unicode: Design Principles (cont)

- **Characters, not Glyphs**
  - Unicode distinguishes
    - characters: smallest components of written language with semantic value
    - glyphs: the shapes characters can have when displayed
  - Unicode standard deals *only* with character codes

Glyphs	Unicode Characters
A 𝐴 𝒜 𝔸 A 𝒜 𝒜 A	U+0041 LATIN CAPITAL LETTER A
a 𝐚 𝒂 𝕒 a a 𝒂 𝒂 𝐚	U+0061 LATIN SMALL LETTER A
п n $\bar{u}$	U+043F CYRILLIC SMALL LETTER PE
ه ه ه ه	U+0647 ARABIC LETTER HEH
fi fi	U+0066 LATIN SMALL LETTER F + U+0069 LATIN SMALL LETTER I

Unicode: Design Principles (cont)

- **Characters, not Glyphs (cont)**
  - ◇ Glyph shapes, how they are selected and rendered are the responsibility of font vendors (font = set of glyphs)



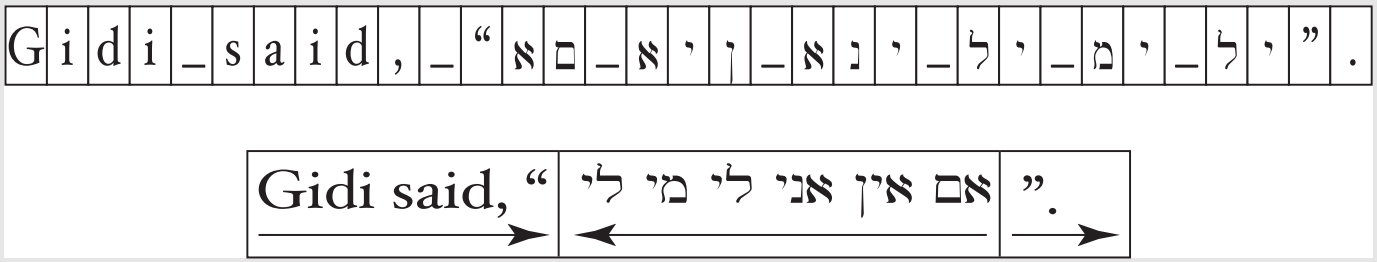
Unicode: Design Principles (cont)

- **Semantics**
  - character property tables are provided for use in, e.g., parsing and sorting algorithms
  - properties include: numeric, spacing, combination, directionality

- **Plain Text**
  - *plain text* is a pure sequence of character codes
    - underlying content stream to which formatting is applied
    - public, standardised, universally readable
  - *fancy/rich text* is plain text plus additional information, such as font size, colour, hypertext links, etc.
    - extra info can be embedded (SGML, HTML, XML, TeX), or in parallel store with links to plain text (word processors)
    - may be implementation-specific, proprietary
  - Unicode encodes plain text only, where
    - "plain text must contain enough information to permit the text to be rendered legibly, and nothing more" (Unicode Standard, Version 10.0.0)

Unicode: Design Principles (cont)

- **Logical Order**
  - Unicode text is stored in logical order – generally the same as the order it would be input via a keyboard
  - In some cases this may differ from display order



  - Directionality property of characters generally sufficient to render plain text, but not always
    - Unicode does contain characters to specify change of direction

- **Unification**
  - duplicate encoding of the same character across languages is avoided in general, e.g.
    - punctuation
    - “Y” (French *i grecque*, German *ypsilon*, English *wye* all represented by U+0057)
  - sometimes this principle is violated to support compatibility with existing standards

Unicode: Design Principles (cont)

