KENT ON-LINE SYSTEM

Document: KOS/AAA/2

Sub-system writer's manual

P.J. Brown
University of Kent at Canterbury
December 1970

TABLE OF CONTENTS

Introduction

One of the principal aims of KOS is that it should be open-ended, i.e., it should be as easy as possible for users to add their own sub-systems. This manual is for users who wish to do this. Obviously, before attempting to write a sub-system, the user must be familiar with using KOS, with its terminology and with the general principles that govern its operation. It is also necessary to know NEAT.

Chapter 1 Hardware considerations

1.1 Basic features

KOS consists of a master control program (MCP) which controls a number of slaves, one slave for each KOS job stream. Slave programs run under a special hardware mode called slave mode. This is similar to the normal mode of working (executive mode) except that addressing is performed differently and there is protection against slaves interfering with one another. The hardware is described in full detail in Volume 1, Part 3, Section 3 of the 4100 Manual, but the user need not concern himself with all the details of this, since some of it is concerned with the scheduling aspects which are performed by MCP.

The hardware contains a very useful facility, called the common program feature, which allows a single slave program to be shared by any number of slaves. KOS sub-systems should be written as common programs. They must be coded in NEAT since currently there is no other way of producing a common program.

The common program hardware works as follows. Each slave has its storage area which is described by a base and range. Every time an address reference is made by a common program the base is automatically added. Thus if a common program executes the instruction

```
LD  200
```

when the value of the base is 8192, then it will load the slave's relative location 200, which is really location 8392. Similarly the same program could be used with another slave whose base was 4096. In this case the address would be taken as 4296. All variables associated with a common program must be in the slave's storage area, since they will in general have different values for each slave currently using the common program. KOS takes care of setting up slaves, fixing the bases, etc., and sub-systems will not be aware of which slave they are running.

The range that the hardware associates with a slave is used to check that the slave does not upset anything outside its own storage area. Each address is compared with the range and if it is greater the slave is trapped (see Section 1.5).

Common programs are distinguished from ordinary slave mode programs by the fact that the S-register has bit 17 set. This fact should not worry the sub-system writer since the sub-system will always be entered with bit 17 of S set and none of the usual NOVAL instructions will upset it.

Ordinary slave mode programs have bit 17 of S zero. They reside in the slave's storage area and thus belong to one and only one slave; the base is automatically added to S in the same way as for addresses. Otherwise they are similar to common mode programs. Ordinary slave mode is used for such things as compiled code.

1.2 Use of storage by KOS

KOS divides the slave storage area into two parts:

(a) The slave fixed locations. The first 800 or so locations, which are reserved for fixed purposes.

(b) The user's workspace. The remaining slave storage, which is allocated dynamically for files and for sub-systems when they need extra storage (e.g. for large arrays, stacks, lists, etc.).

Of the slave fixed locations, locations 100-127 and 300-499 are reserved for sub-systems to use as they please. These are called the sub-system fixed locations. Normally 300-499 are used for variables and 100-127 are reserved for extra-codes or for some other special requirement. Thus the variables for a sub-system are normally declared as follows:-

```
        DATA
        LOCATE  300
VAR1
VAR2
  .
  .
  .
  .
```

Normally a common program uses constants as well as variables. Although each slave must have its own copy of the variables, since these will normally differ for each slave, it is clearly wasteful for each slave to have its own copy of the constants, which will, of course, have the same value for each slave.

To allow for the sharing of constants (i.e. common data) the

hardware provides the following feature: if bit 21 is one in an address reference, the slave base is not added on (nor the range checked) but the address is taken as absolute. To facilitate this KOS sets a fixed location (called £UBIT21) in each slave area to contain the value where bit 21 is one and the remaining bits zero. Hence if a common program contains the instructions

```
LDR   £UBIT21
LD:M  456
```

the value of absolute address 456 will be loaded. Note that it is necessary to use modified (or possibly indirect) addressing when referring to absolute addresses since direct addressing only allows for 15 bits.

Sub-system writers should not construct their own bit 21 nor place bit 21 in constants, but should always use the fixed location £UBIT21. This is because £UBIT21 sometimes contains an extra adjustment to make sub-systems run under DES-2.

As an example, assume that a common program wishes to use a constant called PI. PI would be declared under CONST or FCONST in the ordinary manner. As a result of this one copy of PI would be assembled into an absolute location in the program area in the normal way. When a reference was made to PI it would be done thus

```
LDR     £UBIT21
FL:M    PI
```

A straight reference to PI, such as

```
FL      PI
```

would be incorrect since the hardware would add the slave base to the absolute address of PI.

In this manual the term absolute will be used to describe a pointer or address that contains £UBIT21 and refers to the CONST area; the term relative will describe an address or pointer that refers to the slave's storage area, since the base is added to such pointers or addresses by the hardware. In general, however, the word "relative" will normally be omitted and when "pointer" or "address" are used they should be understood to be relative.

1.3 Notes on addressing

(a) The contents of absolute addresses, referenced using £UBIT21, may only be looked at. The contents must not be changed. If an attempt is made to do so the hardware forces a trap.

(b) V-literals, if used, must be treated exactly as constants, i.e., £UBIT21 must be used.

(c) Signposts and interchapter jumps require special action and it is usually best not to use them.

(d) If variables are to have initial values, or if for some reason it is desired to set up a constant in the slave area, the values must be set up dynamically at the start of the program.

1.4 How to write a common program - a summary

Common programs are very easy to write as they differ so little from ordinary NEAT programs. Simply collect variable declarations together and add

LOCATE 300

after DATA, and use £UBIT21 when referring to constants and V-literals.

Appendix E contains an example of a simple KOS sub-system, which is, of course, a common program.

1.5 Logical errors

Some faults in slave programs cause a trap thereby stopping execution of that slave. Such faults include an address (or an s-value in an ordinary slave mode program) out of range, the issuing of an I/O instruction or an attempt to change bit 19 of the C-register. A trap is called a logical error in the slave. On encountering a logical error, MCP prints an appropriate message on the control teleprinter and stops.

The subject of debugging is covered in Chapter 3.

Chapter 2 Interface with KOS

2.1 COMMAN and UTILIT

In addition to any common programs that are in use as a result of a sub-system being called, KOS contains two common programs for the basic control of the system; these two programs are always in core. They are

(a) COMMAN. This is roughly the KOS equivalent of BATCH. It recognizes and decodes commands. In addition it actually executes all KOS commands that are not part of sub-sytems.

(b) UTILIT. This is a collection of routines (called UTILIT routines) that are needed by sub-systems (and by COMMAN) including I/O, storage allocation, command decoding, etc. UTILIT roughly corresponds to the routines of NICE that are entered through fixed locations (e.g. device routines, £OUTNAME, £ASSEMBLE, etc.).

2.2 Specialized utility programs

In addition to UTILIT, KOS has available some specialized utility programs which are not always in core but may be asked for by any sub-system that needs them. Currently there is only one specialized utility program, namely a floating point package called UTFLOT.

Specialized utility programs will make use of some of the sub-system fixed locations, and will thus cut down the number that is available to the sub-system itself.

2.3 Interface with UTILIT

All UTILIT routines are called by a JIL to a slave fixed location. In addition to the slave fixed locations used for its entry points, UTILIT maintains certain quantities, called the public values, in slave fixed locations in order that sub-systems may examine them. Examples of public values are buffer pointers, line counts of I/O, £UBIT21, etc.

Since the usage of slave fixed locations is liable to change, LOCATEd symbolic names must be used for all slave fixed locations. This includes the sub-systems own variables in addition to the locations associated with UTILIT.

The names of the UTILIT fixed locations all begin with U. If the usage is concerned with I/O the second letter indicates the device concerned as follows

C means the command device.
M means the message device.
D means the data device.
R means the results device.

Thus the routine that outputs a character on the message device is called £UMCHAR.

