# Common Subroutines

OPD/SPEC/1

# 0 DOCUMENT CONTROL

## 0.1 Contents



## 0.2 Changes Since Previous Issue

Minor extension to Field Editor.

## 0.3 Changes Forecast

None.

# 1 GENERAL

This document provides a specification of the functions performed by and the programming interfaces to the Common Subroutines component of the Base Functional Software.

The subroutines are not connected with each other. Each is allocated a separate section to itself in this document. A section generally contains an informal description and explanation of the subroutine and a more formal programming interface specification.

The subroutines which form the subsystem are:

a) Text Expander

b) Field Editor

c) Waiter Subsystem

## 1.1 TRAP entry

The Director trap is used for more than entries to Application Handler itself. The range of possible action values is partitioned and a sub-range is allocated to each subsystem which wishes to participate. Thus, the Common Subroutines subsystem has its own sub-range of action values, so that when the Director trap is entered with an action value within this sub-range, control is actually passed to the subsystem (to CS_ROOT, in fact). To further the illusion, a number of synonyms are available for the Director trap value - in this case T.CMNSUBS. The trap value is the same, the entry is still into Director, only the name has been changed.

The mode of entry provides certain facilities (storage of registers) but requires that 128 bytes are available on the caller's stack.

## 1.2 CS ROOT - The Header Module

This is a component of the Common Subroutines subsystem which must be present. It has no direct function as far as application code is concerned. It provides the subsystem program header which is required by Application Handler. It also receives control from Application Handler when an activity uses the trap T.CMNSUBS. It then examines the action value in D0 and branches to the required common subroutine.

It is recommended that CS_ROOT and the rest of the common subroutines should be placed in a contiguous block or in proximity to each other when a system is linked, since the branches from CS_ROOT use 16 bit displacement operand forms.

# 2 SUBROUTINE SPECIFICATIONS

## 2.1 Text Expander

### 2.1.1 PURPOSE

The purpose of Text Expander is to permit a reduction to be made in the overall ROM size requirement for OPD software by providing an organised means for the expansion of common substrings within text messages output by the software. Thus, if a word or phrase occurs several times in the totality of text messages stored by the OPD software then it may be extracted and stored once, in a special libarary, and each instance of it in a text message replaced by an economically encoded reference to the stored form. Text Expander provides the means by which messages containing such 'shorthand' references to externally held words or phrases are reconstructed before they are displayed.

Note that Text Expander does not provide a facility for the recognition of common substrings in a population of text messages, nor for the assembly of the special library.

### 2.1.2 DESCRIPTION

A population of text messages is assumed by Text Expander to be collected into a single structure, called a Main text library. A population of extracted letters, words or longer strings of characters is also collected into a single structure, called a Common text library.

It is conceivable that several Main text libraries may exist, associated with different parts of the software, and that these may reference one or more than one Common text library. The identities of the two libraries are therefore given as parameters on a call to Text Expander.

The basic action of Text Expander is to process a text message selected by the caller from the Main text library a character at a time into a store buffer provided by the caller, replacing any shorthand references as it goes by the appropriate character strings retrieved from the Common text library.

Two further features are included, to increase its usefulness.

a) It is possible to 'envelope' a number of characters in either library, so that they are sent through to the output buffer transparently, i.e. the characters are not examined to see if they are 'shorthand' references.

b) It is possible to mark sections of text in either library as candidates for selective omission during the copying and expansion process. The omission or the inclusion of such a marked section may be achieved at will by the program which calls Text Expander. This facility is called 'variants' in

the interface specification.

A Common text library may itself contain 'shorthand' references. The depth of such nesting (while expected to be small in practice) is limited only by the amount of free space on the caller's stack.

The segment or segments containing the buffer area, the Main text library and the Common text library may be either of normal or cell allocator type, but must be frozen or immobile on entry to the subroutine. They are not thawed by the subroutine.

### 2.1.3 INTERFACE

#### 2.1.3.1 Call and Register

Trap Name: T.CMNSUBS
Action Value (D0.B): CS.TEXT

#### 2.1.3.2 Additional Call Parameters

D1.W Text number - the number in the Main text library of the text which is to be expanded.

D2.L Length of supplied buffer, in bytes.

D3.B Variant number - can be undefined if the stored text does not have variants; otherwise bits are set (usually only one) to specify the variants to be included (least significant bit specifies variant 1).

A1 Address of buffer to receive the expanded text.

A2 Main text library address.

A3 Common text library address - can be undefined if the specified main text does not use common texts.

#### 2.1.3.3 Return Information

D0.L Return code:-

| | | |
|---|---|---|
| Non-negative | - | successful call |
| ERR.BO | - | buffer too small for expanded text |
| ERR.DI | - | data invalid (illegal escape sequence) |

D2.L Length of buffer actually used, in bytes

D1, D3 to D7, A0 to A7 are unchanged.

The specified text has been expanded into the supplied buffer.

### 2.1.4 Function

The function is to construct the complete text specified by the text number and variant number.

The specified text is copied to the buffer, acting on references to common texts and on variants and enveloped sequences. (See 'Supported codes' below.)

Any of the text may be wholly or partly composed of references to common texts which are automatically inserted at the current position in the buffer.

Any of the text may contain variant masks. If the result of ANDing the variant mask with the variant parameter to this procedure is zero, the following text (including any references to common texts) is skipped up to the next variant mask or the end of the text.

Any of the text may be 'enveloped' so that, within the envelope, reference to common texts is suspended, allowing codes above 127 to represent themselves. Code 127 can be represented by an adjacent pair of 127 codes.

In principle the routine does not validate the library structure or contents. Certain errors are detected and give rise to the returned status values.

The library structure assumed and the supported codes are specified below.

### 2.1.4.1 Supported codes

Main text and common texts may consist of the following codes:

| | |
|---|---|
| 0 to 31 | As supported by Kernel |
| 32 to 126 | Displayable characters, supported by Kernel |
| 127 | Used as the escape code for the following screen construction features: |

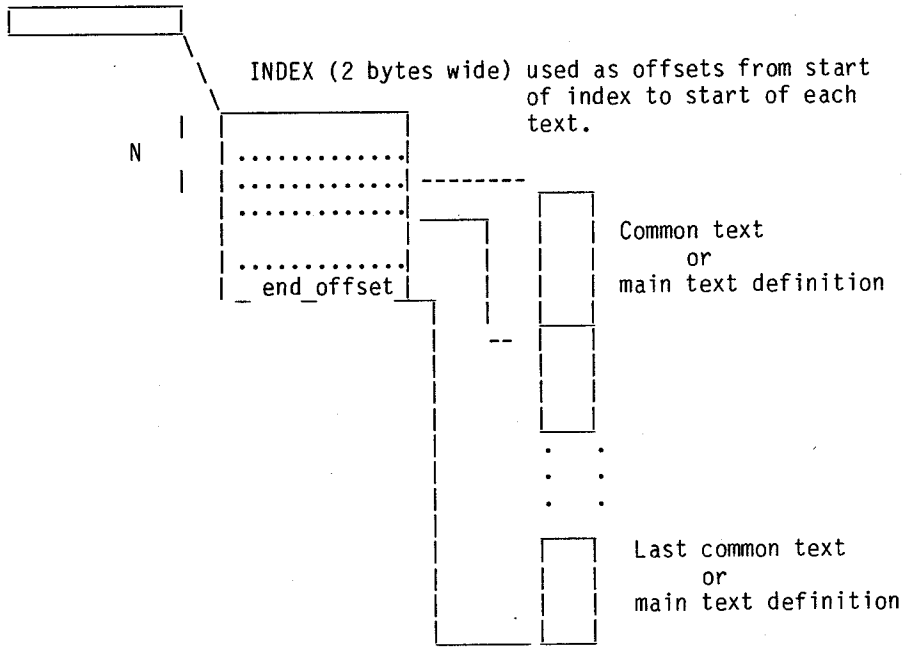
| Following code(s) | Effect |
|---|---|
| 1 , m | m is a variant mask, An all-ones mask restores unqualified output. |
| 2 ... ... 127 , 2 | Envelopes a sequence that includes codes, supported by Kernel, in the range 0 to 255. The code 127 may be represented in the sequence by a pair of contiguous 127 codes. Common text insertion is suspended during an enveloped sequence. The envelope sequence may also be terminated by the end of the text. |

| | |
|---|---|
| 128 to 255 | Refers to common text N in the Common text library, where N (0 to 127) is obtained by subtracting 128. |

### 2.1.4.2 Structure of Libraries

Main text libraries and Common text libraries have the same structure, as follows:

Base address (4 bytes)




Each library consists of main or common texts placed end to end in a contiguous region of store. There are no counts or terminators.

Main text and common text definitions are not mixed in the same library.

## 2.2 Field Editor

### 2.2.1 PURPOSE

The Field Editor is a subroutine which is intended to be called by user-visible applications for the purpose of handling keyboard input from the user when this is expected to be "data" rather than simply menu selection keystrokes. It is not itself user-visible in that it has no function or personality separate from that of the calling application.

The Field Editor exists to meet two objectives: to reduce the overall code space occupied by the software, and to promote uniformity of approach among applications to the process of displaying a modest quantity of data on the screen and allowing this to be altered or new data to be entered. A corollary to this second objective is that it should be possible to extend the interface seen by the user, in terms of permissible key depressions and the subsequent actions, in a compatible manner - i.e. no key actions carried out by the Field Editor need to be changed in an unnatural manner by an application offering an extended interface.

