New Technical Notes

Macintosh

Developer Support

Modifying the Standard String Comparison

Text **M.TE.NewStringComp**

Revised by: March 1988
Written by: Mark Davis
Priscilla Oppenheimer November 1987

This technical note describes how to modify the standard string comparison by constructing an `itl2` resource. Developers may want to modify the standard string comparison if Apple's comparison doesn't meet their needs or if Apple has not written a string comparison routine for the language that concerns them.

General Structure

The `itl2` resource contains a number of procedures that are used for accurate comparison of text by the International Utilities Package. Refer to *Inside Macintosh*, volume V for an explanation of the algorithm used. The default `itl2` for standard English text, which does no special processing, has the following form:

```
	; normal Include/Load statements
	Include 'hd:mpw:aincludes:ScriptEqu.a'
	Print  On,NoMDir

	String AsIs

;----------------------------------------------------------------------
;	dispatch table at the front of the code.
;----------------------------------------------------------------------
Intl1		Proc
	With	IUSortFrame,IUStrData
HookDispatch
	dc.w	ReturnEQ-HookDispatch	; InitProc = 0
	dc.w	ReturnEQ-HookDispatch	; FetchHook = 2
	dc.w	ReturnEQ-HookDispatch	; VernierHook = 4
	dc.w	ReturnEQ-HookDispatch	; ProjectHook = 6
	dc.w	ReturnEQ-HookDispatch	; ReservedHook1 = 8
	dc.w	ReturnEQ-HookDispatch	; ReservedHook2 = 10
```

```
  ;----------------------------------------------------------------
; Some common exit points
;----------------------------------------------------------------------------
ReturnNE
       tst.w  MinusOne            ; set cc NE
       rts
ReturnEQ
       cmp.w  d0,d0         ; set cc EQ
       rts
;----------------------------------------------------------------------------
       EndWith
       EndWith
       End
```

If modifications need to be made to the comparison process, then one or more of the dispatches will be modified to point to different routines:

```
dc.w   InitProc-HookDispatch       ; InitProc = 0
dc.w   FetchProc-HookDispatch      ; FetchHook = 2
dc.w   VernierProc-HookDispatch    ; VernierHook = 4
dc.w   ProjectProc-HookDispatch    ; ProjectHook = 6
```

There are a number of different changes that can be made to the comparison routines. Some of the common modifications include:

1. Comparing two bytes as one character
   Yugoslavian “l” < “lj” < “m”; Japanese… [InitProc, FetchProc]
2. Comparing characters in different order
   Norwegian “z” < “å” [ProjectProc]
3. Comparing one character as two
   German “ä” ≈ “ae” [ProjectProc]
4. Ignoring characters unless strings are otherwise equal:
   “blackbird” < “black-bird” < “blackbirds” [ProjectProc]
5. Changing the secondary ordering
   Bibliographic “a” < “A” [VernierProc]

The comparison hook procedures are all assembly language based, with arguments described below. Since the routines may be called once per character in both strings, the routines should be as fast as possible.

The condition codes are used to return information about the status of the hook routine. Typically the normal processing of characters will be skipped if the `CCR` is set to `NE`, so the default return should always have `EQ` set. Each of these routines has access to the stack frame `(A6)` used in the comparison routine, which has the following form:

```
IUSortFrame   Record {oldA6},Decrement
result ds.w   1
argTop equ    *
aStrText      ds.l   1
bStrText      ds.l   1
aStrLen       ds.w   1
bStrLen       ds.w   1
argSize       equ    argTop-*
return ds.l   1
oldA6         ds.l   1
aInfo         ds     IUStrData
bInfo         ds     IUStrData
wantMag       ds.b   1      ; 1-MagStrig 0-MagIdString.
```

```
weakEq ds.b   1            ; Signals at most weak equality
msLock ds.b   1            ; high byte of master ptr.
weakMag       ds.b   1     ; -1 weak, 1 strong compare
supStorage    ds.b   18    ; extra storage.
localSize     equ    *     ; frame size.
              EndR
```

