Digital text

The joys of character sets

Contents

- Storing text
  - General problems
  - Legacy character encodings
  - Unicode
  - Markup languages
- Using text
  - Processing and display
  - Programming languages

A little bit about writing systems

Overview

```
   Latin   Cyrillic                Devanagari - - - - - - Tibetan
      \     /                   /  Gujarati                  |
       \   / - Armenian        /   Bengali      SOGDIAN - Mongolian
        \ /                   /    Gurumukhi    SCRIPT
       Greek - Georgian      /     Oriya                  Chinese
         |                  /                            /
         |                 /       Telugu               /
     PHOENICIAN         BRAHMI - - Kannada      SINITIC - Japanese
     SCRIPT    \        SCRIPT     Malayalam    SCRIPT  \
    /    |      \          \       Tamil                 \
Hebrew   |      Arabic      \                             Korean
         |        \          \ - - Sinhala
         |         \          \
         |          \          \ _ _ Burmese
         |           \          \    Khmer
         |            \          \
```

The easy ones

- Latin is the alphabet and writing system used in the West and some other places
- Greek and Cyrillic (Russian) are very similar, they just use different characters
- Armenian and Georgian are also relatively similar

More difficult

- Hebrew is written from right−to−left, but numbers go left−to−right...
- Arabic has the same rules, but also requires variant selection depending on context and ligature forming

The far east

- Chinese uses two ’alphabets’: hanzi ideographs and zhuyin syllables
- Japanese mixes four alphabets: kanji ideographs, katakana and hiragana syllables and romaji (latin) letters and numbers
- Korean uses hangul ideographs, combined from jamo components
- Vietnamese uses latin letters...

The Indic languages

- Based on syllabic alphabets
- Require complex ligature forming
- Letters are not written in logical order, but require a strange ’circular’ ordering
- In addition, a single line consists of separate levels where characters are placed
- There are individual differences

Storing digital text

Bits and bytes

Digital text?

- How do you encode text on a computer?
- Using only strings of binary digits?
- Traditional solution:
  - group bits in groups of eight (*bytes*)
  - interpret each byte as a number
  - use a *character set* that maps numbers to characters

ASCII

- The world’s most important character set
- Basis of nearly all today’s character sets
- Only 7 bits:
  - 0−31: controls (newline, tab, ...)
  - 32−64: punctuation, space and digits
  - 65−90: upper−case letters
  - 97−122: lower−case letters
  - 91−96, 123−: more punctuation

The ISO 8859 series

- Based on ASCII, but extended to 8 bits
- 14 different character sets for different world regions
- A very large percentage of today’s computer users use ISO 8859−1 (Latin1)
- See http://czyborra.com

The full list

- 1: Western Europe
- 2: Eastern Europe
- 3: Esperanto, Malta
- 4: Baltic (obsolete)
- 5: Cyrillic
- 6: Arabic
- 7: Greek
- 8: Hebrew
- 9: Turkish
- 10: Sami, Inuit
- 11: Thai
- 13: Baltic
- 14: Celtic
- 15: Western Europe

The 8859 model

- 0 – 127: Identical to ASCII
- 128 – 159: Control characters
- 160 – 191: Punctuation
- 192 – 255: Local characters (æøå etc)

Proprietary stuff

- Windows uses a set of code pages that modify 8859 by using the 128−159 range for characters
- So most of you use Windows−1252
- The Mac has its own set of character sets (MacRoman, MacGreek, ...)

Alternative 8−bit encodings

- koi8−r   Popular Russian encoding
- ISCII   Indian standard
- VISCII   Vietnamese standard
- VIQR   Vietnamese standard
- Iran System   Iranian encoding (Urdu!)
- Win−Sami−2   Sami ’standard’

Character sets and encodings

- These are *not* the same!
- Character sets:
  - collections of characters
  - in coded ones all characters have numbers
  - no digital representation!
- Character encodings:
  - rules for how to map from digital data to character numbers

Oriental systems

- Separate character sets and character encodings
- Important standards from:
  - China (GB 2312)
  - Japan (ISO 2022−JP, EUC−JP & Shift−JIS)
  - Korea (EUC−KR)
  - Taiwan (Big5, EUC−TW)

Japan

- Character sets:
  - JIS 0201 8−bit, ASCII + katakana
  - JIS 0208 16−bit, ASCII, kana + kanji
  - JIS 0212 16−bit, ditto
- Character encodings:
  - ISO 2022−JP uses 0201 and 0208
  - EUC−JP uses ASCII, 0201, 0208 and 0212
  - Shift−JIS uses ASCII, 0201 and 0208

Unicode

The character set to end
all character sets

A bit of history

- Unicode was defined by an industry consortium (the Unicode consortium)
- The consortium was founded in 1991, and published version 1.0 the same year
- Later, the standard was aligned with ISO 10646, and these two are now parallel
- Unicode 3.0 is the current version

Character set principles

- Characters, not glyphs
- Plain text in logical order
- Unify!
  - same meaning, different shape = one character
- Include compatibility characters
- Use combining marks where possible!