The UTILIT routines and public values are described, under the various functional classifications, in the Chapters that follow. Summaries and tables of where names are LOCATEd are given in Appendices A and B.

2.4 Using UTILIT routines

Several of the UTILIT routines have parameters. These are passed in R and/or M. It is the usual convention that R is used for pointers and M for individual characters. Numbers and S-values may be in either R or M. In some cases UTILIT automatically adds bit 21 to R. Several UTILIT routines also produce results. These are returned in R and/or M as appropriate.

Some UTILIT routines have several exits. The routine £UDLINE (get a line of data), for example, returns to the instruction following the point of call in the error case (when there is no data left) and skips one (full-word) instruction in the normal case. A number of UTILIT routines, therefore, are said to have N exits; the first being called exit 1, the second exit 2, etc., where exit j means j full words beyond the calling instruction. When a routine is said, simply, to return, this means it uses the last exit.

The same conventions normally apply to EXENs (q.v.) and routines in specialized utility programs.

A number of UTILIT routines that have pointers as parameters (passed in R) assume that the pointer is always absolute (i.e. the item pointed at - for instance, a message or a table - is in the CONST area), and hence automatically add bit 21 to R to save the sub-system the trouble of doing so.

If, in some special case, the sub-system wishes to supply a relative pointer as parameter to one of these routines, it must subtract £UBIT21 from R.

2.5 Communicating with MCP

In some cases it is necessary for a sub-system to communicate directly with the master control program. This is done by the EXEN instruction in the hardware, which forces a trap. The address field of the EXEN indicates the operation required. The permissible operands are all LOCATEd since their numerical values are liable to change. The EXENs that are available are described in subsequent chapters and a complete table of them is given in Appendix C. One EXEN that is useful in debugging is £LOGERR, which forces a logical error and sets up information for a post-mortem dump. Due to a defect in NEAT, EXENs have to be written as N:750/exen name in the address field. (See example in Section 3.4).

EXENs may have parameters, results, multiple exits,etc., just as UTILIT routines.

2.6 Entering and exiting from a sub-system

A sub-system is entered by typing its name when in command status in KOS. The name is normally followed by a tofrom (see KOS User's Manual) to specify the data and results devices to be used. COMMAN contains a table of all the sub-system names that it recognizes as commands in themselves (e.g. BASIC, DESK). If the writer of a sub-system wishes its name to be added to this table, he should contact the writer of this manual. Otherwise the sub-system can always be entered using the ENTER command.

The first three instructions of each sub-system are entry points from COMMAN. They are as follows:

(1) Initial entry. When COMMAN initially enters a sub-system it enters it at its first instruction. The input buffer pointer (£UIBFPT, see Chapter 4) is set to point at the first character of the argument list of the entry command.

(2) Break entry. When a break is accepted when within a sub-system, COMMAN enters the sub-system at its second instruction. For a description of how sub-systems can arrange to allow or inhibit breaks, see Chapter 6.

(3) Get-off entry. If a sub-system is entered at its third instruction it must immediately release all its resources and surrender control to COMMAN.

The first instruction of a sub-system (i.e. the initial entry point) must always be a "JF 4" instruction. This is used as a marker to indicate to the ENTER command that the program is indeed a KOS sub-system.

Thus a sub-system might commence as follows:-

```
CODE
 JF     4
 JF     *BREAK
 JF     *ABORT
```

An exit is made from a sub-system by obeying the instruction

```
 JI     £UEXIT
```

2.6.1 Messages on entry and exit

A sub-system should output an appropriate message when it is successfully entered initially and again when it exits. The introductory message should identify the sub-system (including which version it is) and make it clear it is starting from scratch. Typical introductory messages might be "DESK CALCULATOR (VERSION KNL3A) READY TO START", "I AM A CHESS PLAYING MACHINE (VERSION KNL1B). TO START THE GAME ....." Typical messages on exit might be: "EXIT FROM DESK CALCULATOR", "GAME TERMINATED". The introductory message should normally be output when all the basic extras have been borrowed (see Section 3.5) and the exit message immediately before going to £UEXIT.

2.7 Naming syb-systems

Any identifier of up to eight characters can be used as a sub-system name, except that names beginning with "UT" are reserved for sub-modules and specialized utility programs. If a name begins with "UT", KOS does not update its count of job steps (see "How to run KOS") when it is entered.

Hence if a sub-system consists of several modules which are loaded separately then the name of the main module should not begin with "UT" but the names of the remaining modules should.

When a sub-system is to be used it should be entered by means of the ENTER command; the only exceptions to this are popular sub-systems like BASIC whose names have been added to a special table in COMMAN to make them global KOS commands.

2.8 Looking at MCP fixed locations

Some very specialized sub-systems, particularly those concerned with timing or monitoring, may wish to look at MCP and the fixed locations of the world it operates in (i.e. a DES-2 slave or the DES-1 or T30C system). These fixed locations must be addressed using a public value called £UKBIT21 in an exactly similar manner to the way £UBIT21 is used to address the CONST area. Usually £UKBIT21 and £UBIT21 will have identical values, but differences are possible in the DES-2 environment.

Lists of MCP fixed locations are not publicized, but they can be obtained from the author.

Chapter 3 Use of extras

3.1 Extras

Sub-systems have available to them a collection of extras, i.e. supplementary facilities for those sub-systems that need them. The extras that are available are: results and data devices, blocks of contiguous storage taken from the user's workspace, and specialized utility programs. When a sub-system is entered it has no extras; the extras that are needed can be borrowed dynamically. They must, however, always be returned before the sub-sytem exits, irrespective of how the exit is made. This Chapter describes how extras are borrowed and returned.

3.2 Data and results devices

Most entry commands and some supplementary commands allow a tofrom as the last argument. There is a routine in UTILIT called £USETDEV to decode tofroms (including null tofroms) and allocate the necessary I/O devices. £USETDEV must be called with £UIBFPT pointing at the tofrom (see Section 4.4). This will always be the case on entry to a sub-system provided that there is no argument between the sub-system name and the tofrom and provided no input is requested (e.g. by a question-and-answer) before £USETDEV is called. £USETDEV has a parameter in M, which has two possible values

(1) zero means get results device only.
(2) two means get both data and results devices.

£USETDEV has two exits. If the tofrom is syntactically incorrect, or if the required devices are not available, £USETDEV allocates no devices to the sub-system, outputs the appropriate messages and uses exit 1. Otherwise it allocates the required devices, sets the public values describing them (see Section 4.4.2 and Appendix A) and uses exit 2.

It is possible to call £USETDEV several times within a sub-system (e.g. for the entry command and then for subsequent supplementary commands), but before an attempt is made to borrow a new data/results device the previous one(s) must have been returned.

£USETDEV should be called with breaks allowed (since the user may be made to wait for a device) and it always returns with them allowed.

The data and/or results devices are returned by calling £UDREND. £UDREND performs the following actions:

(a) it clears the message buffer (see Section 4.3.1).

(b) if a results device is currently borrowed, it clears its buffer, closes the device (see User's Manual) and returns it.

(c) if a data device is currently borrowed, it closes it and returns it.

It is quite legal to call £UDREND as a safety measure even if no devices turn out to be currently borrowed.

£UDREND is automatically called by UTILIT whenever a break occurs or when either of the routines £UEXIT or £UCLINE (q.v.) is called; this means, in fact, that the data and results devices are automatically returned whenever they need to be, namely on exit from a sub-system or when a new command is input, and so it is not often necessary for a sub-system to call £UDREND explicitly.

3.3. User's workspace

The routines £UBSPACE and £URSPACE are used for borrowing and returning blocks of storage from the user's workspace.

£UBSPACE has one parameter (in R) and two exits. It returns a result in R.

£URSPACE has one parameter (in R), only one exit and no result.