There are three fields in this frame that are of interest for altering text comparison. The `supStorage` field is an area reserved for use by the comparison hook procedures as they see fit. The `aInfo` and `bInfo` records contain information about the current byte positions in the two compared strings A and B, and information about the status of current characters in those string. The `IUStrData` record has the following form:

```
IUStrData     Record       0
curChar       ds.w         1      ; current character.
mapChar       ds.w         1      ; projected character.
decChar       ds.w         1      ; decision char for weak equality
bufChar       ds.b         1      ; buffer for expansion.
justAfter     ds.b         1      ; boolean for AE vs ligature-AE.
ignChar       ds.b         1      ; flag: ignore char.
noFetch       ds.b         1      ; flag: no fetch of next.
strCnt ds.w                1      ; length word.
strPtr ds.l                1      ; current ptr to string.
              EndR
```

The Init Procedure

The Init Procedure is used to initialize the comparison process. The main use for this procedure is for double-byte scripts. As an optimization, the International Utilities will perform an initial check on the two strings, comparing for simple byte-to-byte equality. Thus any common initial substrings are checked before the Init procedure is called. The string pointers and lengths in the `IUStrData` records have been updated to point just past the common substrings.

Languages such as Japanese or Yugoslavian, which may consider two bytes to be one character, may have to back up one byte, as shown below.

```
;---------------------------------------------------------------------------
; Routine     InitProc
; Input       A6          Local Frame
; Output      CCR         NE to skip entire sort (usually set EQ)
; Trashes     Standard regs:      A0/A1/D0-D2
; Function    Initialize any special international hooks.
;             Double-byte scripts must synchronize AInfo.StrPtr &
;                  BInfo.StrPtr here!
;---------------------------------------------------------------------------
; Note: this should also check for single-byte nigori or maru, as below
```

InitProc

```
      move.w AStrLen(a6), d0     ; A length
      sub.w  AInfo.StrCnt(a6),d0 ; see if its changed
      beq.s  @FixB               ; A is done if not
      sub.l  #2,sp               ; return param
      move.l AStrText(a6),-(sp)  ; textBuf
      move.w d0,-(sp)            ; textOffset
      _CharByte
      tst.w  (sp)+               ; on character boundary?
      ble.s  @FixB               ; yes, continue
      sub.l  #1,AInfo.StrPtr(A6) ; adjust pointer
      add.w  #1,AInfo.StrCnt(A6) ; adjust count
@FixB
      move.w BStrLen(a6), d0     ; B length
      sub.w  BInfo.StrCnt(a6),d0 ; see if its changed
      beq.s  Quit Init           ; B is done if not
      sub.l  #2,sp               ; return param
      move.l BStrText(a6), -(sp) ; textBuf
      move.w d0, -(sp)           ; textOffset
      _CharByte
      tst.w  (sp)+               ; on character boundary?
      ble.w  @QuitInit           ; yes, continue
      sub.l  #1,BInfo.StrPtr(A6) ; adjust pointer
      add.w  #1,BInfo.StrCnt(A6) ; adjust count
@QuitInit
      bra.s  ReturnEQ            ; return to the caller.
      EndWith
```

The Fetch Procedure

The Fetch Procedure is used to fetch a character from a string, updating the pointer and length to reflect the remainder of the string. For example, the following code changes the text comparison for Yugoslavian:

```
 ;-----------------------------------------------------------------------
 ; Routine          FetchProc
 ; Input            A2          String Data Structure
 ;                  A3          String pointer (one past fetched char)
 ;                  A6          Local Frame
 ;                  D4.W        Character: top byte is fetched character, bottom
 ;                               is zero
 ;                  D5.B        1 if string is empty, otherwise 0
 ; Output           D4.W        Character: top byte set to character, bottom to
 ;                              extension
 ;                  D5.B        1 if string is empty, otherwise 0
 ; Trashes          Standard regs:              A0/A1/D0-D2
 ; Function         This routine returns the characters that are fetched from
 ;                  the string, if they are not just a sequence of single bytes.
 ;-----------------------------------------------------------------------

FetchProc
      tst.b  d5                  ; more characters in string?
      bne.s  ReturnEq            ; no -> bail out.

      move.w d4,d0               ; load high byte.
      move.b (a3),d0             ; load low byte.

      lea    pairTable,a1        ; load table address

@compareChar
```

```
        move.w (a1)+,d1             ; pair = 0?
        beq.s  ReturnEq             ; yes -> end of table.
        cmp.w  d0,d1                ; legal character pair?
        bne.s  @compareChar         ; no -> try the next one.
        add.w  #1,a3                ; increment pointer.
        sub.w  #1,StrCnt(a2)        ; decrement length.
        addx.w d5,d5                ; empty -> set the flag.
        move.w d0,d4                ; copy character pair.
        rts                         ; return to caller with CCR=NE

pairTable
        dc.b   'Lj'                 ; Lj
        dc.b   'LJ'                 ; LJ
        dc.b   'lJ'                 ; lJ
        dc.b   'lj'                 ; lj

        dc.b   'Nj'                 ; Nj
        dc.b   'NJ'                 ; NJ
        dc.b   'nJ'                 ; nJ
        dc.b   'nj'                 ; nj

        dc.b   'D', $be             ; Dz-hat
        dc.b   'D', $ae             ; DZ-hat
        dc.b   'd', $ae             ; dZ-hat
        dc.b   'd', $be             ; dz-hat

        DC.B   $00, $00             ; table end
```