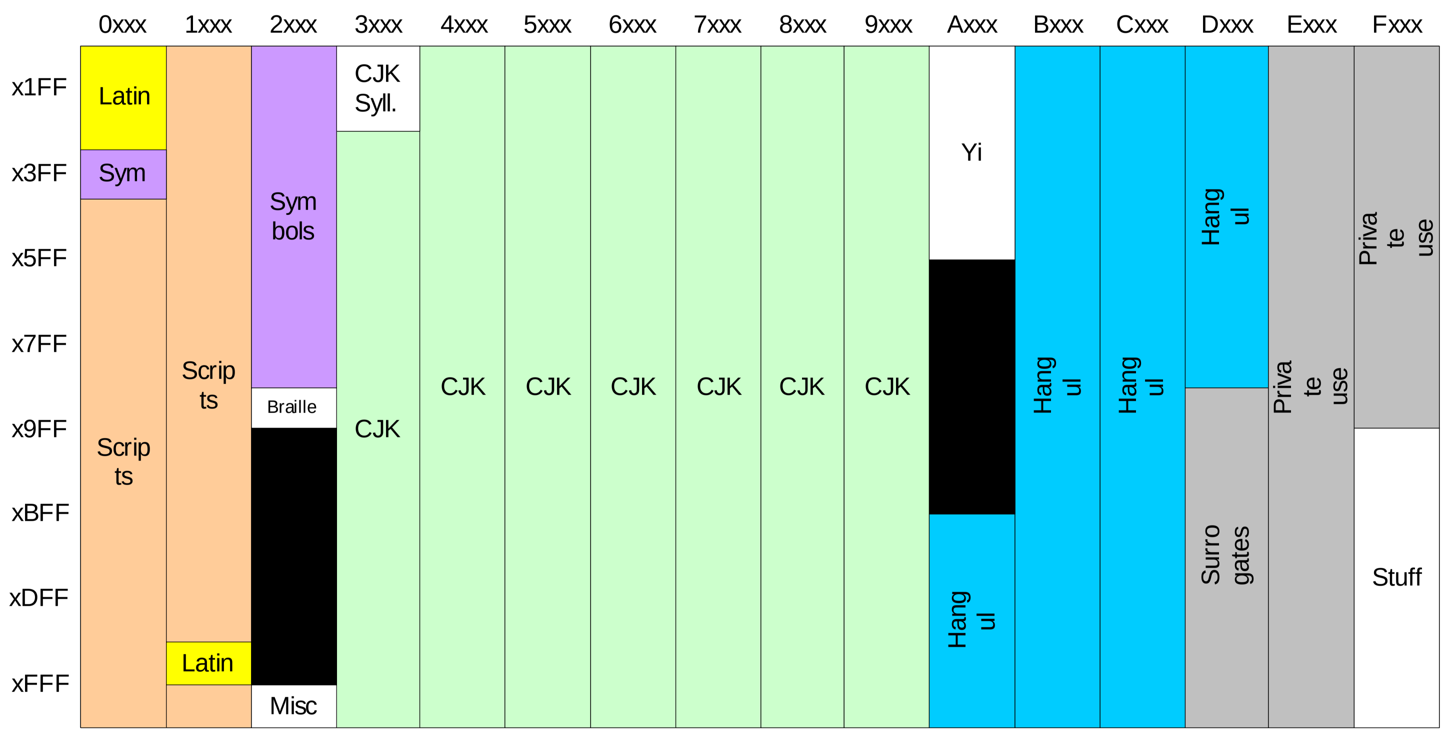
Additional mechanisms

- Character property database
- Display algorithm:
  - dynamic composition
  - bidirectional text rules
- Suggestions for:
  - sorting
  - case folding
  - regexps