### 2.2.2 DESCRIPTION

The Field Editor handles text, i.e. a string of visible characters. No meaning is deduced or assigned to any groups of characters, e.g. it has no concept of words or sentences or numbers. A calling application, which is using Field Editor to perform the low level clerical chores may attach such a meaning and may steer Field Editor in an appropriate manner.

Text is always displayed on the screen within a rectangular box, i.e. m lines of n characters. This box may represent the entire field of data or it may represent a window into a larger collection of data. In the latter case the windowing functions, e.g. scrolling, would have to be managed by the calling application. Field Editor deals with a single box, or field, as a single data array containing, at most, as many characters as will fit into the box.

An application may display a screen containing several fields simultaneously, but needs to direct Field Editor to each field in turn if several are to be amended, since it can deal with only one at once.

A cursor is always displayed in the field in which Field Editor is currently engaged, to denote the position where keyed data characters will be placed. It takes the form of a reverse video block covering the character position concerned.

The user interface to Field Editor is given in the OPD Handbook, which describes which keys can validly be pressed and what their effect is.

The editor may be asked to validate keyed data characters against one of a selection of criteria and to generate an error tone and refuse to accept the character if it is invalid. The purpose of this is to allow keying errors to be rejected as early as possible, at the time of commission if feasible, to improve the user interface.

The editor has a concept of a number of existing characters in a field, which is the number of data characters entered into the field by the user from the keyboard, either directly or indirectly. It is specified by the parameter CS.FARRLEN. It may be nominated by the calling application and will often encompass the entire field. A data entry field starts, implicitly, with no existing data characters. The number of existing characters is updated by the editor as the user works within the field, taking account of text keys and control keys. On exit to the calling application it represents the right-most postition in the field ever reached by text characters or cursor movement, as modified by subsequent character deletions or insertions. It is provided for those applications that need to know exactly how much the user has typed and may be ignored on return by other applications. Character positions in the array which do not yet exist, according to the editor, are always returned as spaces.

### 2.2.3 USER INTERFACE

Field Editor does not (and could not) support all of the interface described below. It provides some of that support plus the facilities needed for a calling application to control its actions so that the application may provide the remainder of the required support in any particular case.

One of the components of the base functional software provides a uniform user interface for all instances of user data entry or amendment. This process is always presented to the user in a form-filling format (though the data entry area on the form is sometimes as small as one character). This component is called the Field Editor. The user is not aware of its existence as a separate entity, merely of the uniformity of approach which it confers.

Any data entry/amendment screen consists of a fixed display, of text and perhaps block shading, and a number of rectangular windows or entry fields. For a data entry screen the fields are initially blank or empty and for a data amendment screen the fields are filled in with the existing data values which the user is being invited to amend. A mixture of entry and amendment fields on one screen is also possible.

A field or window, is not necessarily as wide as the screen, e.g. it may be a rectangular block 10 characters wide and 3 characters deep, a total field size of 30 characters. It may be as small as one or two characters.

A visible cursor is displayed in one of the fields, which denotes the currently active field and the current character position within it. When the screen is first displayed, the cursor is normally at the top left of the 'first' field.

Data characters are entered simply by typing the character required on the keyboard. The character entered is placed in the current character position in the current field, as denoted by the cursor. The cursor is then moved one position to the right in the field. If it is already at the right hand end of the field, it is moved to the left-most character position in the next line down in the field. If the cursor is already in the right-most character position of the lowest line in the field (i.e. the last character position in the field) then the field is deemed complete. At this point, the calling application should intervene to move the cursor into the next field on the screen.

Data already entered can be overtyped by moving the cursor back if necessary and entering the new characters.

Validation of entered data characters by the Field Editor is possible. If the data character keyed by the user is invalid, an audible error tone is generated by the Field Editor, the character is not inserted into the field and the cursor does not move.

### 2.2.3.1 Control and Editing Keys

Several keys are also usable as control or editing keys. These are described below.

#### 2.2.3.1.1 Cursor Control Keys - | | - -

The cursor control keys are used, unshifted, to move the cursor within a single field (or window). The cursor moves one position in the direction indicated by the arrow for each key depression. The cursor may be moved fast by holding down the cursor control key if auto-repeat is in operation on the keyboard. If the cursor reaches the upper or lower edge of the window then it moves no further, though no error tone is generated. If the cursor reaches the left hand or right hand edge of the window then it moves to the end of the preceding line or to the start of the next line, respectively, unless it is already in the first or last character position of the field, in which case a field skip is indicated which should be performed by the calling application.

#### 2.2.3.1.2 RETURN

The RETURN key moves the cursor to the left hand end of the next line down in the field, or window (i.e. it performs CR/LF). Any character positions in the line which contains the cursor at the time RETURN is pressed and which are to the right of or under the

cursor but which do not yet exist are changed to existing characters (with values of 'space'). Existing characters are not changed in any way. RETURN operates only within the current field. If the cursor is in the last, or only, line of a field when RETURN is pressed, a field skip is indicated.

2.2.3.1.3 DELETE

Use of the DELETE key changes the character position before the cursor to a space, and moves the cursor one position to the left. If the cursor is already in the left most character position of a line then it is moved to the right-most character position of the next line above, unless it is already in the top line of the field, in which case the DELETE key has no effect. The key operates only within the current field.

2.2.3.1.4 INSERT - SHIFT/DELETE

Use of the INSERT key combination enters a space at the cursor position, after having first moved all data characters in the field under and to the right of the cursor by one position to the right. Characters moved off the right hand ends of lines appear at the left hand ends of the succeeding lines. The character moved off the end of the last line of the field is lost. The cursor is not moved.

2.2.3.1.5 REMOVE - CTRL/DELETE

REMOVE is the inverse of INSERT. Use of the REMOVE key combination removes the character under the cursor and then moves all data characters in the field to the right of the cursor by one character position left. Characters moved off the left of lines reappear at the right of the preceding lines. A space is moved into the last character of the field (right hand end of lowest line). The cursor is not moved.

2.2.3.1.6 Field Skip Forwards - TAB

This key is used to move the cursor from field to field. It moves the cursor, wherever it is in the current field, to the first character position (top left) of the next field on the screen. Use of the key combination while the cursor is in the last field on the screen moves the cursor to the first character position of the first field on the screen.

Note: This is not a Field Editor action, since it is aware of only one field. It must be implemented by the calling application.

2.2.3.1.7 Field Skip Backwards - SHIFT/TAB

The effect of this key combination depends on the current position of the cursor. If the cursor is not on the first character position of the field, then the key combination moves

the cursor to the first character position of the same field. If the cursor is already at the first character position of the field then the key combination moves the cursor to the first character position of the previous field, or of the last field on the screen if the current field is the first field on the screen.

Note: This is not a Field Editor action. It must be implemented by the calling application.

2.2.3.1.8 ENTER

The ENTER key combination is used to signal to the application that data entry or amendment is complete. The Field Editor returns control to the calling application.

2.2.3.1.9 ALT/RETURN

This key combination acts like the RETURN key except that character positions under and to the right of the cursor in the current line are always changed to spaces and denoted as existing whether or not they currently exist, i.e. the action is destructive in an existing field of data.

When a data character is entered into the last character position of a field, a field skip is indicated.

## 2.2.4 INTERFACE

Field Editor manipulates or services 7 entities, which are:

a) The visible form of the screen field or box.

b) The data array - an array of data characters supplied by the calling application or built up by Field Editor which constitutes the text currently occupying the field.

c) A field description - size of field and foreground/ background colours (but not position of field on the screen; this is a matter for the calling application).

d) Validation specification - how keyed characters are to be validated.

e) Termination specification - a number of conditions, occurrence of one of which is to cause a return to be made to the calling application.

f) Sound output - the audible tone generator, used to signify character validation failure.

g) Keyboard.

The calling application must have opened channels to the normal keyboard and to a screen window and it must have positioned the

window suitably on the screen.

The calling application may elect to:

a) Ask Field Editor to fill the screen field with data from the data array (the normal case for data amendment).

b) Supply an empty data array and ask the Field Editor to fill the screen field with spaces (in the field background colour - this is the normal case for data entry).

c) Write the field contents to the screen itself and instruct the Field Editor to leave the screen field alone until the user starts to use the keyboard. A data array is still supplied. If the number of valid characters in the array is less than the total number of characters in the array then the remaining characters in the array, beyond the end of the existing data, are spacefilled and their corresponding screen positions are blanked. (This mode is used for special purposes or when Field Editor is being re-entered frequently and continual rewriting of the screen field would not be economic.)

   On entry to Field Editor a distinction is made between blank fill mode (CS.FENMODE = 2 or 3, see section 2.2.4.3.4 for parameter listing) and the other modes.

   In blank fill mode the data array is entirely cleared to spaces and the screen field is likewise completely cleared to spaces (in the specified background colour) with the cursor displayed in the first character position of the 'non-protected' area of the field. In 'write current contents' mode (CS.FENMODE = 4 or 5) any characters at the end of the array beyond the limit of existing characters (as given in CS.FARRLEN) are spacefilled. The number of existing characters is not altered. The whole array is then written to the screen field thus blanking the part of the field beyond the end of the existing characters.