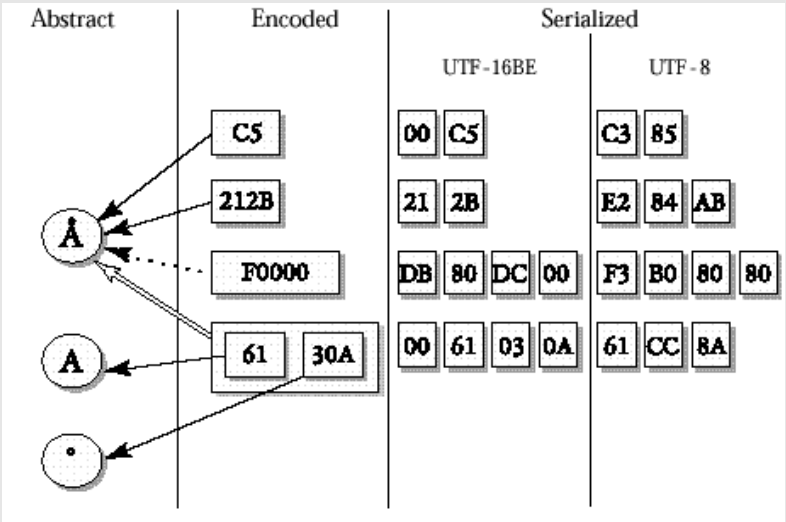
- **Dynamic Composition**
  - by separating codes, for e.g. base characters and accents, Unicode supports the dynamic composition of various forms
  - this process is open-ended – new forms not currently used may be created

- **Stability**
  - Certain aspects of the Unicode Standard must be absolutely stable between versions – so that implementers and users can be guaranteed text data, once encoded will preserve meaning
  - Means once Unicode characters are assigned, their code point assignments cannot be changed, nor can characters be removed
  - Unicode character names also never changed, so that they can be used as valid dentifiers across versions
  - While characters are retained there is a mechanism for deprecating characters whose use is to be discouraged

- **Convertibility**
  - Character identity is preserved for interchange with a number of different base standards (national, international, vendor-specific)

Unicode: Encoding Model

- The Unicode model may be first approximated by a three level model consisting of:
  - ◇ an abstract character repertoire
  - ◇ their mapping to a set of integers also called the codespace
    - a particular integer in this set is called a code point
    - an abstract character is assigned to a code point and is then referred to as an encoded character
    - the Unicode codespace consists of the integers from 0 to $10FFFF_{16}$, comprising 1,114,112 code points
  - ◇ their encoding forms + byte serialization