The same sort of procedure is used for Japanese or other double-byte scripts, in order to combine two bytes into a single character for comparison.

```
FetchProc
        with          IUStrData
        tst.b         d5                ; empty string?
        bne.s         ReturnEq          ; exit if length = 0

; if we have a double-byte char, add the second byte
        lea           CurChar(a2),a0    ; pass pointer
        move.w        d4,(a0)           ; set value at ptr
        clr.w         d0                ; pass length

        sub.l         #2,SP             ; allocate return
        move.l        a0,-(sp)          ; pointer
        move.w        d0,-(sp)          ; offset
        _CharByte
        tst.w  (sp)+                ; test return
        bmi.s  @DoubleByte          ; skip if high byte (first two)

; we don't have a double byte, but two special cases combine second bytes
        move.b (a3),d0                  ; get next byte
        cmp.b  #$DE,d0                  ; nigori?
        beq.s  @DoubleByte              ; add in
        cmp.b  #$DF,d0                  ; maru?
        bne.s  ReturnEq                 ; exit: single byte

@DoubleByte
        move.b (a3)+,d4                 ; get next byte
        subq.w #1,StrCnt(A2)            ; dec string length
        addx.w d5,d5                    ; set x=1 if string len = 0
        rts                             ; return to caller with CCR=NE
```

The Project Procedure

The Project Procedure is used to find the primary ordering for a character. This routine will map characters that differ only in the secondary ordering onto a single character, typically the unmodified, uppercase character. For example, the following changes the comparison order for some Norwegian characters, so that they occur after ‘Z.’

```
;---------------------------------------------------------------------
; Routine     ProjectProc
; Input       A2          String Data Structure
;             D4.W        Character (top byte is char, bottom is extension
;                         (the extension is zero unless set by FetchProc))
; Output      D4.W        Projected Character
;             CCR         NE to skip normal Project
; Trashes     Standard regs:      A0/A1/D0-D2
; Function    This routine projects the secondary characters onto primary
;                  characters.
;             Example: a,ä,Ä -> A
;---------------------------------------------------------------------

ProjectProc
       lea    ProjTable,A1  ; load table address.
@findChar
       move.l (a1)+,D0             ; get entry
       cmp.w  d0,d4                ; original ≥ entry?
       bhi.s  @findChar            ; no, try the next entry.
       bne.s  ReturnEq             ; not equal, process normally

@replaceChar
       swap   d0                   ; get replacement
       move.w d0,d4                ; set new character word.
@doneChar
       rts                         ; CCR is NE to skip project.

ProjTable
;      Table contains entries of the form  r1, r2, o1, o2,
;      where r1,r2 are the replacement word, and
;      o1, o2 are the original character.
;      The entries are sorted by o1,o2 for use in the above algorithm

       DC.B   'Z', 3, 'Å', 0       ; Å after Ø
       DC.B   'Z', 3, 'å', 0       ; å after Ø
       DC.B   'Z', 1, 'Æ', 0       ; Æ after Z
       DC.B   'Z', 2, 'Ø', 0       ; Ø after Æ
       DC.B   'Z', 1, 'æ', 0       ; æ after Z
       DC.B   'Z', 2, 'ø', 0       ; ø after Æ
       DC.L   $FFFFFFFF            ; table end
```