   In 'do not write array' mode (CS.FENMODE = 0) any characters at the end of the array beyond the limit of existing characters are spacefilled. The number of existing characters is not altered. None of the array is written to the screen field.

   The decision is then taken as to whether or not an immediate return to the calling application was requested.

If control remains with the Field Editor, the cursor is displayed at the character position nominated by the calling application. Control then remains in the Field Editor until the user keys a character which the calling application has designated as a terminator character, or until an unexpected event occurs (these are passed back to the calling application).

As the user keys in characters, both the displayed field on the screen and the data array are updated, following the rules in section 2.2.3. When the Field Editor returns control to the calling application, therefore, it returns the data array updated in situ to reflect the screen field contents at the time of the return. In general, when a terminator character is keyed by the user, neither the screen field nor the data array have been updated by it when the return is made (though there are exceptions, noted below).

If the calling application wishes the Field Editor to 'time-out' and return control to it in the event of user inactivity, then it should set the timer before entry to Field Editor and accept the return with an 'unexpected event'.

Field Editor is sharable. It can be used when the screen is in 40 column or 80 column mode and with a field which is in a window with the 'double width' property set in 80 column mode. It cannot cope with fields containing characters which are not displayable data characters (the latter are those in the range $20 to $7F inclusive, and $CA and $CB). In particular, it cannot cope with text containing TAB, CR or LF characters. Neither can it cope with text containing display attribute control characters. All the text characters are displayed with the uniform field foreground and background colours (though see the start offset parameter and note that the calling application may write displays of mixed 'colours', provided that the text in the data array does not actually contain the attribute control characters or sequences).

The calling application must also supply a local event number, in the usual form, which may be used by the Field Editor and which must not be in active use for any other purpose at the time that Field Editor is called.

The corresponding bit in the Event Request Register must be set by the calling application before Field Editor is entered and is left set in that register when control is returned by Field Editor. The event will never occur as a result of Field Editor's activities after it has returned control, even if the return of control was on account of an unexpected event.

Field Editor operates synchronously in the sense that a keyed character is read and then fully processed before the next keyed character is read. Other machine events cannot, therefore, leave Field Editor 'holding' several characters it has read from the keyboard but not yet actioned. Any such keystrokes remain queued in the keyboard channel.

When Field Editor exits, either because of detection of a user-specified condition, or because of an 'unexpected event', its I/O calls are always cancelled and are not left 'hanging about' in the calling activity.

When Field Editor exits, reporting an 'unexpected event', the event is still pending in the Event Notification Register for the activity, so that the calling application can process it normally.

The boundary handling attributes of the screen window are changed by Field Editor to progressive wrap at both edges but no auto-scrolling.

In order to accommodate what may be varying requirements between its calling applications, Field Editor's action on freezing and thawing the caller's data areas may be controlled by two parameter bits, see section 2.2.4.3.4.

Two data areas are involved, the data array and the parameter block, which are both supplied by the caller. They may be in the same segment as each other or in different segments.

They are both frozen while Field Editor is actually processing. A choice is allowed over whether they should be thawed when a return is made back to the caller. Another independent choice is allowed over whether they should be thawed while Field Editor is waiting for the user to press a key. It is clearly desirable that this second choice be exercised whenever possible to promote store management fluidity. It is assumed that the caller's code and stack segments are frozen or immobile (!).

### 2.2.4.3 FIELD EDITOR - formal interface

#### 2.2.4.3.1 Call and registers

Trap Name: T.CMNSUBS
Action Value (D0.B): CS.FIELD

#### 2.2.4.3.2 Additional Call Parameters

D2 : Bits 31 to 16 Offset of parameter block from start of its cell

Bits 15 to 0 Cell tag of cell containing parameter block

#### 2.2.4.3.3 Return Information

D1.L : Contains the character keyed if, and only if, the return was made because one of the calling application's termination conditions was met (i.e. a normal return).

The parameter block remains valid. In particular, the cursor position pointers in it may be regarded as response fields, defining the position of the cursor in the data array and within the screen box at the time of return.

The data array has been updated in the light of the user's editing activities.

Normal and Error Returns:

On exit from Field Editor, D0.L contains a response code. If this is positive or zero the return is designated as a normal return. If it is negative, the return is designated as an error return. Condition codes are set to reflect the value in D0.

Normal Returns:

D0.L contains a binary-coded response code denoting the reason for the exit from the field editor. The values which may be returned are,

0 Immediate exit requested (i.e. CS.FENMODE = 1)
2 Field skip forwards (see note 1 in section 2.2.4.3.4)
4 Field skip backwards A (see note 6)
6 Field skip backwards B (see note 11)
8 Exit made on any keystroke, before it is actioned
10 Exit made on any valid keystroke, after it is actioned
12 Caller-defined terminator
14 Function key 0
16 Function key 1
18 Function key 2
20 Function key 3

22 Function key 4
24 Function key 5
26 Function key 6
28 Function key 7
30 Function key 8
32 Function key 9
34 DELETE
36 INSERT
38 REMOVE
40 | and cursor is at window edge
42 | and cursor is at window edge
44 - and cursor is at window edge (see note 15)
46 - and cursor is at window edge (see note 15)
48 Return as a result of condition specified by bit 9 of CS.FCFLAGS
50 ENTER
52 RETURN (see note 14)
54 DELETE and cursor is at start of field (see note 17)

A response code value is returned only if the corresponding bit in CS.FCFLAGS is set, e.g. if bit 5 of CS.FCFLAGS = 0 then response values between 14 and 32 will not be given. This means that applications do not have to cope with return codes in which they have no interest, since they will not occur. See note on use in section 2.2.4.3.5.

Error Returns:

BP : bad parameter
UV : unexpected event

See also section 2.2.4.3.6.

2.2.4.3.4 Field Editor Parameter Block

The parameter block is a contiguous area of RAM used to convey instructions to the Field Editor and responses back from it.

The parameter block must be in a cell allocator segment and need not be at the start of the cell, but must be word-aligned within it.

The following items are defined. The order is that given in this list. Block moves of parameters are therefore permissible.

Where an item is described as a parameter item, this means that it is treated as an input parameter to the Field Editor and may be used to control the action of the Field Editor.

Where an item is described as a return item this means that it is information returned by the Field Editor on exit, which may be used by the calling application.

An 'INCLUDE' file containing all the items defined below is provided as part of the subsystem. It is called CSFDEFS.DG.

| | | |
|---|---|---|
| CS.FARRDSP | 2 bytes | Displacement of data array from start of its cell, in bytes. (The data array must be in a cell allocator segment, need not be at the start of the cell, but must be word-aligned within it.) |
| CS.FARRCTG | 2 bytes | Cell tag of cell containing data array. |
| CS.FARRLEN | 2 bytes | Number of existing data characters in the data array. A return item as well as a parameter item. |
| CS.FKEYCHAN | 4 bytes | Channel identifier for the normal keyboard. |
| CS.FWINCHAN | 4 bytes | Channel identifier for the window which Field Editor is to use. |
| CS.FLOCEVENT | 2 bytes | Number of a local event free for Field Editor's use, coded in binary. |
| CS.FENMODE | 2 bytes | Entry mode. Controls the initial action of Field Editor. The value must be one of the following. |
| | 0 - | Enter normal processing immediately (i.e. do not blank fill or data fill the screen field - the calling application is assumed already to have |

written the screen field).

1 - Undefined; do not use.

2 - Fill the screen field and data array with spaces and then enter normal processing.

3 - Fill the screen field and data array with spaces and then exit to the calling application.

4 - Write the contents of the data array to the screen field and then enter normal processing.

5 - Write the contents of the data array to the screen field and then exit to the calling application.

The effect of other values is undefined.

| | | |
|---|---|---|
| CS.FAROFF | 2 bytes | Number of character positions along the field at which the start of the data array is aligned. See note 2. |
| CS.FINK | 1 byte | Ink/Foreground colour for field. Value is coded as described in Kernel PSD. Bits 4 to 7 must be zero. See note 12. |
| CS.FPAPER | 1 byte | Paper/Background colour for field. Value is coded as described in Kernel PSD. Bits 4 to 7 must be zero. See note 12. |
| CS.FWIDTH | 2 bytes | Number of character positions across the field horizontally. Minimum 1, maximum 80. |
| CS.FHEIGHT | 2 bytes | Number of character positions across the field vertically, i.e. number of lines in field. Minimum 1, maximum 24. |
| CS.FCURARR | 2 bytes | Cursor position within data array as an index pointer from the start of the array, i.e. a value of 0 means the first byte of the data array.<br><br>This is a parameter item if CS.FENMODE has a value of 0 or 4 and is a return item if CS.FENMODE has a value of 0, 2 or 4. See note 3. |

| | | |
|---|---|---|
| CS.FCURH | 2 bytes | Character cursor position horizontally within the window or field. This is a return item only if CS.FENMODE has a value of 0, 2 or 4. It is never used as a parameter item. See note 13. |
| CS.FCURV | 2 bytes | Character cursor position vertically within the window or field. This is a return item only if CS.FENMODE has a value of 0, 2 or 4. It is never used as a parameter item. |
| CS.FVAL | 2 bytes | Validation flags. Specify what validation criteria are to be applied to each data character received from the keyboard. See note 4.<br><br>The bits listed below are individually significant. If two or more of bits 0 to 4 are set then a data character meeting at least one of the criteria (but not necessarily all of them) is valid. A bit is 'on' if set to 1.<br><br>Bit 0 - Alphabetic (upper or lower case).<br><br>Bit 1 - Numeric.<br><br>Bit 2 - Numeric or space.<br><br>Bit 3 - Printable ASCII character, $20 to $7F, inclusive, or $CA or $CB.<br><br>Bit 4 - Telephone digit ($20 to $39 inclusive, or $CA).<br><br>Bit 5 - User-defined set (see below).<br><br>Bit 6 - Leading space in field not permitted. See note 5.<br><br>Bit 7 - Must be 0. |
| CS.FCFLAGS | 4 bytes | Control flags, mainly concerned with specifying the conditions under which return should be made to the calling application. The bits listed below are individually significant; if at least one of the criteria is met, then return is made. |

The internal codings for the characters referred to appear in the Kernel specification.