Breaks must be inhibited when either routine is called, and the routines will leave them inhibited on return. It is possible to borrow any number of blocks of workspace of any size, subject to available space.

3.3.1 Borrowing user's workspace

£UBSPACE is used to borrow user's workspace. The parameter specifies the size of the block required. This must exceed zero.

If the requested amount of workspace is available, £UBSPACE allocates it to the sub-system and uses exit 2. The result in R is a pointer to a block of workspace of the size requested or perhaps one word larger. The second word of the block will contain its actual total size.

If there is not enough room, £UBSPACE uses exit 1. In this case the result in R is a pointer to the largest available block (with its size in the second word). The block is not, however, allocated to the sub-system. Hence to obtain the largest available block of workspace, the following instructions should be executed:

```
LDR:L    32767          Impossibly large request.
 JIL     £UBSPACE
 JF        2            Not available.
 JF        6            Is available!
LDR:M      1            Load size of largest
                        available block.
 JIL     £UBSPACE
         N:750/£LOGERR  Should be impossible.
Preserve R, etc.
```

If a sub-system is such that it must have a certain amount of workspace, there is a UTILIT routine call £UNOROOM which is useful for the case when that workspace is not available. £UNOROOM outputs the message

```
NOT ENOUGH WORKSPACE - BREAK
```

and then enters the sub-system exactly as if a break had occurred. This applies even if breaks are inhibited. Thus a calling sequence for £UBSPACE might be

```
LDR:L    1ØØ
JIL      £UBSPACE     Ask for 1ØØ words.
JI       £UNOROOM     Not available.
```

The internal workings of £UBSPACE have been designed to minimize the fragmentation of user's workspace.

3.3.2 Returning user's workspace

£URSPACE is used to return user's workspace. The parameter in R must point to the block to be returned. The length of the block must be in its second word. (Hence when workspace is used it is best to leave the first two words intact.) All workspace must be returned before exiting from a sub-system.

3.4 Specialized utility programs

Specialized utility programs are borrowed and returned by communicating directly with MCP by means of the EXENs £BORPRG and £RETPRG.

To borrow a program £BORPRG is used. It has two exits. The parameter in R is an absolute pointer to the name of the program to be borrowed. However, bit 21 should not be included in R since it is added automatically. The name should be stored in two words of packed internal code characters, padded with blanks on the end. If the program is not available, £BORPRG uses exit 1. If it is available, £BORPRG uses exit 2 and gives the S-value of the start of the program (with bit 17 one) as the result in M. In either case breaks are inhibited on return, and the contents of R are not changed from their calling value.

To cater for the error exit from £BORPRG, there is a UTILIT routine £UNOPRG, which outputs the message

program name NOT AVAILABLE (NOW)
(THIS SESSION)

and goes to £UEXIT. However, £UNOPRG should only be used when there is no workspace and no other programs to be returned.

A typical calling sequence for £BORPRG would be

```
        CONST
UTNAME          C:UTFL
                C:OT
          .
          .
          .
          .
          .
```

```
LDR:L   UTNAME          R points at name.
        N:75Ø/£BORPRG
JI      £UNOPROC        not available.
ST      UTENTRY         save S-value.
        :
JIL     UTENTRY         enter program.
```

Specialized utility programs are returned using £RETPRG. The parameter in R should contain the S-value of the program to be returned, exactly as it was supplied by the corresponding £BORPRG. Hence continuing the example above, the returning sequence would be

```
LDR     UTENTRY
        N:75Ø/£RETPRG
```

3.5 Ordering requests for extras

Ideally a sub-system should start by borrowing any necessary specialized utility programs. It should then borrow its workspace. Up to this point breaks should remain inhibited, as they are on entry to a sub-system (see Chapter 6). Finally, it should allow breaks, borrow its devices, and if this succeeds, and only then, it should output its introductory message (see Section 2.6.1). Sometimes it may not be possible to follow this sequence, for instance it might be desired to ask the user how much workspace he needs. However, in all cases it should be ensured that, however an exit is made, exactly those extras that have been borrowed are returned. Great care should be taken if an exit is caused by a break.

3.6 Running in executive mode

There is a facility for sub-systems to switch into executive mode in order to do something that is impossible in slave mode. (By "executive mode" is meant the level above the usual KOS slave mode, i.e. the level at which MCP operates; this may, in fact, itself be slave mode relative to DES-2.) This facility is fraught with dangers and should only be used as a last resort. When running in executive mode the program is untimed, unbreakable and unprotected. Executive mode code should therefore be designed so that it does not do any of the following things whatever errors may occur:

(a) use a device that does not belong to it.

(b) get into an endless loop.

(c) execute an error exit such as

```
JI    £END
```

Executive mode is entered by the following EXEN command

```
N: 750/£SWEXEC
```

This returns at the next instruction with R and M unchanged. To return to slave mode the instruction

```
JIL   £MTOSLAV
```

should be obeyed. This also returns at the next instruction with R and M unchanged. (In actual fact MCP goes down the scheduling queue before returning to the original program. This prevents executive mode programs getting too big a share of the available time.)

The following example shows how executive mode could be used to punch an undecoded character on paper tape. The paper tape punch must have been borrowed before this code is entered.

```
LD      CHAR
        N:750/SWEXEC
LDR:L   3
JIL     £POUT
JIL     £MTOSLAV
```

Note that when a program is in executive mode its view of the world is completely different from in slave mode. In particular, when in executive mode, slave fixed locations cannot be addressed directly. The CONST area should be addressed using modified addressing relative to a location called £MBIT21, rather than the usual £UBIT21, where £MBIT21 is a fixed location provided by MCP. MCP provides some fixed locations for use as temporary variables by executive mode programs. These are described in Appendix C. Note, however, that their values will not remain intact between one executive mode entry and the next. Executive mode programs should be very wary of changing any locations other than these MCP fixed locations and location 0.

Chapter 4 Input and output

4.1 Introduction

A sub-system will, in general, be completely unaware of which I/O devices it is using. It will perform all its I/O on a line-at-a-time or character-at-a-time basis using UTILIT routines.

Devices are automatically opened (e.g. supplying of run out at start of paper tape, form feed on printer) before they are first used; devices are closed when £UDREND is called.

Note that the command and message devices always exist but the data and results devices only exist if they have been successfully borrowed and not subsequently returned (either explicitly by £UDREND or implicitly by, for example, a break). An attempt to use a non-existent device causes an immediate logical error.

4.2 Character codes

Apart from the routines for the output of messages (see £UMSTR, etc.), all UTILIT routines work in single characters or lines of characters in KOS 7-bit code stored one to a word in a buffer. KOS 7-bit code is a sub-set of the standard 4100 7-bit code (i.e. internal code with 64 added for out-shift characters). The characters omitted from the standard code are carriage return, null and halt code.

KOS uses the standard 4100 device routines to convert from external representation to KOS 7-bit code. Whatever the input medium, each line is terminated with a newline character (internal code 2) when the line is converted to KOS 7-bit code. Characters not in the KOS 7-bit code are ignored on input, and are automatically supplied on output when the physical device warrants them. The way this is done is described in the KOS User's Manual and in device routine specifications. Tabs are treated just as any other character by KOS; the way they are represented externally is determined by the device routine for the device that is used.

Where not otherwise stated, the word "character" should hence-forward be taken to mean "KOS 7-bit code character".

4.3 Output

The sets of routines for output on the message device are exactly similar to those used for the results device, so they are described in pairs below.

UTILIT maintains two buffers, one of 126 characters for the current line of results and one of 67 characters for the current line of messages. Characters are added to these buffers until a newline is supplied, and then the whole line is output. If either of the buffers is about to overflow (i.e. a character other than a newline is fed to the buffer when it is already full), the newline is automatically supplied in order to output the buffer and the next character is placed at the start of a new line.