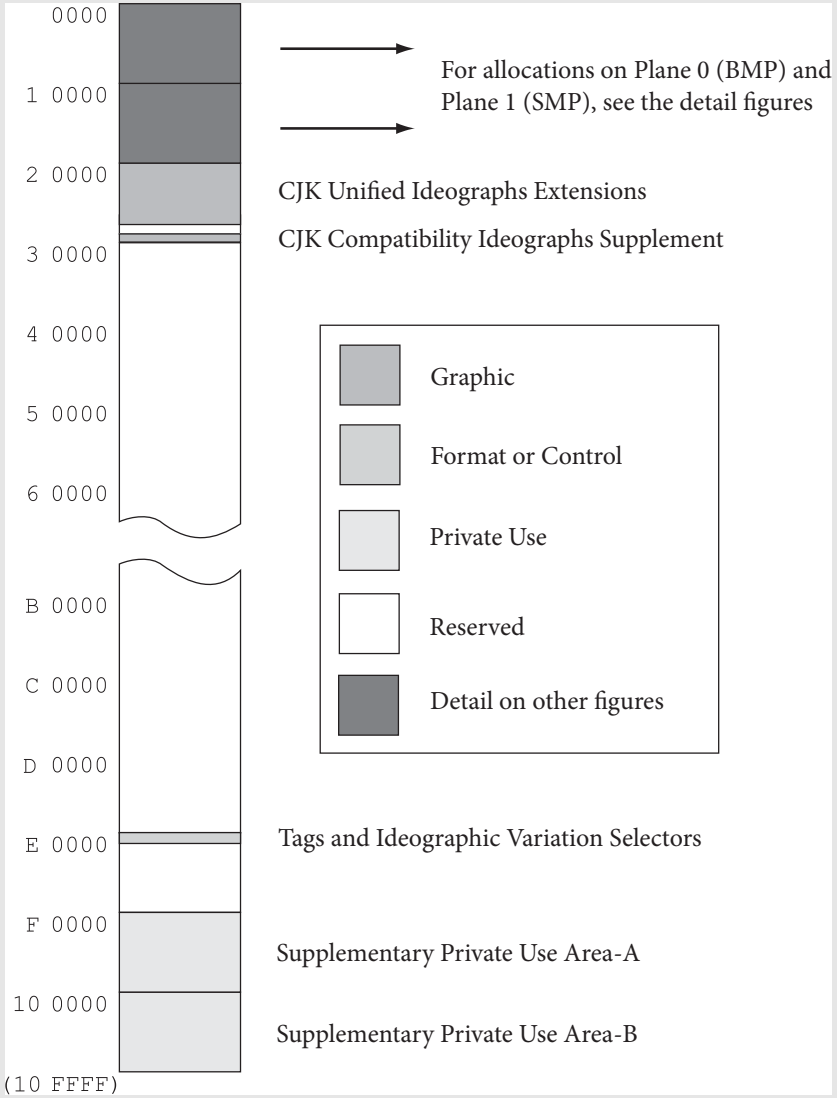


Unicode: Encoding Model (cont)

- Encoded characters are alternate ways the same text element may be assigned an integer code in the Unicode code value space
  - C5 is the precomposed static form of "A-ring"
  - 212B is the code for Angstrom unit
  - F0000 is a hypothetical "private use" surrogate pair
  - 61 30A are "A" and "ring" – to be dynamically composed