The bits listed below are 'on' or active if set to 1. Bits for unwanted or undefined conditions should be set to 0.

Bit 0 - Must be 0

Bit 1 - Must be 0

Bit 2 - Must be 0

Bit 3 - Exit if RETURN (this provides the calling application with the ability to intercept RETURN keystrokes, to take special action on them. It is provided for applications which need to do this, e.g. Messaging, and should not be used by other applications). If bit 3 and bit 9 are both set, then bit 3 takes precedence (i.e. return code is 52).

Bit 4 - Exit with return code of 54 if DELETE when cursor is already in first character position of field (see note 17).

Bit 5 - Exit if function key pressed (but see CS.FFNKMSK).

Bit 6 - Exit if DELETE, INSERT or REMOVE. (D0 or D1 show which; exit is made before the character has been actioned.)

Bit 7 - Exit if | or | and cursor is already at window edge. (D0 or D1 show which.)

Bit 8 - Exit if - or - and cursor is already at window edge. (D0 or D1 shows which.)

Bit 9 - Exit if INSERT, REMOVE,

DELETE, RETURN, SHIFT/TAB within the field, ALT/RETURN, -, - , | or | (after it has been actioned). See notes 10 and 16.

Bit 10 - Exit on any data keystroke i.e. a valid keystroke which is not a caller-defined terminator. Exit after the data array, its length and the cursor pointers have been updated.

Bit 11 - Exit on any keystroke (exit before the keystroke is actioned in any way; the keystroke is in D1; the data array, its length and the cursor pointers are not updated; no character validation is performed). The return code is 8.

Bit 12 - Exit if caller-defined terminator, see below (exit before the data array, its length and the cursor pointers have been updated; the character, whose value is in D1, has not been validated).

Bit 13 - Must be 0.

Bit 14 - Must be 0.

Bit 15 - Freeze control bit A. If set to 1, signifies that the segment or segments containing both the data array and the parameter block should be thawed before a return is made to the caller. If set to 0, the segment or segments remain frozen after the return.

Bit 16 - Set to 1 if the channel display mode is either 80 column mode or 64 column mode and the character

attribute 'Double Width' is currently if force. Otherwise, set to 0.

Bits 17 to 22 — Must be 0.

Bit 23 - Freeze control bit B. If set to 0, the segment or segments containing both the data array and the parameter block should be thawed while awaiting user input from the keyboard. (This should be regarded as the normal case.) If set to 1, such thawing while waiting for user input does not take place.

Bits 24 to 31 — Must be 0.

| | | |
|---|---|---|
| CS.FFNKMSK | 2 bytes | Further qualifies bit 5 of CS.FCFLAGS by indicating a subset of function keys which is to cause an exit. Function keys outside this set are beeped and ignored.<br><br>Function key N (0 through 9) is part of the subset which causes an exit if bit N of this word is 1 and is not part of the subset if bit N is 0. Bits 10 to 15 must be 0.<br><br>If an exit is made, the return code is as specified in 2.2.4.3.3. |
| CS.FCALVAL | 6 bytes | (Up to) 6 character codes which are to be considered as valid input characters. These caller supplied codes are used for validation if, and only if, bit 5 of CS.FVAL = 1. Unused positions should be set to $FF (though the values of all 6 bytes are irrelevant if bit 5 of CS.FVAL = 0). |
| CS.FCALTERM | 3 to 257 bytes | A table of internal codes, the keying of any one of which causes an exit to be made to the calling application if, and only if, bit 12 of CS.FCFLAGS = 1. If bit 12 of CS.FCFLAGS = 0 then this table (if it exists) is not examined. |

Byte 0 and 1 of CS.FCALTERM contains a non-zero integer, N, which is the number of potential terminators held in the remainder of the table. Bytes 2 to N each contain the internal code of such a potential terminator.

Any editing control characters which the Field Editor does not action but which the calling application wishes to action must appear here.

Notes

Notes 7, 8 and 9 are general notes.

Note 1 : This return occurs for one of the following reasons,

a) The field is full - specifically, a data character has been typed into the last character position of the field. (INSERT does not cause a 'field full' condition.)

b) TAB has been keyed.

c) RETURN or ALT/RETURN has been keyed while the cursor is in the last line of the field, and bit 3 of CS.FCFLAGS = 0.

d) ALT/RETURN has been keyed while the cursor is in the last line of the field.

e) - has been keyed while the cursor is in the last character position of the field.

Note 2 : This provides a means of 'protecting' a number of characters at the start of the field. The cursor is not allowed to enter the 'protected' area, which has become not part of the field at all - i.e. is not represented in the data array.

In normal use this parameter is 0.

There must always be at least 1 character position in the field which is not 'protected'.

Note 3 : If normal processing is entered, i.e. the immediate exit escape is not specified, then CS.FCURARR, CS.FCURH and CS.FCURV are always returned in a consistent state, pointing up the current position of the cursor within the data array and the screen field.

Note 4 : If no bits are set in CS.FVAL then character validation always fails. It is impossible, therefore, to enter data characters.

Note 5 : This bit provides a negative check. If set, an attempt to enter a space character into the first character position of the field is rejected (though a space could be created implicitly by INSERT). It must be used in conjunction with one of the other bits specifying an allowable range. It is intended for use on fields which are name fields for telephone directories or are shortcodes.

Note 6 : Field skip backwards A - this return occurs if SHIFT/TAB is keyed while the cursor is already in the first character postion of the field (if the cursor is elsewhere, only cursor homing within the current field is implied and this is actioned internally within the editor, there is no return to the application).

Note 7 : The data array must be at least large enough to accommodate the number of 'non-protected' characters in the field, since the user is able to cause Field Editor to write data into any part of the array. Items CS.FCALVAL and CS.FCALTERM need not be present in the parameter block if they are not called for by the other parameters.

Note 8 : If blank fill mode is selected (see CS.FENMODE) then the data array is entirely spacefilled at the start but the cursor remains at the first character position in the field and CS.FARRLEN returns the right-most point in the field subsequently reached by the user with explicit or implicit cursor movements.

Note 9 : If the calling application has already written the screen field when the Field Editor is called then the data array must contain as many characters as are present in the 'non-protected' part of the field. This is because explicit cursor movement will cause CS.FARRLEN to record the furthest position reached in the data array, but will not actually deposit data characters into the data array.

Note 10: If it is required to use the bit 9 facility, then bits 7 and 8 should not be set, since these will take precedence at window edges.

Note 11 : Field skip backwards B - this return occurs if - is keyed when the cursor is already in the first character position in the field.

Note 12 : The specified values for ink and paper colours take effect only for characters displayed or redisplayed as a result of keyboard input. i.e. if CS.FENMODE = 0 then the already-displayed field contents are not immediately altered.

Note 13 : If bit 16 of CS.FCFLAGS = 1, i.e. the window has the 'double width' property, then CS.FCURH does not contain the character position but a value of twice this. It may therefore be used with a cursor positioning command to Kernel to address the current character, just as for 'single width' cases.

Note 14 : This return only occurs if bit 3 of CS.FCFLAGS = 1.

Note 15 : This return occurs if bit 8 of CS.FCFLAGS = 1 and the cursor is not already in the first (for - ) or last (for - ) character postion of the field, i.e. it does not take precedence over the normal field skip return codes.

Note 16 : If exit is made as a result of this bit, it is the responsibility of the calling application to detect cases where a field skip is implied and to act accordingly.

Note 17: If bit 4 is to be used, then bit 6 must not be set, since bit 6 takes precedence.

2.2.4.3.5 Using Response Codes

The positive response code values returned by Field Editor are designed for use with 2 byte jump tables. If desired, they may be doubled and used with 4 byte jump tables.

For use with 2 byte jump tables, code of the nature outlined below may be used.

a) To effect the switch:

```
LABEL MOVE.W  TABLE(PC,D0),D0
      JMP     LABEL(PC,D0)
```

b) The jump table takes the form:

```
TABLE DC.W  Destination 1 - LABEL
      DC.W  Destination 2 - LABEL
      DC.W  Destination 3 - LABEL
       .
       .
       .
       .
```

The destinations must be within 32 Kb, forwards or backwards, of LABEL. TABLE must lie within 127 bytes, forwards or backwards, of LABEL, though the details can be rearranged to suit individual circumstances.

2.2.4.3.3 Error Handling

I/O errors on the screen and keyboard channel are passed back to the calling application.