4.3.1 UTILIT routines for output

The following are specifications of the output routines in UTILIT.

A. £UMCHAR, £URCHAR

Output the character in M.

B. £UMVAL, £URVAL

Output, as a decimal number, the binary value in M. Redundant leading zeros are suppressed and the sign is printed only if the number is negative. No extra spaces are supplied either at the beginning or the end of the number.

C. £UMSTR, £URSTR

Output the packed 6-bit code string pointed at by R. (Packed 6-bit code means the standard 4100 internal code, with characters packed four to a word. The same characters are permissible as in KOS 7-bit code.) It is imagined that the string will always be in the CONST area so R is an absolute pointer. However, it should not include £UBIT21 as this is automatically added by UTILIT. The string may include shift characters. It must end with either a newline (in-shift 2) or an end marker; the latter is represented by out-shift octal 15. If the string ends with a newline, this is output as part of the string. If the string ends with an end marker, the end marker is not taken as part

of the string; no newline is supplied and so subsequent characters that are output on the same device follow on the same line. Since a newline ends a string, it can be seen that it is only possible to output one line at a time.

(There is a special facility which applies if an end marker occurs other than at the end of a word. In this case the next pair of octal digits is examined. Let N be the value of this octal number. Then the previous N characters of the string preceding the end marker are deleted, for example

```
C:ABCD
C:EXXX
0:7615Ø3ØØ
```

means the string "ABCDE" since N has value 3. The purpose of this feature is to avoid having to write characters as octal constants in NEAT. The feature will not work properly if the characters to be deleted overflow the buffer. In general end markers should be padded to the right with zeroes so that the above feature is not used by mistake.)

It is not required that a string occupy an integral number of words; apart from the special facility described above anything beyond its terminating newline or end marker is ignored. It is always assumed that a string starts in in-shift, irrespective of any preceding output.

The word string will henceforward be used to mean a string of characters fitting the above specification.

D. £UMCSTR, £URCSTR

Similar to £UMSTR and £URSTR except that the string is placed at the start of a new line. This is done by outputting the appropriate buffer if it is not initially empty; this is called clearing the buffer (hence the 'C').

E. £UILCOM

£UILCOM is a very specialized UTILIT routine. It prints the message "ILLEGAL COMMAND" and returns.

F. £UMLVAL, £URLVAL

Similar to £UMVAL and £URVAL except that the parameter in M must be in the range 1 to 9999 and this number is output followed by a dot and enough spaces to make five characters in all. (This is the KO3 format for line numbers.)

4.3.2 Example of output instructions

```
        CONST
LNMS            C:LINE
                0:ØØ7615ØØ          Space + end marker.

OUTMS           C: OF
                C:OUTP
                C:UT"               In card NEAT format.
        .
        .
        CODE
        .
        .
        LD:L    C:   X
        JIL     £UMCHAR
        LDR:L   LNMS
        JIL     £UMCSTR
        LD:L    6
        JIL     £UMVAL
        LDR:L   OUTMS
        JIL     £UMSTR
```

would output the messages

***X

***LINE 6 OF OUTPUT

(The asterisks at the start of messages are automatically supplied by UTILIT.)

4.4 Input

The most important thing to remember about the input routines is that there is a single input buffer (of 126 characters). Thus if a line of data is read, this will overwrite any command or answer to a question-and-answer that was left in the buffer, and vice versa.

The input buffer is described by the three public values £UISIGST, £UIBFMAX and £UIBFPT. £UISIGST points at the first significant character in the buffer (i.e. the the first character the sub-system should look at - this is the character following the "&" for a command, the answer

to a question, the first significant column for card input, otherwise normally the first character in the buffer), £UIBFPT points at the next character to be scanned and £UIBFMAX points at the newline at the end of the buffer. These pointers are set every time a line is read, the initial value of £UIBFPT being the same as £UISIGST. In addition, the routines that scan the input buffer start their scanning at £UIBFPT and update £UIBFPT to the end of their scan; routines in this category are £UDECODE, £USETDEV, £UDCHAR and £UIBCHAR. A sub-system can, if it wishes, change the value of £UIBFPT but care should be taken if any of the above routines are being used on the same buffer.

Commands and data are automatically listed by UTILIT when the circumstances warrant (see KOS User's Manual).

4.4.1 Input of commands

The routine £UCLINE is used to input commands. It reads a line from the command device and returns. There is no error exit as it always continues to request input until it gets a command. The "&" preceding the command is not taken as part of the command and £UIBFPT and £UISIGST will be set to point at the character beyond it.

£UCLINE automatically calls £UDREND to return any devices associated with the preceding command.

4.4.2 Input of data

Data can be input using £UDCHAR or £UDLINE. Each of these routines has two exits, the first exit being an error exit for the case where data has been exhausted. The data terminator (full stop on a console, halt code on paper tape) is not taken as part of the data and directly it is encountered the error exit is taken. (However, the device is not automatically returned when the terminator is found; it is only returned when £UDREND is called. This is because returning a device often produces a message and it might upset the output format to output this message at the wrong time. Both £UDLINE and £UDCHAR "stick" at the data terminator and will continue to give the error exit on all calls until the data is replenished by £USETDEV.) If a command is encountered when an attempt is made to input data, the error exit is taken but input is "backspaced" so that the next £UCLINE reads the same command.

If a pseudo-command is input from the data device, UTILIT automatically processes it and inputs the next line. Thus sub-systems do not need to worry about them.

Four public values can be used to control the reading of data. They are initialised by £USETDEV, but may subsequently be changed by sub-systems. The variables are as follows:

Name	Initial value	Meaning
£UDLINO	0	Line number of input data, updated each time a line is read - useful for error messages.
£UDCDST	1	Starting column if data is from cards.
£UDCDEND	72	Last column if data is from cards.
£UDLIST	0	(2 means list with line numbers. (1 means list. (0 means don't list.

(The value of £UDLIST is controlled dynamically by the DLIST and DUNLIST commands and the LIST and UNLIST pseudo-commands.)

The exact specification of the two data input routines is as follows.

£UDLINE tries to input a line of data. If it succeeds it uses the second exit; if it fails it uses the first.

£UDCHAR tries to input a single character of data. If it succeeds it places the character in M and uses the second exit; if it fails it uses the first. (Except when it needs to input a new line, £UDCHAR simply performs the action:

```
LD:I      £UIBFPT
COMP:L      2          NEWLINE
JZ          2          NEVER ADVANCE £UIBFPT BEYOND
INCS      £UIBFPT      TERMINATING NEWLINE
```

Although £UIBFPT is not updated when a newline is encountered, a special marker is set so that on the next call to £UDCHAR a new line is input.)

4.4.3 Questions-and-answers

£UCQLINE is the UTILIT routine for command questions-and-answers and £UDQLINE is the corresponding routine for data question-and-answers. The former has a parameter in R and the latter has parameters in both R and M. Both routines have two exits. £UCQLINE sets up the question in the message buffer and £UDQLINE sets up the question in the results buffer.

In both cases the question must fit into the buffer. In most uses of £UCQLINE and £UDQLINE the question is a predefined string. In this case the parameter in R points at the string, in a similar way to the parameter to £UMSTR. The appropriate buffer is then cleared and the string is then copied into the buffer and taken as the question. The string must be terminated with an end marker (and therefore it cannot contain a newline). An alternative way of using £UCQLINE and £UDQLINE is to place the question in the appropriate buffer in advance and then to call the routine with R zero. This is useful if the question contains some variable element.

To avoid confusing the user, questions should never begin with an ampersand or colon. They should normally end with an equals sign. (There is no question mark character on a 4130 console.)

The parameter in M to £UDQLINE determines whether the question is to be compulsory. Zero means compulsory and one means optional.

