ISO/IEC JTC/1 SC/2 WG/2 **N2312** L2/01-025
2001-01-08

ISO/IEC JTC/1 SC/2 WG/2
Universal Multiple-Octet Coded Character Set (UCS)

Title:	**Presentation of tone contours encoded as UCS tone letter sequences**
Doc. Type:	Expert contribution
Source:	Peter Constable, SIL International
Action:	For consideration by WG2
References:	WG2 N2307, N2195
Distribution:	WG2 members

Introduction

Concern was expressed in section 3.3 of WG2 N2195 that the existing tone letter characters in the UCS (U+02E5..U+02E9) are not adequate to represent contour tones, such as high-rising, that occur in languages such as Cantonese. N2307 explains that the existing tone letter characters are, in fact, adequate for representing contour tones by the use of tone letter sequences. For example, a high-rising tone (a 45 contour) can be represented using a sequence of characters 02E6+02E5.

The concern expressed in N2195 appears to be motivated not by the need for an encoded representation of contour tones, but by the problem of appropriate presentation of an encoded representation using glyphs with contour shapes. This is a prototypical case of the distinction between characters and glyphs, as asserted by ISO/IEC TR 15285, and of the need for information systems to provide complex character sequence-to-glyph mappings as specified in TR 15285.

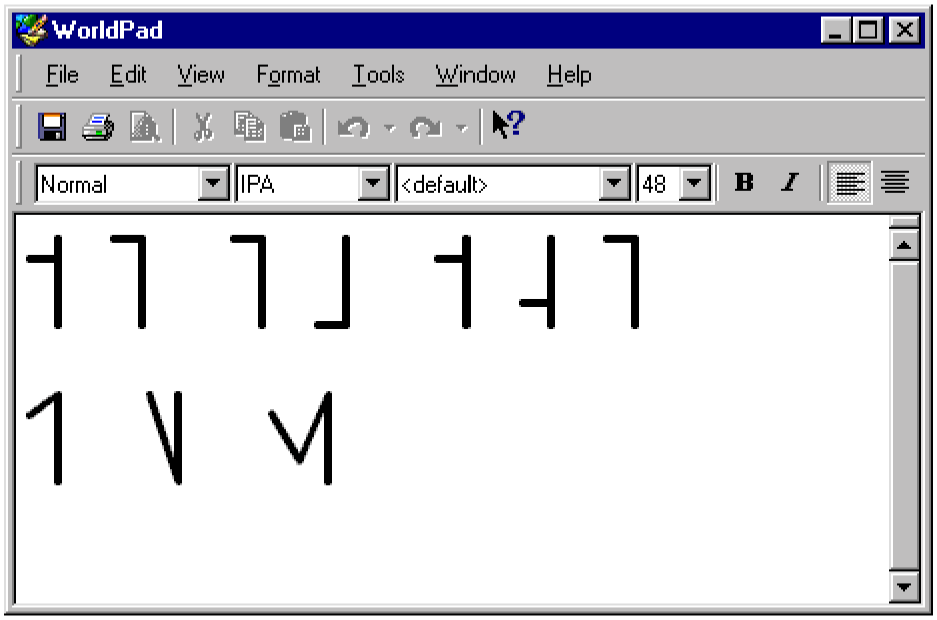
I wish to support the opinion expressed in N2307, that the existing tone letter characters 02E5..02E9 are adequate for representing tone contours, by describing an existing implementation that is capable of presenting contour tone glyphs that are mapped from UCS tone letter sequences.

Existing implementation: SIL *Graphite* system and IPA fonts

SIL International has for several years distributed IPA fonts from our web site. These fonts were designed using to work with older technologies that assumed one-to-one character- to-glyph mappings (following the “coded font” model of TR 15285) and used a proprietary encoding. We are currently engaged in a new implementation of IPA fonts that conforms to the UCS encoding and that makes use of complex character-to-glyph mapping technologies (following the “intelligent font” model of TR 15285).

In our initial implementation, complex character-to-glyph mapping has been implemented using the SIL Graphite font rendering system. The current implementation of this system provides a generic character-to-glyph mapping engine that is packaged as a COM object, which can be called by client application software. The engine makes use of finite-state tables found in a Graphite-enabled font. Such a font is created as an extension of the TrueType font format by adding the additional tables used by the Graphite engine. These additional tables define the script-specific mapping behaviour and are compiled from a high-level description of the script behaviour, written in the *Graphite Description Language (GDL)*. (Complete information on the Graphite rendering system is available at http://www.sil.org/computing/graphite/.) The GDL code used for handling the presentation of contour tones is provided below in Annex A. In this implementation, sequences of tone letter characters get mapped to single glyphs for the particular contour. This is exactly the same type of mapping that might be used for Arabic ligatures or Devanagari conjuncts.

Figure 1 shows a screen shot from an application using the Graphite rendering system. It contains two lines of text. Both lines of text contain exactly the same tone letter characters from the range 02E5..02E9 and encoded in UTF-16. The only difference is that, in the first line, the tone letters are separated by space characters to prevent the selection of the ligated contour glyphs.