If a negative return code from Field Editor is neither ERR.BP nor ERR.UV, then it is a normal Kernel error response returned from one of Field Editor's Kernel I/O calls. ERR.BP might also be passed back from a Kernel call rather than the call of Field Editor itself.

The audible error tone generated is produced by calling the Director procedure MAKE SOUND, with an action value of TG.BADKEY.

2.2.4.3.7 Order of processing

When Field Editor has received a key depression, the order of processing is,

1. If exit on any keystroke specified, exit with return code of 8.
2. Check whether it is a caller-terminator; if so, exit.
3. Validate the key value as a data character in accordance with CS.FVAL. If success, process as a data character.
4. Validate the key value as a caller-defined valid character. If success, process as a data character.
5. Attempt to identify the key value as a control character (one of - DELETE, INSERT, REMOVE, ENTER, RETURN, ALT/RETURN, TAB SHIFT/TAB, | , | , - or - ). If identified, process accordingly.
6. Attempt to identify the key value as a permitted function key. If identified, process accordingly.
7. The keyed character is illegal. An error tone is generated and the character is otherwise ignored.

## 2.3 The Waiter Subsystem

### 2.3.1 PURPOSE

The Waiter subsystem is a collection of subroutines which are designed to perform for applications some of the chores necessary when displaying information on the screen and awaiting a user response.

Typically, it is used to display a menu and to await the user's keystroke in reply. It is not, however, intended to be restricted to this purpose and so the interface contains a number of parameters which may be used to control the action of the subsystem in the desired manner.

### 2.3.2 DESCRIPTION

The main functions of the subsystem are the display of text, the reading of keyed input and being constantly ready for the occurrence of certain events.

#### 2.3.2.1 Text Display

The word 'text' is used loosely; it may comprise any data acceptable to the screen driver.

A complete screen display consists of a number of lines, each which is a main text definition in the terms of Text Expander. Such a line may actually be less than, equal to or greater than one visible line of display on the screen. They should not however be made too long otherwise responsiveness may suffer since keystrokes are only detected at the end of each line.

The collection of lines comprising the complete screen display is called, in this document, the Main text library. There is usually an associated Common text library containing expansions of common strings, and also a variant number (see section 2.1.3.2).

A common requirement is to want to delay the start of the display process for a fraction of a second (currently guessed to be half a second) while looking for user input. This is for use when processing a hierarchy of menus by an experienced user without several screen displays each starting to appear and each persisting only briefly before being overwritten by the next display. The requirement is met by the provision of alternative entry points to the main routine.

#### 2.3.2.2 Key Reading

The source of potential keystrokes may be varied by the calling application to suit its requirements. The variations provided are:

a) Use READ MENU KEY until this stream is exhausted, then use the normal keyboard channel (for processing menus dangling from the top level Menu).

b) Use normal keyboard channel only (for processing non-menu display).

c) Use READ REVIEW KEY (for use in Review Mode code).

A minimal validation feature is incorporated which permits the keyed character to be checked as one of a specified set of numeric digits. It is intended to be used in simple menu processing where all the valid responses are digits.

### 2.3.2.3 Events

It is imperative that an application is alert at all times for the events Suspend Foreground, Terminate and Abandon. This includes the time spent by the application in the Waiter subsystem. The Waiter subsystem therefore checks frequently for the occurrence of events. If one occurs (other than the events used for its own I/O operations), an immediate exit is normally made to the calling application, with a distinctive return code. The event or events detected are specified in the Parameter Block in the parameter CS.WEVENTS (see section 2.3.4.1). In general this parameter may be used by the calling application and by the Waiter subsystem. Events which are notified should have the corresponding bits set in CS.WEVENTS; when an event is processed its bit should be cleared (except the foreground allocated event bit which must remain set, unless cleared by the routine CS WSUSP). Further details are given in sections 2.3.4.1 and 2.3.6.1.

### 2.3.2.4 Other features of the subsystem

1 The subroutine that displays one line of text on the screen, CS_WDISP, is made visible separately. While it is possible to call this from outside the subsystem, if required, the main purpose is to allow the calling application to provide another display routine to be used by the Waiter subsystem which meets the specific requirements of the application. There is no limitation on the functionality of such a substituted display routine, provided that it meets the interface specification of CS_WDISP in respect of method of call, registers on entry, and return information (except CS.WDLNO which may have any value) - see sections 2.3.5.4.1 and 2.3.5.4.3.

2 The subroutine that initiates or completes a keyboard read is made visible separately, so that an application can call it directly if required.

3 A facility is provided for the subsystem to exit back to the calling application as soon as the display has been

completed. In this case, a keyboard read (if required at all) is still pending.

4 A re-entry feature is provided to allow the calling application to re-enter the subsystem after is has returned to the caller (for any reason).

### 2.3.3 FORM OF SUBSYSTEM

The subsystem consists of 8 subroutines which share a common database, the Waiter Parameter Block. The names of these and an outline of their functions appear below.

All of these are entered using a single action value, CS.WAITER, on the Trap entry to COMSUB. They are then distinguished on the basis of an action value variant, held in D1.

#### 2.3.3.1 CS_WMAIN

This routine contains the control logic for the subsystem.

It has 2 entry points:-

a) Optionally wait for user-specified initial delay, display to the screen and/or read a single keyboard depression.

b) Re-entry for any reason, where the desired action is to carry on where the main routine left off before a prior exit, e.g. when an unexpected event occurred, or when the calling application has detected an invalid key.

These 2 "routines" are called CS_WMN0 and CS_WMN1 respectively.

#### 2.3.3.2 CS_WINIT

Initialising routine which:

- Requests future notification of screen and keyboard I/O events.
- Requests foreground (unless suppressed due to REVIEW or SHIFT/SPECIAL key depression).
- Opens screen channel.

The routine must (if used) be called prior to calling any other routine in the subsystem. If the routine is not called then before calling other routines, certain actions should be carried out as described in section 2.3.5.3.7.

#### 2.3.3.3 CS_WDISP

Routine to format and display a text library "line" (a modest amount) to the screen. Provision is made for the substitution of this routine by a caller-supplied alternative.

#### 2.3.3.4 CS_WPUTSTR

Routine to display an already formatted line to the screen and wait for the screen I/O event.

### 2.3.3.5 CS_WREAD

Routine to ask for keyboard input. If any is available, it is returned. Otherwise, an autonomous read is issued and the routine exits. Availability of a key later is signalled by an event. A further call will then retrieve that key value (unless it fails an optional numeric validation test).

### 2.3.3.6 CS_WSUSP

Routine to handle the 'Suspend Foreground' event. Should be called by an application which has been using the subsystem, whether the subsystem or the caller has detected this event.

The routine causes the foreground to be suspended and the normal keyboard channel (if open) to be closed.

### 2.3.3.7 CS_WKOPEN

Routine to open and select a normal keyboard channel. The Waiter subsystem will open and select a channel when it is about to read a key from a normal keyboard, if the channel is not yet open. However, a user application may wish to explicitly open such a channel when it is not opened by Waiter e.g. for Field Editor.

## 2.3.4 DATA STRUCTURES USED

### 2.3.4.1 The Waiter Parameter Block

The Waiter Parameter Block is an area of RAM provided by the calling application, which is used to convey information to and from the Waiter routines and also as workspace by the Waiter routines. It may be in either a normal segment or a cell allocator segment. It must however be either frozen or immobile on entry to the Waiter routines. It is not thawed by the Waiter routines.

A convenient place for the block will often be on the calling application's stack, though this not mandatory.

The block must be word aligned.

The contents of the individual fields in the Parameter Block are described below. The names and field lengths are available in an INCLUDE file (see Release Notice for its name).

| | | | |
|---|---|---|---|
| CS.WKBCHIDE | 4 bytes | | Identifier for currently selected normal keyboard channel. Zero indicates that Waiter is to open and select a normal keyboard channel, if required by CS.WRFLAG. |
| CS.WSCCHIDE | 4 bytes | | Identifier for screen channel. Normally set by a call of CS_WINIT. |
| CS.WEVENTS | 4 bytes | a) | An events bit map signifying events which have occurred during entry to Waiter. These include:<br>- suspend foreground, abandon, terminate events (as described in Kernel PSD, 76.97.3.1)<br>- any user event for which notification requested<br>- foreground allocated event may be set if an exit is made from Waiter for one of the above events (event defined in Kernel PSD, 76.97.3.1; once set by Waiter it remains set unless the foreground is suspended by a call of CS_WSUSP) |
| | | b) | On entry to Waiter the bit map should indicate which Waiter |

| | | |
|---|---|---|
| | | events are outstanding from the previous entry, plus those Waiter events which have been notified to the calling application: these are the foreground allocated and suspend foreground events. |
| CS.WMTLAD | 4 bytes | Address of (first byte of index of) the Main text library for the screen display to be made. |
| CS.WCTLAD | 4 bytes | Address of (first byte of index of) the Common text library, if the Main text library references one. |
| CS.WTVAR | 1 byte | Text variant number for the Text Expander. |
| CS.WDLINES | 2 bytes | Number of lines (i.e. entries) in the Main text library, which constitute the screen display to be made by the current entry to Waiter.<br><br>0 = no line to display, presumably because Waiter is only being entered for keyboard read - see CS.WRFLAG. |
| CS.WDLNO | 2 bytes | Line number of the next line in the Main text library to be displayed. Normally starts at 0 but caller may set >0. This field is updated by Waiter as the display progresses. |
| CS.WDBAD | 4 bytes | Address of first byte of buffer supplied for the receipt of expanded text. See section 2.3.4.2. |
| CS.WDBLEN | 4 bytes | Length of buffer supplied for the receipt of expanded text. Not required if CS.WDLINES = 0. |
| CS.WCDR | 4 bytes | 0 - subsystem is to use subroutine CS_WDISP to display each line.<br><br>≠0 - the address of a caller -supplied line display routine which is to be used by the subsystem. |

