Crash Course on Character Encodings

Yusuke Shinyama

NYCNLP
Oct. 27, 2006
Introduction
Are they the same?

- Unicode
- UTF
<table>
<thead>
<tr>
<th>Character</th>
<th>Character Code</th>
<th>Byte Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>ض</td>
<td>1590</td>
<td>216 182</td>
</tr>
<tr>
<td>美</td>
<td>32654</td>
<td>231 190 142</td>
</tr>
</tbody>
</table>
Two Mappings

<table>
<thead>
<tr>
<th>Character</th>
<th>Unicode</th>
<th>Character Code</th>
<th>Byte Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>64</td>
<td>1590</td>
<td>216 182</td>
</tr>
<tr>
<td>ض</td>
<td>32654</td>
<td>231 190 142</td>
<td></td>
</tr>
</tbody>
</table>

“Character Set”  “Encoding Scheme”
Terminology

• **Character Set**
  - Mapping from abstract characters to numbers.

• **Encoding Scheme**
  - Way to represent (encode) a number in a byte sequence in a decodable way.
  - Only necessary for character sets that have more than 256 characters.
In ASCII...

<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII Code</th>
<th>Byte Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>A</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>m</td>
<td>109</td>
<td>109</td>
</tr>
</tbody>
</table>
Character Sets
Character Sets

• ≤ 256 characters:
  - ASCII (English)
  - ISO 8859-1 (English & Western European languages)
  - KOI8 (Cyrillic)
  - ISO-8859-6 (Arabic)

• 256 < characters:
  - Unicode
  - GB 2312 (Simplified Chinese)
  - Big5 (Traditional Chinese)
  - JISX 0208 (Japanese)
  - KPS 9566 (North Korean)
### Character Sets

<table>
<thead>
<tr>
<th></th>
<th>ASCII</th>
<th>ISO 8859-1</th>
<th>ISO 8859-6</th>
<th>GB 2312</th>
<th>Unicode</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>65</td>
<td>65</td>
<td>65</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>ë</td>
<td>-</td>
<td>235</td>
<td>-</td>
<td>-</td>
<td>235</td>
</tr>
<tr>
<td>ض</td>
<td>-</td>
<td>-</td>
<td>214</td>
<td>-</td>
<td>1590</td>
</tr>
<tr>
<td>美</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>50112</td>
<td>32654</td>
</tr>
<tr>
<td>❤️</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>9829</td>
</tr>
</tbody>
</table>
Unicode Standard

- History
  - ISO Universal Character Set (1989)
  - Unicode 1.0 (1991)
    - 16-bit fixed length codes.
  - Unicode 2.0 (1996)
    - Oops, we’ve got many more.
    - Extended to 32 bits.
  - Unicode 5.0 (2006)
    - Keep growing...
Unicode Standard

- Hexadecimal notation (U+XXXX).
- ISO 8859-1 is preserved as the first 256 characters.

<table>
<thead>
<tr>
<th></th>
<th>ISO 8859-1</th>
<th>Unicode</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>65</td>
<td>U+0041 (65)</td>
</tr>
<tr>
<td>ë</td>
<td>235</td>
<td>U+00EB (235)</td>
</tr>
</tbody>
</table>
Problems in Unicode

• Politics (Microsoft, Apple, Sun, ...)
• Lots of application specific characters.
  - Musical notes, circuit symbols, meteorological symbols, etc.
  - Some incomplete, others too detailed.
  - http://unicode.org/charts/
• CJK unification.
# Spaces and Hyphens

<table>
<thead>
<tr>
<th>Space</th>
<th>Code</th>
<th>Description</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>U+0020</td>
<td>Hyphen-minus (-)</td>
<td>U+002D</td>
</tr>
<tr>
<td>No-break Space</td>
<td>U+00A0</td>
<td>Hyphen (-)</td>
<td>U+2010</td>
</tr>
<tr>
<td>Em Space</td>
<td>U+2003</td>
<td>Minus (−)</td>
<td>U+2212</td>
</tr>
<tr>
<td>Thin Space</td>
<td>U+2009</td>
<td>Figure Dash (−)</td>
<td>U+2012</td>
</tr>
<tr>
<td>Zero-width Space</td>
<td>U+200B</td>
<td>Em Dash (—)</td>
<td>U+2014</td>
</tr>
<tr>
<td>Ideographic Space</td>
<td>U+3000</td>
<td>Quotation Dash (―)</td>
<td>U+2015</td>
</tr>
</tbody>
</table>
## Different “A”s