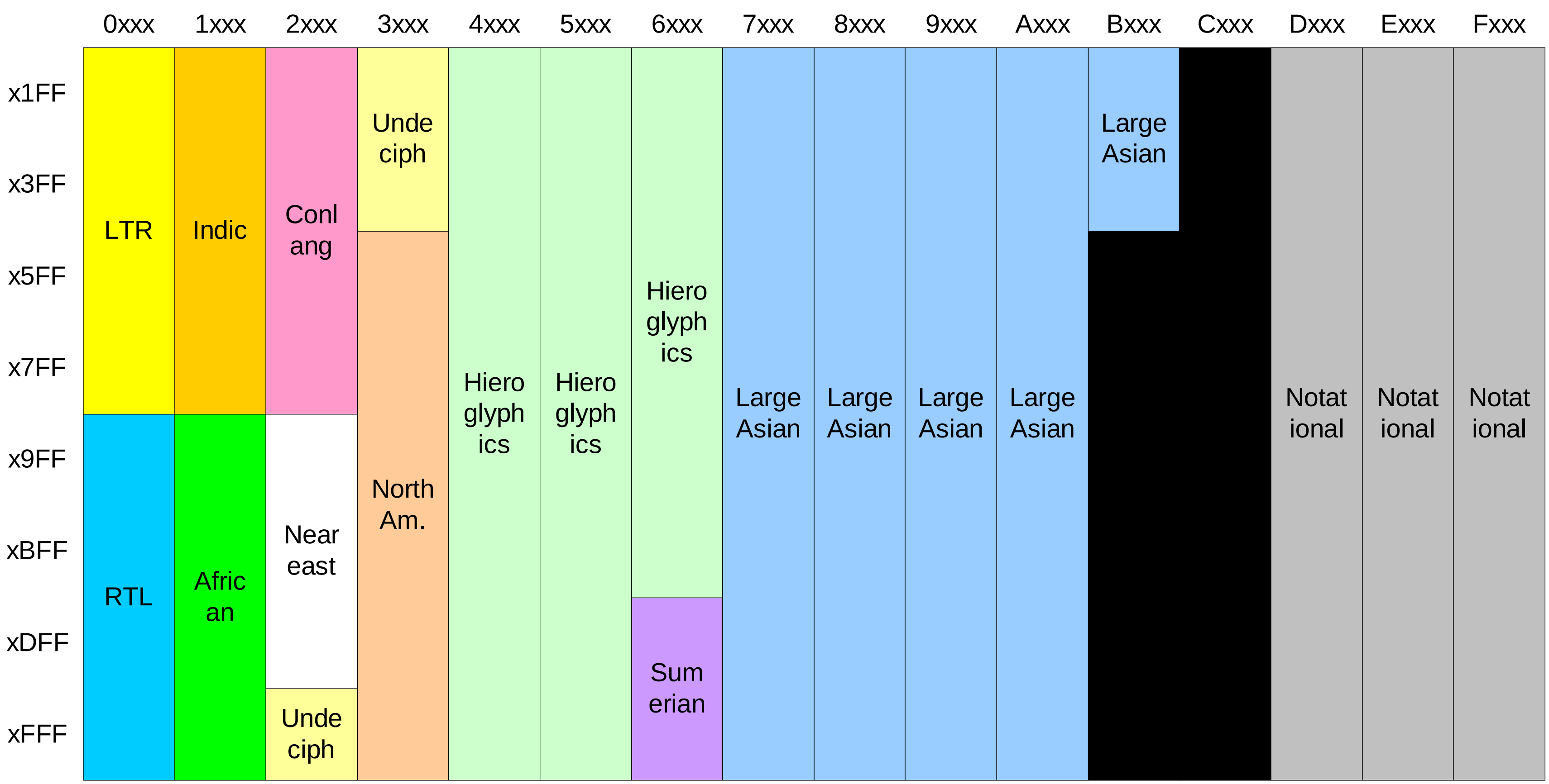
The character set

- Unicode and ISO 10646 are divided into *planes* of 65536 characters:
  - 0: BMP
  - 1: Non−han suppl (dead & invented)
  - 2: Han supplementary (Chinese chars)
  - 14: Language tags
- Only the BMP is currently in use, planes 1, 2 and 14 will be used shortly

BMP structure



Plane 1



0xxx
1xxx
2xxx
3xxx
4xxx
5xxx
6xxx
7xxx
8xxx
9xxx
Axxx
Bxxx
Cxxx
Dxxx
Exxx
Fxxx
x1FF
x3FF
x5FF
x7FF
x9FF
xBFF
xDFF
xFFF
LTR
RTL
Indic
Afric an
Conl ang
Near east
Unde ciph
Unde ciph
North Am.
Hiero glyph ics
Hiero glyph ics
Hiero glyph ics
Sum erian
Large Asian
Large Asian
Large Asian
Large Asian
Large Asian
Notat ional
Notat ional
Notat ional