| | | |
|---|---|---|
| | | Not required if CS.WDLINES = 0. |
| CS.WDELAY | 2 bytes | Optional time delay before display of first line from Main text text library, in units of 20ms (approx).<br>0 = no delay, -1 = default delay (currently 0.5 sec). |
| CS.WSMODE | 1 byte | Channel display mode for screen channel. Encoded as for SET CHANNEL DISPLAY MODES - see Kernel PSD, 76.97.3.1. |
| CS.WRFLAG | 1 byte | 0 - the routine is <u>not</u> required to look for keyboard input<br>1 - read normal keyboard channel<br>2 - READ MENU KEY, then normal keyboard channel<br>3 - READ REVIEW KEY |
| CS.WVFLAG | 2 bytes | 0 - no key validation is performed; any key value triggers an exit to the calling application.<br>≠0 - only bits 0 to 9 are significant; bits 10 to 15 must be 0. Key validation is performed. The only valid keys are the unshifted numeric digits. Digit N is valid if bit N in this flag's word is a 1, and is invalid otherwise. |
| CS.WEFLAG | 1 byte | 0 - control remains in CS_WMAIN until an exit to the caller is caused by either a valid key depression or by terminating event.<br>1 - an exit is made to the caller as soon as the display is complete (unless a valid key depression or a terminating event occurs before the display has been completed). |

| | | |
|---|---|---|
| CS.WDFLAG | 1 byte | Used as workspace, not an input parameter to the Waiter subsystem.<br><br>0 - display disabled (not yet started)<br><br>1 - display enabled |
| CS.WTLEFT | 2 bytes | Used as workspace, not an input parameter to the Waiter subsystem. No. of text entries still to be displayed. |
| CS.WCDLNO | 2 bytes | Used as workspace, not an input parameter to the Waiter subsystem. Copy of CS.WDLNO on entry at action variant CS_WMNO; used to reset CS.WDLNO on return after foreground suspended. |
| CS.WFFLAG | 1 byte | Used as workspace, not an input parameter to the Waiter subsystem.<br><br>0 - foreground allocated event not yet processed (initially cleared by CS_WINIT).<br><br>1 - the event has been processed. |
| CS.WRMFLAG | 1 byte | Used as workspace, not an input parameter to the Waiter subsystem.<br><br>0 - Waiter has not switched from READ MENU KEY to read normal keyboard.<br><br>1 - the switch has been made. |
| CS.WENTRY | 1 byte | Used as workspace, not an input parameter to the Waiter subsystem.<br><br>0 - entered via TRAP mechanism<br><br>Note: only applies to routines that return values in registers (other than D0). |
| CS.WDEFSZ | EQU | Size of parameter block in bytes. |

2.3.4.2 Text Expansion Buffer

This is a separate area from the Waiter Parameter Block itself, since size requirements may vary widely. Nevertheless it may often be convenient to site it next to the parameter block, or on the calling application's stack.

The size of the buffer needs to be adequate to contain the longest line from the Main text library in expanded form (taking account not only of straight text characters but also of the expansion caused by text substitution and of any screen control characters embedded in the line).

The buffer may be in a normal segment or a cell allocator segment, but the segment must be frozen or immobile on entry to the subsystem. It is not thawed by the subsystem.

## 2.3.5 INTERFACES

### 2.3.5.1 Interface - Routine CS_WMNO

2.3.5.1.1 Call and Registers

Trap Name: T.CMNSUBS
Action Value (D0.B): CS.WAITER
Action Value Variant (D1.B): CS.WMNO

A3: Address of standard Waiter Parameter Block.

2.3.5.1.2 Additional Call Parameters

CS.WKBCHIDE — See section see 2.3.4.1.2. Not required if CS.WRFLAG = 0.

CS.WSCCHIDE — See section 2.3.4.1.2. Not required if CS.WDLINES = 0.

CS.WEVENTS — Not normally a CS_WMNO parameter, but see sections 2.3.2.3., 2.3.6.1. and CS.WEVENTS in section 2.3.4.1.2.

CS.WMTLAD — Address of Main text library. Not required if CS.WDLINES = 0 or CS.WCDR ≠ 0.

CS.WCTLAD — Address of Common text library. Not required if CS.WDLINES = 0 or CS.WCDR ≠ 0.

CS.WTVAR — Text variant number for Text Expander. Not required if CS.WDLINES = 0 or CS.WCDR ≠ 0.

CS.WDLINES — Total number of lines to be displayed. 0 = keyboard read without screen display.

CS.WDLNO — Normally set to 0, but may indicate any Main text library entry from which 1 or more lines are to be displayed. Not required if CS.WDLINES = 0 or CS.WCDR ≠ 0.

CS.WDBAD — Address of first byte of buffer supplied for text expansion, in bytes. Not required if CS.WDLINES = 0.

CS.WDBLEN — Length of buffer supplied for text expansion, in bytes. Not required if CS.WDLINES = 0.

CS.WCDR — Address of caller subroutine to be used by Waiter subsystem in place of CS WDISP. Not required if CS.WDLINES = 0.

CS.WDELAY — Time delay, in units of 20ms, before the display is initiated. Not required if CS.WDLINES = 0; -1

= default delay.

CS.WRFLAG
0 - keyboard input is not required
1 - read normal keyboard channel
2 - READ MENU KEY, then normal keyboard channel
3 - READ REVIEW KEY

CS.WVFLAG
0 - no key validation is required; any key value triggers an exit to the calling application.

≠0 - bits 0 to 9 are significant, bits 10 to 15 must be 0. Key validation is performed, the only valid keys being unshifted numeric digits, such that digit N is valid if bit N of this word is 1.

CS.WEFLAG
0 - control remains in CS_WMAIN until the exit to the caller is caused by either a valid key depression or by a terminating event.

1 - an exit is made to the caller as soon as the display is complete, with D0 = 0, unless a valid key depression or a terminating event occurs before the display has been completed. When an exit is made for this reason, a keyboard read is still active and the associated event (14) may subsequently occur in the calling application.

2.3.5.1.3 Return Information

D0.L
0 - Return due to display complete - no valid key has been read (or key read was not required).

1 - Return due to a valid key depression (the display may be incomplete if CS.WEFLAG=0) - see D1.B.

ERR.UV - A terminating event has occurred (see CS.WEVENTS).

Other -ve - Kernel/Application Handler/Text Expander error response.

D1.B - When D0.L = 1, hexadecimal value of the key read if no validation, or binary value of validated integer.

CC - Reflects the state of D0.

CS.WKBCHIDE - When a normal keyboard channel has been opened by Waiter, its channel identifier

is present in this parameter i.e. the channel is still open and may be used by the caller.

CS.WEVENTS - A (bit significant) register of events which have been detected during execution of CS WMAIN. This is significant only of D0 = ERR.UV. The event used by Waiter for screen, keyboard I/O are not reported in this field. See also under CS.WEVENTS in section 2.3.4.1.2.

2.3.5.1.4 Function

The function of entry to the main Waiter subsystem at entry point CS WMNO is to:

- optionally display one or more lines to the screen (CS.WDLINES)

- optionally read a keyboard depression (CS.WRFLAG)

The lines displayed are built up by the Text Expander from a Main text library, possibly referencing text strings in a Common text library and possibly using a text variant. Alternatively, a user routine (specified by CS.WCDR) may be called from within Waiter to build and display each line.

Display may be subject to an initial delay (see CS.WDELAY).

When a key is to be read, it can be validated if only a numeric digit is expected (CS.WVFLAG) - failure causes an invalid key beep to be passed to the internal speaker followed by another attempt to read a key.

When all lines have been displayed before a key has been read, an immediate exit may be made (CS.WEFLAG).

Waiter looks for all events for which the activity has requested notification including user events.

2.3.5.1.5 Registers Destroyed

All registers except D0 are preserved on exit.

2.3.5.1.6 Waiter Parameter Block

This must be in a segment that is frozen or immobile on entry to the subroutine. It is not thawed by the subroutine.

### 2.3.5.2 Interface - Routine CS_WMN1

2.3.5.2.1 Call and Registers

Trap name: T.CMNSUBS
Action Value (D0.B): CS.WAITER
Action Value Variant (D1.B): CS.WMN1

2.3.5.2.2 Additional Call Parameters

The parameter block values set on the previous exit should remain unchanged, with the following possible exception:-

CS.WEVENTS See sections 2.3.2.3, 2.3.6.1 and CS.WEVENTS in section 2.3.4.1.2.

If it is desired to change the mode of operation of CS_WMAIN on re-entry, then the parameter block should be set accordingly, and a fresh entry made with action value variant CS.WMN0.

2.3.5.2.3 Return Information

As for action value variant CS.WMN0 (see section 2.3.5.1.3).