The Project procedure can also be used to undo the effects of the normal projection. For example, suppose that “œ” is not to be expanded into “oe”: in that case, a simple test can be made against 'œ',0, returning `NE` if there is a match, so that the normal processing is not done. To expand one character into two, the routine should return the first replacement character in `D4.W`, and modify two fields in the `IUStrData` field. For example, given that `A1` points to a table entry of the form (primaryCharacter: Word; secondaryCharacters: Word), the following code could be used:

```
       …
       move.w (a1)+,d4             ; return first, primary character
       move.w (a1)+,CurChar(A2)    ; original => first, modified char.
       addq.b #1,JustAfter(A2)     ; set to one (otherwise zero)
       move.b (a1),BufChar(A2)     ; store second character (BYTE!)
       …
```

`CurChar` is where the original character returned by `FetchChar` is stored. If characters are different even after being projected onto their respective primary characters, then the `CurChar` values for each string will be compared. `JustAfter` indicates that the expanded character should sort after the corresponding unexpanded form. This field must be set whenever `CurChar` is modified in order for the comparison to be fully ordered. `BufChar` stores the next byte to be retrieved from the string by `FetchChar`.

To handle the case where characters are ignored unless the two compared strings are otherwise equal, the `IgnChar` flag can be set. This can be used to handle characters such as the hyphen in English, or vowels in Arabic.

```
…
cmp.w  #hyphen,d0          ; is it a ignorable?
seq    IgnChar(a2)         ; set whether or not
…
```

The Vernier Procedure

The Vernier Procedure is used to make a final comparison among characters that have the same primary ordering. It is only needed if the `CurChar` values are not ordered properly. For example, according to the binary encoding, å < Ã. To change this ordering so that uppercase letters are before lowercase letters, Ã is mapped to $7F in normal comparison. Notice that only the characters in the secondary ordering are affected: Ã can be mapped onto Z, but not onto Ä, since that would cause a collision.

```
;---------------------------------------------------------------------------
; Routine     VernierProc
; Input       D4.B                High byte of character
;             D5.B           Low byte of character
; Output      D4.B           High byte of character
;             D5.B           Low byte of character
;             CCR            NE if to skip standard Vernier
; Trashes     Standard regs:      A0/A1/D0-D2
; Function    The Vernier routine compares characters within the secondary
;                    ordering if two strings are otherwise equal.
;                    Example: (a,A,Ä,ä)
;---------------------------------------------------------------------------

VernierProc
       not.b        d4            ; invert secondary ordering
       not.b        d5            ; ditto for lower byte
       bra.s        ReturnEq      ; normal processing afterwards
```

Installing an `itl2` resource

To write an `itl2` resource, follow the guidelines in M.PT.StandAloneCode for writing standalone code in MPW. The code should be written in assembly language, and it must follow the specifications given in this technical note or serious system errors could occur whenever string comparisons are made.

The default comparison routine is in the `itl2` resource of the System file. In order to use a comparison routine other than the standard one, you should include an `itl2` resource in your application with the same name and resource ID as the one in the System file that you wish to change. The Resource Manager will look for the resource in the application resource file before

it looks in the System resource file, so your string comparison routine will be used instead of the default one.

It is generally a dangerous practice to change a system resource since other applications may depend on it, but if you have good reasons to permanently change the system `itl2` resource so that all applications use a different comparison routine, then you should write an installer script to change the `itl2` resource in the System resource file. Writing an installer script is documented in M.TP.Installer. You are required to write an installer script if you are planning to ship your application on a licensed system software disk and your application makes a permanent change to any resources in the System file. We strongly discourage changing the System `itl2` as that would change the behavior of string comparison and sorting for all applications. If that is your intent, then you should write an installer script. However, if you are changing the `itl2` resource in the System file for academic or internal use, then you can use a resource editor such as ResEdit to copy your `itl2` resource into the System file.

Further Reference:

- The International Utilities
- M.TP.Installer
- M.PT.StandAloneCode