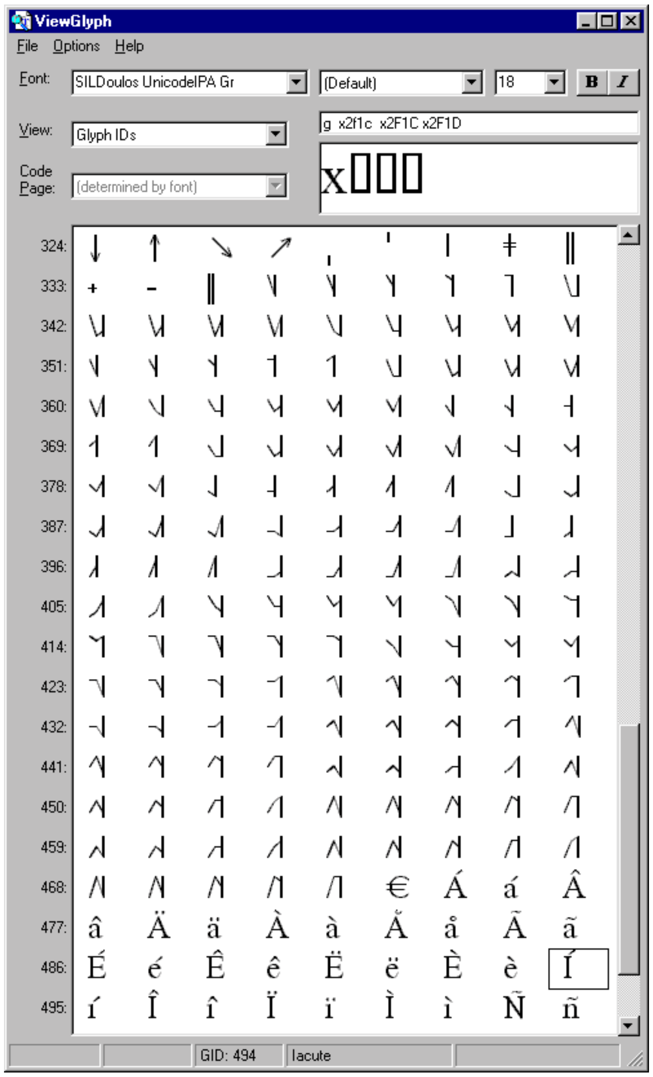
˦ ˥ ˥ ˩ ˦ ˨ ˥

˦˥ ˥˩ ˦˨˥

Figure 1. Tone letter sequences rendered as contour tone glyphs.

Note that contiguous sequences of tone letter characters are mapped to the appropriate tone contour glyphs. This is done without requiring any specific user action and without any additional markup or other information beyond the plain text.

This font implementation will support contours corresponding to any combination of tone letters up to three characters long. Figure 2 shows a screen shot of a utility that can view the glyph palette of a TrueType font. In this screen shot, the complete inventory of contour glyphs available in the IPA font is shown.



Font: SILDoulos UnicodeIPA Gr (Default) 18

View: Glyph IDs

Code Page: (determined by font)

g x2f1c x2F1C x2F1D

GID: 494 Iacute

Figure 2: inventory of contour tone glyphs in SIL IPA font

All of these tone glyphs can be presented using the Graphite system from an underlying encoded character representation using only the tone letter characters 02E5..02E9.

Note that Graphite is not the only system that can support a font implementation to handle presentation of contour tones. Apple Computer's *AAT* technology and Microsoft / Adobe's *OpenType* technology are equally capable of supporting similar implementations. We are considering the possibility of implementing our IPA fonts using these technologies in addition to the Graphite rendering technology in the future.

Recommendations

The existence of a working implementation demonstrates that the existing tone letter characters 02E5..02E9 are adequate for encoded representation and presentation of contour tones. It is recommended that presentation forms for contour tone letters not be added to the UCS. Rather, the needs of presentation of contour tone letters, and of many non-Latin scripts, would be better served by efforts to promote the development and adoption of technologies such as Graphite, AAT and OpenType that implement the “intelligent font” model described in TR 15285.

Annex A: GDL code for presentation of contour tones

The GDL code (pre-release version) for handling the presentation of contour tones is presented here. There are two parts to the code: a glyph table, which is used to define symbols for refering to individual glyphs or classes of glyphs, and a substitution table, which provides the actual mapping. (Note that an initial mapping from character codes to intial glyph identifiers is provided by the “cmap” table in the font. The GDL mappings are done entirely in terms of glyphs.)

Only the classes defined in the glyph table have been duplicated here. For every symbolic glyph name used in defining the classes, a definition of that name would also have been provided in the glyph table. For example, the symbolic name `g02E802E9` is associated with a particular glyph from the font using the following GDL statement:

```
g02E802E9 = glyphid(380);
```

(A *glyph ID* is a unique 16-bit integer identifier used in the TrueType font format to reference glyphs in a font’s glyph palette.) The complete set of such rules has been suppressed here to conserve space. Only the definitions of glyph classes are shown here.

