A.10 The optional Facility word set

A.10.6 Glossary

A.10.6.1.0742 AT-XY

Most implementors supply a method of positioning a cursor on a CRT screen, but there is great variance in names and stack arguments. This version is supported by at least one major vendor.

A.10.6.1.1755 KEY?

The Technical Committee has gone around several times on the stack effects. Whatever is decided will violate somebody's practice and penalize some machine. This way doesn't interfere with type-ahead on some systems, while requiring the implementation of a single-character buffer on machines where polling the keyboard inevitably results in the destruction of the character.

Use of KEY or KEY? indicates that the application does not wish to bother with non-character events, so they are discarded, in anticipation of eventually receiving a valid character. Applications wishing to handle non-character events must use EKEY and EKEY?. It is possible to mix uses of KEY? / KEY and EKEY? / EKEY within a single application, but the application must use KEY? and KEY only when it wishes to discard non-character events until a valid character is received.

A.10.6.2.1305 EKEY

EKEY provides a standard word to access a system-dependent set of **raw** keyboard events, including events corresponding to members of the standard character set, events corresponding to other members of the implementation-defined character set, and keystrokes that do not correspond to members of the character set.

EKEY assumes no particular numerical correspondence between particular event code values and the values representing standard characters. On some systems, this may allow two separate keys that correspond to the same standard character to be distinguished from one another.

In systems that combine both keyboard and mouse events into a single **event stream**, the single number returned by EKEY may be inadequate to represent the full range of input possibilities. In such systems, a single **event record** may include a time stamp, the x,y coordinates of the mouse position, the keyboard state, and the state of the mouse buttons. In such systems, it might be appropriate for EKEY to return the address of an **event record** from which the other information could be extracted.

Also, consider a hypothetical Forth system running under MS-DOS on a PC-compatible computer. Assume that the implementation-defined character set is the **normal** 8-bit PC character set. In that character set, the codes from 0 to 127 correspond to ASCII characters. The codes from 128 to 255 represent characters from various non-English languages, mathematical symbols, and some graphical symbols used for line drawing. In addition to those characters, the keyboard can generate various other **scan codes**, representing such non-character events as arrow keys and function keys.

There may be multiple keys, with different scan codes, corresponding to the same standard character. For example, the character representing the number **1** often appears both in the row of number keys above the alphabetic keys, and also in the separate numeric keypad.

When a program asks the MS-DOS operating system for a keyboard event, it receives either a single non-zero byte, representing a character, or a zero byte followed by a **scan code** byte, representing a non-character keyboard event (e.g., a function key).

EKEY represents each keyboard event as a single number, rather than as a sequence of numbers. For the system described above, the following would be a reasonable implementation of EKEY and related words:

The MAX-CHAR environmental query would return 255.

Assume the existence of a word DOS-KEY (-- char) which executes the MS-DOS **Direct STDIN Input** system call (Interrupt 21h, Function 07h) and a word DOS-KEY? (-- flag) which executes the MS-DOS **Check STDIN Status** system call (Interrupt 21h, Function 0Bh).

```
: EKEY?  ( -- flag )  DOS-KEY?  0<>  ;

: EKEY  ( -- u )  DOS-KEY  ?DUP 0= IF  DOS-KEY 256 +  THEN ;

: EKEY>CHAR  ( u -- u false | char true )
    DUP 255 > IF          ( u )
    DUP 259 = IF           \ 259 is Ctrl-@ (ASCII NUL)
        DROP 0 TRUE EXIT   \ so replace with character
      THEN FALSE EXIT      \ otherwise extended character
    THEN  TRUE             \ normal extended ASCII char.
;

VARIABLE PENDING-CHAR   -1 PENDING-CHAR !

: KEY?  ( -- flag )
    PENDING-CHAR @ 0< IF
      BEGIN  EKEY? WHILE
        EKEY EKEY>CHAR IF
          PENDING-CHAR !  TRUE EXIT
        THEN DROP
      REPEAT  FALSE EXIT
    THEN  TRUE
;

: KEY  ( -- char )
    PENDING-CHAR @ 0< IF
      BEGIN  EKEY  EKEY>CHAR 0=
      WHILE
        DROP
      REPEAT  EXIT
    THEN  PENDING-CHAR @  -1 PENDING-CHAR !
;
```

This is a full-featured implementation, providing the application program with an easy way to either handle non-character events (with EKEY), or to ignore them and to only consider **real** characters (with KEY).