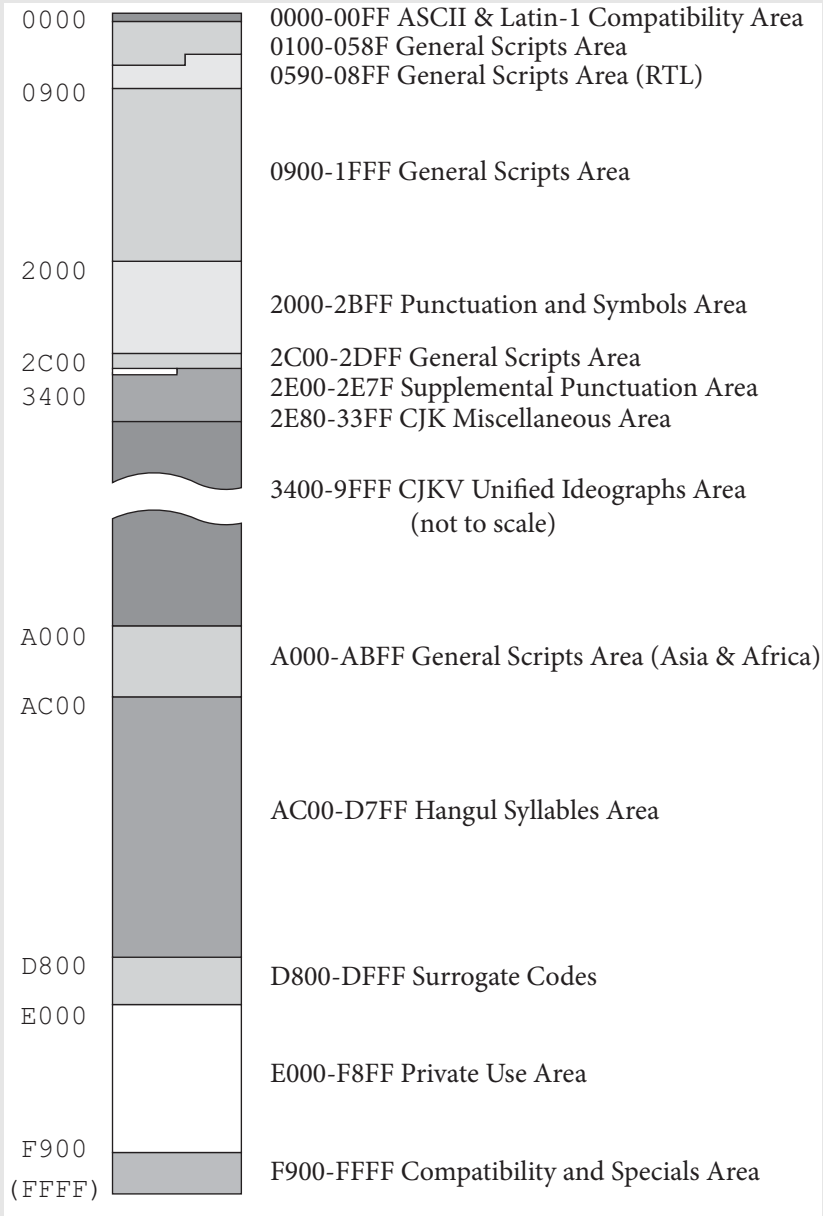
Unicode: Allocation Areas

- Unicode codespace of $10FFFF_{16}$ codepoints is divided into planes each consisting of FFFF (64k decimal) code points
- The first plane (codepoints 0-FFFF), is called the basic multilingual plane(BMP)



Unicode: Allocation Areas (cont)

- Codespace assignment attempts to follow certain principles – e.g. scripts grouped together, in order of accepted standards
- No guarantee of correlation with sort order or case



An Aside on Emoticons and Emojis

- Antecedents for typographic arrangements of punctuation characters to express emotions in text can be found back into the 19th century

Emoticons published in the March 30, 1881 issue of *Puck*.[15]

https://en.wikipedia.org/wiki/Emoticon

- Scott Fahlman, an American AI researcher, is credited with the creation of the :-) and :-(emoticons in 1982; many other textual emoticons followed
- Emoticons (from the English words "emotion" and "icon") are pictorial representations of facial expressions usually made up of punctuation marks (now increasingly rendered as pictures – see below)
- Emojis (from the Japanese *e* (meaning "picture") and *moji* (meaning "character") originated on Japanese mobile phones in 1999 – they are like emoticons but are pictures rather than typographics and are represent much more than facial expressions (animals, weather, activities, flags, etc.)

- Most emoticons are in a single Unicode block (right); in Unicode 11 emojis use 1,250 characters spread across 22 Unicode blocks.

Emoticons[1]
Official Unicode Consortium code chart (https://www.unicode.org/charts/PDF/U1F600.pdf) (PDF)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+1F60x	😀	😁	😂	😃	😄	😅	😆	😇	😈	😉	😊	😋	😌	😍	😎	😏
U+1F61x	😐	😑	😒	😓	😔	😕	😖	😗	😘	😙	😚	😛	😜	😝	😞	😟
U+1F62x	😠	😡	😢	😣	😤	😥	😦	😧	😨	😩	😪	😫	😬	😭	😮	😯
U+1F63x	😰	😱	😲	😳	😴	😵	😶	😷	😸	😹	😺	😻	😼	😽	😾	😿
U+1F64x	🙀	🙁	🙂	🙃	🙄	🙅	🙆	🙇	🙈	🙉	🙊	🙋	🙌	🙍	🙎	🙏

https://en.wikipedia.org/wiki/Emoticon

Encoding Forms and Encoding Schemes

- Once a mapping from an abstract character set to a set of integers has been defined two further mappings need to be specified
  - Character Encoding Form – a mapping from a set of integers to a set of sequences of code units of specified width (e.g. 8-bit bytes).
  - Character Encoding Scheme – a mapping from a set of sequences of code units to a serialized sequence of bytes
- Often character encoding forms and character encoding schemes have the same names – because some default serialization is assumed – but they should not be confused.
- The most common character encoding forms are:
  - UTF-32 (Unicode (or UCS) Transformation Format-16)
  - UTF-16 (Unicode (or UCS) Transformation Format-16)
  - UTF-8 (Unicode (or UCS) Transformation Format-8)