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>U+0041</td>
<td>Latin</td>
</tr>
<tr>
<td>A</td>
<td>U+0391</td>
<td>Greek</td>
</tr>
<tr>
<td>A</td>
<td>U+0410</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>A</td>
<td>U+FF21</td>
<td>Japanese</td>
</tr>
</tbody>
</table>
### Problems with CJK

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>写 写 写</td>
<td>U+5199</td>
<td>Japanese</td>
</tr>
<tr>
<td>葛 葛 葛</td>
<td>U+845B</td>
<td>Chinese</td>
</tr>
<tr>
<td>饒 饒 饒</td>
<td>U+9949</td>
<td>Japanese</td>
</tr>
<tr>
<td>饒 饒 饒</td>
<td>U+9949</td>
<td>Chinese</td>
</tr>
</tbody>
</table>
Inconsistency

- They “focused on semantic distinction,” but...

<table>
<thead>
<tr>
<th>fi</th>
<th>Ligature for “f i”</th>
</tr>
</thead>
<tbody>
<tr>
<td>a⁴</td>
<td>Superscript number</td>
</tr>
<tr>
<td>⊒</td>
<td>Box drawing characters</td>
</tr>
</tbody>
</table>

- Printing industry (e.g. Adobe) doesn’t rely on Unicode.
Encoding Schemes
Encodings

Same character set, different encoding schemes.

Unicode

美 → U+7F8E

UTF-8: 231 190 142
UTF-16LE: 142 127 127 142
UTF-16BE: 142 127 127 142
Punycode: 108 116 48 97
Encodings

• Most encoding schemes are associated with a certain character set.
  - UTF-8 or UTF-16 → Unicode
  - EUC-CN → GB 2312 (Simplified)
  - Big5 → Big5 (Traditional)
  - ISO 2022-JP or EUC-JP or Shift_JIS → JISX 0208 (Japanese)

• Often considered representing the character set itself.
Strictly Speaking...

**Character Set**

**Encoding Scheme**

<table>
<thead>
<tr>
<th>Character</th>
<th>Character Code</th>
<th>Byte Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>ض</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>美</td>
<td>50112</td>
<td>195 192</td>
</tr>
</tbody>
</table>

GB 2312  EUC-CN
Often Viewed as...

Character | Character Code | Byte Sequence
---|---|---
A | | 64
ضر | | 
美 | GB-2312 | 195 192
Unicode Encodings

- UTF-8 (Thompson, 1992)
  - Variable length (1~6 bytes).
    - 0xxxxxx (U+0000~U+007F)
    - 110xxxxx 10xxxxxx (U+0080~U+087F)
    - 1110xxx 10xxxxxx 10xxxxxx (U+0880~U+1187F)
    - ...
  - Easy to parse and recover.
    - MSB = 0 → an individual character.
    - MSB = 10 → a middle byte of a character.
    - MSB = 110, 1110, ... → the first byte of a character.
Unicode Encodings

- UTF-8 (Thompson, 1992)
  - ASCII characters are always represented as a single byte regardless of its context. (Good for SGML or URL)

  \[
  \text{<TEXT>_ASMQ<TEXT>}
  \]

  \[
  \text{<TEXT>}</TEXT>
  \]

- As long as you only care ASCII characters and keep others untouched, your code is UTF-8 safe.
Unicode in Documents
Need to Know...

• You have to know the encoding scheme (therefore its character set) of a document in advance.

• Encodings can be expressed in-document or externally.
  - HTML: `<meta>` tag, web server header.
  - E-mail: `Content-Type:` header.
Misrecognition

Unicode UTF-8

美 231 190 142

Greek ζ 3/4

ISO 8859-1

GB 2312 EUC-CN

JISX 0802 Shift_JIS
HTML 4.0

• Specify the encoding
  - By an external HTTP header:
    
    Content-Type: text/html; charset=UTF-8
  
  - By an in-document HTML tag:
    
    <meta http-equiv="Content-Type"
    content="text/html; charset=UTF-8">

• Use entity notation.
  - Copyright © 2006 (U+00A9 ©)
  
  - Oct. 26, 2006 (U+2014 —)
XML 1.0

• Default: UTF-8
  - “Strongly recommended.”

• Can specify the encoding with a special directive.
  - `<?xml version="1.0"?>`
  - `<?xml version="1.0" encoding="iso-8859-1"?>`
Handling Unicode with Unix Tools
To View

- FireFox
  - “View” → “Character Encoding” → “Unicode”
- xterm + GNU unifont
  
  $ xterm -en utf-8
- Less (no emulation)
  
  LESSCHARSET=UTF-8
- LV (with emulation)
  
  $ lv -Iu8 -Oej file.txt
How xterm works?