2.3.5.2.4 Function

The function of the Main routine at this re-entry point is the same as that at the action value variant CS.WMN0 entry point, except that the routine continues where it left off after the previous exit.

2.3.5.2.5 Registers Destroyed

All registers except D0 are preserved on exit.

2.3.5.2.6 Waiter Parameter Block

This must be in a segment that is frozen or immobile on entry to the subroutine. It is not thawed by the subroutine.

### 2.3.5.3 Interface - Routine CS WINIT

2.3.5.3.1 Call and Registers

Trap Name: T.CMNSUBS
Action Value (D0.B): CS.WAITER
Action Variant Value (D1.B): CS.WINIT

A3: Address of standard Waiter Parameter Block

D2.W: The value required in D1.W by the REQUEST FOREGROUND procedure for the desired effects.

D3.B: 0 - foreground required

1 - foreground already allocated e.g. due to REVIEW or SHIFT/SPECIAL key.

2.3.5.3.2 Additional Call Parameters

CS.WSMODE Screen channel display mode, encoded as for SET CHANNEL DISPLAY MODES - see Kernel PSD, 76.97.3.1.

2.3.5.3.3 Return Information

D0.L 0 - Successful call.

-1 - Application Handler/Kernel error response.

CC - Reflects the state of D0.L.

CS.WSCCHIDE - Identifier for screen channel.

CS.WEVENTS - The saved events parameter is zeroised.

2.3.5.3.4 Function

The subroutine provides the initial processing for the Waiter subsystem.

The Waiter parameter CS.WEVENTS is zeroised.

The routine adds to the event request register the events for

Waiter Keyboard I/O - 14
Waiter Screen I/O - 15

It calls the Application Handler procedure REQUEST FOREGROUND if foreground is not already allocated, but does not wait for the foreground allocated event - the Waiter subsystem waits when entered with action variants CS.WMNO or CS.WMN1.

Finally, a screen channel is opened if the routine has just

requested the foreground. See also section 2.3.5.3.8 below.

2.3.5.3.5 Registers Destroyed

All registers except D0 are preserved on exit.

2.3.5.3.6 Waiter Parameter Block

This must be in a segment that is frozen or immobile on entry to the subroutine. It is not thawed by the subroutine.

2.3.5.3.7 Using Another Screen Channel

If a caller of the Waiter subsystem wishes to use an already -available screen channel (for example the one already provided by Application Handler for Review mode applications) then there is still a need for CS_WINIT to be called. If the routine is not called, the Waiter keyboard event (currently 14) must be added to the activity event request register if any keyboard input will be requested, and the channel identifier must be written to CS.WSCCHIDE; the Waiter screen event (currently 15) must be added to the activity event request register if Waiter display facilities will be used; the saved events register CS.WEVENTS, and the internal flag CS.WFFLAG should be set to zero.

### 2.3.5.4 Interface - Routine CS_WDISP

#### 2.3.5.4.1 Call and Registers

Trap name: T.CMNSUBS
Action Value (D0.B): CS.WAITER
Action Variant Value (D1.B): CS.WDISP

A3: Address of standard Waiter Parameter Block

#### 2.3.5.4.2 Additional Call Parameters

CS.WSCCHIDE — Screen channel identifier - normally set by a call of CS.WINIT.

CS.WDBAD — Address of display buffer for Text Expander - see section 2.3.4.2.

CS.WDBLEN — Length of display buffer in bytes - see section 2.3.4.2.

CS.WMTLAD — Address of Main text library

CS.WCTLAD — Address of Common text library

CS.WDLNO — Number of next display line in the text library (numbered from zero).

CS.WTVAR — Text variant number.

CS.WTLEFT — When CS_WDISP is called by the Waiter subsystem, this internal field = no. of text entries left to display (Waiter sets CS.WTLEFT = CS.WDLINES on entry at CS_WMNO, or CS_WMN1 after a suspend foreground event). When CS_WDISP is called by an application, the caller may use this field similarly if making repeated calls of CS_WDISP, or ignore it if only one call is being made.

#### 2.3.5.4.3 Return Information

D0.L
- 0 - Successful return.
- 1 - Transfer cancelled.
- -ve - Kernel or Text Expander error response.

CC - Reflects the state of D0.L.

CS.WDLNO - Incremented by 1 if line has been displayed successfully.

CS.WEVENTS - Event bits are set for any non-screen I/O events which were notified by Kernel during

the display of the current line - other event bits will be unchanged.
Note: D0 is not set = ERR.UV when events are reported since 'transfer cancelled' may be relevant.

CS.WTLEFT - This parameter will have been decremented by 1, to indicate the no. of text entries left to display.

2.3.5.4.4 Function

The subroutine displays a text entry (normally one line) from the Main text library to the screen.

The line displayed is the 'next' line, which may be controlled by the parameter CS.WDLNO. When it has been displayed successfully the number in CS.WDLNO is incremented by 1 to provide a new 'next' line, and CS.WTLEFT is decremented by 1. The line is presented to the Text Expander to expand with any necessary substitutions. It is then displayed to the screen, starting at the current cursor position (unless amended by embedded screen control characters).

The routine always waits for its screen transfer to finish, even if other events occur before this completion. The screen transfer completion event will never, therefore, occur after exit from the subroutine.

Cancellation of the screen transfer by Kernel is not treated as an error. The return is a normal 'successful' one, though CS.WDLNO is not incremented. The next entry to the subroutine therefore attempts to display the same line again.

In the event of another device error being reported by Kernel on the screen channel, then it is this device error code (as obtained from GIVE CHANNEL STATUS) which is returned to the caller.

This subroutine may, if required, be substituted (by setting CS.WCDR) within the overall Waiter subsystem by a caller-specified replacement. Such a replacement may do whatever is required (e.g. it might not use the Text Expander to output a pre-defined "menu") provided that it returns information as described in section 2.3.5.4.3. The subroutine should check frequently for events (all events not just its own) and when it returns it must have cleared and recorded events that occurred, in CS.WEVENTS.

2.3.5.4.5 Registers Destroyed

All registers except D0 are preserved on exit.

2.3.5.4.6 Waiter Parameter Block

This must be in a segment that is frozen or immobile on entry to the subroutine. It is not thawed by the subroutine.

### 2.3.5.5 Interface - Routine CS_WPUTSTR

#### 2.3.5.5.1 Call and Registers

| | |
|---|---|
| Trap Name | T.CMNSUBS |
| Action Value (D0.B): | CS.WAITER |
| Action Value Variant Value (D1.B) | CS.WPUTSTR |

A3: Address of standard Waiter Parameter Block

#### 2.3.5.5.2 Additional Call Parameters

A1: Address of string to display, with text and screen control characters.

D2.L: Length of string to display in bytes.

CS.WSCCHIDE: Screen channel identifier - must be non-zero.

#### 2.3.5.5.3 Return Information

D0.L
- 0 - Successful call.
- 1 - Transfer cancelled.
- -ve - Kernel error response.

CC - Reflects the state of D0.

CS.WEVENTS - Event bits are set for any non-screen I/O events which were notified by Kernel during the display of the current line - other event bits will be unchanged.

Note: D0 is not set = ERR.UV when events are reported, since 'transfer cancelled' may be relevant.

#### 2.3.5.5.4 Function

The subroutine displays an already formatted text string, and then waits for the screen I/O event. Any other events that are notified in the meantime are saved in CS.WEVENTS. When the screen I/O event occurs, the channel status is obtained and passed to the caller in D0, except ERR.TX which is treated as a 'successful' reply. The routine may be called again when a transfer was cancelled.

#### 2.3.5.5.5 Registers Destroyed

All registers except D0 are preserved on exit.

#### 2.3.5.5.6 Waiter Parameter Block

This must be in a segment that is frozen or immobile on entry to

the subroutine. It is not thawed by the subroutine.

### 2.3.5.6 Interface - Routine CS_WREAD

#### 2.3.5.6.1 Call and Registers

Trap Name: T.CMNSUBS
Action Value (D0.B): CS.WAITER
Action Variant Value (D1.B): CS.WREAD

A3: Address of standard Waiter Parameter Block

#### 2.3.5.6.2 Additional Call Parameters

CS.WKBCHIDE — Currently selected normal keyboard channel identifier, if required by CS.WRFLAG. This keyboard channel will be opened by CS_WREAD if CS.WKBCHIDE = 0.

CS.WRFLAG
- 1 - All reading is from the normal keyboard channel
- 2 - Reading is from the stored "menu keys" until exhaustion, then from the normal keyboard channel
- 3 - All reading is from the "Review key channel"

CS.WVFLAG
- 0 - Any key value is acceptable and is passed back to the caller
- ≠0 - Key validation is required. See section 2.3.4.1.2

CS.WRMFLAG — Not normally an input parameter. When repeated calls are made, this parameter will have been set by a previous entry if CS.WRFLAG = 2 and the routine has switched from reading the menu key channel to the normal keyboard channel.

The flag is cleared by routine CS_WSUSP; if the latter is not used, then the calling application should clear CS.WRMFLAG when processing the 'suspend foreground' event.

0 = read a key as indicated by CS.WRFLAG

1 = key reading has switched from read menu key to read normal keyboard

#### 2.3.5.6.3 Return Information

D0.L
- 0 - Return with keyboard read initiated but still outstanding.
- 1 - Return due to a valid key depression - see D1.B.