Character Encoding Forms: UTF-32

- UTF-32 (Unicode Transformation Format-32)
  - Each Unicode code point is represented directly by a single 32-bit (4 byte) code unit
  - advantages:
    - Simple: allows all Unicode code points to be mapped into 1 fixed length code units (bytes)
    - Note: a 32-but code could handle up to 4.29 billion distinct characters
  - disadvantages:
    - files containing only Latin texts are four times as large as they are in single byte encodings, such as ASCII or ISO Latin-1 or UTF-8
    - not backwards/forwards compatible with ASCII – so programs that expect a single-byte character set won't work on a UTF-32 file *even if it only contains Latin text*.

Character Encoding Forms: UTF-16

- UTF-32 (Unicode Transformation Format-32)
  - original/default Unicode encoding form – characters are assigned a 16-bit value, except for surrogate pairs which consist of two 16-bit values
  - advantages:
    - allows all Unicode code points to be mapped into 2 code units (bytes)
  - disadvantages:
    - files containing only Latin texts are twice as large as they are in single byte encodings, such as ASCII or ISO Latin-1
    - not backwards/forwards compatible with ASCII – so programs that expect a single-byte character set won't work on a UTF-16 file *even if it only contains Latin text*.

An Aside on Surrogate Pairs

- Unicode UTF-16 text consists of sequences of 16-bit character codes
  - in principle this allows for the encoding of 65,536 distinct values (enough for all scripts currently used)

- Since this may not be sufficient for all existing and possible future scripts, an extension mechanism has been built in, allowing two 16-bit values to represent a single character – these are called **surrogate pairs**
  - the first value in the pair is the **high surrogate**
  - the second value in the pair is the **low surrogate**

- To avoid introducing an escape character to indicate the beginning of a surrogate pair, codes in the range U+D800 to U+DFFF are reserved for use in surrogate pairs (i.e. 2048 codes are sacrificed)
  - U+D800 - U+DBFF are used for the high surrogate
  - U+DBFF - U+DFFF are used for the low surrogate

An Aside on Surrogate Pairs (continued)

- Unicode scalar value (aka "code point") N in the range 0 to $10FFFF_{16}$ is assigned to a character sequence S as follows:

$$N = U \qquad \text{If } S \text{ is a single, non-surrogate value } \langle U \rangle$$

$$N = (H - D800_{16}) * 400_{16} + (L - DC00_{16}) + 10000_{16} \qquad \text{If } S \text{ is a surrogate pair } \langle H, L \rangle$$

- The reverse mapping from a Unicode scalar value S to a surrogate pair is given by:

$$H = (S - 10000_{16})/400_{16} + D800_{16} \text{ (high surrogate)}$$
$$L = (S - 10000_{16})\%400_{16} + DC00_{16} \text{ (low surrogate)}$$

- Using this mechanism Unicode can represent more than 1 million distinct characters ($10FFFF_{16} = 1{,}114{,}111_{10}$)

Character Encoding Forms: UTF-8

- UTF-8 (Unicode Transformation Format-8)
  - maps a Unicode scalar value onto 1 to 4 bytes (see table)
    - 1st byte indicates number of bytes to follow
    - one byte sufficient for ASCII code values (1..127)
    - two bytes sufficient for most non-ideographic scripts
    - four bytes needed only for surrogate pairs
  - advantages:
    - existing ASCII files are already UTF-8
    - most broadly supported encoding form today
  - disadvantages:
    - ideographic (mostly Asian) languages require 3 bytes/character – so UTF-8 encodings are larger for Asian languages than UTF-16 and most existing encodings