- Byte sequences are interpreted as a specific encoding.

UNIX Process ➔ stdout ➔ xterm

Default: interpreted as ISO 8859-1
To Edit

- Emacs + Mule-UCS
  
  (require 'un-define)

- XEmacs

- yudit (http://www.yudit.org)

- gedit (buggy!)
To Convert

- iconv

  ```
  $ iconv -f ISO-8859-6 -t UTF-8
  arabic > utf8
  ```

  (Converts ISO 8859-6 into UTF-8)

- LV

  ```
  $ lv -Iu8 -Oec utf8 > gb2312
  ```

  (Converts UTF-8 into GB 2312)

- Python
Irreversible Nature

• Once you convert a text into Unicode, you cannot get it back to the original character set without losing information.

• Welcome to the World of the Future!
When Clueless...

- Make a guess with hex dump:
  - $ od -tx1 unknown.txt
    ff fe 3c 00 68 00 74 00
    6d 00 6c 00 3e 00 0a 00
    ...
  - UTF-16?
    - BOM (U+FEFF) at the first byte.
    - 00 appears many times.
    - “<html>
      ”
Programming for Unicode
Modelling Characters

- In modern programming languages, internal/external representation of characters are separated:
  - Byte (external representation)
  - Character (abstract entity)

- Absorb the surface differences with encoder and decoders. ("codecs")
Modelling Characters

- **Example**
  - **Input**
    - “Decode” a UTF-8 sequence as characters.
  - **Process**
    - Manipulate an array of two character objects.
  - **Output**
    - “Encode” the characters as a GB 2312 sequence.

\[
\begin{array}{c}
\text{UTF-8 Decoder} \\
\text{Character Sequence} \\
\text{GB 2312 Encoder} \\
\text{Byte Sequence} 41 \text{ e7 be 8e} \\
\end{array}
\]
Java

- Two important types:
  - String object (Unicode string)
  - ByteBuffer object (byte sequence)
- Non-ASCII characters can be embedded with \u notation.
  - "ABC" (U+0041, U+0042, U+0043)
  - "A\u7f8e" (U+0041, U+7F8E)美
Java

- java.io.DataInput/DataOutput
  - Only UTF-8 sequence is supported.

    // Read UTF-8 characters.
    DataInputStream in = ...;
    String s = in.readUTF();

    // Write UTF-8 characters.
    DataOutputStream out = ...;
    out.writeUTF("美");
Java

- java.nio.Charset
  - Supports various encodings.

```java
// Obtain a Charset object.
Charset cs = Charset.forName("gb2312");

// Encode the string to GB 2312.
ByteBuffer buf = cs.encode("\u7f8e");

// Decode the byte sequence as GB 2312.
String s = cs.decode(buf);
```
• Two string types:
  - Byte sequence: `str. ("abc")`
  - Unicode string: `unicode. (u"abc")`

```python
>>> unicode(\xe7\xbe\x8e", "utf-8")
u\u7f8e'    (Construct a Unicode string)

>>> u\u7f8e'.encode("utf-8")
'\xe7\xbe\x8e'    (encode in UTF-8)

>>> u\u7f8e'.encode("gb2312")
\xc3\xc0'    (encode in GB 2312)
```
Python

• Implicit Conversion

```python
>>> print u"\u0041"
A
>>> print u"\u7f8e"
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
UnicodeEncodeError: 'ascii' codec can't encode character u'美' in position 0: ordinal not in range(128)
```
Python

• Easy to embed string constants.

```python
# -*- encoding: gb2312 -*-  (Magic comments)
print u"美 means beauty."  
```

• XML parsing in 3 lines of code.

```python
# parse "apf.xml" file.
fp = file("apf.xml")
root = minidom.parse(fp)
# get all <element> tags.
print root.getElementsByTagName("entity")
```
Common Lisp

• SBCL + SLIME + Mule-UCS
  - Supports UTF-8.

```lisp
(setq slime-net-coding-system 'utf-8-unix)

CL-USER> (code-char #x7f8e)
; => #\U7F8E

CL-USER> (coerce '(#\U7F8E) 'string)
; => ”美”
```
Conclusion

• Character sets (e.g. Unicode) and encoding schemes (e.g. UTF-8) are two different things.

• Unicode is a messy standard, but we have to get along with it.

• In modern programming languages, abstract characters and their external representation are separated.