Dasher and Unicode

Chris Ball

cjb@mrao.cam.ac.uk

June 6, 2005
Motivations for this talk

- Internationalisation is important
- There are pitfalls
- Dasher seems to have got it right – over 100 languages!
- We will tell you all our dirty little secrets
  - (Well, our professional ones)
The problem

- Historically, there have been hundreds of ways for representing characters as numbers – ASCII, EBCDIC, Shift-JIS, ...
- Internationalised software would have to detect and support all of these encodings
- Unicode tries to be a single solution for internationalisation
- Contains glyphs for over 100,000 characters
- Each character is defined by a “code-point” in hexadecimal
- E.g.:
  - $\text{U+221E} = \infty$
  - $\text{U+00E9} = é$
Example codepoints

![Codepoint chart with Japanese characters and corresponding code points]
In its simplest encoding, Unicode needs two (or even four) bytes per character.

UTF-8 is a “variable-width” encoding, $1 \leq \text{bytes} < 6$.

ASCII is valid UTF-8.

When writing Roman text, UTF-8 uses one byte per character.
In UTF-8, the high bit denotes whether there are subsequent bytes
- 01000001 = 65 = A, leading zero says only one byte

When you need a multiple-byte character:
- The two high bits are set to (11): begin a multi-byte character
- The two high bits are set to (10): continue that character

This makes it possible not to “waste” bytes on Roman text
Dasher defines a language by:

- An alphabet file
- A training text
- A colour scheme (optional)
Alphabet file

- Lists the valid characters for a language
- Organises the characters into “groups”
- Tells Dasher where to find the training text
- May specify colour scheme, writing orientation
Alphabet file example: English

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE alphabets SYSTEM "alphabet.dtd">
<?xml-stylesheet type="text/xsl" href="alphabet.xsl"?>
<alphabets>
  <alphabet name="English alphabet - limited punctuation">
    <orientation type="LR"/>
    <palette>European/Asian</palette>
    <train>training_english_GB.txt</train>
    <group name="Lower case Latin letters" b="0">
      <s d="a" t="a"/>
      <s d="b" t="b"/>
      <s d="c" t="c"/>
      ...
    </group>
    <group name="Upper case Latin letters" b="111">
      <s d="A" t="A"/>
      <s d="B" t="B"/>
      <s d="C" t="C"/>
      ...
    </group>
    <group name="Punctuation" b="112">
      <s d="!" t="!"/>
      <s d="," t=","/>
      <s d="." t="."/>
      ...
    </group>
  </alphabet>
</alphabets>
```
Alphabet file example: Japanese

```xml
<?xml version="1.0"?>
<!DOCTYPE alphabets SYSTEM "alphabet.dtd">
<?xml-stylesheet type="text/xsl" href="alphabet.xsl"?>
<alphabets>
  <alphabet name="Nihongo / Japanese Kana and 7000 Kanji">
    <orientation type="UD"/>
    <palette>Hiragana</palette>
    <train>training_Japanese_JP.txt</train>
    <group name="Hiragana" b="0">
      <s d="&amp;#3041;" t="&amp;#3041;" b="10" />
      <s d="&amp;#3042;" t="&amp;#3042;" b="10" />
      <s d="&amp;#3043;" t="&amp;#3043;" b="10" />
      ...
    </group>
    <group name="kanji" b="9">
      <s d="&amp;#3303;" t="&amp;#3303;" />
      <s d="&amp;#3300;" t="&amp;#3300;" />
      <s d="&amp;#3301;" t="&amp;#3301;" />
      ...
    </group>
    <group name="Punctuation" b="112">
      <s d="&amp;#300C;" t="&amp;#300C;" note="Asian left single quotation mark" />
      <s d="&amp;#300D;" t="&amp;#300D;" note="Asian right single quotation mark" />
      <s d="&amp;#300E;" t="&amp;#300E;" note="Asian left double quotation mark" />
      ...
    </group>
  </alphabet>
</alphabets>
```
A corpus of text, with no other information attached

When Dasher trains, it will increment the PPM count for the context each symbol appears in

The encodings in the alphabet file and training text must match!
Normalisation

What about when one character can *alter* the previous character?

- Examples: French (e-acute), Arabic, Hiragana (accents)
Normalisation

- This would be a mess if we had to do it ourselves
- But we don’t!
- Unicode contains characters that combine with previous ones
Normalisation

Example:
- U+0065 (E) followed by
- U+0301 (Combining acute)

Generates:
- U+00E9 (Latin small letter E with acute)
Normalisation

The two strings both represent \textit{e-acute}, but in different forms.

- \texttt{U+0065 U+0301} is in NFD (Normalized Form Decomposition)
- \texttt{U+00E9} is in NFC (Normalized Form Composition)
There is no such thing as a plain text file

- Text = encoding + data
- Always know your encoding

XML is useful for character interchange
- Handles encoding, cross-platform issues for you

Choose a normalisation form and enforce it throughout
Conclusion: Adding new languages to Dasher

- We need an alphabet file and a training text for the new language
- Both are stored in UTF-8
- Some languages have variants for composed/decomposed alphabets
Thank you!

Questions?