Other -ve - Kernel/Application Handler error response.

D1.B - When D0.L = 1: hexadecimal value of key read if no validation, or binary value of validated integer.

CC - Relects the state of D0.

CS.WKBCHIDE - When a normal keyboard channel has been opened by CS.WREAD, its identifier is returned in this parameter.

CS.WRMFLAG - See 2.3.5.6.2.

2.3.5.6.4 Function

The subroutine reads a key depression value, or else initiates such a read and then exits.

If an exit is made with a return code of 0, 'no key available', then an event may subsequently occur when the user makes a key depression. The number of this event is given by CS.WENKB (and is currently 14). When this event occurs, the subroutine may be re-entered to read the key value; if re-entered before the event has occurred the resulting error IU (channel in use) will be passed back as NB (no byte available yet).

The Keyboard "channel" used by the subroutine is controllable by the caller. If CS.WRFLAG = 3, then all reading is attempted using the READ REVIEW KEY procedure and the normal keyboard channel is not examined at all. If CS.WRFLAG = 1, then all reading is from the normal keyboard channel. If CS.WRFLAG = 2, then the routine first reads to exhaustion any key depressions that were stored by Application Handler before the normal keyboard channel was opened (i.e. using the READ MENU KEY procedure) and then reads from the normal keyboard channel. [This last should be the normal case for reading while the user is initially selecting down a menu hierarchy, starting from the Top Level Menu.]

Limited key validation is provided, if required. It is possible to validate a key value as a digit in a specified set of digits. If a key value is received which is outside this set then the normal error tone is generated (TG.BADKEY) and the keyboard channel is flushed (including the normal keyboard channel when an invalid key is returned by READ MENU KEY). Internal flags are set so that when the routine is called from within the Waiter subsystem, the screen display is enabled and the user can continue with a new key or key sequence (from the invalid point in the sequence). A new read is then attempted. (This means that the subroutine may sometimes exit with a return code of 0, 'no key available', even if it was entered in response to the event notifying the availability of a key.)

### 2.3.5.7 Interface - Routine CS_WSUSP

2.5.5.7.1 Call and Registers

Trap Name: T.CMNSUBS
Action Value (D0.B) CS.WAITER
Action Variant Value (D1.B): CS.WSUSP

A3: Address of Standard Waiter Parameter Block

2.3.5.7.2 Additional Call Parameters

CS.WKBCHIDE Normal keyboard channel identifier. Zero indicates the channel is not open

2.3.5.7.3 Return Information

D0.L 0 - OK reply.

-ve - Kernel/Application Handler error response.

CC - Reflects the state of D0.

CS.WKBCHIDE 0 - Normal keyboard channel not open.

CS.WEVENTS - The foreground allocated event bit is cleared.

2.3.5.7.4 Function

The subroutine is provided to handle the 'Suspend Foreground' event

It calls the Application Handler procedure SUSPEND FOREGROUND. It then closes the normal Keyboard channel if open (this ensures that the 'key data available' event will not occur while the application is suspended and flushes unprocessed key depressions from the internal read buffer).

Internal flags CS.WFFLAG and CS.WRMFLAG are set back to zero.

2.3.5.7.5 Registers Destroyed

All registers except D0 are preserved on exit.

2.3.5.7.6 Waiter Parameter Block

This must be in a segment that is frozen or immobile on entry to the subroutine. It is not thawed by the subroutine.

### 2.3.5.8 Interface - Routine CS_WKOPEN

#### 2.3.5.8.1 Call and Registers

Trap Name: T.CMNSUBS

Action Value (D0.B): CS.WAITER

Action Variant Value (D1.B): CS.WKOPEN

A3: Address of Standard Waiter Parameter Block

#### 2.3.5.8.2 Additional Call Parameters

CS.WKBCHIDE 0 - Open and select a normal keyboard channel

≠0 - Channel identifier for currently open and selected normal keyboard

#### 2.3.5.8.3 Return Information

D0.L non -ve - OK reply.

-ve - Kernel error response.

CC - Reflects state of D0.

CS.WKBCHIDE - Currently selected normal keyboard channel identifier.

#### 2.3.5.8.4 Function

The subroutine is provided for applications to make explicit opens of a normal keyboard channel, when such a channel has not been opened by Waiter, but is required for use outside Waiter e.g. for passing to Field Editor.

Note, that Waiter only opens a normal keyboard channel if it requires to read from such a channel. When the main Waiter routine has been entered with CS.WRFLAG = 2, and has successfully read a key using READ MENU KEY, a normal keyboard channel will not have been opened.

The keyboard channel will be the currently selected channel, whether it was open on entry or not.

The routine exits immediately with an OK reply if a normal keyboard channel is already open (CS.WKBCHIDE non-zero).

#### 2.3.5.8.5 Registers Destroyed

All registers except D0 are preserved on exit.

2.3.5.8.6 Waiter Parameter Block

This must be in a segment that is frozen or immobile on entry to the subroutine. It is not thawed by the subroutine.

## 2.3.6 FURTHER INFORMATION

### 2.3.6.1 Foreground Events

a) The 'Foreground Allocated' Event

When the subsystem is called to write a display to the screen it cannot start to do so until the 'Foreground Allocated' event has been received, since the application does not actually own the title to the screen until this time. This wait is regardless of any initial delay, or lack of it.

The initial action of the subsystem is therefore simply to await this event i.e. when entered with action value variant CS.WMNO.

Similarly if, say, re-entry is made with action value variant CS.WMN1, no display will be started until the event has occurred. If the event has been intercepted by the calling application, then before entry the event bit should be set in CS.WEVENTS, to prevent the subsystem from waiting for ever.

b) The 'Suspend Foreground' Event

When Waiter detects the 'suspend foreground' event the calling application should carry out any essential closing down action, followed by a call of CS WSUSP. The application may re-enter Waiter at entry point CS.WMN1, to wait for the 'foreground allocated' event and continue where it left off. Note that this re-entry will require the 'suspend foreground' event bit in CS.WEVENTS to be set still, and will clear the bit after using it.

When re-entry is not required, the calling application should clear the 'suspend foreground' event bit in CS.WEVENTS before making any new entry to Waiter at entry point CS.WMNO.

Similarly, if the calling application receives notification of the 'suspend foreground' event itself, it should set/clear the event bit in CS.WEVENTS depending on whether the next entry to Waiter is at CS.WMN1/CS.WMNO respectively.

### 2.3.6.2 Kernel, Application Handler and Text Expander Errors

In general, these are passed straight back to the calling application, since there is nothing that the subsystem can do with them. (Since most of them "can never happen", the calling application will presumably often abandon itself.) They are passed back to the calling application as negative return codes in D0, unaltered, so that the calling application can test them using the normal Kernel, Text Expander and Application Handler error values. One such value, ERR.UV, is not in fact an error, but signifies that an event has occurred that is not one of the two used by the local subsystem for I/O control.

Three Application Handler/Kernel error and device warnings are not passed back since they can happen legitimately - ERR.BO, ERR.NB and data lost.

When an invalid key is pressed or when buffer overflow occurs due to a user pressing too many keys, a 'bad key' sound is generated; a further key may then be pressed.

### 2.3.6.3 Screen Modes

The Waiter subsystem is transparent to the channel display modes in use. When the screen channel is opened, the screen mode specified in the Parameter Block is established. The subsystem does not thereafter set, change, adjust or enquire after the channel display modes. (Kernel preserves the channel display modes in the channel control area, which persists across temporary losses of the foreground. Application Handler provides a way to set the initial hardware mode when the application first requests foreground and then preserves the hardware mode when foreground is lost, reinstating it when foreground is regained.)

### 2.3.6.4 Windows

The Waiter subsystem takes no account of windows. The initial window size/placement is determined by Kernel when the channel is opened. The calling application may change the window definition, as desired. Neither does the subsystem set or interfere with the boundary condition settings for the window. All window manipulation is a matter between the calling application and Kernel.

### 2.3.6.5 Use of Local Events by Subsystem

Two local events are used by the subsystem for the control of I/O. These are currently defined in INCLUDE file CSWDEFS.DG as 14 (keyboard) and 15 (screen) and may be referenced by CS.WENKB, CS.WENSC (for event numbers) and CS.WEVKB, CS.WEVSC (for the event bits).

It is guaranteed that the screen event will never occur in the calling application after a return from the Waiter subsystem as a result of any operation undertaken by the subsystem. It may be used for a different purpose in the calling application (but should never be allowed to occur in the subsystem as a result of an unconnected operation in the calling application). Specifically, this is guaranteed after an "unexpected event" exit to the calling application.

A similar guarantee is given on the keyboard event when an exit is made from the main Waiter routine except when READ REVIEW KEY has an outstanding read (CS.WRFLAG=3) or CS.WEFLAG=1 on entry and the display is complete without a key having been read. A keyboard event may occur after exit from the routine CS WREAD, when called directly by an application.

### 2.3.6.6 Relationship with Field Editor

Waiter is intended for the display of essentially fixed information, often from ROM, with the ability for the user to input control information to the application, e.g. a menu selection key. Field Editor is intended for the display of a variable data field with the ability for the user to alter the data which has been displayed. An application may well wish to display a screen using the Waiter with its embedded Text Expander calls, and then process windows within the screen by use of Field Editor.