In all cases exit 1 is used if the question is unmatched. (This can only occur in the non-conversational case.) If the question is optional the input medium is "backspaced" so that the same line is read again on the next request for a line from the device (in fact it is even possible to try to match the line with a different question); in the compulsory case an error message is produced and no backspacing takes place.

Exit 2 is used in all other cases. The answer is placed in the input buffer, in the same way as for £UCLINE. The routine £UDECODE (q.v.) is very useful for decoding the answer. The action of a sub-system on getting an unsatisfactory answer should normally be to repeat the question.

Unlike £UCLINE, £UCQLINE does not release the data device. However it must not be called when the data device is in a "backspaced" state as described above since this would corrupt the buffer. (Hence KOS forces a logical error if this happens.) Therefore £UCQLINE must not be called immediately after a call of £UDQLINE with an optional question.

One last point. If £UDCHAR is used to scan ordinary data immediately after £UDQLINE, it is necessary to perform an initial call of £UDLINE. This clears out any data left in the buffer after £UDQLINE.

4.4.4 Example of use of £UCQLINE

Here the question is "CONTINUATION=" and the answer must be "YES" or "NO".

```
        CONST
CNTMS           C:CONT
                C:INUA
                C:TION
                0:35761500              Equals plus end marker.

        NOTE    DECODE TABLE -SEE CHAPTER 5
YNTAB           3
                C:  Y                   If YES,
                C:  E
                C:  S
                2                       ...use exit 2.

                2
                C:  N                   If NO,
                C:  O
                4                       ...use exit 3

                -4                      If anything else,
                0                       ...use exit 1.

        .
        .
        CODE
        .
        .
        .
ASK     LDR:L   CNTMS
        JIL     £UCQLINE
        JF      ERROR                   Error exit.
        LDR:L   YNTAB
        JIL     £UDECODE
        JB .    ASK                     Unmatched answer.
        JB      CONT                    Answer YES.
        ---                             Answer NO.
```

As a second example, if the line labelled ASK and the line that follows it were replaced by the lines

```
ASK     LD      CHAR
        JIL     £URCHAR
        LD:L    C:  =
        JIL     £URCHAR
        LD:L    0                       Compulsory question.
        LDR:L   0                       Use buffer as question.
        JIL     £UDQLINE
```

where the variable CHAR contained the letter "C", then the effect would be identical to the above except that the question would be "C=" and it would be a data question-and-answer rather than a command one.

4.4.5 Input of characters from the buffer

The routine £UIBCHAR can be used to get the next character from the input buffer, irrespective of the nature of the line in the buffer. £UIBCHAR has 3 exits, depending on the nature of the character found.

Exit 1 is used if the character is a terminator (newline or semicolon).

Exit 2 is used if the character is a separator (tab, space or comma).

Exit 3 is used otherwise.

The character is returned in M and £UIBFPT is increased by one, except when the first exit is used. It can be seen, therefore, that £UIBCHAR never reads beyond the terminator of the current line.

4.5 Identity of I/O devices

It is possible for a sub-system to find out what physical devices are in use by examining the four public values : £UCDVB, £UDDVB, £UMDVB, £URDVB. These contain the KOS device number (see "How to run KOS") of the command, data, message and results devices. The value 500 currently means that the device does not exist , a value exceeding 600 means that the device is a job file, and a negative value means a DC.

It is sometimes useful to examine if certain devices are the same. For example if £UCDVB equals £UMDVB, KOS is in conversational mode, and if £UDDVB equals £UMDVB there is no point in giving a line number in an error message (similarly if £URDVB equals £UMDVB and £UDLIST exceeds zero - if the reader can work that one out).

It is very bad practice to test the individual values of the various device numbers since one of the central principles of KOS is that it be device-independent. Moreover, the device numbering system is likely to change when KOS is extended.

Chapter 5 Decoding of Input

UTILIT contains a very important routine, called £UDECODE, which is useful for implementing supplementary commands, dealing with arguments to commands, processing answers to questions-and-answers and, in some cases, processing data.

£UDECODE works on the set of characters in the input buffer, beginning with the character pointed at by £UIBFPT. It first advances £UIBFPT if necessary to scan over separators (commas, spaces or tabs) until £UIBFPT points at a non-separator. It then scans for the next separator or terminator (semi-colon or newline). The set of characters in between is called the decodee; note that if the first character scanned is a terminator, the decodee will be null. Except where otherwise stated, on return from £UDECODE, £UIBFPT is further advanced (if necessary) to point at the first non-separator beyond the end of the decodee. This facilitates the use of £UDECODE to process several successive decodees on a line.

Example

```
        26, 35
        ↑
    £UIBFPT
```

If £UDECODE were called with £UIBFPT pointing as indicated above, the characters "26" would be the decodee and on return £UIBFPT would point at the character "3".

The action to be performed by £UDECODE is controlled by the decode table. The parameter of £UDECODE, which is in R, is an absolute pointer (but with bit 21 omitted, since this is added automatically) to the decode table.

The decode table consists of a set of contiguous table entries. The first word of each table entry identifies the type of entry. The various types of entry have different lengths. In most cases the last word of an entry is a return offset. This means that if £UDECODE matches the table entry (see later) it is to add the return offset to the link before returning to the calling program. Thus if the return offset is 2, £UDECODE will skip one instruction on its return to the calling program. A return offset of minus one is interpreted as "go back to start of decode table."

One class of table entry will match the decodee only if the decodee is of a certain form, a second class will always match, and a third will never match. Members of this third class have no return offsets; they are effectively unconditional commands to £UDECODE.

£UDECODE scans the table entries one by one, performing the action of each, until a match is found that causes it to return to the calling program. The table must end with an entry that will match any decodee (i.e. one of the entries listed in Section 5.1.2 below).

5.1 Types of table entry

The types of table entry are listed below; their names have no significance except for description and documentation. Section 4.4.4 contains an example of £UDECODE and there are other examples at the end of this Chapter. Appendix D contains a summary of the types of table entry.

5.1.1 Conditional matches

The following types of table entry match the decodee only if it is of a certain form.

A. FIND table entry

Format

N	N characters	return offset

$N \geq 0$

The N characters are in KOS 7-bit code and are stored one to a word. FIND matches the decodee only if it corresponds exactly to the given N characters.

B. GETNUM table entry

Format

-1	binary integer 1	binary integer 2	return offset

GETNUM matches the decodee only if it is, in character form, a (possibly signed) integer not less than binary integer 1 and not greater than binary integer 2. When a match is made the value of the decodee is returned in M.

5.1.2 Unconditional matches

The following types of table entry will match any decodee.

A. SYSDO table entry

Format	-2	return offset

Treat the decodee as a global KOS command. If it contains no errors it is executed; in this case SYSDO will return to the calling program at the get-off entry point if the KOS command is one that causes an exit from the current sub-system (e.g. entry commands, JOB, EXIT), and at the return offset from the point of call otherwise. If the decodee is not a correct global KOS command the appropriate error message is output and a return is made at the return offset from the point of call.

SYSDO automatically performs the action of £UDREND.

The setting of £UIBFPT after a SYSDO is indeterminate.

B. ERROR table entry

Format	-3	return offset

The message "ILLEGAL COMMAND" is output and a return is made to the calling program at the given return offset.

C. ALL table entry

Format	-4	return offset

A return is made at the given return offset.

D. RESET table entry

Format	-9	return offset

RESET is the same as ALL except that, on return, £UIBFPT is left to point to the first character of the decodee, so that it can be re-scanned if necessary.

5.1.3 Non-matches

The following types of table entry never result in a match; they therefore contain no return offset.

A. GETCOM table entry

Format

-5

£UCLINE is called. (Two features of £UCLINE should be noted in this context because of the side effects that they may have: firstly that it calls £UDREND; secondly that it resets £UIBFPT.) GETCOM can only occur as the first entry in the table; when it does, the decodee is taken from the start of the new line input by £UCLINE, not from the line that was in the buffer when £UDECODE was called.