Note that EKEY maps scan codes from 0 to 255 into numbers from 256 to 511. EKEY maps the number 259, representing the keyboard combination Ctrl-Shift-@, to the character whose numerical value is 0 (ASCII NUL). Many ASCII keyboards generate ASCII NUL for Ctrl-Shift-@, so we use that key combination for ASCII NUL (which is otherwise unavailable from MS-DOS, because the zero byte signifies that another scan-code byte follows).

One consequence of using the **Direct STDIN Input** system call (function 7) instead of the **STDIN Input** system call (function 8) is that the normal DOS **Ctrl-C interrupt** behavior is disabled when the system is waiting for input (Ctrl-C would still cause an interrupt while characters are being output). On the other hand, if the **STDIN Input** system call (function 8) were used to implement EKEY, Ctrl-C interrupts would be enabled, but Ctrl-Shift-@ would also cause an interrupt, because the operating system would treat the second byte of the 0,3 sequence as a Ctrl-C, even though the 3 is really a scan code and not a character. One **best of both worlds** solution is to use function 8 for the first byte received by EKEY, and function 7 for the scan code byte. For example:

```
: EKEY  ( -- u )
    DOS-KEY-FUNCTION-8  ?DUP  0=  IF
      DOS-KEY-FUNCTION-7 DUP 3  =  IF
        DROP 0  ELSE  256 +
      THEN
    THEN
;
```

Of course, if the Forth implementor chooses to pass Ctrl-C through to the program, without using it for its usual interrupt function, then DOS function 7 is appropriate in both cases (and some additional care must be taken to prevent a typed-ahead Ctrl-C from interrupting the Forth system during output operations).

A Forth system might also choose a simpler implementation of KEY, without implementing EKEY, as follows:

```
: KEY   ( -- char )  DOS-KEY  ;
: KEY?  ( -- flag )  DOS-KEY? 0<>  ;
```

The disadvantages of the simpler version are:

a) An application program that uses KEY, expecting to receive only valid characters, might receive a sequence of bytes (e.g., a zero byte followed by a byte with the same numerical value as the letter **A**) that appears to contain a valid character, even though the user pressed a key (e.g., function key 4) that does not correspond to any valid character.

b) An application program that wishes to handle non-character events will have to execute KEY twice if it returns zero the first time. This might appear to be a reasonable and easy thing to do. However, such code is not portable to other systems that do not use a zero byte as an **escape** code. Using the EKEY approach, the algorithm for handling keyboard events can be the same for all systems; the system dependencies can be reduced to a table or set of constants listing the system-dependent key codes used to access particular application functions. Without EKEY, the algorithm, not just the table, is likely to be system dependent.

Another approach to EKEY on MS-DOS is to use the BIOS **Read Keyboard Status** function (Interrupt 16h, Function 01h) or the related **Check Keyboard** function (Interrupt 16h, Function 11h). The advantage of this function is that it allows the program to distinguish between different keys that correspond to the same character (e.g. the two **1** keys). The disadvantage is that the BIOS keyboard functions read only the keyboard. They cannot be **redirected** to another **standard input** source, as can the DOS STDIN Input functions.

A.10.6.2.1306 EKEY>CHAR

EKEY>CHAR translates a keyboard event into the corresponding member of the character set, if such a correspondence exists for that event.

It is possible that several different keyboard events may correspond to the same character, and other keyboard events may correspond to no character.

A.10.6.2.1325 EMIT?

An indefinite delay is a device related condition, such as printer off-line, that requires operator intervention before the device will accept new data.

A.10.6.2.1905 MS

Although their frequencies vary, every system has a clock. Since many programs need to time intervals, this word is offered. Use of milliseconds as an internal unit of time is a practical **least common denominator** external unit. It is assumed implementors will use **clock ticks** (whatever size they are) as an internal unit and convert as appropriate.

A.10.6.2.2292 TIME&DATE

Most systems have a real-time clock/calendar. This word gives portable access to it.

Table of Contents
Next Section