The Unicode encodings

- UTF−7 7−bit safe encoding
- UTF−8 8−bit encoding, ASCII−compatible
- UCS−2 Straight 16−bit, only BMP
- UTF−16 Improved UCS−2, all planes
- UCS−4 Straight 32−bit
- UTF−32 Ditto

’æ’ in some encodings

- UTF−7: 0x2B 0x41 0x34 0x59 (’+A4Y’)
- UTF−8: 0xC3 0xA6 (’Ã|’)
- UCS−2: 0xE6 0x00
- UTF−16: 0xE6 0x00
- UCS−4: 0xE6 0x00 0x00 0x00
- UTF−32: 0xE6 0x00 0x00 0x00

Markup languages

Their handling of i18n

SGML

- *Document character set* declared in SGML declaration
- This controls *abstract characters* that may appear in documents
- Character references follow DCS
- Actual documents may be converted to DCS by the entity manager
- Characters outside DCS are not allowed!

SGML practice

- Old systems used lots of SDATA entities
- This was essentially a character set in itself, with SGML as the encoding
- This is now an *obsolete* practice!
- Unicode is here, and that makes SDATA entities pointless

HTML

- DCS is Unicode
- Any character set can be used, provided it is declared in the header
- <meta http−equiv="content−type"
  content="text/html; charset=...">
- Character references are to Unicode character

XML

- DCS is Unicode
- Default encodings are UTF−8 and UTF−16
- Autodetection decides which is used
- *All* other encodings *must* be declared in the XML declaration
- <?xml version="1.0" encoding="..."?>
- Each entity can have its own encoding

Encoding identification

- IANA maintains a registry of ’charset’ names useful for encoding identification
- This is used by:
  - HTML and HTTP
  - MIME
  - CSS
  - XML
- Java uses a completely different naming system

Actually using the text

Problems problems problems

How do you sort internationally?

- Many languages (Hungarian!) have very complex sorting rules
- How do you sort text in a script with thousands of characters?
- Swedish and Norwegian order the same characters differently
- So what is the general solution?

Sorting solutions

- Unicode TR#10 presents one collation algorithm with locale tailoring
- ISO 14561 presents another, which is also tailorable
- Java has a collation API in java.text.Collator

Case mapping

- Not all scripts have cases!
- Casing is language−dependent and context−dependent!
- Some letters are in title−case! (Dz)
- Case mapping is not always reversible
- The Unicode character properties database has general mapping tables
- There are also locale−specific tables

Searching

- The same character can be represented:
  - in different character sets
  - with different mechanisms (entities, char refs)
  - in different ways (combining marks)
- So, how to solve this?
- Canonical representation!

Display

- Supporting different writing directions, and mixing of these
- Many languages require complex ligature forming based on context
- Indic languages require character reordering
- What if your selected font does not have all the necessary characters?
- Line breaking rules can be very complex

Programming language support

Unicode, or not Unicode,

that is the question

C

- The only string types are ’char’ and ’wchar’
- ’char’ is in general abused to mean both character and byte
- ’wchar’ is in general not portable
- all standardized APIs use ’char’
- Modifying 8−bit code to 16−bit can be very difficult, due to buffer size problems and memory allocation issues

C++

- In theory much better, due to the standard ’string’ and ’wstring’ classes
- In practice developers generally use ’char’
- Not all environments have good support for the C++ Standard Template Library
- Most environments use their own string types (QString, LPWSTR, ...)

Python

- Version 2.0 added Unicode support
  - Unicode string literals, functions and regexps
  - Unicode stream objects
- Internal encoding is UTF−16
- Strict separation of 8−bit strings with no semantics and 16−bit Unicode strings

Perl

- Version 5.005 added Unicode support
- Much the same functionality as Python
- Internal encoding is UTF−8
- Apparently, this support has been very buggy and many people claim that it is unusable

Java

- Defined with Unicode support
- ’char’ is 16−bit, ’String’s hold Unicode
- APIs cleanly separate byte streams and character streams
- Very good APIs for many aspects of internationalization

Miscellaneous

- tcl 8.0 has Unicode support
- Ada95 has Unicode support in the Ada standard
- Many Common Lisp implementations have very good Unicode support