B. ISLAST, NOTLAST, CANLAST table entries

Formats

ISLAST	-8
NOTLAST	-7
CANLAST	-6

(1) ISLAST means that subsequent conditional matches are only to succeed if the first non-separator beyond the end of the decodee is a terminator (i.e. in the case of a command line, if the decodee is the last argument or a command with no arguments).

(2) NOTLAST means that subsequent conditional matches are only to succeed if the first non-separator beyond the end of the decodee is not a terminator.

(3) CANLAST means that subsequent conditional matches are to be independent of what follows the decodee.

Any occurrence of one of these three table entries overrides any previous one. Initially CANLAST is always assumed.

5.2 Examples of decode tables

Example 1

Neat address field		Comment	
-5		GETCOM	1st Table Entry
4		FIND	
C:	C		2nd Table Entry
C:	O		
C:	N		
C:	T		
Ø		Return offset	
-8		ISLAST	3rd Table Entry
2		FIND	
C:	G		4th Table Entry
C:	O		
2		Return offset	
-2		SYSDO	5th Table Entry
-1		Go back to start of table	

This decode table recognises two commands: CONT and GO. The former may have arguments, the latter cannot. Any global KOS commands are executed without returning to the calling program. A call of £UDECODE using this table would have two exits, the first for CONT and the second for GO.

Example 2

-1	GETNUM	1st Table Entry
-32768	Lower bound	
-1	Upper bound	
0	Return offset	
-1	GETNUM	2nd Table Entry
0	Lower bound	
0	Upper bound	
2	Return offset	
-1	GETNUM	3rd Table Entry
1	Lower bound	
32767	Upper bound	
4	Return offset	
0	FIND(matches null decodee)	4th Table Entry
6	Return offset	
-4	ALL	5th Table Entry
8	Return offset	

(continued overleaf)

This decode table tests if the decodee is a negative, zero or positive integer or if it is null. A call of £UDECODE with this table would have five exits as follows:

1st exit : negative number (in M).
2nd exit : zero number (in M).
3rd exit : positive number (in M).
4th exit : null decodee.
5th exit : anything else.

Chapter 6 Breaks

6.1 Routines for breaks

UTILIT contains two routines for controlling the break status, viz:

£UALBRK	Allow breaks.
£UINBRK	Inhibit breaks.

Neither routine has any parameters nor any results. If the console user tries to break while breaks are inhibited the break is "remembered" and comes into effect immediately breaks are allowed again. When a sub-system is entered (by any of the three entry points) breaks are inhibited by COMMAN.

The action of UTILIT on a break is to inhibit breaks, call £UDREND, output the message "BREAK" and enter the sub-system at its break entry point.

A sub-system can cause a forced break when some unrecoverable condition has occurred. This is done by calling the routine £UBREAK with R pointing at a string as for £UMSTR. UTILIT performs the same action as for a user generated break except that it prefixes the message "BREAK" with the string pointed at by R. This string should, therefore, be terminated by an end marker (rather than a newline) so "BREAK" (which is preceded automatically by a space) can follow on the same line. £UBREAK is used by £UNOROOM, the message being "NOT ENOUGH WORKSPACE - BREAK". Forced breaks occur even if breaks are inhibited.

6.2 What to do at a break

The easiest thing to do on a break is simply to exit. This is, however, only really acceptable to the user in the case of a simple sub-system. In more elaborate sub-systems, which may perform several different actions in a sequence, the user would be very annoyed if a break in onestep invalidated all his previous work. For example he might have compiled a program and be in the process of testing it against several sets of data when oneset caused the program to go into an endless loop. What he would like to do is break the run without losing his compiled program. Moreover he would like to be able to examine the values of variables to try to find

out what went wrong. In other words he wants his compiled program, his dictionary and the values of his variables to survive breaks.

If a sub-system is to provide facilities such as this it must be very careful when it allows breaks. For example if a linked list is to survive breaks, breaks must be inhibited whenever it is in an unstable state, for example when the chain is momentarily broken to add or delete an item. In general, when a number of related variables describe an entity, if that entity is changed breaks must be inhibited until all the variables have been updated to describe the new state of the entity. Examples are: a string described by a length field followed by some characters, an array described by a dope vector, a stack describing the state of a program.

Hence a sub-system needs to satisfy two conflicting aims:

(a) Breaks must never be inhibited for longer than, say, a tenth of a second of computing time.

(b) If reasonable recovery facilities are to be offered, great care must be taken to inhibit breaks whenever the sub-system can be in an unstable state.

A sub-system offering recovery facilities must return to command status after a break to let the user say what to do next and in particular to exit from the sub-system if that is what he wants.

It is sometimes desirable to have different "levels" of breaks, for example

(1) a break during initialization causes an exit.

(2) a break during compiling destroys everything and returns to command status.

(3) a break during running aborts the run but some diagnostic information is output and the user can examine the values of scalar variables.

6.3 When breaks must be inhibited

Breaks are automatically inhibited by the KOS system:

(a) on any entry to a sub-system.

(b) on return from EXEN £BORPRG.

(c) after a logical error or after a job stream has been killed.

The sub-system should make sure breaks are inhibited:

(a) on a call of £UBSPACE or £URSPACE.

(b) between being entered at the get-off entry and jumping to £UEXIT.

(c) when anything that is to survive the break is in an unstable state.

6.4 When breaks must be allowed

A sub-system should allow breaks whenever possible, in particular:

(a) whenever any input is requested.

(b) whenever £USETDEV is called.

(c) at least once every tenth of a second of computing time.

6.5 Changing break status

Apart from the UTILIT routines that explicitly deal with breaks (i.e. £UALBRK, £UINBRK, £UNOROOM and £UBREAK), no UTILIT routines change the break status provided that none of the rules given in the two previous Sections is broken.

Chapter 7 Documentation

All sub-systems must have manuals written for them. There are certain conventions that must be observed in KOS documentation; these are listed elsewhere.

In the description of a sub-system, the following information must be given about its interface with KOS;

(a) The form of its entry command.

(b) A list of its supplementary commands.

(c) Its treatment of breaks; when they are allowed, what they do.

(d) Its usage of user's workspace (state if none).

The writer of a manual should, whenever applicable, use terms from the KOS User's Glossary (see Appendix to KOS User's Manual). Synonyms for these terms must not be used. Documentation for sub-system writers (e.g. specifications of specialized utility programs) can and should make use of terms in the KOS Sub-system Writer's Glossary (see Appendix F). Writers of manuals for users must not, however, make use of these terms without explaining them.

The program representing the sub-system itself must be properly commented and documented. This is vitally important.

Chapter 8 Debugging

Debugging common programs is relatively easy because they tend to stop on an addressing fault when anything goes wrong; moreover, all variables are LOCATEd and thus easy to find. The program PM (see "How to run KOS") is invaluable for diagnosing faults. Debugging runs are normally best done in the batch with card input and printer output. The UTILIT public values are often useful in interpreting dumps (e.g. buffer pointers, etc., see Appendix A).

Before its dumps, PM produces some other diagnostic information. Typically it may look like this.

1. LOGICAL ERROR IN COMMON MODE PROGRAM BASIC
2. (MONITORED AT LOCATION 22571
3. THIS IS 53 RELATIVE TO START OF MCP)
4. BASE = 30720
5. FAILED AT LOCATION 19573
6. THIS IS 249 RELATIVE TO MAIN ENTRY POINT

The information in lines 1 and 5 is derived by subtracting two from the s-value at the point of failure; occasionally this will give unusual results. If s does not contain bit 17, line 1 reads

LOGICAL ERROR IN SLAVE MODE PROGRAM

and line 6 is omitted.