The basic effect of a substitution rule such as

```
cPitches g02E9 > _ cPitch0$1:(1 2) / _ ^ _;
```

is this: whenever a glyph in the class `cPitches` is followed by the glyph `g02E9` (which happens to be the default glyph mapped in the cmap table from the character U+02E9), this sequence of two glyphs is substituted by a glyph from the class `cPitch0`. The operator `$1` specifies that the glyph to be selected from `cPitch0` is determined by the index that was used for matching the glyph in the first position of the sequence. (More generally, `$`*`n`* would use the index applied to the *nth* position in the glyph sequence.) Thus, for example, if the glyph in the first position of the sequence matched the fifth glyph in the class `cPitches`, then the output glyph would be the fifth glyph in `cPitch0`.

The remaining aspects of this substitution rule and the complete details on the GDL language are available at http://www.sil.org/computing/graphite/GDL.pdf.

Rules related to tone contours, excerpted from SILDU.GDL:

```
table(glyph) {MUnits = 1000};
//pitch glyph classes
cPitch0 = (g02E9, g02E902E802E9, g02E902E702E9, g02E902E602E9,
    g02E902E502E9, g02E802E9, g02E802E902E9, g02E802E702E9,
    g02E802E602E9, g02E802E502E9, g02E702E9, g02E702E902E9,
    g02E702E9, g02E702E602E9, g02E702E502E9, g02E602E9,
    g02E602E902E9, g02E602E802E9, g02E602E702E9, g02E602E502E9,
    g02E502E9, g02E502E902E9, g02E502E802E9, g02E502E9,
    g02E502E602E9);

cPitch1 = (g02E902E8, g02E902E802E8, g02E902E702E8, g02E902E602E8,
    g02E902E502E8, g02E8, g02E802E902E8, g02E802E702E8,
    g02E802E602E8, g02E802E502E8, g02E702E8, g02E702E902E8,
    g02E702E802E8, g02E702E602E8, g02E702E502E8, g02E602E8,
    g02E602E902E8, g02E602E802E8, g02E602E8, g02E602E502E8,
    g02E502E8, g02E502E902E8, g02E502E802E8, g02E502E702E8,
    g02E502E602E8);

cPitch2 = (g02E902E7, g02E902E7, g02E902E702E7, g02E902E602E7,
    g02E902E502E7, g02E802E7, g02E802E902E7, g02E802E702E7,
    g02E802E602E7, g02E802E502E7, g02E7, g02E702E902E7,
    g02E702E802E7, g02E702E602E7, g02E702E502E7, g02E602E7,
    g02E602E902E7, g02E602E802E7, g02E602E702E7, g02E602E502E7,
    g02E502E7, g02E502E902E7, g02E502E802E7, g02E502E702E7,
    g02E502E7);

cPitch3 = (g02E902E6, g02E902E802E6, g02E902E702E6, g02E902E602E6,
    g02E902E502E6, g02E802E6, g02E802E902E6, g02E802E6,
    g02E802E602E6, g02E802E502E6, g02E702E6, g02E702E902E6,
    g02E702E802E6, g02E702E602E6, g02E702E502E6, g02E6,
    g02E602E902E6, g02E602E802E6, g02E602E702E6, g02E602E502E6,
    g02E502E6, g02E502E902E6, g02E502E802E6, g02E502E702E6,
    g02E502E602E6);

cPitch4 = (g02E902E5, g02E902E802E5, g02E902E5, g02E902E602E5,
    g02E902E502E5, g02E802E5, g02E802E902E5, g02E802E702E5,
    g02E802E602E5, g02E802E502E5, g02E702E5, g02E702E902E5,
    g02E702E802E5, g02E702E5, g02E702E502E5, g02E602E5,
    g02E602E902E5, g02E602E802E5, g02E602E702E5, g02E602E502E5,
    g02E5, g02E502E902E5, g02E502E802E5, g02E502E702E5,
    g02E502E602E5);

cPitches = (g02E9, g02E902E8, g02E902E7, g02E902E6,
    g02E902E5, g02E8, g02E802E9, g02E802E7,
    g02E802E6, g02E802E5, g02E7, g02E702E9,
    g02E702E8, g02E702E6, g02E702E5, g02E6,
    g02E602E9, g02E602E8, g02E602E7, g02E602E5,
    g02E5, g02E502E9, g02E502E8, g02E502E7,
    g02E502E6);
endtable

table(substitution);
// Handle pitch character ligating
cPitches g02E9 > _ cPitch0$1:(1 2) / _ ^ _;
cPitches g02E8 > _ cPitch1$1:(1 2) / _ ^ _;
cPitches g02E7 > _ cPitch2$1:(1 2) / _ ^ _;
cPitches g02E6 > _ cPitch3$1:(1 2) / _ ^ _;
cPitches g02E5 > _ cPitch4$1:(1 2) / _ ^ _;

endtable;
```