Scalar Value	**UTF-16**	**UTF-8 Bit Distribution**			
		1st Byte	**2nd Byte**	**3rd Byte**	**4th Byte**
000000000xxxxxxx	000000000xxxxxxx	0xxxxxxx			
00000yyyyyxxxxxx	00000yyyyyxxxxxx	110yyyyy	10xxxxxx		
zzzzyyyyyyxxxxxx	zzzzyyyyyyxxxxxx	1110zzzz	10yyyyyy	10xxxxxx	
uuuuuzzzzyyyyyyxxxxxx	110110wwwwzzzzyy+ 11011yyyyxxxxxx	11110uuu[a]	10uuzzzz	10yyyyyy	10xxxxxx

[a] Where uuuuu = wwww + 1 (to account for addition of 10000_{16} in computing scalar value of surrogate pair)

Character Encoding Schemes

- For code units wider than 1 byte, platform discrepancies mean byte order must be specified in encoding to ensure portability
- Thus, for two byte encoding forms like UTF-16, there are two possible byte orderings
  - In the UTF-16BE big-endian character encoding scheme
    <004D 0061 0072 006B> is serialized as
    <00 4D 00 61 00 72 00 6B>
  - In the UTF-16LE little-endian character encoding scheme
    <004D 0061 0072 006B> is serialized as
    <4D 00 61 00 72 00 6B 00>
- To determine which byte ordering is being used, Unicode has two special codes called byte order markers
  - these are mirror images of each other (U+FEFF and U+FFFE)
  - Conformance to the Unicode standard does not require use of byte order marks; but their presence in other text stream codings is sufficiently rare that they can generally be used as a indicator of a Unicode text (i.e. as a Unicode signature)

Summary

- Character encoding is necessary for the representation of texts in digital form
- Standards must be agreed if such data is to be widely shared
- Many standards have emerged over the years, even for single languages and languages as relatively straightforward as English
- A single multilingual encoding standard permits
  - processing of multiple languages in one document
  - reuse of software for products dealing with multiple languages (especially Web-related software)
- Unicode is emerging as a global standard, though not without contention

Summary (cont)

- Key features of Unicode include:
  - a commitment to encode plain text only
  - attempt to unify characters across languages, modulo support for existing standards
  - use of well-defined character property tables to associate additional information with characters
  - an encoding model which clearly separates:
    - the abstract character repertoire
    - their mapping to a set of integers
    - their encoding forms + byte serialization
- Unicode model involves specifying
  - the abstract character repetoire to be encoded
  - a mapping from the characters to the integers in the range 0 - $10FFFF_{16}$
  - a mapping from integers in this range onto sequences of bytes

Summary (cont)

- Basic Unicode only uses integers in the range 0 - $FFFF_{16}$
  However, pairs of Unicode values can be used, via the **surrogate pair** mechanism, to extend the code space up to $10FFFF_{16}$
  All encodings for existing languages are in the range 0 - $FFFF_{16}$
- The most common encoding forms for Unicode values are
  - UTF-32 a fixed with encoding with one code point per 4 bytes
  - UTF-16, a fixed width encoding, which straightforwardly maps integers in the range 0 - $FFFF_{16}$ onto 2 bytes
    - UTF-16 comes in 2 flavours, UTF-16BE/LE, to support big-endian/little-endian processors
  - UTF-8, a variable width (1-4 byte) encoding which allows current ASCII-based applications/data to function transparently in a Unicode framework
- Despite its complexity, UTF-8 is currently the most widely supported encoding in programming languages (Java, Perl) and markup languages (XML)

References

- Crystal, D. The Cambridge Encyclopedia of Language. Cambridge University Press, 1987.
- Daniels, P. T. and Bright, W. eds, The World's Writing Systems. Oxford University Press, 1996.
- Steven J. Searle. "A Brief History of Character Codes in North America, Europe and Asia". Available at: `http://tronweb.super-nova.co.jp/characcodehist.html`. Site last visited 06/11/17.
- Unicode Consortium. The Unicode Standard Version 11.0.0. Available at `www.unicode.org`. Site last visited 23/10/18.
- Unicode Technical Report # 17: Character Encoding Model. Available at `www.unicode.org`. Site last visited 06/11/17.