When PM is dumping it prints several values to a line. If all the numbers on a line are zero, it omits the line. When one or more lines are omitted in this way, PM prints an asterisk. The storage area for each KOS job stream is zeroized when the job stream is created (on a set-up or reset entry to KOSEX - see "How to Run KOS"). This is done to reduce the size of post-mortem dumps.

If a logical error occurs during a KOS console session, a post-mortem should, if possible, be taken before KOS is set going again; if not, most of the information about the logical error will be lost.

Sometimes a logical error will occur in UTILIT or a specialized utility program. This is caused by the sub-system calling a routine with illegal parameters or under illegal circumstances (e.g. an attempt to use a device without borrowing it). UTILIT will often force a logical error using EXEN £LOGERR in such circumstances.

It is planned to add on-line debugging aids to KOS in the future.

A special KOS command is available for testing new versions of KOS programs at disc installations. This is the TRY command. TRY is identical to the ENTER command except that if the program for the sub-system to be entered is not already loaded it is loaded from the PAD on the KOS default disc rather than from the system. Hence if a new version of a sub-system called DOALL was on the PAD it could be entered by

TRY DOALL

(An alternative method would be to load DOALL statically from the PAD (see "How to run KOS") and then to use ENTER.)

Hence a good way to test a new sub-system is to write its program to the PAD using NEATERD and then to check it out using TRY before adding it to the system. Ordinary users should not be informed of the existence of the TRY command.

Appendix A List of public values

The following is a list of the public values. Sub-system writers can make use of these values but must be very wary of changing them, especially those not explicitly mentioned in the documentation. The locations in which the values are stored are liable to change.

Location	Identifier	Meaning	See Section
3	£UBASE*	base, i.e. absolute address of storage area	2.4
4	£UBIT21*	used to address CONST area	1.2
5	£UKBIT21*	used to address MCP area	2.8
128	£UDLIST	status of data listing option	4.4.2
129	£UDLINO	data line number	4.4.2
130	£URLINO	results line number	-
131	£UDCDST	first card column	4.4.2
132	£UDCDEND	last card column	4.4.2
133	£UIBFPT	points at next input character	4.4
134	£URBFST	points at start of results buffer	-
135	£URBFMAX	points at physical end of results buffer	-
136	£UMBFST	points at start of message buffer	-
137	£UMBFMAX	points at physical end of message buffer	-
138	£UDDVB	data device number	4.5
144	£UIBFMAX	points at last character in input buffer	4.4

146	£UCDVB	command device number	4.5
154	£URDVB	results device number	4.5
156	£URBFPT	points beyond last character in results buffer	-
162	£UMDVB	message device number	4.5
164	£UMBFPT	points beyond last character in message buffer	-
170) 174)	£UTEMP1- £UTEMP5	(temporaries for UTILIT. Can be (used by sub-systems between (calls of UTILIT	- -
185	£UISIGST	points at first significant character in input buffer	4.4

* £UBIT21, £UKBIT21 and £UBASE contain specially adjusted values when KOS runs under DES-2. Sub-system writers need not, however, be concerned with this.

Appendix B List of UTILIT routines

Location	Identifier	Parameters	Results	Exits	Meaning	See Section
50	£UALBRK	-	-	1	allow breaks	6.1
51	£UINBRK	-	-	1	inhibit breaks	6.1
52	£USETDEV	M	-	2	borrow D/R devices	3.2
53	£UDREND	-	-	1	return D/R devices	3.2
54	£UDECODE	R	(M)	many	decode	5
55	£UMSTR	R	-	1	output string to messages	4.3.1
56	reserved					
57	£UMVAL	M	-	1	output number to messages	4.3.1
58	£UMCHAR	M	-	1	output character to messages	4.3.1
59	£URCHAR	M	-	1	output character to results	4.3.1
60	£UDLINE	-	-	2	get a line of data	4.4.2
61	£UCLINE	-	-	1	get a line of command	4.4.1
62	reserved					
63	£URLVAL	M	-	1	output line number to results	4.3.1
64	£UMLVAL	M	-	1	output line number to messages	4.3.1
65	reserved					
66	£URVAL	M	-	1	output number to results	4.3.1

Location	Identifier	Parameters	Results	Exits	Meaning	See Section
67	reserved					
68	£UILCOM	-	-	1	"ILLEGAL COMMAND" message	4.3.1
69	reserved					
70	£URSTR	R	-	1	output string to results	4.3.1
71	reserved					
72	£UBSPACE	R	R	2	borrow workspace	3.3
73	£URSPACE	R	-	1	return workspace	3.3
74	reserved					
75	reserved					
76	reserved					
77	£UIBCHAR	-	M	3	get input character from buffer	4.4.5
78	£UNOROOM	-	-	0	no workspace - forced break	3.3.1
79	£UMCSTR	R	-	1	clear, then £UMSTR	4.3.1
80	£URCSTR	R	-	1	clear, then £URSTR	4.3.1
81	£UEXIT	-	-	0	exit from sub-system	2.6
82	reserved					
83	£UDCHAR	-	M	2	get input character	4.4.2
84	£UNOPRG	-	-	0	no program exit	3.4
85	£UCQLINE	R	-	2	command question-and-answer	4.4.3
86	£UBREAK	R	-	0	force break	6.1
87	reserved					
88	£UDQLINE	R,M	-	2	data question-and answer	4.4.3

Appendix C List of EXENs and MCP fixed locations

The following table lists the EXENs that sub-systems may use, and indicates where they should be LOCATEd.

Location	Mnemonic	Meaning	See Section
12736	£LOGERR	force logical error	2.5
12800	£BORPRG	borrow program	3.4
12864	£RETPRG	return program	3.4
13184	£SWEXEC	switch to executive mode	3.6

Note that if the above LOCATEs are used EXENs should be written using the "N:" feature of NEAT, e.g

N:750/£RETPRG

The following are fixed locations in MCP that are of use if the executive mode facility described in Section 3.6 is used.

Location	Mnemonic	Meaning
313	£MTOSLAV	subroutine to return to slave mode
480	£MBIT21	analogous to £UBIT21
481-485	-	temporary variables for executive mode programs

Appendix D List of decode table entries

1st word	Mnemonic	Description
N ≥ 0	FIND	Try to match next N characters.
-1	GETNUM	Try to match number in given range.
-2	SYSDO	Treat decodee as global KOS command.
-3	ERROR	Treat decodee as an error.
-4	ALL	Treat decodee as matched.
-5	GETCOM	Call £UCLINE.
-6	CANLAST	Something may follow decodee.
-7	NOTLAST	Something must follow decodee.
-8	ISLAST	Nothing can follow decodee.
-9	RESET	As ALL but reset £UIBFPT.

Appendix E A sample sub-system

This Appendix illustrates a complete, albeit rather trivial, KOS sub-system. Firstly a listing of the program, in NEAT, is given and secondly a sample of its actual use at a console is shown.

```
&JOB;PS/R001/TEST (DEMO);


&ASSIGN;5;DC;2;TEST;


 &NEAT;
NEATERR
ACID
MIN
MAX


 &LIST;

                        PROGRAM     TEST        KNL5A. P J BROWN JAN 3 1971
                        CHARTER     ONLY
                        BLOCK       ONLY

                        NOTE        THIS IS A VERY SIMPLE PROGRAM TO SHOW HOW A KOS
                        NOTE        SUB-SYSTEM IS WRITTEN

                        NOTE        ITS ACTION IS TO COUNT THE NUMBER OF CHARACTERS
                        NOTE        FED TO IT AND OUTPUT THE TOTAL SO FAR AT THE
                        NOTE        END OF EACH LINE.       ON ENCOUNTERING AN UPARROW
                        NOTE        IT RESETS THE COUNT TO MINUS ONE


                        NOTE        FIRST COMES THE DATA AREA
                        NOTE        ALL DATA IS STORED IN SLAVE WORKSPACE
```

```
                NOTE        I.E. EACH SLAVE HAS ITS OWN COPY

                NOTE        DATA PART 1: ENTRY POINTS TO UTILIT

                DATA
                LOCATE      50
£UALBRK                     2           50 ALLOW BREAKS
£USETDEV                    3           52 SET IO DEVICES
£UMSTR                      11          55 OUTPUT A MESSAGE STRING
£URVAL                      4           66 OUTPUT A NUMBER TO RESULTS
£URSTR                      11          70 OUTPUT A STRING TO RESULTS
£UEXIT                      2           81 EXIT BACK TO KOS
£UDCHAR                                 83 GET A CHAR OF DATA

                NOTE        DATA PART 2: VARIABLES USED BY SUB-SYSTEM
                NOTE        LOCATIONS 300 TO 499 ARE AVAILABLE FOR THESE

                DATA
                LOCATE      300
COUNT                                   COUNT OF INPUT CHARACTERS

                NOTE        DATA PART 3: PUBLIC VALUE CONTAINING BIT 21
                NOTE        ... OR ITS EQUIVALENT IN THE DES-2 KOS ..
                NOTE        ... ENVIRONMENT

                DATA
                LOCATE      4
£UBIT21

                NOTE        CONSTANT AREA
                NOTE        THERE IS ONLY ONE COPY OF THIS
                NOTE        CONSTANTS HAVE ABSOLUTE ADDRESSES AND MUST BE
                NOTE        ADDRESSED USING £UBIT21

                CONST
```

```
UPARROW                 124         CODE FOR UPARROW

                NOTE        MESSAGES

INITMS                      C:COUN
                            C:TING
                            C: PRO
                            C:GRAM
                            C: (VE
                            C:RSIO
                            C:N KN
                            C:LEAD
                            C: STA
                            C:RTS

SFATMS                      C: CHA
                            C:RS S
                            C:O FA
                            C:R

EXITMS                      C:END
                            C:OF C
                            C:OUNT
                            C:ING

                NOTE        NOW COMES THE CODE

                CODE

                NOTE        CODE PART 1: INITIALISATION
                NOTE        INITIALISATION FOLLOWS A SIMILAR PATTERN FOR
                NOTE        NEARLY ALL SUB-SYSTEMS
                NOTE        THE FIRST THREE INSTRUCTIONS ARE ALWAYS
                NOTE        ENTRY POINTS FROM KOS

                J#          4           ENTRY 1: MAIN ENTRY
                J#          BREAK       ENTRY 2: BREAK ENTRY
```

```
                        JF          GETOFF      ENTRY 3: GET-OFF ENTRY

                        JIL         £UALBRK     ALLOW BREAKS
                        LD:L        2           SET UP IO DEVICES (PARAMETER ...
                        JIL         £USETDEV    ... OF 2 MEANS BOTH DATA AND ...
                        NOTE                    ... RESULTS DEVICES NEEDED)
                        JF          GETOFF      INVALID EXIT
                        LDR:L       INITMS      OUTPUT INTRODUCTORY MESSAGE
                        JIL         £UMSTR      (UMSTR AUTOMATICALLY ADDS BIT 21
                        NOTE                    ... TO ITS PARAMETER)


                        NOTE        CODE PART 2: MAIN SECTION OF CODE

                        CLS         COUNT
                LOOP    JIL         £UDCHAR     GET AN INPUT CHARACTER
                        JF          GETOFF      INVALID EXIT, INPUT EXHAUSTED
                        INCS        COUNT
                        COMP:L      2           TEST IF NEWLINE
                        JZ          NEWLINE
                        LIR         £UBIT21     TEST IF UPARROW
                        COMP:M      UPARROW     (THIS SHOWS USE OF BIT 21 TO ...
                        JNZ         NOT         ... REFERENCE THE CONST AREA)
                        LD:M        V:-1        (SIMILARLY USE BIT 21 FOR ...
5/4     ****** WARNING  1/5
                        ST          COUNT       ... V-LITERALS)
                NOT     JR          LOOP


                        NOTE        ACTION AT END OF LINE
                        NOTE        OUTPUT A LINE OF FORM: ... LINES SO FAR

                NEWLINE LD          COUNT

                        JIL         £URVAL      OUTPUT COUNT
                        LDR:L       SFARMS
                        JIL         £URSTR      OUTPUT REST OF MESSAGE
                        JR          LOOP
```

KCS/AAA

KCL5

```
                        NOTE        CODE PART 3: FINALISATION

                GETOFF


                BREAK       LDR:L       EXITMS      CLOSING MESSAGE

                            JIL         £UMSTP
                            JI          £UEXIT      EXIT BACK TO KOS


                            END
RELATIVE ADDRESSES OF STACKS IN MAIN CHAPTER
  I         0
  C        11
  D        34
PROGRAM LENGTH
  C        74
  D        60
     NO ERRORS
      1 WARNING


 &KOSEX;PROD:
KOSEX
COMMON MODE NOT USED
COMMAN
UTILIT
MCP
"KOSEX KNL5E    1   JOB STREAMS SET UP
"/// MCP KNL5M   IN CONTROL
```

```
*** KOS READY - VERSION KNL5 E

***&NP TEST RUN IN THE BATCH

***&TRY TEST

***COUNTING PROGRAM (VERSION KNL5A) STARTS
&LIST,L
1.    12345
5 CHARS SO FAR
2.    6789
11 CHARS SO FAR
3.    †
0 CHARS SO FAR
4.    KKK
4 CHARS SO FAR
5.    JJJ+123
3 CHARS SO FAR
6.    4
5 CHARS SO FAR

***END OF COUNTING

***&KILL;

0001    KOS JOB STEPS EXECUTED"
ELETE
SER NOT DELETED


&END;
IME =  0000  12.29?
  G
```

Usage at a console

```
%NR: THIS IS A RUN OF 'TEST' ON A CONSOLE
%NR: ... AFTER IT HAS BEEN UPDATED INTO THE SYSTEM

%ENTER TEST

***COUNTING PROGRAM (VERSION KNL5A) STARTS
:12345678
9 CHARS SO FAR
:
10 CHARS SO FAR
:ABCDEFGHIJKLMN
25 CHARS SO FAR
:!
0 CHARS SO FAR
:KKKKK
6 CHARS SO FAR
:BBBBB!123
3 CHARS SO FAR
:!
5 CHARS SO FAR
:.

***END OF COUNTING

%
```

Appendix F Sub-system writer's glossary

The glossary given below, which is supplementary to that in the KOS User's Manual, has, like the latter, a dual purpose.

Firstly it can be used as a dictionary. The reference against each term indicates the Section in which it is first defined or used.

Secondly it should be used by persons giving lectures, writing manuals or even talking about KOS. Here it should be used, together with the User's Manual glossary, to define a standard terminology which must be adhered to whenever the context is appropriate. Synonyms should not be used.

Terminology	Definition in Section
MCP (master control program)	1.1
Slave, slave mode	1.1
Common program	1.1
Base, range	1.1
Slave fixed locations	1.2
Sub-system fixed locations	1.2
Relative pointer, relative address	1.2
Absolute pointer, absolute address	1.2
Logical error	1.5
COMMAN	2.1
UTILIT, UTILIT routines	2.1
Specialized utility programs	2.2
Public values	2.3
Parameters) re UTILIT routines	2.4
Results) and EXENs	
Exits)	
Return)	
EXEN	2.5
Initial entry	2.6
Break entry	2.6
Get-off entry	2.6
Extras	3.1
Borrowing and returning extras	3.1
KOS 7-bit code	4.2
Newline (character)	4.2
Character	4.2
Packed 6-bit code	4.3.1
End marker	4.3.1
String	4.3.1
Clearing a buffer	4.3.1
KOS device number	4.5
Decodee	5
Decode table	5
Return offset	5
Match (of decodee with decode table entry)	5
Allow breaks	6.1
Inhibit breaks	6.1